Bölüm 13: Öncelikli Kuyruk ve Heap

C ile Veri Yapıları ve Algoritmalar

Öncelikli kuyruk · Maksimum / minimum heap · Dizide tam ikili ağaç · Ekleme: yukarı kaydırma · Silme: aşağı kaydırma · Karşılaştırma · Heapify · d-ary

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Tanımlamak

Öncelikli kuyruğu ve maksimum/minimum heap'i tanımlamak; heap'i ikili arama ağacından ayırmak.

Gösterimi kurmak

Tam ikili ağacı 1 tabanlı dizide saklamak: ebeveyn `i/2`, çocuklar `2i`, `2i+1`.

Ekleme ve silme

`maksimum_yigin_ekle` ve `maksimum_yigin_al` yordamlarını adım adım izlemek.

Analiz etmek

Ekleme ve silmenin neden O(lg n) olduğunu, en iyi durumun Ω(1) olduğunu göstermek.

Karşılaştırmak

Sırasız/sıralı dizi ve liste ile heap'i maliyet tablosu üzerinden kıyaslamak.

Genişletmek

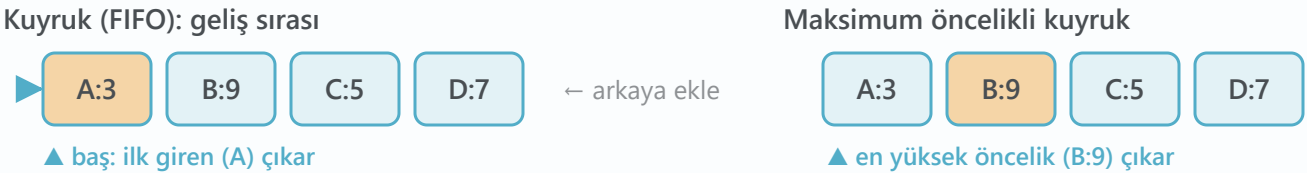
Aşağıdan yukarı heap kurmanın (heapify) O(n) olduğunu açıklamak; d-ary heap'i tanımak.

ÖNCELİKLİ KUYRUK

Öncelikli kuyruk: ilk çıkan en öncelikli öğe

Kuyruktan ilk alınacak (silinecek) öğe, geliş sırasına göre değil **en yüksek** ya da **en düşük** değere göre belirlenir.

- **Maksimum öncelikli kuyruk** (*max priority queue*): önce en yüksek değer ele alınır.
- **Minimum öncelikli kuyruk** (*min priority queue*): önce en düşük değer ele alınır.



Hocadan *"Priority queue diyoruz, bakalım kuyruğu ne kadar biliyoruz? Bazen bir şeyleri bilmek de sakıncalı olabilir. Bildiğimiz kuyrukta sondan eklenir, baştan alınır. Bu ise ondan farklı..."*

ÖNCELİKLİ KUYRUK

Öncelikli kuyruğa neden gereksinim duyarız?

1 · Fotokopi makinesi

Her kullanıcı sabit ücret ödüyor (3 sayfa da 30 sayfa da aynı). Kazancı en büyük yapmak için kullanım süresi **en kısa** olana öncelik verilir → **minimum** öncelikli kuyruk.

2 · Havaalanı

İniş isteyen uçaklardan **yakıtı en az** olana öncelik verilmelidir → **minimum** öncelikli kuyruk.

Gerçek sistemlerde de böyledir: örneğin işletim sistemi işlemleri (process) öncelikli kuyruğa koyar.

Hocadan *"Kazancımı maksimum yapmak istiyoruz. Kullanma süresi minimum olana öncelik vererek kazanç maksimize edilir."*

Hocadan *"'Kuyruğun arkasındasın. Yakıtın az, ama kuyruk böyle. Ne yapalım?' diyemeyiz."*

Soyut veri tipi: öncelikli kuyruk

```
/* maksimum öncelikli kuyruk arayüzü */
bool bos_mu(void);
bool dolu_mu(void);
void ekle(eleman item);      /* O(lg n) */
bool maks_al(eleman *cikan); /* sil + döndür */
bool maks_bak(eleman *ust);  /* silmeden bak */
```

Ekle

Öğe, önceliğiyle birlikte yapıya konur.

Al / Sil

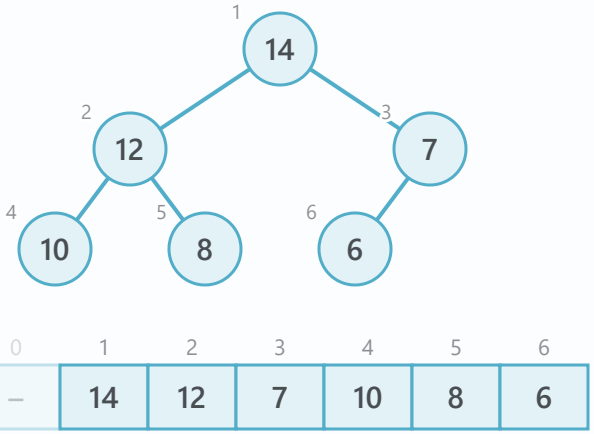
En öncelikli öğe döndürülür ve yapıdan çıkarılır.

Öncelikli kuyruk bir **arayüzdür**; farklı veri yapılarıyla gerçeklenebilir. **Heap** yalnızca biridir, ama en dengelisidir: ekleme ve silme O(lg n).

Hocadan *"Heap, nerede ve nasıl kullanılacağı kavranırsa çok çok önemli bir veri yapısıdır."*

HEAP YAPISI

Heap bir dizide tam ikili ağaç olarak durur



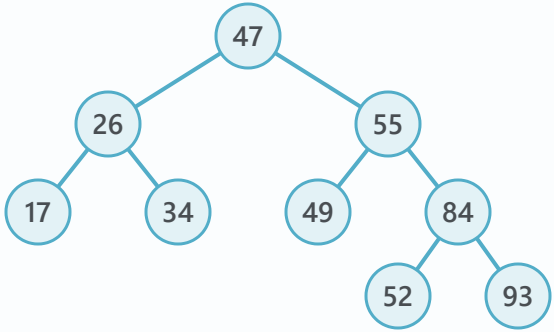
Dizinin 0. indisi kullanılmaz; kök `yigin[1]`.

Düğüm	İndis
kök	`1`
i'nin ebeveyni	`i / 2`
sol çocuk	`2i`
sağ çocuk	`2i + 1`
son öğe	`n`

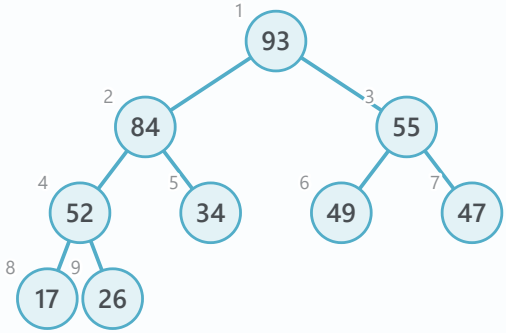
0 tabanlı dizide: ebeveyn `(i-1)/2`, çocuklar `2i+1`, `2i+2`.

Dizi **sıralı değildir**; yalnızca 1. indisteki öğenin en öncelikli (ilk alınacak) öğe olduğu garanti edilir. Tam ikili ağaçta boşluk olmadığından işaretçiye gerek kalmaz.

Heap denince akla tam ikili ağaç gelmeli, arama ağacı değil



İkili arama ağacı (binary search tree)



Aynı değerlerle bir maksimum heap

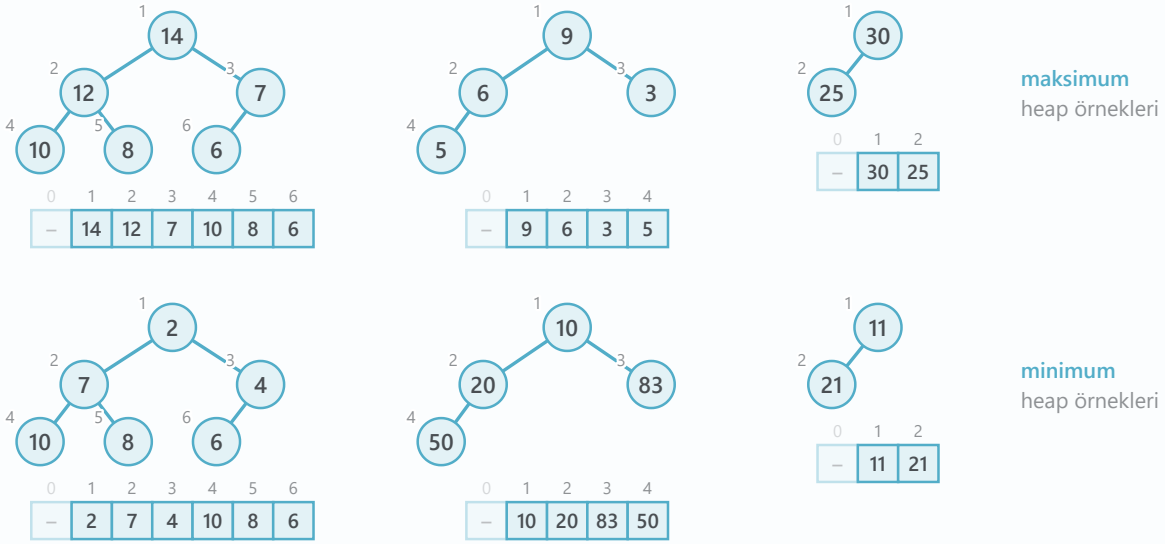
- **Tam ikili ağaç:** tüm düzeyler dolu, son düzey soldan sağa doğru dolu.
- Arama ağacında *sol* < *kök* < *sağ*; heap'te yalnızca *ebeveyn* ≥ *çocuk* (sol-sağ arasında sıra yok).

Hocadan *"'Hocam bir ikili arama ağacı çizer misiniz?' Genel istek üzerine çizelim, bununla karıştırmıyoruz!"*

Tanım: maksimum heap ve minimum heap

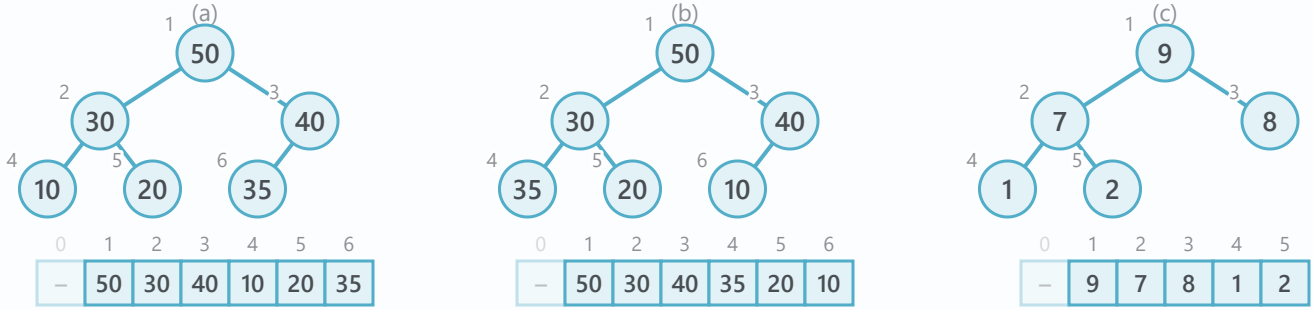
Maksimum heap (*max heap*): tam ikili ağaçta her düğümün değeri çocuklarından **küçük değilse**.

Minimum heap (*min heap*): tam ikili ağaçta her düğümün değeri çocuklarından **büyük değilse**.



Bu tanımlar maksimum/minimum öncelikli kuyruk kullanabilmek için verilmiştir.

Hızlı kontrol: hangisi maksimum heap?



Her `k ≥ 2` için `yigin[k/2] ≥ yigin[k]` mi? Yalnızca ebeveyn–çocuk çiftlerine bakılır; kardeşler ve kuzenler arasında sıra aranmaz.

▶ **Cevap**

C ile tanımlar: dizi, sayaç ve makrolar

```c
#define ELEMAN_SAYISI 200   /* maksimum öğe + 1 */
#define YIGIN_DOLUMU(n) ((n) == ELEMAN_SAYISI - 1)
#define YIGIN_BOSMU(n)  (!(n))

typedef struct {
  int x;          /* öncelik (anahtar) */
  /* diğer alanlar */
} eleman;

eleman yigin[ELEMAN_SAYISI];  /* [0] kullanılmaz */
int n = 0;                    /* öğe sayısı */
```

Kapasite

1..199 indisleri kullanılır; `n == 199` ise heap doludur.

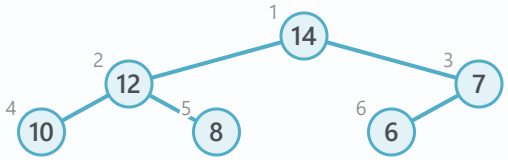
Makrolarda parantez

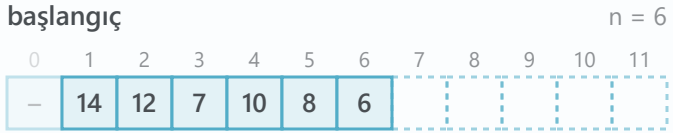
`(n)`, `YIGIN_BOSMU(a+b)` gibi çağrılarda öncelik hatasını önler.

Sık yapılan hata: diziyi `eleman yigin[ELEMAN_SAYISI - 1]` (199 öğe, indis 0..198) diye tanımlamak. Dolu sayılmadan önce `n` 199'a çıkabilir ve `yigin[199]` dizinin dışına yazılır; boyut `ELEMAN_SAYISI` olmalıdır.

EKLEME

Maksimum heap'e öğe ekleme: önce yere konur, sonra kaydırılır





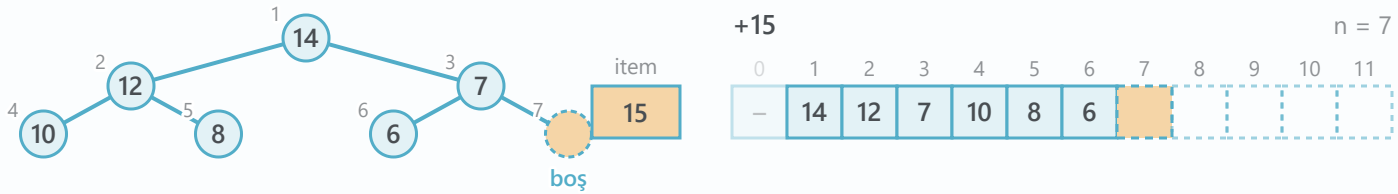
Başlangıç: `14, 12, 7, 10, 8, 6`. Sırasıyla 15, 109, 107, 3, 15 eklenecek.

Adım 1 / 5

Hocadan *"Eklenecek öğenin yeri kesinlikle hazırdır. Bundan kuşku yok. Önce o yere eklenir. Sonra eyvah! Çocuğundan büyük olmak zorundaydı. Öyleyse uygun konuma kaydırırız."*

EKLEME

Maksimum heap'e öğe ekleme: önce yere konur, sonra kaydırılır



Önce yeni yer açılır: tam ağacın ilk boş konumu (**7**).

