

ALGORITMA ANALİZİ #

+ → SÜRE-HİZ HESAPLARI

→ → KARMAŞIKLIKLAR

+ → NOTASYONLAR

+ → ALGORITMA TASARIM STRATEJİLERİ

→ → RECURSIVE ALG. KARMAŞIKLIĞI

[Master method
Yenile kayma "
Recursion Tree"]

Rakursif
Backtracking - geriye idare
Bell ve Jones
Dinamik programlama
Greed yaklaşımı
Branch and bound - dolu ve sırıç
Heuristic - Serziseli
Iteratif
Decrease and cover
Transform and cover

+ → ARAMA ALGORİTMALARI (kodları)

[Linear Search
Binary "

+ → SIRALAMA ALGORİTMALARI (kodları)

[Dairel Sıralama - Bubble sort
Halka " " - Sıvık " "
Seçme " " - Seçmeli " "
Birleştirici " " - Merge " "
Yığın " " - Nayıp " "

→ → AMORTIZED ANALİZ

+ → DENGELİ AĞACLAR

[Red-Black Tree
B-Tree
AVL]

+ → BİNARY SEARCH TREE (Ağacları ziyare, sıralama, ekleme, silme, arama)

+ → MINIMUM SPANNING TREE

[Prim
Kruskal
Dijkistra]

Örnek

✓

	n^2	$\frac{3}{2} \cdot n$	\sqrt{n}
1 dakika	$(24 \cdot 10^4)$	$(4 \cdot 10^{10})$	$(36 \cdot 10^{20})$
1 saat	$(16 \cdot 10^5)$	$(24 \cdot 10^{10})$	$(36)^2 \cdot 10^{20}$

1 GHz UK PC de
 $\log \text{ verisi istenir?}$
 10^9 Hz

$$\frac{1 \text{ sn}' \text{de } 10^9}{60 \text{ sn} \quad p^2}$$

$$10^9 \cdot 60 = n^2$$

$$n = \sqrt{16 \cdot 10^5}$$

$$n = 4 \cdot 10^5$$

$$n = 24 \cdot 10^5$$

$$\frac{1 \text{ sn} \quad 10^9}{60 \text{ sn} \quad \frac{3}{2} n}$$

$$10^9 \cdot 60 = \frac{1}{2} n$$

$$10^9 \cdot 4 = n$$

$$\frac{1 \text{ sn} \quad 10^9}{60 \text{ sn} \quad \sqrt{n}}$$

$$10^{10} \cdot 6 = \sqrt{n}$$

$$10^{20} \cdot 36 = n$$

$$\frac{1 \text{ sn} \quad 10^9}{3600 \quad n^2}$$

$$(3600 \cdot 10^9) = n^2$$

$$3600 \cdot 10^8 \cdot 10 = n^2$$

$$60 \cdot 10^4 \cdot 2 \leq n$$

$$18 \cdot 10^5 \approx n$$

$$\frac{1 \text{ sn} \quad 10^9}{3600 \quad \frac{3}{2} n}$$

$$\frac{1}{2} n = 10^{11} \cdot 36$$

$$n = 24 \cdot 10^9$$

$$\frac{1 \text{ sn} \quad 10^9}{3600 \quad \sqrt{n}}$$

$$\sqrt{n} = 36 \cdot 10^9$$

=

Örnek:

✓

	10000	1000000
$\log n$	$13 \cdot 10^{-9} \text{ sn}$	$20 \cdot 10^{-9} \text{ sn}$
n	10^{-5} sn	10^{-3} sn
n^2	10^{-1} sn	10^3 sn
2^n		

Verilen bilgisayarda her konut
 $10^{-9} \text{ sn}' \text{de işlenir.}$
 Karmasılıklara göre log sn'ide işler?

$$\frac{1 \text{ konut } 10^{-9} \text{ sn}' \text{de}}{x \quad 1 \text{ sn}}$$

$$x \cdot 10^{-9} = 1$$

$$x = 10^9$$

$1 \text{ sn } 10^9$ konut istenir

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 1}$$

$$x \cdot 10^9 = \log 10^4$$

$$x = 13 \cdot 10^{-9}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 10^4}$$

$$x \cdot 10^9 = 10^4$$

$$x = 10^{-5}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 10^6}$$

$$x \cdot 10^9 = 10^6$$

$$x = 10^{-3}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 10^3}$$

$$x = 10^{-6}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad (10^4)^2}$$

$$10^8 = 10^4 \cdot x$$

$$x = 10^{-1}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad (10^6)^2}$$

$$10^9 \cdot x = 10^{12}$$

$$x = 10^3$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 2^{10000}}$$

$$x \cdot 10^9 = 2^{10000}$$

$$x = 2^{10000} \cdot 10^{-9}$$

$$\frac{1 \text{ sn} \quad 10^9}{x \quad 2^{1000000}}$$

$$x = 2^{1000000} \cdot 10^{-9}$$

Örnek: $g(n) = n^2$ olan bir algoritma $\frac{1}{\text{sn}} 10^9$ konut işliyor, bu da $\frac{1}{\text{dk}}$ içinde işlenerek veri sayısı nedir?

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ 60 \frac{\text{sn}}{n^2} \\ \hline n^2 = 6 \cdot 10^{10} \\ n = 6 \cdot 10^5 \Rightarrow n \approx 24 \cdot 10^4 \quad n \approx 24 \cdot 10^4 \end{aligned}$$

Örnek:

n		
10000	100000	1000000
$\log n$	$13 \cdot 10^{-5} \text{ sn}$	$17 \cdot 10^{-5} \text{ sn}$
n	10^{-5} sn	10^{-4} sn
$n \log n$	$13 \cdot 10^{-5} \text{ sn}$	$17 \cdot 10^{-4} \text{ sn}$
n^2	10^{-4} sn	10^{-3} sn
n^3	10^3 sn	10^6 sn
2^n		

$\frac{1}{\text{sn}} 10^9$ konut
ideye ise;

Süre hesapları?

10

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times \log 10^4 \\ \times 10^9 = 10^9 \log 10^4 \\ x = 13 \cdot 10^{-9} \text{ sn} \end{aligned}$$

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times 10^4 \log 10^4 \\ \times 10^9 = 10^4 \log 10^4 \\ x = 13 \cdot 10^4 \cdot 10^{-9} \end{aligned}$$

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times 10^5 \log 10^5 \\ \times 10^9 \cdot x = 10^5 \log 10^5 \end{aligned}$$

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times 10^6 \log 10^6 \\ \times 10^9 = 10^6 \log 10^6 \end{aligned}$$

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times (10^6)^2 \\ 10^9 \cdot x = 10^{12} \end{aligned}$$

$$\begin{aligned} \frac{1}{\text{sn}} 10^9 \\ \times 10^6 \\ x = 10^3 \end{aligned}$$

Örnek: $g(n) = n^2$ $\frac{1}{\text{dk}}$

$$\frac{1}{60} \frac{10^9}{n^2}$$

$\frac{1}{\text{dk}}$ hızla kaç veri işler?

$$\begin{aligned} n &= 16 \cdot 10^5 \\ n &\approx 2,4 \cdot 10^5 \Rightarrow n \approx 24 \cdot 10^4 \end{aligned}$$

Örnek: $g(n) = 2n^2$

$$\frac{1}{60} \frac{2 \cdot 10^9}{2(1 \cdot 10^4)}$$

$2n^2$ hızla 600.000 veri kaç dk'da işler?

$$\begin{aligned} 2 \cdot 10^8 \cdot x &= 2 \cdot 36 \cdot 10^9 \\ x &= 360 \text{ sn} \Rightarrow \\ x &\Rightarrow 6 \text{ dk} \end{aligned}$$

→ KARMAŞIKLIKLAR ←

Bir program kodunun zaman karmaşıklığını hesaplamak için 5 kural;

1-) DÖNGÜLER : Döngünün çalışma zamanı en çok döngü içindeki çalışma zamanının iterasyon sayısıyla çarpılması kadardır.

n defa
calısır. $\left\{ \begin{array}{l} \text{for } (i=1; i \leq n; i++) \\ \quad \{ \\ \quad \quad m = m+2; \\ \quad \} \end{array} \right. \xrightarrow{\text{K}} \text{sabit zaman}$

$$\text{Toplam zaman} = \text{sabit} \cdot c + n = \underline{cn} \Rightarrow O(n)$$

2-) İÇİĞE DÖNGÜLER : içeki analiz yapılır. Toplam zaman, bütün döngülerin çalışma sayılarının çarpımına eşittir.

dış döngü n defa calısır $\left\{ \begin{array}{l} \text{for } (i=1; i \leq n; i++) \{ \\ \quad \text{for } (j=1; j \leq n; j++) \{ \\ \quad \quad k = k+1; \\ \quad \} \end{array} \right. \xrightarrow{\text{sabit zaman}} \text{içeki döngü } n \text{ defa calısır.}$

$$\text{Toplam zaman} = c + n + n = cn^2 \Rightarrow O(n^2)$$

3-) ARDISIK DEYİMLER : Her deyimin zamanı birbirine eklenir.

$x = x+1; \xrightarrow{\text{sabit zaman}}$
 $\text{for } (i=1; i \leq n; i++) \{$
 $\quad m = m+2; \xrightarrow{\text{sabit zaman}}$
 $\}$

dış döngü n defa calısır $\left\{ \begin{array}{l} \text{for } (i=1; i \leq n; i++) \{ \\ \quad \text{for } (j=1; j \leq n; j++) \{ \\ \quad \quad k = k+1; \\ \quad \} \end{array} \right. \xrightarrow{\text{sabit zaman}}$

İç içe döngüler çarpılır, ardışık olarak gelanlar toplanır

$$\text{Toplam zaman} = c_0 + c_1 \cdot n + c_2 \cdot \underline{n^2} \Rightarrow \underline{O(n^2)}$$

$$c_0 + c_1 \cdot n \Rightarrow \underline{n^2}$$

4-) IF - THEN - ELSE DİYİMLERİ : En kötü çalışma durumu ; test sonrakina then veya else 'den hangisi billykse o eklenir.

```

sabit   → if (depth() != otherStack.depth()) {
          return false;
      } } Then: sabit
else {
    for (int i = 0; i < depth(); i++) {
        if (list[i].equals(otherStack.list[i])) {
            return false;
        }
    } } else: (sabit + sabit) * n
    
```

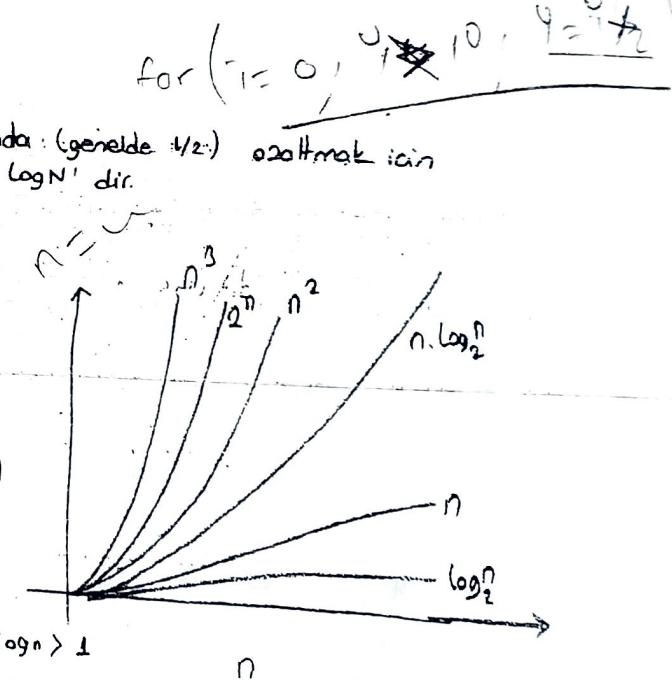
$$\text{toplam zaman} = c_0 + c_1 + (c_2 + c_3) * n = O(N)$$

5-) LOGARİTMİK KARMASILIKLAR :

Problemin billykliğinin betti oranda : (genelde $\frac{1}{2}$) $O(2^{\log n})$ için
sabit bir zaman harcanıyorsa bu $O(\log n)$ dir.

