※ ALGORİTMA ANALİZİ ※ (2018 Dönemi)

⇒ SÜRE-HIZ HESAPLAMALARI

⇒ ALGORİTMA TASARIM STRATEJİLERİ

⇒ NOTASYONLAR

⇒ KARMAŞIKLIKLAR

⇒ RECURSİVE ALGORİTMALARI
(Master Metodu, Yerine Koyma, Recursion Tree)

⇒ ARAMA ALGORİTMALARI
(Lineer Search, Binary Search)

⇒ SIRALAMA ALGORİTMALARI
(Insertion Sort, Merge Sort, Quick Sort, Heap Sort, Selection Sort, Bubble Sort, Shall Sort, Radix Sort, Counting Sort)

⇒ DENGELİ AĞAÇLAR
(AVL Tree, B-Tree, Heap Tree, Red-Black Tree, Binary Search Tree)

⇒ MİNİMUM SPANNING TREE
(Prim, Kruskal, Dijkstra)

Helin YARDIMCI

ÖRNEK

A Bilgisayarı

Insertion Sort

1 milyon eleman

1 milyar komut/sn

En iyi yazılmış

Makina kodu

$2n^2$

B Bilgisayarı

Merge Sort

1 milyon eleman

10 milyon komut/sn

Ortalama bir programcı

Yüksek seviyeli bir dil

$50 n \log n$

A ve B bilgisayarı ne kadar sürede işlem yapar?

Çözüm:

A Bilgisayarı

NOT = n = eleman sayısı (veri)

$$\frac{2 \cdot (10^6)^2 \text{ komut}}{10^9 \text{ komut/sn}} = 2000 \text{ sn}$$

B Bilgisayarı

$$\frac{50 \cdot 10^6 \log 10^6 \text{ komut}}{10^7 \text{ komut/sn}} = 100 \text{ sn}$$

! B bilgisayarı A bilgisayarından hızlıdır.

ÖRNEK

Zaman karmaşıklığı $g(n) = 3n^3$

İşlemci hızı 500 mHz

1000 veriyi kaç dk yapar?

Çözüm

$$\frac{3 \cdot (10^3)^3 \text{ komut}}{500 \cdot 10^6} = 6 \text{ sn} = 0{,}1 \text{ dk}$$

NOT

10^3 Hz = kHz

10^6 Hz = mHz

10^9 Hz = GHz

10^{12} Hz = THz

ÖR

A Bilgisayarı	B Bilgisayarı
200 mHz	40 GHz
$4n^2$	n^3
10.000 veri	10.000 veri

Çöz

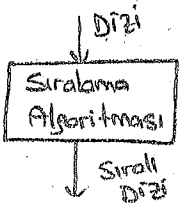
A

$$\frac{4 \cdot (10^4)^2}{200 \cdot 10^6} = 2 \text{ sn}$$

B

$$\frac{(10^4)^3}{40 \cdot 10^9} = \frac{100}{4} = 25 \text{ sn}$$

A bilgisayarı B bilgisayarından daha hızlıdır.



En iyi durum (Best case) $O(1)$
En kötü durum (Worst case) $O(n)$
Ortalama durum (Average case) $O(n)$

ÖR

$5n^2$
500 mHz
500.000 veriyi kaç dkda işler?

çözüm

$$\frac{5 \cdot (5 \cdot 10^5)^2}{500 \cdot 10^6} = \frac{5 \cdot 25 \cdot 10^{10}}{500 \cdot 10^6} = \frac{125 \cdot 10^{10}}{500 \cdot 10^6} = \frac{10^4}{4} = 2500 \text{ saniye}$$

Yaklaşık 41 dk

ÖR

a) $T_{wc} \leq T_{avg} \leq T_{bc}$

b) $T_{wc} \leq T_{bc}$

c) $T_{bc} \leq T_{avg} \leq T_{wc}$

d) Hepsi

Karmaşıklık için hangisi doğrudur?
wc → en kötü avg → ortalama
bc → en iyi
T → zaman

C şıkkı doğrudur

→ Çünkü en kötü durumda en çok zaman harcanır, sonra ortalama da, sonra en iyide

Ör

```
sum 0;
for (i=0; i<n; i++)
sum++;
for (j=0; j<n; j++)
sum++;
```

$T(n) = 1+n+n-1+n$

$= 4n+1$ olur.