Adım 2 / 5

Hocadan *"Eklenecek öğenin yeri kesinlikle hazırdır. Bundan kuşku yok. Önce o yere eklenir. Sonra eyvah! Çocuğundan büyük olmak zorundaydı. Öyleyse uygun konuma kaydırırız."*

EKLEME

Maksimum heap'e öğe ekleme: önce yere konur, sonra kaydırılır



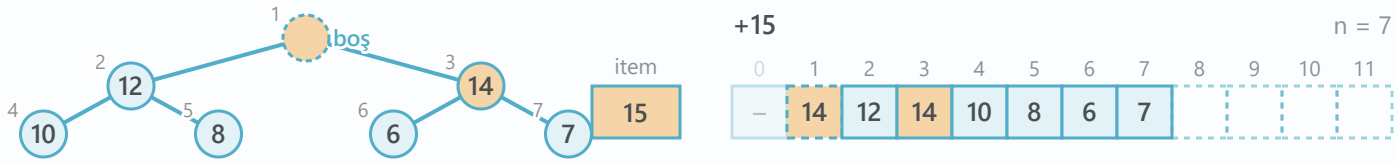
15 > 7: ebeveyn **7** aşağı kaydırılır, boşluk 3. konuma çıkar.

Adım 3 / 5

Hocadan *"Eklenecek öğenin yeri kesinlikle hazırdır. Bundan kuşku yok. Önce o yere eklenir. Sonra eyvah! Çocuğundan büyük olmak zorundaydı. Öyleyse uygun konuma kaydırırız."*

EKLEME

Maksimum heap'e öğe ekleme: önce yere konur, sonra kaydırılır



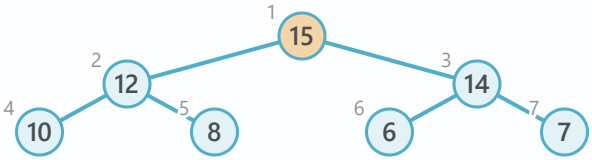
15 > 14: ebeveyn **14** aşağı kaydırılır, boşluk 1. konuma çıkar.

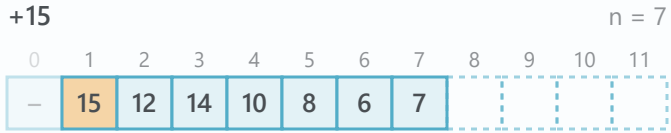
Adım 4 / 5

Hocadan *"Eklenecek öğenin yeri kesinlikle hazırdır. Bundan kuşku yok. Önce o yere eklenir. Sonra eyvah! Çocuğundan büyük olmak zorundaydı. Öyleyse uygun konuma kaydırırız."*

EKLEME

Maksimum heap'e öğe ekleme: önce yere konur, sonra kaydırılır





15, 1. konuma yazılır. Heap özelliği yeniden sağlandı.

Adım 5 / 5

Hocadan *"Eklenecek öğenin yeri kesinlikle hazırdır. Bundan kuşku yok. Önce o yere eklenir. Sonra eyvah! Çocuğundan büyük olmak zorundaydı. Öyleyse uygun konuma kaydırırız."*

EKLEME

`maksimum_yigin_ekle`: takas değil, kaydırma

```
void maksimum_yigin_ekle(eleman item) {
  if (YIGIN_DOLUMU(n)) {
    yigin_dolu();              /* hata iletisi */
    return;
  }
  int i = ++n;
  while (i != 1) {             /* i == 1: kökteyiz */
    if (item.x <= yigin[i / 2].x) break;
    yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
    i /= 2;                    /* bir üst ebeveyne bak */
  }
  yigin[i] = item;
}
```

Boşluk (delik) yöntemi

Takas yok: büyük atalar birer düzey **aşağı kaydırılır**, öğe en sonda bir kez yazılır (takas 3, kaydırma 1 atama).

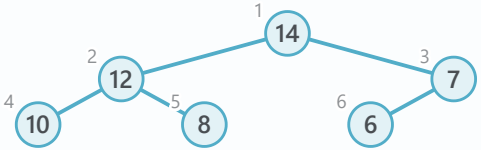
Döngü koşulu

`i ≠ 1`: kökte ebeveyn yoktur. `item.x ≤ ebeveyn` olduğunda yerleşim yeri bulunmuştur.

Dolu heap

`yigin_dolu()` bir hata iletisi yordamıdır; ekleme yapılmadan dönülür.

Kod izleme: `maksimum_yigin_ekle(15)`



ekle(15) n = 6

0	1	2	3	4	5	6	7	8	9	10	11
–	14	12	7	10	8	6					

ekle(15) çağrılır; heap'te 6 öğe var: `14, 12, 7, 10, 8, 6` .

Adım 1 / 13

```
▶ void maksimum_yigin_ekle(eleman item) {
2   if (YIGIN_DOLUMU(n)) {
3     yigin_dolu();              /* hata iletisi */
4     return;
5   }
6   int i = ++n;
7   while (i != 1) {             /* i == 1: kökteyiz */
8     if (item.x <= yigin[i / 2].x) break;
9     yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10    i /= 2;                    /* bir üst ebeveyne bak */
11  }
12  yigin[i] = item;
13 }
```

DEĞİŞKENLER

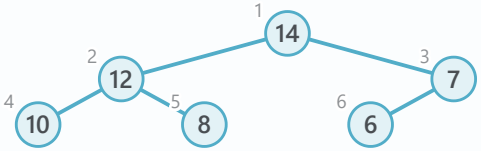
`item.x = 15` `n = 6`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(15)`



ekle(15) n = 6

0	1	2	3	4	5	6	7	8	9	10	11
–	14	12	7	10	8	6					

`YIGIN_DOLUMU(6)` yanlış: 6 ≠ 199, yer var.

Adım 2 / 13

```
 1 void maksimum_yigin_ekle(eleman item) {
 2   if (YIGIN_DOLUMU(n)) {
 3     yigin_dolu();              /* hata iletisi */
 4     return;
 5   }
 6   int i = ++n;
 7   while (i != 1) {             /* i == 1: kökteyiz */
 8     if (item.x <= yigin[i / 2].x) break;
 9     yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

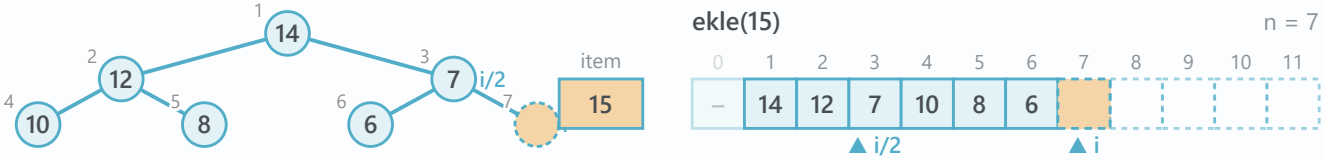
`item.x = 15` `n = 6`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(15)`



i = ++n = 7: tam ağacın ilk boş konumu. Yeni öğenin yeri kesinlikle hazır; ama önce ebeveyniyle kıyaslanacak, `item` henüz yazılmaz.

Adım 3 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
▶    int i = ++n;
7    while (i != 1) {            /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 7`

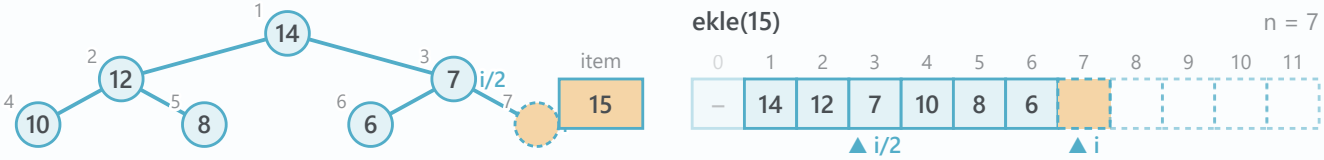
`i/2 = 3` `yigin[i/2].x = 7`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(15)`



i = 7 ≠ 1: henüz kökte değiliz, döngüye girilir.

Adım 4 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {           /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 7` `i/2 = 3` `yigin[i/2].x = 7`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`



15 ≤ 7? Hayır. Yeni öğe ebeveyninden büyük: ebeveyn aşağı inmeli.

Adım 5 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
▶      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 7`
`i/2 = 3` `yigin[i/2].x = 7`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`



yigin[7] = yigin[3]: ebeveyn (7) bir düzey aşağı kaydı. Takas yok; boşluk (turuncu) yukarı çıktı.

Adım 6 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
▶      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 7`

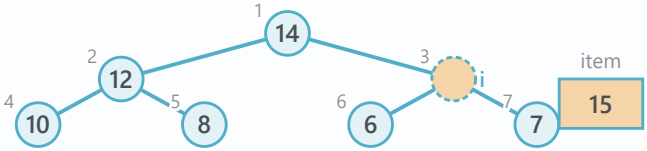
`i/2 = 3` `yigin[i/2].x = 7`

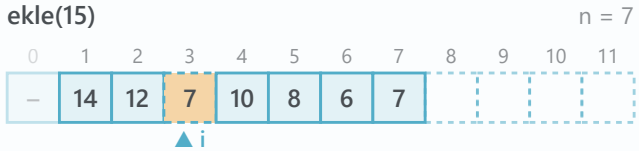
ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(15)`





i /= 2 → i = 3: bir üstteki ebeveyne geçilir.

Adım 7 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
▶      i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

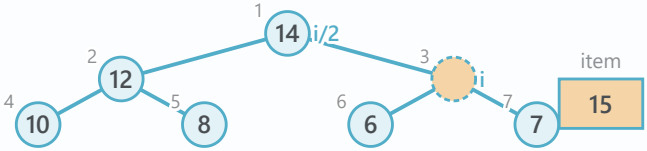
DEĞİŞKENLER

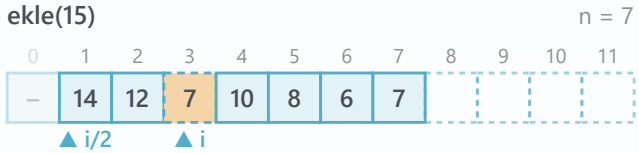
`item.x = 15` `n = 7` `i = 3` `i/2 = 1` `yigin[i/2].x = 14`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`





i = 3 ≠ 1: henüz kökte değiliz, döngüye girilir.

Adım 8 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {           /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 3`

`i/2 = 1` `yigin[i/2].x = 14`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`



15 ≤ 14? Hayır. Yeni öğe ebeveyninden büyük: ebeveyn aşağı inmeli.

Adım 9 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
▶      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

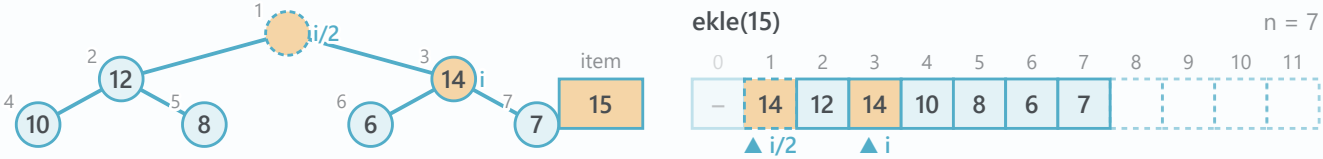
`item.x = 15` `n = 7` `i = 3`

`i/2 = 1` `yigin[i/2].x = 14`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`



yigin[3] = yigin[1]: ebeveyn (14) bir düzey aşağı kaydı. Takas yok; boşluk (turuncu) yukarı çıktı.

Adım 10 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
▶      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

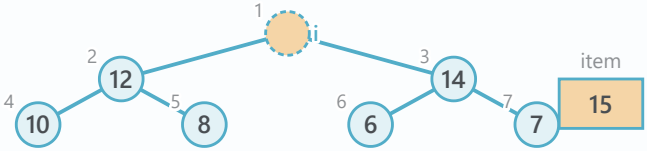
`item.x = 15` `n = 7` `i = 3`

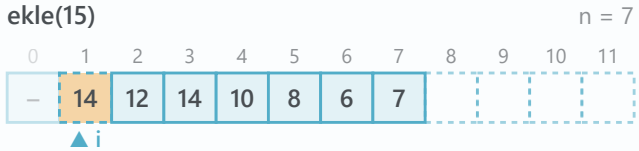
`i/2 = 1` `yigin[i/2].x = 14`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`





i /= 2 → i = 1: bir üstteki ebeveyne geçilir.

Adım 11 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
▶      i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

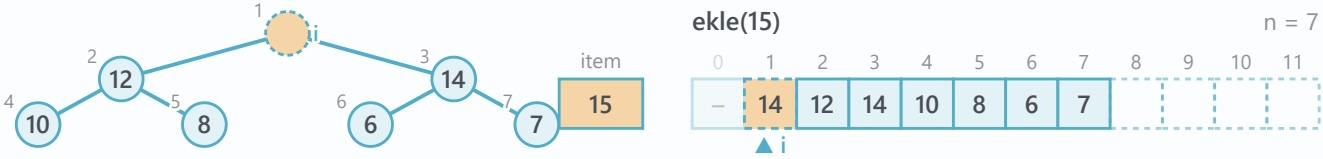
`item.x = 15` `n = 7` `i = 1`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(15)`



i == 1: köke ulaşıldı, döngü biter.

Adım 12 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {           /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

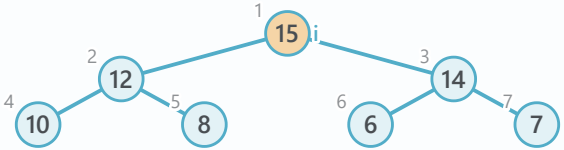
DEĞİŞKENLER

`item.x = 15` `n = 7` `i = 1`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(15)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(15)`



ekle(15) n = 7

0	1	2	3	4	5	6	7	8	9	10	11
–	15	12	14	10	8	6	7				

▲ i

yigin[1] = item: 15 yerine oturdu. Heap: `15, 12, 14, 10, 8, 6, 7` .

Adım 13 / 13

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
▶    yigin[i] = item;
13 }
```

DEĞİŞKENLER

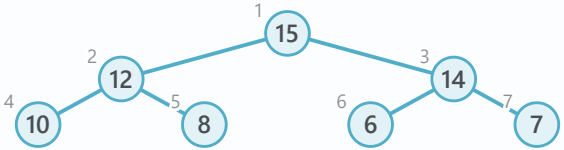
`item.x = 15` `n = 7` `i = 1`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(15)`

`main()`

Kod izleme: `maksimum_yigin_ekle(109)`



ekle(109) n = 7

0	1	2	3	4	5	6	7	8	9	10	11
–	15	12	14	10	8	6	7				

ekle(109): tüm öğelerden büyük bir değer. En kötü durum: köke kadar çıkacak.

Adım 1 / 17

```
▶ void maksimum_yigin_ekle(eleman item) {
2   if (YIGIN_DOLUMU(n)) {
3     yigin_dolu();              /* hata iletisi */
4     return;
5   }
6   int i = ++n;
7   while (i != 1) {             /* i == 1: kökteyiz */
8     if (item.x <= yigin[i / 2].x) break;
9     yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10    i /= 2;                    /* bir üst ebeveyne bak */
11  }
12  yigin[i] = item;
13 }
```

DEĞİŞKENLER

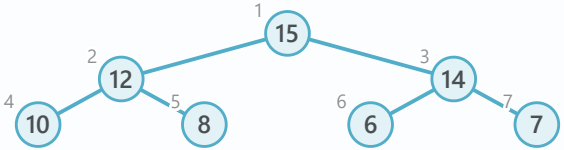
`item.x = 109` `n = 7`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`

`main()`

Kod izleme: `maksimum_yigin_ekle(109)`



ekle(109) n = 7

0	1	2	3	4	5	6	7	8	9	10	11
–	15	12	14	10	8	6	7				

`YIGIN_DOLUMU(7)` yanlış: 7 ≠ 199, yer var.