- $O(1)$ = sabit
- $O(\log n)$ = logaritmik
- $O(\log \log n)$ = polilogaritmik
- $O(n)$ = lineer
- $O(n \log n)$ = Loglineer
- $O(n^2)$ = karesel
- $O(n^c)$ = polinomsal
- $O(c^n)$ = üssel
- $O(n!)$ = faktoriyel
- $O(n^n)$ = gen kombinasyonel

$$n^n > n! > c^n > n^c > n^2 > n \log n > n > [\log n]^c > \log n > 1$$



Örnek :

```

for (i=1; i<n; i++) { O(n)
    for (j=1; j<i; j++) { O(i)
        a[i][j] = a[0][0] + i + j; O(1)
    }
}
    
```

$O(n^2 + n)$

$T(n) =$

Örnek: $\text{void barl (first, last) } \{$

$$\underline{\text{size} - t = \text{last} - \text{first}} \rightarrow O(1)$$

if $\text{last} < \text{first}$ $\leftarrow \text{if } (\text{size} <= 1) \rightarrow O(1)$
ve son verse bile
1 kez istenildiği için
 $O(1)$ olur.

return $\rightarrow O(1)$

$$\underline{\text{middle} = (\text{first} + \text{size}) / 2} \rightarrow O(1)$$

burda fonksiyon
çalıştırılmış önce
bastırılarak
sonra yanıtın
sona kadar

$\underline{\text{barl(first, middle)}} \rightarrow T(n/2)$

$\underline{\text{barl(middle, last)}} \rightarrow T(n/2)$

$\underline{\text{for } i = \text{first}; i < \text{last}; i++}$ } $\rightarrow O(n)$

$$\text{toplam zaman} = 4 \cdot O(1) + 2 \cdot T(n/2) + O(n)$$

Örnek:

$\text{FindMedian}(x, y) \{ \rightarrow O(1)$

$\underline{\text{if } n == 1 \rightarrow O(1)}$
 $\underline{\text{return } x(1) + y(1) / 2 \rightarrow O(1)}$

$f = b \cdot (n+1) / 2c \rightarrow O(1)$
 $c = d \cdot (n+1) / 2e \rightarrow O(1)$

$$mx = (x(f) + x(c)) / 2 \rightarrow O(1)$$

$$my = (y(f) + y(c)) / 2 \rightarrow O(1)$$

$\underline{\text{if } mx == my \rightarrow O(1)}$

return $mx \rightarrow O(1)$

$\underline{\text{else if } mx < my \rightarrow O(1)}$

return $\text{findMedian}(x(c...n), y(1...f)) \rightarrow T(n/2)$

$\underline{\text{else}}$

return $\text{findMedian}(x(1...f), y(c...n)) \rightarrow T(n/2)$

$2 \cdot T(n/2)$
olmayaçık
çünkü if
şartlı ifade old.
2inden biri
çalışırılocak

$$T(n) = 10 \cdot O(1) + T(n/2)$$



ÖRNEKLER :

- ① Bir dizinin ortametik ortalamasını bulan ve sonucu kullanıcıya geri döndüren bulOrta() adlı fonk. verilmiştir. Bu fonk. yürütmeye zamanını $T(n)$ bağıntısını elde ediniz.

```
float bulOrta (float A[], int n) { →  $T(n)$ 
    float ortlama; toplam=0; →  $O(1)$ 
    int k; →  $O(1)$ 
    for (k=0;  $k < n$ ; k++) →  $O(n)$ 
        toplam+=A[k]; →  $O(1)$ 
    ortlama=toplam/n; →  $O(1)$ 
    return ortlama; →  $O(1)$ 
}
```

$$T(n) = \underbrace{5 \cdot O(1)}_{\text{}} + O(n) \Rightarrow T(n) = O(n) //$$

- ② En küçük elemeni bulmak için; $T(n)$ hesabi

```
float bulEnKucuk (float A[]) { →  $T(n)$ 
    float enKucuk;
    int k;
    enKucuk = A[0]; /* ilk elemen en küçük →  $O(1)$ 
    for (k=1; k<n; k++) →  $O(n)$ 
        if (A[k] < enKucuk) →  $O(1)$ 
            enKucuk = A[k];
    return enKucuk; →  $O(1)$ 
}
```

$$T(n) = 6 \cdot O(1) + O(n) \Rightarrow T(n) = O(n) //$$

$$T(n) = 5 \cdot O(1) + O(n)$$

③ Matris Toplama için $T(n)$ Hesabı iki matrisi toplayıp üzerine matrise yesteşiren

void ToplaMatris(A, B, C) { $\xrightarrow{\text{fonk.}}$ $T(n)$

int A[n][m], B[n][m], C[n][m]; $\xrightarrow{\text{O(1)}}$

int i, j; $\xrightarrow{\text{O(1)}}$

$\xrightarrow{\text{for (i=0; i<n; i++)}}$ $\xrightarrow{\text{O(n)}}$

$\xrightarrow{\text{for (j=0; j<m; j++)}}$ $\xrightarrow{\text{O(m)}}$

$C[i][j] = A[i][j] + B[i][j]$ $\xrightarrow{\text{O(1)}}$

3

14

m.n

m=n

yakınsa satır yerine stlin

$\xrightarrow{T(n) = O(n^2)}$

$O(n^2)$

$O(n^2)$

④ Reküratif faktoriyel hesabı $T(n) = 3 \cdot O(1) + O(n) + O(n^2)$

$T(n)$ unsigned int faktoriyel (unsigned int n) { $\xrightarrow{O(1) + O(n) + O(n^2)}$

$\xrightarrow{O(1)}$ if ($n \leq 1$)
return 1;

$\xrightarrow{O(n^2)}$ $T(n) = O(n^2) + O(n)$

$\xrightarrow{O(n)}$ else
return ($n * \xrightarrow{\text{faktoriyel}} \text{faktoriyel}(n-1)$);

mantar
yer

⑤ Sonlu bir dizide max. elemeni bulan algoritma;

$T(n)$ procedure max(a1, a2, ..., an : integers)

$\xrightarrow{O(1)}$ max := a1

$\xrightarrow{T(n) = 2 \cdot T(\frac{n}{2}) + 1}$

$\xrightarrow{O(n)}$ for i := 2 to n
if max < ai then max := ai $\xrightarrow{\text{if max} \leq a}$

{ max. dizinin en büyük elemanıdır. } $\xrightarrow{T(\frac{n}{2}) = 2 \cdot T(\frac{n}{2}) + 1}$

⑥ Sirali Arama Algoritmasi ;

$T(n)$ Procedure Linear Search (x : integer, a_1, a_2, \dots, a_n : distinct integer)

```
i := 1
while (i < n and  $x \neq a_i$ )
    i := i + 1
```

```
if i  $\leq n$  then location := i
else location := 0
```

{ location x 'e esit dakterim , eger " x " 0 ise bolummada }

⑦ Lineer Arama Algoritmasi

procedure Linear Search (x : integer, a_1, a_2, \dots, a_n : distinct integer)
 i := 1
 while ($i \leq n$ ve $x \neq a_i$)
 i := i + 1
 if $i \leq n$ then location := i

⑧ ikili Arama Algoritmasi ;

procedure binary search (x : integer, a_1, a_2, \dots, a_n : increasing integers)

i : 1 { i is left endpoint of search interval }
j : n { j " right " }

while ($i < j$)

$m := \lceil (i+j)/2 \rceil \rightarrow \text{logn}$

if $x > a_m$ then $i := m + 1$

else $j := m$

end

if $x = a_i$ then location := i

else location := 0

$O(n)$

$O(\log n)$

$$O(f(n)) + O(g(n)) = O(f(n) + g(n)) //$$

⑨

```

i=1; → O(1)
sum = 0; → O(1)
while (i <= n) { → O(n)
    i = i + 1; → O(1)
    sum = sum + i; → O(1)
}

```

$\overbrace{\hspace{10em}}^{O(n)}$

⑩

```

i=1;
sum = 0;
while (i <= n) {
    j=1;
    while (j <= n) {
        sum = sum + i;
        j = j + 1;
    }
    i = i + 1
}

```

$O(n^2)$

⑪ örnek:

```

s=0
for(i=1; i<=n; i=i+1) →
    s=s+i;
    print(s);

```

a-) $n=32$ için program çalışması nedir?
 b-) satır 3 kaç kez yürütülür?

$$a-) \frac{32 \cdot 33}{2} =$$

$$b-) n-1$$

⑫ örnek:

```

s=0
for(i=1; i<=n; i=i+2)
    s=s+i;
    print(s);

```

a-) $n=32$ için program çalışır mı?
 b-) satır 3 kaç kez yürütülür?

$$a-) 1+2+4+8+\dots+16+32 = 63$$

$$b-) \log n + 1$$

⑬ örnek;

RecursiveSort (A, i, j) {

```

if j = i + 1
    if A[i] > A[j]
        swap (A[i], A[j])
else
    k = ceiling (2 * (j - i + 1) / 3) - 1
    RecursiveSort (A, i, i + k)
    RecursiveSort (A, j - k, j)
    Merge (A, i, j + k)

```

ceiling → tavanı
yapırır

Gestki Algoritma Örnekleri

- ① Sıralı bir dizideki en büyük elemanın bulan Algoritma ;

```
procedure max( a1, a2, ..., an : integers )
  max := a1
  for i := 2 to n
    if max < ai then max := ai
  end {max en büyük eleman}
```

- ② Euclidean Algoritması ;

```
procedure gcd( a, b : positive integers )
  x := a
  y := b
  while y ≠ 0
    begin
      r := x mod y
      x := y
      y := r
    end {gcd(a,b) is x }
```

- ③ Constructing Base b Expansions

```
procedure base b expansion( n : positive integer )
  q := n
  k := 0
  while q ≠ 0
    begin
      ak := q mod k
      q := [q/b]
      k := k + 1
    end {the base b expansion of n is (ak-1, ..., a1, a0)b}
```

- ④ Tamsayıların Toplamları ;

```
procedure add( a, b : positive integer )
  {the binary expansions of a and b are (an-1, an-2, ..., a1, a0)2
  and (bn-1, bn-2, ..., b1, b0)2, respectively}
  c := 0
  for j := 0 to n-1
    begin
      d := [(aj + bj + c) / 2]
      sj := aj + bj + c - 2d
      c := d
    end
  sn := c
  {the binary expansion of the sum is (sn, sn-1, ..., s0)2}
```

⑤ Tamsayıların Çarpımı:

