

Bölüm 11: Böl ve Fethet Yaklaşımı

Algoritmalar

Böl · Parçala · Yönet · Birleştir

Özyineleme ağacı

Ana teorem

İkili arama ve sıralama

Hızlı üs alma

Quickhull

En yakın nokta çifti

Karatsuba

Strassen

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Paradigma

Böl ve fethet (divide and conquer) yaklaşımının adımlarını açıklamak; bir problemin bu yaklaşıma uygun olup olmadığına karar vermek.

Analiz

Özyineleme bağıntısını (recurrence) kurmak; özyineleme ağacı ve ana teorem (master theorem) ile çözmek.

Geometrik algoritmalar

Quickhull ile dışbükey örtüyü ve şerit yöntemiyle en yakın nokta çiftini adım adım bulmak.

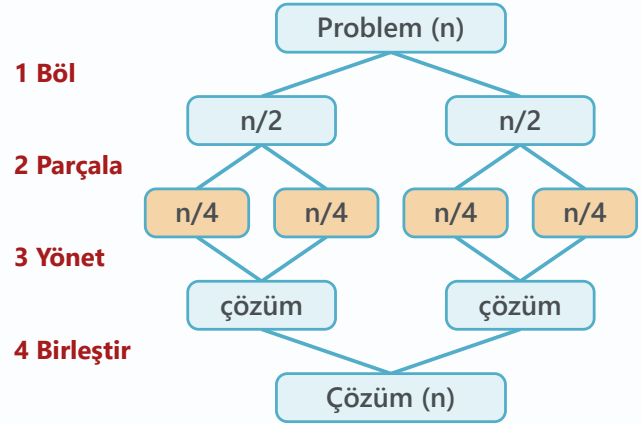
Hızlı çarpma

Karatsuba (tam sayı) ve Strassen (matris) çarpımının, bir özyinelemeli çağrışı eksilterek nasıl kazanç sağladığını göstermek.

Böl-Parçala-Yönet Paradigması

Karmaşık problemleri çözmek için kullanılan etkili bir yöntemdir (*divide and conquer*, böl ve fethet). Şu adımları izler:

1. **Böl:** karmaşık problemi daha küçük, yönetilebilir alt problemlere böl.
2. **Parçala:** alt problemleri, doğrudan çözülebilecek kadar küçülene dek bölmeye devam et (özyineleme).
3. **Yönet:** her bir alt problemi ayrı ayrı çöz.
4. **Birleştir:** alt problemlerin çözümlerini birleştirerek asıl çözümü kur.



Anahtar: alt problemler asıl problemle **aynı türdendir** ama daha küçüktür ve birbirinden **bağımsızdır**; bu yüzden aynı fonksiyon kendini özyinelemeli çağırır.

Neden Böl ve Fethet?

- Karmaşık problemi **küçük ve yönetilebilir** parçalara ayırır; problem daha anlaşılır hale gelir, çözümü kolaylaşır.
- Alt problemlerin çözüm yöntemi (aynı fonksiyon) her düzeyde **tekrar kullanılır**: kısa ve yeniden kullanılabilir kod.
- Sıralama, arama, hesaplamalı geometri, büyük sayı ve matris aritmetiği, yapay zekâ oyunları gibi alanlarda kullanılır.

Avantajlar

- + Çoğu zaman **asimptotik kazanç**: $O(n^2) \rightarrow O(n \log n)$, $O(n) \rightarrow O(\log n)$.
- + Bağımsız alt problemler **paralel** çözülebilir (Bölüm 10).
- + Alt problemler küçüldükçe önbellege sığar: iyi bellek yerelliği.
- + Doğruluk kanıtı tümevarımla kolaydır.

Dezavantajlar

- Özyineleme yükü: çağrı yığını, ek bellek (ör. birleştirme sıralamasında $O(n)$).
- Alt problemler **örtüşüyorsa** aynı iş tekrar tekrar yapılır $\rightarrow$ üstel süre.
- Küçük n için sabit çarpanlar ağır basar; pratikte bir eşiğin altında basit yöntemle geçilir.

Genel Şablon ve Bağıntı

```
Sonuc bolVeFethet(Problem p) {  
    if (p.boyut() <= ESIK)  
        return dogrudanCoz(p);           // taban durumu  
    Problem[] alt = bol(p);               // a alt problem, boyut n/b  
    Sonuc[] cozum = new Sonuc[alt.length];  
    for (int i = 0; i < alt.length; i++)  
        cozum[i] = bolVeFethet(alt[i]); // yönet: özyineleme  
    return birlestir(cozum);             // birleştir  
}
```

Çalışma süresi:

$$T(n) = a \cdot T(n/b) + f(n)$$

a: alt problem sayısı · n/b: alt problem boyutu ·

f(n): bölme + birleştirme maliyeti

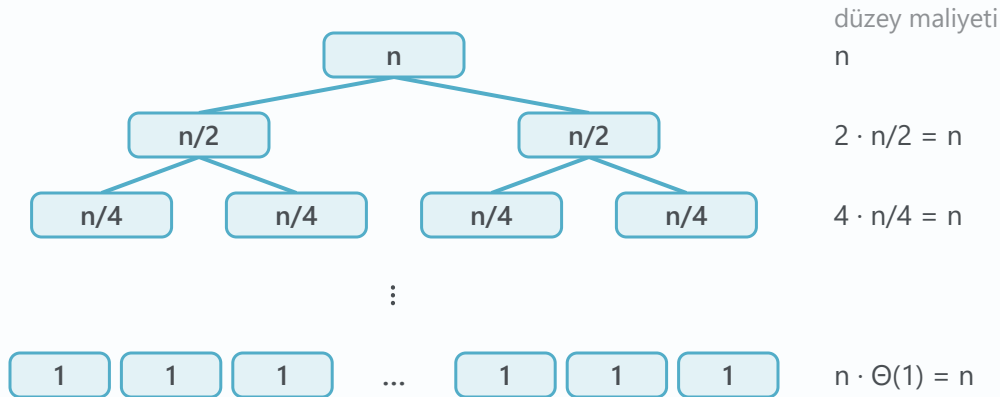
- Taban durumu olmadan özyineleme bitmez.
- Bölme **dengeli** olmalı: $n/2 + n/2$ iyi, $1 + (n - 1)$ kötü.
- Hız kazancı genellikle **a**'yı azaltmaktan (Karatsuba, Strassen) ya da **f(n)**'yi küçültmekten gelir.

Klasik Böl ve Fethet Algoritmaları

Algoritma	Böl	Birleştir	Bağıntı	Süre
İkili arama	ortadaki öğeye bak, bir yarıyı at	yok	$T(n/2) + O(1)$	$O(\log n)$
Birleştirme sıralaması	diziyi ortadan ikiye böl	sıralı yarıları birleştir	$2T(n/2) + O(n)$	$O(n \log n)$
Hızlı sıralama	pivota göre bölümler	yok (yerinde)	$T(k) + T(n-k-1) + O(n)$	ort. $O(n \log n)$
Hızlı üs alma	$x^n = (x^{\lfloor n/2 \rfloor})^2 \cdot (x)$	kare al	$T(n/2) + O(1)$	$O(\log n)$
Quickhull	en uzak noktayla üçgen	örtü parçalarını ekle	$T(n_1) + T(n_2) + O(n)$	ort. $O(n \log n)$
En yakın nokta çifti	x'e göre ortadan böl	orta şeridi tara	$2T(n/2) + O(n)$	$O(n \log n)$
Karatsuba	sayıyı iki yarıya böl	3 çarpımı topla	$3T(n/2) + O(n)$	$O(n^{1,585})$
Strassen	matrisi 4 bloğa böl	7 çarpımı topla	$7T(n/2) + O(n^2)$	$O(n^{2,807})$

Özyineleme Ağacı (Recursion Tree)

Her düğüm bir çağrının **kendi** işini (bölme + birleştirme) gösterir. Düzey toplamaları alt alta toplanınca $T(n)$ bulunur.

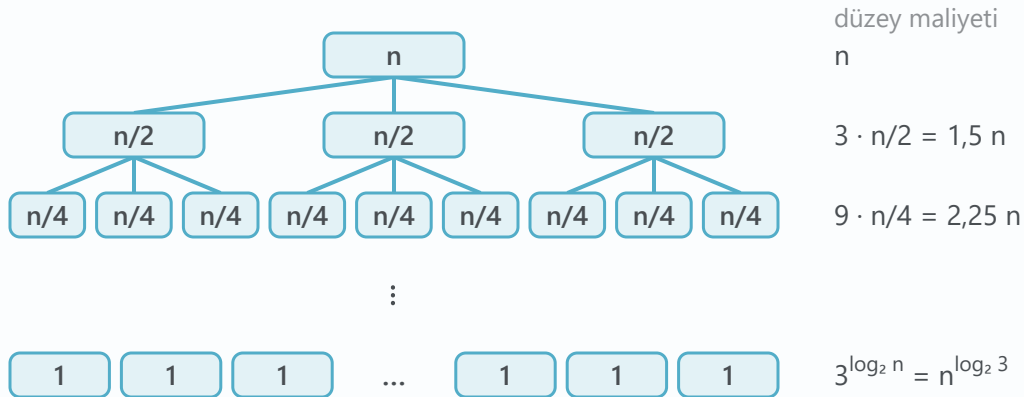


$$\log_2 n \text{ düzey} \times n \rightarrow T(n) = \Theta(n \log n)$$

$T(n) = 2T(n/2) + n$ (birleştirme sıralaması): her düzeyin toplamı n , düzey sayısı $\log_2 n \rightarrow \Theta(n \log n)$.

Özyineleme Ağacı (Recursion Tree)

Her düğüm bir çağrının **kendi** işini (bölme + birleştirme) gösterir. Düzey toplamları alt alta toplanınca $T(n)$ bulunur.

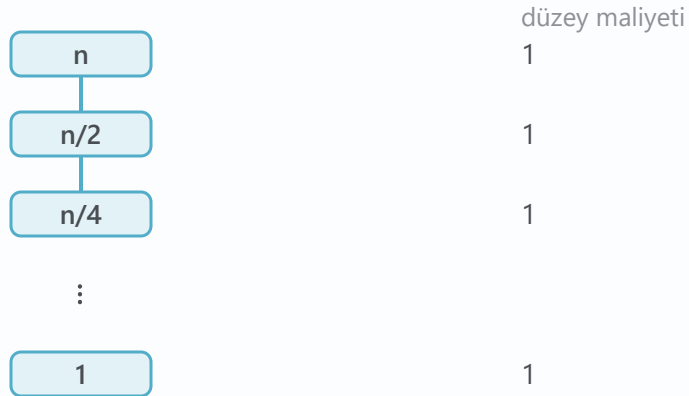


artan geometrik seri $\rightarrow$ yapraklar baskın: $T(n) = \Theta(n^{1,585})$

$T(n) = 3T(n/2) + n$ (Karatsuba): düzey toplamaları 1,5 kat büyür; en çok iş yapraklarda $\rightarrow \Theta(n^{\log_2 3})$.

Özyineleme Ağacı (Recursion Tree)

Her düğüm bir çağrının **kendi** işini (bölme + birleştirme) gösterir. Düzey toplamaları alt alta toplanınca $T(n)$ bulunur.



$$\log_2 n \text{ düzey} \times O(1) \rightarrow T(n) = \Theta(\log n)$$

$$T(n) = T(n/2) + 1 \text{ (ikili arama, hızlı üs alma): tek dal, her düzey } O(1) \rightarrow \Theta(\log n).$$

Ana Teorem (Master Theorem)

$a \geq 1, b > 1$ olmak üzere $T(n) = a \cdot T(n/b) + f(n)$. Kritik üs $c^* = \log_b a$: yaprak sayısı n^{c^*} olur. $f(n)$ ile n^{c^*} karşılaştırılır:

Durum	Koşul	Sonuç	Yorum
1	$f(n) = O(n^{c^* - \epsilon}), \epsilon > 0$	$\Theta(n^{c^*})$	yapraklar baskın
2	$f(n) = \Theta(n^{c^*})$	$\Theta(n^{c^*} \cdot \log n)$	her düzey eşit pay
3	$f(n) = \Omega(n^{c^* + \epsilon})$ ve $a \cdot f(n/b) \leq k \cdot f(n), k < 1$	$\Theta(f(n))$	kök baskın

Uygulanamadığı durumlar: eşit olmayan bölmeler ($T(n) = T(n/3) + T(2n/3) + n$), sabit azaltma ($T(n) = T(n - 1) + n$), arada kalan $f(n)$ ($T(n) = 2T(n/2) + n/\log n$). Bunlarda özyineleme ağacı veya yerine koyma yöntemi kullanılır.

ÖZYİNELEME BAĞINTILARI

Ana Teorem Uygulamaları

Algoritma	a	b	f(n)	$n^{\log_b a}$	Durum	T(n)
İkili arama, hızlı üs alma	1	2	1	$n^0 = 1$	2	$\Theta(\log n)$
Birleştirme sıralaması, en yakın çift	2	2	n	n	2	$\Theta(n \log n)$
Karatsuba	3	2	n	$n^{1,585}$	1	$\Theta(n^{1,585})$
Saf blok matris çarpımı	8	2	n^2	n^3	1	$\Theta(n^3)$
Strassen	7	2	n^2	$n^{2,807}$	1	$\Theta(n^{2,807})$
Ağaç gezinme (dengeli)	2	2	1	n	1	$\Theta(n)$
$T(n) = 2T(n/2) + n^2$	2	2	n^2	n	3	$\Theta(n^2)$

Karatsuba ve Strassen'in tüm kazancı **a**'yı bir azaltmaktan gelir: $4 \rightarrow 3$ ve $8 \rightarrow 7$. Durum 1'de üs doğrudan $\log_b a$ 'dır.

İkili Arama (Binary Search): 23'ü Ara

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91
↑ alt				↑ orta					↑ ust

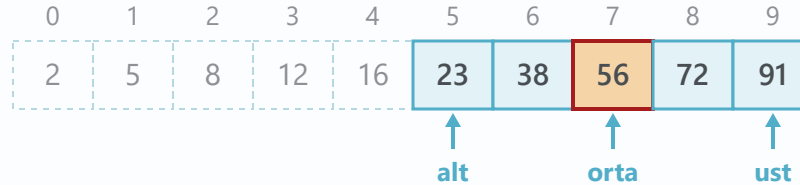
alt = 0, ust = 9 → orta = 4: dizi[4] = 16 < 23 → **sol yarı atılır**, alt = 5.

Adım 1 / 3

```
int ara(int[] dizi, int alt, int ust, int aranan) {  
    if (alt > ust) return -1;           // boş aralık  
    int orta = (alt + ust) / 2;  
    if (dizi[orta] == aranan) return orta;  
    if (dizi[orta] < aranan)  
        return ara(dizi, orta + 1, ust, aranan); // sağ yarı  
    return ara(dizi, alt, orta - 1, aranan);    // sol yarı  
}
```

- **Böl:** ortadaki öge. **Yönet:** yalnızca **bir** yarıda devam. **Birleştir:** yok.
- $T(n) = T(n/2) + O(1) = O(\log n)$; dizi sıralı olmalı.
- Tek dallı olduğu için "azalt ve fethet" (decrease and conquer) de denir.

İkili Arama (Binary Search): 23'ü Ara



alt = 5, ust = 9 → orta = 7: dizi[7] = 56 > 23 → **sağ yarı atılır**, ust = 6.

Adım 2 / 3

```
int ara(int[] dizi, int alt, int ust, int aranan) {  
    if (alt > ust) return -1;           // boş aralık  
    int orta = (alt + ust) / 2;  
    if (dizi[orta] == aranan) return orta;  
    if (dizi[orta] < aranan)  
        return ara(dizi, orta + 1, ust, aranan); // sağ yarı  
    return ara(dizi, alt, orta - 1, aranan);    // sol yarı  
}
```

- **Böl:** ortadaki öge. **Yönet:** yalnızca **bir** yarıda devam. **Birleştir:** yok.
- $T(n) = T(n/2) + O(1) = O(\log n)$; dizi sıralı olmalı.
- Tek dallı olduğu için "azalt ve fethet" (decrease and conquer) de denir.

İkili Arama (Binary Search): 23'ü Ara

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

↑ ↑
alt,orta ust

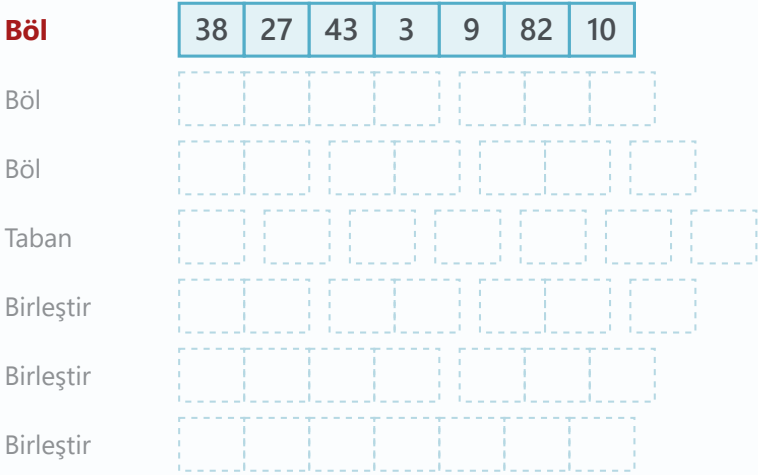
alt = 5, ust = 6 → orta = 5: dizi[5] = **23** = **23** → bulundu. Yalnızca 3 karşılaştırma (doğrusal aramada 6).

