



Bölüm 1: Giriş

Algoritmalar



Algoritma

- Bir problemi çözmek veya görevi yerine getirmek için,
 - adım adım yönergeler dizisidir.
- Her adım belirli, açık ve anlaşılır olmalıdır.
- Girdi verileri alınır, işlemler yapılır, çıktı elde edilir.
- Algoritma, sınırlı sayıda adımda sonlanmalıdır.



Çay Hazırlama Algoritması

Girdi: Su, Çay yaprakları, Şeker (isteğe bağlı), Süt (isteğe bağlı)

Çıktı: Çay

1. Suyu kaynat.
2. Kaynamış suyu, demliğe koy.
3. Demliğin içine çay yapraklarını ekle.
4. Çayın demlenmesini bekle.
5. Demlenen çayı fincana süzerek koy.
6. İsteğe bağlı olarak şeker ve süt ekle.



Ortalama Bulma Algoritması

Girdi: Sayı1, Sayı2, Sayı3

Çıktı: Ortalama

1. $\text{Toplam} = \text{Sayı1} + \text{Sayı2} + \text{Sayı3}$
2. $\text{Ortalama} = \text{Toplam} / 3$
3. Ortalama değerini ekrana yazdır.



Ortalama Bulma

```
def ortalama_hesapla(sayi1, sayi2, sayi3):  
    ortalama = (sayi1 + sayi2 + sayi3) / 3  
    return ortalama
```

```
sayi1 = float(input("Birinci sayıyı girin: "))  
sayi2 = float(input("İkinci sayıyı girin: "))  
sayi3 = float(input("Üçüncü sayıyı girin: "))
```

```
ort = ortalama_hesapla(sayi1, sayi2, sayi3)  
print("Girilen sayıların ortalaması:", ort)
```



Ortalama Bulma

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    System.out.println("Birinci sayıyı girin: ");  
    double sayi1 = scanner.nextDouble();  
    System.out.println("İkinci sayıyı girin: ");  
    double sayi2 = scanner.nextDouble();  
    System.out.println("Üçüncü sayıyı girin: ");  
    double sayi3 = scanner.nextDouble();  
    double ortalama = (sayi1 + sayi2 + sayi3) / 3;  
    System.out.println("Sayıların ortalaması: " + ortalama);  
}
```



Algoritma Analizi

- Bir algoritmanın performansını anlamak ve değerlendirmek için kullanılır.
- İşlemci ve bellek gibi kaynakların verimli kullanılmasını sağlar.
- **Zaman Analizi:**
 - Algoritmanın çalışma süresini ölçer.
 - Genellikle *Big-O* notasyonu ile gösterilir.
- **Bellek Analizi:**
 - Algoritmanın bellek kullanımını değerlendirir.
 - Bellek karmaşıklığını gösterir.
- *Big-O notasyonu*, algoritmaların en kötü durum performansını ifade eder.

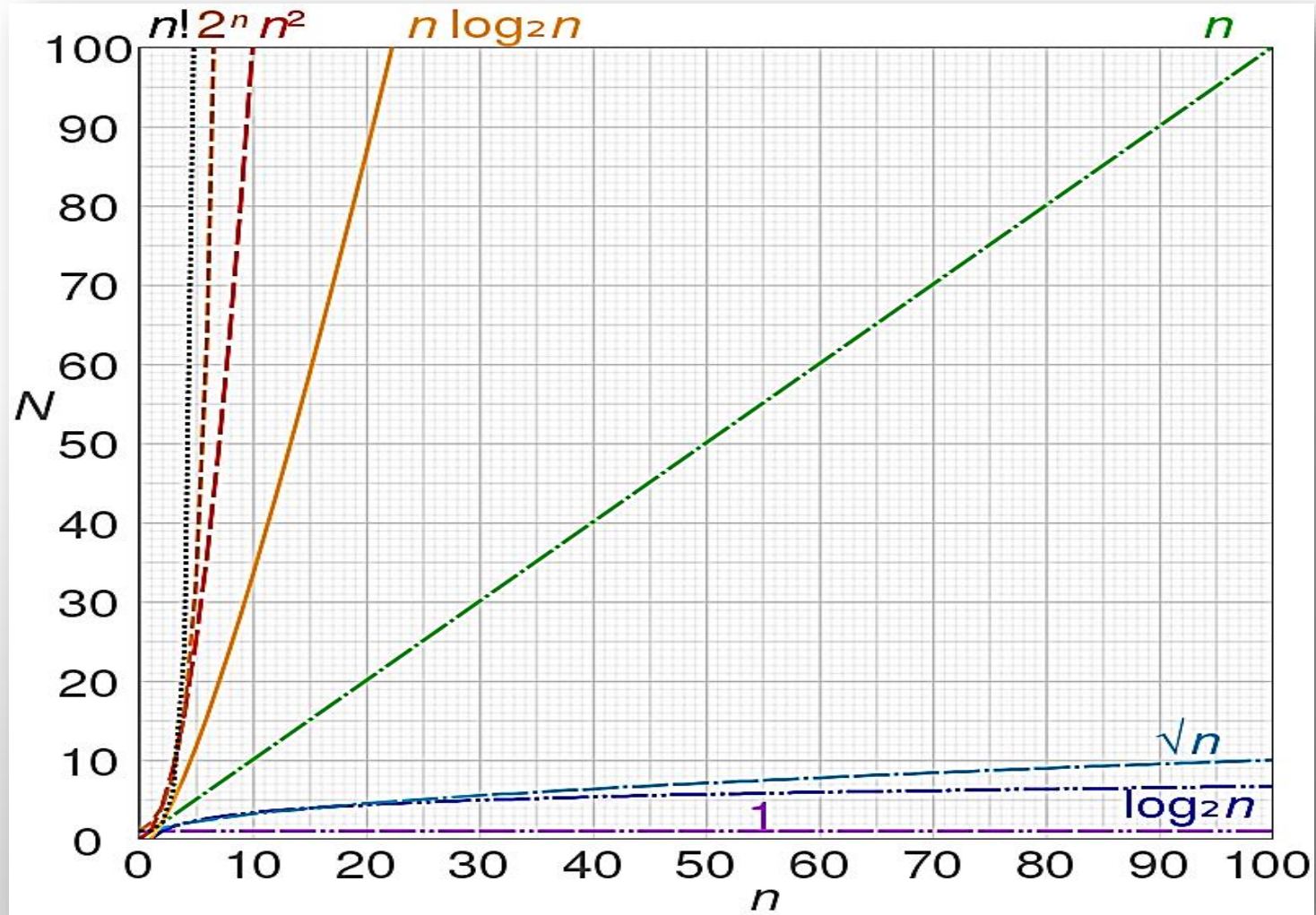


Zaman Karmaşıklığı

- Bir algoritmanın çalışma süresini ölçen bir kavram.
- Çalışma süresini, girdi boyutu (n) ile ilişkilendirir.
- Veri setine bağlı olarak algoritmanın performans değişimini inceler.
- Kod değerlendirilirken zaman ve alan karmaşıklığı kullanılır.
- Bir algoritmanın *iyi* veya *kötü* olduğunu belirlemede kritik bir faktör.



Zaman Karmaşıklığı





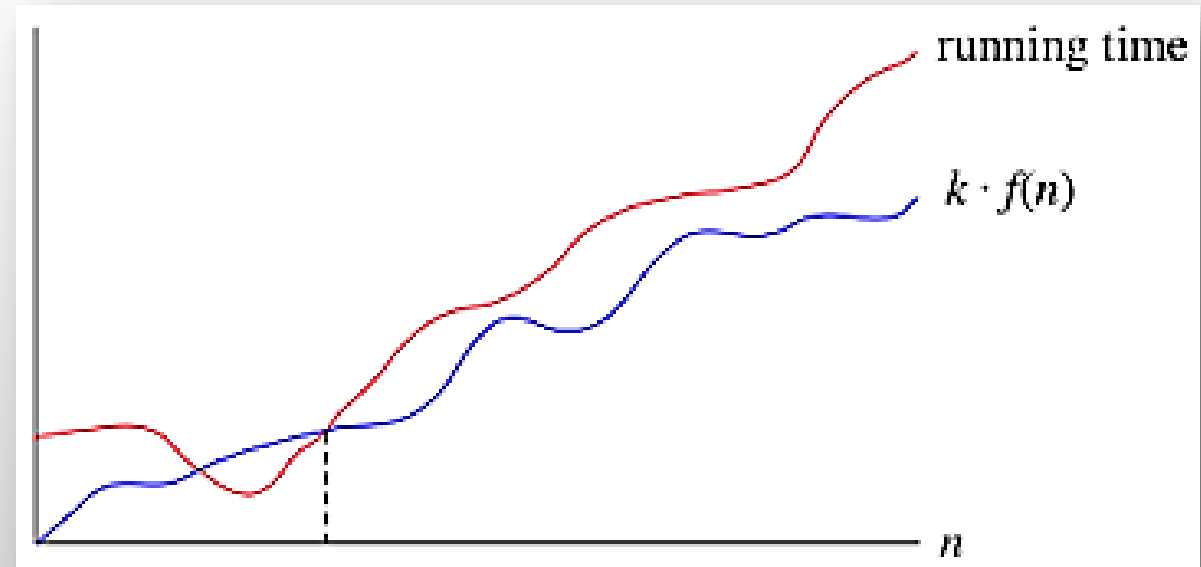
Asimptotik Gösterim Türleri

- *Omega* Ω Gösterimi
- *Big O* Gösterimi
- *Theta* Θ Gösterimi



Omega (Ω) Gösterimi

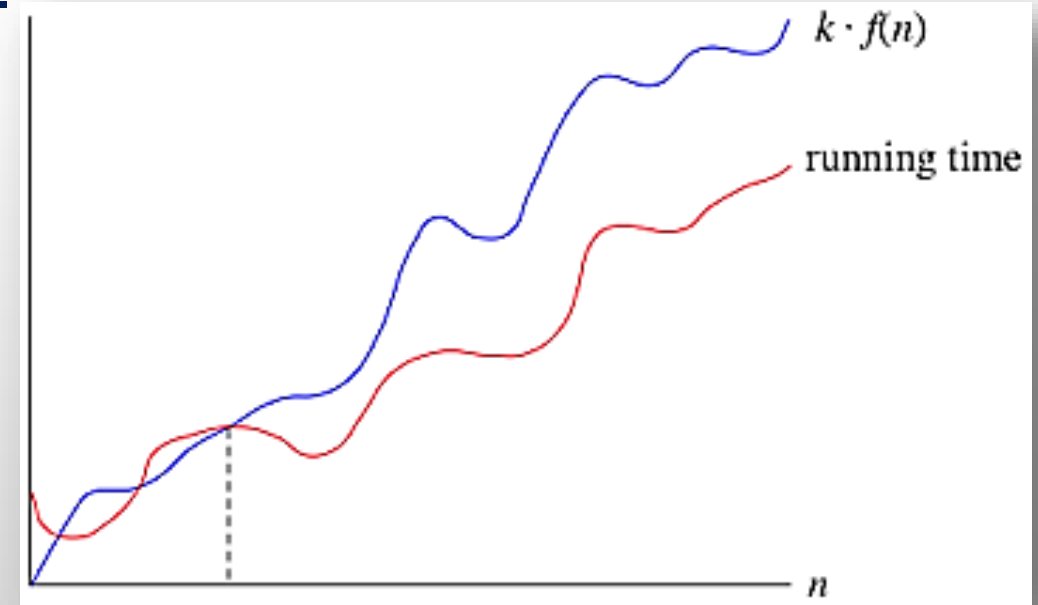
- Bir algoritmanın *en iyi durum* çalışma süresini ifade eder.
- Çalışma süresinin alt sınırını ifade eder.
- En iyi durum çalışma süresi *100 saniye* ise,
- Çalışma süresi *100 saniyeden* uzun sürebilir.
- Ancak, *100 saniyeden* kısa olamaz.





Büyük (Big) O Gösterimi

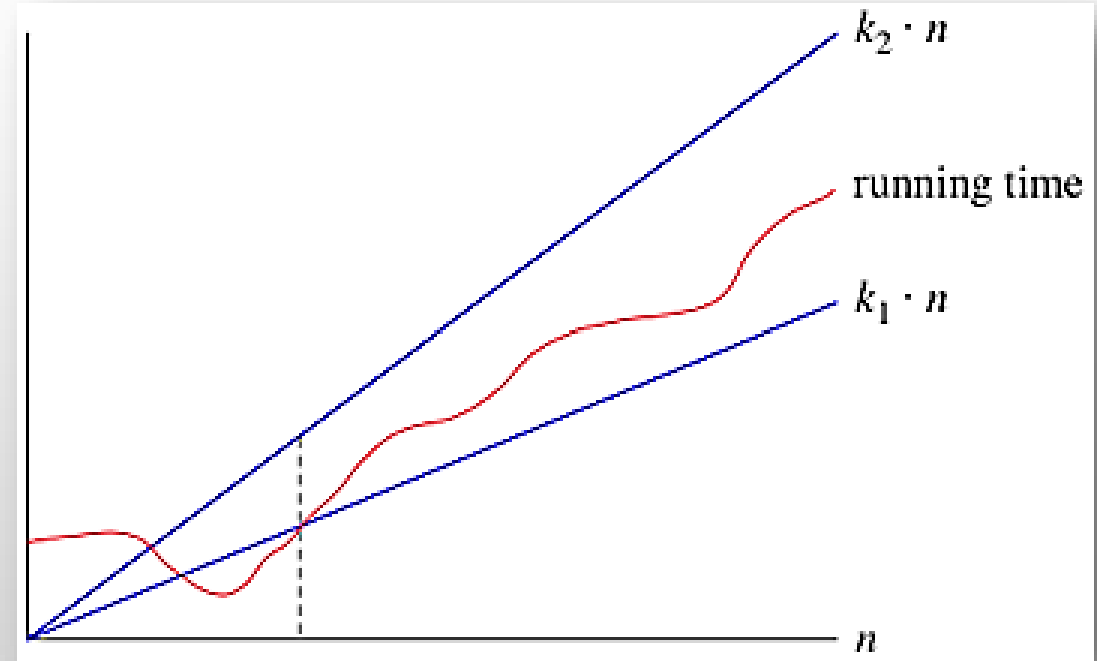
- Bir algoritmanın *en kötü durum* çalışma süresini ifade eder.
- Çalışma süresinin üst sınırını ifade eder.
- En kötü durum çalışma süresi *100 saniye* ise,
- Çalışma süresi *100 saniyeden* kısa sürebilir.
- Ancak, *100 saniyeden* uzun olamaz.





Theta (Θ) Gösterimi

- Algoritmanın hem *en kötü* hem *en iyi durum* çalışma süresini ifade eder.
- Örneğin bir sıralama algoritması,
 - En kötü durum karmaşıklığı $O(n \log n)$
 - En iyi durum karmaşıklığı $\Omega(n)$ ise,
 - Çalışma süresi $\Theta(n \log n)$ ile gösterilir.





Doğrusal Arama