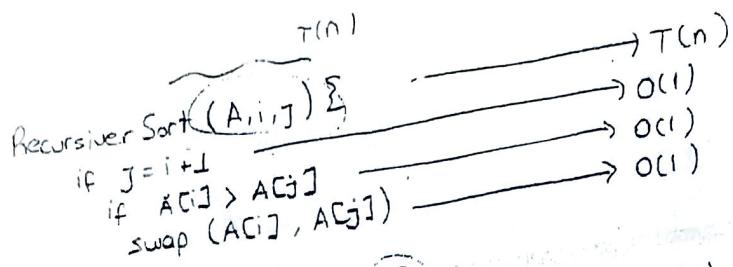
```
procedure multiply (a,b : positive integer)
{the binary expansions of a and b are  $(a_{n-1}, a_{n-2}, \dots, a_1, a_0)_2$ 
and  $(b_{n-1}, b_{n-2}, \dots, b_1, b_0)_2$ , respectively}
for j := 0 to n-1
begin
  if  $b_j = 1$  then  $c_j := a$  shifted j places
  else  $c_j := 0$ 
end
{ $c_0, c_1, \dots, c_{n-1}$  are the partial products}
p := 0
{p is the value of a.b}
```

⑥ Matris Çarpımı:

```
procedure matrix multiplication (A,B : matrices)
for i := 0 to m
begin
  for j := 1 to n
  begin
     $c_{ij} := 0$ 
    for q := 1 to k
       $c_{ij} := c_{ij} + a_{iq}b_{qj}$ 
  end
end
{ $C = [c_{ij}]$  is the product of A and B}
```

⑦ Boolean Çarpımı:

```
procedure Boolean Product (A,B : 2m-one matrices)
for i := 1 to m
begin
  for j := 1 to n
  begin
     $c_{ij} := 0$ 
    for q := 1 to k
       $c_{ij} := c_{ij} \vee a_{iq} \wedge b_{qj}$ 
  end
end
{ $C = [c_{ij}]$  is the Boolean product of A and B}
```



ceiling
4/tavşan
yaratır

$$T(n) = 3 \cdot T\left(\frac{n}{3}\right) + O(1)$$

$$a=3 \quad b=\frac{3}{2} \quad n^{\log_b a} = n^{\log_{\frac{3}{2}} 3}$$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 3} ?$$

Örnek:

```

    T(n)
    funny(x)
    if (n == 1) → O(1)
    return ... → O(1)
    for (i=1 to n) → O(n)
    print; →
    x = rand int(1, n) → O(1)
    if (x > n/2) → O(1)
    funny(n/2) → T(n/2)
    else:
    return ...

```

→ Tekrarlı bağıntısını ve karmaşıklığını bulunuz
 → Bu kod hangi algoritma tasarım stratejisi göre yazılmıştır? (rekürsif)
 ↳ itaretif olmaz çünkü kendini çağırıyor

zott!!

funny'nin $n/2$ kadar çalışması if'e bağlı yani bunu sınırlıyper.
 if koşulundan dolayı $1/2$ kadar çağırır.

$$T(n) = \frac{1}{2} \cdot T\left(\frac{n}{2}\right) + O(n) + T(O(1))$$

$$\frac{1}{2} \cdot 2^n$$

$O(n) \rightarrow$ karmaşıklık

Örnek:

```

    function deneme (A[1, ..., n]) {
        md = (l+r)/2
        l = A[md-1]
        r = A[md+1]
        if l < A[md] & r > A[md]
            return A[md]
        else
            if l < A[md] & r < A[md]
                return deneme (A[1, ..., md]) → T(n/2)
            else
                return deneme (A[md, ..., n]) - T(n/2)
    }

```

$$T(n) = \frac{1}{2} T\left(\frac{n}{2}\right) + O(1)$$

Ornek: İki değişkenin içeriğini başka bir değişken kullanmadan değiştiren Algoritmayı yazınız.

int a, b;

a=5;
b=7;

a = a + b; a = 12
b = a - b; b = 5
a = a - b; a = 7)



Ornek: Yukardaki soruya 3 değişken için yapmak istenirse olusacak Algoritma;

int a, b, c;

(
 ↑ a=5;
 b=7;
 ↓
 c=9;
)

a → b
b → c
c → a

yani; a = 9
 b = 5
 c = 7

Ornek:

C₁ for (i=1; i<=n¹⁰; i++) {
C₂ m=n/i;
C₃ for (j=1; j<=m; j++) {
C₄ int j;
 }
 }

cost maliyet

C₁ n
C₂ n-1
C₃ (n-1). log n
C₄ (n-1). log n

+

2. for (n-1) for döngüsü
icinde olduğunu için gelecek
çarpımdan sonra m'e kadar
gidersek m'de n/i^1 dir.
(Burada n'e sabit 10 deşer
verirsek.)

O(n log n)

i=1 için m=10

i=2 " m=5

i=3 " m=10/3=3.3

$1.3/3=1$

$1.2/2=1$

$1.1/1=1$

deşerler

azalan bir fonksiyonda
olduğu için "log n" dir.

Ornek:

```
s=0
for(i=1; i< 2+n; i++) → n
for(j=1; j<i+i; j++) → n2
for(k=1; k<j; k++) → n2
    if (j % i)
        s++;

```

$\frac{n^5}{O(n^5)}$

Ornek:

```
s=0
for(i=0; i<sqrt(n)/2; i++) → √n/2
    s++
for(j=0; j<sqrt(n)/4; j++) → √n/4
    s++
for(k=0; k<8+j; k++) → 8 + √n/4
    s++

```

$O(\sqrt{n})$

$$\frac{8 + \frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{4} + \frac{\sqrt{n}}{4}}{8 + \sqrt{n}}$$

Ornek:

① int mss () {

```
int ms=0;
for(i=0; i<A.size(); i++)
    for(j=i; j<A.size(); j++)
        int ts = 0;
        for(k=i; k<=j; k++)
            ts += A[k];
            if (ts > ms)
                ms = ts;
}
return ms;
}
```

Brute-force ile yazılmıştır.

$O(n^3)$ yazılır gelir

② int mss () {

```
int ms=0;
for(i=0; i<A.size(); i++)
    int ts=0;
    for(j=i; j<A.size(); j++)
        ts += A[j];
        if (ts > ms)
            ms = ts;
return ms;
}
```

Bda brute-force'dur
ancak da iyileştirilmiş
halidir.

$O(n^2)$ ye yazılır
gelir.

* ikiside aynı işi yapar. Tek farkı; brute-force ve iyileştirilmiş olmalıdır.

- itaretif olarak yazılmıştır

↓

?

*

NOTASYON LAR :

① O Notasyonu (En Küçük Durum) :

$$f(n) = O(g(n)) \text{ olmak üzere ;}$$

$$0 \leq f(n) \leq C \cdot g(n) \text{ olur.}$$

$g(n)$, $f(n)$: Üstten sınırlar.

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \Rightarrow \text{constant} < \infty$$

② Θ Notasyonu (Ortalama Durum) :

$$f(n) = \Theta(g(n)) \text{ olmak üzere ;}$$

$$C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \Rightarrow \text{constant} \neq 0$$

$g(n)$, $f(n)$: Hem üstten hem alttan sınırlar.

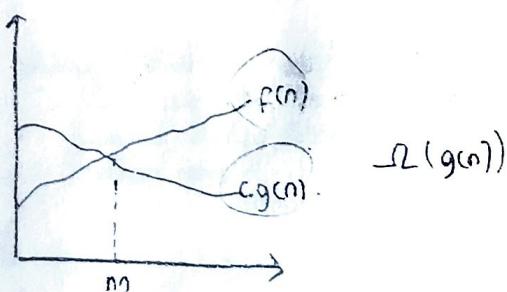
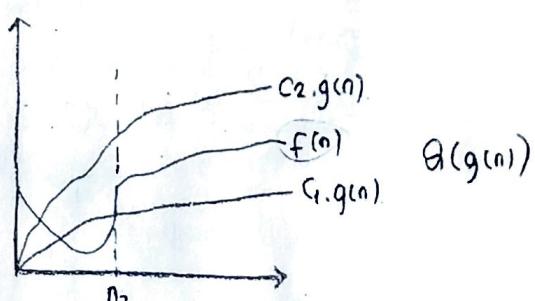
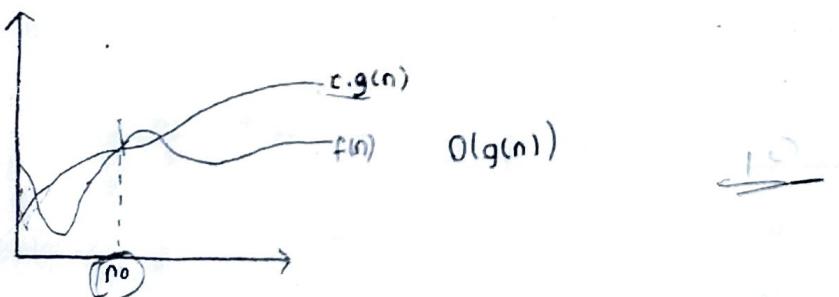
③ Ω Notasyonu (En İyi Durum) :

$$f(n) = \Omega(g(n)) \text{ olmak üzere ;}$$

$$0 \leq c \cdot g(n) \leq f(n)$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \text{constant}$$

$g(n)$, $f(n)$: Altta sınırlar.



Ornek: Aşağıdaki sayıların doğruluklarını test ediniz.

① $4n = O(5n)$ en kötü durum notasyonu dir. $g(n), f(n)$ i üstten sınırlar burda sağlar

$$0 \leq 4n < c \cdot 5n$$

$$0 \leq \frac{4}{5} n \leq c \quad 1 \leq \frac{c}{\frac{4}{5}} \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \text{constant} \rightarrow \text{islemde görelde sağlar.}$$

② $4n+3 = \Omega(n)$ en iyi durum notasyonu dir. $g(n), f(n)$ i alttan sınırlar

$$0 \leq c_1 \cdot n \leq 4n+3$$

$$0 \leq c_1 \leq 4 + \frac{3}{n} \rightarrow$$

$$\lim_{n \rightarrow \infty} 4 + \frac{3}{n} = 4 \Rightarrow c_1 = \text{constant}$$

✓ sağlar

③ $4n+3 = \Theta(n)$ ortalama durumdur. ve sağlar

$$0 \leq c_1 \cdot n \leq 4n+3 \leq c_2 \cdot n$$

$$0 \leq c_1 \leq 4 + \frac{3}{n} \leq c_2$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{4n+3}{n} = 4 \rightarrow$$

✓ sağlar.

Ornek

$3n^2 - 100n + 6$	\neq	$O(n^2)$	1)
-	=	$O(n^3)$	2)
#	\neq	$O(n)$	3)
=	\neq	$\Omega(n^2)$	4)
=	\neq	$\Omega(n^3)$	5)
#	\neq	$\Omega(n)$	6)
=	\neq	$\Omega(n^2)$	7)
=	\neq	$\Omega(n^3)$	8)
#	\neq	$g(n)$	9)

$$(3n^2 - 100n + 6) < c_1 n^3$$

$$1000 \cdot 100 < 10^3 \cdot 10^3 = 10^6$$

$$n^3 < 3 \cdot 10^6 - 10^5$$

$$10^3 < 3 \cdot 10^6 - 10^5$$

Kesin

2-) Sağlar. Günlük; $g(n)$ i üstten sınırlar

3-) Sağlansız, $g(n), f(n)$ üstten sınırları gerakirken sınırlanaz

$$0 \leq 3n^2 - 100n + 6 < c \cdot n$$

$$0 \leq 3n - 100 + \frac{6}{n} < c$$

$$\lim_{n \rightarrow \infty} 3n - 100 + \frac{6}{n} = \infty$$

1-) Külli durum $g(n), f(n)$ i üstten sınırlar ve sağlar.

$$0 \leq 3n^2 - 100n + 6 < c \cdot n^2 \quad \lim_{n \rightarrow \infty} 3 - \frac{100}{n} + \frac{6}{n^2} = 3 \quad \checkmark$$

2-) İyi durum; $g(n), f(n)$ i alttan sınırlaması gerelir

$$0 \leq c \cdot n^2 \leq 3n^2 - 100n + 6 \quad \checkmark$$

5) $g(n), f(n)$ i alttan sınırlaması gerakirken sınırlansın.

$$0 \leq cn^3 \leq 3n^2 - 100n + 6$$

$$0 \leq c \leq \frac{100}{n} + \frac{6}{n^3} \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \text{constant}$$

İsteme göre sınırlar.