Adım 3 / 3

```
int ara(int[] dizi, int alt, int ust, int aranan) {  
    if (alt > ust) return -1;           // boş aralık  
    int orta = (alt + ust) / 2;  
    if (dizi[orta] == aranan) return orta;  
    if (dizi[orta] < aranan)  
        return ara(dizi, orta + 1, ust, aranan); // sağ yarı  
    return ara(dizi, alt, orta - 1, aranan);    // sol yarı  
}
```

- **Böl:** ortadaki öge. **Yönet:** yalnızca **bir** yarıda devam. **Birleştir:** yok.
- $T(n) = T(n/2) + O(1) = \mathbf{O(\log n)}$; dizi sıralı olmalı.
- Tek dallı olduğu için "azalt ve fethet" (decrease and conquer) de denir.

Birleştirme Sıralaması (Merge Sort)



Dizi: [38, 27, 43, 3, 9, 82, 10], n = 7.

Birleştirme Sıralaması (Merge Sort)

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

--	--	--	--	--	--	--

Taban

--	--	--	--	--	--	--

Birleştir

--	--	--	--	--	--	--

Birleştir

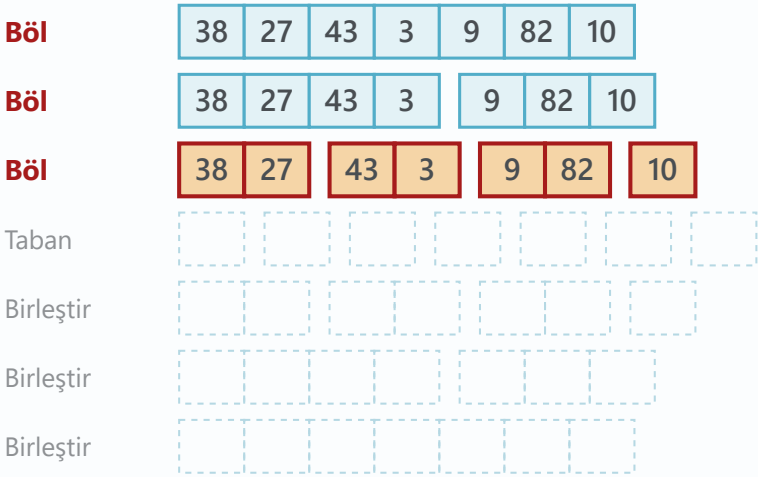
--	--	--	--	--	--	--

Birleştir

--	--	--	--	--	--	--

Böl: $\text{orta} = (0 + 6)/2 = 3 \rightarrow [38 \ 27 \ 43 \ 3] \text{ ve } [9 \ 82 \ 10].$

Birleştirme Sıralaması (Merge Sort)



Yarılar yeniden bölünür: [38 27] [43 3] [9 82] [10].

Birleştirme Sıralaması (Merge Sort)

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Taban

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Birleştir

--	--	--	--	--	--	--

Birleştir

--	--	--	--	--	--	--

Birleştir

--	--	--	--	--	--	--

Tek öğeli diziler zaten sıralıdır: **taban durumu**.

Birleştirme Sıralaması (Merge Sort)

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Taban

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Birleştir

27	38	3	43	9	82	10
----	----	---	----	---	----	----

Birleştir

--	--	--	--	--	--	--

Birleştir

--	--	--	--	--	--	--

Birleştir: ikililer sıralı birleştir: [27 38] [3 43] [9 82] [10].

Birleştirme Sıralaması (Merge Sort)

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Taban

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Birleştir

27	38	3	43	9	82	10
----	----	---	----	---	----	----

Birleştir

3	27	38	43	9	10	82
---	----	----	----	---	----	----

Birleştir

--	--	--	--	--	--	--

[3 27 38 43] ve [9 10 82]: her birleştirme, iki yarının uzunluk toplamı kadar adım.

Birleştirme Sıralaması (Merge Sort)

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Böl

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Taban

38	27	43	3	9	82	10
----	----	----	---	---	----	----

Birleştir

27	38	3	43	9	82	10
----	----	---	----	---	----	----

Birleştir

3	27	38	43	9	10	82
---	----	----	----	---	----	----

Birleştir

3	9	10	27	38	43	82
---	---	----	----	----	----	----

Son birleştirme: **[3 9 10 27 38 43 82]**. Her düzey $O(n)$, $\lceil \log_2 7 \rceil = 3$ düzey.

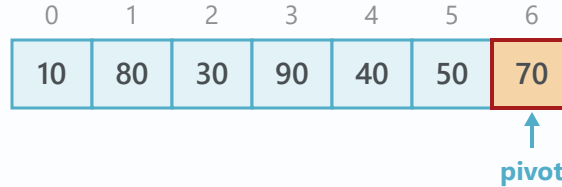
Birleştirme Sıralaması: Kod ve Analiz

```
void birlestirSiralama(int[] dizi, int alt, int ust) {
    if (alt >= ust) return;          // taban: 0-1 eleman
    int orta = (alt + ust) / 2;
    birlestirSiralama(dizi, alt, orta); // sol yarı
    birlestirSiralama(dizi, orta + 1, ust); // sağ yarı
    birlestir(dizi, alt, orta, ust);    // birleştir: O(n)
}

void birlestir(int[] dizi, int alt, int orta, int ust) {
    int[] t = new int[ust - alt + 1];
    int i = alt, j = orta + 1, k = 0;
    while (i <= orta && j <= ust)
        t[k++] = (dizi[i] <= dizi[j]) ? dizi[i++] : dizi[j++];
    while (i <= orta) t[k++] = dizi[i++];
    while (j <= ust) t[k++] = dizi[j++];
    System.arraycopy(t, 0, dizi, alt, t.length);
}
```

- İş **birleştirme** adımında yapılır; bölme $O(1)$.
- $T(n) = 2T(n/2) + \Theta(n) \rightarrow$ durum 2 $\rightarrow \mathbf{\Theta(n \log n)}$; en iyi, ortalama ve en kötü durumda aynı.
- `dizi[i] ≤ dizi[j]` eşitlikte soldakini alır $\rightarrow$ **kararlı** (stable).
- Ek bellek: **$O(n)$** geçici dizi; çağrı yığını $O(\log n)$.
- Ayrıntılı görselleştirme: Bölüm 2.

Hızlı Sıralama (Quick Sort)



Dizi [10 80 30 90 40 50 70]; Lomuto parçalaması: **son öge pivot** (70).

Adım 1 / 7

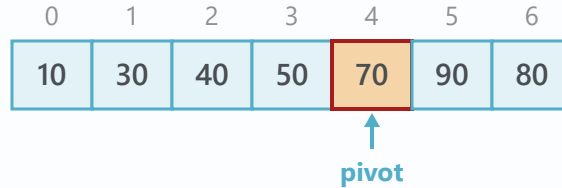
Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öge pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortancası.

Hızlı Sıralama (Quick Sort)



bölümle(dizi, 0, 6), pivot **70** → indis 4'e yerleşti. Sol: [10 30 40 50], sağ: [90 80].

Adım 2 / 7

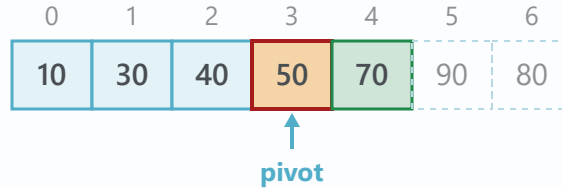
Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortalancası.

Hızlı Sıralama (Quick Sort)



bölümle(dizi, 0, 3), pivot **50** → indis 3'e yerleşti. Sol: [10 30 40], sağ: [—]. **Dengesiz bölme!**

Adım 3 / 7

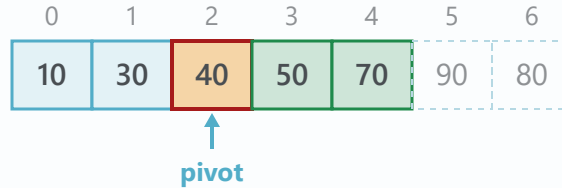
Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortalancası.

Hızlı Sıralama (Quick Sort)



bölümle(dizi, 0, 2), pivot **40** → indis 2'ye yerleşti. Sol: [10 30], sağ: [—]. **Dengesiz bölme!**

Adım 4 / 7

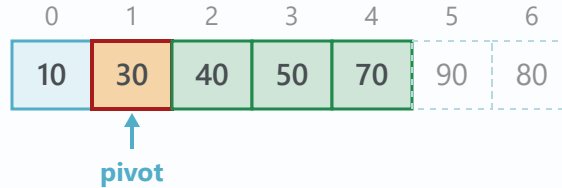
Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortancası.

Hızlı Sıralama (Quick Sort)



bölümle(dizi, 0, 1), pivot **30** → indis 1'e yerleşti. Sol: [10], sağ: [—]. **Dengesiz bölme!**

Adım 5 / 7

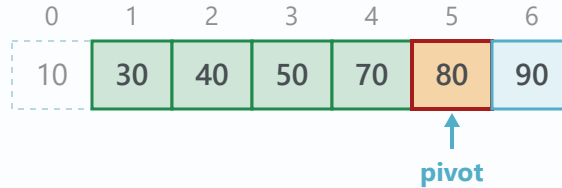
Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortancası.

Hızlı Sıralama (Quick Sort)



bölümle(dizi, 5, 6), pivot **80** → indis 5'e yerleşti. Sol: [—], sağ: [90]. **Dengesiz bölme!**

Adım 6 / 7

Dengeli bölme

Pivot ortalanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortancası.

Hızlı Sıralama (Quick Sort)

0	1	2	3	4	5	6
10	30	40	50	70	80	90

Sıralı: **[10 30 40 50 70 80 90]**. Birleştirme adımı yok; iş parçalamada yapılır.

Adım 7 / 7

Dengeli bölme

Pivot ortanca civarındaysa $T(n) = 2T(n/2) + O(n)$
→ **$O(n \log n)$** (ortalama durum).

Dengesiz bölme

Sıralı girişte son öğe pivot: $T(n) = T(n - 1) + O(n)$
→ **$O(n^2)$** . Çare: rastgele pivot, üçün ortancası.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y; // çift
5   return x * y * y;             // tek
6 }

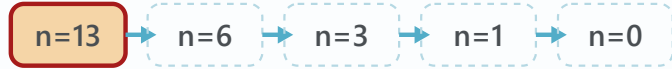
```

ÇAĞRI YIĞINI

us(3, 13)

main()

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım → **$O(\log n)$** :
 $T(n) = T(n/2) + O(1)$.



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 13) çağrıldı → önce **us(3, 6)** hesaplanmalı ($n/2 = 6$).

Adım 1 / 9

DEĞİŞKENLER

x = 3

n = 13

y = ?

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y;   // çift
5   return x * y * y;              // tek
6 }

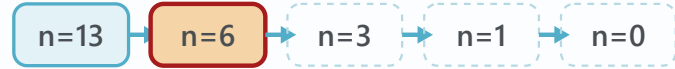
```

ÇAĞRI YIĞINI

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 6) çağrıldı → önce **us(3, 3)** hesaplanmalı (n/2 = 3).

Adım 2 / 9

DEĞİŞKENLER

x = 3

n = 6

y = ?

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım → **$O(\log n)$** :
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y;   // çift
5   return x * y * y;               // tek
6 }

```

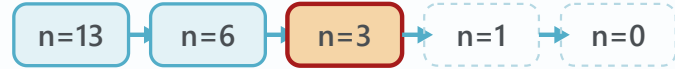
ÇAĞRI YIĞINI

us(3, 3)

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 3) çağrıldı → önce **us(3, 1)** hesaplanmalı ($n/2 = 1$).

Adım 3 / 9

DEĞİŞKENLER

x = 3

n = 3

y = ?

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım → **$O(\log n)$** :
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```
1 long us(long x, int n) {  
2   if (n == 0) return 1;           // taban  
3   long y = us(x, n / 2);  
4   if (n % 2 == 0) return y * y; // çift  
5   return x * y * y;              // tek  
6 }
```

ÇAĞRI YIĞINI

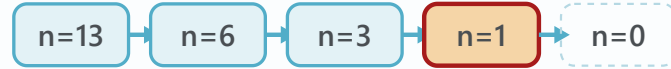
us(3, 1)

us(3, 3)

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 1) çağrıldı → önce **us(3, 0)** hesaplanmalı (n/2 = 0).

Adım 4 / 9

DEĞİŞKENLER

x = 3

n = 1

y = ?

Saf yöntem 12 çarpım; bu yöntem 4
düzeyde en çok 2'şer çarpım → **O(log n)**:
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2     if (n == 0) return 1; // taban
3     long y = us(x, n / 2);
4     if (n % 2 == 0) return y * y; // çift
5     return x * y * y; // tek
6 }

```

ÇAĞRI YIĞINI

us(3, 0)

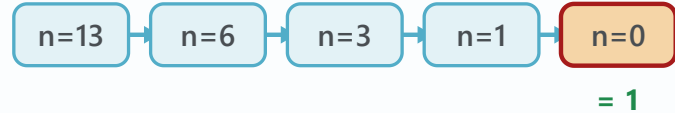
us(3, 1)

us(3, 3)

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

$n = 0 \rightarrow$ **taban durumu:** $3^0 = 1$ döner.

Adım 5 / 9

DEĞİŞKENLER

x = 3

n = 0

y = -

Saf yöntem 12 çarpım; bu yöntem 4
düzeyde en çok 2'şer çarpım $\rightarrow O(\log n)$:
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   ▶ long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y; // çift
5   ▶ return x * y * y;           // tek
6 }

```

ÇAĞRI YIĞINI

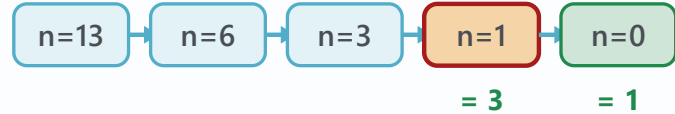
us(3, 1)

us(3, 3)

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 1): $y = 1$; 1 tek $\rightarrow x \cdot y \cdot y = 3 \cdot 1 \cdot 1 = 3$.

Adım 6 / 9

DEĞİŞKENLER

x = 3

n = 1

y = 1

Saf yöntem 12 çarpım; bu yöntem 4
 düzeyde en çok 2'şer çarpım $\rightarrow O(\log n)$:
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   ▶ long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y; // çift
5   ▶ return x * y * y;             // tek
6 }

```

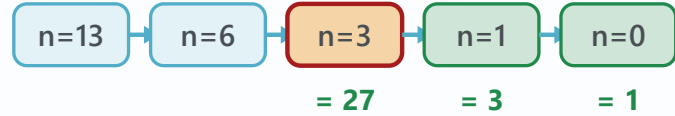
ÇAĞRI YIĞINI

us(3, 3)

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 3): $y = 3$; 3 tek $\rightarrow x \cdot y \cdot y = 3 \cdot 3 \cdot 3 = 27$.

Adım 7 / 9

DEĞİŞKENLER

x = 3

n = 3

y = 3

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım $\rightarrow \mathbf{O(\log n)}$:
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y; // çift
5   return x * y * y;              // tek
6 }

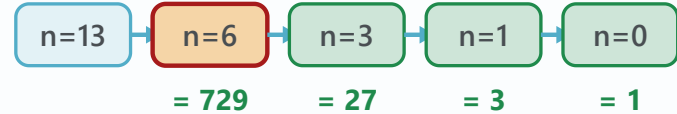
```

ÇAĞRI YIĞINI

us(3, 6)

us(3, 13)

main()



$$3^{13} = 3 \cdot (3^6)^2; 3^6 = (3^3)^2; 3^3 = 3 \cdot (3^1)^2$$

us(3, 6): $y = 27$; 6 çift $\rightarrow y \cdot y = 27 \cdot 27 = 729$.

Adım 8 / 9

DEĞİŞKENLER

x = 3

n = 6

y = 27

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım $\rightarrow O(\log n)$:
 $T(n) = T(n/2) + O(1)$.

