BM 210 Algoritma Analizi (Analysis of Algorithms)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Divide and Conquer

- *Divide-and-conquer* tekniğiyle algoritma tasarımı. Örnek problemler:
  - Searching (*binary search*)
  - Sorting (*merge sort*)

Divide and Conquer

- *Divide-and-conquer* metoduyla algoritma tasarımı:
  - Problem kendisine benzer küçük boyutlu alt problemlere bölünür. Alt problemler çözülür ve bulunan çözümler birleştirilir.
    - **Divide**: Problem iki veya daha fazla alt probleme bölünür
    - **Conquer**: divide-and-conquer özyinelemeli (recursively) olarak kullanılarak alt problemleri çözer
    - **Combine**: Alt problemlerin çözümleri alınır ve orijinal problemin çözümü olacak şekilde birleştirilir

Binary Search

- Sıralı bir dizide bir sayının bulunması:
  - Bir eleman olup olmadığı kontrol edilir
  - Birden fazla eleman varsa iki eşit parçaya bölünür ve herbirisi ayrı çözülür
  - Sonuçlar birleştirilir

INPUT: A[1..n] – sıralı (azalmayan) integer sayı dizisi, *s* – aranan integer sayı.
OUTPUT: *j* bulunan sayının indeksi A[*j*] = *s*. *NIL*, if $\forall j\ (1 \le j \le n)$: A[*j*] ≠ *s*

```
Binary-search(A, p, r, s):
   if p = r then
      if A[p] = s then return p
      else return NIL
   q←⌊(p+r)/2⌋
   ret ← Binary-search(A, p, q, s)
   if ret = NIL then
      return Binary-search(A, q+1, r, s)
   else return ret
```

Recurrences

- Özyinelemeli algoritmalarda çalışma süresi recurrences'lar kullanılarak tanımlanabilir
- Bir **recurrence** herhangi bir fonksiyonu kendisinin küçük girişleriyle tanımlar

T(n)=
- **denemeyle çözülür, n=1**
- **Alt problem sayısı * T(n/alt problem boyutu)+bölme+birleşim, n=1**

- Örnek $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Binary Search (gelişmiş)

Daha iyi bir **conquer** metodu:

- Dizinin yarısı çözülür (aranır)

INPUT: A[1..n] – sıralı (azalmayan) integer dizisi, *s – aranan* integer sayı.
OUTPUT: *j* bulunan sayının indeksi A[*j*] = *s*. *NIL*, if $\forall j$ ($1 \le j \le n$): A[*j*] ≠ *s*

```
Binary-search(A, p, r, s):
   if p = r then
      if A[p] = s then return p
      else return NIL
   q←⌊(p+r)/2⌋
   if A[q] ≤ s then return Binary-search(A, p, q, s)
   else return Binary-search(A, q+1, r, s)
```

Running Time of BS

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n/2) + \Theta(1) & \text{if } n > 1 \end{cases}$$

- T(n) = Θ(lg n) !

Merge Sort Algoritması

- **Divide**: Eğer S en az iki elemana sahipse (S sıfır veya bir elemana sahipse hiçbir işlem yapılmaz), bütün elemanlar S'den alınır ve S_1 ve S_2 adlı iki alana yerleştirilir, herbiri S dizisinin yarısına sahiptir. (örn. S_1 ilk $\lceil n/2 \rceil$ elemana ve S_2 ise ikinci $\lfloor n/2 \rfloor$ elemana sahiptir).
- **Conquer**: S_1 ve S_2 Merge Sort kullanılarak sıralanır.
- **Combine**: S_1 and S_2 içindeki sıralı elemanlar tekrar S içerisine tek bir sıralı dizi oluşturacak şekilde aktarılır.