6) tonime göre sınırlar örnek
İsteme sonucuna göre sınırlar.

7)
8)
9)

Üçlide sınırlaması gerelir
her üst sınıfla her alt sınıfla kesişir ancak
9.ew istende deper saglamaz.

Ornek:

	$f(n)$	0	Θ	Ω	$g(n)$
1)	$(n+3) \cdot \log n$	✓			$n^2 + \log n$
2)	$n^2 \cdot \log n$		✓	✓	$n^2 + n \log n$
3)	$n \log n$	✓		✓	$n \sqrt{n}$
4)	$n^3 + 10 \log n$		✓	✓	$n^3 + n^2$
5)	$\log_3 n^3$	✓	✓	✓	$2 \log_2 n$

$$16 = 2^4 + 4 \cdot 2^3$$

1) ifadeleri karşılaştırırsak $g(n)$, $f(n)$ 'i üstten sınırlar oda
0 durum sağlanır ve Θ durumda sağlanabilir.

2) ifadelerinde $g(n)$, $f(n)$ 'i üstten sınırlar yani 0 ve Θ durumu sağlanabilir.
 $2^4 < 4 \cdot 2^3$

Ornek:

	$f(n)$	0	Θ	Ω	$g(n)$
1)	$n^3 + n \log n$	✓	✓		$n^3 + n^2 \log n$
2)	$\log(\sqrt{n})$		✓	✓	$\sqrt{\log n}$
3)	$n \log n$		✓	✓	$n \log_4 n$
4)	2^n				$2^{n/2}$
5)	$\log(2^n)$	✓			$\log(3^n)$

1) 3 durumda sağlanır intimalı vardır.

2) $g(n)$, $f(n)$ 'i alttan sınırladığı için; Ω intimalı vardır

3) $g(n)$, $f(n)$ 'i alttan sınırladığı için; Ω

5) $g(n)$, $f(n)$ 'i üstten " " ; 0

Ornek: $f(n) = g(n)$

$(n+5) \log n \leq O(n^2 + \log n) \rightarrow$ 0 durumu demek;
 $g(n), f(n)$ 'i üstten sınırlıysa demektir bunu söyle

$(n+5) \log n \neq \Omega(n^2 + \log n) \rightarrow$ 0 durumunda; $g(n)$, $f(n)$ 'i kon üst hem alttan sınırlaması gereklükken sadece üstten sınırlar yani sağlanır

$(n+5)(\log n) \neq \Omega(n^2 + \log n)$

4) Ω durum demek; $g(n)$, $f(n)$ 'i alttan sınırlıysa demek, yani bunu söyle

Ornek

⑩ Ornek:

$$① 2^{2n} = O(2^n)$$

\rightarrow kötü durumda, Yani; $g(n) = f(n)$ i alttan sınırlar, bu nedenle saptanır.

#6 4

$$② n^3 = \Omega(n^2)$$

↓

Yani; $g(n) = f(n)$ i alttan sınırlar
bu durumu saptar.

$$0 \leq c \cdot n^2 \leq n^3$$

$$0 < \frac{c}{n} \leq 1$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^3}{n^2} = \infty$$

saptanır 2.

$$0 \leq 2^{2n} \leq c \cdot g(n)$$

$$\frac{2^n}{c} \leq n \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

İşte göre de saptanır 2

⑪ Ornek:

dereme(n) {

if ($n \leq 1$)

return 1

else

{ return (dereme(n-1) + dereme(n-2)) }

fibonacci

a-) Algoritmanın çıktılarını veriniz

b-) $n=6$ için recursion tree ağacını çiziniz.

c-) $1 \leq i \leq 5$ için $dereme(i)$ kaç kez yürütülür?

d-) $dereme(6)$ y bulmak için kaç tane toplam işlemi yapılır?

e-) Her toplamanın sabit zaman aldığına dair $dereme(n)$ için bir tekrarlı boşut,

ve karmaşıklığını veriniz.

f-) Sonluude 1 milyon tane toplam upon bir bilgisayarda $dereme(100)$ 'ü ne kadar sürede

bulur?

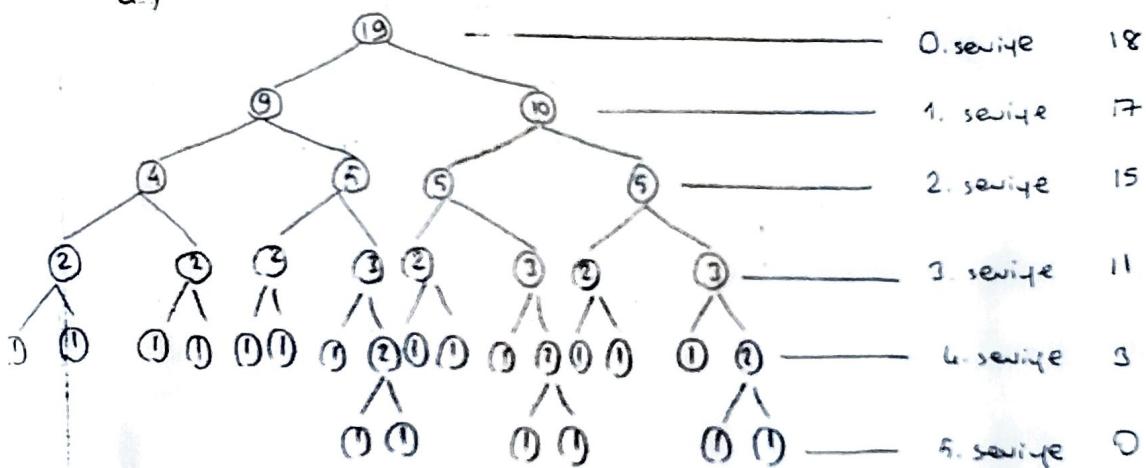
g-) Daha verimli bir algoritma yazılırsa yukarıdak karmaşıklığını bulursa?

a-)

Ornek: n elementli bir diziyi sıralanmak için merge sort kullanarak

- $n=19$ için recursion tree çizin
- Ağacın seviye sayısı kaçtır?
- En kötü durumda her bir seviyedeki karşılaştırma sayısı?
- Toplam karşılaştırma sayısı nedir?
- n , 2^n 'in katı olmak üzere n sayıdaki elemen için seviye sayısını ve her seviyedeki karşılaştırma sayısını nedir?
- Toplam karşılaştırma sayısının karmaşıklığı nedir?

a-)



b-) Ağacı 6 seviyelidir.

$$\begin{array}{ll} \text{c-) } 0.\text{seviye} \rightarrow 18 & 3.\text{seviye} \rightarrow 11 \\ 1.\text{seviye} \rightarrow 17 & 4.\text{seviye} \rightarrow 3 \\ 2.\text{seviye} \rightarrow 15 & 5.\text{seviye} \rightarrow 0 \end{array}$$

d-) Toplam karşılaştırma sayısı = 66 dir.

e-) $1 + \log n$

f-) Toplam karşılaştırmanın karmaşıklığı $\rightarrow n \log n$

* Her seviyedeki karşılaştırma sayısı
veya
seviyelerdeki karmaşıklık \rightarrow

Örnek : $-2, 11, -6, 13, -4, 2, 5, 1, -3$

$-3, 4, -2, -1, 6$

a-) Bu algoritmayı brute-force ile yapıp oradakini verin.

b-) Data verenin bir algoritma versa yapıp oradakini verin.

a)

```
int maxSubsum (A) {  
    maxsum = 0;  
    for (i=0; i<n; i++) {  
        for (j=i; j<n; j++) {  
            this sum = 0;  
            for (k=i; k<=j; k++)  
                this sum += A[k];  
            if (this sum > maxsum)  
                maxsum = this sum;  
    }  
    return maxsum;  
}
```

$O(n^3)$

b)

Örnek : Verilen bir tam sayı dizisinde elemanları toplamı maks. olan alt diziyi bularak toplamı geri döndüren bir algoritma yazılacaktır.

ALGORİTMALARIN TASARIM STRATEJİLERİ

- 1-) Rekursif
- 2-) Backtracking (geriye izleme)
- 3-) Divide and conquer (böl ve yönet)
- 4-) Dinamik Programlama
- 5-) Greedy (Aç gözlu yaklaşım)
- 6-) Brute-force (Kaba-kuvvet)
- 7-) Branch and bound (Dal ve sınır)
- 8-) Heuristic (Sergisel)
- 9-) İteratif
- 10-) Decrease and conquer
- 11-) Transform and conquer



Seçilme Kriterleri

- problemin özelliklerine göre
- " boyutuna "
- küləniləbilir köynəklər

Problem Tipleri

- arama
- sıralama
- dizi işleme
- graf problemleri
- kombinasiyon "
- geometrik "
- sayısal "



Vari Yapıları

- list (liste)
 - array
 - string
 - linked list
- stack (yığın)
- queue (kuyruk)
- graph (graf)
- tree (ağaç)
- set and dictionary

1-) Rekursif Algoritmalar : Doğrudan veya dolaylı olarak kendini çağırın yordandır.

- problemleri daha basit alt problemlere bölerek çözecektir.
- alt problemlerde kendi içinde başka alt problemlere bölme.
- alt problemler çözülecek kadar kılınca bölme işlemi durur.

2-) Backtracking (Geriye Izleme) Algoritması :

Çalışma şekli ; amaca ulaşmaya kadar doğru seçimleri deperföndürmektedir.

Örnek ; Bir labirent doldurmak için sağa ve sola 2 seçim varsa ve sol tarafta seboldiden sonra çıkışa ulaşmamışsa geriye dönüp sağ taraftan seçmesi işlemidir.

3-) Böl ve Yönöt Yöntemi (Divide and conquer) :

- problemi küçük parçalara bööl.
- her bir parçayı bağımsız şekilde çöz.
- parçaları birleştirerek ana problemin çözümüne ulaş.

4-) Diferansiyel Programlama :

Böl ve yönöt yöntemine benzer olarak ; alt problemlerin çözümlerini birleştirerek gözleme gitme montigi sahip olup alt problem tekrarı varsa bunlardan bir tanesi kullanılır, ve bu çözüm diğer tekrarlarında kullanılır.

5-) Greedy (Ağıraklı) Yaklaşımı :

Tanım amac ; En iyi sonuc elde etmek için en iyi ana adımları çözümlerini seçmeye yönelik bir yöntem old. için en gözleme yaklaşım denir.

6-) Kaba-Kuvvet (Brute-Force) Algoritması :

En ilkel şekilde her yolu deneyerek yapılış işlemidir. En düşük performanslı algoritmalarдан biridir.

7-) Branch and bound (Dal ve sınırlı) Algoritması :

Optimizasyon problemini çözmek için kalkan bir ağacı budanata teknigidir. En iyi yolu bularak en fazla kar sağlayan algoritmasını doldurmaktadır.

- Branch olarak odaklanılar her dal farklı bir adım göstermektedir. Amac ; her dal üzerinde analiz yaparak en iyi sonuc elde etmektedir.

Ağacı yapısı bu algoritmanın çalışma montigini en iyi şekilde tanımır.

8-) Sergisel (Heuristic) Algoritmalar :

Uygulanan yöntemin doğruluğunu ispat edilmesi gerekmektedir. Tek istenen karmaşık bir problemi daha basit hale getirmesi veya algoritmanın şartının sağlı bir sonuc bulabilmesidir.