```
var doLinearSearch = function(array, targetValue) {  
  for (var guess = 0; guess < array.length; guess++) {  
    if (array[guess] === targetValue) {  
      return guess; // bulundu!  
    }  
  }  
  return -1; // bulunamadı  
};
```



Doğrusal Arama Analizi

- **Zaman Karmaşıklığı:**
 - $c1 \times n + c2$
 - $c1$ ve $c2$ bilgisayarın hızı, programlama dili gibi faktörlere bağlıdır.
- **Big- Θ Notasyonu:**
 - $\Theta(n)$
 - Zaman, girdi boyutu n ile orantılı olarak büyür.
- $k1$ ve $k2$ için $k1 \times n \leq T(n) \leq k2 \times n$ geçerlidir.



İlk Elemanı Yazdırma

```
def print_first_element(arr):  
    if len(arr) > 0:  
        print("First element:", arr[0])  
    else:  
        print("Array is empty")
```




İlk Elemanı Yazdırma

```
def print_first_element(arr):  
    if len(arr) > 0:  
        print("First element:", arr[0])  
    else:  
        print("Array is empty")
```

$$T(n) = O(1)$$



İkili Arama

```
def binary_search(arr, target):  
    low, high = 0, len(arr) - 1  
    while low <= high:  
        mid = (low + high) // 2  
        guess = arr[mid]  
        if guess == target:  
            return mid # Hedef eleman bulundu  
        elif guess < target:  
            low = mid + 1  
        else:  
            high = mid - 1  
    return -1 # Hedef eleman bulunamadı
```



İkili Arama

```
def binary_search(arr, target):  
    low, high = 0, len(arr) - 1  
    while low <= high:  
        mid = (low + high) // 2  
        guess = arr[mid]  
        if guess == target:  
            return mid # Hedef eleman bulundu  
        elif guess < target:  
            low = mid + 1  
        else:  
            high = mid - 1  
    return -1 # Hedef eleman bulunamadı
```

$$T(n) = O(\log n)$$



Asal Sayı Kontrol

```
def is_prime(n):  
    if n <= 1:  
        return False  
    elif n == 2:  
        return True  
    elif n % 2 == 0:  
        return False  
    sqrt_n = int(math.sqrt(n)) + 1  
    for i in range(3, sqrt_n, 2):  
        if n % i == 0:  
            return False  
    return True
```



Asal Sayı Kontrol

```
def is_prime(n):  
    if n <= 1:  
        return False  
    elif n == 2:  
        return True  
    elif n % 2 == 0:  
        return False  
    sqrt_n = int(math.sqrt(n)) + 1  
    for i in range(3, sqrt_n, 2):  
        if n % i == 0:  
            return False  
    return True
```

$$T(n) = O(\sqrt{n})$$



En Büyük Elemanı Bulma

```
def find_max(arr):  
    if not arr:  
        return None # Dizi boşsa  
    max_value = arr[0]  
    for element in arr:  
        if element > max_value:  
            max_value = element  
    return max_value
```



En Büyük Elemanı Bulma

```
def find_max(arr):  
    if not arr:  
        return None # Dizi boşsa  
    max_value = arr[0]  
    for element in arr:  
        if element > max_value:  
            max_value = element  
    return max_value
```

$$T(n) = O(n)$$



Çabuk Sıralama

```
def quicksort(arr):  
    if len(arr) <= 1:  
        return arr  
    else:  
        pivot = arr[0]  
        less = [x for x in arr[1:] if x <= pivot]  
        greater = [x for x in arr[1:] if x > pivot]  
        return quicksort(less) + [pivot] + quicksort(greater)
```




Çabuk Sıralama

```
def quicksort(arr):  
    if len(arr) <= 1:  
        return arr  
    else:  
        pivot = arr[0]  
        less = [x for x in arr[1:] if x <= pivot]  
        greater = [x for x in arr[1:] if x > pivot]  
        return quicksort(less) + [pivot] + quicksort(greater)
```

$$T(n) = O(n \log n)$$



Kabarcık Sıralama

```
def bubble_sort(arr):  
    n = len(arr)  
  
    for i in range(n):  
        for j in range(0, n-i-1):  
            if arr[j] > arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]
```



Kabarcık Sıralama

```
def bubble_sort(arr):  
    n = len(arr)
```

```
    for i in range(n):
```

```
        for j in range(0, n-i-1):
```

```
            if arr[j] > arr[j+1]:
```

```
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

$$T(n) = O(n^2)$$



Matris Çarpımı

```
def matrix_multiply(A, B):  
    result = []  
    for i in range(len(A)):  
        row = []  
        for j in range(len(B[0])):  
            element = 0  
            for k in range(len(B)):  
                element += A[i][k] * B[k][j]  
            row.append(element)  
        result.append(row)  
    return result
```



Matris Çarpımı

```
def matrix_multiply(A, B):  
    result = []  
    for i in range(len(A)):  
        row = []  
        for j in range(len(B[0])):  
            element = 0  
            for k in range(len(B)):  
                element += A[i][k] * B[k][j]  
            row.append(element)  
        result.append(row)  
    return result
```

$$T(n) = O(n^3)$$



Permütasyon Oluşturma

```
def generate_all_permutations(elements):  
    if len(elements) == 0:          return [[]]  
    all = []  
    for i in range(len(elements)):  
        current = elements[i]  
        remainings = elements[:i] + elements[i+1:]  
        subs = generate_all_permutations(remainings)  
        for sub in subs:  
            all.append([current] + sub)  
    return all
```



Permütasyon Oluşturma

```
def generate_all_permutations(elements):  
    if len(elements) == 0:          return [[]]  
    all = []  
    for i in range(len(elements)):  
        current = elements[i]  
        remainings = elements[:i] + elements[i+1:]  
        subs = generate_all_permutations(remainings)  
        for sub in subs:  
            all.append([current] + sub)  
    return all
```

$$T(n) = O(n!)$$



Fibonacci Serisi

```
def fibonacci(n, memo={}):  
    if n <= 1:  
        return n  
    elif n not in memo:  
        memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)  
    return memo[n]
```




Fibonacci Serisi

```
def fibonacci(n, memo={}):  
    if n <= 1:  
        return n  
    elif n not in memo:  
        memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)  
    return memo[n]
```

$$T(n) = O(2^n)$$





Sıralama Problemi

- **Girdi:** $\langle a_1, a_2, \dots, a_n \rangle$ sayı dizisi.
- **Çıktı:** $\langle a'_1, a'_2, \dots, a'_n \rangle$ sayı dizisi ($a'_1 \leq a'_2 \leq \dots \leq a'_n$)

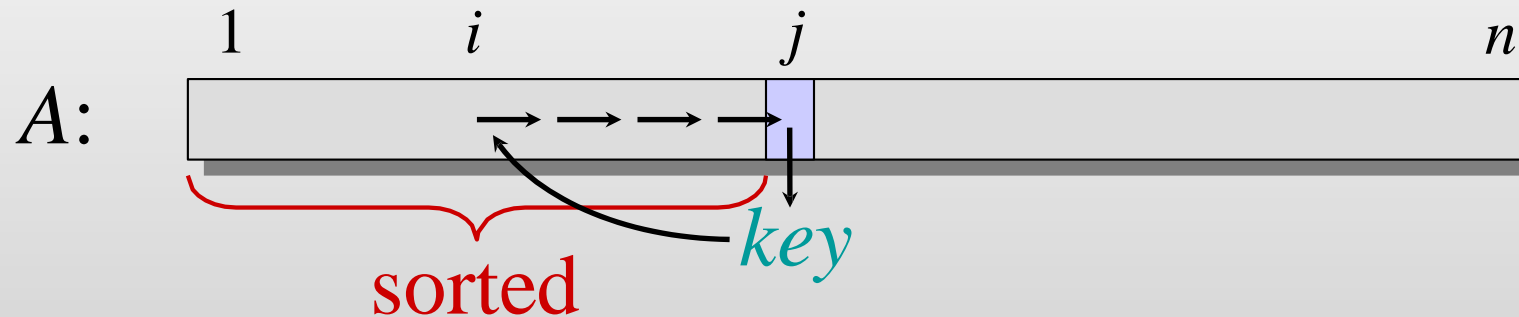
- **Örnek:**

- Girdi: 8 2 4 9 3 6
- Çıktı: 2 3 4 6 8 9



Araya Ekleyerek Sıralama

- *Insertion sort*
- Algoritma, bir diziyi sıralamak için iç içe geçmiş iki döngü kullanır.
- Dıştaki döngü, dizinin her elemanını bir kere geçer.
- İçteki döngü ise, mevcut elemanı uygun konuma yerleştirir.
- Zaman karmaşıklığı $O(n^2)$.





Araya Ekleyerek Sıralama

```
def Sira1a(A):  
    for i in range(1, len(A)):  
        key = A[i]  
        j = i - 1  
        // A[0..i-1] alt dizisi sıralı.  
        // Key, uygun konuma yerleştirilir.  
        while j >= 0 and A[j] > key:  
            A[j + 1] = A[j]  
            j = j - 1  
        A[j + 1] = key
```

n-1 adımlı
dış döngü

en kötü durumda n
adımlı iç döngü



Araya Ekleyerek Sıralama

```
public void Sirala(int[] A) {  
    for (int i = 1; i < A.length; ++i) {  
        int key = A[i];  
        int j = i - 1;  
        while (j >= 0 && A[j] > key) {  
            A[j + 1] = A[j];  
            j = j - 1;  
        }  
        A[j + 1] = key;  
    }  
}
```