Adım 2 / 17

```
 1 void maksimum_yigin_ekle(eleman item) {
 2   if (YIGIN_DOLUMU(n)) {
 3     yigin_dolu();              /* hata iletisi */
 4     return;
 5   }
 6   int i = ++n;
 7   while (i != 1) {             /* i == 1: kökteyiz */
 8     if (item.x <= yigin[i / 2].x) break;
 9     yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 7`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`

`main()`

Kod izleme: `maksimum_yigin_ekle(109)`



i = ++n = 8: tam ağacın ilk boş konumu. Yeni öğenin yeri kesinlikle hazır; ama önce ebeveyniyle kıyaslanacak, `item` henüz yazılmaz.

Adım 3 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
▶    int i = ++n;
7    while (i != 1) {            /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

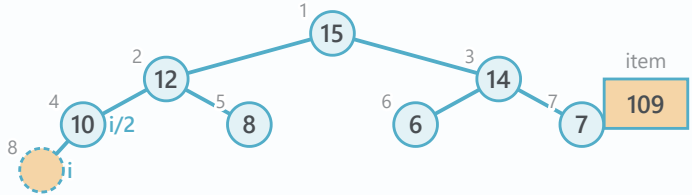
`item.x = 109` `n = 8` `i = 8` `i/2 = 4` `yigin[i/2].x = 10`

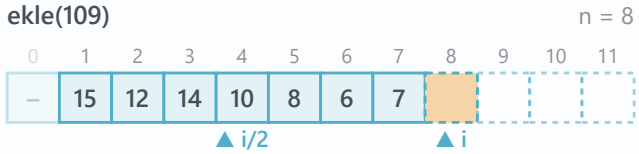
ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`

`main()`

Kod izleme: `maksimum_yigin_ekle(109)`





i = 8 ≠ 1: henüz kökte değiliz, döngüye girilir.

Adım 4 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {            /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 8`
`i/2 = 4` `yigin[i/2].x = 10`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`
`main()`

Kod izleme: `maksimum_yigin_ekle(109)`



109 ≤ 10? Hayır. Yeni öğe ebeveyninden büyük: ebeveyn aşağı inmeli.

Adım 5 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
▶      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 8` `i/2 = 4` `yigin[i/2].x = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



yigin[8] = yigin[4]: ebeveyn (10) bir düzey aşağı kaydı. Takas yok; boşluk (turuncu) yukarı çıktı.

Adım 6 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
▶      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 8`
`i/2 = 4` `yigin[i/2].x = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



i /= 2 → i = 4: bir üstteki ebeveyne geçilir.

Adım 7 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
▶      i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 4` `i/2 = 2` `yigin[i/2].x = 12`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



i = 4 ≠ 1: henüz kökte değiliz, döngüye girilir.

Adım 8 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {            /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 4` `i/2 = 2` `yigin[i/2].x = 12`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



109 ≤ 12? Hayır. Yeni öğe ebeveyninden büyük: ebeveyn aşağı inmeli.

Adım 9 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
▶      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 4`
`i/2 = 2` `yigin[i/2].x = 12`

ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`
`main()`

Kod izleme: `maksimum_yigin_ekle(109)`



yigin[4] = yigin[2]: ebeveyn (12) bir düzey aşağı kaydı. Takas yok; boşluk (turuncu) yukarı çıktı.

Adım 10 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
▶      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 4`
`i/2 = 2` `yigin[i/2].x = 12`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



i /= 2 → i = 2: bir üstteki ebeveyne geçilir.

Adım 11 / 17

```
 1 void maksimum_yigin_ekle(eleman item) {
 2   if (YIGIN_DOLUMU(n)) {
 3     yigin_dolu();              /* hata iletisi */
 4     return;
 5   }
 6   int i = ++n;
 7   while (i != 1) {             /* i == 1: kökteyiz */
 8     if (item.x <= yigin[i / 2].x) break;
 9     yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
▶      i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

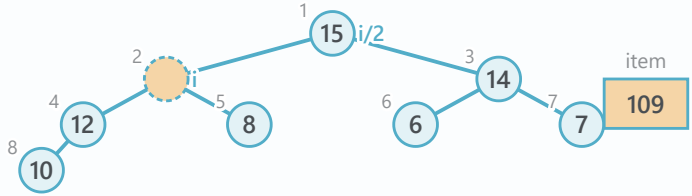
DEĞİŞKENLER

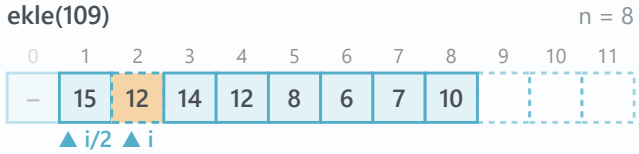
`item.x = 109` `n = 8` `i = 2` `i/2 = 1` `yigin[i/2].x = 15`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`





i = 2 ≠ 1: henüz kökte değiliz, döngüye girilir.

Adım 12 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {           /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 2` `i/2 = 1` `yigin[i/2].x = 15`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



109 ≤ 15? Hayır. Yeni öğe ebeveyninden büyük: ebeveyn aşağı inmeli.

Adım 13 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
▶      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

`item.x = 109` `n = 8` `i = 2` `i/2 = 1` `yigin[i/2].x = 15`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`



yigin[2] = yigin[1]: ebeveyn (15) bir düzey aşağı kaydı. Takas yok; boşluk (turuncu) yukarı çıktı.

Adım 14 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
▶      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

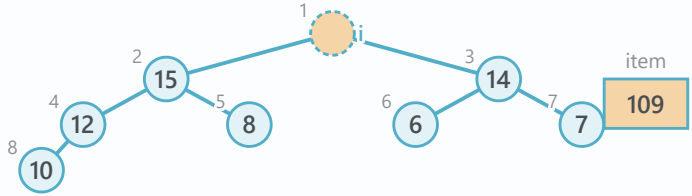
DEĞİŞKENLER

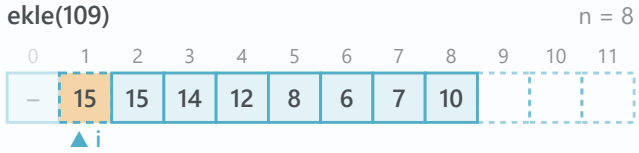
`item.x = 109` `n = 8` `i = 2` `i/2 = 1` `yigin[i/2].x = 15`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`





i /= 2 → i = 1: bir üstteki ebeveyne geçilir.

Adım 15 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
▶      i /= 2;                    /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

DEĞİŞKENLER

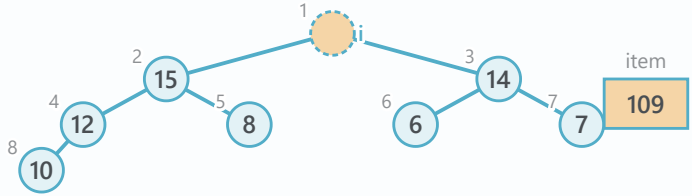
`item.x = 109` `n = 8` `i = 1`

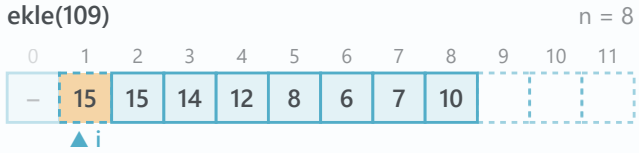
ÇAĞRI YIĞINI

`maksimum_yigin_ekle(109)`

`main()`

Kod izleme: `maksimum_yigin_ekle(109)`





i == 1: köke ulaşıldı, döngü biter.

Adım 16 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
▶    while (i != 1) {           /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];  /* ebeveyni aşağı kaydır */
10     i /= 2;                   /* bir üst ebeveyne bak */
11   }
12   yigin[i] = item;
13 }
```

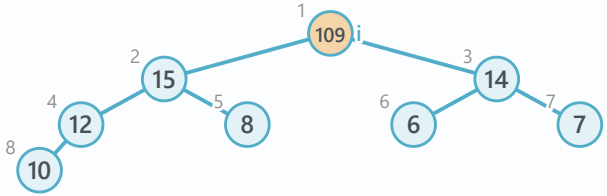
DEĞİŞKENLER

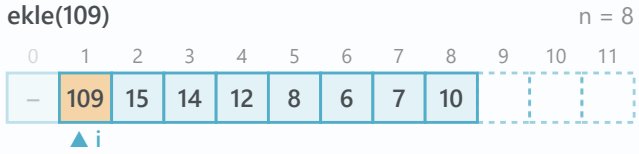
`item.x = 109` `n = 8` `i = 1`

ÇAĞRI YIĞINI

- `maksimum_yigin_ekle(109)`
- `main()`

Kod izleme: `maksimum_yigin_ekle(109)`





yigin[1] = item: 109 yerine oturdu. Heap: `109, 15, 14, 12, 8, 6, 7, 10` .

Adım 17 / 17