9-) İteratif :

Recursive Algoritmaların Komsaklılığı

Tekrarlı Bağıntılar :

$$\left. \begin{array}{l} a(n) = 5n + 4 \\ a(1) = 9 \end{array} \right\} \quad a(n+1) = a(n) + 5$$

$$\left. \begin{array}{l} a(n) = 2n! \\ a(1) = 2 \end{array} \right\} \quad a(n+1) = a(n)(n+1)$$

$$\left. \begin{array}{l} a(n) = \sum_{i=1}^n n \\ a(1) = 1 \end{array} \right\} \quad a(n+1) = a(n) + n + 1$$

Tekrarlı Bağıntıların Çözümü

- Master Teoremi Yöntemi
- Yerine Kayma "
- Recursive Tree "

(1) Master Yöntemi :

$a > 1, b > 2$ iain;

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + cn$$

$$T(n) = \begin{cases} \Theta(n^i \cdot \log_b^n), & a = b^i, i = \log_b^a \\ \Theta(n^{\log_b^a}), & a > b^i, i < \log_b^a \\ \Theta(n^i), & a < b^i, i > \log_b^a \end{cases}$$

Örnekler

$$(1) T(n) = 8 \cdot T\left(\frac{n}{2}\right) + n^2$$

$$\begin{array}{ll} a = 8 & \\ b = 2 & a = b^i \\ i = 2 & 8 > 2^2 \end{array} \quad \Rightarrow \quad n^{\log_2^8} = \Theta(n^3) \quad \underline{\Theta(n^3)}$$

$$\textcircled{2} \quad T(n) = T\left(\frac{3n}{7}\right) + 1$$

$$\begin{array}{l} a=1 \\ b=2 \\ i=0 \end{array} \quad \begin{array}{l} a \stackrel{?}{=} b \\ i = \left(\frac{3}{7}\right)^0 \end{array}$$

$$n^0 \cdot \log_{\frac{3}{7}}^n$$

$$\underline{S(\log_{\frac{3}{7}}^n)}$$

$$\textcircled{3} \quad T(n) = T\left(\frac{n}{2}\right) + n$$

$$\begin{array}{l} a=1 \\ b=2 \\ i=1 \end{array} \quad \begin{array}{l} a \stackrel{?}{=} b \\ i = 2^1 \end{array}$$

$$n^1$$

$$\underline{\Theta(n)}$$

$$\textcircled{4} \quad T(n) = 16 \cdot T\left(\frac{n}{4}\right) + n$$

$$\begin{array}{l} a=16 \\ b=4 \\ i=1 \end{array} \quad 16 > 4^1$$

$$n^{\log_4^{16}} \quad n^2$$

$$\underline{\Theta(n^2)}$$

~~(5)~~

$$\textcircled{5} \quad T(n) = \underbrace{T\left(\frac{n}{2}\right)}_{\log n} + \underbrace{T\left(\frac{n}{4}\right)}_{\log_4^n} + \underbrace{T\left(\frac{n}{8}\right)}_n + n$$

$$\begin{array}{l} a=1 \\ b=2 \\ i=0 \end{array}$$

$$\begin{array}{l} a=1 \\ b=4 \\ i=0 \end{array}$$

$$\begin{array}{l} a=1 \\ b=8 \\ i=1 \end{array}$$

$$n^0 \cdot \log_2^n \quad n^0 \cdot \log_4^n$$

$\log_2^n + \log_4^n + n \rightarrow n$ nin yanında \log i how edildiginden

$$\underline{\Theta(n)}$$

$$\textcircled{6} \quad T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n \sqrt{n} \quad ?$$

$$\begin{array}{l} a=9 \\ b=3 \\ i=\frac{n}{3} \end{array} \quad \begin{array}{l} g \stackrel{?}{=} 3^{\frac{n}{3}} \\ 9 > \dots \end{array}$$

$$n^{\log_3^9} \Rightarrow n^2$$

$$\underline{\Theta(n^2)}$$

$$\textcircled{7} \quad T(n) = T\left(\frac{2n}{3}\right) + 1$$

$$\begin{array}{l} a=1 \\ b=\frac{2}{3} \\ i=0 \end{array}$$

$$1 = \left(\frac{2}{3}\right)^0$$

$$n^0 \cdot \log_{\frac{2}{3}}^n$$

$$\log_{\frac{2}{3}}^n$$

$$\Theta(\log_{\frac{2}{3}}^n)$$

\downarrow
 $\log n$
mi
acaba?

~~(8)~~

~~3+3+2+~~

$$⑧ T(n) = 3 \cdot T\left(\frac{n}{2}\right) + n^2$$

$$\begin{array}{l} a=3 \\ b=2 \\ i=2 \end{array} \quad \begin{array}{l} 3 \stackrel{?}{=} 2^2 \\ 3 < 4 \end{array} \quad n^2 \Rightarrow \underline{\Theta(n^2)}$$

$$\boxed{n^{1+\log_2 3}} \quad 3 > 2$$

$$⑨ T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2 \rightarrow n^2 \cdot \log_2 n$$

$$\begin{array}{l} a=4 \\ b=2 \\ i=2 \end{array} \quad \begin{array}{l} n^{1+\log_2 2} \\ b=4 \end{array} \quad n^2 \cdot \log_2 n \Rightarrow \underline{\Theta(n^2 \log n)}$$

$\log n$ ifadesi redilir $\underline{\Theta(n^2)}$

$$⑩ T(n) = T\left(\frac{n}{2}\right) + 4 \cdot T\left(\frac{n}{4}\right) + n^2$$

$$\begin{array}{l} a=1 \\ b=2 \\ i=0 \end{array} \quad \begin{array}{l} (\log n) \\ 1=2^0 \end{array} \quad \begin{array}{l} + \overbrace{n^2} \\ a=4 \\ b=4 \\ i=2 \end{array} \quad \begin{array}{l} n^2 \\ 4 < 4^2 \end{array} \quad \begin{array}{l} \underline{n^2 + \log n} \\ \underline{\Theta(n^2)} \end{array}$$

$$n^0 \cdot \log_2^n$$

$$⑪ T(n) = 3 \cdot T\left(\frac{n}{4}\right) + n \log n \rightarrow \text{yerine kayırmaya örmek gerekliyormus}$$

$$\begin{array}{l} a=3 \\ b=4 \\ f(n)=n \log n \end{array} \quad \begin{array}{l} n \log^3 \frac{3}{4} n^1 \\ n \log n > n^1 \end{array} \quad ?$$

bu redende
cevap $\rightarrow n \log n$

$$⑫ n \geq 1 \text{ } 4^i \text{ en kuuti ise}$$

$$T(n) = 4 \cdot T\left(\frac{n}{4}\right) + \underline{n \log n} \Rightarrow \underline{\Theta(n \log^2 n)}$$

$$\begin{array}{l} a=4 \\ b=4 \end{array} \quad \begin{array}{l} n \log n \cdot \log n \end{array}$$

Örnek: $\sum_{i=1}^n \log i + 2^i \rightarrow$ ifadesinin çölüp komsılığını bulın.

$$\log 1 + 2 + \log 2 + 4 + \log 3 + 8 + \dots$$

$$2+4+8+\dots+2^n$$

$$\log 1 + \log 2 + \log 3 + \dots + \log n$$

② Terine Kayma (Substitution) Yöntemi :

ARAMA ALGORİTMALARI #

(1) Lineer (Döngüsel) Arama:

17	23	6	55	31	35	7	21
----	----	---	----	----	----	---	----

Önemli: verilen dizi içerisinde 7 aranırsa; liste sırasıyla kontrol edilir.
Listeyi gezerek 7 sayısı bulunur.

Avantajı: uygulanması oldukça kolaydır.

Desavantajı: çok yavaştır.

Kodu:

```
int LinearSearch (int[] dizi, int n, int hedef) {
    for (int i=0; i<n; i++) {
        if (dizi[i] == hedef) {
            return i;
        }
    }
}
```

Karmaşıklığı:

En iyi $\Rightarrow O(1)$

Ortalama $\Rightarrow O(n)$

En kötü $\Rightarrow O(n)$

Lineer arama için en iyi durum:
aranan elementin listenin başında
olmalıdır

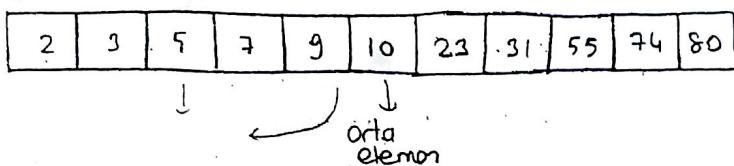
En kötü durum ise; listenin sonunda
olmalıdır.

② Binary (ikili) Arama :

- Yapı dairek böl-yöneti yaklaşımının uygulanmasıdır.
- Dizi sıralı olmalıdır.

Mantığı ;

- aranacak dizinin tam ortasına bak.
- aranan bulunduysa bit.
- aranan değer ; orta elemenden kucakla , kucaklarat (sol taraftan) kontrol et
- aranan değer ; kucakla , kucak (sağ) taraftan kontrol et.



aranan=3 ise sağa git , orta elementi bul deon et istene

aranan=31 ise solo "

Kodu :

```
int [] dizi ;
int boyut ;

public boolean binarySearch (int aranclar)
{
    int low = 0
    int high = boyut - 1;

    while (high >= low) {
        int orta = (low+high)/2;
        if (dizi[orta] == aranclar) {
            return true;
        }
        else if (dizi[orta] < key) {
            low = orta + 1;
        }
        else {
            high = orta - 1;
        }
    }
}
```

Karşıtlıklar

En iyi durum $\Rightarrow O(1)$

Ortalama $\Rightarrow O(\log n)$

En kötü $\Rightarrow O(n)$

SIRALAMA ALGORITMALARI

(1) Kabarcık (Bubble Sort) Sıralama :-

- Sıralanacak eleman kümnesinden ilk eleman silinir.
- Kendinden sonraki büyükse yer değiştirilir.
- Sonraki elemene gelir ve devam edilir.
- Dizi sonra verildiğinde en büyük elemende sonda yer alır.

Kodu :

④ public static void bubbleSort (int[] dizi) {

```

    int temp;
    for (int i=1; i< dizi.length; i++) {
        for (int j=0; j< dizi.length-1; j++) {
            if (dizi[j] > dizi[j+1]) {
                temp = dizi[j];
                dizi[j] = dizi[j+1];
                dizi[j+1] = temp;
            }
        }
    }
}

```

⑤ func bubbleSort (var a os array)

for i from 1 to N

for j from 0 to N-1

if a[i] > a[i+1]

swap (a[i], a[i+1])

end func

Karmaşıklıklar

en iyi durum $\Rightarrow \Theta(n)$

Ortalama " $\Rightarrow \Theta(n^2)$

en kötü " $\Rightarrow \Theta(n^2)$

Bubble Sort için;
En kötü durum En iyi durum

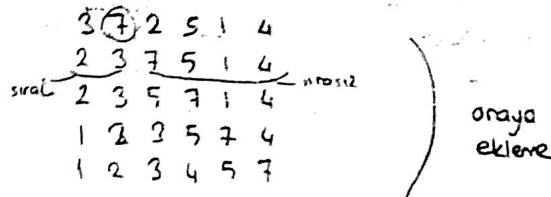
```

int BubbleSort ( int [ ] dizi ) {
    int temp;
    for ( i := 1; i < dizi.length(); i++ ) {
        for ( j = 0; j < dizi.length() - 1; j++ ) {
            if ( dizi[j] > dizi[j+1] ) {
                temp = dizi[j];
                dizi[j] = dizi[j+1];
                dizi[j+1] = temp;
            }
        }
    }
}

```

② Eklenteli (Insertion Sort) Sıralama :

- Sıralanacak element kütmesinden 2. element referans alınır.
- Kendinden önceki elementlere karşılaştırılıp büyük olanı sopa kaldırma istenidir.