Örnek



8 2 4 9 3 6

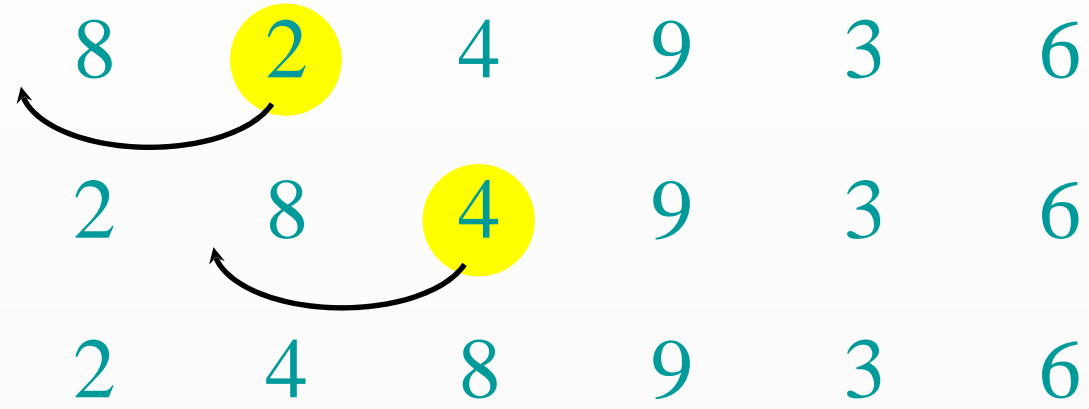


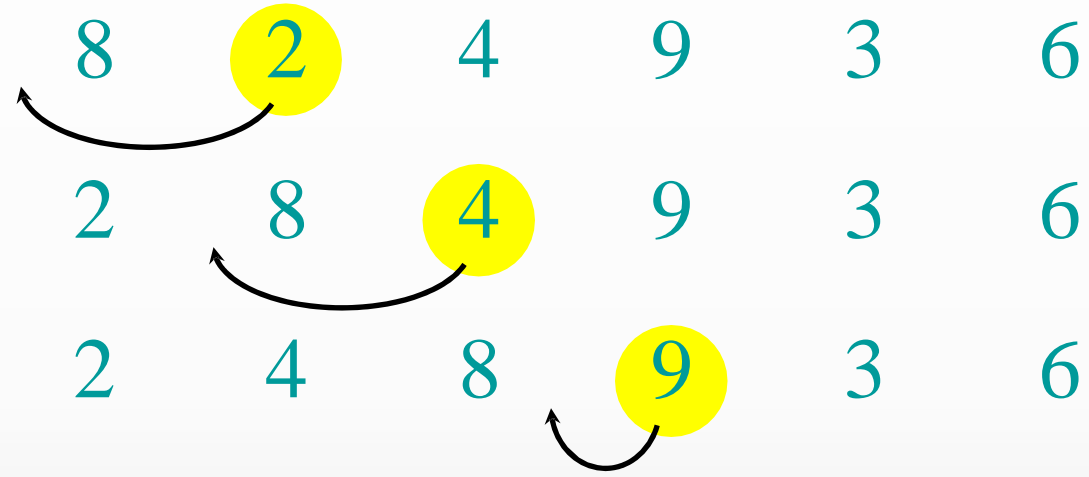
8 2 4 9 3 6

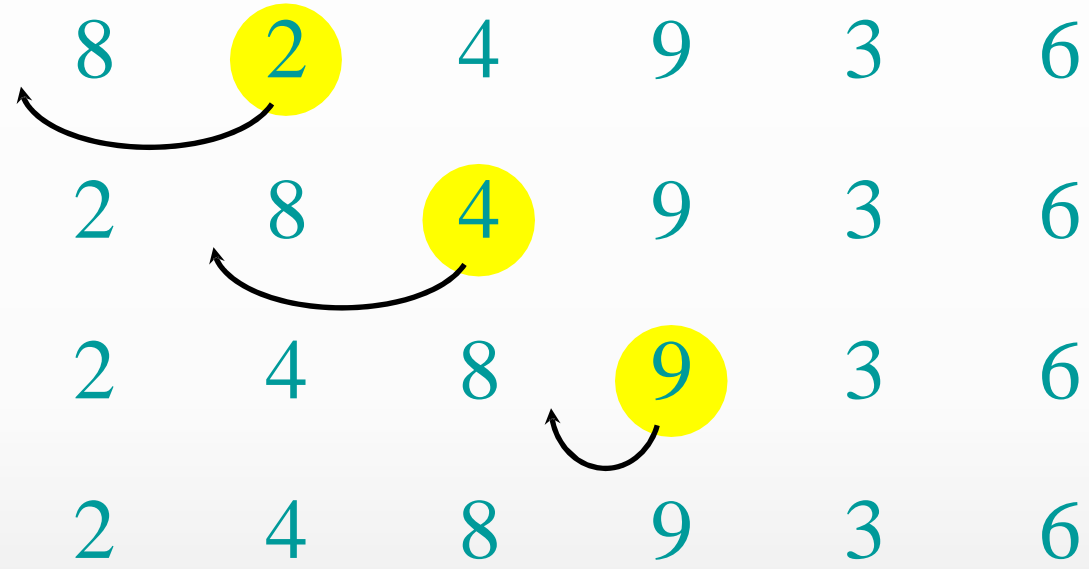


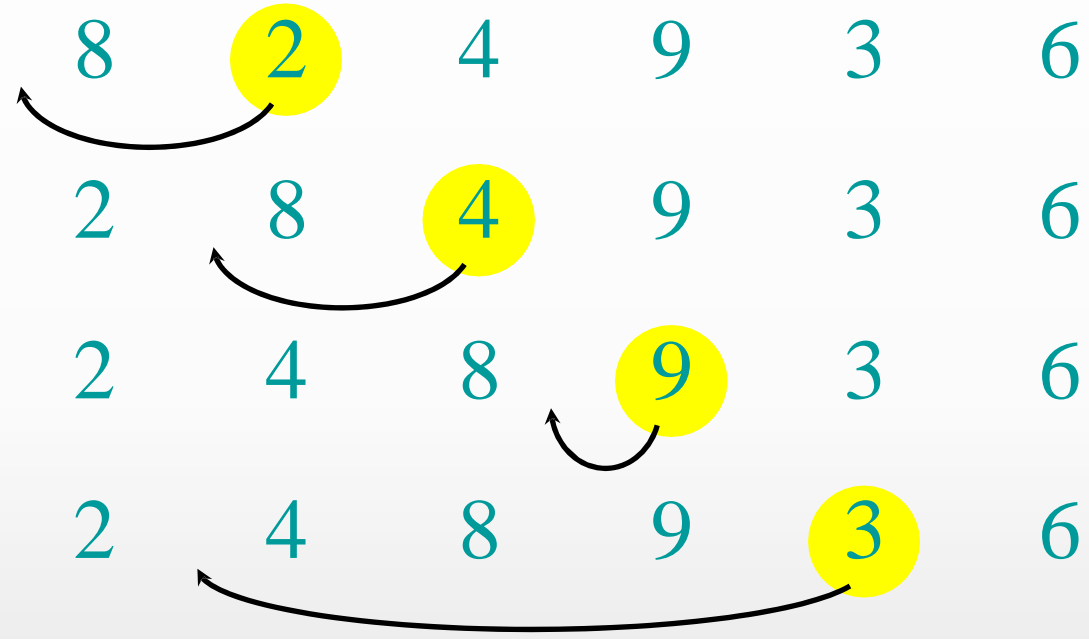
8 2 4 9 3 6
2 8 4 9 3 6

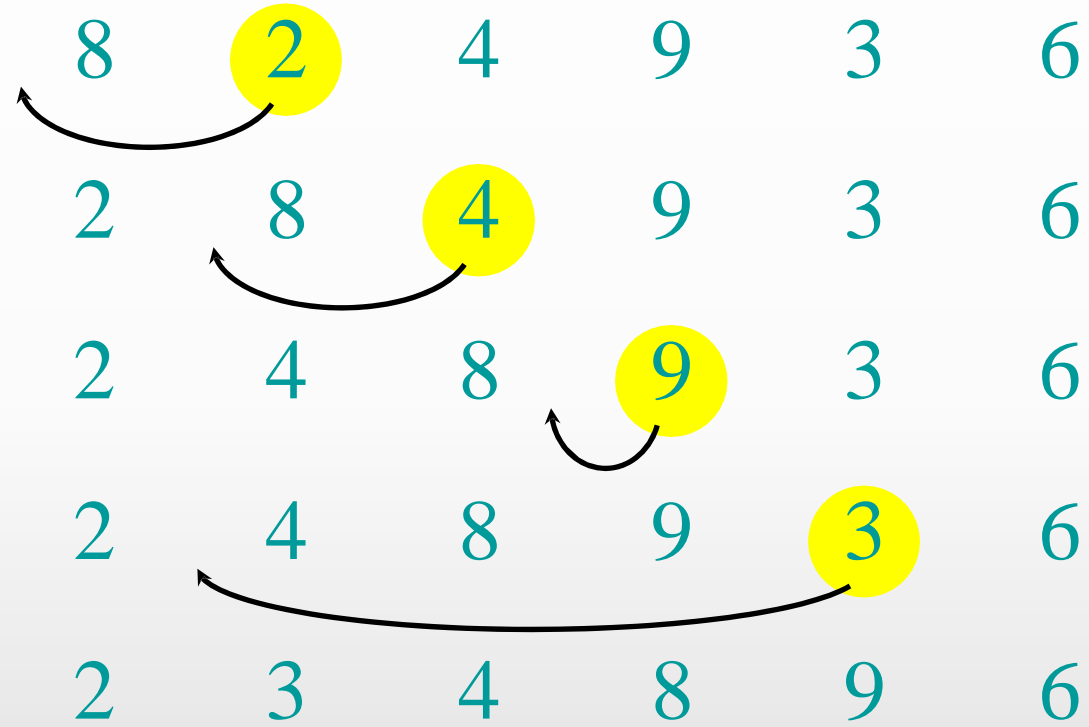


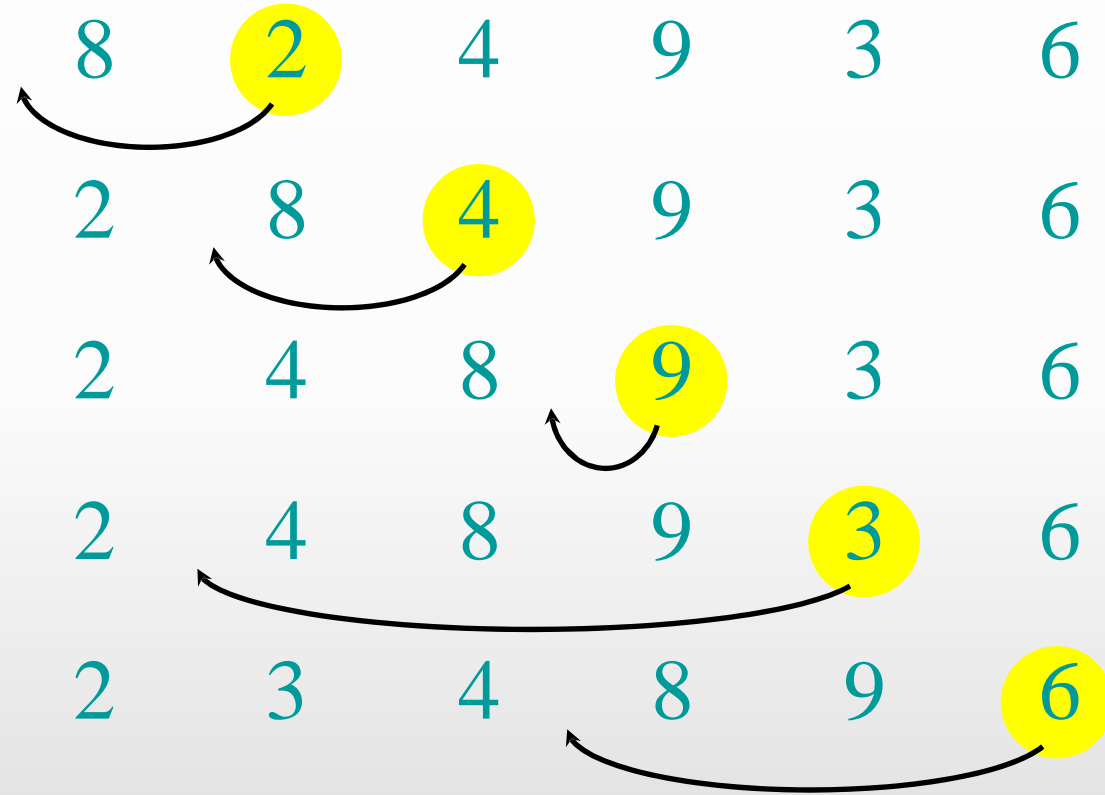


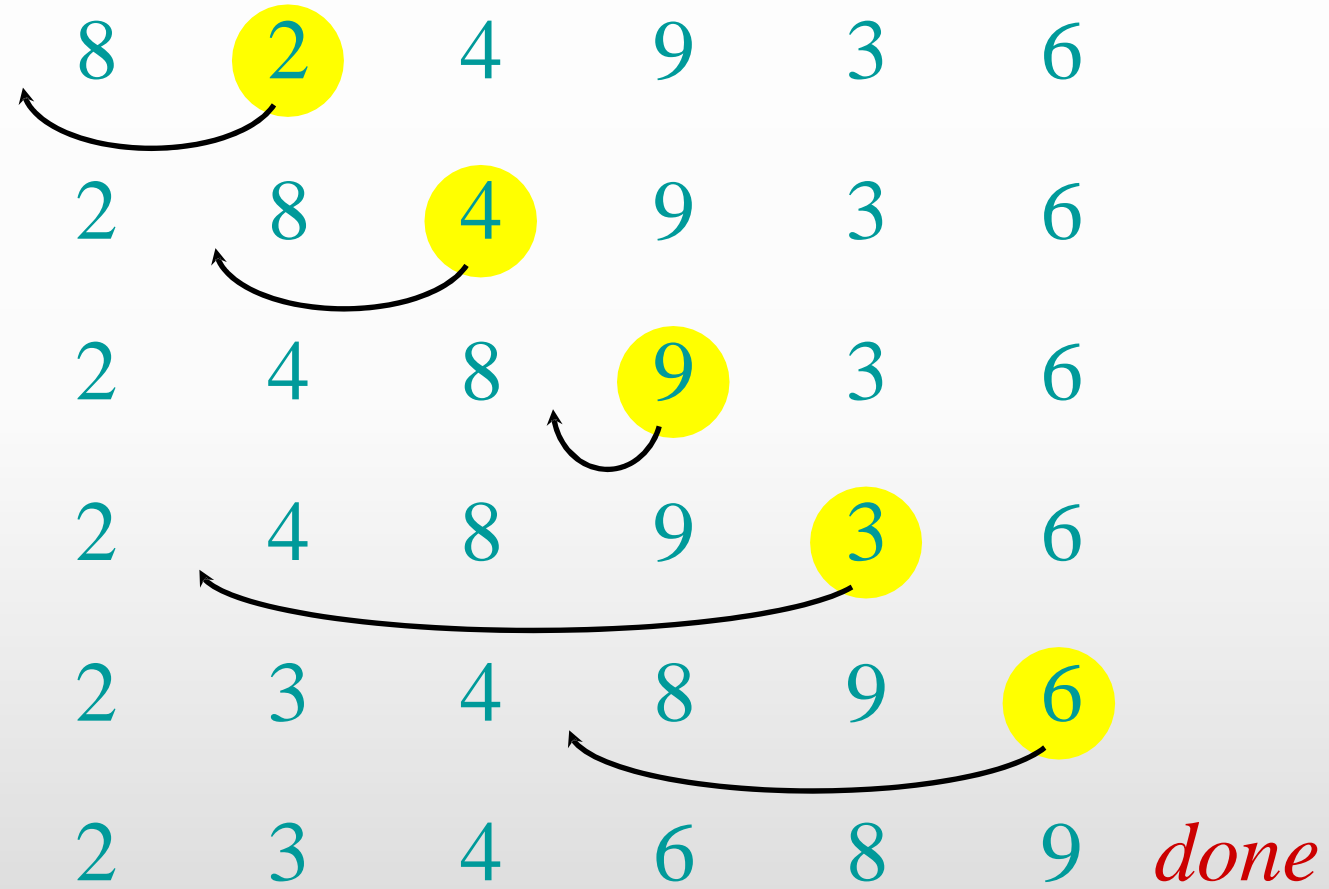
















Çalışma Süresi

- Girdi durumuna bağlı.
 - Dizi zaten sıralı ise, sıralamak kolay. 😊
- Girdi boyutuna bağlı.
 - Dizi az elemanlı ise, sıralamak kısa sürer. 😊
- Çalışma süresi olarak en kötü durum ele alınır.
 - Çünkü herkes garanti ister. 😊



Analiz Türleri

- **Kötü Durum:** (*usually*)
 - $T(n) = n$ boyutlu diziyi sıralamak için gereken maksimum süre
- **Orta Durum:** (*sometimes*)
 - $T(n) = n$ boyutlu diziyi sıralamak için beklenen süre
- **İyi Durum:** (*bogus*)
 - Özel veya hileli bir girdiyi sıralamak için geçen süre



Makineden Bağımsız Çalışma Süresi

- Araya sokarak sıralamanın en kötü durum zaman karmaşıklığı: $O(n^2)$
 - Girdi *tersten sıralı* ise gerçekleşir.
- Bilgisayarın hızına göre çalışma süresi değişebilir.
 - *Asimptotik analiz*, büyüme oranına odaklanır.
 - Makineye bağlı *sabit faktörler* göz ardı edilir.
- Analiz için $T(n)$ 'nin büyüme oranına bakılır.
 - Girdi boyutu n sonsuza gittikçe, ($n \rightarrow \infty$)
 - Yalnızca, en önemli (*üssü büyük olan*) terim değerlendirilir.
- $T(n) = 3n^2 + 2n + 5 = O(n^2)$



Θ -notation

- $\Theta(g(n)) = \{ f(n) : \text{tüm } n \geq n_0 \text{ değerleri için } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ şartını sağlayan } c_1, c_2, \text{ ve } n_0 \text{ pozitif katsayılar vardır.} \}$
- **Örnek:**
 - $T(n) = 3n^3 + 90n^2 - 5n + 6046$
 - Düşük kuvvetli terimleri sil.
 - $T(n) = 3n^3$
 - Katsayıları kaldır.
 - $T(n) = \Theta(n^3)$



Birleştirek Sıralama

MergeSort A[1 .. n]

$n = 1$ ise işlem tamam.

liste1 = MergeSort A[1 .. n/2]

liste2 = MergeSort A[n/2 +1 .. n]

2 sıralı listeyi birleştir. (*Merge*)



Birleştirek Sıralama

MergeSort A[1 .. n] $T(n)$

$n = 1$ ise işlem tamam.

liste1 = MergeSort A[1 .. n/2]

liste2 = MergeSort A[n/2 +1 .. n]

2 sıralı listeyi birleştir. (*Merge*)



Birleştirek Sıralama

MergeSort A[1 .. n] $T(n)$
n = 1 ise işlem tamam. $\Theta(1)$
liste1 = MergeSort A[1 .. n/2]
liste2 = MergeSort A[n/2 +1 .. n]
2 sıralı listeyi birleştir. (*Merge*)



Birleştirek Sıralama

MergeSort A[1 .. n] $T(n)$
n = 1 ise işlem tamam. $\Theta(1)$
liste1 = MergeSort A[1 .. n/2] $T(n/2)$
liste2 = MergeSort A[n/2 +1 .. n]
2 sıralı listeyi birleştir. (*Merge*)