Merge Sort Algoritması

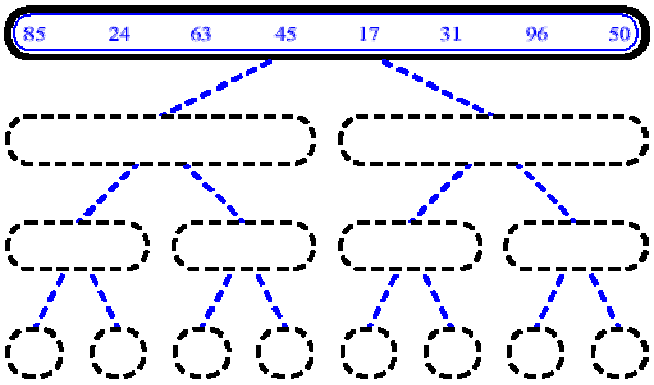
```
Merge-Sort(A, p, r)
   if p < r then
      q←⌊(p+r)/2⌋
      Merge-Sort(A, p, q)
      Merge-Sort(A, q+1, r)
      Merge(A, p, q, r)
```

Merge(A, p, q, r)
A[p..q] ve A[q+1..r] sıralı elemanlarından ilk iki eleman karşılaştırılır ve sonuç dizisine küçük olan aktarılır. Bu işlem dizilerin ikiside boş oluncaya kadar tekrar edilir. Sonuçta elde edilen sıralanmış elemanlar *A[p..r]* dizisine aktarılır.

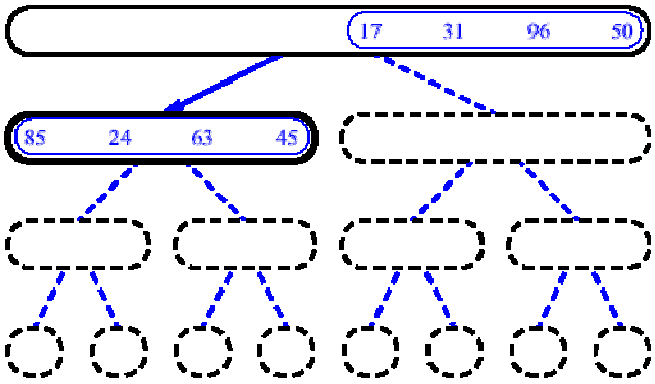
Merge Sort Algoritması

```
Merge(A, p, q, r)
  n1 ← q - p + 1
  n2 ← r - q
  create arrays L[1..n1] and R[1..n2]
 for i ← 1 to n1
    do L[i] ← A[p+i-1]
 for j ← 1 to n2
    do R[j] ← A[q+j]
 i ← 1
 j ← 1
 for k ← p to r
   do if L[i] ≤ R[j]
       then A[k] ← L[i]
            i ← i + 1
       else A[k] ← R[j]
            j ← j + 1
```