```
1  void maksimum_yigin_ekle(eleman item) {
2    if (YIGIN_DOLUMU(n)) {
3      yigin_dolu();              /* hata iletisi */
4      return;
5    }
6    int i = ++n;
7    while (i != 1) {             /* i == 1: kökteyiz */
8      if (item.x <= yigin[i / 2].x) break;
9      yigin[i] = yigin[i / 2];   /* ebeveyni aşağı kaydır */
10     i /= 2;                    /* bir üst ebeveyne bak */
11   }
▶    yigin[i] = item;
13 }
```

DEĞİŞKENLER

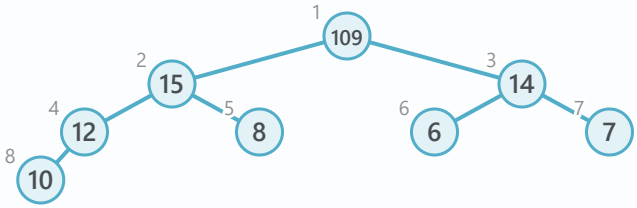
`item.x = 109` `n = 8` `i = 1`

ÇAĞRI YIĞINI

- **`maksimum_yigin_ekle(109)`**
- `main()`

EKLEME

Art arda ekleme: +107, +3, +15



başlangıç n = 8

0	1	2	3	4	5	6	7	8	9	10	11
–	109	15	14	12	8	6	7	10			

Şu an: `109, 15, 14, 12, 8, 6, 7, 10`. Sırada 107, 3 ve 15 var.

Adım 1 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



Önce yeni yer açılır: tam ağacın ilk boş konumu (**9**).

Adım 2 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



107 > 12: ebeveyn **12** aşağı kaydırılır, boşluk 4. konuma çıkar.

Adım 3 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



107 > 15: ebeveyn **15** aşağı kaydırılır, boşluk 2. konuma çıkar.

Adım 4 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



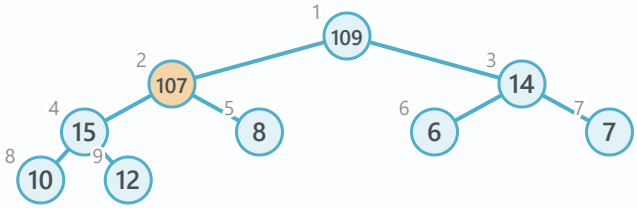
107 ≤ 109: ebeveyn yeterince büyük, durulur.

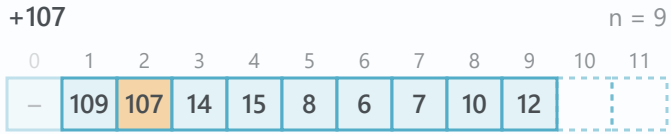
Adım 5 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15





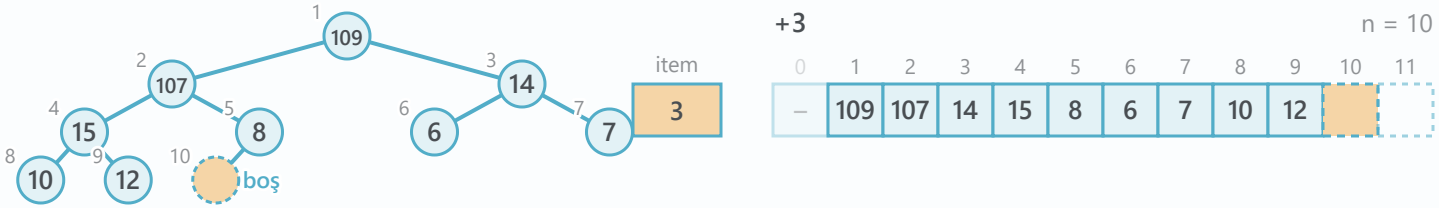
107, 2. konuma yazılır. Heap özelliği yeniden sağlandı.

Adım 6 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



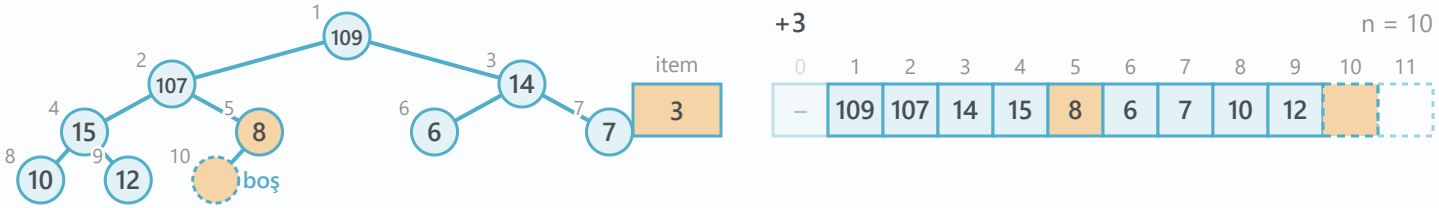
Önce yeni yer açılır: tam ağacın ilk boş konumu (**10**).

Adım 7 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



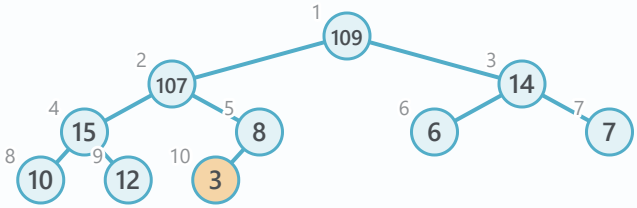
3 ≤ 8: ebeveyn yeterince büyük, durulur.

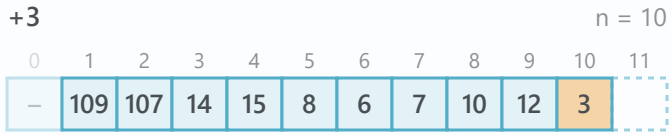
Adım 8 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15





3, 10. konuma yazılır. Heap özelliği yeniden sağlandı.

Adım 9 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



Önce yeni yer açılır: tam ağacın ilk boş konumu (**11**).

Adım 10 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



15 > 8: ebeveyn **8** aşağı kaydırılır, boşluk 5. konuma çıkar.

Adım 11 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



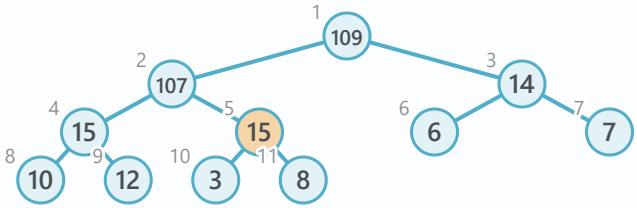
15 ≤ 107: ebeveyn yeterince büyük, durulur.

Adım 12 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Art arda ekleme: +107, +3, +15



+15 n = 11

0	1	2	3	4	5	6	7	8	9	10	11
–	109	107	14	15	15	6	7	10	12	3	8

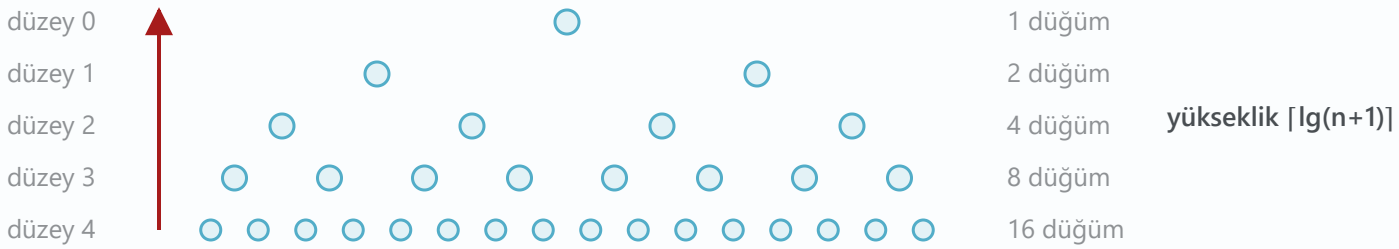
15, 5. konuma yazılır. Heap özelliği yeniden sağlandı.

Adım 13 / 13

3 eklenirken hiç kaydırma olmaz (en iyi durum, Ω(1)); 109 köke kadar çıkmıştı (en kötü durum, O(lg n)).

EKLEME

Ekleme maliyeti: en çok ağacın yüksekliği kadar

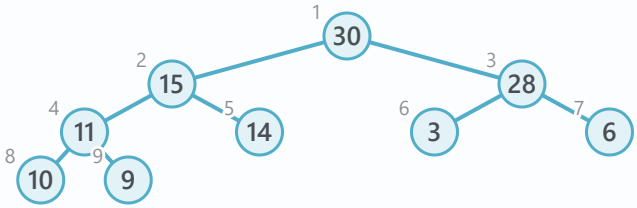


Durum	Örnek	Kaydırma	Maliyet
En iyi	3'ün eklenmesi	0	Ω(1)
En kötü	109'un eklenmesi	yükseklik kadar	O(lg n)
Genel	—	0 … ⌊lg n⌋	O(lg n)

Tam ikili ağaçta n öğe için yükseklik ⌈lg(n+1)⌉'dir. Döngü her turda bir düzey yukarı çıktığından karmaşıklığın üst sınırı lg n, alt sınırı 1'dir; zaman karmaşıklığı **O(lg n)**.

SİLME

Maksimum heap'ten öğe alma: kök alınır, sondaki yerleştirilir



başlangıç n = 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	9

Veri: `30, 15, 28, 11, 14, 3, 6, 10, 9`. İlk alınacak öğe kökteki 30.

Adım 1 / 4

Her adımda dikkat edilecek 3 değer var: iki çocuk ve sondaki öğe. En büyüğü i'nin gösterdiği yere taşınır; `i` yerleştirilecek konumu, `j` çocukları gösterir.

Hocadan *"Sadece bir öğeyi almak kolay! Ama iş burada bitmiyor. Öğeyi silmeli ve max öncelikli kuyruk yapısını da korumalıyız."*

SİLME

Maksimum heap'ten öğe alma: kök alınır, sondaki yerleştirilir



Kök (30) alındı. Sondaki öğe **9** çıkarıldı, kökteki boşluğa aday.

Adım 2 / 4

Her adımda dikkat edilecek 3 değer var: iki çocuk ve sondaki öğe. En büyüğü i'nin gösterdiği yere taşınır; `i` yerleştirilecek konumu, `j` çocukları gösterir.

Hocadan *"Sadece bir öğeyi almak kolay! Ama iş burada bitmiyor. Öğeyi silmeli ve max öncelikli kuyruk yapısını da korumalıyız."*

SİLME

Maksimum heap'ten öğe alma: kök alınır, sondaki yerleştirilir



İki çocuktan büyüğü (28) deger'den (9) büyük → yukarı taşındı.

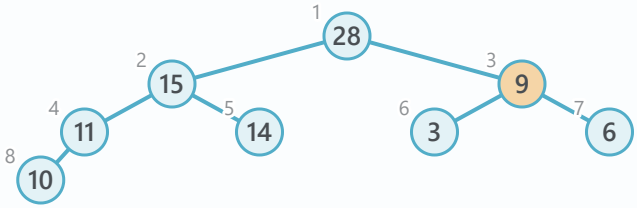
Adım 3 / 4

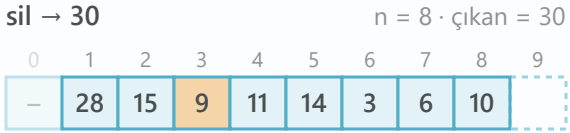
Her adımda dikkat edilecek 3 değer var: iki çocuk ve sondaki öğe. En büyüğü i'nin gösterdiği yere taşınır; `i` yerleştirilecek konumu, `j` çocukları gösterir.

Hocadan *"Sadece bir öğeyi almak kolay! Ama iş burada bitmiyor. Öğeyi silmeli ve max öncelikli kuyruk yapısını da korumalıyız."*

SİLME

Maksimum heap'ten öğe alma: kök alınır, sondaki yerleştirilir





9, 3. konuma yerleşti.

Adım 4 / 4

Her adımda dikkat edilecek 3 değer var: iki çocuk ve sondaki öğe. En büyüğü i'nin gösterdiği yere taşınır; `i` yerleştirilecek konumu, `j` çocukları gösterir.

Hocadan *"Sadece bir öğeyi almak kolay! Ama iş burada bitmiyor. Öğeyi silmeli ve max öncelikli kuyruk yapısını da korumalıyız."*

`maksimum_yigin_al`: düzeltilmiş sürüm

```
bool maksimum_yigin_al(eleman *cikan) {
  if (YIGIN_BOSMU(n)) return false;
  *cikan = yigin[1];          /* kök döndürülecek */
  eleman deger = yigin[n--];  /* yeri aranacak */
  int i = 1, j = 2;
  while (j <= n) {
    if (j < n && yigin[j].x < yigin[j + 1].x) j++;
    if (deger.x >= yigin[j].x) break;
    yigin[i] = yigin[j];      /* çocuğu yukarı al */
    i = j; j *= 2;
  }
  yigin[i] = deger;
  return true;
}
```

Sık yapılan hata

Boş heap'te `return 0;` yazmak **derleme hatası** verir (tamsayı bir struct'a dönüşmez).

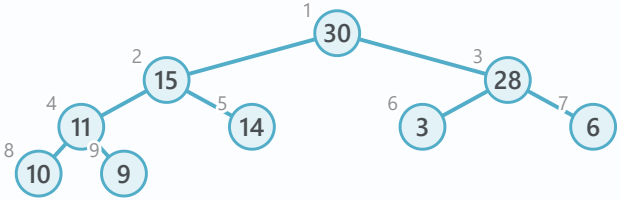
Düzeltme

Önerilen çözüm: `bool` dönüş (başarı/başarısızlık) ve alınan öğe **işaretçi parametre** ile döndürülür.

`j < n` neden?

`yigin[j+1]` okumadan önce sağ çocuğun var olduğunu garanti eder.

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	9

Çağrı: heap'te 9 öğe var; kök 30. Çıkan öğe `e` 'ye yazılacak.

Adım 1 / 15

```
▶ bool maksimum_yigin_al(eleman *cikan) {
  if (YIGIN_BOSMU(n)) return false;
  *cikan = yigin[1];          /* kök döndürülecek */
  eleman deger = yigin[n--];  /* yeri aranacak */
  int i = 1, j = 2;
  while (j <= n) {
    if (j < n && yigin[j].x < yigin[j + 1].x) j++;
    if (deger.x >= yigin[j].x) break;
    yigin[i] = yigin[j];      /* çocuğu yukarı al */
    i = j; j *= 2;
  }
  yigin[i] = deger;
  return true;
}
```

DEĞİŞKENLER

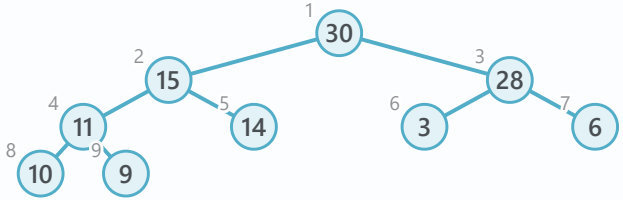
n = **9**

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	9

`YIGIN_BOSMU(9)` yanlış; alınacak öğe var.

Adım 2 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

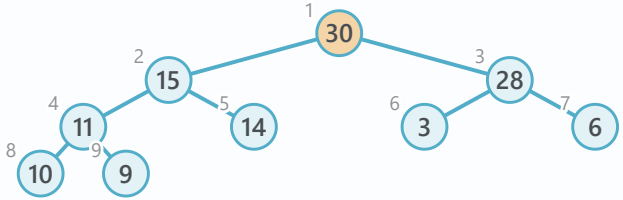
`n = 9`

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 9 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	9

***cikan = yigin[1]** = 30: tanım gereği ilk alınacak öğe dizinin ilk öğesidir.

Adım 3 / 15

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
▶    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

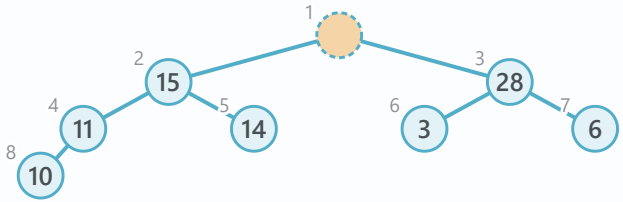
`n = 9` `cikan→x = 30`

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	

deger = yigin[n--] = 9, n = 8. Tam ağaç bozulmasın diye sondaki öğe çıkarıldı; ona kökten başlayarak yer aranacak.

Adım 4 / 15

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
▶    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

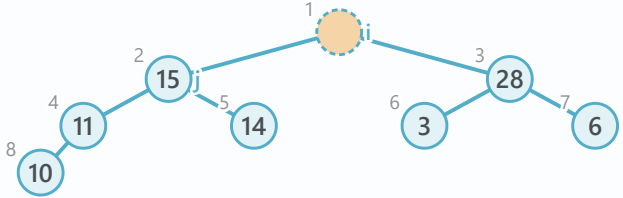
`n = 8` `cikan→x = 30` `deger.x = 9`

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	
	▲i	▲j							

i = 1 (deger'in aday konumu), **j = 2** (i'nin sol çocuğu).

Adım 5 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

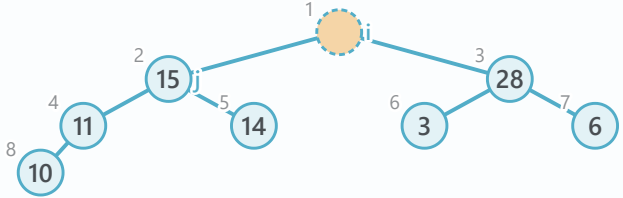
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	
	▲i	▲j							

j = 2 ≤ n = 8: i'nin çocuğu var, döngüye girilir.

Adım 6 / 15

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];           /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
▶    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

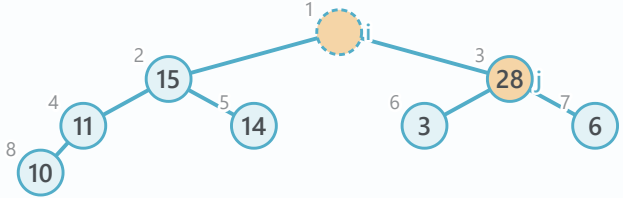
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	
	▲i		▲j						

Sağ çocuk daha büyük (15 < 28) → **j++**, j = 3.

Adım 7 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 ▶     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

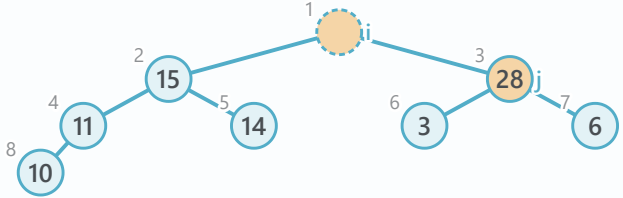
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 1` `j = 3`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	30	15	28	11	14	3	6	10	
	▲i		▲j						

9 ≥ 28? Hayır: büyük çocuk yukarı çıkmalı.

Adım 8 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 ▶     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

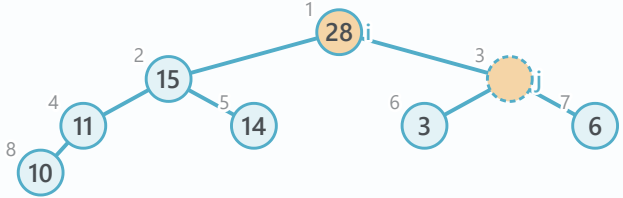
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 1` `j = 3`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

deger: 9

0	1	2	3	4	5	6	7	8	9
–	28	15	28	11	14	3	6	10	
	▲i		▲j						

yigin[1] = yigin[3]: 28 bir düzey yukarı taşındı; boşluk 3. konuma indi.

Adım 9 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 ▶     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

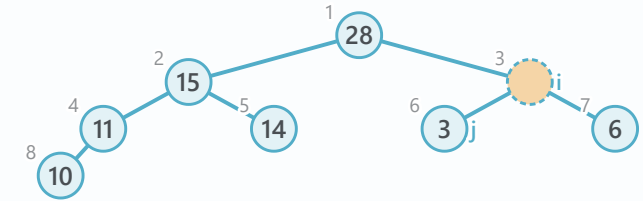
DEĞİŞKENLER

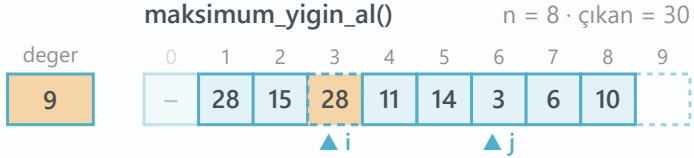
`n = 8` `cikan→x = 30` `deger.x = 9` `i = 1` `j = 3`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)





i = 3, **j = 6**: taşınan değerin eski konumuna ilerlenir.

Adım 10 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
▶      i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

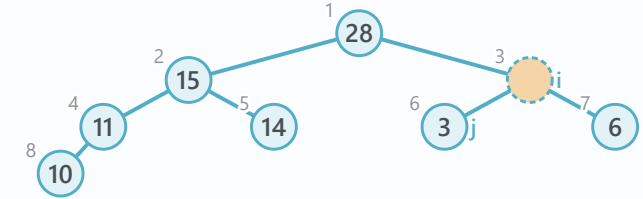
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 6`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



deger: 9

maksimum_yigin_al() — n = 8 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	28	15	28	11	14	3	6	10	
			▲i			▲j			

j = 6 ≤ n = 8: i'nin çocuğu var, döngüye girilir.

Adım 11 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];           /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 ▶   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

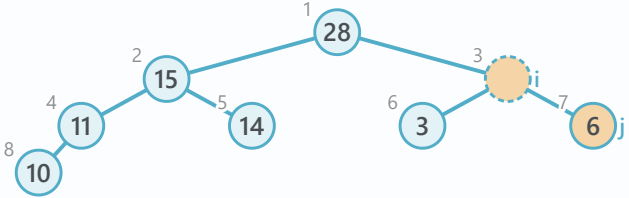
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 6`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



deger

9

maksimum_yigin_al() n = 8 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	28	15	28	11	14	3	6	10	
			▲i				▲j		

Sağ çocuk daha büyük (3 < 6) → **j++**, j = 7.

Adım 12 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];           /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 ▶     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

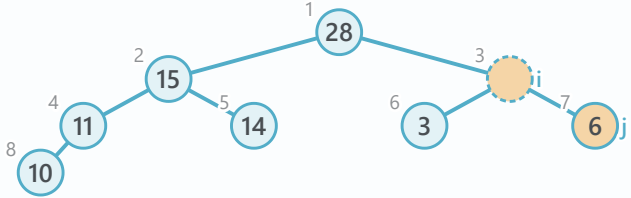
`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 7`

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: `maksimum_yigin_al` (30)



deger
9

maksimum_yigin_al() n = 8 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	28	15	28	11	14	3	6	10	
			▲i				▲j		

9 ≥ 6? Evet: deger iki çocuktan da büyük/eşit → `break`.

Adım 13 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 ▶     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

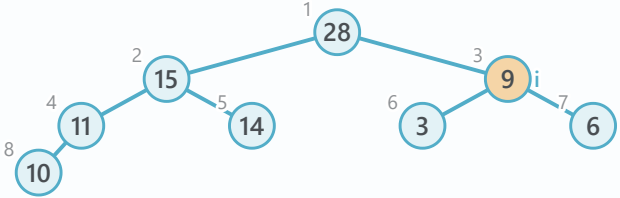
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 7`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	28	15	9	11	14	3	6	10	
			▲ i						

yigin[3] = deger (9). Heap: `28, 15, 9, 11, 14, 3, 6, 10` .

Adım 14 / 15

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
▶  yigin[i] = deger;
13   return true;
14 }
```