Birleştirek Sıralama

MergeSort A[1 .. n] $T(n)$
n = 1 ise işlem tamam. $\Theta(1)$
liste1 = MergeSort A[1 .. n/2] $T(n/2)$
liste2 = MergeSort A[n/2 +1 .. n] $T(n/2)$
2 sıralı listeyi birleştir. (*Merge*)



Birleştirek Sıralama

MergeSort A[1 .. n] $T(n)$

 n = 1 ise işlem tamam. $\Theta(1)$

 liste1 = MergeSort A[1 .. n/2] $T(n/2)$

 liste2 = MergeSort A[n/2 +1 .. n] $T(n/2)$

 2 sıralı listeyi birleştir. (*Merge*) $\Theta(n)$



Birleřtirme İřlemi (Merge)

20 12

13 11

7 9

2 1

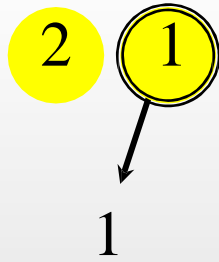


Birleřtirme İřlemi (Merge)

20 12

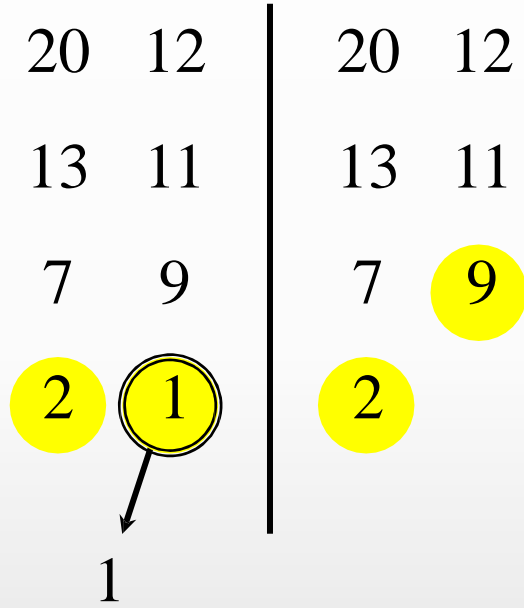
13 11

7 9



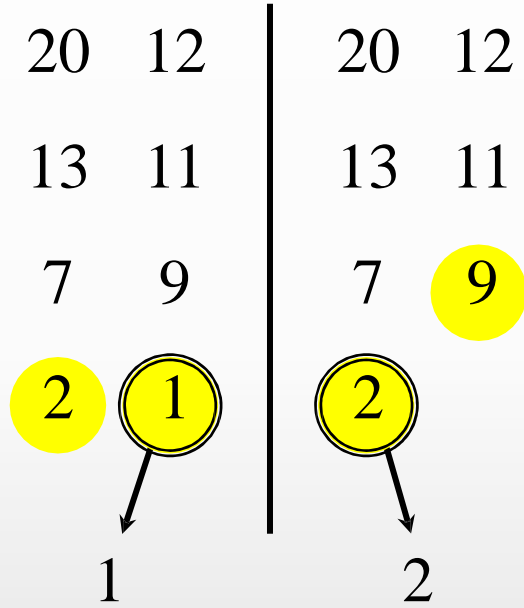


Birleřtirme İřlemi (Merge)



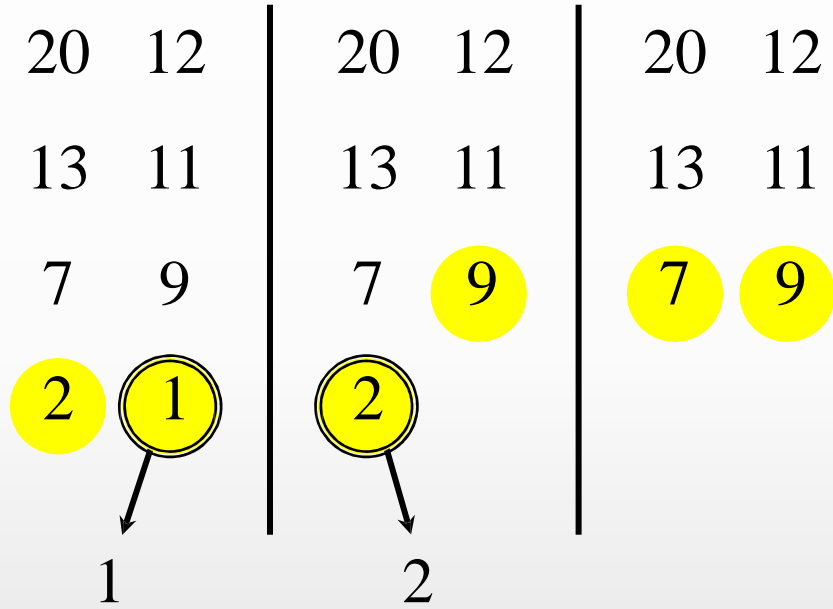


Birleřtirme İřlemi (Merge)



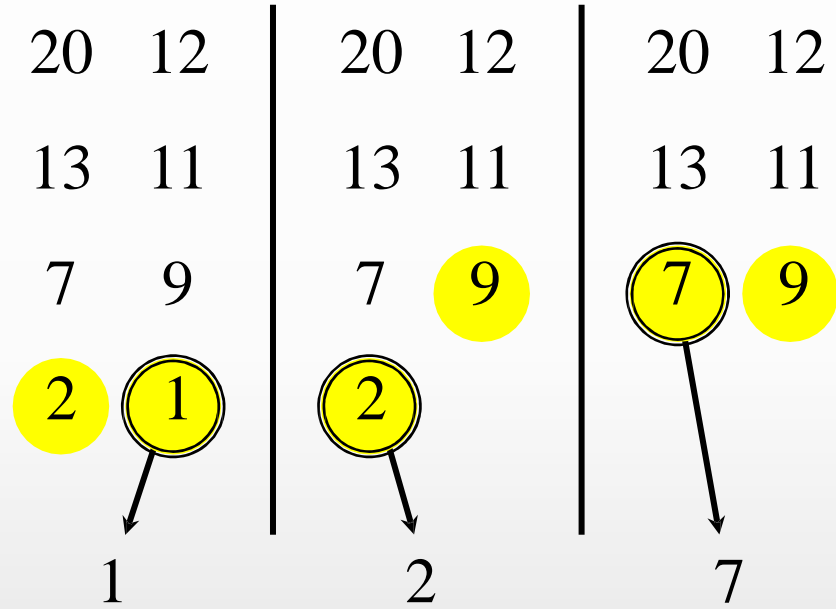


Birleřtirme İřlemi (Merge)



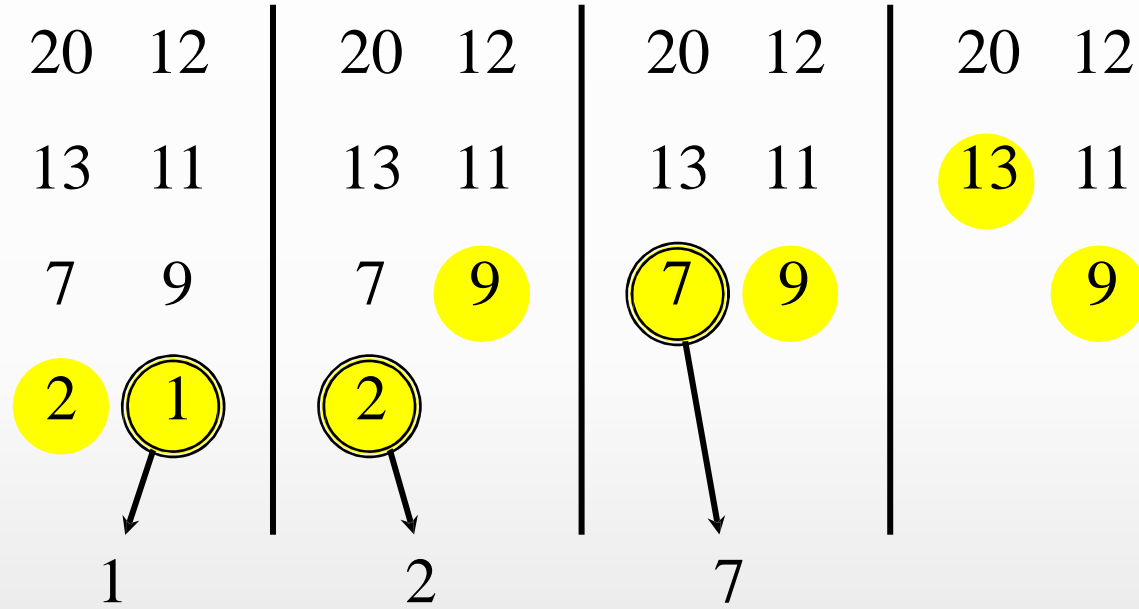


Birleřtirme İřlemi (Merge)



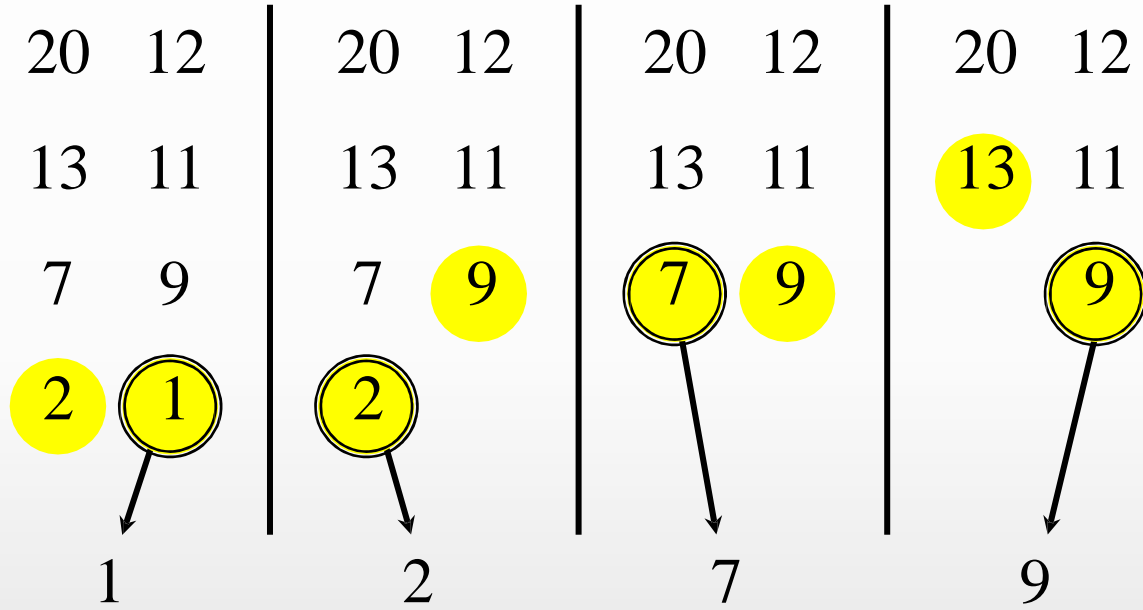


Birleřtirme İřlemi (Merge)



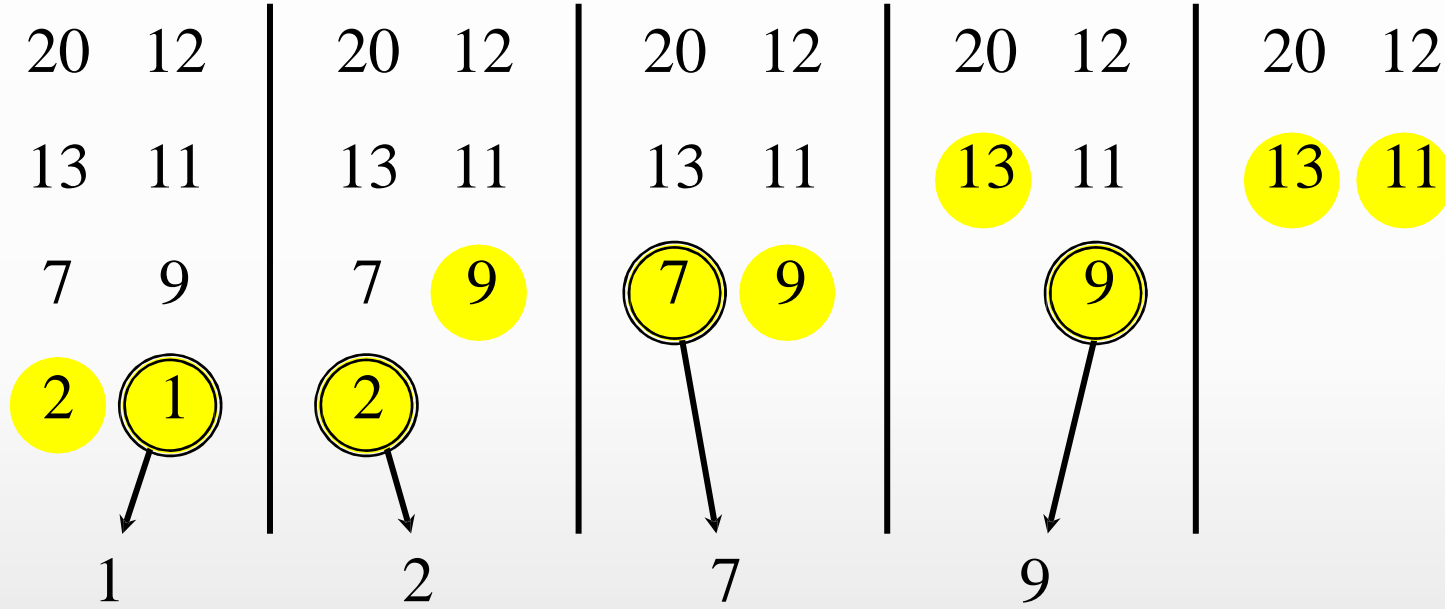


Birleřtirme İřlemi (Merge)