$O(n)$ dir.

Eğer süslü parantez yoksa "for" sadece bir alt satırı etkiler.

Ör

```
sum 0;          ---> 1
for (i=0; i<n; i++)   ---> n
for (j=0; j<n; j++)   ---> n(n-1)
sum++;          ---> (n-1)(n-1)
```

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

$= 2n^2-2n+2$

$O(n^2)$ dir.

ALGORİTMA TASARIM STRATEJİLERİ

* İterative
* Recürsive
* Böl - Yönet
* Greedy Alp
* Kaba Kuvvet
* Dinamik Programlama
* Break freking
* Hewretic

} Problemin özelliklerine, Problemin Boyutuna, Kullanılabilir Kaynaklar

Önemli Problem Tipleri

* Arama Problemleri
* Sıralama Problemleri
* Matris/Dizi İşleme
* Graf Problemleri
* Kombinasyon Problemleri
* Geometrik Problemler
* Sayısal Problemler

Temel Veri Yapıları

* Listeler
  - Dizi
  - Bağlı Liste
* Yığın
* Queue
* Graf
* Tree / Ağaç
* Set / Küme

* Problemi alt problemlere böl
* Alt problemleri rekürsif olarak çöz
* Alt problem çözümlerini birleştir.

Merge - Sort

Böl: Her bir $n/2$ elemanlı 2 alt diziye bölünür.

Yönet: Sıralama kullanarak 2 alt dizi sıralanır.

Birleştir: 2 sıralı dizi sıralanarak birleştirilir.

$T(n)$

Merge-Sort (A, p, r) → Metod adı

* Metod adıyla çağırılır
* T ile ifade edilir.

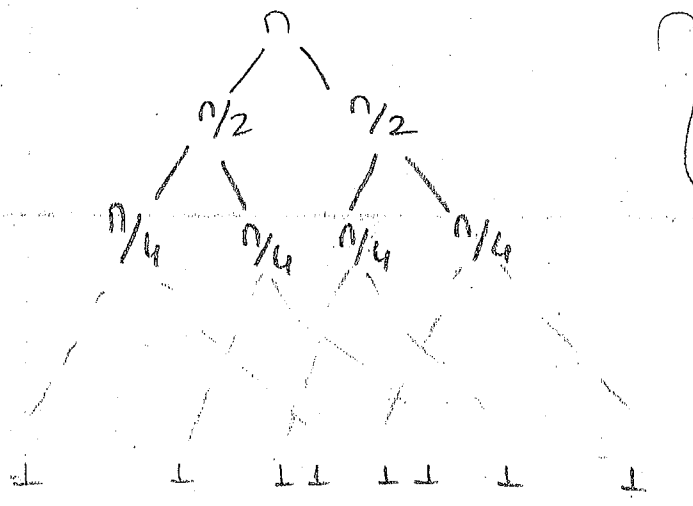
1. if $p < r$ ⟶ $\perp$
2. then / else $q \leftarrow \lfloor (p+r)/2 \rfloor$ ⟶ $\perp$
3. MERGE-SORT (A, p, q) ⟶ $T(\frac{n}{2})$ } metod
4. MERGE-SORT $(A, q+1, r)$ ⟶ $T(\frac{n}{2})$ } metod
5. MERGE-SORT (A, p, q, r) ⟶ $\Theta(n)$ } yönet

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

NOT Master Teoremine göre yorumlayış

$$T(n) \begin{cases} \Theta(1) & , n \leq c \\ a \cdot T\left(\frac{n}{b}\right) + D(n) + C(n) & , \text{diğer} \end{cases}$$

$a \cdot T(\frac{n}{b})$: Yönet, $D(n)$: Böl, $C(n)$: Birleştir



Gerçek karmaşıklığı n log n

Karmaşıklığı log n dir.

Sürekli ikiye bölüp ikili dava yapıyor

* Merge-sort karmaşıklığı n log n dir.