Hızlı Üs Alma: 3^{13} İzleme

```

1 long us(long x, int n) {
2   if (n == 0) return 1;           // taban
3   long y = us(x, n / 2);
4   if (n % 2 == 0) return y * y; // çift
5   return x * y * y;              // tek
6 }

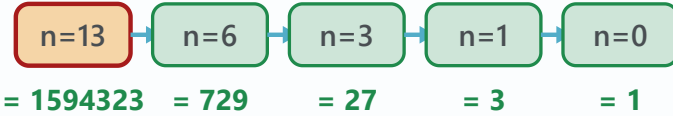
```

ÇAĞRI YIĞINI

us(3, 13)

main()

Saf yöntem 12 çarpım; bu yöntem 4 düzeyde en çok 2'şer çarpım → **$O(\log n)$** :
 $T(n) = T(n/2) + O(1)$.



$$3^{13} = 3 \cdot (3^6)^2; \quad 3^6 = (3^3)^2; \quad 3^3 = 3 \cdot (3^1)^2$$

us(3, 13): $y = 729$; 13 tek → $x \cdot y \cdot y = 3 \cdot 729 \cdot 729 = 1594323$.

Adım 9 / 9

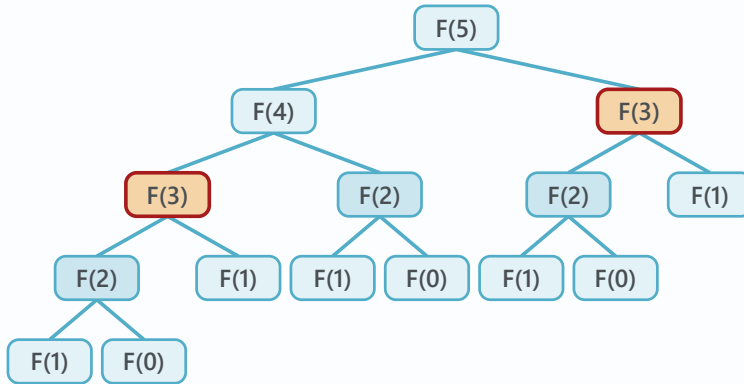
DEĞİŞKENLER

x = 3

n = 13

y = 729

Ne Zaman Uygun Değil? Örtüşen Alt Problemler



```

long fib(int n) {
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}
    
```

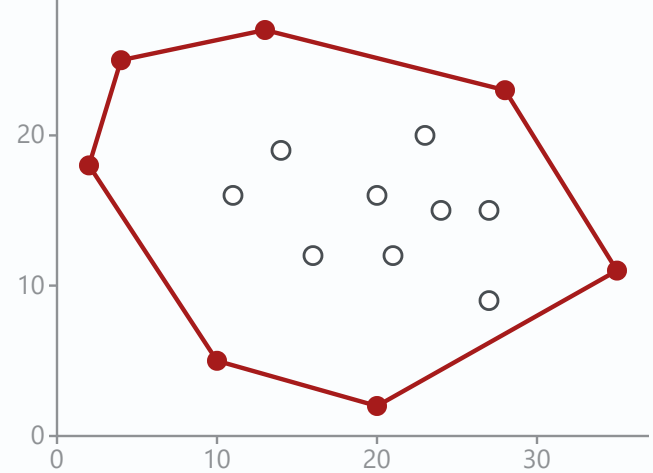
- F(5) için **15 çağrı**: F(3) 2 kez, F(2) 3 kez hesaplanır.
- Alt problemler **bağımsız değil**, örtüşüyor
 $\rightarrow T(n) = T(n-1) + T(n-2) + O(1) = \mathbf{O(\varphi^n)}$
 $\approx O(1,618^n)$.
- Çözüm: sonuçları sakla $\rightarrow$ **dinamik programlama** (Bölüm 6), $O(n)$.

Fibonacci özyinelemesi böl ve fethet biçiminde yazılabilir; sonuç doğrudur, ama bu **kötü** bir örnektir: böl ve fethet, alt problemler birbirinden bağımsız olduğunda verimlidir.

DIŞBÜKEY ÖRTÜ

Dışbükey Örtü (Convex Hull)

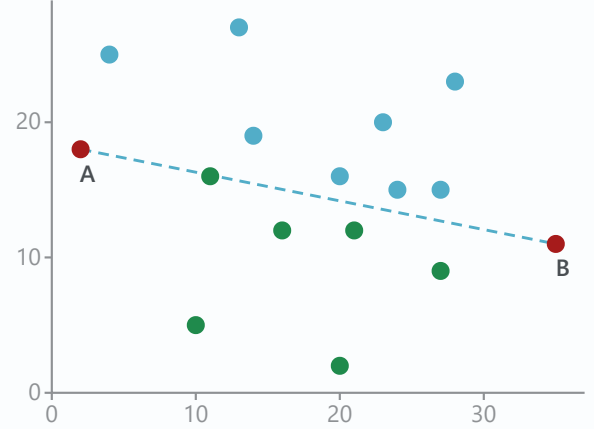
- Düzlemde n noktalı bir küme S verilsin. S 'nin **dışbükey örtüsü**, S 'yi içeren **en küçük dışbükey çokgendir**.
- Sezgi: noktalara çivi çakıp etrafına bir **lastik bant** geçirin; bantın aldığı şekil örtüdür.
- Örtünün köşeleri S 'nin noktalarıdır; çıktı bu köşelerin (saat yönünün tersine) sıralı listesidir.
- Dışbükey: içindeki herhangi iki noktayı birleştiren doğru parçası tamamen çokgenin içindedir.



Kullanım: çarpışma algılama (oyunlar, robotik), coğrafi bilgi sistemleri, görüntü işlemede şekil analizi, en uzak nokta çifti, kümeleme.

Quickhull Algoritması

1. x'i en küçük (A) ve en büyük (B) noktaları bul; ikisi de örtüdedir.
2. AB doğrusu ile noktaları **iki alt kümeye** ayır: doğrunun üstü (solu) ve altı (sağı).
3. Bir alt kümede AB'ye **en uzak** C noktasını bul; C örtüdedir.
4. ACB üçgeninin **içindeki noktaları yok say**: örtüde olamazlar.
5. AC'nin dışında kalanlar ve CB'nin dışında kalanlar için **özyinele**.
6. Hiç nokta kalmayınca dur; bulunan noktalar örtüyü oluşturur.



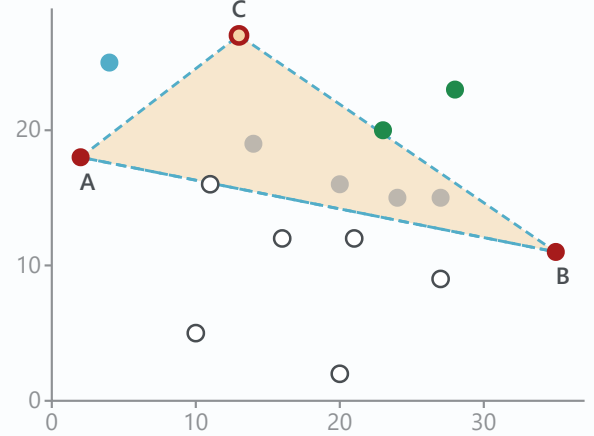
Adım 1–2: en soldaki A ve en sağdaki B örtüdedir; AB doğrusu noktaları **iki kümeye** ayırır (üst / alt).

Adım 1 / 3

Hızlı sıralamaya benzer (adı da buradan): "pivot" en uzak nokta, parçalama üçgenin iki kenarına göre yapılır.

Quickhull Algoritması

1. x'i en küçük (A) ve en büyük (B) noktaları bul; ikisi de örtüdedir.
2. AB doğrusu ile noktaları **iki alt kümeye** ayır: doğrunun üstü (solu) ve altı (sağı).
3. Bir alt kümede AB'ye **en uzak** C noktasını bul; C örtüdedir.
4. ACB üçgeninin **içindeki noktaları yok say**: örtüde olamazlar.
5. AC'nin dışında kalanlar ve CB'nin dışında kalanlar için **özyinele**.
6. Hiç nokta kalmayınca dur; bulunan noktalar örtüyü oluşturur.



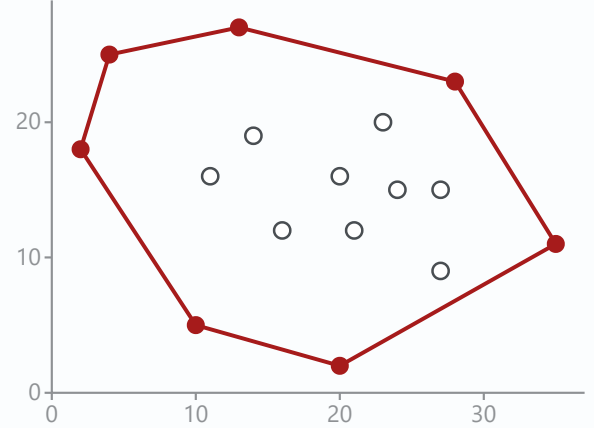
Adım 3–5: üst kümede AB'ye **en uzak** C (13,27) örtüdedir. ACB **üçgeninin içindeki 4 nokta elenir**; AC ve CB dışında kalanlarla özyineleme.

Adım 2 / 3

Hızlı sıralamaya benzer (adı da buradan): "pivot" en uzak nokta, parçalama üçgenin iki kenarına göre yapılır.

Quickhull Algoritması

1. x'i en küçük (A) ve en büyük (B) noktaları bul; ikisi de örtüdedir.
2. AB doğrusu ile noktaları **iki alt kümeye** ayır: doğrunun üstü (solu) ve altı (sağı).
3. Bir alt kümede AB'ye **en uzak** C noktasını bul; C örtüdedir.
4. ACB üçgeninin **içindeki noktaları yok say**: örtüde olamazlar.
5. AC'nin dışında kalanlar ve CB'nin dışında kalanlar için **özyinele**.
6. Hiç nokta kalmayınca dur; bulunan noktalar örtüyü oluşturur.



Adım 6: dışarıda nokta kalmayana dek özyineleme → **7 köşeli örtü**.

Adım 3 / 3

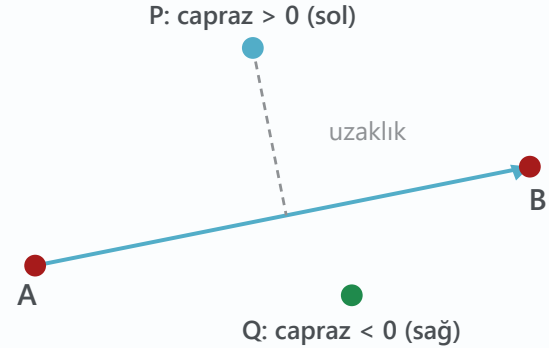
Hızlı sıralamaya benzer (adı da buradan): "pivot" en uzak nokta, parçalama üçgenin iki kenarına göre yapılır.

Geometri: Yönelim Testi ve Uzaklık

Quickhull yalnızca **tam sayı** çarpımıyla çalışır; aç, karekök, bölme gerekmez:

```
long capraz(Nokta A, Nokta B, Nokta P) {  
    return (long)(B.x - A.x) * (P.y - A.y)  
        - (long)(B.y - A.y) * (P.x - A.x);  
}
```

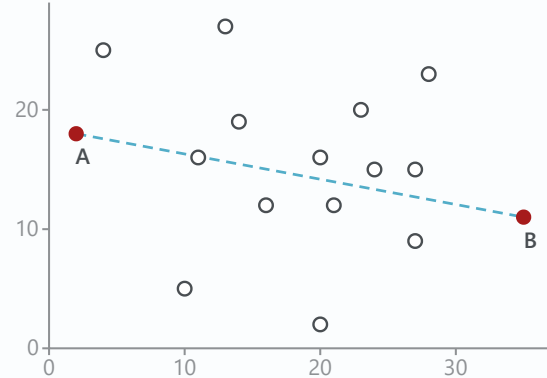
- **capraz > 0**: P, A→B'nin solunda; **< 0**: sağında; **= 0**: aynı doğrudur.
- $|\text{capraz}| = 2 \times (\text{ABP üçgeninin alanı}) = |\text{AB}| \times \text{uzaklık}$. $|\text{AB}|$ sabit olduğundan **en uzak nokta = en büyük capraz**.



Örnek: A = (2,18), B = (35,11). C = (13,27) için $\text{capraz} = 33 \cdot 9 - (-7) \cdot 11 = \mathbf{374}$; (28,23) için $33 \cdot 5 + 7 \cdot 26 = \mathbf{347}$. En büyük değer C'de → C en uzak nokta.

Quickhull: Kod İzleme

```
1 void quickHull(List<Nokta> P) {  
    ▶ Nokta A = enSol(P), B = enSag(P);  
    ▶ ortu.add(A); ortu.add(B);  
    4 bul(solda(P, A, B), A, B); // üst küme  
    5 bul(solda(P, B, A), B, A); // alt küme  
    6 }  
    7 void bul(List<Nokta> S, Nokta A, Nokta B) {  
    8     if (S.isEmpty()) return; // AB örtü kenarı  
    9     Nokta C = enUzak(S, A, B); // max capraz  
    10    ortu.add(C);  
    11    bul(solda(S, A, C), A, C); // AC'nin dışı  
    12    bul(solda(S, C, B), C, B); // CB'nin dışı  
    13 } // ACB üçgeninin içi elenir
```



En sol A = (2,18), en sağ B = (35,11) örtüye eklenir.

ÇAĞRI YIĞINI

quickHull(P)

Adım 1 / 15

DEĞİŞKENLER

A = (2,18)

B = (35,11)

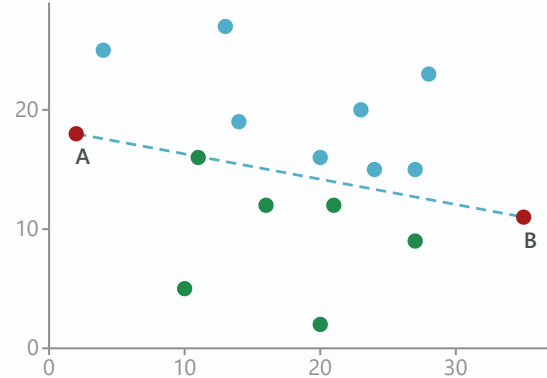
|ortu| = 2

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   ► bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



AB'nin solunda (üst) **8**, sağında (alt) **6** nokta. Önce üst küme.

ÇAĞRI YIĞINI

bul(8 nk, (2,18)→(35,11))

quickHull(P)

Adım 2 / 15

DEĞİŞKENLER

A = (2,18)

B = (35,11)

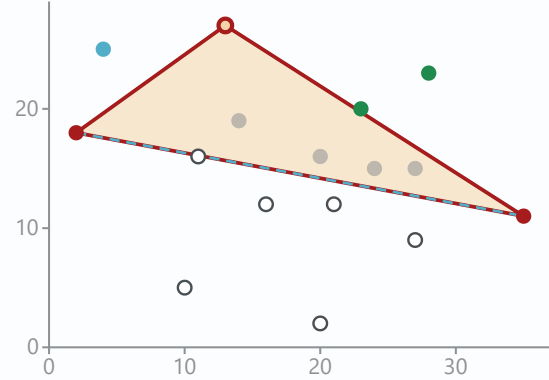
|ortu| = 2

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   ▶ Nokta C = enUzak(S, A, B); // max capraz
10  ▶ ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



$|S| = 8$: en uzak **C = (13,27)** örtüye girer. Üçgen içinde 4 nokta elenir; AC dışında 1, CB dışında 2 nokta.

ÇAĞRI YIĞINI

```
bul(8 nk, (2,18)→(35,11))
```

```
quickHull(P)
```

Adım 3 / 15

DEĞİŞKENLER

A = (2,18)

B = (35,11)

C = (13,27)

$|S| = 8$

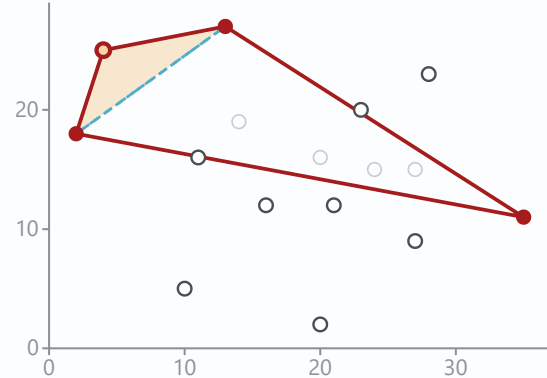
$|ortu| = 3$

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   ▶ Nokta C = enUzak(S, A, B); // max capraz
10  ▶ ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



$|S| = 1$: en uzak **C = (4,25)** örtüye girer. Üçgen içinde 0 nokta elenir; AC dışında 0, CB dışında 0 nokta.