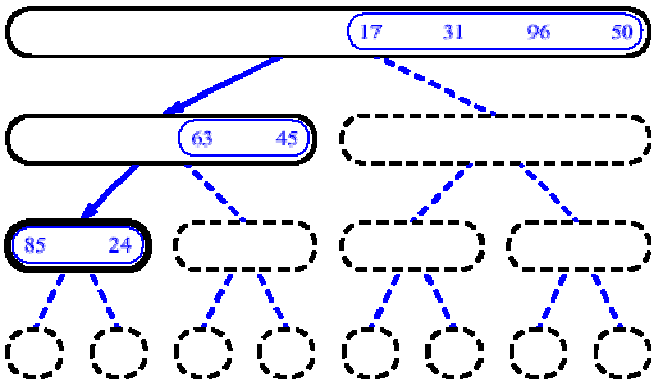
MergeSort (Örnek) - 1



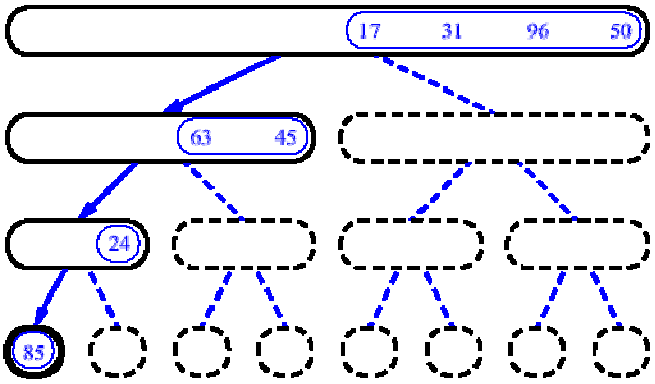
MergeSort (Örnek) - 2



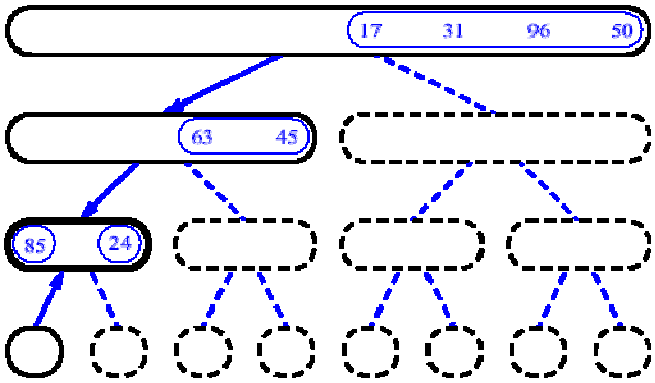
MergeSort (Örnek) - 3



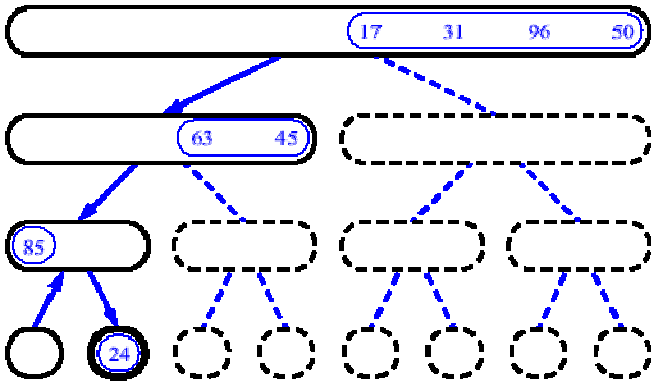
MergeSort (Örnek) - 4



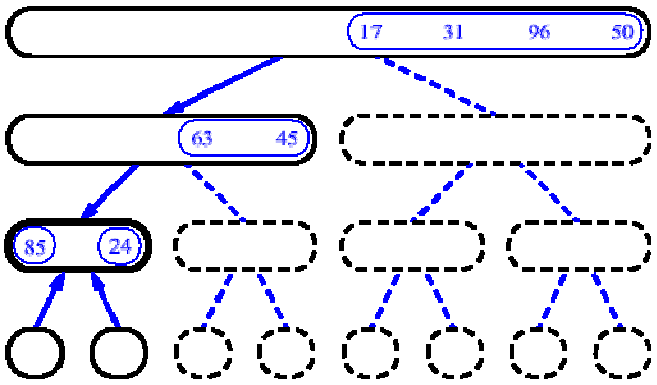
MergeSort (Örnek) - 5



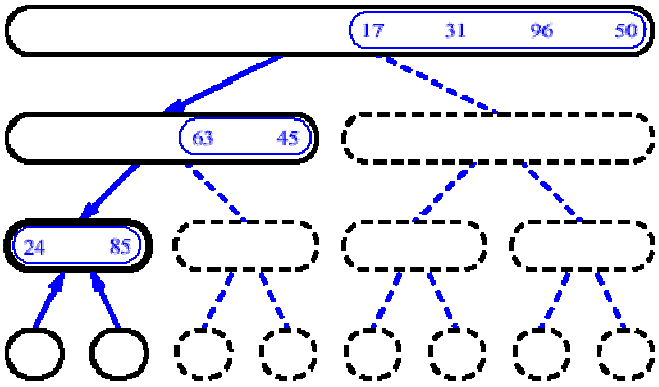
MergeSort (Örnek) - 6



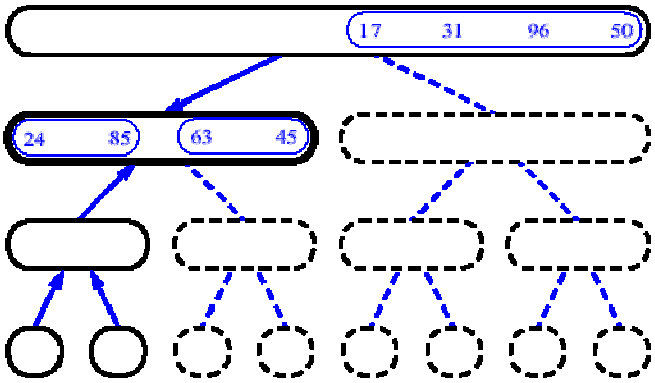
MergeSort (Örnek) - 7



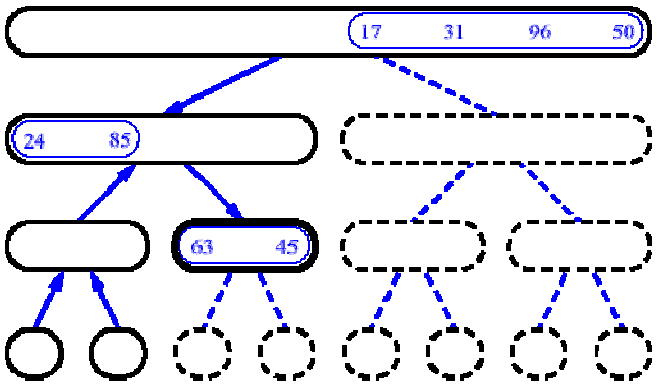
MergeSort (Örnek) - 8



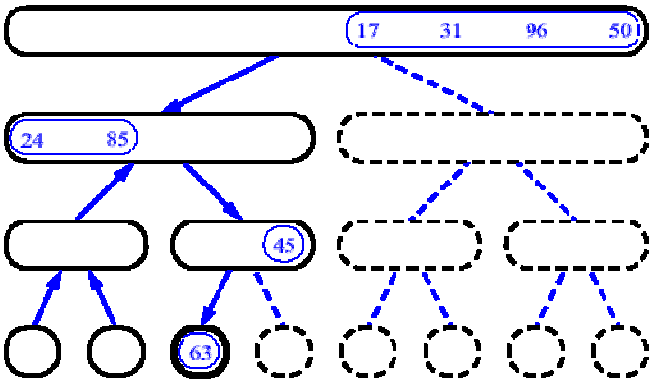
MergeSort (Örnek) - 9



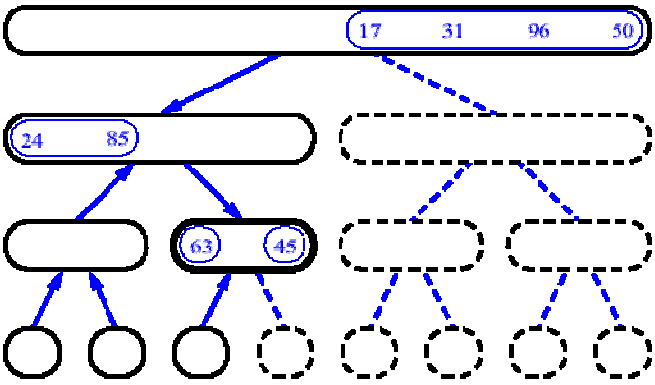