Kodu : public static void insertionSort (int[], int n)

```
{  
    for (int i = 1; i < n; i++) {  
        for (int j = i; j > 0; j--) {  
            if (a[j] > a[j - 1]) {  
  
                int temp = a[j];  
                a[j] = a[j + 1];  
                a[j + 1] = temp;  
            } else {  
                break;  
            }  
        }  
    }  
}
```

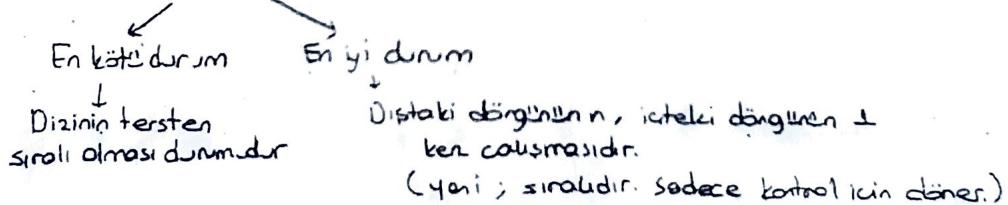
Kompleksiteleri :

En iyi durum $\Rightarrow O(n)$

Ortalama " $\Rightarrow O(n^2)$

En kötü " $\Rightarrow O(n^2)$

insertion Sort için :



dıştaki döngü → kontrol etmek için.

İçteki " → yerleştirip, deşifirmek için.

③ Seçme (Selection Sort) Sıralaması :

Mantığı : en küçük değerin başa getirildiği sıralamadır.

- Başlangıçta dizinin ilk öğesi en küçük kabul edilir. (tobu geçicidir.)
- Dizinin ilk elemanı diğer elemanlara karşılaştırılıp en küçük hangisi ise bu baştan (birinci, ikinci, ...) sıraya yerleştirilir.

3 9 4 7 1
1 9 4 7 3
1 3 4 7 9 //

Kodu :

④ public static void selectionSort (int[] dizi, int n) {
 int temp;
 int enKucuk;

 for (int i = 0; i < n - 1; i++) {
 enKucuk = i;
 for (int j = i + 1; j < n; j++) {
 if (dizi[j] < dizi[enKucuk]) {
 enKucuk = j;
 }
 }
 temp = dizi[i];
 dizi[i] = dizi[enKucuk];
 dizi[enKucuk] = temp;
 }
}

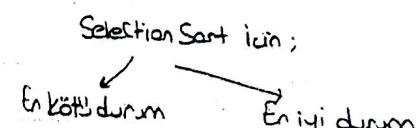
⑤ for i ← 0 to n-1 do
 min ← i
 for j ← i+1 to n
 if A[j] < A[min]
 min ← j
 swap (A[j], A[min])

Karmaşıklıkı

En iyi durum $\Rightarrow \Theta(n^2)$

Ortalama " $\Rightarrow \Theta(n^2)$

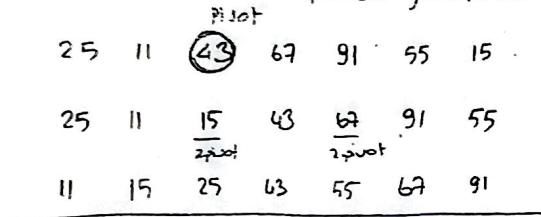
En kötü " $\Rightarrow \Theta(n^2)$



④ Hızlı (Quick Sort) Sıralama :

Mantığı ; aynı veri türne göre bölgelerin arayışına göre sıralanmasıdır.

- sayı dizisinden herhangi bir eleman pivot seçilir.
- pivotın küçük olaları pivot öncesi, büyük olalar pivotin arkasında gecerek şekilde yerlesirler.



Kod :

④ void quickSort (int dizi[], int sol, int sağ) {

```

int i = sol, j = sağ;
int temp;
int pivot = dizi[(sol+sağ)/2];
while (i <= j) {
    while (dizi[i] < pivot)
        i++;
    while (dizi[j] > pivot)
        j--;
    if (i <= j) {
        temp = dizi[i];
        dizi[i] = dizi[j];
        dizi[j] = temp;
        i++;
        j--;
    }
}
    
```

④ quicksort (A, p, r)

```

if (p < r)
    q ← partition (A, p, r)
    quicksort (A, p, q)
    quick sort (A, q+1, r)
    
```

Karmaşıklıklar :

En iyi durum $\Rightarrow (n \log n)$

Ortalama " $\Rightarrow (n \log n)$

En kötü " $\Rightarrow n^2$

Quick Sort için :

En kötü durum

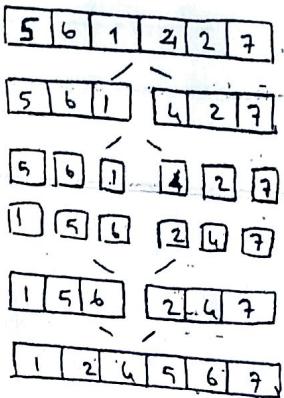
Seçilen pivotun
çok büyük veya
çok küçük olması
durumunda olur.

En iyi durum

Seçilen pivotun
sağında büyük
solunda küçüklerin
olması durumudur.

⑤ Merge Sort Sıralaması :

Mesnitiği, sıralı olmayan dizigi ortadan eşit olarık 2 alt listeeye böler. Alt listeleri kendi aralarında sıralayıp birleştirir.



Kodu:

```

    ⑧ void merge(int alt, int üst, int m) {
        int i, j, k;
        for(i=alt; i<=üst; i++)
            b[i] = a[i];
        i=alt; j=m+1; k=alt;
        while(i<=m && j<=üst)
            if(b[i] <= b[j])
                a[k++] = b[i++];
            else
                a[k++] = b[i++];
        while(i<=m)
            a[k++] = b[i++];
    }

```

⑨

```

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

Komplekslikler

En iyi durum $\Rightarrow O(n \log n)$

Ortalama " $\Rightarrow O(n \log n)$

En kötü " $\Rightarrow O(n^2)$

MergeSort için;

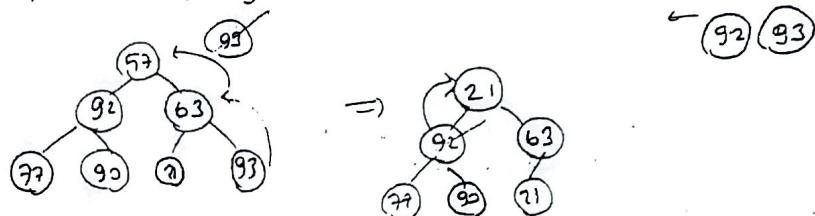
Dizi tersten de sıralı olsa, sıralıda olsa tüm adımlar işlenir. Yani; en iyi ve en kötü durum aynıdır.

(6) Yığınlama (HeapSort) Sıralaması :

Mantığı ; özel ağacı yapısıyla eleman kümelerindeki değerlerin kümelerne ismini yollamaya ortaya çıkan algoritmadır.

- Verilen diziyi ağaca yerlestiriyoruz.
- Daha sonra en büyük elemeni yerdeğiştirek köke taşıyorumuz.
- Kökü atıp , en büyük elemen kabul ediyoruz , sonra en küçük en üstे taşıyorumuz .
sonra yine deşisme yapıp atıyoruz.

57 92 63 77 90 21 93



en büyük en üsteki taşıyip siliyorumuz listenin en sonuna yerleştirmeyiz.
daha sonra en küçük i köke atıp devam ediyoruz.

Kodu :

```
public void HeapSort (int []A) {
    int temp;
    Build heap (A);
    for (int i = A.length - 1; i >= 0, --) {
        temp = A[0];
        A[0] = A[i];
        A[i] = temp;
        heapSize = heapSize - 1;
        heapify (A, 0);
    }
}
```

Karmaşıklıkları

En iyi durum $\Rightarrow O(n \log n)$

Ortalama " $\Rightarrow O(n \log n)$

En kötü " $\Rightarrow O(n^2)$

ortalama

$O(n \log n)$

⑦ Sayısal (Counting Sort) Sıralama :

Mənətiq : Verinin hərəkətə sərhədindən tətbiq olunan alqoritmlərdən biridir.
Bəsiçə sıralanacak dizinin her sayından kənara old. fərqli bir dizidə sayı.

$\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 7 & 2 & 9 & 6 & 1 & 3 & 7 \end{matrix}$

dizi indexi : 0 1 2 3 4 5 6 7 8 9

dizi degeri : 0 1 1 1 0 1 1 2 0 1

Kodu : countingSort (A,B,k)