ÇAĞRI YIĞINI

```
bul(1 nk, (2,18)→(13,27))
```

```
bul(8 nk, (2,18)→(35,11))
```

```
quickHull(P)
```

Adım 4 / 15

DEĞİŞKENLER

A = (2,18)

B = (13,27)

C = (4,25)

$|S| = 1$

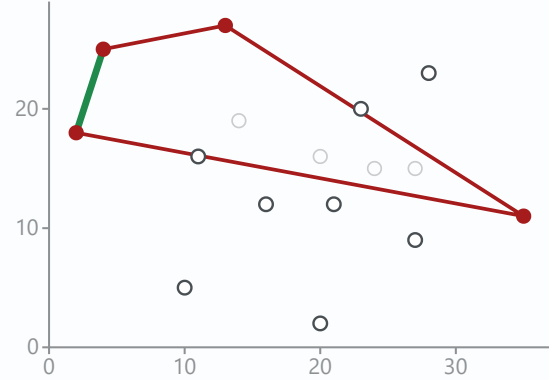
$|ortu| = 4$

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş $\rightarrow$ (2,18)–(4,25) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (2,18) $\rightarrow$ (4,25))

bul(1 nk, (2,18) $\rightarrow$ (13,27))

bul(8 nk, (2,18) $\rightarrow$ (35,11))

quickHull(P)

Adım 5 / 15

DEĞİŞKENLER

A = (2,18)

B = (4,25)

C = -

|S| = 0

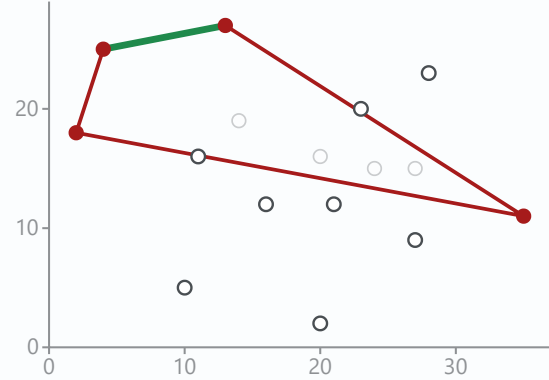
|ortu| = 4

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş $\rightarrow$ (4,25)–(13,27) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (4,25)→(13,27))

bul(1 nk, (2,18)→(13,27))

bul(8 nk, (2,18)→(35,11))

quickHull(P)

Adım 6 / 15

DEĞİŞKENLER

A = (4, 25)

B = (13, 27)

C = -

|S| = 0

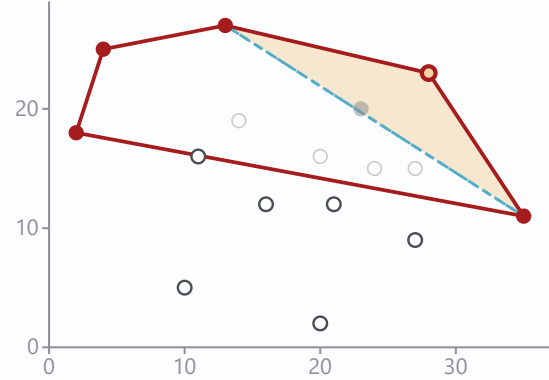
|ortu| = 4

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   ▶ Nokta C = enUzak(S, A, B); // max capraz
10  ▶ ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



$|S| = 2$: en uzak **C = (28,23)** örtüye girer. Üçgen içinde 1 nokta elenir; AC dışında 0, CB dışında 0 nokta.

ÇAĞRI YIĞINI

```
bul(2 nk, (13,27)→(35,11))
```

```
bul(8 nk, (2,18)→(35,11))
```

```
quickHull(P)
```

Adım 7 / 15

DEĞİŞKENLER

A = (13,27)

B = (35,11)

C = (28,23)

$|S| = 2$

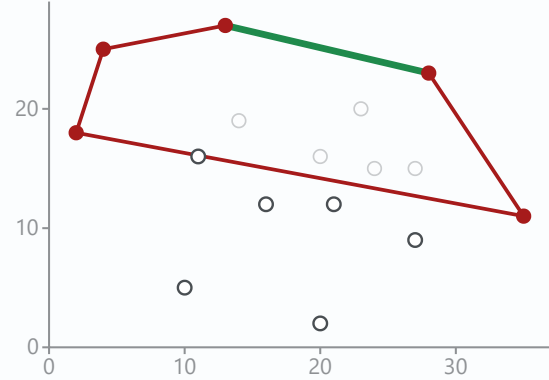
$|ortu| = 5$

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş → (13,27)–(28,23) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (13,27)→(28,23))

bul(2 nk, (13,27)→(35,11))

bul(8 nk, (2,18)→(35,11))

quickHull(P)

Adım 8 / 15

DEĞİŞKENLER

A = (13,27)

B = (28,23)

C = -

|S| = 0

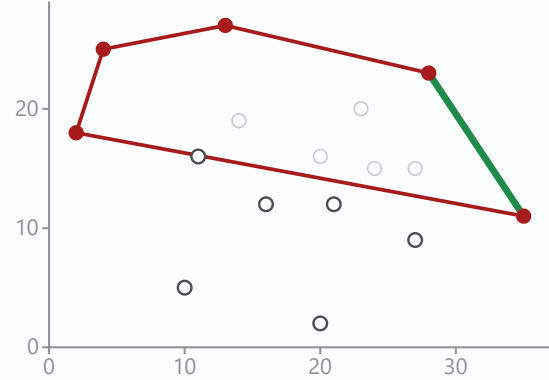
|ortu| = 5

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş → (28,23)–(35,11) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (28,23)→(35,11))

bul(2 nk, (13,27)→(35,11))

bul(8 nk, (2,18)→(35,11))

quickHull(P)

Adım 9 / 15

DEĞİŞKENLER

A = (28,23)

B = (35,11)

C = -

|S| = 0

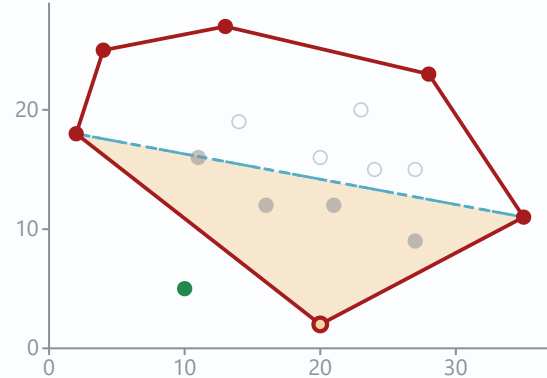
|ortu| = 5

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   ▶ Nokta C = enUzak(S, A, B); // max capraz
10  ▶ ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



$|S| = 6$: en uzak **C = (20,2)** örtüye girer. Üçgen içinde 4 nokta elenir; AC dışında 0, CB dışında 1 nokta.

ÇAĞRI YIĞINI

```
bul(6 nk, (35,11)→(2,18))
```

```
quickHull(P)
```

Adım 10 / 15

DEĞİŞKENLER

A = (35,11)

B = (2,18)

C = (20,2)

$|S| = 6$

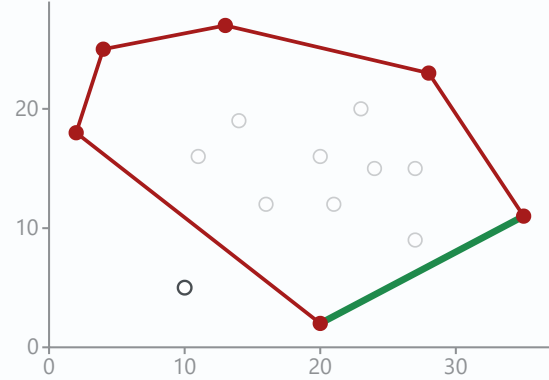
$|ortu| = 6$

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş → (35,11)–(20,2) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (35,11)→(20,2))

bul(6 nk, (35,11)→(2,18))

quickHull(P)

Adım 11 / 15

DEĞİŞKENLER

A = (35,11)

B = (20,2)

C = -

|S| = 0

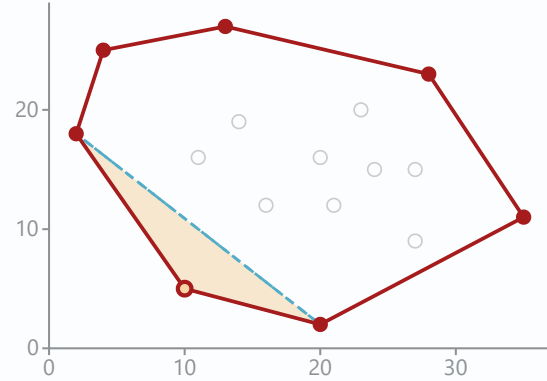
|ortu| = 6

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   ▶ Nokta C = enUzak(S, A, B); // max capraz
10  ▶ ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



$|S| = 1$: en uzak **C = (10,5)** örtüye girer. Üçgen içinde 0 nokta elenir; AC dışında 0, CB dışında 0 nokta.

ÇAĞRI YIĞINI

```
bul(1 nk, (20,2)→(2,18))
```

```
bul(6 nk, (35,11)→(2,18))
```

```
quickHull(P)
```

Adım 12 / 15

DEĞİŞKENLER

A = (20,2)

B = (2,18)

C = (10,5)

|S| = 1

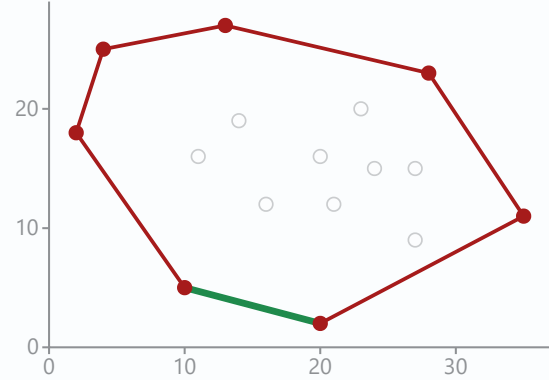
|ortu| = 7

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş → (20,2)–(10,5) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (20,2)→(10,5))

bul(1 nk, (20,2)→(2,18))

bul(6 nk, (35,11)→(2,18))

quickHull(P)

Adım 13 / 15

DEĞİŞKENLER

A = (20,2)

B = (10,5)

C = -

|S| = 0

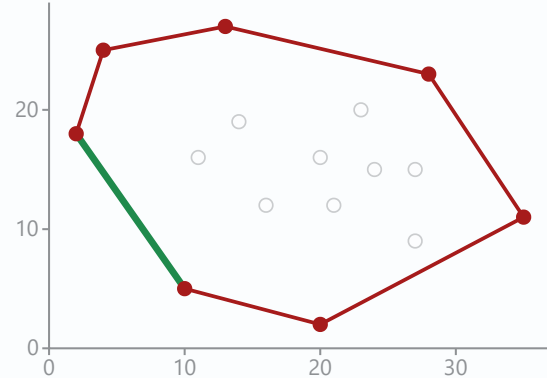
|ortu| = 7

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   ► if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



S boş → (10,5)–(2,18) bir **örtü kenarıdır**; geri dön.

ÇAĞRI YIĞINI

bul(0 nk, (10,5)→(2,18))

bul(1 nk, (20,2)→(2,18))

bul(6 nk, (35,11)→(2,18))

quickHull(P)

Adım 14 / 15

DEĞİŞKENLER

A = (10,5)

B = (2,18)

C = -

|S| = 0

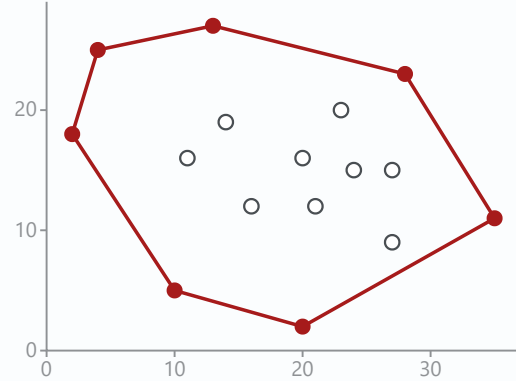
|ortu| = 7

Quickhull: Kod İzleme

```

1 void quickHull(List<Nokta> P) {
2   Nokta A = enSol(P), B = enSag(P);
3   ortu.add(A); ortu.add(B);
4   bul(solda(P, A, B), A, B); // üst küme
5   bul(solda(P, B, A), B, A); // alt küme
6 }
7 void bul(List<Nokta> S, Nokta A, Nokta B) {
8   if (S.isEmpty()) return; // AB örtü kenarı
9   Nokta C = enUzak(S, A, B); // max capraz
10  ortu.add(C);
11  bul(solda(S, A, C), A, C); // AC'nin dışı
12  bul(solda(S, C, B), C, B); // CB'nin dışı
13 } // ACB üçgeninin içi elenir

```



Bitti: **7 köşe** bulundu, 9 nokta elendi. Toplam 12 **bul** çağırısı.

ÇAĞRI YIĞINI

quickHull(P)

Adım 15 / 15

DEĞİŞKENLER

A = (2,18)

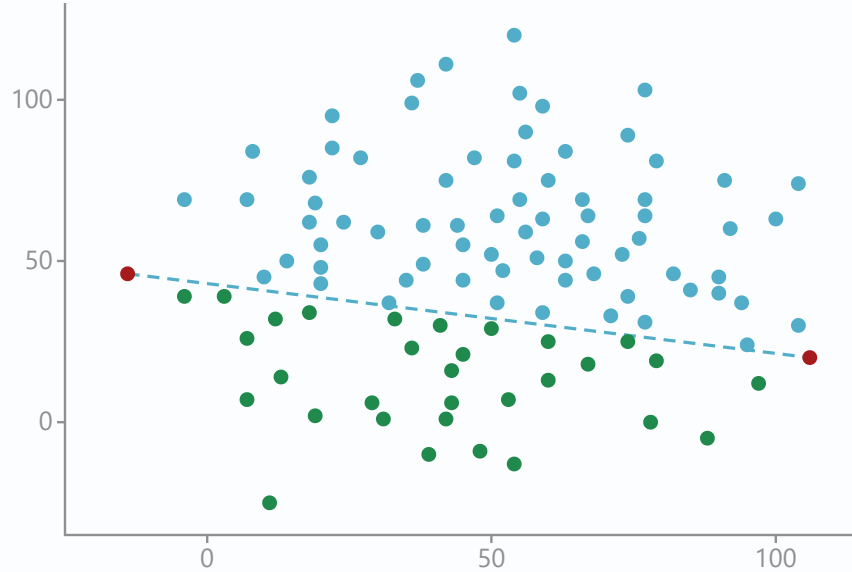
B = (35,11)

C = -

|S| = 0

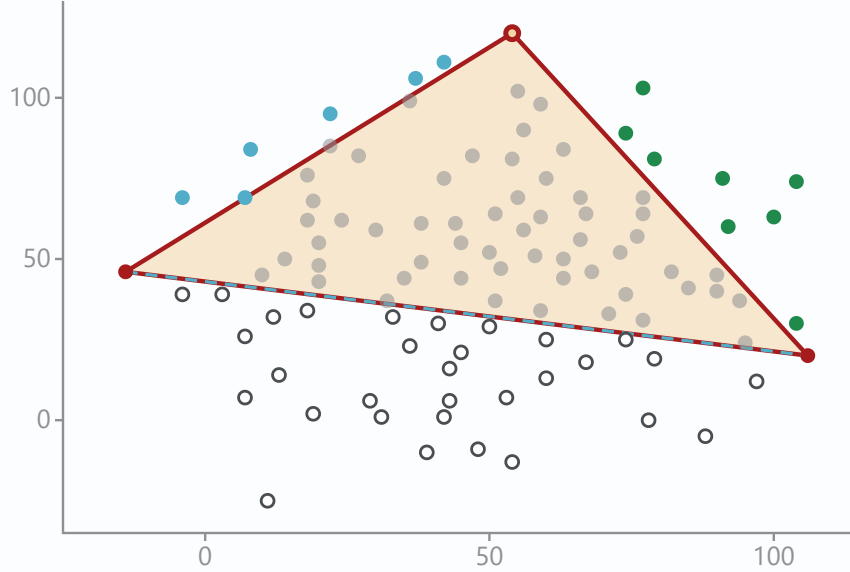
|ortu| = 7

Quickhull: 104 Noktalı Örnek



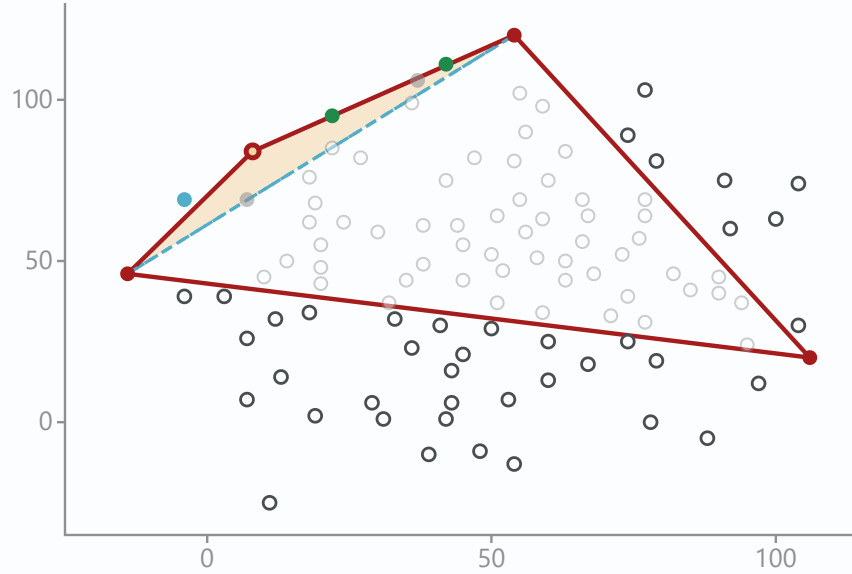
104 nokta. En sol (-14,46) ve en sağ (106,20) birleştirilir; üstte 71, altta 31 nokta.