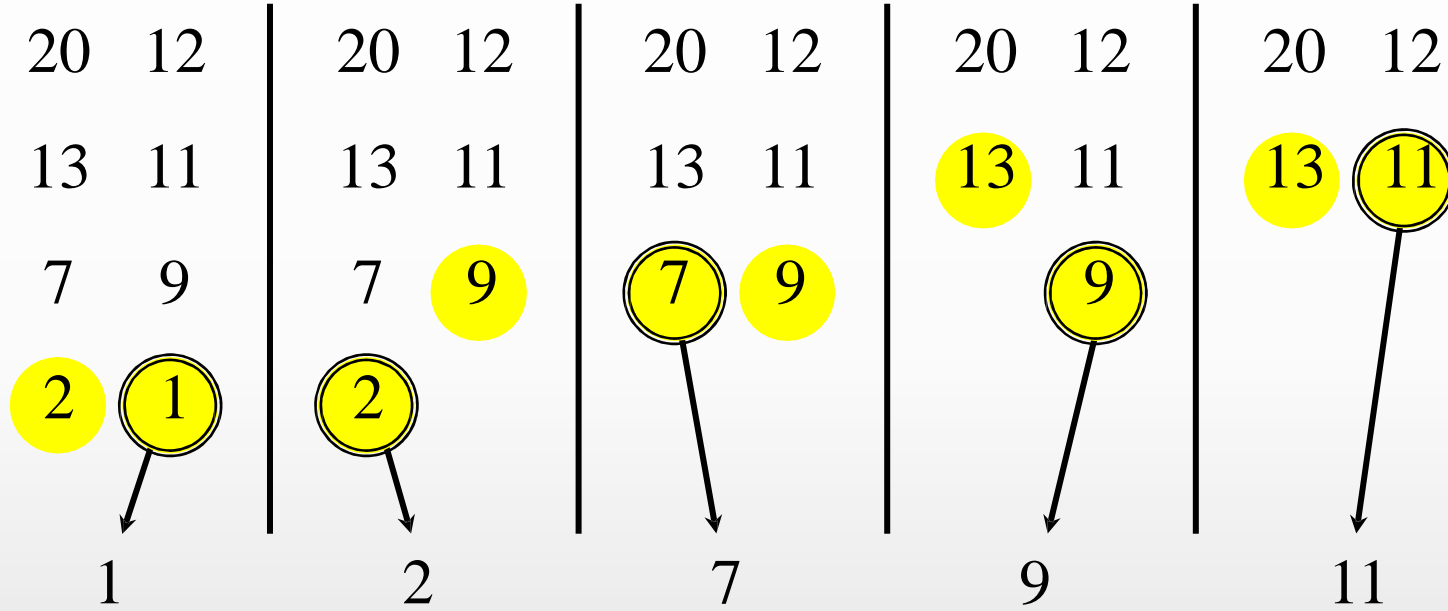


Birleřtirme İřlemi (Merge)



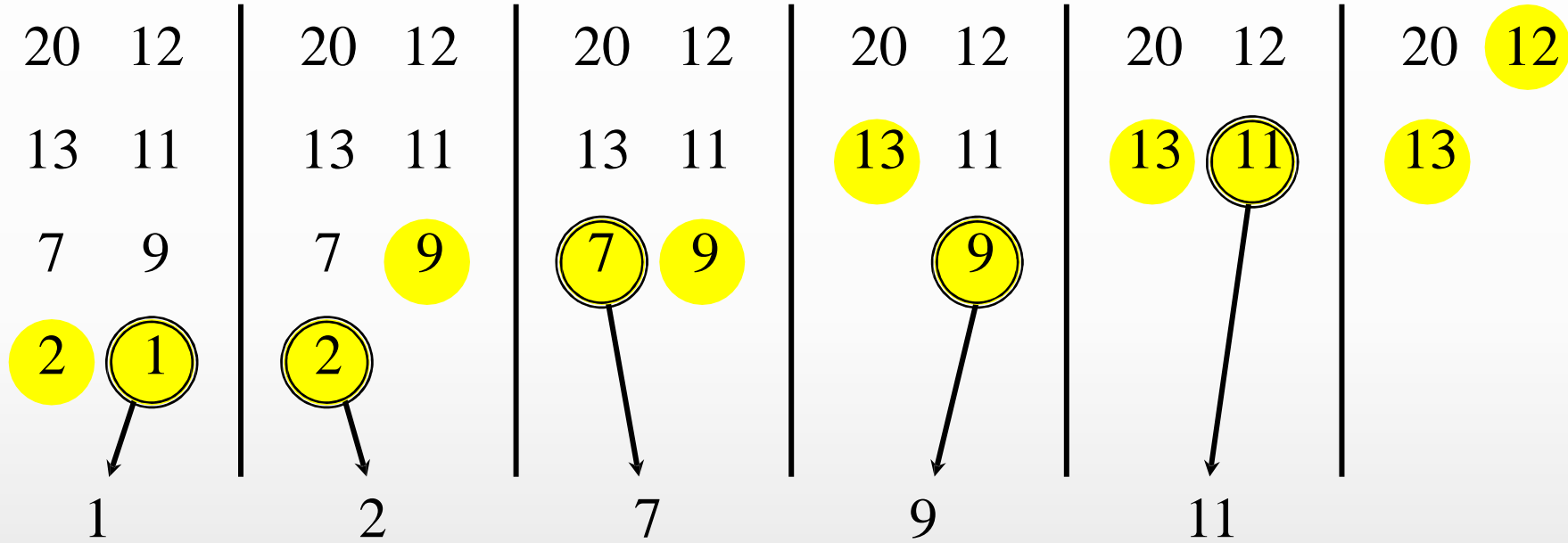


Birleřtirme İřlemi (Merge)



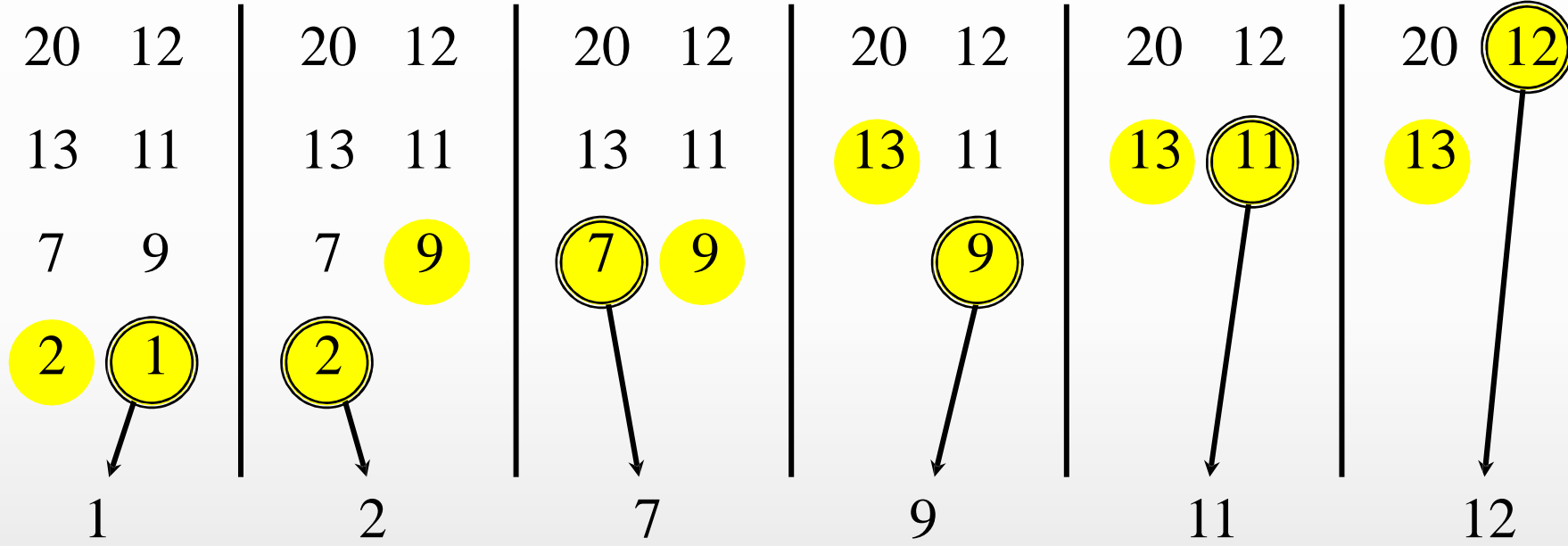


Birleřtirme İřlemi (Merge)



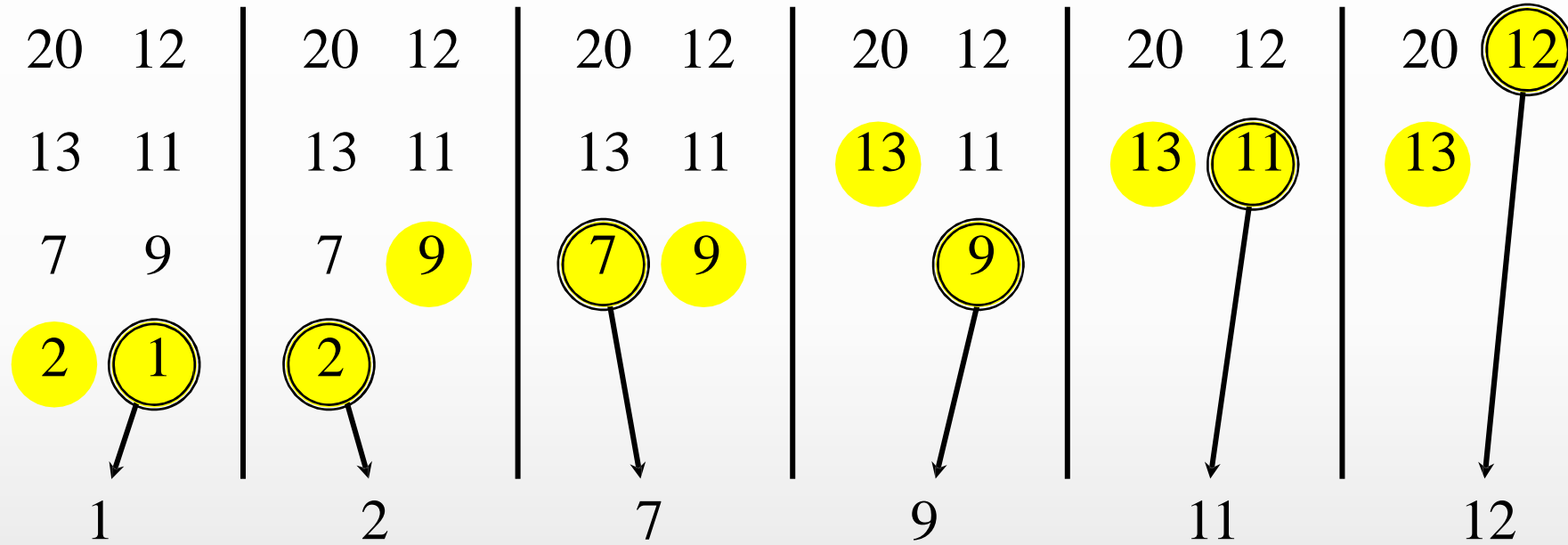


Birleřtirme İřlemi (Merge)





Birleştirme İşlemi (Merge)



- n elemanlı listeyi birleştirmek için $T(n) = \Theta(n)$.



İkili Arama

- *Binary Search*
- Sıralı listede bir elemanı arar.
 - Böl (*Divide*): orta elemanı kontrol et.
 - Fethet (*Conquer*): seçilen alt diziyi özyinelemeli olarak ara.
 - Birleştir (*Combine*): önemsiz (*trivial*).



İkili Arama

- **Örnek:** 9'u bul.

3 5 7 8 9 12 15



İkili Arama

- Örnek: 9'u bul.

3 5 7 8 9 12 15

3 5 7 8 9 12 15



İkili Arama

- Örnek: 9'u bul.

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15



İkili Arama

- Örnek: 9'u bul.

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15



İkili Arama

- Örnek: 9'u bul.

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15



İkili Arama

- Örnek: 9'u bul.

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15

3 5 7 8 9 12 15



İkili Arama

- *Binary Search*

- Sıralı listede bir elemanı arar.

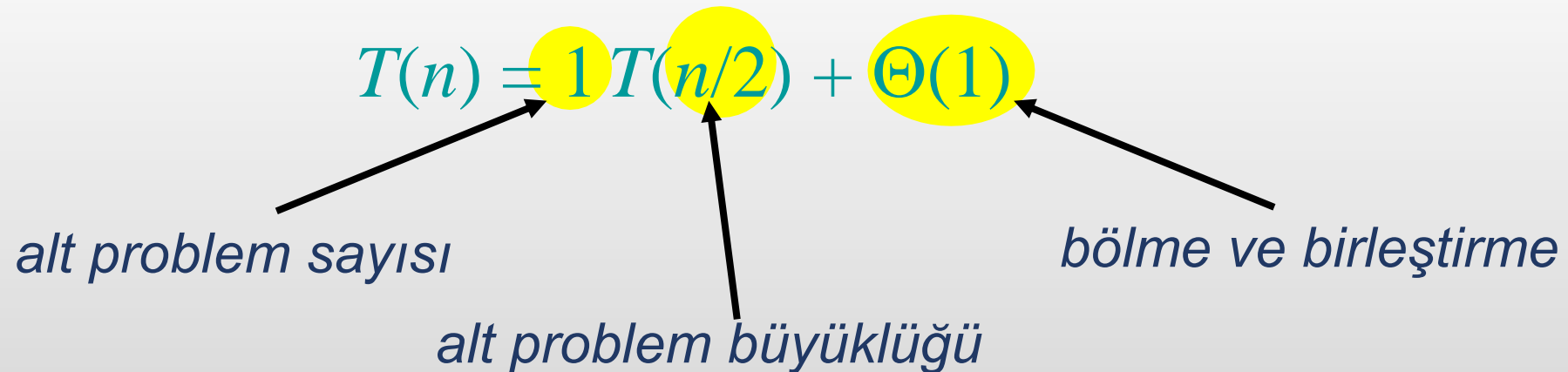
- Böl (*Divide*): orta elemanı kontrol et.
- Fethet (*Conquer*): seçilen alt diziyi özyinelemeli olarak ara.
- Birleştir (*Combine*): önemsiz (*trivial*).