```
for i ← 0 to k
    do C[i] ← 0
    for j ← 1 to length[A]
        do C[A[j]] ← C[A[j]] + 1
    for i ← 1 to k
        do C[i] ← C[i] + C[i-1]
    for j ← length[A] down to 1
        do B[C[A[j]]] ← A[j]
```

Karmaşıklıkları

En iyi durum $\Rightarrow O(n+k)$
Ortalama " $\Rightarrow O(n+k)$
En kötü " $\Rightarrow O(n+k)$

⑧ Basamaga göre (Radix Sort) Sıralama :

Mənətiq : Sayıları basamaklarına göre sıralayan bir alqoritmdir. Basamaga göre sıralma 2'ye ayndır
↳ En azanti basamaga göre, en azanti basamaga göre.

57 43 213 24 44 102 70 37 111 23

Birler basamagini göre ; 70 111 102 43 213 23 24 44 57 37

Onlar " " ; 102 111 213 23 24 37 43 44 57 70

Yüzler " " ; 23 24 37 43 44 57 70 102 111 213

Radix-Sort (A,d)

for i ← 1 to d d (digit) \rightarrow sabit sayı

do use a stable sort to sort array A on digit

En iyi durum $\Rightarrow O(n+k)$

Ortalama " $\Rightarrow O(d(n+k))$

En kötü " $\Rightarrow O(d(n+k))$

olur.

⑨ Kova (Bucket Sort) Sıralama :

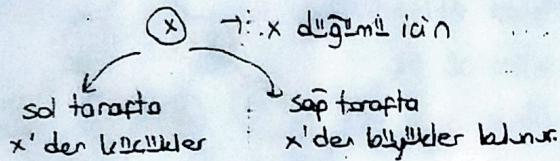
Mənətiq : sıralanacak bir diziyi parçalara ayıraqla sınırlı sayıda kovalar yada sepetlərə atan bir sıralama alqoritmasıdır.

Aynanın işləminin ardından her kova kendi icinde ya fərqli bir alqoritma kullanılarak yada kova sıralamasını özyinişli olaraq əsaslı sıralanır.

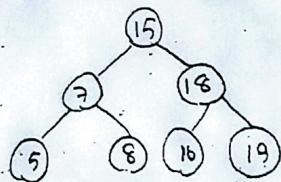
AGACLAR (TREES) #

Binary Search Tree $\rightarrow O(n)$

- Düğümle rin en fazla 2 çocuğu satılıp oldğuğunu yapısıdır

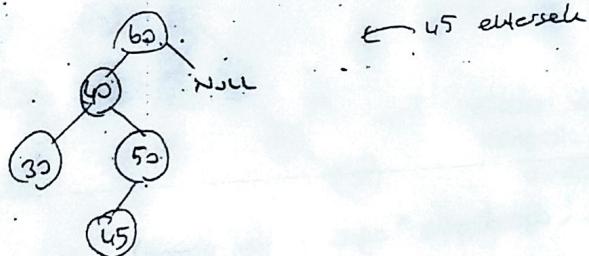


Arama :

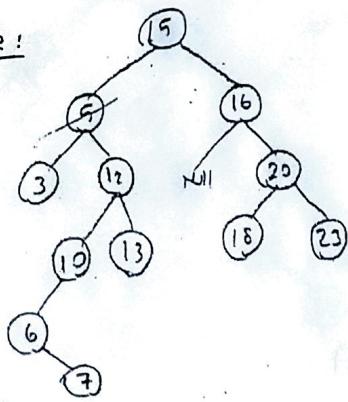


16'yi ararsak
köke soma klden kllik sağa
da bu soma sola sellinde
ararır.

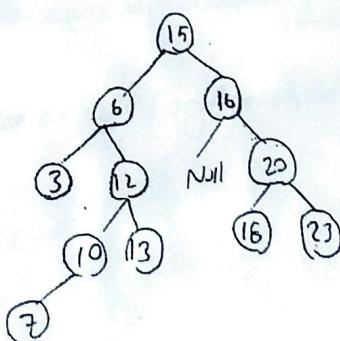
Ekleme :



Silme :



5' : sil
14
5 in successor 6'dır.



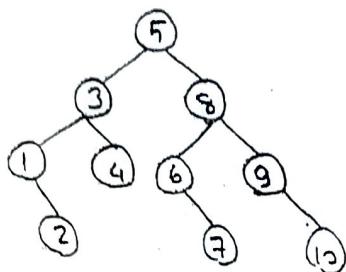
	Array (unsorted)	Array (sorted)	BST (balanced)
Search \Rightarrow	$O(n)$	$O(bgn)$	$O(\log n)$
insert \Rightarrow	$O(1)$	$O(n)$	$O(\log n)$
Remove \Rightarrow	$O(n)$	$O(n)$	$O(\log n)$

Ağacın Döşeme

① Preorder : Kök - Sol - Sağ

② Inorder : Sol - Kök - Sağ

③ Postorder : Sol - Sağ - Kök



Preorder →

Inorder →

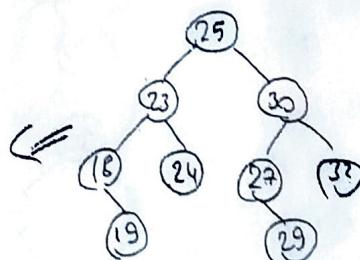
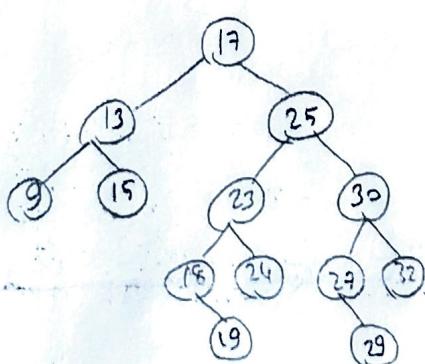
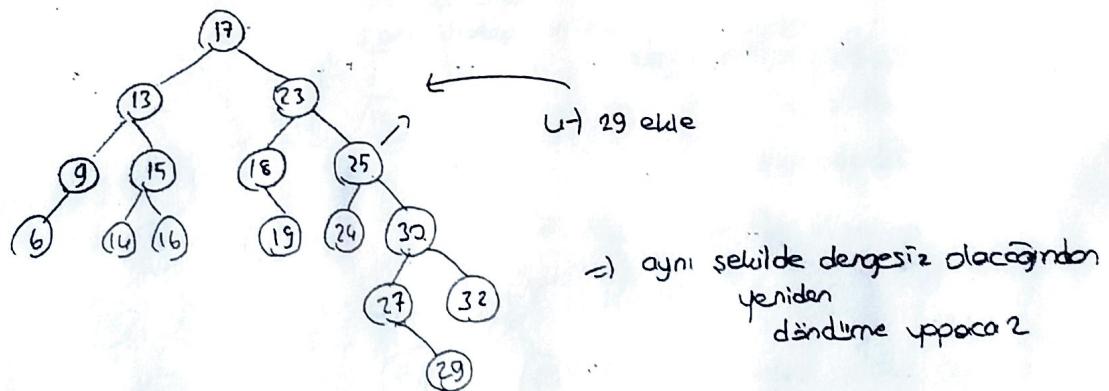
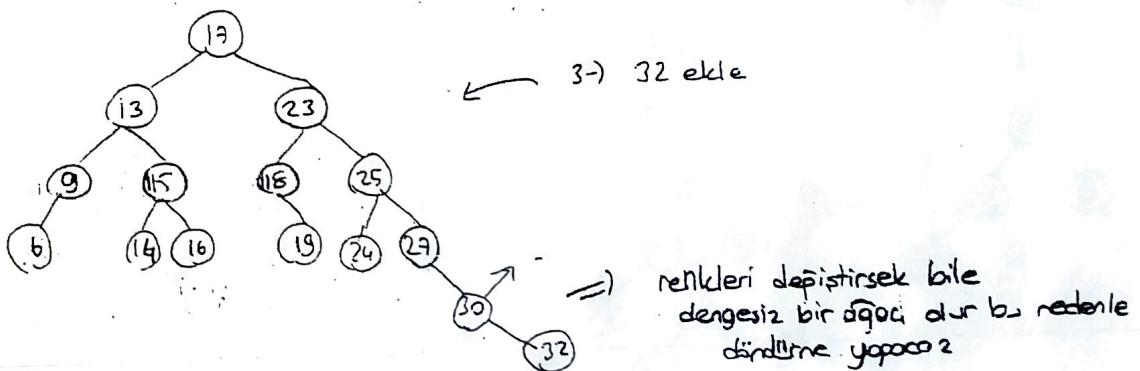
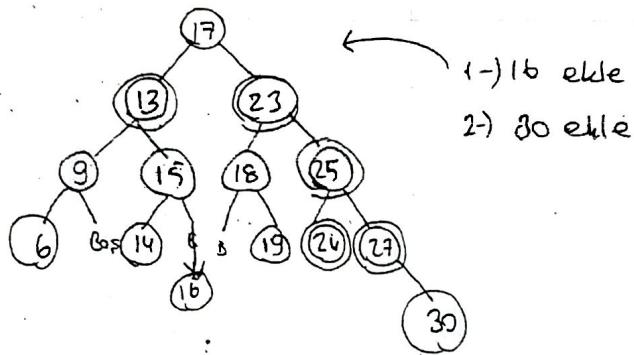
Postorder →

İkili Arama Ağacı ;

```
void ikiliAramaAgaci (agacduzugum * agac , veri aranan) {  
    if (agac == NULL)  
        return ;  
  
    if (aranan == agac->anahtar)  
        yazEkrana (agac);  
  
    if (aranan < agac->anahtar)  
        ikiliAramaAgaci (agac->sol) ;  
  