Quickhull: 104 Noktalı Örnek



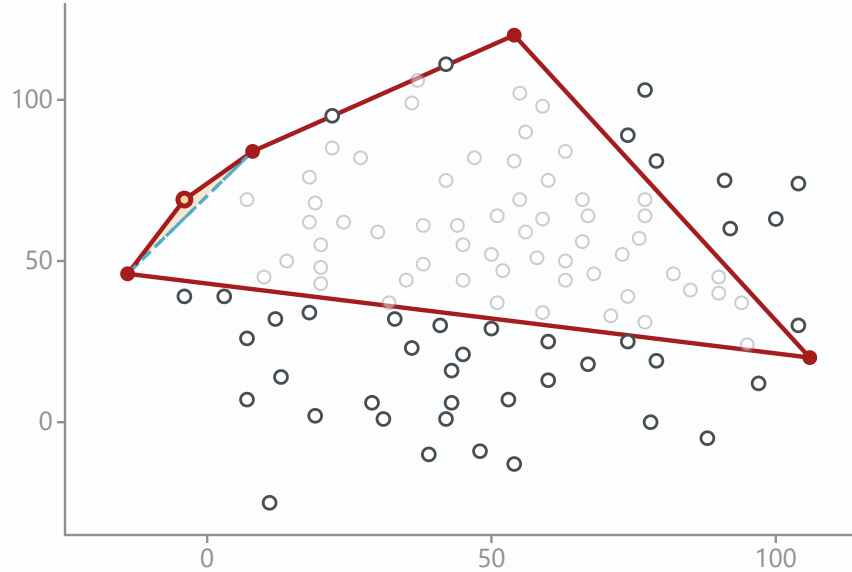
$(-14,46) \rightarrow (106,20)$ için en uzak: **(54,120)**; üçgen içindeki **56** nokta elendi. Aday kaldı: 45.

Quickhull: 104 Noktalı Örnek



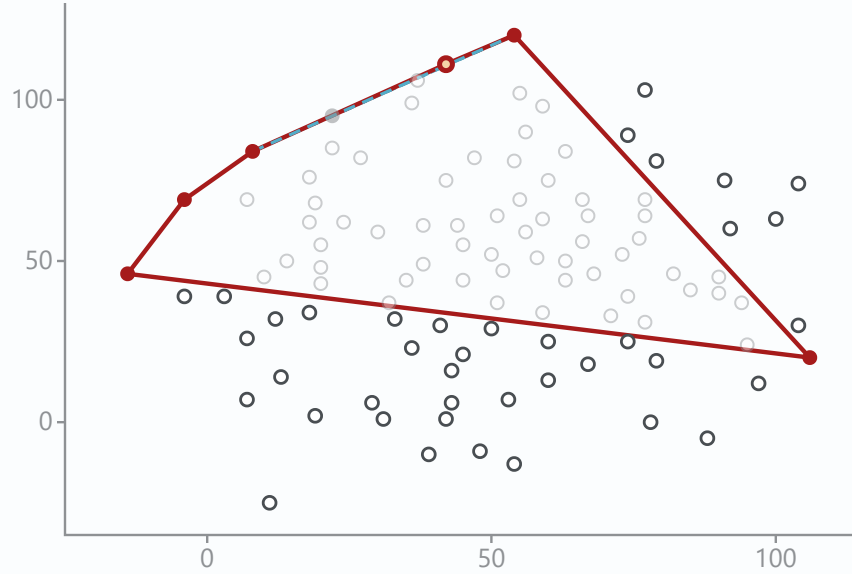
$(-14,46) \rightarrow (54,120)$ için en uzak: **(8,84)**; üçgen içindeki **2** nokta elendi. Aday kaldı: 42.

Quickhull: 104 Noktalı Örnek



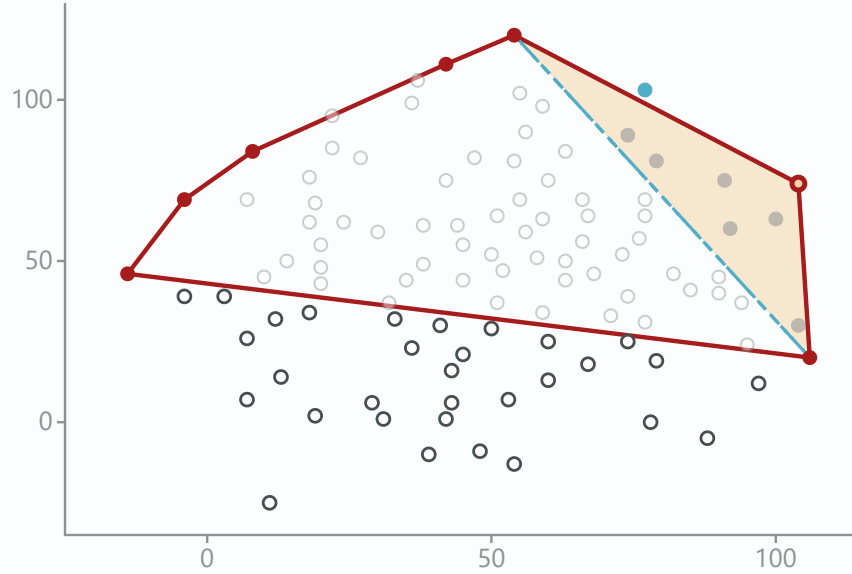
$(-14,46) \rightarrow (8,84)$ için en uzak: **$(-4,69)$** ; üçgen içindeki **0** nokta elendi. Aday kaldı: 41.

Quickhull: 104 Noktalı Örnek



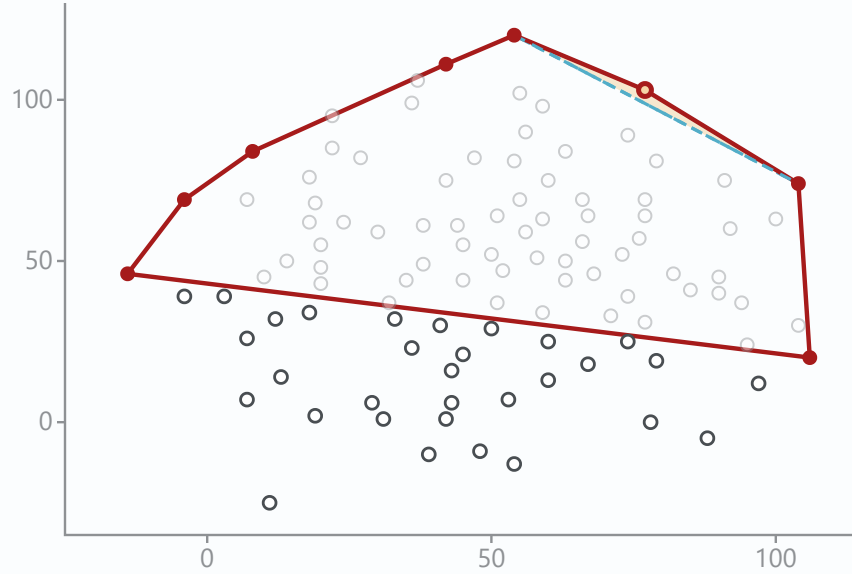
$(8,84) \rightarrow (54,120)$ için en uzak: **(42,111)**; üçgen içindeki **1** nokta elendi. Aday kaldı: 39.

Quickhull: 104 Noktalı Örnek



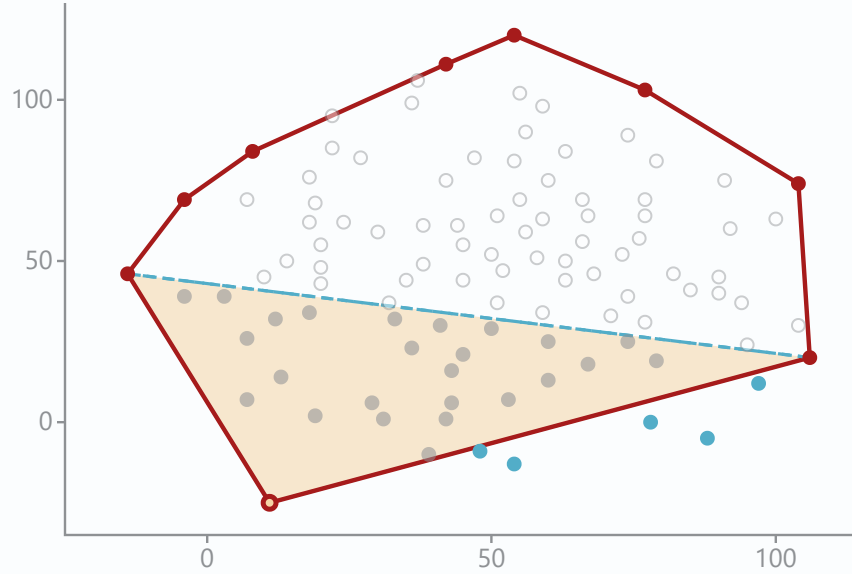
$(54,120) \rightarrow (106,20)$ için en uzak: **(104,74)**; üçgen içindeki **6** nokta elendi. Aday kaldı: 32.

Quickhull: 104 Noktalı Örnek



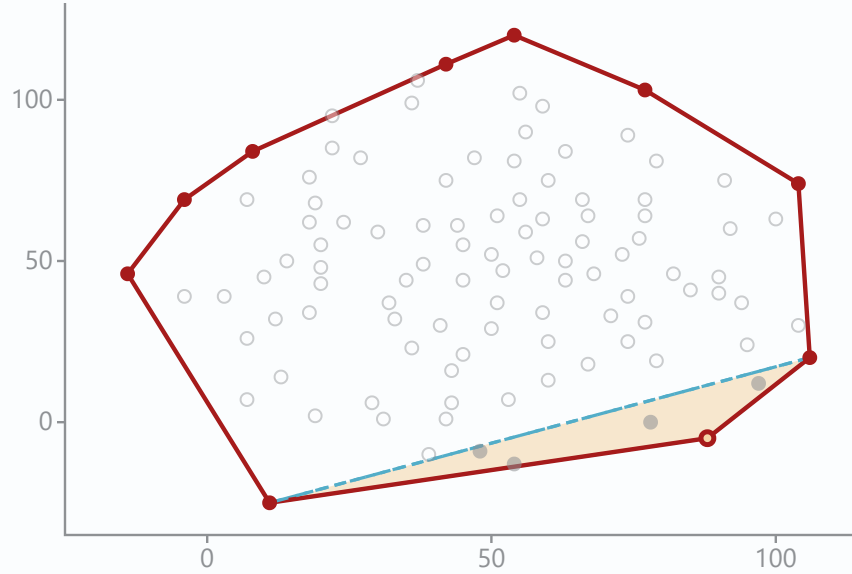
$(54,120) \rightarrow (104,74)$ için en uzak: **(77,103)**; üçgen içindeki **0** nokta elendi. Aday kaldı: 31.

Quickhull: 104 Noktalı Örnek



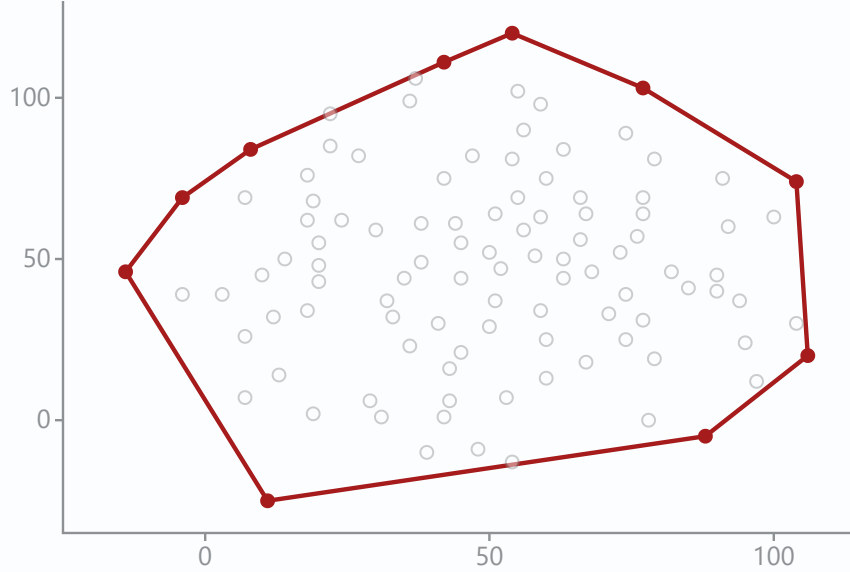
$(106,20) \rightarrow (-14,46)$ için en uzak: **(11,-25)**; üçgen içindeki **25** nokta elendi. Aday kaldı: 5.

Quickhull: 104 Noktalı Örnek



(106,20)→(11,-25) için en uzak: **(88,-5)**; üçgen içindeki **4** nokta elendi. Aday kaldı: 0.

Quickhull: 104 Noktalı Örnek



Bitti: 104 noktanın örtüsü **10 köşe**. Her adımda noktaların çoğu elendiği için iş hızla azalır.

Quickhull Analizi ve Diğer Örtü Algoritmaları

- Her çağrı S'yi bir kez tarar (en uzak nokta + iki parçalama): $O(|S|)$.
- $T(n) = T(n_1) + T(n_2) + O(n)$, $n_1 + n_2 \leq n - 1$.
- **Ortalama** (dengeli bölmeler, çok nokta elenir): $O(n \log n)$; rastgele düzgün dağılımda çoğu zaman daha da hızlı.
- **En kötü** durum: her adımda yalnızca bir nokta ayrılır $\rightarrow T(n) = T(n - 1) + O(n) = O(n^2)$.
- Çıktıya duyarlı sınır: h köşeli örtü için $O(n \cdot h)$.

Algoritma	Süre
Kaba kuvvet (her kenar adayı)	$O(n^3)$
Jarvis yürüyüşü (gift wrapping)	$O(n \cdot h)$
Graham taraması	$O(n \log n)$
Andrew monoton zincir	$O(n \log n)$
Quickhull	ort. $O(n \log n)$, en kötü $O(n^2)$
Chan	$O(n \log h)$

Alt sınır: örtü sıralamayı da çözebildiği için (noktaları $y = x^2$ parabolüne yerleştirin) karşılaştırmalı modelde $\Omega(n \log n)$ zaman gerekir.

En Yakın Nokta Çifti (Closest Pair)

Düzlemde n nokta verilsin; aralarındaki **Öklid uzaklığı** $d(p, q) = \sqrt{((p_x - q_x)^2 + (p_y - q_y)^2)}$ en küçük olan çifti bulun.

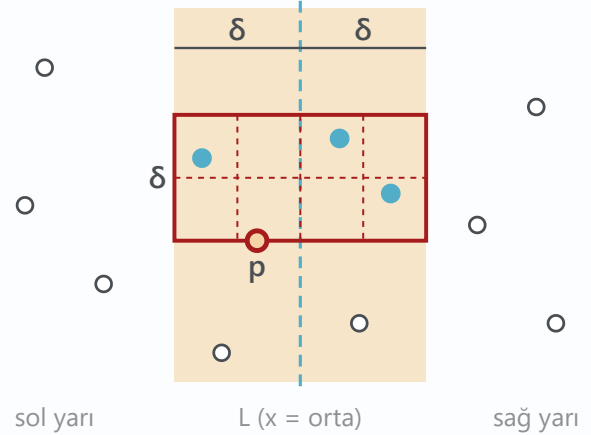
```
double kabaKuvvet(List<Nokta> p) {  
    double enAz = Double.MAX_VALUE;  
    for (int i = 0; i < p.size(); i++)  
        for (int j = i + 1; j < p.size(); j++)  
            enAz = Math.min(enAz, mesafe(p.get(i), p.get(j)));  
    return enAz;  
}
```

- **Kaba kuvvet:** $n(n - 1)/2$ çift $\rightarrow O(n^2)$.
 $n = 10^6$ için $\approx 5 \cdot 10^{11}$ uzaklık.
- **Tek boyutta** kolay: sırala, yalnızca komşuları karşılaştır $\rightarrow O(n \log n)$.
- İki boyutta x 'e göre sıralı komşular yetmez; **böl ve fethet** ile $O(n \log n)$ mümkün.

Uygulamalar: hava trafiği kontrolünde en yakın iki uçak, kümeleme, çarpışma algılama, coğrafi verilerde yakın tesisler.