Ödev 1 Soru 1

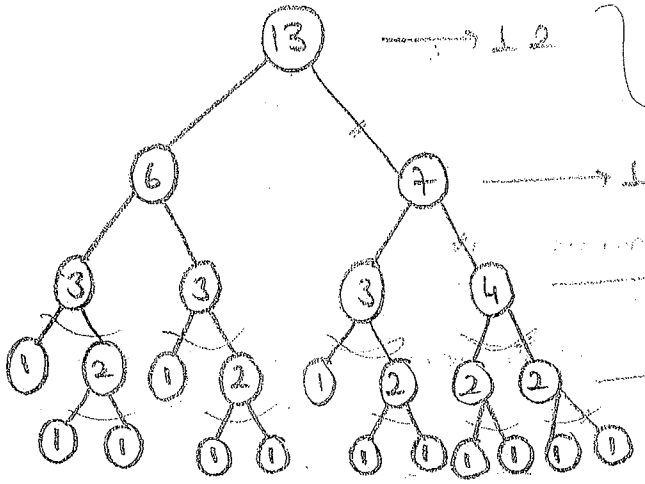
n elemanlı bir dizinin sıralanması için mergesort algoritmi sıralanacaktır. n=13 için recursive tree çizimi.

Çözüm

Kök bir: 13

... Kökü saymıyoruz, 4 seviye

En uzun yol



1. 2

1. 1

9

5

$\log_2^{13} = 3, \ldots = 4$

Karşılaştırma sayıları

(Her düşünü 1 eksiltip say)

- Kaç seviyelidir ? ⇒ 4 seviyelidir.
- Her seviyedeki karşılaştırma sayısı ? ⇒ Toplamda 37

seviyeler: 5, 9, 11, 12

- n 2'nin üssü bir değer ise kaç seviye dir ? ⇒ $\log n$

Sürekli ikiye bölüp düşü için

- Toplam karşılaştırma ?

Her aşamada $\log n$ kadar karşılaştırma yapar. n seviye olduğu için toplam karşılaştırma $n \log n$ olur.

Worse case karşılaştırma olur bu durum

→ Seviye sayısı → $1 + \log n$ / { tek + $\log n$?

ASİMPTOTİK NOTASYONLAR

Big-O Notasyonu: $f(n) = O(g(n))$ için

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

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

Θ - Notasyonu: $f(n) = \Theta(g(n))$ için

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

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

Ω - Notasyonu: $f(n) = \Omega(g(n))$ için

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

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \text{Sabit (constant)} \neq 0$$

Sabit	$O(1)$
Logaritmik	$O(\log n)$
Polilogaritmik	$O(\log n)^c$
Lineer	$O(n)$
Loglineer	$O(n \cdot \log n)$
Karesel	$O(n^2)$
Polinom veya cebirsel	$O(n^c)$
Üssel	$O(c^n)$
Faktöriyel	$O(n!)$
Geokombinal	$O(n^n)$

Karmaşıklık artar.

Ör:

```
for (i=0; i<n ; i++)   → n
for (j=0; j<n ; j++)   → n(n-1)
    S = S+1;           → (n-1)(n-1)
```

$\underbrace{a \cdot n^2 + b \cdot n + c}_{f(n)} = \underbrace{O(n^2)}_{g(n)}$

$2n^2 + 1 = O(n^2)$

Ör

```
for(i=0 ; i<n ; i++)        ——→ c1·n
for (i=0 ; j<n ; j++)       ——→ c2 · (n-1) n
    j = j+1 ;               ——→ c3 · (n-1)(n-1)
```

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

$T(n) = 2n^2 - 2n + 1 = O(n^2)$

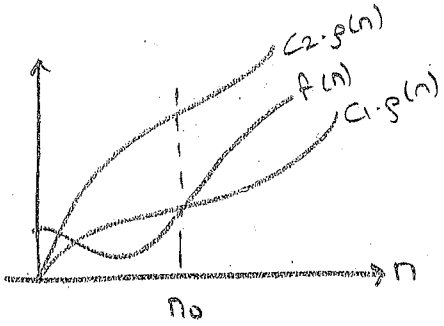
Ör

```
a = b+c   ——— 1
b = 2+a*2 ——— 1
c = d     ——— 1
d = a+b+c ——— 1
x = a/2   ——— 1
y = d+x   ——— 1
```

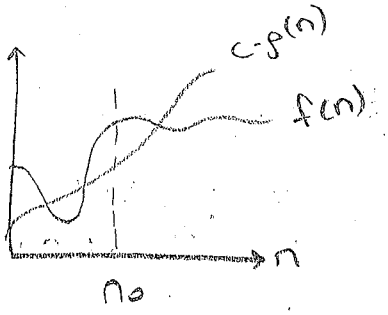
T(n) ve O bulunur.