$T(n)$

$\Theta(1)$

$T(n/2)$

$\Theta(1)$





Matris Çarpımı

$$\left. \begin{array}{l} \text{Girdi: } A = [a_{ij}], B = [b_{ij}]. \\ \text{Çıktı: } C = [c_{ij}] = A \cdot B. \end{array} \right\} i, j = 1, 2, \dots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \text{L} & c_{1n} \\ c_{21} & c_{22} & \text{L} & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \text{L} & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \text{L} & a_{1n} \\ a_{21} & a_{22} & \text{L} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \text{L} & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \text{L} & b_{1n} \\ b_{21} & b_{22} & \text{L} & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \text{L} & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$



Matris Çarpımı

```
for  $i = 1$  to  $n$   
  do for  $j = 1$  to  $n$   
    do  $c_{ij} = 0$   
      for  $k = 1$  to  $n$   
        do  $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
```



Matris Çarpımı

```
for  $i = 1$  to  $n$ 
  do for  $j = 1$  to  $n$ 
    do  $c_{ij} = 0$ 
      for  $k = 1$  to  $n$ 
        do  $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
```

Çalışma süresi $T(n) = \Theta(n^3)$



Bir Sayının Üssünü Alma

- **Problem:** a^n 'i hesapla ($n \in \mathbb{N}$).
- Doğal algoritma:
 - $a^n = a \times a \times a \dots$
 - $\Theta(n)$.
- Böl ve Fethet algoritması:

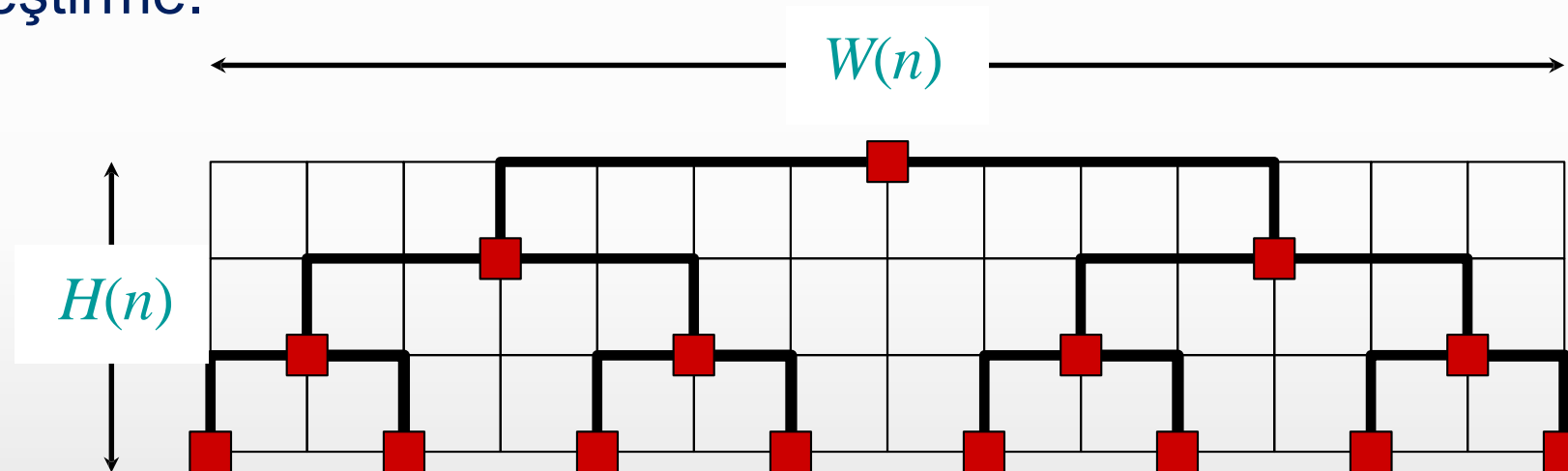
$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & n \text{ çift ise;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & n \text{ tek ise.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \quad \Rightarrow \quad T(n) = \Theta(\lg n) .$$



VLSI (Very Large Scale Integration) Yerleşimi

- **Problem:** n yapraklı ikili tam ağacı ızgara düzeninde minimum alana yerleştirme.

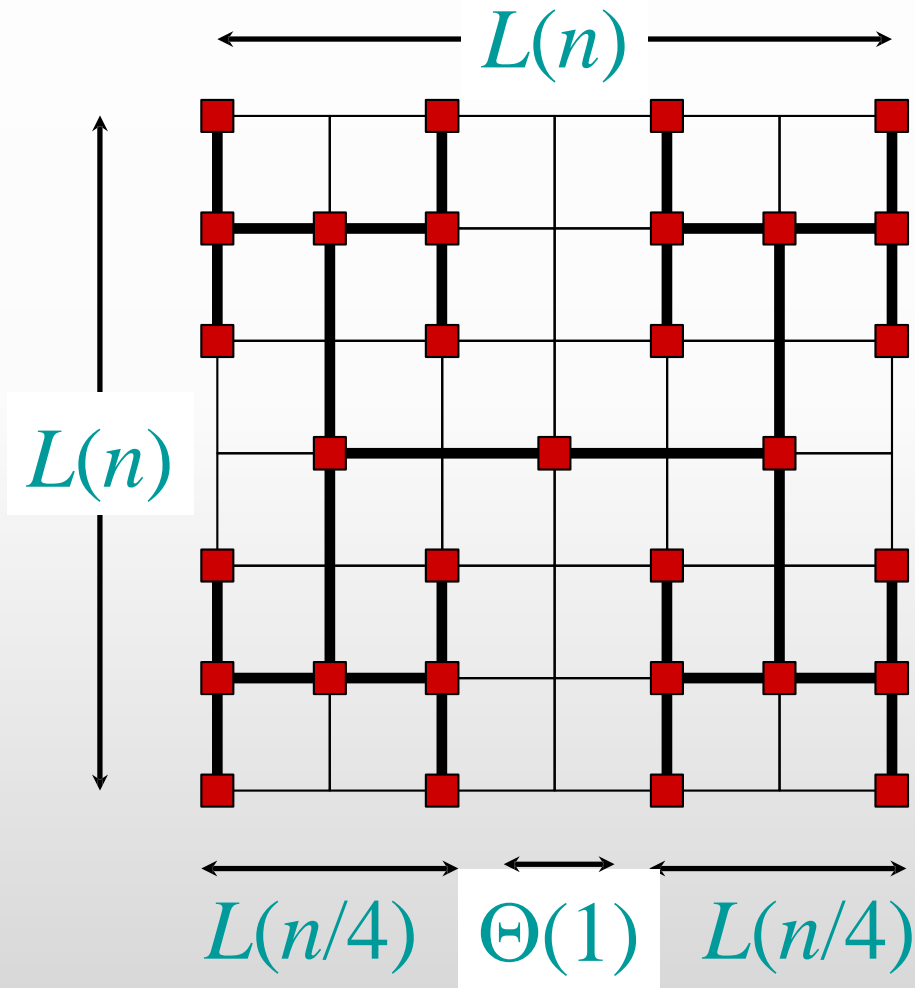


$$\begin{aligned} H(n) &= H(n/2) + \Theta(1) & W(n) &= 2W(n/2) + \Theta(1) \\ &= \Theta(\lg n) & &= \Theta(n) \end{aligned}$$

$$\text{Alan} = \Theta(n \lg n)$$



H Ağacı Gömme (Embedding)



$$\begin{aligned}L(n) &= 2L(n/4) + \Theta(1) \\ &= \Theta(\sqrt{n})\end{aligned}$$

$$\text{Alan} = \Theta(n)$$





Euclid Algoritması

- En büyük ortak bölen (EBOB) bulmaya yarar.
- En eski ve en basit EBOB bulma algoritması.
- Verimli ve hızlı bir şekilde çalışır.



Euclid Algoritması

- Adım 1: Verilen iki sayıdan büyük olanı diğerine böler ve kalanı alır.
- Adım 2: Kalanı sıfır yapana kadar bu işlemi tekrarlar.
- Adım 3: Kalan sıfır olduğunda, bölen sayı en büyük ortak bölen olur.