Böl ve Fethet: Orta Şerit

1. Noktaları bir kez **x'e** ve bir kez **y'ye** göre sırala.
2. **Böl:** ortadaki noktanın x'inden geçen L doğrusu ile iki yarı.
3. **Yönet:** iki yarıda özyinelemeyle δ_L ve δ_R ; $\delta = \min(\delta_L, \delta_R)$.
4. **Birleştir:** daha yakın bir çift varsa L'yi keser $\rightarrow$ yalnızca $|x - L| < \delta$ olan **şerit** noktalarına bak.
5. Şeridi y sırasıyla tara; her nokta yalnızca y farkı δ 'dan küçük olan sonrakilerle karşılaştırılır.



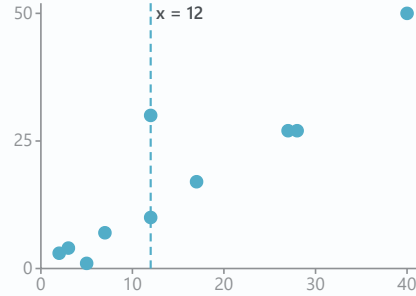
Neden en çok 7? p'nin üstündeki $\delta \times 2\delta$ dikdörtgen, kenarı $\delta/2$ olan 8 kareye bölünür. Bir karedeki iki nokta aynı yarıda olur ve aralarındaki uzaklık $< \delta$ olurdu; bu imkânsız $\rightarrow$ her karede ≤ 1 nokta $\rightarrow$ p en çok **7** noktayla karşılaştırılır.

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   ▶ int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   ▶ Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size()
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 10 > 3 \rightarrow \text{orta} = 5, m = (12, 10)$. Sol: 5, sağ: 5 nokta.

Adım 1 / 13

ÇAĞRI YIĞINI

enYakin(X[0..9])

main()

DEĞİŞKENLER

n = 10

m = (12, 10)

dSol = ?

dSag = ?

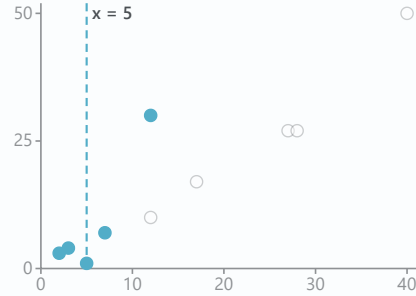
d = ?

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size()
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 5 > 3 \rightarrow \text{orta} = 2, m = (5, 1)$. Sol: 2, sağ: 3 nokta.

Adım 2 / 13

DEĞİŞKENLER

$n = 5$

$m = (5, 1)$

$d_{\text{Sol}} = ?$

$d_{\text{Sag}} = ?$

$d = ?$

ÇAĞRI YIĞINI

enYakin(X[0..4])

enYakin(X[0..9])

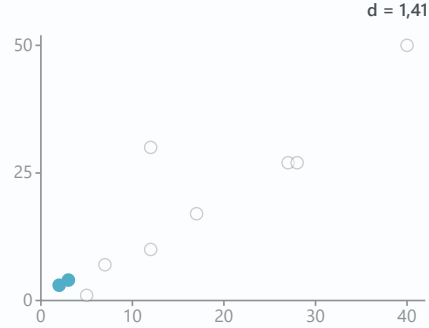
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14          && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 2 \leq 3 \rightarrow$ kaba kuvvet: (2,3), (3,4). En yakın (2,3)–(3,4), $d = 1,41$.

Adım 3 / 13

DEĞİŞKENLER

$n = 2$ $m = -$ $dSol = -$ $dSag = -$ $d = 1,41$

ÇAĞRI YIĞINI

enYakin(X[0..1])

enYakin(X[0..4])

enYakin(X[0..9])

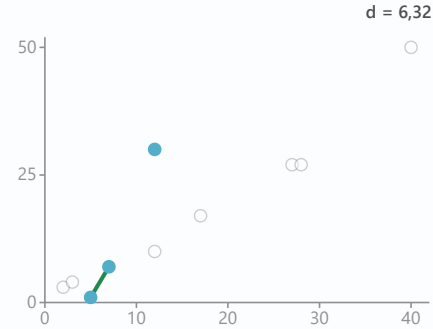
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size(); j++)
14       && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 3 \leq 3 \rightarrow$ kaba kuvvet: (5,1), (7,7), (12,30). En yakın (5,1)–(7,7), $d = 6,32$.

Adım 4 / 13

DEĞİŞKENLER

$n = 3$ $m = -$ $dSol = -$ $dSag = -$ $d = 6,32$

ÇAĞRI YIĞINI

enYakin(X[2..4])

enYakin(X[0..4])

enYakin(X[0..9])

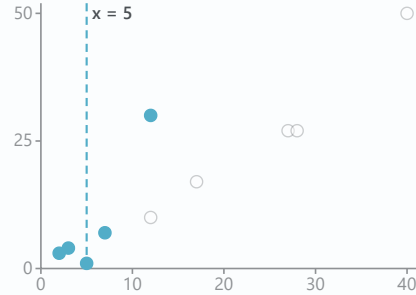
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   ▶ double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size()
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



İki yarı döndü: dSol = 1,41, dSag = 6,32 → d = **1,41**.

Adım 5 / 13

ÇAĞRI YIĞINI

enYakin(X[0..4])

enYakin(X[0..9])

main()

DEĞİŞKENLER

n = 5

m = (5,1)

dSol = 1,41

dSag = 6,32

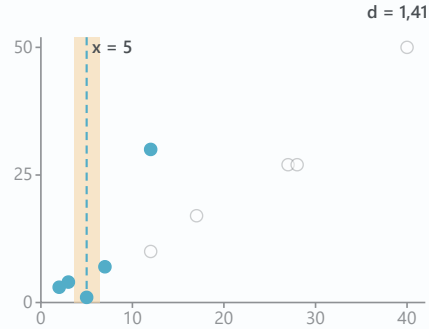
d = 1,41

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y)                // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



Şerit $|x - 5| < 1,41$: [(5,1)]; tek nokta → karşılaştırma yok.
Dönüş: **1,41**.

Adım 6 / 13

DEĞİŞKENLER

$n = 5$ $m = (5,1)$ $dSol = 1,41$ $dSag = 6,32$
 $d = 1,41$ $|serit| = 1$

ÇAĞRI YIĞINI

enYakin(X[0..4])

enYakin(X[0..9])

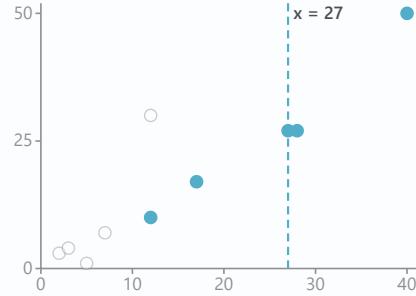
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   ▶ int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   ▶ Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 5 > 3 \rightarrow \text{orta} = 2, m = (27, 27)$. Sol: 2, sağ: 3 nokta.

Adım 7 / 13

DEĞİŞKENLER

$n = 5$

$m = (27, 27)$

$dSol = ?$

$dSag = ?$

$d = ?$

ÇAĞRI YIĞINI

enYakin(X[5..9])

enYakin(X[0..9])

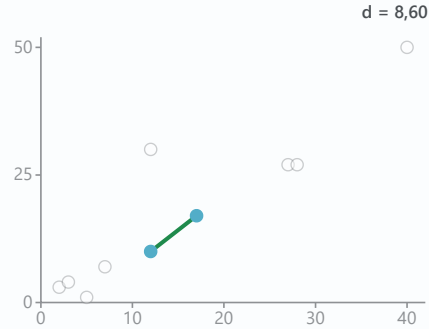
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size()
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 2 \leq 3 \rightarrow$ kaba kuvvet: (12,10), (17,17). En yakın (12,10)–(17,17), $d = 8,60$.

Adım 8 / 13

DEĞİŞKENLER

$n = 2$ $m = -$ $dSol = -$ $dSag = -$ $d = 8,60$

ÇAĞRI YIĞINI

enYakin(X[5..6])

enYakin(X[5..9])

enYakin(X[0..9])

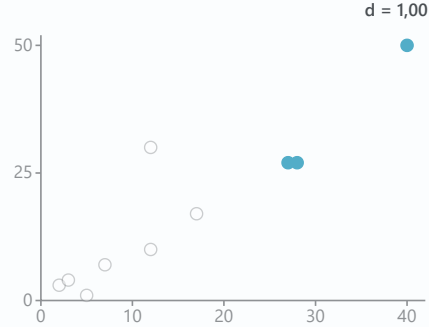
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size()
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



$n = 3 \leq 3 \rightarrow$ kaba kuvvet: (27,27), (28,27), (40,50). En yakın (27,27)–(28,27), $d = 1,00$.

Adım 9 / 13

DEĞİŞKENLER

$n = 3$

$m = -$

$dSol = -$

$dSag = -$

$d = 1,00$

ÇAĞRI YIĞINI

enYakin(X[7..9])

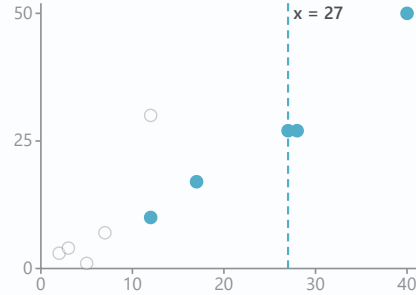
enYakin(X[5..9])

enYakin(X[0..9])

main()

En Yakın Çift: Kod İzleme

```
1 double enYakin(List<Nokta> X, List<Nokta> Y) {  
2   int n = X.size(), orta = n / 2;  
3   if (n <= 3) return kabaKuvvet(X);  
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari  
5   double dSol = enYakin(X.subList(0, orta), Ysol);  
6   double dSag = enYakin(X.subList(orta, n), Ysag);  
7   ▶ double d = Math.min(dSol, dSag);  
8   List<Nokta> s = new ArrayList<>(); // şerit  
9   for (Nokta p : Y) // y sırası  
10    if (Math.abs(p.x - m.x) < d) s.add(p);  
11   for (int i = 0; i < s.size(); i++) {  
12     Nokta p = s.get(i);  
13     for (int j = i + 1; j < s.size();  
14         && s.get(j).y - p.y < d; j++)  
15       d = Math.min(d, mesafe(p, s.get(j)));  
16   }  
17   return d;  
18 }
```



İki yarı döndü: dSol = 8,60, dSag = 1,00 → d = **1,00**.

Adım 10 / 13

ÇAĞRI YIĞINI

enYakin(X[5..9])

enYakin(X[0..9])

main()

DEĞİŞKENLER

n = 5

m = (27,27)

dSol = 8,60

dSag = 1,00

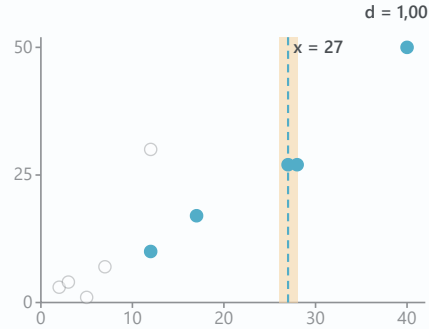
d = 1,00

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarılari
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14          && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



Şerit $|x - 27| < 1,00$: [(27,27)]; tek nokta → karşılaştırma yok. Dönüş: **1,00**.

Adım 11 / 13

DEĞİŞKENLER

n = 5 m = (27,27) dSol = 8,60 dSag = 1,00
 d = 1,00 |serit| = 1

ÇAĞRI YIĞINI

enYakin(X[5..9])

enYakin(X[0..9])

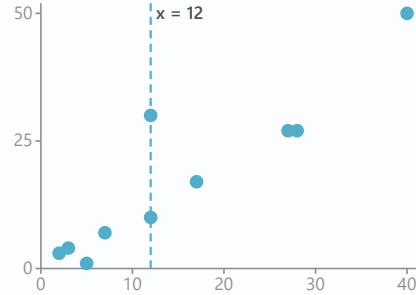
main()

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   ▶ double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14           && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



İki yarı döndü: dSol = 1,41, dSag = 1,00 → d = **1,00**.

Adım 12 / 13

ÇAĞRI YIĞINI

enYakin(X[0..9])

main()

DEĞİŞKENLER

n = 10 m = (12,10) dSol = 1,41

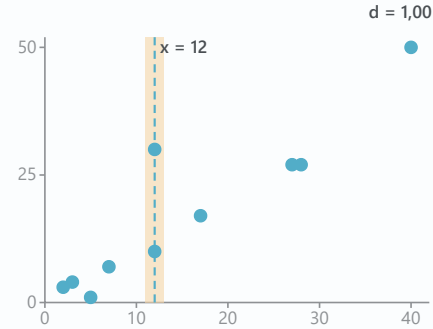
dSag = 1,00 d = 1,00

En Yakın Çift: Kod İzleme

```

1 double enYakin(List<Nokta> X, List<Nokta> Y) {
2   int n = X.size(), orta = n / 2;
3   if (n <= 3) return kabaKuvvet(X);
4   Nokta m = X.get(orta); // Ysol/Ysag: Y'nin yarıları
5   double dSol = enYakin(X.subList(0, orta), Ysol);
6   double dSag = enYakin(X.subList(orta, n), Ysag);
7   double d = Math.min(dSol, dSag);
8   List<Nokta> s = new ArrayList<>(); // şerit
9   for (Nokta p : Y) // y sırası
10    if (Math.abs(p.x - m.x) < d) s.add(p);
11   for (int i = 0; i < s.size(); i++) {
12     Nokta p = s.get(i);
13     for (int j = i + 1; j < s.size();
14          && s.get(j).y - p.y < d; j++)
15       d = Math.min(d, mesafe(p, s.get(j)));
16   }
17   return d;
18 }

```



Şerit $|x - 12| < 1,00$: [(12,10), (12,30)]; y farkı d'den büyük
→ **hiç karşılaştırma yok**. Dönüş: **1,00**.

Adım 13 / 13

DEĞİŞKENLER

n = 10 m = (12,10) dSol = 1,41

dSag = 1,00 d = 1,00 |serit| = 2

ÇAĞRI YIĞINI

enYakin(X[0..9])

main()

EN YAKIN NOKTA ÇİFTİ

En Yakın Çift: Analiz

Adım	Maliyet	Not
x ve y'ye göre ön sıralama	$O(n \log n)$	yalnızca bir kez, özyinelemeden önce
Bölme (X ikiye, Y'yi sırayı koruyarak Ysol/Ysag'ye)	$O(n)$	yeniden sıralama yok
Şeridi kurma	$O(n)$	Y zaten y'ye göre sıralı
Şeridi tarama	$O(7n) = O(n)$	her nokta ≤ 7 komşu
Toplam	$O(n \log n)$	$T(n) = 2T(n/2) + O(n)$

Yaygın tuzak

Şeridi her çağrıda yeniden **sıralamak**: $T(n) = 2T(n/2) + O(n \log n) = \mathbf{O(n \log^2 n)}$.

Örnekte

10 noktanın en yakın çifti **(27,27)–(28,27)**, $d = 1,00$. Kaba kuvvet 45 uzaklık hesaplardı.

Karatsuba: 4 Çarpım Yerine 3

n basamaklı x, y sayılarını $m = \lfloor n/2 \rfloor$ noktasından böl: $x = a \cdot 10^m + b$, $y = c \cdot 10^m + d$.

Saf böl ve fethet

$$x \cdot y = ac \cdot 10^{2m} + (ad + bc) \cdot 10^m + bd$$

4 çarpım: ac, ad, bc, bd

$$T(n) = 4T(n/2) + O(n) = \mathbf{O(n^2)} \rightarrow \text{kazanç yok!}$$

Karatsuba (1962)

$$(a + b)(c + d) = ac + (\mathbf{ad + bc}) + bd$$

$$\Rightarrow ad + bc = (a + b)(c + d) - ac - bd$$

3 çarpım: $z2 = ac, z0 = bd, z1 = (a + b)(c + d)$

$$T(n) = 3T(n/2) + O(n) = \mathbf{O(n^{1.585})}$$

Örnek: $1234 \cdot 5678$ ($m = 2, 10^m = 100$)

$$a = 12, b = 34, c = 56, d = 78$$

$$z2 = 12 \cdot 56 = \mathbf{672} \cdot z0 = 34 \cdot 78 = \mathbf{2652}$$

$$z1 = (12 + 34)(56 + 78) = 46 \cdot 134 = \mathbf{6164}$$

$$z1 - z2 - z0 = 6164 - 672 - 2652 = \mathbf{2840} (= ad + bc)$$

$$672 \cdot 10^4 + 2840 \cdot 10^2 + 2652 = 6\,720\,000 + 284\,000 + 2652 = \mathbf{7\,006\,652} \checkmark$$

Karatsuba: 1234 · 5678 İzleme

1234·5678

karatsuba(1234, 5678): $n = 4$, $p = 100 \rightarrow a = 12$, $b = 34$, $c = 56$, $d = 78$. Üç çağrı: (34, 78), (46, 134), (12, 56).