MergeSort (Örnek) - 10



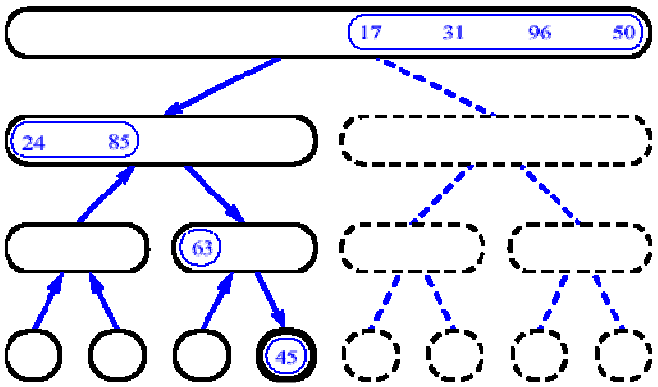
MergeSort (Örnek) - 11



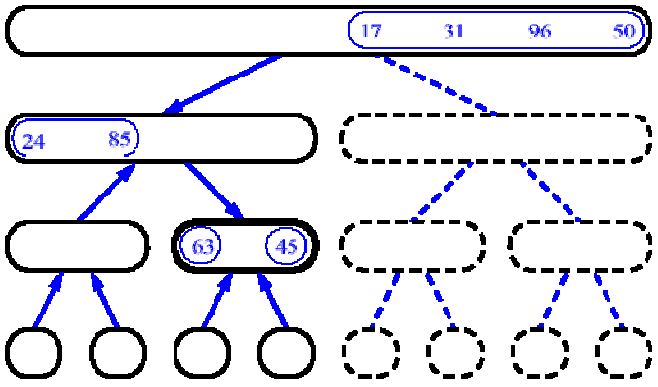
MergeSort (Örnek) - 12



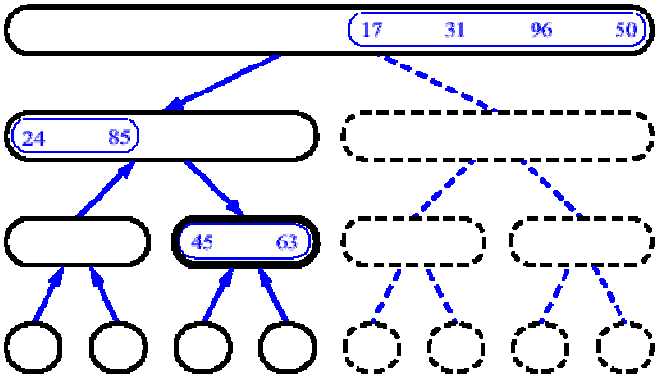
MergeSort (Örnek) - 13



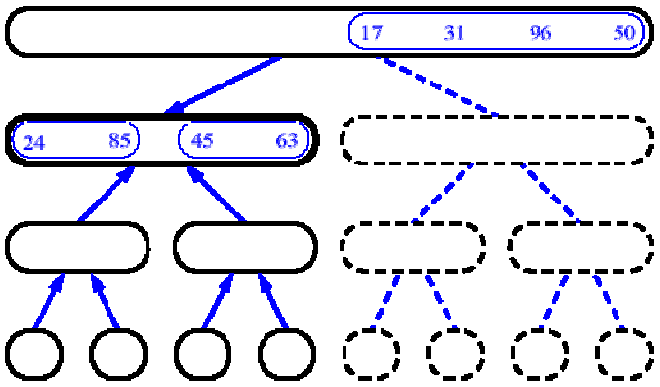
MergeSort (Örnek) - 14



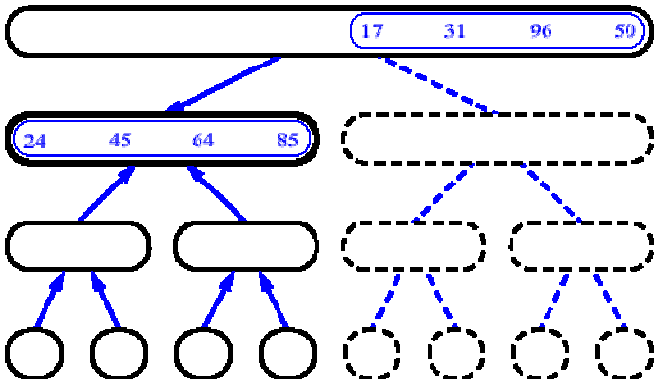
MergeSort (Örnek) - 15



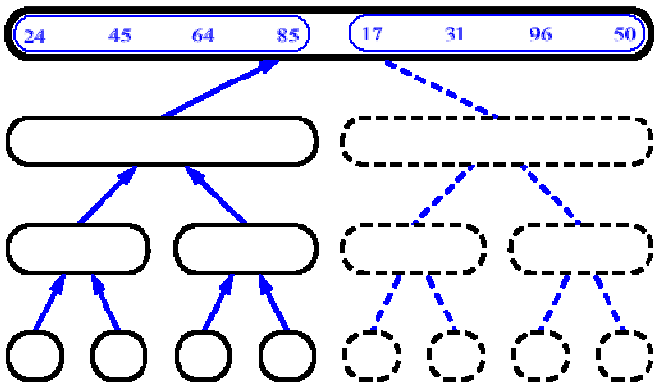
MergeSort (Örnek) - 16



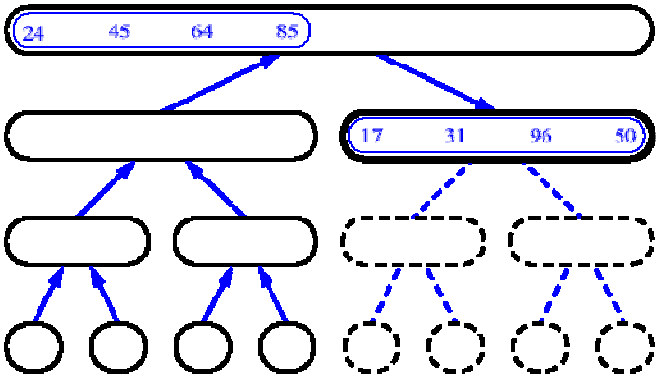
MergeSort (Örnek) - 17



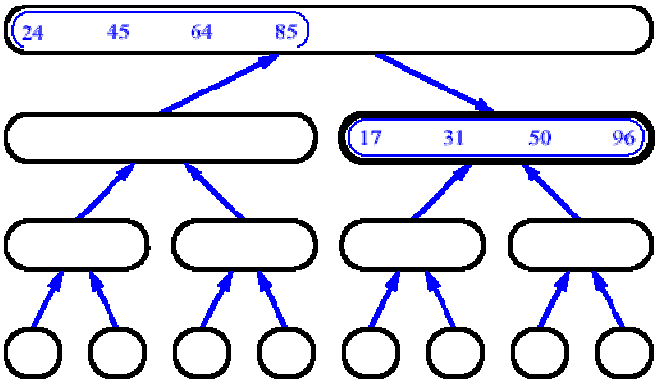
MergeSort (Örnek) - 18



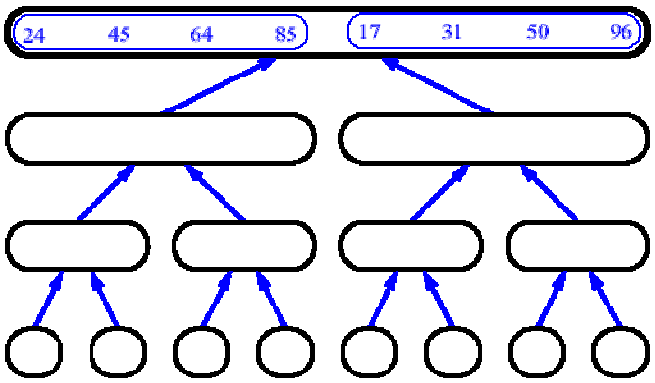
MergeSort (Örnek) - 19