T(n) = 6 dır.

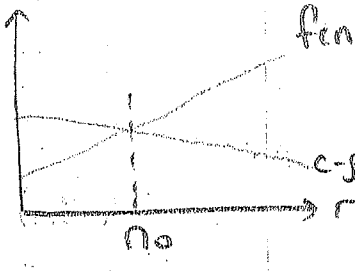
Bir p - O(1) olur. Çünkü sabittir



Θ - Notasyonu



O - Notasyonu



Ω - Notas

→ n_0'dan önce değerlere bakabiliriz ama n_0'dan sonra sabitler

Ör $\underbrace{\frac{1}{2}n^2 - 3n}_{f(n)} = \underbrace{\Theta(n^2)}_{g(n)}$ bu eşitlik doğru mudur? değil midir?

Çözüm

I. Yol $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$c_1 \cdot n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 \cdot n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

$n_0 = 7 \quad c_1 = \frac{1}{14} \quad c_2 = \frac{1}{2}$

II. Yol $\lim_{n\to\infty} \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{1}{2}$

sağlar,

$6n^3 = \Theta(n^2)$ bu eşitlik doğru mu?

I. Yol

$c_1 \cdot n^2 \leq 6n^3 \leq c_2 \cdot n^2$

$c_1 \leq 6n \leq c_2$

II. Yol

$\lim_{n \to \infty} \frac{6n^3}{n^2} = 6n = \infty$

* Sağlanaz, doğru değildir. Sabit çıkması gerektiği, sonsuz çıktı.
* c_1 ve c_2 nin de sabit olması gerektiği, değişken çıktı.

Küçük - O Notasyonu

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

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Küçük ω Notasyonu

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

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

(Sınav Sorusu)

$4n = O(5n)$ doğru mudur?

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

$4 \leq c \cdot 5$

$c = 1$

$n = 1$

Doğru

(Sınav Sorusu)

$4n + 3 = \Theta(n)$ doğru mudur?

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

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

$c_1 = 1 \quad c_2 = 5$

$n_0 \geq 3$

Doğru

(Sınav Sorusu)

$n \cdot 2^n$ → Üssel (2)

n^3 → Polinom (3)

$n!$ → Faktöriyel (1)

$\log n$ → Loparitmik (4)

Bu karmaşıklıkları büyükten küçüğe doğru sırala

$n! > n \cdot 2^n > n^3 > \log n$

Faktöriyel > Üssel > Polinom > Loparitmik

Ör $2^{n+1} = O(2^n)$ bu eşitlik doğru mudur?

$2^{n+1} \leq c \cdot 2^n$

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

$c \geq 2$

$n \geq 1$

Ör $2^{2n} = O(2^n)$ eşitlik doğru mudur?

$2^n \leq 2^n$

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

$2^n \leq c$

n'e bağlı olmuyor, sağlamaz.

Ör Aşağıdaki karmaşıklıkları küçükten büyüğe sırala.

a) $\frac{1}{3}$

b) $n^2 \cdot \log n$

c) $\left(\frac{3}{2}\right)^n$

d) $4 \log n$

e) $\frac{7}{3n}$

* n'in yerine 10 veya 100 yazarak deneyelim

10 yazarsak;

a) $\frac{1}{3}$ ②

b) $100 \cdot \log 10 = 300$ ④

c) $\left(\frac{3}{2}\right)^{10}$ ⑤

d) $4 \log 10 = 12$ ③

e) $\frac{7}{30} = 0.23$ ①

$e < a < d < b < c$

REKÜRSİF ALGORİTMALARI

$a_n = 4n + 5$

$f(x) = 4x + 5$

$a_{n+1} = a_n + 4$

$a_1 = 9$

Bu ikisi birbirine eşittir.

$n = 0 \quad 5 - 5$

$n = 1 \quad 9 - 9$

$n = 2 \quad 13$

1) master metodu
2) Substitution yönt.
3) Recurrence Tree

1. Master Metodu (Teoremi)

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

$$T(n) = \begin{cases} \Theta(n^i \cdot \log_b^n) & a = b^i, \quad i = \log_b^a \quad (1) \\ \Theta(n \log_b^a) & a > b^i, \quad i < \log_b^a \quad (2) \\ \Theta(n^i) & a < b^i, \quad i > \log_b^a \quad (3) \end{cases}$$

5) $T(n) = T(\frac{n}{2}) + n$ karmaşıklık bulunuz.