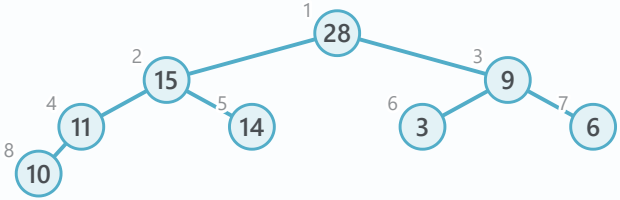
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 7`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: `maksimum_yigin_al` (30)



maksimum_yigin_al() n = 8 · çıkan = 30

0	1	2	3	4	5	6	7	8	9
–	28	15	9	11	14	3	6	10	

return true; çağıran `cikan` içinde **30** değerini alır. Yol boyu ≤ ⌊lg n⌋ düzey → O(lg n).

Adım 15 / 15

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
▶    return true;
14 }
```

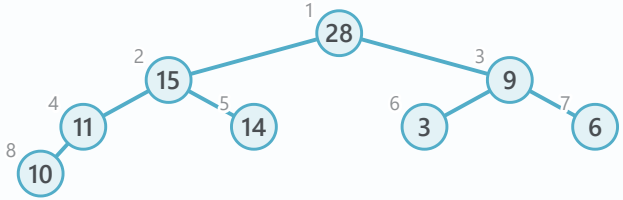
DEĞİŞKENLER

`n = 8` `cikan→x = 30` `deger.x = 9` `i = 3` `j = 7`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 8

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	10

İkinci çağrı: `28, 15, 9, 11, 14, 3, 6, 10`. Bu kez sondaki öğe (10) yaprağa kadar inecek.

Adım 1 / 18

```
▶ bool maksimum_yigin_al(eleman *cikan) {
  if (YIGIN_BOSMU(n)) return false;
  *cikan = yigin[1];          /* kök döndürülecek */
  eleman deger = yigin[n--];  /* yeri aranacak */
  int i = 1, j = 2;
  while (j <= n) {
    if (j < n && yigin[j].x < yigin[j + 1].x) j++;
    if (deger.x >= yigin[j].x) break;
    yigin[i] = yigin[j];      /* çocuğu yukarı al */
    i = j; j *= 2;
  }
  yigin[i] = deger;
  return true;
}
```

DEĞİŞKENLER

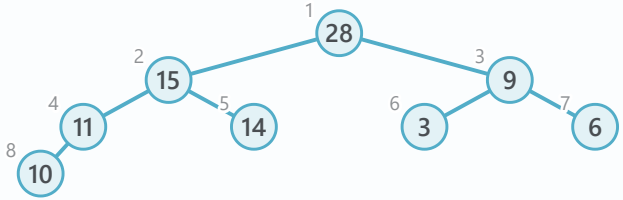
n = 8

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 8

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	10

`YIGIN_BOSMU(8)` yanlış; alınacak öğe var.

Adım 2 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

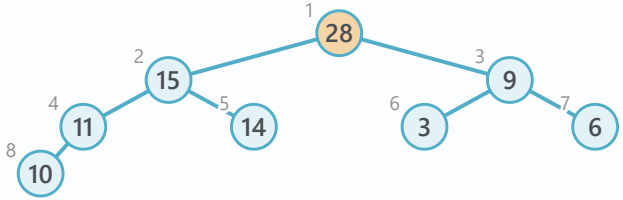
n = **8**

ÇAĞRI YIĞINI

maksimum_yigin_al(&e)

main()

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 8 · çıkan = 28

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	10

***cikan = yigin[1]** = 28: tanım gereği ilk alınacak öğe dizinin ilk öğesidir.

Adım 3 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

DEĞİŞKENLER

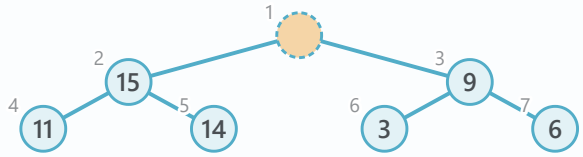
`n = 8` `cikan→x = 28`

ÇAĞRI YIĞINI

`maksimum_yigin_al(&e)`

`main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	

deger = yigin[n--] = 10, n = 7. Tam ağaç bozulmasın diye sondaki öğe çıkarıldı; ona kökten başlayarak yer aranacak.

Adım 4 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
▶    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

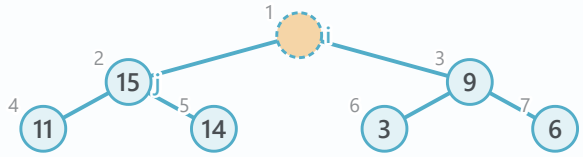
DEĞİŞKENLER

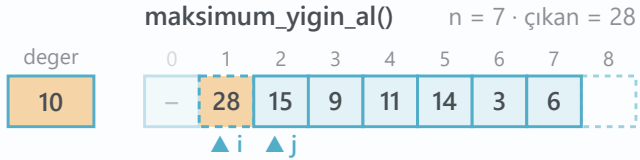
`n = 7` `cikan→x = 28` `deger.x = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)





i = 1 (deger'in aday konumu), **j = 2** (i'nin sol çocuğu).

Adım 5 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 ▶   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

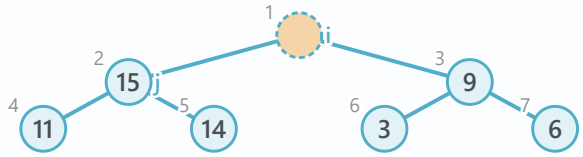
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	
	▲i	▲j						

j = 2 ≤ n = 7: i'nin çocuğu var, döngüye girilir.

Adım 6 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];           /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

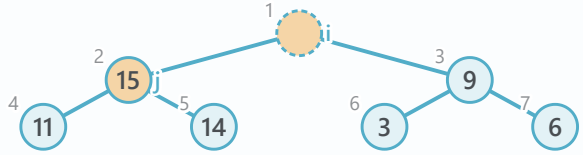
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	
	▲i	▲j						

Sol çocuk (15) ≥ sağ çocuk (9): j = 2 kalır.

Adım 7 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
▶      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

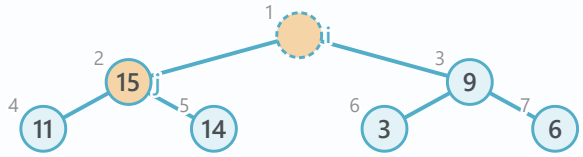
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	28	15	9	11	14	3	6	
	▲i	▲j						

10 ≥ 15? Hayır: büyük çocuk yukarı çıkmalı.

Adım 8 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];           /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 ▶     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

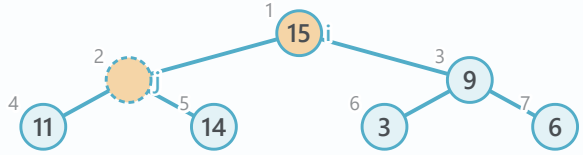
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	15	15	9	11	14	3	6	
	▲i	▲j						

yigin[1] = yigin[2]: 15 bir düzey yukarı taşındı; boşluk 2. konuma indi.

Adım 9 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 ▶     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

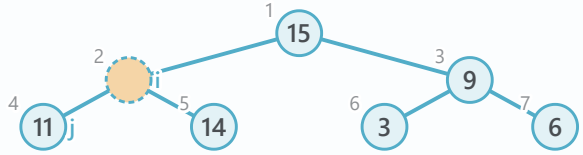
DEĞİŞKENLER

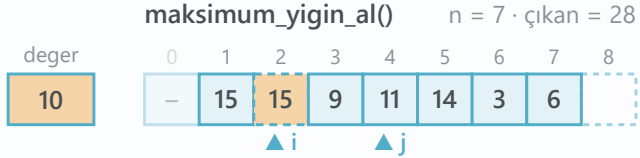
`n = 7` `cikan→x = 28` `deger.x = 10` `i = 1` `j = 2`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)





i = 2, **j = 4**: taşınan değerin eski konumuna ilerlenir.

Adım 10 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
▶      i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

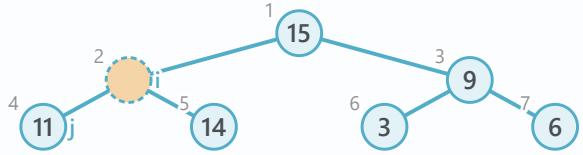
DEĞİŞKENLER

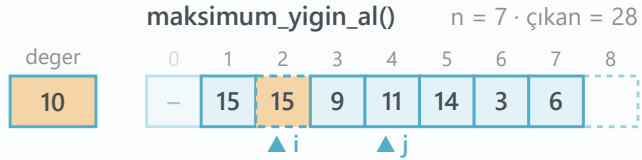
`n = 7` `cikan→x = 28` `deger.x = 10` `i = 2` `j = 4`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)





j = 4 ≤ n = 7: i'nin çocuğu var, döngüye girilir.

Adım 11 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];           /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

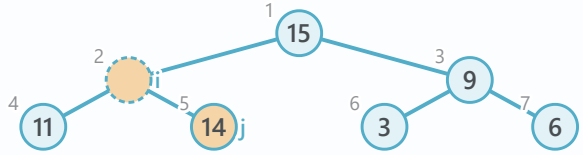
DEĞİŞKENLER

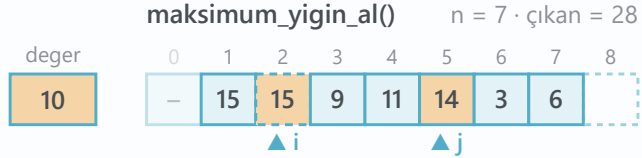
`n = 7` `cikan→x = 28` `deger.x = 10` `i = 2` `j = 4`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)





Sağ çocuk daha büyük (11 < 14) → **j++**, j = 5.

Adım 12 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
▶      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

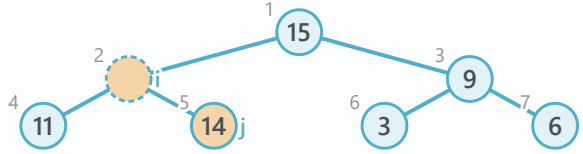
DEĞİŞKENLER

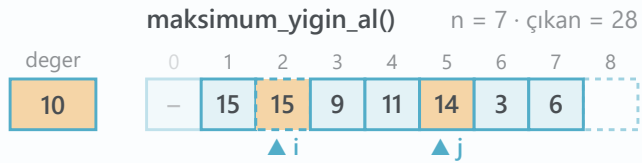
`n = 7` `cikan→x = 28` `deger.x = 10` `i = 2` `j = 5`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)





10 ≥ 14? Hayır: büyük çocuk yukarı çıkmalı.

Adım 13 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];           /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
▶      if (deger.x >= yigin[j].x) break;
9      yigin[i] = yigin[j];       /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

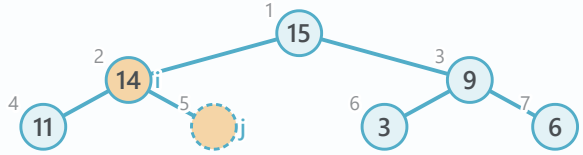
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 2` `j = 5`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	15	14	9	11	14	3	6	
		▲i			▲j			

yigin[2] = yigin[5]: 14 bir düzey yukarı taşındı; boşluk 5. konuma indi.

Adım 14 / 18

```
1  bool maksimum_yigin_al(eleman *cikan) {
2    if (YIGIN_BOSMU(n)) return false;
3    *cikan = yigin[1];          /* kök döndürülecek */
4    eleman deger = yigin[n--];  /* yeri aranacak */
5    int i = 1, j = 2;
6    while (j <= n) {
7      if (j < n && yigin[j].x < yigin[j + 1].x) j++;
8      if (deger.x >= yigin[j].x) break;
▶      yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

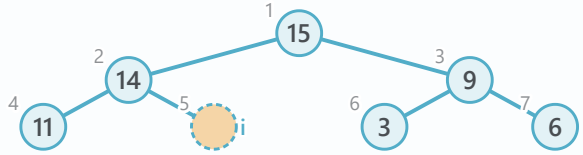
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 2` `j = 5`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	15	14	9	11	14	3	6	

▲ i

i = 5, **j = 10**: taşınan değerin eski konumuna ilerlenir.

Adım 15 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
▶      i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

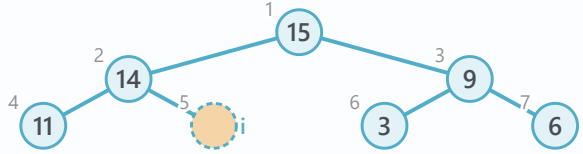
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 5` `j = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

deger: 10

0	1	2	3	4	5	6	7	8
–	15	14	9	11	14	3	6	

▲ i

j = 10 > n = 7: i bir yaprak; döngüden çıkılır.