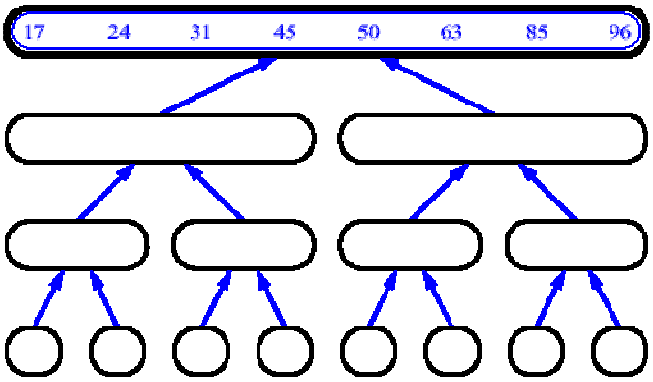
MergeSort (Örnek) - 20



MergeSort (Örnek) - 21

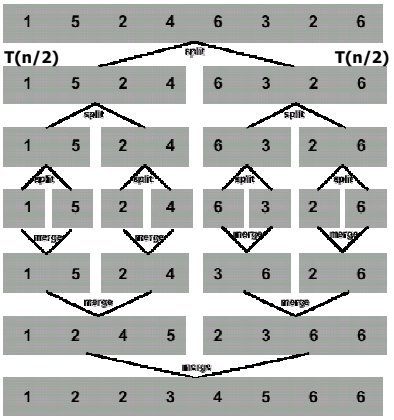


MergeSort (Örnek) - 22



Merge Sort Özet

- *n* adet sayıyı sıralamak için
  - eğer n=1 işlem yapma
  - özyinelemeli olarak $\lfloor n/2 \rfloor$ ve $\lceil n/2 \rceil$ adet elemanı sırala
  - 2 sıralı listeyi $\Theta(n)$ süresinde birleştir
- Strateji
  - Problemi benzer altproblemlere böl
  - Özyinelemeli olarak alt problemleri çöz
  - Çözümleri birleştir



MergeSort çalışma süresi

T(n)=

(trivial case) denemeyle çözülür, n=1

Alt problem sayısı * T(n/alt problem boyutu)+bölme+birleşim , n=1