Çözüm

$a \cdot T(\frac{n}{b}) + c \cdot n^i$

$a = 1$
$b = 2$
$c = 1$
$i = 1$

$1 < 2^1$

$\Theta(n)$

Durum 3

6) $T(n) = 8 \cdot T(\frac{n}{2}) + cn^2$ karmaşıklık bulunuz.

Çözüm

$a = 8$
$b = 2$
$c = 1$
$i = 2$

$8 > 2^2$

$n^{\log_2 8}$

$\Theta(n^3)$

Durum 2

7) $T(n) = T(3n/4) + 1$

Çözüm

$a = 1$
$b = \frac{4}{3}$
$i = 0$

$1 < (\frac{4}{3})^0$

$1 = 1$

$\Theta(n^0 \cdot \log_{\frac{4}{3}} n)$

$= \log_{\frac{4}{3}} n = \log n = \Theta(\log n)$

Master 7

8) $T(n) = T(n-1) + 1$ karmaşıklığını bulunuz.

$a = 1$
$b = ?$
$i = 0$

Master Teoremine uymaz, bu yüzden yerine koyma yöntemini deneriz.

n pardüpün yere n-1 yaz.

$T(n-1) = T(n-2) + 1$ (n-2 yaz n pardüpün yere)

$T(n) = T(n-2) + 1 + 1$

$T(n-2) = T(n-3) + 1$

$T(n) = T(n-3) + 1 + 1 + 1$

Buraya 0 yapan n değeri kadar 1 oluyor

$T(n) = T(n-n) + 1 + 1 + 1 + \dots + 1$

$T(0)$ — n

$T(n) = n$

$= \Theta(n)$

Ör $T(n) = T(n-1) + \frac{1}{n}$

$\Theta(\log n)$ //

Ör $T(n) = 2^n T(n-1)$ karmaşıklığını bulunuz.

n yerdeğiştiğin yere n−1 yaz.

$T(n-1) = 2^{n-1} T(n-2)$ //

$\Rightarrow T(n) = 2^n \cdot 2^{n-1} \cdot T(n-2)$ //

$T(n-2) = 2^{n-2} \cdot T(n-3)$

$T(n) = 2^n \cdot 2^{n-1} \cdot 2^{n-2} \cdot T(n-3)$

⋮

$T(n) = 2^n \cdot 2^{n-1} \cdot 2^{n-2} \cdots 2^{n-n} \cdot T(n-n)$

$T(n) = 2^{\frac{n(n+1)}{2}} = O(\sqrt{2})^{n^2+n}$ //

Ör

```
T(n)
program(n) {
  if n == 1  return 1  ⟶ Θ(1)
  else  x = program(n-1)  ⟶ T(n-1)
  return x+x ⟶ Θ(1)
  }
```

- Metodu aşınca T'li terim oluyordu.

$T(n) = T(n-1) + \Theta(1) +$

$= \Theta(n)$ olur //

Ör

```
T(n)
computer(x) {
if x == 0 return 0 ⟶ Θ(1)
else return max(computer(x-1), computer(x-2))
}                    T(n-1)      T(n-2)
```

$T(n) = T(n-1) + T(n-2) + \Theta(1)$

Peki karmaşıklığı ?

$\Theta(1,5)^n$

2^n de diyebilirsiniz.

// $T(n) = T(n-1) + n$

- Yerine koyma yöntemi uygulayın

$T(n-1) = T(n-2) + n-1$

$\rightarrow T(n) = T(n-2) + n + n-1$

$T(n-2) = T(n-3) + n-2$

$T(n) = T(n-3) + n + n-1 + n-$

⋮

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

$O(n^2)$ //

T(n)

```
Deneme(n) {
if(n==1) return 1          → 1
a=n+1;                     → 1
b=n-1;                     → 1
for(i=0; i<n; i++) }       → Θ(n)
c=a+b+i;
if(__) return Deneme(n/2)+Deneme(n/2) }  2·T(n/2)
else if(__) return Deneme(n-1) }         T(n-1)
else return Deneme(n/3)+Deneme(n/3)+Deneme(n/3) }  3·T(n/3)
```