Adım 16 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
▶6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
13   return true;
14 }
```

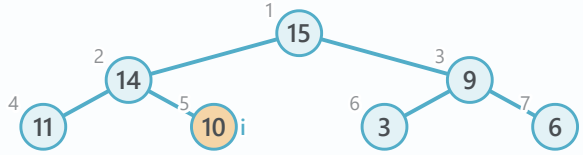
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 5` `j = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

0	1	2	3	4	5	6	7	8
–	15	14	9	11	10	3	6	

▲ i

yigin[5] = deger (10). Heap: `15, 14, 9, 11, 10, 3, 6` .

Adım 17 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
▶  yigin[i] = deger;
13   return true;
14 }
```

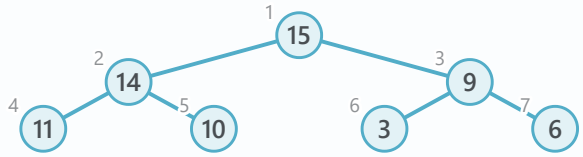
DEĞİŞKENLER

`n = 7` `cikan→x = 28` `deger.x = 10` `i = 5` `j = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

Kod izleme: ikinci `maksimum_yigin_al` (28)



maksimum_yigin_al() n = 7 · çıkan = 28

0	1	2	3	4	5	6	7	8
–	15	14	9	11	10	3	6	

return true; çağıran `cikan` içinde **28** değerini alır. Yol boyu ≤ ⌊lg n⌋ düzey → O(lg n).

Adım 18 / 18

```
 1 bool maksimum_yigin_al(eleman *cikan) {
 2   if (YIGIN_BOSMU(n)) return false;
 3   *cikan = yigin[1];          /* kök döndürülecek */
 4   eleman deger = yigin[n--];  /* yeri aranacak */
 5   int i = 1, j = 2;
 6   while (j <= n) {
 7     if (j < n && yigin[j].x < yigin[j + 1].x) j++;
 8     if (deger.x >= yigin[j].x) break;
 9     yigin[i] = yigin[j];      /* çocuğu yukarı al */
10     i = j; j *= 2;
11   }
12   yigin[i] = deger;
▶    return true;
14 }
```

DEĞİŞKENLER

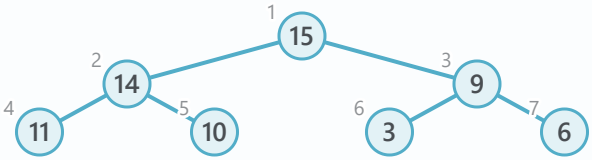
`n = 7` `cikan→x = 28` `deger.x = 10` `i = 5` `j = 10`

ÇAĞRI YIĞINI

- `maksimum_yigin_al(&e)`
- `main()`

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır



durum n = 7

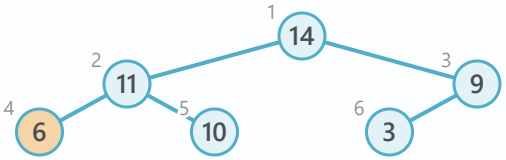
0	1	2	3	4	5	6	7	8	9
–	15	14	9	11	10	3	6		

Şu an: `15, 14, 9, 11, 10, 3, 6`. Çıkanlar: 30, 28.

Adım 1 / 8

Hocadan *"İş inada bindi! ☺"*

Heap boşalana dek alınır: çıkış sırası azalan sıradır



sil → 15 n = 6 · çıkan = 15

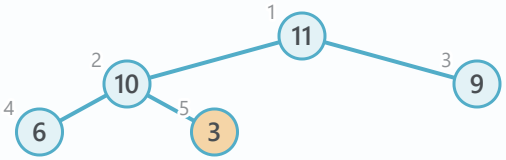
0	1	2	3	4	5	6	7	8	9
–	14	11	9	6	10	3			

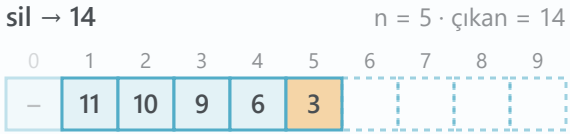
15 alındı; sondaki öğe 6 4. konuma yerleşti. Çıkış sırası: 30, 28, 15

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır





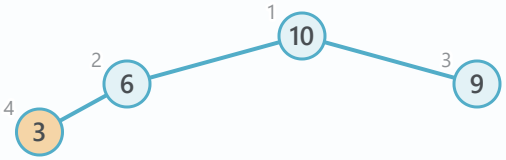
14 alındı; sondaki öğe 3 5. konuma yerleşti. Çıkış sırası: 30, 28, 15, 14

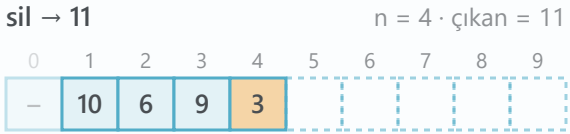
Adım 3 / 8

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır





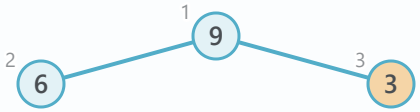
11 alındı; sondaki öğe 3 4. konuma yerleşti. Çıkış sırası: 30, 28, 15, 14, 11

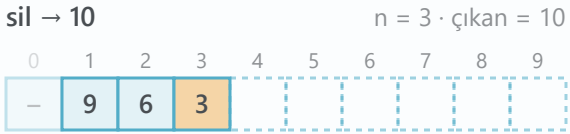
Adım 4 / 8

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır



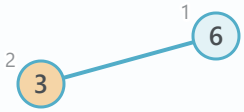


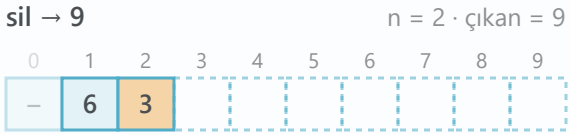
10 alındı; sondaki öğe 3 3. konuma yerleşti. Çıkış sırası: 30, 28, 15, 14, 11, 10

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır





9 alındı; sondaki öğe 3 2. konuma yerleşti. Çıkış sırası: 30, 28, 15, 14, 11, 10, 9

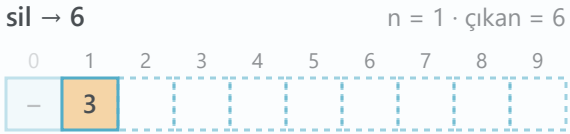
Adım 6 / 8

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır





6 alındı; sondaki öğe 3 1. konuma yerleşti. Çıkış sırası: 30, 28, 15, 14, 11, 10, 9, 6

Hocadan *"İş inada bindi! ☺"*

SİLME

Heap boşalana dek alınır: çıkış sırası azalan sıradır

sil → 3

n = 0 · çıkan = 3



3 alındı; sondaki öğe 3 son öğeydi, heap boşaldı. Çıkış sırası: 30, 28, 15, 14, 11, 10, 9, 6, 3

Adım 8 / 8

Hocadan *"İş inada bindi! ☺"*

SİLME

Silme maliyeti ve sondaki öğenin yolculuğu

Araya girebilir

1. silmede 9, üç düzey inmeden 3. konuma yerleşti.

En sona konabilir

2. silmede 10 yaprağa (5. konum) kadar indi: en kötü durum.

Maliyet

Döngü her turda bir düzey iner: en çok $\lfloor \lg n \rfloor$ tur, tur başına 2 karşılaştırma → **O(lg n)**.

Yan ürün

n kez silmek öğeleri **azalan sırada** verir: bu, heap sort'un fikridir.

Silmede en iyi durum da Ω(1)'dir: sondaki öğe kökün iki çocuğundan büyük/eşitse hemen yerleşir (ör. heap'te 2 öğe kalmışken).

Yordamları daha güzel yazmak: global yerine yapı

```
typedef struct { eleman *a; int n, kapasite; } Yigin;

Yigin *yigin_olustur(int kapasite) {
  Yigin *h = malloc(sizeof *h);
  if (h == NULL) return NULL;
  h->a = malloc((kapasite + 1) * sizeof *h->a);
  if (h->a == NULL) { free(h); return NULL; }
  h->n = 0; h->kapasite = kapasite;
  return h;
}
void yigin_yok_et(Yigin *h) { if (h) { free(h->a); free(h); } }
```

Öneri

Başarı `TRUE` / `FALSE` ile bildirilsin; öğe ve `n` referansla aktarılsın.

Burada

Dizi ve `n` bir yapıda: birden çok heap olabilir; `malloc` hep denetlenir.

Hocadan *"Kitaptaki kodlar zaten çok doğru kodlar değil. Ama tanımlar doğru olduğundan, benim gördüğüm kadarıyla veri yapılarını öğrenmek için son derece ideal bir kitap..."*

Min hâlleri: ad ve karşılaştırmalar çevrilir

```
void minimum_yigin_ekle(eleman item) {
  /* ... dolu denetimi aynı ... */
  int i = ++n;
  while (i != 1) {
    if (item.x >= yigin[i / 2].x) break; /* >= ! */
    yigin[i] = yigin[i / 2];
    i /= 2;
  }
  yigin[i] = item;
}
```

Yer	Maks.	Min.
ekle: dur	`item ≤ ebeveyn`	`item ≥ ebeveyn`
al: çocuk	büyüğü	küçüğü
al: dur	`deger ≥ [j]`	`deger ≤ [j]`

Hocadan *"Yordamın adı değiştirilip küçük ve büyük işaretleri de tersine çevrilirse başarılı bir şekilde min heap yazılmış olur."*

SİLME

Öncelik zamanla değişirse: uçakların yakıtı

Gözlem

Her uçağın saniyede harcadığı yakıt farklıdır: her saniye kesme (interrupt) ile yakıtlar günlenmeli.

Heap'te karşılığı

Önceliği artan öğe **yukarı**, azalan öğe **aşağı** kaydırılır: O(lg n). Konum k bir indeks tablosunda tutulur.

```
/* k konumundaki öğenin önceliği arttı: yukarı kaydır */
void oncelik_artir(int k, int yeni) {
  eleman e = yigin[k];
  e.x = yeni;
  while (k != 1 && e.x > yigin[k / 2].x) {
    yigin[k] = yigin[k / 2];
    k /= 2;
  }
  yigin[k] = e;
}
```

Hocadan *"Bir uçağın bir saniyede 10 galon (37,85 litre) yakıtı giderken diğeri 10 galonla bir saat uçabilir belki."*

KARŞILAŞTIRMA

Niçin heap? Basit dizi ya da liste olmaz mı?

#	Yapı	Ekle	Sil (en büyük)	Neden
1	Sırasız dizi	Θ(1)	Θ(n)	Sona eklenir; en büyüğü bulmak için tamamına bakılır.
2	Sırasız liste	Θ(1)	Θ(n)	Başa eklenir; yine hepsinde gezinilir.
3	Sıralı dizi (küçükten büyüğe)	O(n)	Θ(1)	binsearch O(lg n) + n kaydırma; en büyük sonda.
4	Sıralı bağlı liste (büyükten küçüğe)	O(n)	Θ(1)	Yeri bulunur; baştan silinir, baş günlenir.
5	Maksimum heap	O(lg n)	O(lg n)	Yalnızca bir kök–yaprak yolu.

En iyi (Ω) ve en kötü (O) durum aynı derecedense sınır sıkıdır ve Θ ile ifade edilir. Sıralı dizide binsearch kullanılmazsa ekleme O(n) değil Θ(n) olur.

Hocadan *"'En büyük değeri bir max değişkeniyle tutsak?' O uçak inince ne yapılacak? Öyleyse sıralı dizi kullanmak gerekirdi."*

KARŞILAŞTIRMA

Maliyet tablosunu sayılara dökelim

#	Ekle	Sil
1 · sırasız dizi	1	2.000.000
2 · sırasız liste	1	2.000.000
3 · sıralı dizi	2.000.000	1
4 · sıralı liste	2.000.000	1
5 · maks. heap	≈ 21	≈ 21

Bir komut = 1 saniye varsayımı; lg 2.000.000 ≈ 20,9 (yuvarlanarak 21).

Hocadan *"5 farklı yaklaşımı hiç kod yazmadan, tuşlara dokunmadan analiz ediyorum."*

Hocadan *"Ben 2 milyon sayısını çok seviyorum, çünkü logaritmasını almak kolay. […] max heap yapısıyla gerçeklediğim programı şaka olsun diye 100.000 kez çalıştırıyorum. Öteki yapılarla 1 kez çalıştırmak kadar sürüyor."*

KARŞILAŞTIRMA

“Kim anlayacak benim algoritmanın kötü olduğunu?”

Hocadan *“‘Komut saniye değil, mikro saniye düzeyindedir.’ denebilir. […] ama bir işletim sisteminin parçası olduğunda aralarında ne kadar fark var! ‘Kardeşim! Havaalanına yarım saatte, bir saatte bir uçak geliyor. Kim anlayacak benim algoritmanın kötü olduğunu?’ […] Uzmanlık isteyen işlerde ise o şekilde yazılmış kodun akademik ve bilimsel hiçbir değeri yoktur.”*

Hocadan *“Problem böyle olmalı: Bir öğe ekle, en küçüğünü hemen ver. […] Hocam bana sıralı bağlı liste hoş geldi diyen varsa hâlâ, yapacak bir şey yok.”*

Veri yapısı, işlem karışımına göre seçilir. Hem sık ekleme hem sık “en küçüğü ver” isteyen bir problemde iki işlemi de O(lg n)'de yapan heap, birini Θ(1) diğerini Θ(n) yapan yapılardan üstündür.

KARŞILAŞTIRMA

İleri düzey: binomial ve Fibonacci heap'leri

İşlem	İkili heap	Binomial heap	Fibonacci heap*
ekle	O(lg n)	O(lg n)	O(1)
en küçüğü bul	O(1)	O(lg n)	O(1)
en küçüğü sil	O(lg n)	O(lg n)	O(lg n)
anahtar azalt	O(lg n)	O(lg n)	O(1)
iki heap'i birleştir	O(n)	O(lg n)	O(1)

* Fibonacci heap süreleri **amortize** (ortalama, işlem dizisi üzerinden) sürelerdir (CLRS). Dijkstra ve Prim gibi çok sayıda "anahtar azalt" yapan algoritmalarda kuramsal kazanç sağlar.

Hocadan *"Bu yapı binary heap. Bundan daha ileri düzeyde fibonacci heap, binomial heap algoritmaları var. Daha hızlanıyor. Adamlar lgn yeter dememişler. — Citius! Altius! Fortius! (Daha hızlı! Daha yükseğe! Daha güçlü!)"*

Heapify: diziyi yerinde heap'e çevirmek

```
void yigin_kur(eleman a[], int n) {
  for (int k = n / 2; k >= 1; k--) {
    eleman deger = a[k];
    int i = k, j = 2 * k;
    while (j <= n) {
      if (j < n && a[j].x < a[j + 1].x) j++;
      if (deger.x >= a[j].x) break;
      a[i] = a[j];
      i = j; j *= 2;
    }
    a[i] = deger;
  }
}
```

Fikir

Yapraklar (k > n/2) zaten tek öğelik heap'tir. `k = n/2` 'den köke doğru her düğüm **aşağı kaydırılır**: iki çocuğu heap olan bir kök düzeltilir.