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Tekrarlı Substitution Metodu

- Merge sort çalışma süresi ($n=2^b$ olduğu herhangi bir b sayısı için kabul edilirse).

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2\,T(n/2) + n \\ &= 2\,(2\,T(n/4)+n/2) + n \\ &= 2^2\,T(n/4) + 2n \\ &= 2^2\,(2\,T(n/8)+n/4) + 2n \\ &= 2^3\,T(n/8) + 3n \\ &= 2^b\,T(n/2^b) + bn \\ &= n\,T(n/n) + n\lg n \\ &= n + n\lg n \end{aligned}$$

Haftalık Ödev

- Sırasız dizide minimum ve maksimum elemanları bulan algoritmanın analizini yapınız. Ödev çıktı alınarak ve bir kapak sayfasıyla birlikte teslim edilecek. Kapak sayfası örneği http://w3.gazi.edu.tr/web/akcayol adresinde downloads bölümünden elde edilebilir.

INPUT: A[l..r] – sırasız integer dizisi, $l \le r$.
OUTPUT: (*min, max*), $\forall j\ (l \le j \le r)$: A[*j*] ≥ *min* ve A[*j*] ≤ *max*

```
MinMax(A, l, r):
    if l = r then return (A[l], A[r])          Trivial case
    q←⌊(l+r)/2⌋                                Divide
    (minl, maxl)← MinMax(A, l, q)              } Conquer
    (minr, maxr)← MinMax(A, q+1, r)            }
    if minl < minr then min = minl else min = minr   } Combine
    if maxl > maxr then max = maxl else max = maxr   }
    return (min, max)
```