Karmaşıklığı?

a) * $2 \cdot T(\frac{n}{2}) + n$ → a) Master Teoremi

$a=2 \quad b=2 \quad c=1 \quad i=1$

$a = b^i$

$2 = 2^1$

$\Theta(n^1 \cdot \log_b^n)$

$= n^1 \cdot \log_2^n = n\log n$ //

b) * $T(n-1) + n$

c) * $3 \cdot T(\frac{n}{3}) + n$

) Master Teoremi

$a=3 \quad b=3 \quad c=1 \quad i=1$

$a = b^i$

$3 = 3^1$

$\Theta(n^i \log_b^n)$

$\Theta(n^1 \cdot \log_3^n)$

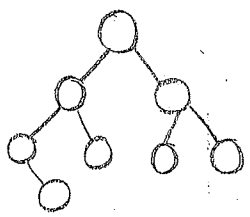
$= n\log n$ //

b)

Arama - Sıralama Algoritmaları

→ Karşılaştırma Tabanlı

	Worst Case	Avarage Case	Best Case
Lineer Arama	$O(n)$	$O(n)$	$O(1)$
İkili Arama	$O(\log n)$	$O(\log n)$	$O(1)$
Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n)$
Merge Sort	$O(n * \log n)$	$O(n \log n)$	$O(n \cdot \log n)$
Quick Sort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n)$
Shall Sort	$O(n^2)$	$O(n^2)$	$O(n)$
Radix Sort	$O(n+k)$	$O(n+k)$	$O(n)$
Cointing Sort	$O(n+k)$	$O(n+k)$	$O(n+k)$
Ducket Sort	$O(n^2)$	$O(n+k)$	$O(n+k)$



Heaptree ikili ağaçtır.
Ama eleman eklerken önce sola eklememiz gerekir
Bu yüzden heaptree değildir.

SınavSorusu

En az iki arama algoritması ve 10 tane sıralama algoritmasının karmaşıklıklarını yaz.

SınavSorusu

a) Binary Search
b) Insertion Sort
c) Merge Sort

Yukarıdaki algoritmalar için en kötü çalışma zamanı bağıntısını yaz.
Karmaşıklığını yaz.

ÇÖZÜM

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

b) $T(n) = T(n-1) + n$
$\Rightarrow O(n^2)$

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

Soru a) Insertion Sort
b) Merge Sort
c) Radix Sort
d) Heap Sort
e) Counting Sort

Durumlara göre hangisi en verimli algoritma olur?

=> Bir kütüphane kitaplığı kitap isimlerine göre yerleştirilmiştir. Sıralı bir şekilde dizmek için hangisini kullanmalıyız?
(Sıralı olan bir şeyi bir daha sıralayacağız!)
Insertion Sort kullanırız.

=> Bir bankanın atm cihazında bulunan bir hafıza alanında 2 milyon tane işlemi para çekme sırasına göre sıralayacağız.
HeapSort kullanılır.

=> Facebook'ta 64 bitlik e-kitaplar var. Hangi sıralamayı kullanırız?
Radix Sort kullanırız.

Soru Bir dizideki bütün elemanlar aynı ise ve siz bu diziyi QuickSort algoritması ile sıralamak istiyorsanız. Karmaşıklık ne olur?

① Lineer Arama (Doğrusal Arama)

17	23	6	55	31	95	7	21

Örneğin; verilen dizi içerisinde 7 aranıyorsa; liste sırasıyla kontrol edilir.
Listeyi gezerek 7 sayısı bulunur.

avantajı: oldukça kolaydır.

dezavantajı: çok yavaştır.

Kodu

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

Karmaşıklığı
En iyi → O(1)
Ortalama → O(n)
En kötü → O(n)

Lineer arama için en iyi durum; aranan elemanın listenin başında olmasıdır.

En kötü durum ise; listenin sonunda olmasıdır.

② Binary (İkili) Arama

- Yapı olarak böl-yönet yaklaşımının uygulamasıdır.
- Dizi sıralı olmalıdır.