    else  
        ikiliAramaAgaci (agac->sag) ;  
}
```

DENGELİ AĞACLAR

① Red-Black Tree :



→ 85

→ 15

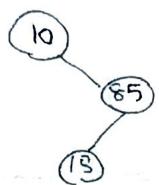
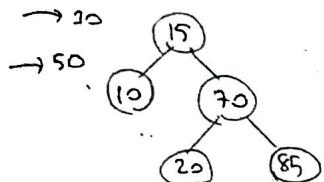
→ 70

→ 20

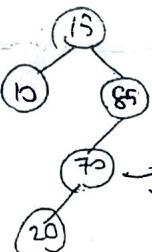
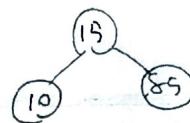
→ 60

→ 20

→ 50

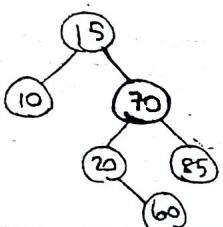


→ döndür

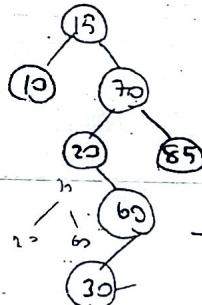


renk depistirmek yeteri olmaz
döndürme yaparız

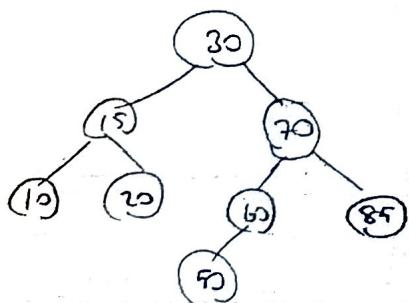
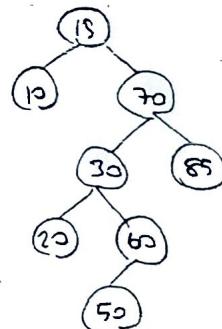
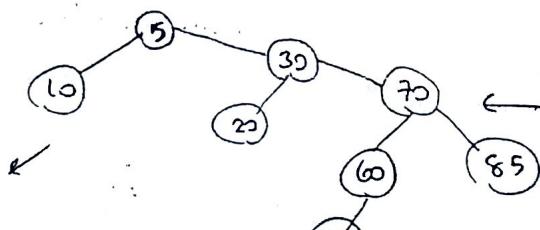
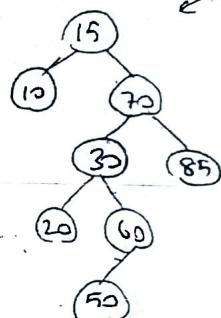
↓



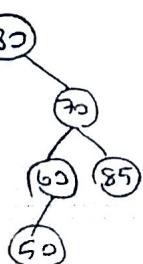
→

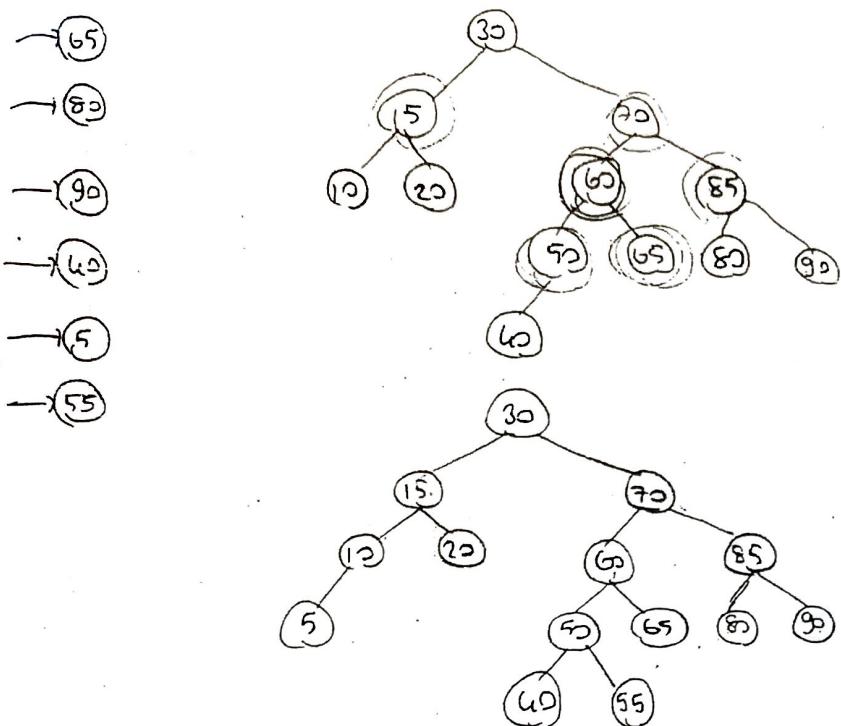


→ renk depistirmek
yeteri olma 2

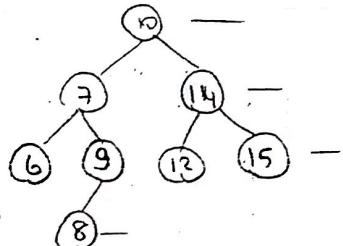


→



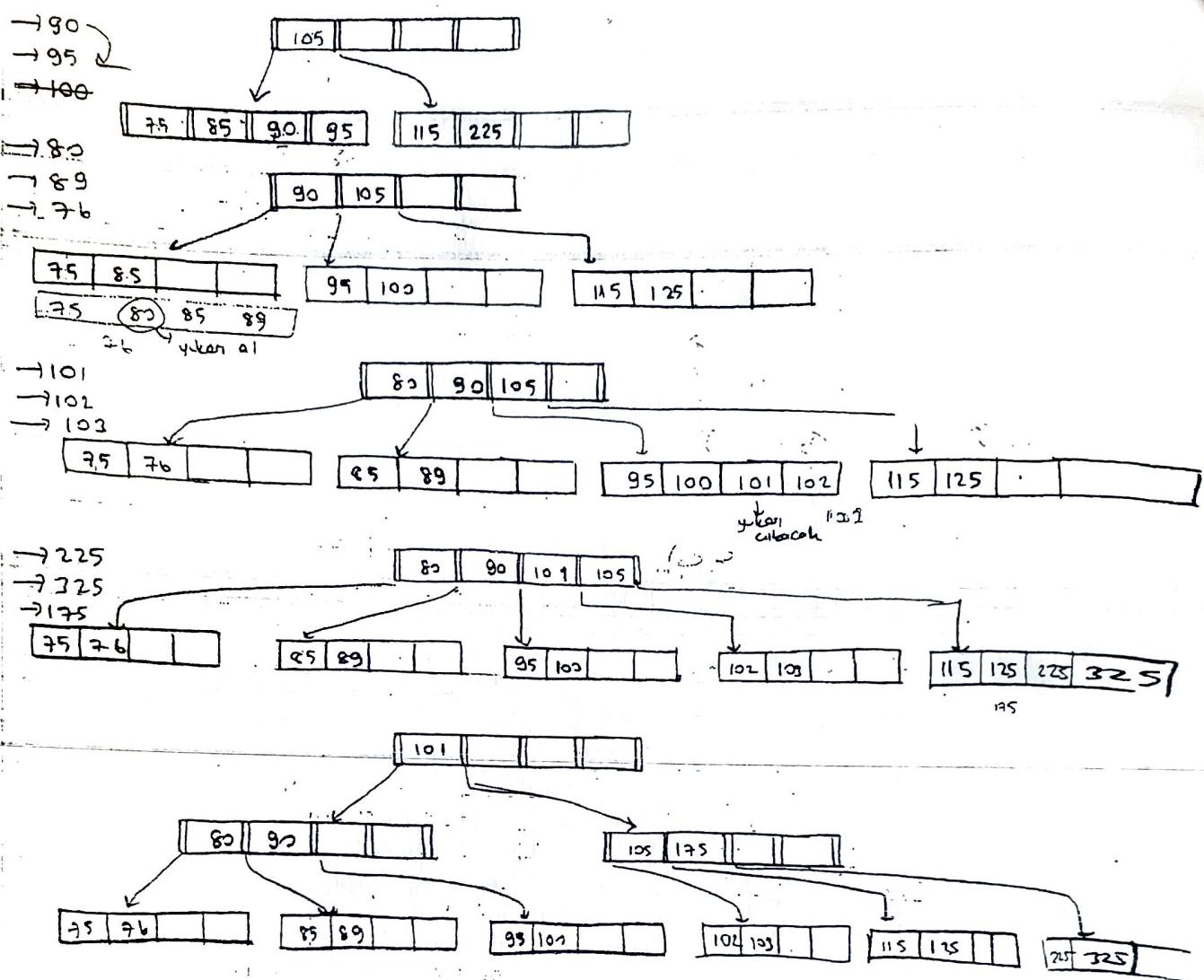


(2) AVL Tree :



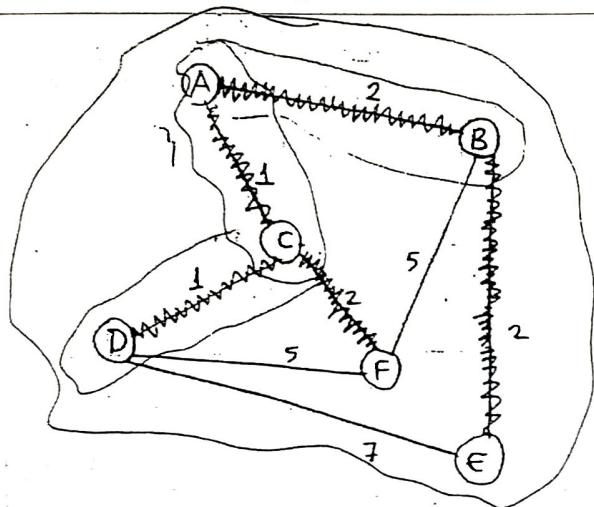
- * Ağacın seviyeleri arasında en fazla $+1, -1$ fark olmak zorundadır.
- * Ağac, ikili ağacı kurallına uyumak zorundadır ; büyük - küçük ilişkisi olmalıdır.
- * Ağac seviye farkı kurallını sağlarsa oncağ ; ikili ağacı kurma uyumazsa ; AVL tree olmaz.

③ B-Tree



Minimum Spanning Tree

① Prim Algoritması



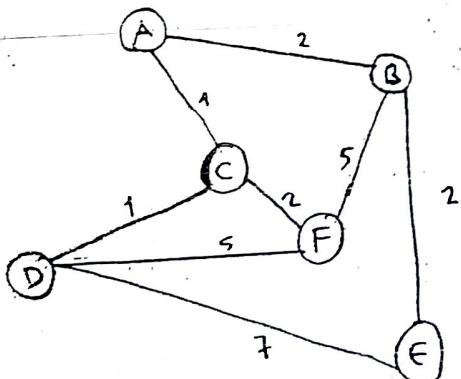
- Sanal kapsama alanı tonim konur.
- En küçük deperdi kenarları ilerler
- Yeni bir kapsama alanına girinir.
- Bütün düğümleri kapsayana kadar ilerler.

düğüm listesi

ACDBEF

* Eşit ırzılıktaki bulunabilir, ancak
birkaç düğüm bulunmaz.

② Kruskal Algoritması :



- Kenarları sırayla ek basır
- Eşit ırzılıktaki sıra önemlidir.
- Kenarları sırasıyla dolarır.

CD = 1

AC = 1

BE = 2

CF = 2

AD = 2

BF = 5

DF = 5

DE = 7

İlgilenen Listesi

CDA BEF

CDABEF

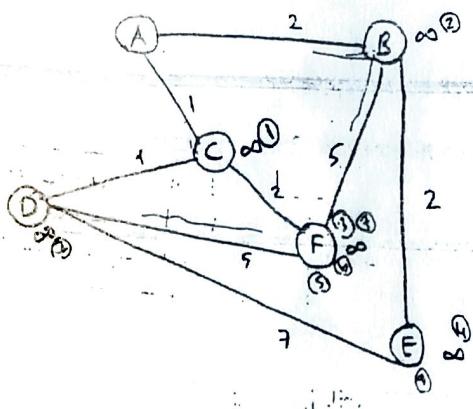
↳ gezilmiş düğüme
tekrar gidilmez.

* prim, herhangi bir noktadan başlar, kapali döngü oluşturmadan tüm
düğümlere ulaşır.

* Kruskal ise, en küçük ırzılıktaki yerden başlar ve kendisi içinden
kontrol eder, kusadan başlar

birbirinden farklı budur.

③ Dijikstra Algoritması



- ilk önce hep ∞ dır.
- daha sonra komşuların maliyetini hesapları.
- Komşulara geçilir, komşuların gecenken maliyet hesabı yapılır.
- Maliyet kılavuzla o yol seçili.

Kruskal Sürde Kodu ←

Kruskal (G) :

```

 $A = \emptyset$ 
for each  $v \in G, v;$ 
    MAKE-SET ( $v$ )
for each  $(u,v)$  ordered by
    weight  $(u,v)$ , increasing;
if FIND-SET ( $u$ ) ≠ FIND-SET ( $v$ )
     $A = A \cup \{(u,v)\}$ 
    UNION ( $u,v$ )
return  $A$ ;
```

Prim Sürde Kodu ←

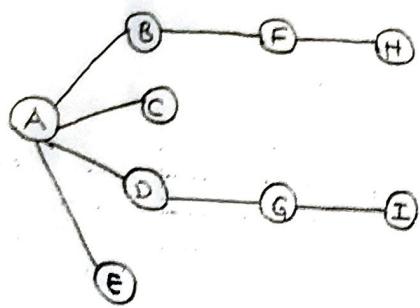
```

MST-PRIM ( $G, w, r$ )
for each  $v \in [G]$ 
do key [ $v$ ] ←  $\infty$ 
     $\pi[v] \leftarrow \text{NIL}$ 
key [ $r$ ] ← 0
 $\Theta \leftarrow V[G]$ 
while  $\Theta \neq \emptyset$ 
do  $v \leftarrow \text{EXTRACT-MIN}(\Theta)$ 
    for each  $v \in \text{Adj}[v]$ 
        do if  $v \notin \Theta$  and  $w(v,v) < \text{key}[v]$ 
            then  $\pi[v] \leftarrow v$ 
            key [ $v$ ] ←  $w(v,v)$ 
```

function Prim (G, g, π)

T : kırpsayı oğası
 B : eklemevi düğüm
 B ← rasgele bir düğüm
while $B \neq N$ do
 $e = (u,v)$ şeklinde
en hafif aynı türkiyle ekle
 u, B nin elementi olsun
ve N/B nin elementi olsun
 $T \leftarrow T \cup \{e\}$
 $B \leftarrow B \cup \{v\}$
end while
return T

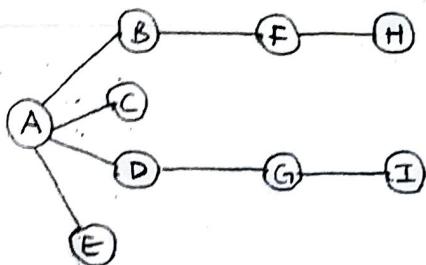
DFS ←



A
AB
ABF
ABFH
ABF
AB
A
AC
A
AD
ADG
ADGI
ADG
AD
A
AE
A

Zigaret et ;
döndüste oksar,
zigaret ettiğin
yerlerि.

BFS ←



A
B
BC
BCD
BCDE
CDE
CDEF
DEF
EF
EFG
FG
G
GH
H
HI
I

Zigaret etip
döndüğün yeri
baştan oksar,

$$(n^2 \log n) \quad T(n) = 4 \cdot T(n/2) + n^2$$

$$(n^2) \quad T(n) = 2 \cdot T(n/2) + n^2$$

$$(n^2) \quad T(n) = T(n-1) + n$$

$$\sum_{i=1}^n \log i + 2i$$

$$\log 1 + 2 + \log 2 + 4 + \log 3 + 6 + \dots + \log n + n$$

$$2 + 4 + 6 + \dots + n \rightarrow \frac{n(n+1)}{2} = \frac{O(n^2)}{2} + O(n)$$

$$\log(n!)$$

11

12