Ne kazandırır?

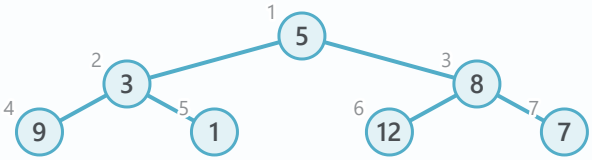
n kez `ekle` O(n lg n) sürer; aşağıdan yukarı kurma **O(n)**.

İç döngü

`maksimum_yigin_al` içindeki döngünün aynısı (Horowitz'de `adjust`).

HEAPİFY · KODU ADIM ADIM ÇALIŞTIR

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7)

0	1	2	3	4	5	6	7
–	5	3	8	9	1	12	7

Girdi: `5, 3, 8, 9, 1, 12, 7` (henüz heap değil). n/2 = 3'ten başlanır.

Adım 1 / 15

```
▶ void yigin_kur(eleman a[], int n) {
2   for (int k = n / 2; k >= 1; k--) {
3     eleman deger = a[k];
4     int i = k, j = 2 * k;
5     while (j <= n) {
6       if (j < n && a[j].x < a[j + 1].x) j++;
7       if (deger.x >= a[j].x) break;
8       a[i] = a[j];
9       i = j; j *= 2;
10    }
11    a[i] = deger;
12  }
13 }
```

DEĞİŞKENLER

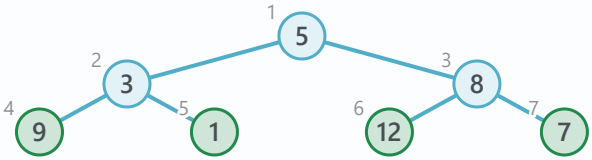
n = 7

ÇAĞRI YIĞINI

yigin_kur(a, 7)

main()

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 3

0	1	2	3	4	5	6	7
–	5	3	8	9	1	12	7

▲ k

k = 3: 3 köklü alt ağaç düzeltilir. Yeşil düğümler zaten birer heap'in köküdür (yapraklar ve daha önce düzeltilenler).

Adım 2 / 15

```
1  void yigin_kur(eleman a[], int n) {
▶    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

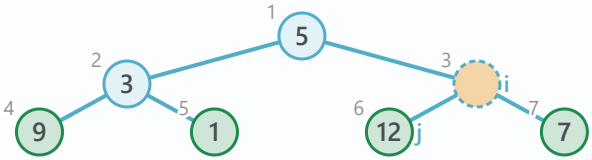
n = **7** k = **3**

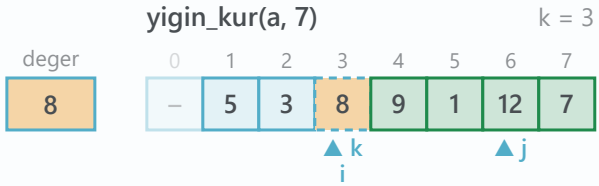
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





deger = a[3] = 8; i = 3, j = 6. Silmedeki aşağı kaydırmanın aynısı başlıyor.

Adım 3 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
▶      eleman deger = a[k];
▶      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

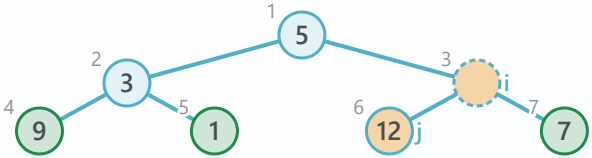
`n = 7` `k = 3` `i = 3` `j = 6` `deger.x = 8`

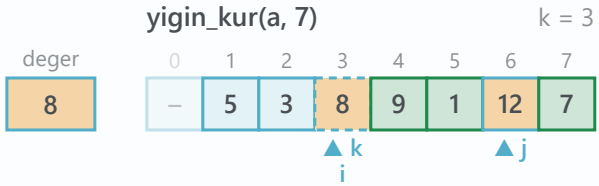
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





Çocuklar 12 ve 7: büyüğü **a[6] = 12**.

Adım 4 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
▶        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

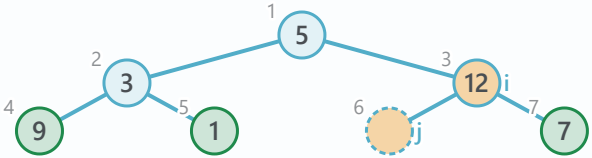
`n = 7` `k = 3` `i = 3` `j = 6` `deger.x = 8`

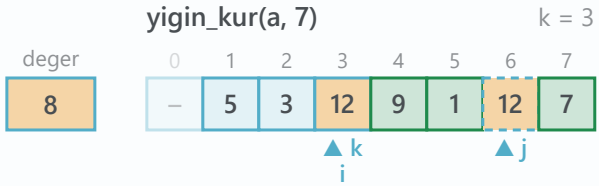
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





8 < 12 → **a[3] = a[6]**; boşluk 6. konuma iner.

Adım 5 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
▶        if (deger.x >= a[j].x) break;
▶        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

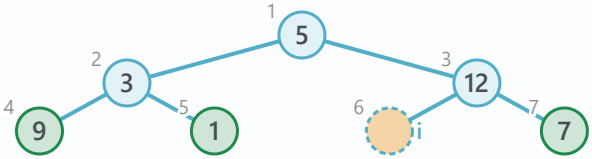
`n = 7` `k = 3` `i = 3` `j = 6` `deger.x = 8`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 3

deger: 8

0	1	2	3	4	5	6	7
–	5	3	12	9	1	12	7
			▲ k			▲ i	

i = 6, **j = 12**: boşluğun yeni yerinden devam.

Adım 6 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
▶        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

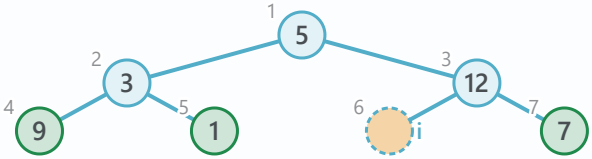
`n = 7` `k = 3` `i = 6` `j = 12` `deger.x = 8`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 3

deger
8

0	1	2	3	4	5	6	7
–	5	3	12	9	1	12	7
			▲ k			▲ i	

j = 12 > n: i = 6 yaprak, döngü biter.

Adım 7 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

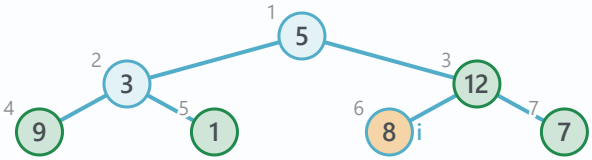
`n = 7` `k = 3` `i = 6` `j = 12` `deger.x = 8`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 3

0	1	2	3	4	5	6	7
–	5	3	12	9	1	8	7
			▲ k			▲ i	

a[6] = deger (8). 3 köklü alt ağaç artık bir heap: `5, 3, 12, 9, 1, 8, 7` .

Adım 8 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
▶      a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

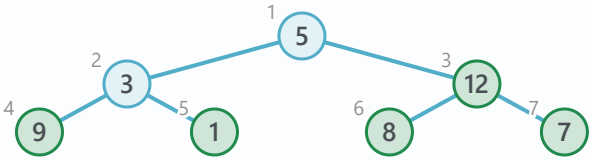
`n = 7` `k = 3` `i = 6` `j = 12` `deger.x = 8`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 2

0	1	2	3	4	5	6	7
–	5	3	12	9	1	8	7
		▲ k					

k = 2: 2 köklü alt ağaç düzeltilir. Yeşil düğümler zaten birer heap'in köküdür (yapraklar ve daha önce düzeltilenler).

Adım 9 / 15

```
1  void yigin_kur(eleman a[], int n) {
▶    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

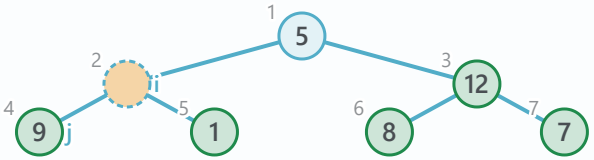
n = **7** k = **2**

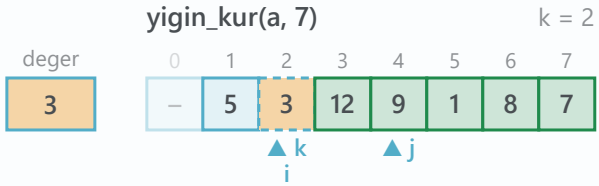
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





deger = a[2] = 3; i = 2, j = 4. Silmedeki aşağı kaydırmanın aynısı başlıyor.

Adım 10 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
▶      eleman deger = a[k];
▶      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

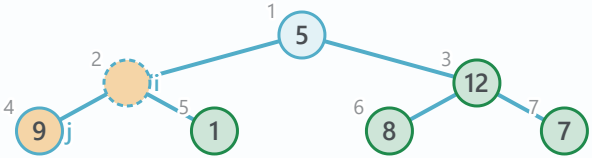
`n = 7` `k = 2` `i = 2` `j = 4` `deger.x = 3`

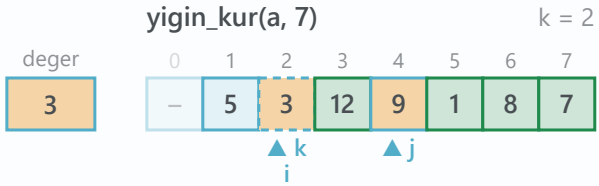
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





Çocuklar 9 ve 1: büyüğü **a[4] = 9**.

Adım 11 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
▶        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

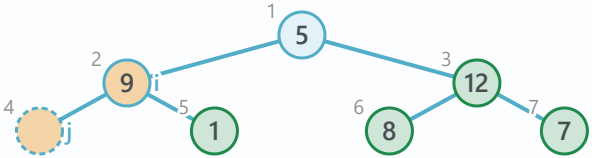
DEĞİŞKENLER

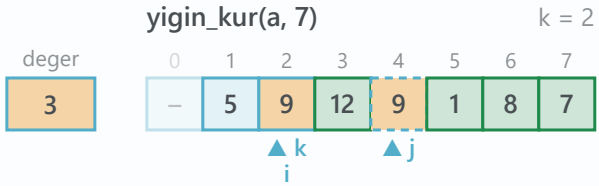
`n = 7` `k = 2` `i = 2` `j = 4` `deger.x = 3`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





3 < 9 → **a[2] = a[4]**; boşluk 4. konuma iner.

Adım 12 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
▶        if (deger.x >= a[j].x) break;
▶        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

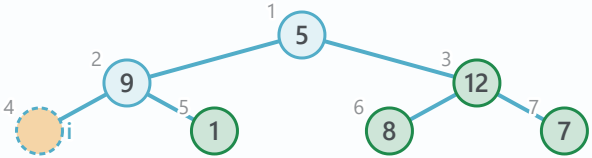
DEĞİŞKENLER

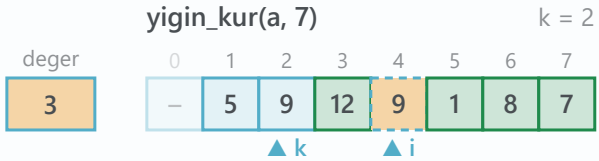
`n = 7` `k = 2` `i = 2` `j = 4` `deger.x = 3`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2





i = 4, **j = 8**: boşluğun yeni yerinden devam.

Adım 13 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
▶        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

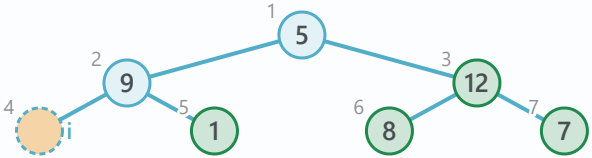
`n = 7` `k = 2` `i = 4` `j = 8` `deger.x = 3`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 2

deger: 3

0	1	2	3	4	5	6	7
–	5	9	12	9	1	8	7
		▲ k		▲ i			

j = 8 > n: i = 4 yaprak, döngü biter.

Adım 14 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

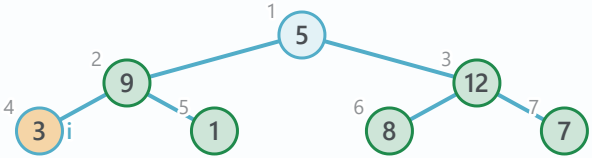
`n = 7` `k = 2` `i = 4` `j = 8` `deger.x = 3`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 3 ve k = 2



yigin_kur(a, 7) k = 2

0	1	2	3	4	5	6	7
–	5	9	12	3	1	8	7
		▲ k		▲ i			

a[4] = deger (3). 2 köklü alt ağaç artık bir heap: `5, 9, 12, 3, 1, 8, 7`.

Adım 15 / 15

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
▶      a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

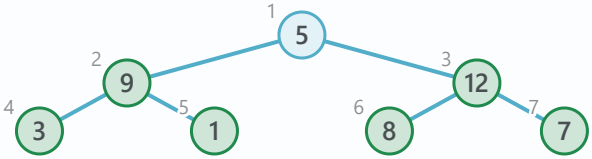
`n = 7` `k = 2` `i = 4` `j = 8` `deger.x = 3`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1



yigin_kur(a, 7) k = 1

0	1	2	3	4	5	6	7
–	5	9	12	3	1	8	7
	▲ k						

k = 1: 1 köklü alt ağaç düzeltilir. Yeşil düğümler zaten birer heap'in köküdür (yapraklar ve daha önce düzeltilenler).