Mantığı;
- aranacak dizinin tam ortasına bak
- aranan bulunduysa bit
- aranan değer; orta elemandan küçükse, küçük tarafı (sol) kontrolet
- aranan değer: orta elemandan büyükse, büyük (sağ) tarafı kontrol et

2	3	5	7	9	10	23	31	55	74	80

orta eleman

aranan 3 ise → sola
aranan 31 ise → sağa

KARMAŞIKLIKLAR
En iyi durum → O(1)
Ortalama → O(log n)
En kötü → O(log n)

Kodu

```
int[] dizi;
int boyut;
public boolean binarySearch(int anahtar)
{
int low = 0
int high = boyut - 1;
while (high >= low) {
int orta = (low + high) / 2;
if (dizi[orta] == anahtar) {
    return true;
}
else if {
  (dizi[orta] < key) {
      low = orta + 1;
}
else {
      high = orta - 1;
    }
```

① Eklemeli (Insertion Sort) Sıralama

- Sıralanacak eleman kümesinden 2. eleman referans alınır.
- Kendinden önceki elemanlarla karşılaştırılıp büyük olanı sağa kaydırma işlemi

Dizimiz ⇒ 3 (7) 2 5 1 4

2 3 7 5 1 4
2 3 5 7 1 4
1 2 3 5 7 4
1 2 3 4 5 7

} Araya ekleme

Kodu

```
public static void insertionSort (int[] a, 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;
      }
    }
  }
```

Karmaşıklıkları

En iyi durum → $O(n)$
Ortalama " → $O(n^2)$
En kötü " → $O(n^2)$

İnsertion Sort için

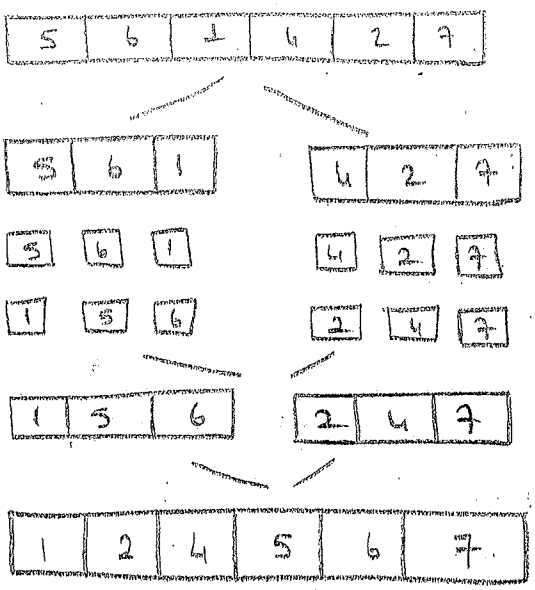
En kötü durum
Dizinin tersten sıralı olması durumudur.

En iyi durum
Dıştaki döngünün, içteki döngüm 1 kez çalışmasıdır.
(Yani; sıralıdır. Sadece kontrol) için döner

dıştaki döngü → kontrol etmek için
içteki döngü → yerleştirilip, değiştirmek için.

(2) Merge Sort

Mantığı ; sıralı olmayan dizeyi ortadan eşit olarak 2 alt listeye böler.
Alt listeleri kendi aralarında sıralayıp birleştirir.



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

Karmaşıklıkları

En iyi → O(nlogn)
Ortalama → O(nlogn)
En kötü → O(nlogn)

Kodu

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

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)
```

5) Quick Sort (Hızlı Sıralama)

Mantığı ; aynı veri türüne göre böl-yönet anlayışına göre sıralamakla yükümlüdür.

- Sayı dizisinden herhangi bir eleman pivot seçilir.
- Pivottan küçük olanlar ; pivotun önüne, büyük olanlar pivotun arkasına gelecek şekilde yerleşirler.

pivot

Dizimiz ⇒ 25 11 **43** 67 91 55 15

25 11 15 43 67 91 55

11 15 25 43 55 67 91

sıralandı //

En kötü durum

Seçilen pivotun çok büyük veya çok küçük olması //

En iyi durum

Seçilen pivotun sağında büyüklerin, solunda küçüklerin olması //

Karmaşıklıkları

- En iyi durum ⇒ (nlogn)
- Ortalama ⇒ (nlogn)
- En kötü ⇒ n^2

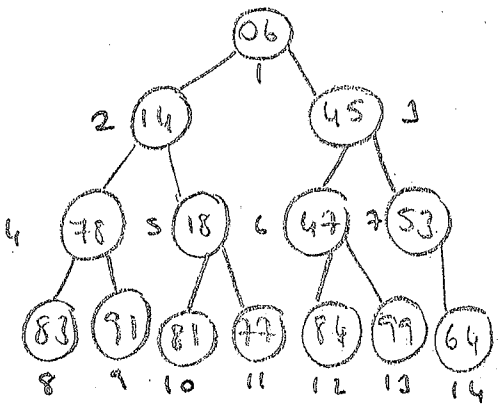
Kodu