Adım 1 / 21

```
1 long karatsuba(long x, long y) {  
2   if (x < 10 || y < 10) return x * y; // taban  
3   ▶ int n = Math.max(basamak(x), basamak(y));  
4   ▶ long p = (long) Math.pow(10, n / 2);  
5   ▶ long a = x / p, b = x % p; // x = a·p + b  
6   ▶ long c = y / p, d = y % p; // y = c·p + d  
7   long z0 = karatsuba(b, d);  
8   long z1 = karatsuba(a + b, c + d);  
9   long z2 = karatsuba(a, c);  
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;  
11 }
```

DEĞİŞKENLER

x = 1234 y = 5678 n = 4 p = 100

a = 12 b = 34 c = 56 d = 78

ÇAĞRI YIĞINI

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme

34·78

1234·5678

karatsuba(34, 78): $n = 2$, $p = 10 \rightarrow a = 3$, $b = 4$, $c = 7$, $d = 8$. Üç çağrı: (4, 8), (7, 15), (3, 7).

Adım 2 / 21

```
1 long karatsuba(long x, long y) {  
2   if (x < 10 || y < 10) return x * y; // taban  
3   ▶ int n = Math.max(basamak(x), basamak(y));  
4   ▶ long p = (long) Math.pow(10, n / 2);  
5   ▶ long a = x / p, b = x % p; // x = a·p + b  
6   ▶ long c = y / p, d = y % p; // y = c·p + d  
7   long z0 = karatsuba(b, d);  
8   long z1 = karatsuba(a + b, c + d);  
9   long z2 = karatsuba(a, c);  
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;  
11 }
```

DEĞİŞKENLER

x = 34

y = 78

n = 2

p = 10

a = 3

b = 4

c = 7

d = 8

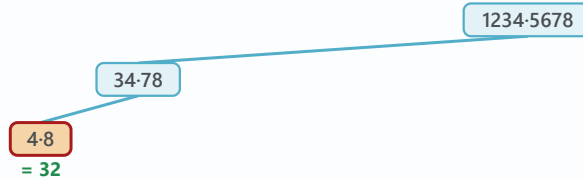
ÇAĞRI YIĞINI

karatsuba(34, 78)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



4 · 8: tek basamaklı çarpan var → **taban**, doğrudan 32.

Adım 3 / 21

```
1 long karatsuba(long x, long y) {  
2   if (x < 10 || y < 10) return x * y; // taban  
3   int n = Math.max(basamak(x), basamak(y));  
4   long p = (long) Math.pow(10, n / 2);  
5   long a = x / p, b = x % p; // x = a·p + b  
6   long c = y / p, d = y % p; // y = c·p + d  
7   long z0 = karatsuba(b, d);  
8   long z1 = karatsuba(a + b, c + d);  
9   long z2 = karatsuba(a, c);  
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;  
11 }
```

DEĞİŞKENLER

x = 4

y = 8

ÇAĞRI YIĞINI

karatsuba(4, 8)

karatsuba(34, 78)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



7 · 15: tek basamaklı çarpan var → **taban**, doğrudan 105.

Adım 4 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 7 y = 15

ÇAĞRI YIĞINI

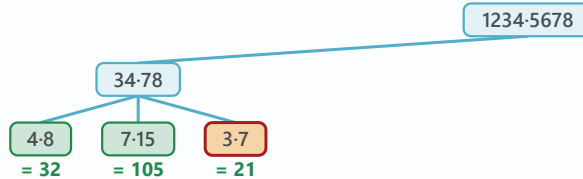
karatsuba(7, 15)

karatsuba(34, 78)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



3 · 7: tek basamaklı çarpan var → **taban**, doğrudan 21.

Adım 5 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 3 y = 7

ÇAĞRI YIĞINI

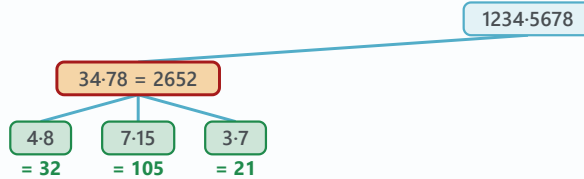
karatsuba(3, 7)

karatsuba(34, 78)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$$21 \cdot 100 + (105 - 21 - 32) \cdot 10 + 32 = 2100 + 520 + 32 = \mathbf{2652}.$$

Adım 6 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

$x = 34$ $y = 78$ $p = 10$ $z0 = 32$
 $z1 = 105$ $z2 = 21$

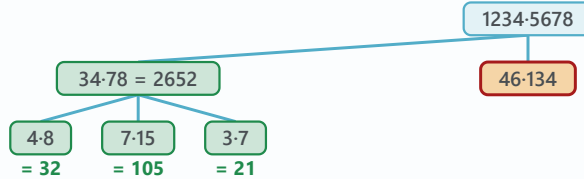
ÇAĞRI YİĞİNİ

karatsuba(34, 78)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



karatsuba(46, 134): $n = 3$, $p = 10 \rightarrow a = 4$, $b = 6$, $c = 13$, $d = 4$. Üç çağrı: (6, 4), (10, 17), (4, 13).

Adım 7 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   ▶ int n = Math.max(basamak(x), basamak(y));
4   ▶ long p = (long) Math.pow(10, n / 2);
5   ▶ long a = x / p, b = x % p; // x = a·p + b
6   ▶ long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

$x = 46$ $y = 134$ $n = 3$ $p = 10$
 $a = 4$ $b = 6$ $c = 13$ $d = 4$

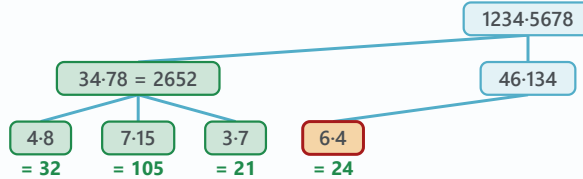
ÇAĞRI YİĞİNİ

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



6 · 4: tek basamaklı çarpan var → **taban**, doğrudan 24.

Adım 8 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 6 y = 4

ÇAĞRI YİĞİNİ

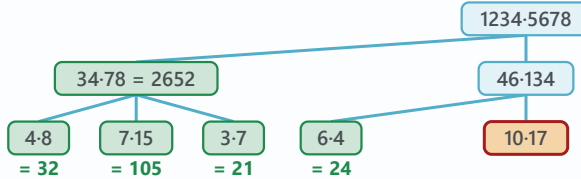
karatsuba(6, 4)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



karatsuba(10, 17): $n = 2$, $p = 10 \rightarrow a = 1$, $b = 0$, $c = 1$, $d = 7$. Üç çağrı: (0, 7), (1, 8), (1, 1).

Adım 9 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   ▶ int n = Math.max(basamak(x), basamak(y));
4   ▶ long p = (long) Math.pow(10, n / 2);
5   ▶ long a = x / p, b = x % p; // x = a·p + b
6   ▶ long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

$x = 10$ $y = 17$ $n = 2$ $p = 10$
 $a = 1$ $b = 0$ $c = 1$ $d = 7$

ÇAĞRI YİĞİNİ

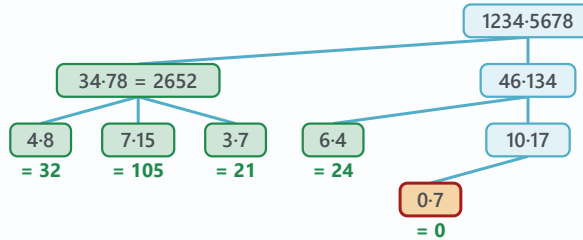
karatsuba(10, 17)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$0 \cdot 7$: tek basamaklı çarpan var → **taban**, doğrudan 0.

Adım 10 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

$x = 0$ $y = 7$

ÇAĞRI YİĞİNİ

karatsuba(0, 7)

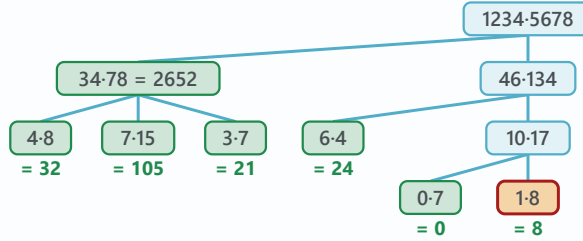
karatsuba(10, 17)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



1 · 8: tek basamaklı çarpan var → **taban**, doğrudan 8.

Adım 11 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 1 y = 8

ÇAĞRI YİĞİNİ

karatsuba(1, 8)

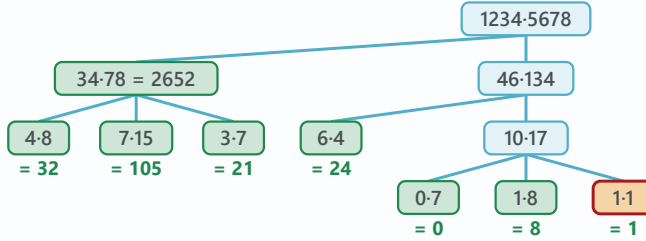
karatsuba(10, 17)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



1 · 1: tek basamaklı çarpan var → **taban**, doğrudan 1.

Adım 12 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 1 y = 1

ÇAĞRI YIĞINI

karatsuba(1, 1)

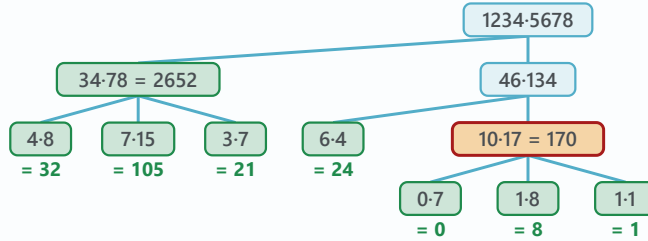
karatsuba(10, 17)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$$1 \cdot 100 + (8 - 1 - 0) \cdot 10 + 0 = 100 + 70 + 0 = \mathbf{170}.$$

Adım 13 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }

```

DEĞİŞKENLER

$x = 10$ $y = 17$ $p = 10$ $z0 = 0$
 $z1 = 8$ $z2 = 1$

ÇAĞRI YİĞİNİ

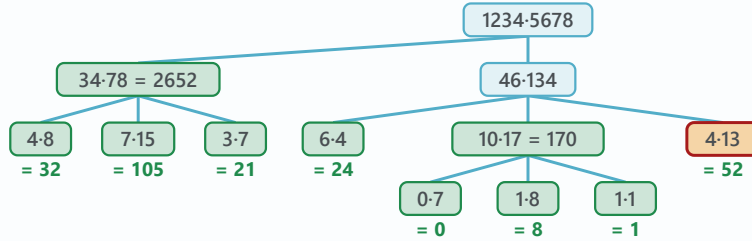
karatsuba(10, 17)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



4 · 13: tek basamaklı çarpan var → **taban**, doğrudan 52.

Adım 14 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 4 y = 13

ÇAĞRI YIĞINI

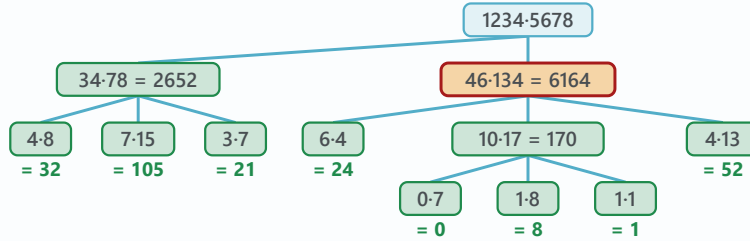
karatsuba(4, 13)

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$$52 \cdot 100 + (170 - 52 - 24) \cdot 10 + 24 = 5200 + 940 + 24 = \mathbf{6164}.$$

Adım 15 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }

```

DEĞİŞKENLER

$x = 46$ $y = 134$ $p = 10$ $z0 = 24$
 $z1 = 170$ $z2 = 52$

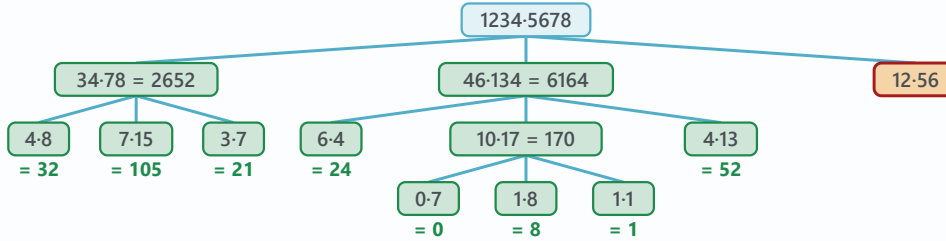
ÇAĞRI YİĞİNİ

karatsuba(46, 134)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



karatsuba(12, 56): $n = 2$, $p = 10 \rightarrow a = 1$, $b = 2$, $c = 5$, $d = 6$. Üç çağrı: (2, 6), (3, 11), (1, 5).

Adım 16 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   ▶ int n = Math.max(basamak(x), basamak(y));
4   ▶ long p = (long) Math.pow(10, n / 2);
5   ▶ long a = x / p, b = x % p; // x = a·p + b
6   ▶ long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

$x = 12$ $y = 56$ $n = 2$ $p = 10$
 $a = 1$ $b = 2$ $c = 5$ $d = 6$

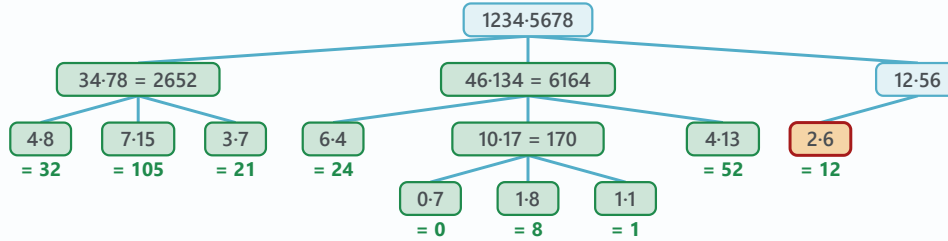
ÇAĞRI YIĞINI

karatsuba(12, 56)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



2 · 6: tek basamaklı çarpan var → **taban**, doğrudan 12.

Adım 17 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 2 y = 6

ÇAĞRI YIĞINI

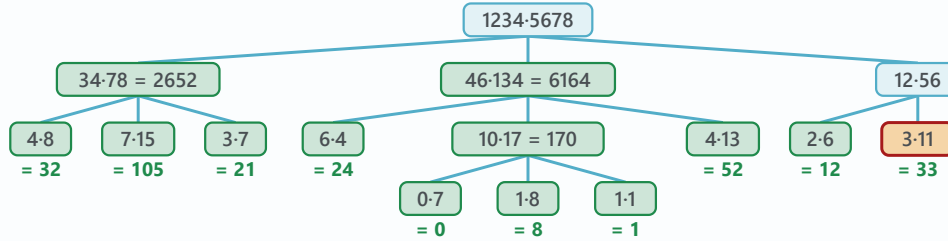
karatsuba(2, 6)

karatsuba(12, 56)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



3 · 11: tek basamaklı çarpan var → **taban**, doğrudan 33.

Adım 18 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 3 y = 11

ÇAĞRI YIĞINI

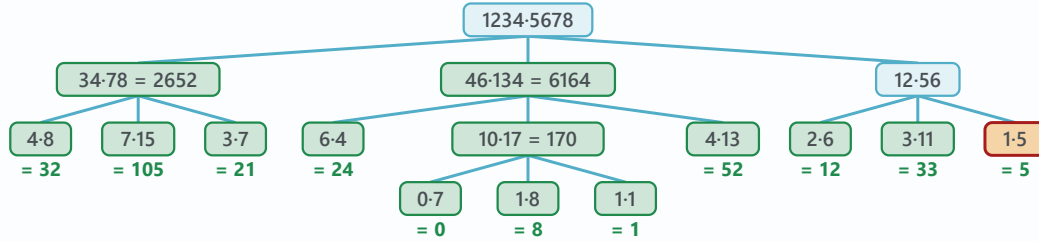
karatsuba(3, 11)

karatsuba(12, 56)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



1 · 5: tek basamaklı çarpan var → **taban**, doğrudan 5.