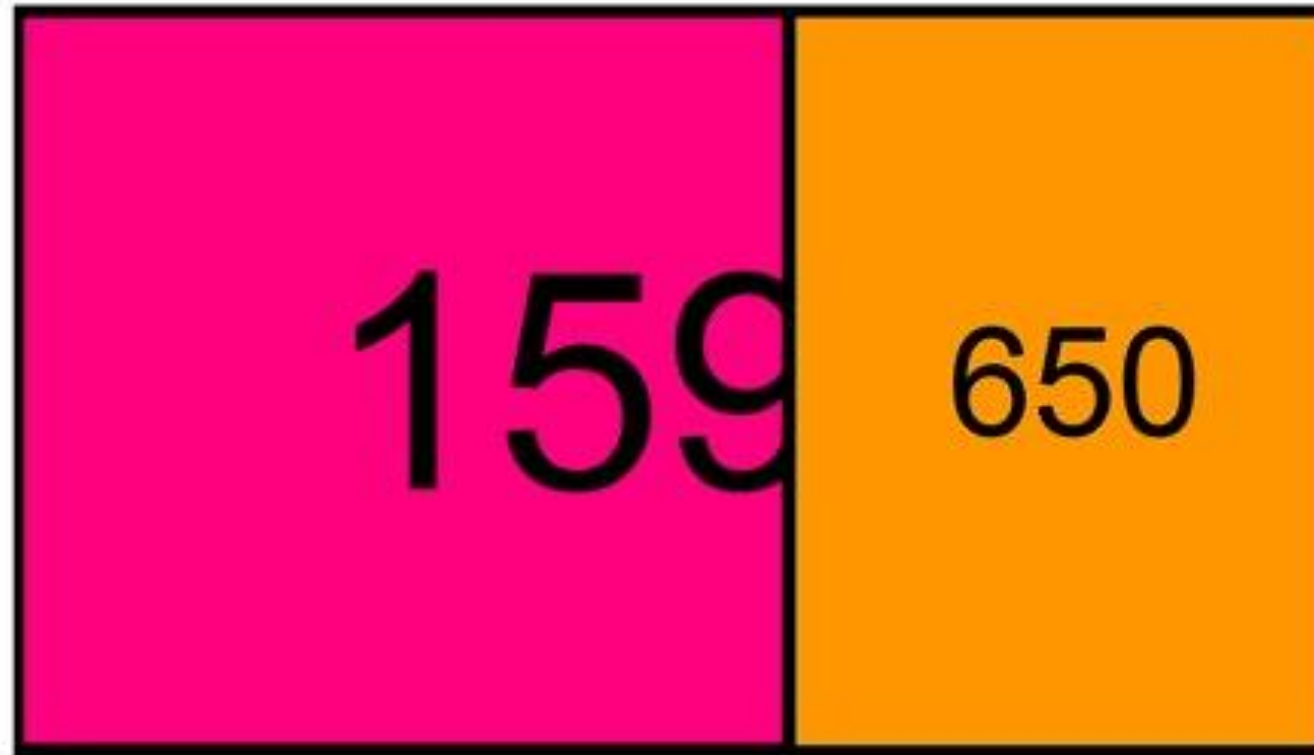


Euclids Algorithm

1599

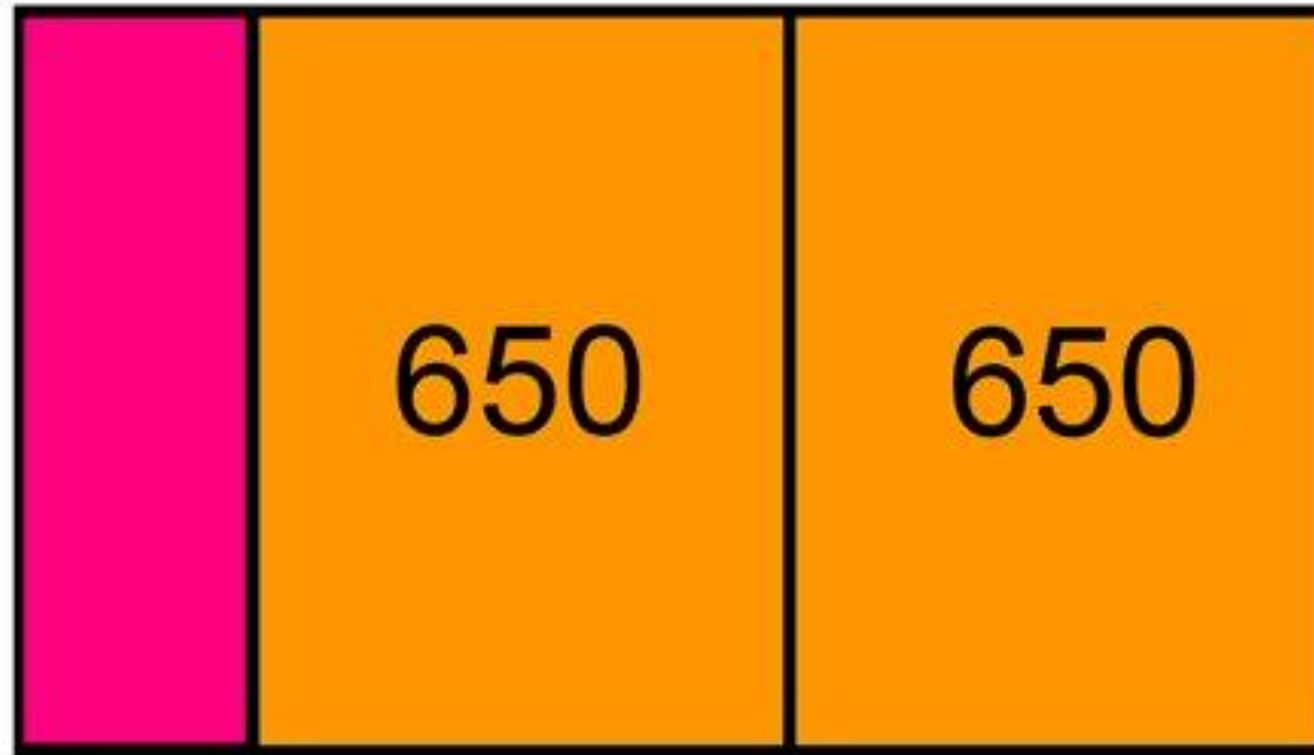


Euclids Algorithm



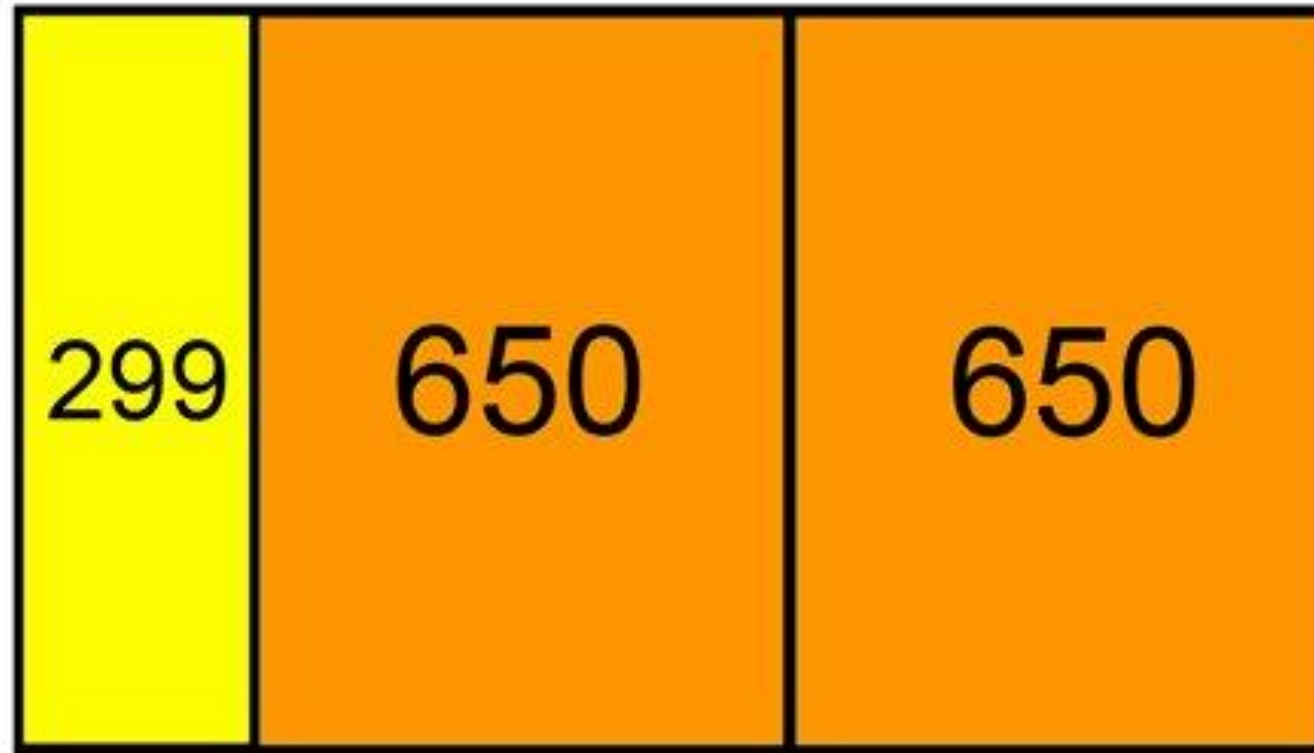


Euclids Algorithm



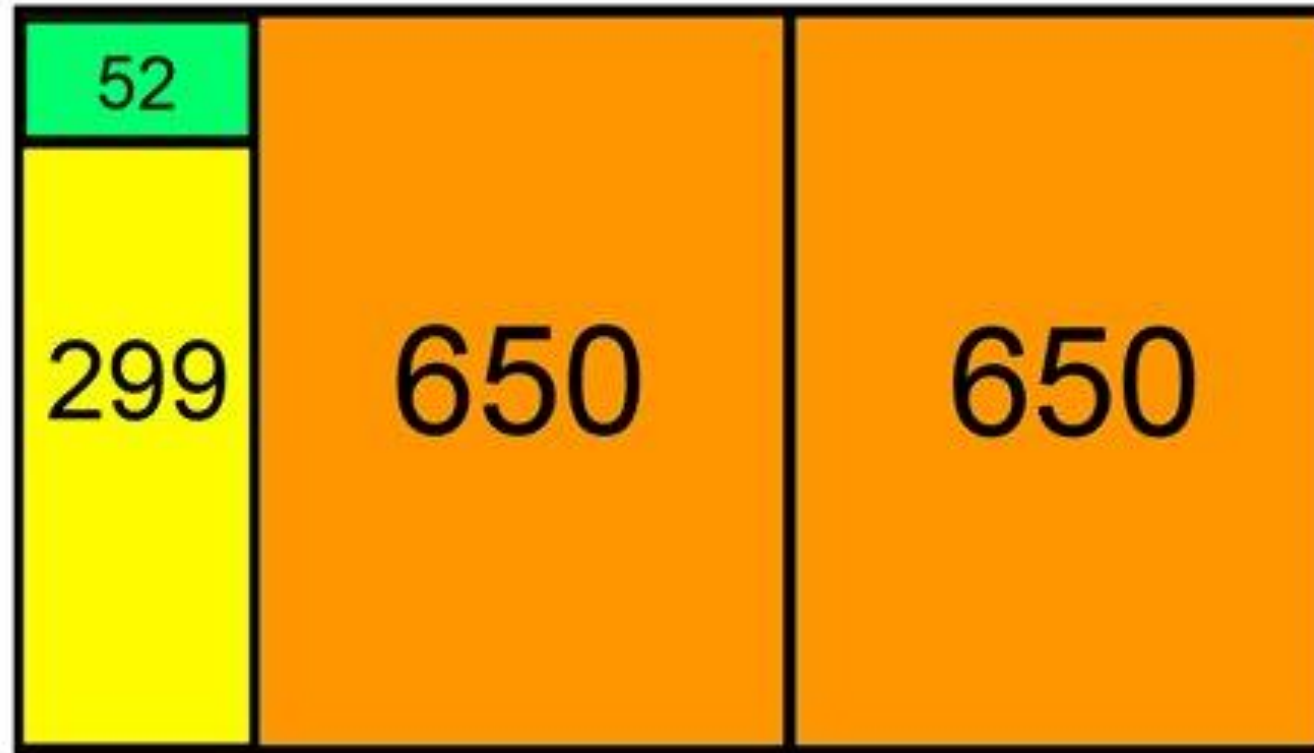


Euclids Algorithm



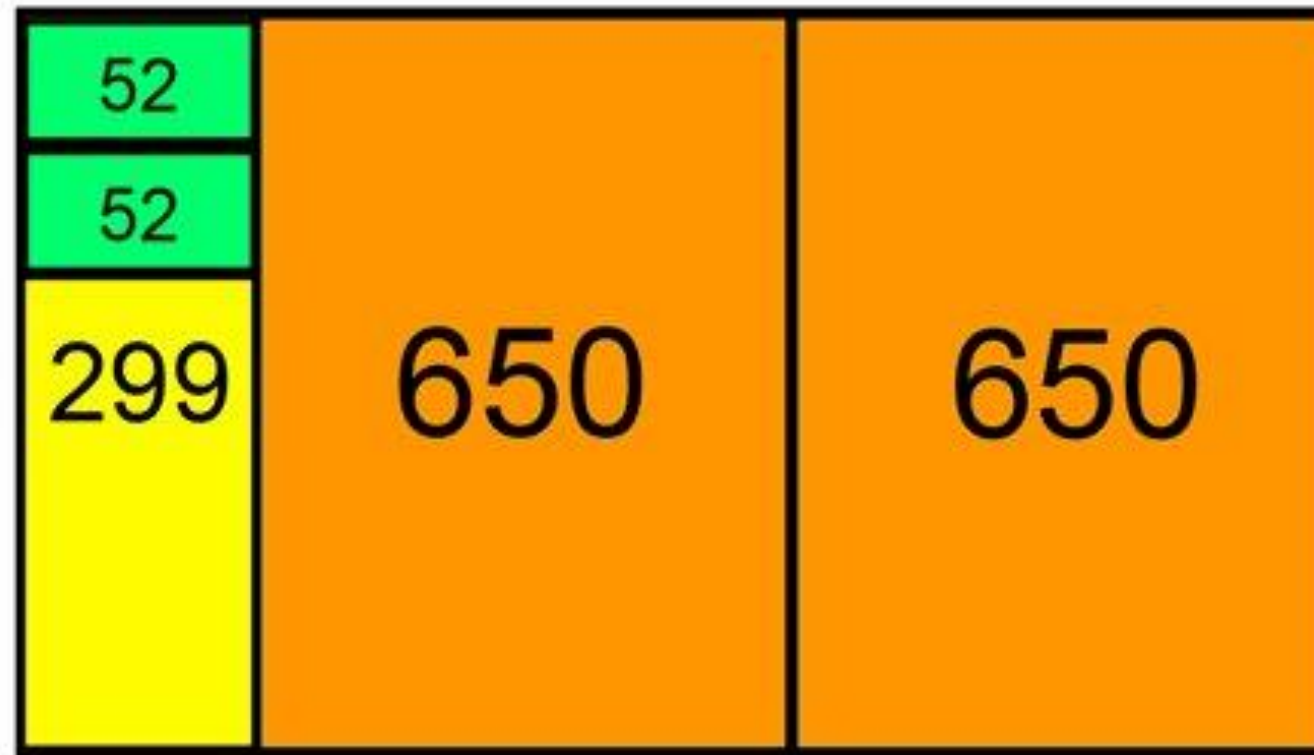


Euclids Algorithm



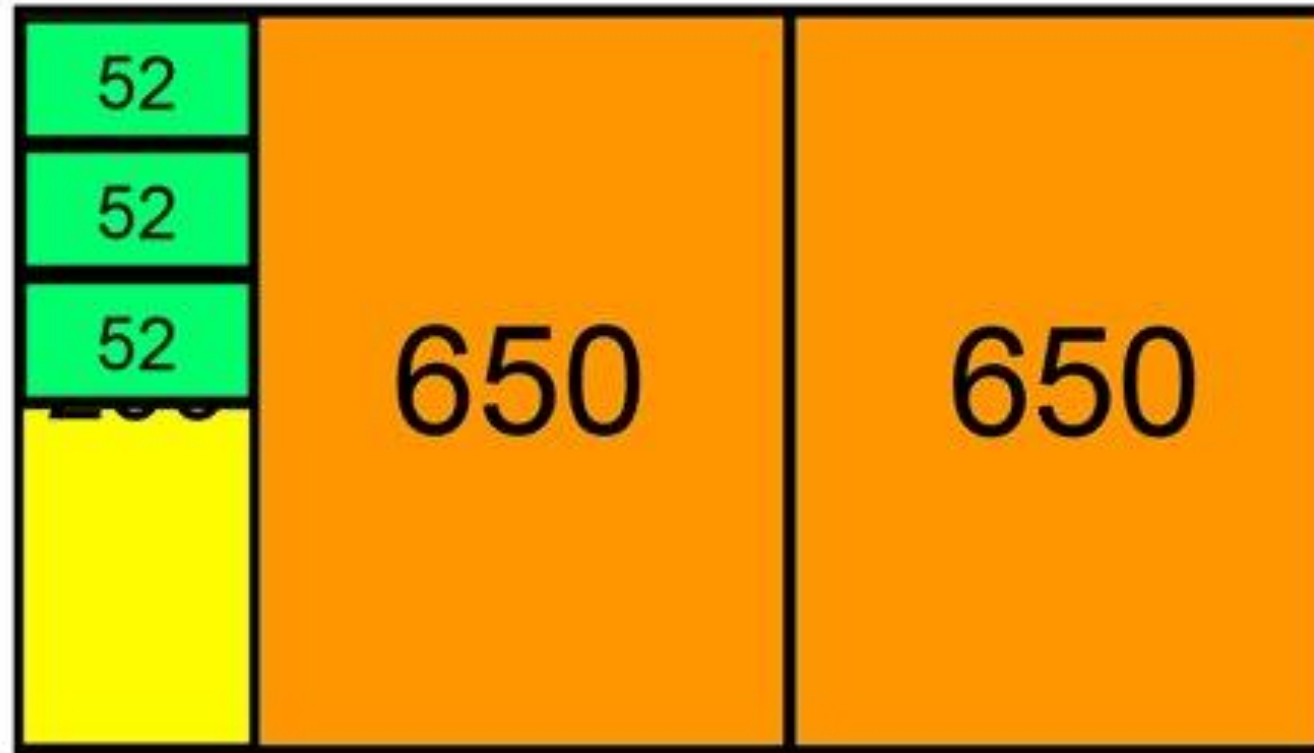


Euclids Algorithm