Adım 1 / 11

```
1  void yigin_kur(eleman a[], int n) {
▶    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

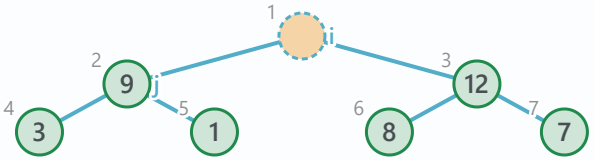
n = **7** k = **1**

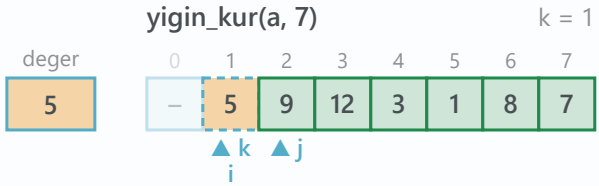
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1





deger = a[1] = 5; i = 1, j = 2. Silmedeki aşağı kaydırmanın aynısı başlıyor.

Adım 2 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
▶      eleman deger = a[k];
▶      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

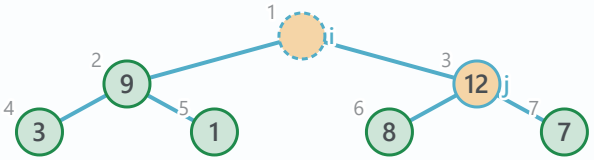
`n = 7` `k = 1` `i = 1` `j = 2` `deger.x = 5`

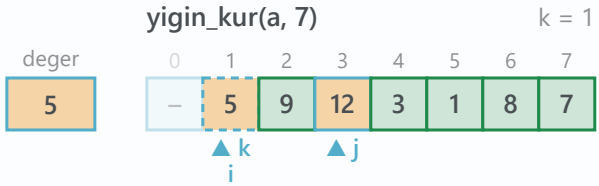
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1





Çocuklar 9 ve 12: büyüğü **a[3] = 12**.

Adım 3 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
▶        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

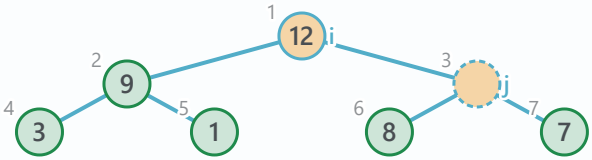
`n = 7` `k = 1` `i = 1` `j = 3` `deger.x = 5`

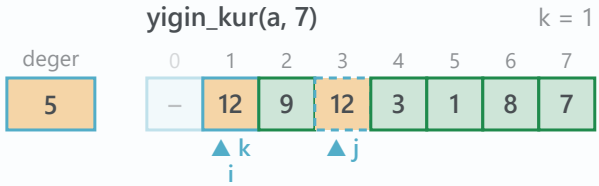
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1





5 < 12 → **a[1] = a[3]**; boşluk 3. konuma iner.

Adım 4 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
▶        if (deger.x >= a[j].x) break;
▶        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

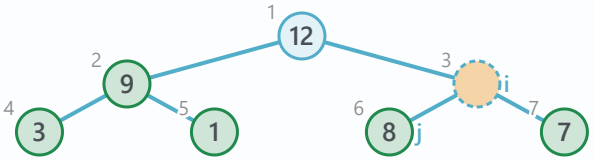
`n = 7` `k = 1` `i = 1` `j = 3` `deger.x = 5`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1



yigin_kur(a, 7) k = 1

deger: 5

0	1	2	3	4	5	6	7
–	12	9	12	3	1	8	7
	▲k		▲i			▲j	

i = 3, **j = 6**: boşluğun yeni yerinden devam.

Adım 5 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
▶        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

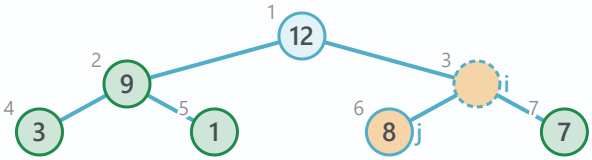
DEĞİŞKENLER

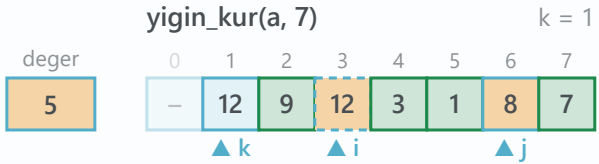
`n = 7` `k = 1` `i = 3` `j = 6` `deger.x = 5`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 1





Çocuklar 8 ve 7: büyüğü **a[6] = 8**.

Adım 6 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
▶        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

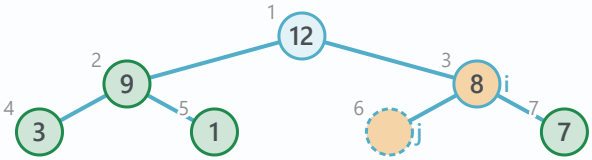
DEĞİŞKENLER

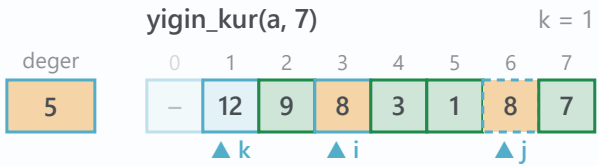
`n = 7` `k = 1` `i = 3` `j = 6` `deger.x = 5`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 1





5 < 8 → **a[3] = a[6]**; boşluk 6. konuma iner.

Adım 7 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
▶        if (deger.x >= a[j].x) break;
▶        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

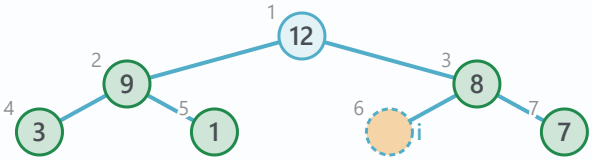
`n = 7` `k = 1` `i = 3` `j = 6` `deger.x = 5`

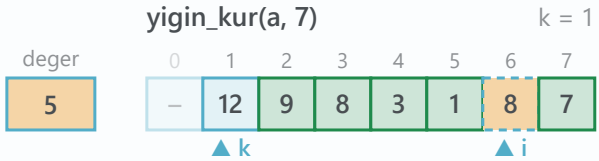
ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1





i = 6, **j = 12**: boşluğun yeni yerinden devam.

Adım 8 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
▶        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

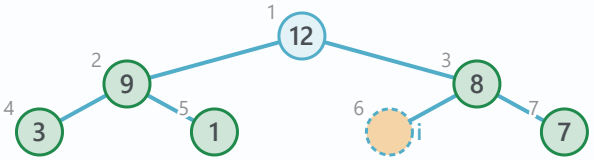
DEĞİŞKENLER

`n = 7` `k = 1` `i = 6` `j = 12` `deger.x = 5`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 1



yigin_kur(a, 7) k = 1

deger: 5

0	1	2	3	4	5	6	7
–	12	9	8	3	1	8	7
	▲ k					▲ i	

j = 12 > n: i = 6 yaprak, döngü biter.

Adım 9 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
▶      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

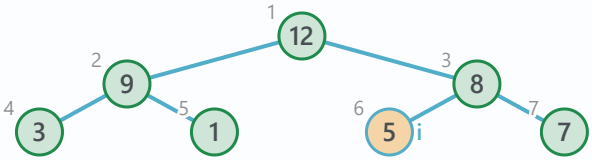
DEĞİŞKENLER

`n = 7` `k = 1` `i = 6` `j = 12` `deger.x = 5`

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Kod izleme: `yigin_kur` — k = 1



yigin_kur(a, 7) k = 1

0	1	2	3	4	5	6	7
–	12	9	8	3	1	5	7
	▲ k					▲ i	

a[6] = deger (5). 1 köklü alt ağaç artık bir heap: `12, 9, 8, 3, 1, 5, 7` .

Adım 10 / 11

```
1  void yigin_kur(eleman a[], int n) {
2    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
▶      a[i] = deger;
12   }
13 }
```

DEĞİŞKENLER

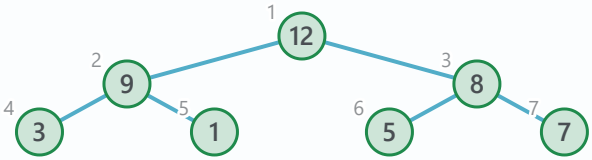
`n = 7` `k = 1` `i = 6` `j = 12` `deger.x = 5`

ÇAĞRI YIĞINI

`yigin_kur(a, 7)`

`main()`

Kod izleme: `yigin_kur` — k = 1



yigin_kur(a, 7)

0	1	2	3	4	5	6	7
–	12	9	8	3	1	5	7

k = 0: döngü bitti. Tüm dizi bir maksimum heap: `12, 9, 8, 3, 1, 5, 7`.

Adım 11 / 11

```
1  void yigin_kur(eleman a[], int n) {
▶    for (int k = n / 2; k >= 1; k--) {
3      eleman deger = a[k];
4      int i = k, j = 2 * k;
5      while (j <= n) {
6        if (j < n && a[j].x < a[j + 1].x) j++;
7        if (deger.x >= a[j].x) break;
8        a[i] = a[j];
9        i = j; j *= 2;
10     }
11     a[i] = deger;
12   }
13 }
```

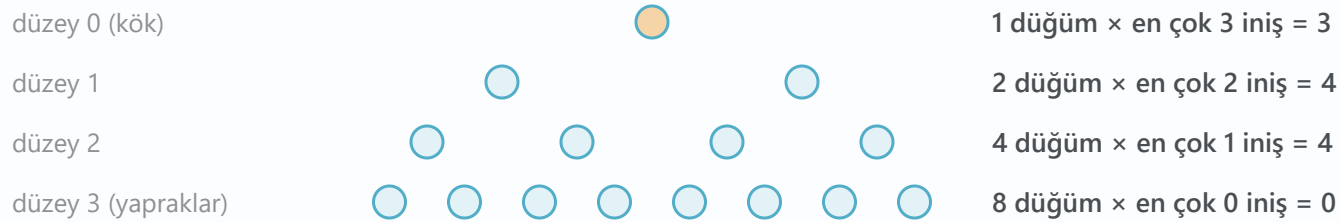
DEĞİŞKENLER

n = 7 | k = –

ÇAĞRI YIĞINI

- `yigin_kur(a, 7)`
- `main()`

Heapify neden O(n)?

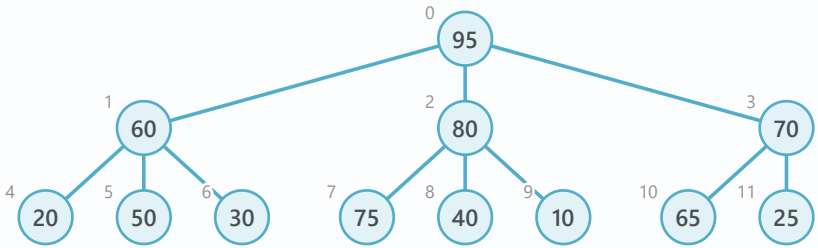


Düğümlerin yarısı yapraktır ve hiç inmez; çeyreği en çok 1, sekizde biri en çok 2 düzey iner... Toplam iş:

$$\Sigma_{h \geq 0} \lceil n / 2^{h+1} \rceil \cdot O(h) = O(n \cdot \Sigma\, h/2^{h}) = O(2n) = O(n)$$

Örnekte 15 düğümlü tam ağaç için toplam iniş en çok 0·8 + 1·4 + 2·2 + 3·1 = 11 < n. Tek tek ekleme ise alttaki düğümleri de köke kadar çıkarabilir: n lg n.

d-ary heap: her düğümün d çocuğu



	İkili (d = 2)	d-ary (0 tabanlı)
ebeveyn	`(i−1)/2`	`(i−1)/d`
çocuklar	`2i+1, 2i+2`	`d·i+1 … d·i+d`
yükseklik	≈ lg n	≈ $\log_d n$
ekle / anahtar artır	O(lg n)	O($\log_d n$) — daha hızlı
kökü sil	O(lg n)	O(d · $\log_d n$) — d çocuk kıyaslanır

Ekleme/anahtar azaltma çok, silme az ise (ör. yoğun çizgelerde Dijkstra) d > 2 seçmek kazançlıdır; sığ ağaç önbellek (cache) dostudur.

İLERİ KONULAR

Öncelikli kuyruk nerelerde kullanılır?

İşletim sistemi
İşlem (process) ve kesme zamanlaması, zamanlayıcı olayları.

Çizge algoritmaları
Dijkstra en kısa yol ve Prim MST: en yakın düğümü seçmek.

Heap sort
Heap kurulur, n kez kök alınır: O(n lg n), yerinde sıralama.

Olay simülasyonu
Olaylar zaman damgasına göre minimum heap'ten çekilir.

k en büyük / en küçük
Boyutu k olan min heap (en küçükler için max heap): akan veride O(n lg k).

Huffman kodlaması
En küçük iki frekansı art arda birleştirir.

ÖZET

Karmaşıklık özeti (n öğeli ikili heap)

İşlem	Zaman	Ek bellek	Not
en büyüğe bak	O(1)	O(1)	`yigin[1]`
ekle	Ω(1) … O(lg n)	O(1)	yukarı kaydırma
en büyüğü al	O(lg n)	O(1)	aşağı kaydırma
önceliği değiştir	O(lg n)	O(1)	konum biliniyorsa
aşağıdan yukarı kurma	O(n)	O(1)	yerinde
n kez ekleyerek kurma	O(n lg n)	O(1)	
keyfi öğeyi ara	O(n)	O(1)	heap arama yapısı değildir

ÖZET

Sık yapılan hatalar

1. 1 tabanlı (`i/2`, `2i`) ve 0 tabanlı (`(i-1)/2`, `2i+1`) formülleri karıştırmak.
2. Dizi boyutunu kapasiteden bir eksik tanımlamak: `yigin[199]` dışarı yazılır.
3. `eleman` döndüren yordamda `return 0;` yazmak (derleme hatası).
4. `j < n` sınamasını unutup `yigin[j+1]`'i okumak (heap dışı; `yigin_kur`'da dizi dışı).
5. Silmede büyük çocuk yerine ilk çocuğu seçmek: heap özelliği bozulur.
6. Heap'i sıralı dizi ya da ikili arama ağacı sanmak.

ÖZET

Sınav tarzı soru: ekle, ekle, … al, al

ÖRNEK VERİYLE

Başlangıçta `yigin[1..6] = 14, 12, 7, 10, 8, 6` olan maksimum heap'e sırasıyla **15, 109, 107, 3, 15** ekleniyor; ardından `maksimum_yigin_al` iki kez çağrılıyor.
(i) Eklemelerden sonra dizinin içeriği nedir? (ii) Alınan öğeler ve son dizi nedir? (iii) Hangi ekleme en az, hangisi en çok kaydırma yaptı?

▶ **Çözüm**

ÖZET

Alıştırmalar

1. Boş bir maksimum heap'e 20, 5, 30, 10, 40 ekleyin; her eklemeden sonra diziyi yazın.
2. `maksimum_yigin_al` yordamını minimum heap için yazın. Hangi satırlar değişir?
3. n = 1000 öğeli bir heap'te ekleme en çok kaç kaydırma yapar?
4. 1 tabanlı bir heap'te 13. düğümün ebeveyni ve çocukları hangi indislerdedir?
5. `5, 3, 8, 9, 1, 12, 7` dizisine yigin_kur yerine art arda `ekle` uygulanırsa sonuç aynı mı olur?
6. Bir heap'te en küçük öğe nerelerde olabilir? Bulma maliyeti nedir?

▶ **Cevapları göster**

ÖZET

Bölüm özeti

Öncelikli kuyruk
İlk alınacak öğe en yüksek (max) ya da en düşük (min) değerli öğedir.

Heap
Tam ikili ağaç + ebeveyn ≥ çocuk. 1 tabanlı dizide ebeveyn $i/2$, çocuklar $2i$ ve $2i+1$.

Ekleme
İlk boş konum açılır, büyük atalar aşağı kaydırılır: $O(\lg n)$.

Silme
Kök alınır, sondaki öğeye kökten yer aranır, büyük çocuklar yukarı alınır: $O(\lg n)$.

Neden heap?
Diğer yapılar bir işlemi $\Theta(n)$'e iter; heap ikisini de logaritmik tutar.

Ötesi
Heapify $O(n)$, d-ary heap, binomial ve Fibonacci heap'leri.

ÖZET

Kaynaklar ve devam

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, Bölüm 5.6 (Heaps), Bölüm 7 (Heap Sort).
- T. H. Cormen vd., *Introduction to Algorithms* (CLRS), Bölüm 6 (Heapsort, öncelikli kuyruklar) ve Fibonacci heap'leri bölümü.
- R. Sedgewick, *Algorithms in C*, Kısım 1–4, Bölüm 9 (Priority Queues and Heapsort).

← Bölüm 12: Kılavuzlu İkili Ağaçlar

Ders sayfası

Bölüm 14: Hash Tabloları →

Son

Bölüm 13: Öncelikli Kuyruk ve Heap

Sonraki bölüm: Bölüm 14: Hash Tabloları · Ders sayfası