Adım 19 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 1 y = 5

ÇAĞRI YIĞINI

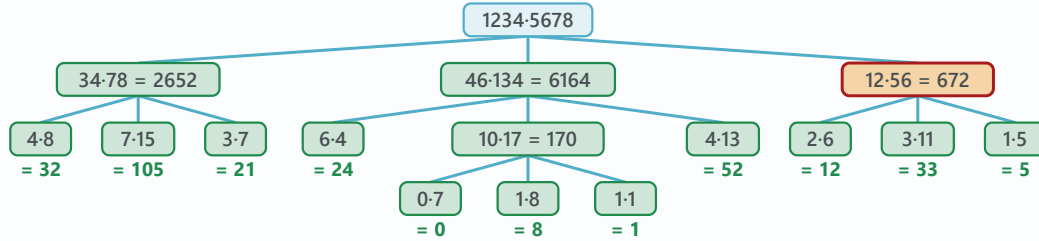
karatsuba(1, 5)

karatsuba(12, 56)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$$5 \cdot 100 + (33 - 5 - 12) \cdot 10 + 12 = 500 + 160 + 12 = \mathbf{672}.$$

Adım 20 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }

```

DEĞİŞKENLER

x = 12 y = 56 p = 10 z0 = 12
z1 = 33 z2 = 5

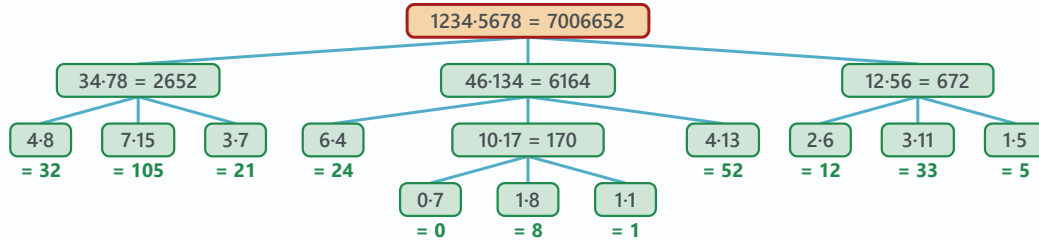
ÇAĞRI YİĞİNİ

karatsuba(12, 56)

karatsuba(1234, 5678)

main()

Karatsuba: 1234 · 5678 İzleme



$672 \cdot 10000 + (6164 - 672 - 2652) \cdot 100 + 2652 = 6720000 + 284000 + 2652 = \mathbf{7006652}$.

Adım 21 / 21

```

1 long karatsuba(long x, long y) {
2   if (x < 10 || y < 10) return x * y; // taban
3   int n = Math.max(basamak(x), basamak(y));
4   long p = (long) Math.pow(10, n / 2);
5   long a = x / p, b = x % p; // x = a·p + b
6   long c = y / p, d = y % p; // y = c·p + d
7   long z0 = karatsuba(b, d);
8   long z1 = karatsuba(a + b, c + d);
9   long z2 = karatsuba(a, c);
10  return z2 * p * p + (z1 - z2 - z0) * p + z0;
11 }
  
```

DEĞİŞKENLER

x = 1234 y = 5678 p = 100

z0 = 2652 z1 = 6164 z2 = 672

ÇAĞRI YİĞİNİ

karatsuba(1234, 5678)

main()

Karatsuba Analizi

- $T(n) = 3T(n/2) + O(n) \rightarrow \text{durum 1} \rightarrow \Theta(n^{\log_2 3}) \approx \Theta(n^{1,585})$.
- Toplama, çıkarma ve 10^m ile kaydırma $O(n)$; çarpım sayısını 4'ten 3'e indirmek üssü 2'den 1,585'e düşürür.
- Örnekte 16 çağrı, 11 tek basamaklı çarpım (okul yöntemi 16).
- Küçük sayılarda ek toplamalar yüzünden yavaştır $\rightarrow$ bir **eşik** altında okul yöntemi.
- Java `BigInteger.multiply` büyük sayılarda Karatsuba ve Toom-Cook kullanır. Daha hızlıları: Schönhage–Strassen $O(n \log n \log \log n)$, Harvey–van der Hoeven $O(n \log n)$ (2019).

n (basamak)	n^2	$n^{1,585}$	oran
10	100	38	2,6×
100	10 000	1 479	6,8×
1 000	$1,0 \cdot 10^6$	56 871	17,6×
10 000	$1,0 \cdot 10^8$	$2,2 \cdot 10^6$	45,7×
$1,0 \cdot 10^6$	$1,0 \cdot 10^{12}$	$3,2 \cdot 10^9$	309,2×

Matris Çarpımı: Klasik ve Blok

Klasik: $\Theta(n^3)$

```
for (int i = 0; i < n; i++)  
  for (int j = 0; j < n; j++)  
    for (int k = 0; k < n; k++)  
      C[i][j] += A[i][k] * B[k][j];
```

n^2 hücrenin her biri n çarpım $\rightarrow n^3$ çarpım.

Blok böl ve fethet

A, B, C'yi $n/2 \times n/2$ 'lik dört bloğa böl:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

8 blok çarpımı + 4 blok toplamı: $T(n) = 8T(n/2) + \Theta(n^2) \rightarrow n^{\log_2 8} = \Theta(n^3)$. Sadece bölmek yetmez; çarpım sayısı azaltılmalı.

Strassen (1969): 8 Yerine 7 Çarpım

Çarpım	Tanım
P1	$A_{11} \cdot (B_{12} - B_{22})$
P2	$(A_{11} + A_{12}) \cdot B_{22}$
P3	$(A_{21} + A_{22}) \cdot B_{11}$
P4	$A_{22} \cdot (B_{21} - B_{11})$
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$

Sonuç blokları

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

7 çarpım + 18 blok toplama/çıkarma.
Bloklar da Strassen ile özyinelemeli
çarpılır; taban 1×1 .

Dikkat: C22'de P7 **çıkartılır**. "+ P7" yazılırsa program hata vermeden yanlış matris üretir; uygulamayı küçük bir örnekte klasik yöntemle karşılaştırarak test edin.

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	...	
P3	$(A_{21} + A_{22}) \cdot B_{11}$	...	
P4	$A_{22} \cdot (B_{21} - B_{11})$	...	
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	...	
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	...	
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P1 = 1 \cdot (8 - 2) = 6.$

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	...	
P4	$A_{22} \cdot (B_{21} - B_{11})$	...	
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	...	
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	...	
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P2 = (1 + 3) \cdot 2 = 8.$

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	$(7 + 5) \cdot 6$	72
P4	$A_{22} \cdot (B_{21} - B_{11})$	...	
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	...	
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	...	
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P3 = (7 + 5) \cdot 6 = \mathbf{72}.$

Strassen Adım Adım: 2 × 2 Örnek

$A = \begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \times B = \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	$(7 + 5) \cdot 6$	72
P4	$A_{22} \cdot (B_{21} - B_{11})$	$5 \cdot (4 - 6)$	-10
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	...	
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	...	
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P4 = 5 \cdot (4 - 6) = -10.$

Strassen Adım Adım: 2 × 2 Örnek

$A = \begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \times B = \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	$(7 + 5) \cdot 6$	72
P4	$A_{22} \cdot (B_{21} - B_{11})$	$5 \cdot (4 - 6)$	-10
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	$(1 + 5) \cdot (6 + 2)$	48
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	...	
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P5 = (1 + 5) \cdot (6 + 2) = 48.$

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	$(7 + 5) \cdot 6$	72
P4	$A_{22} \cdot (B_{21} - B_{11})$	$5 \cdot (4 - 6)$	-10
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	$(1 + 5) \cdot (6 + 2)$	48
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	$(3 - 5) \cdot (4 + 2)$	-12
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	...	

$P6 = (3 - 5) \cdot (4 + 2) = -12.$

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

P1	$A_{11} \cdot (B_{12} - B_{22})$	$1 \cdot (8 - 2)$	6
P2	$(A_{11} + A_{12}) \cdot B_{22}$	$(1 + 3) \cdot 2$	8
P3	$(A_{21} + A_{22}) \cdot B_{11}$	$(7 + 5) \cdot 6$	72
P4	$A_{22} \cdot (B_{21} - B_{11})$	$5 \cdot (4 - 6)$	-10
P5	$(A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	$(1 + 5) \cdot (6 + 2)$	48
P6	$(A_{12} - A_{22}) \cdot (B_{21} + B_{22})$	$(3 - 5) \cdot (4 + 2)$	-12
P7	$(A_{11} - A_{21}) \cdot (B_{11} + B_{12})$	$(1 - 7) \cdot (6 + 8)$	-84

$P7 = (1 - 7) \cdot (6 + 8) = \textbf{-84}.$

Strassen Adım Adım: 2 × 2 Örnek

A =

1	3
7	5

×

B =

6	8
4	2

C11 = P5 + P4 - P2 + P6	48 + (-10) - 8 + (-12)	18
C12 = P1 + P2	6 + 8	14
C21 = P3 + P4	72 + (-10)	62
C22 = P5 + P1 - P3 - P7	48 + 6 - 72 - (-84)	66

C = [[18, 14], [62, 66]]. Kontrol: 1·6 + 3·4 = 18, 7·8 + 5·2 = 66 ✓. 7 çarpım kullanıldı (klasik 8).

Strassen Analizi ve Pratik

- $T(n) = 7T(n/2) + \Theta(n^2) \rightarrow \text{durum 1} \rightarrow \Theta(n^{\log_2 7}) \approx \Theta(n^{2,807})$.
- $n = 2^k$ için skaler çarpım: 7^k (klasik 8^k). $n = 1024$: $7^{10} \approx 2,8 \cdot 10^8$, $8^{10} \approx 1,07 \cdot 10^9 \rightarrow$ yaklaşık **3,8 kat** az.
- n 2'nin kuvveti değilse matris sıfırlarla **doldurulur** (padding).
- **Eşik:** 18 ek toplama ve geçici matrisler yüzünden küçük n 'de yavaş; kütüphaneler $n \lesssim 64$ – 128 'de klasik yöntemle geçer.
- **Sayısal kararlılık:** kayan noktalı sayılarda çıkarma işlemleri hatayı büyütebilir.
- **Bellek:** her düzeyde geçici bloklar $\rightarrow$ ek $O(n^2)$.
- Kuramsal rekor $\omega < 2,372$ (2024); sabitler çok büyük, pratikte kullanılmaz.

Ders: çarpımı toplamalarla takas etmek, **a**'yı bir azaltarak üssü düşürür. Aynı fikir Karatsuba'da $4 \rightarrow 3$, Strassen'de $8 \rightarrow 7$.

Saf Yöntem ve Böl ve Fethet

Problem	Saf yöntem	Böl ve fethet	Bağıntı
Sıralama	$O(n^2)$ (kabarcık)	$O(n \log n)$	$2T(n/2) + n$
Sıralı dizide arama	$O(n)$	$O(\log n)$	$T(n/2) + 1$
Üs alma x^n	$O(n)$ çarpım	$O(\log n)$	$T(n/2) + 1$
Dışbükey örtü	$O(n^3)$	ort. $O(n \log n)$	$T(n_1) + T(n_2) + n$
En yakın nokta çifti	$O(n^2)$	$O(n \log n)$	$2T(n/2) + n$
Tam sayı çarpımı	$O(n^2)$	$O(n^{1,585})$	$3T(n/2) + n$
Matris çarpımı	$O(n^3)$	$O(n^{2,807})$	$7T(n/2) + n^2$

Kazancın kaynağı: ya iş dengeli bölünüp her düzey $O(n)$ 'de birleştiriliyor (sıralama, en yakın çift) ya da alt problem sayısı azaltılıyor (Karatsuba, Strassen) ya da yarısı atılıyor (arama, üs).

Böl ve Fethet, Dinamik Programlama, Açgözlü

	Böl ve fethet	Dinamik programlama	Açgözlü
Alt problemler	bağımsız	örtüşen	tek bir alt problem kalır
Yön	yukarıdan aşağı (özyineleme)	çoğunlukla aşağıdan yukarı (tablo)	adım adım, geri dönüş yok
Karar	sabit bölme kuralı	tüm seçenekleri dener, en iyisini saklar	o anda en iyi görüneni seçer
Örnek	birleştirme sıralaması, Strassen	Fibonacci, sırt çantası	Dijkstra, Huffman, Kruskal

Soru sırası: alt problemler bağımsız mı? → **böl ve fethet**. Örtüşüyor mu? → **dinamik programlama**. Yerel en iyi seçim küresel en iyiye götürüyor mu? → **açgözlü** (Bölüm 12).

ÖZET

Özet

Konu	Hatırlanacaklar
Paradigma	böl → (parçala) → yönet → birleştir; bağımsız, aynı türden alt problemler; taban durumu şart
Analiz	$T(n) = aT(n/b) + f(n)$; özyineleme ağacı; ana teorem: $f(n)$ 'yi $n^{\log_b a}$ ile karşılaştır
Temel örnekler	ikili arama ve hızlı üs $O(\log n)$; birleştirme sıralaması $O(n \log n)$; hızlı sıralama ort. $O(n \log n)$, en kötü $O(n^2)$
Quickhull	A, B uçlar; en uzak C (max capraz); üçgen içi elenir; ort. $O(n \log n)$, en kötü $O(n^2)$
En yakın çift	x'e göre böl; $\delta = \min(\delta_L, \delta_R)$; 2δ şeritte ≤ 7 komşu; $O(n \log n)$
Karatsuba	$ad + bc = (a + b)(c + d) - ac - bd$; 3 çarpım $\rightarrow O(n^{1,585})$
Strassen	7 blok çarpımı $\rightarrow O(n^{2,807})$; $C22 = P5 + P1 - P3 - P7$

ÖZET

Etkileşimli Simülatörler

Özyineleme ve Ana Teorem

$T(n) = aT(n/b) + f(n)$ biçimindeki bağıntıların özyineleme ağacını düzey düzey çizer ve ana teorem durumunu belirler.

Sıralama Algoritmaları

Birleştirme sıralaması ve hızlı sıralama dahil on sıralama algoritmasını adım adım görselleştirir.

Matris Algoritmaları

Matris toplama, çarpma, devrik, determinant ve Gauss eliminasyonu; klasik $\Theta(n^3)$ çarpım ile Strassen'in $O(n^{2,807})$ sınırı.

ÖZET

Sık Yapılan Hatalar

Taban durumunu unutmak

Ya da hiç ulaşılamayan bir taban yazmak: sonsuz özyineleme, `StackOverflowError`.
Karatsuba'da $x < 10 \parallel y < 10$.

Dengesiz bölme

Sıralı dizide son öge pivot ya da Quickhull'da her adımda tek nokta: $T(n) = T(n - 1) + n = O(n^2)$.

Örtüşen alt problemler

Fibonacci'yi düz özyinelemeyle hesaplamak üstel süre alır; sonuçları saklayan dinamik programlama gerekir.

Birleştirme maliyetini küçümsemek

Şeridi her çağrıda sıralamak $O(n \log^2 n)$ yapar; $2T(n/2) + n \log n \neq \Theta(n \log n)$.

Ana teoremi yanlış yerde kullanmak

$T(n) = T(n - 1) + n$, eşit olmayan bölmeler veya $f(n) = n/\log n$ için geçerli değildir.

İşaret hatası

Strassen'de $C22 = P5 + P1 - P3 - P7$; tek işaret hatası hata vermeden yanlış matris üretir. Küçük örneklerle klasik yöntemle karşı test edin.

ÖZET

Örnek Problemler

1. $T(n) = 4T(n/2) + n$ bağıntısını ana teoremle çözün.
2. $T(n) = 2T(n/2) + n^2$ bağıntısını çözün; düzenlilik koşulunu kontrol edin.
3. Karatsuba ile $21 \cdot 43$ 'ü hesaplayın ($m = 1$).
4. $A = (0,0)$, $B = (10,0)$. $P = (3,4)$, $Q = (5,-2)$, $R = (7,6)$ için $\text{capraz}(A, B, \cdot)$ değerlerini bulun. Quickhull üst kümede hangi noktayı seçer?
5. `us(3, 10)` kaç çarpım yapar? Saf yöntemle kıyaslayın.
6. $[5, 1, 4, 2, 3]$ dizisini birleştirme sıralamasıyla sıralarken son birleştirmede hangi iki sıralı dizi birleşir?

► Cevapları göster

ÖZET

Alıştırmalar

1. Dizinin en büyük ögesini böl ve fethet ile bulun; bağıntıyı çözün.
2. Birleştirme sıralamasını değiştirip **ters çiftleri** ($i < j, a[i] > a[j]$) sayın. [2, 4, 1, 3, 5] için sonuç?
3. **En büyük toplamalı alt dizi**: [-2, 1, -3, 4, -1, 2, 1, -5, 4] için böl ve fethet çözümü ve süresi.
4. $A = [[1, 2], [3, 4]]$, $B = [[5, 6], [7, 8]]$ çarpımını Strassen ile hesaplayın; C22'yi $\pm P7$ ile deneyin.
5. Quickhull için $O(n^2)$ süren bir girdi tarif edin.
6. En yakın çift algoritmasında taban durumu $n \leq 3$ yerine $n \leq 1$ olsaydı ne değişirdi?

► Cevaplar / ipuçları

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 4: Böl ve fethet; Bölüm 33: Hesaplamalı geometri).
- Kleinberg ve Tardos, *Algorithm Design*, 2005 (Bölüm 5: Divide and Conquer).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011.
- Barber, Dobkin ve Huhdanpaa, "The Quickhull Algorithm for Convex Hulls", *ACM TOMS* 22(4), 1996.
- Karatsuba ve Ofman, "Multiplication of Multidigit Numbers on Automata", 1962; Strassen, "Gaussian Elimination is not Optimal", *Numerische Mathematik* 13, 1969.

Son

Bölüm 11: Böl ve Fethet Yaklaşımı

Sonraki bölüm: Bölüm 12: Açgözlü Algoritmalar · [Terim sözlüğü](#)