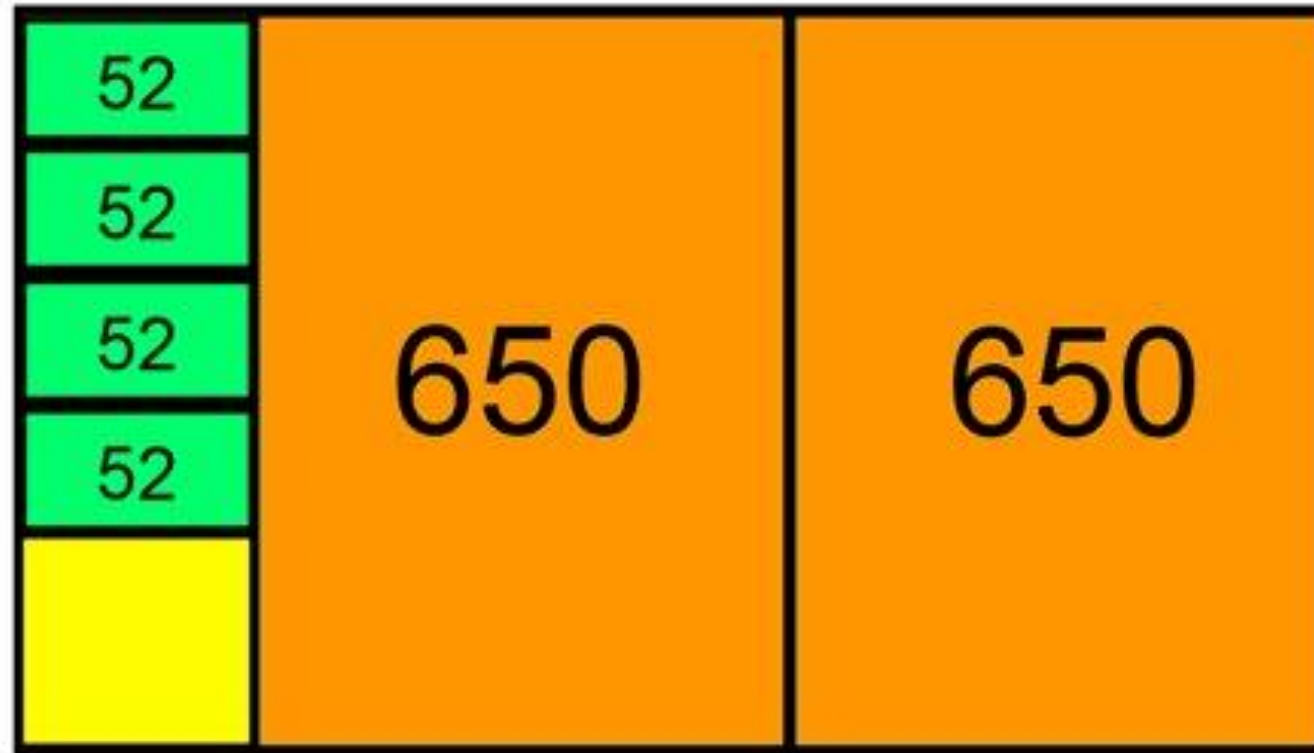


Euclids Algorithm



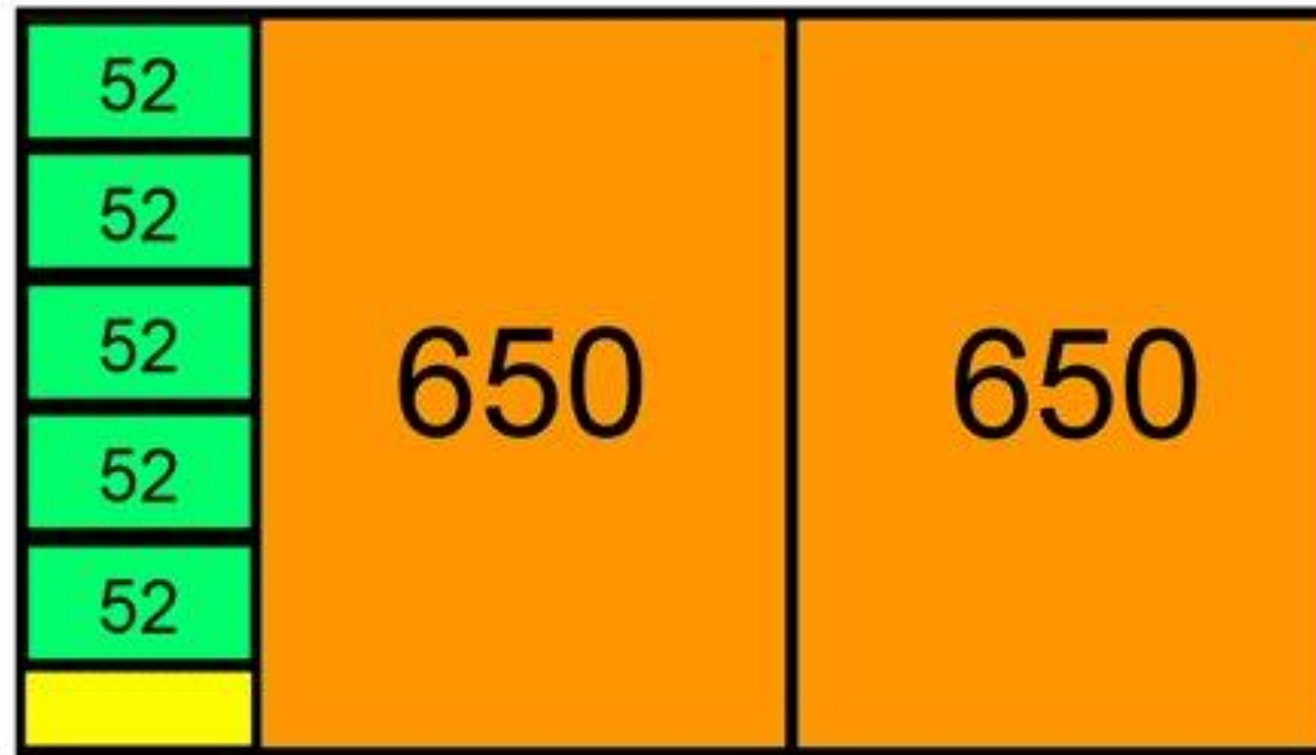


Euclids Algorithm



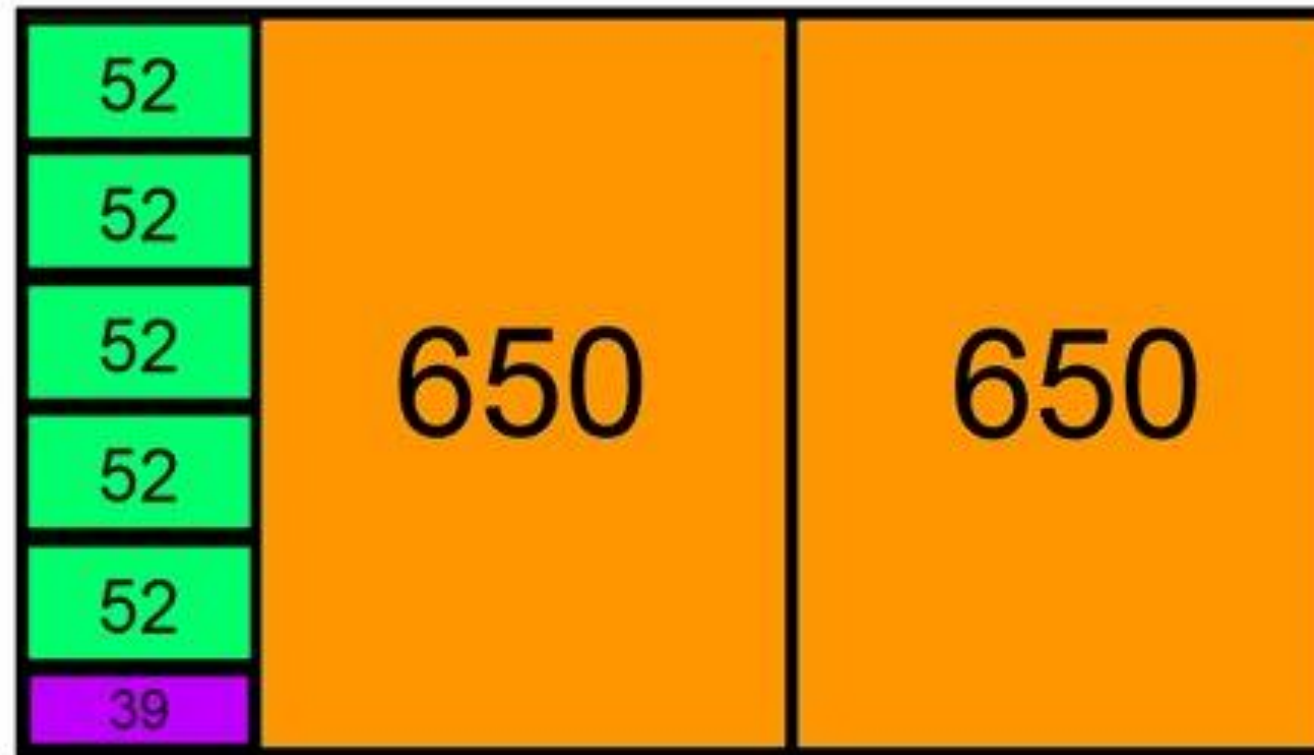


Euclids Algorithm



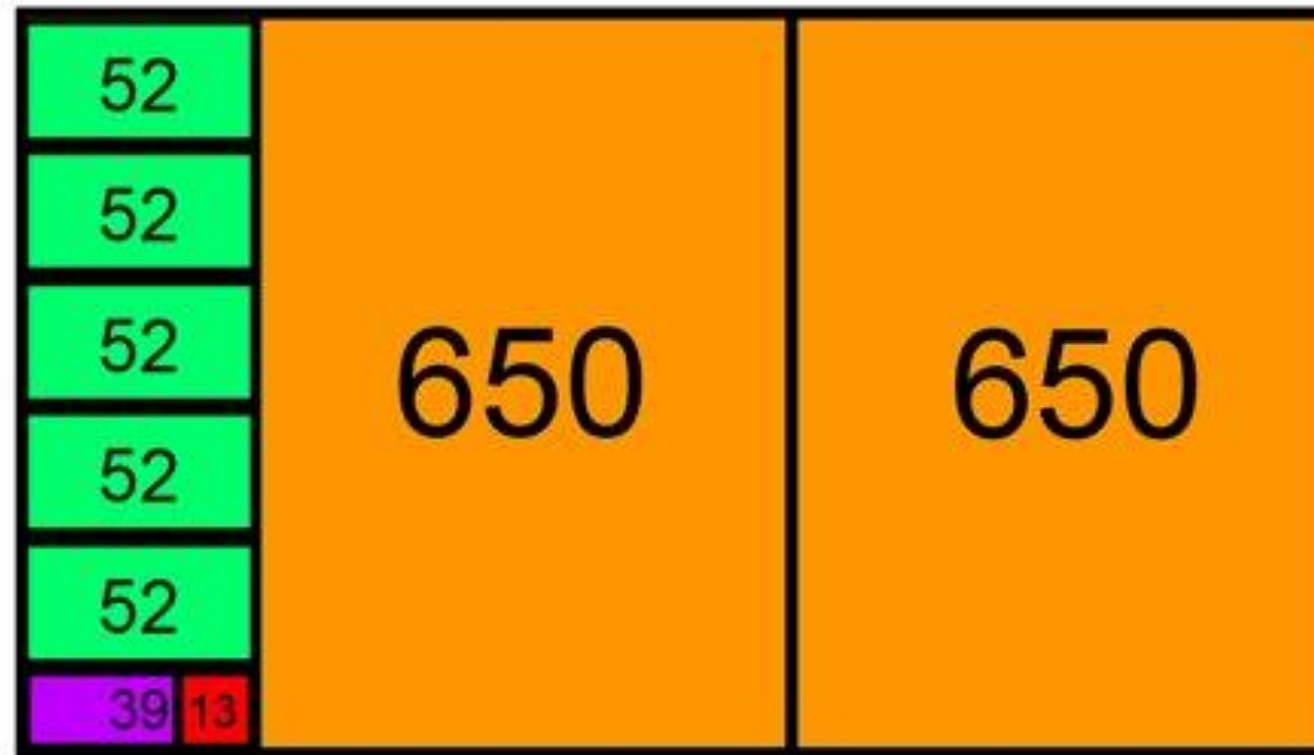


Euclids Algorithm



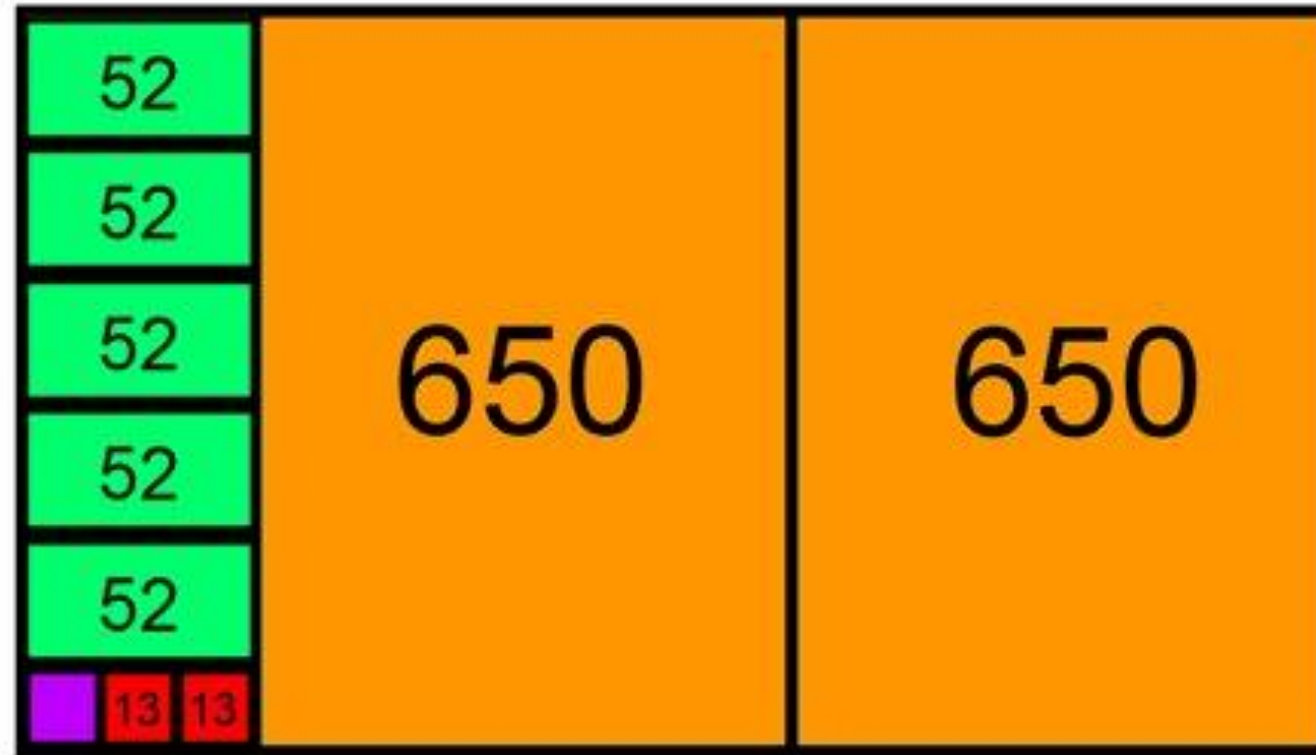


Euclids Algorithm



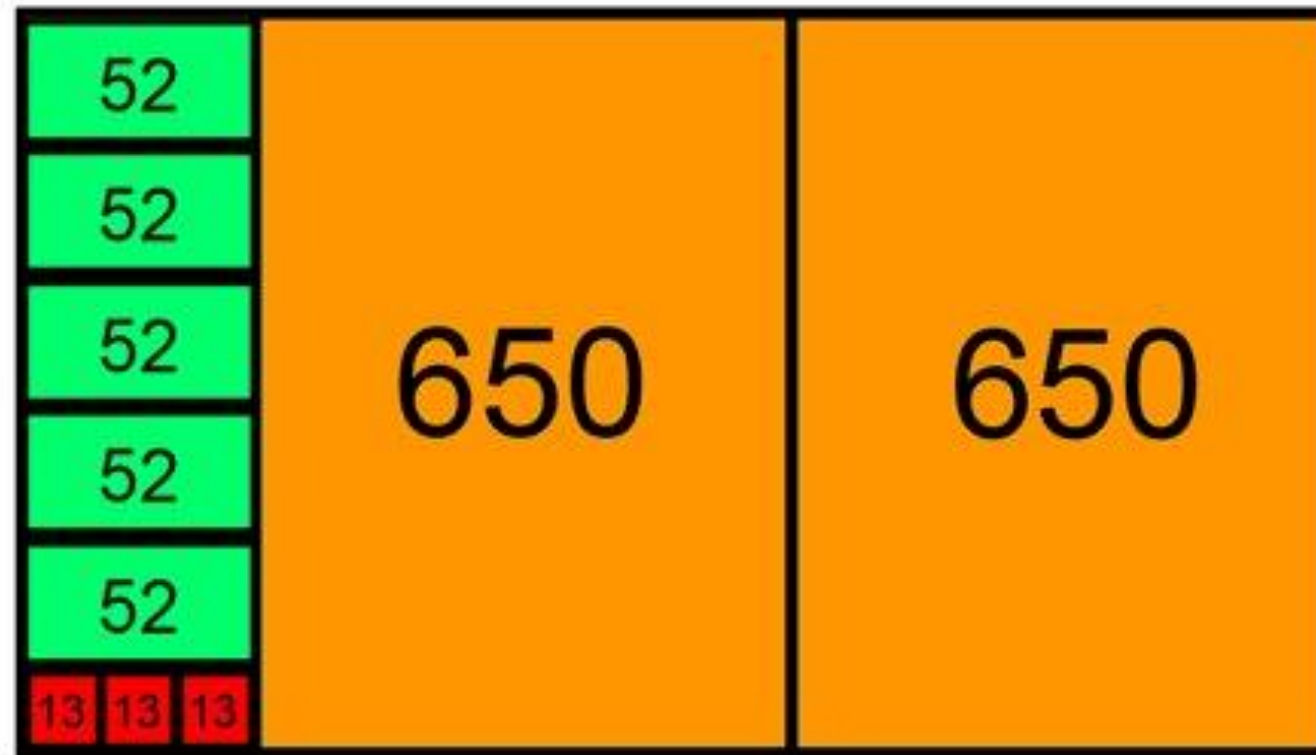


Euclids Algorithm





Euclids Algorithm





SON