```
void quickSort (int dizi[], int sol, int sag)
  int i = sol, j = sag;
  int temp;
  int pivot = dizi[(sol+sag)/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 Heap Sort (Yığınlama Sıralaması)

Min Heap → Çocukları kendinden büyük olacak.

$\log_2 N$



$N = 14$ $\quad \log_2 14 = 3$

Derinlik = 3

Parent(i) = ⌈i/2⌉

Left(i) = 2i

Right(i) = 2i+1

6, 14, 45, 78, 18, 47, 53, 83, 91, 81, 77, 84, 99, 64

1 2 3 4 5 6 7 8 9 10 11 12 13 14

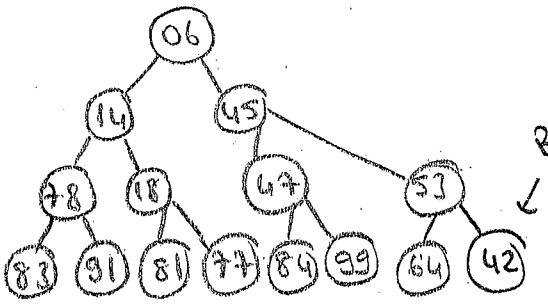
↳ 14'ün solunda kim var merak ediyorsak:

$2 \times 2 = 4$ ⟶ 78 varmış.

Heap'in avantajı → ekler aynı anda kadlayabilmesi.

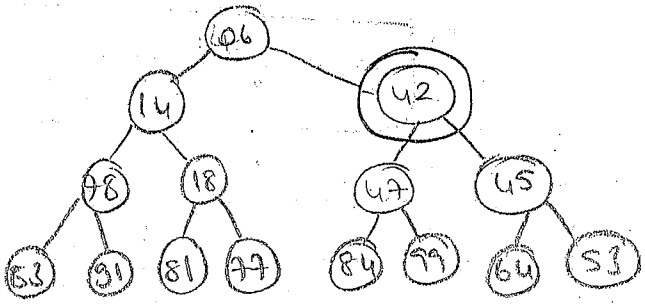
Eklemeye nasıl yapılır?

Meselo 42 ekleyelim.



Burası boş, ama 53'ten büyük olması gerekir ekler

- 53 ve 42 yer değiştirir.
Sonra 42 ve 45 yer değiştir.



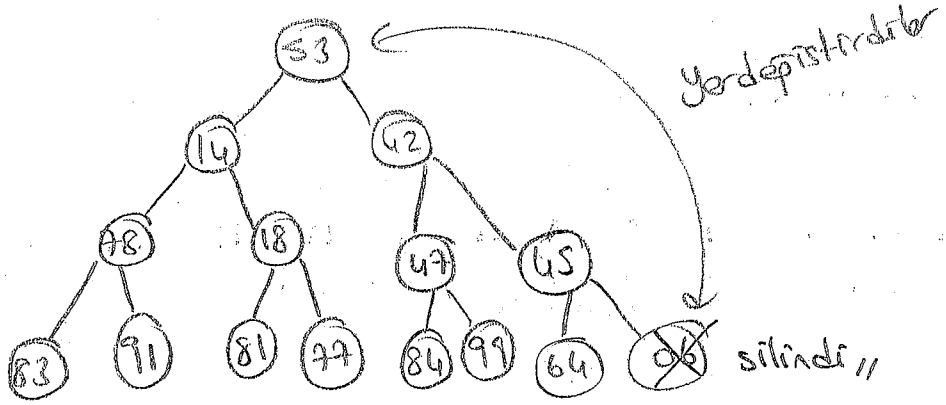
O(log N) adımda biter.
Maksimum derinliğimize kadardır / değiştirme sayısı

Silme nasıl yapılır?

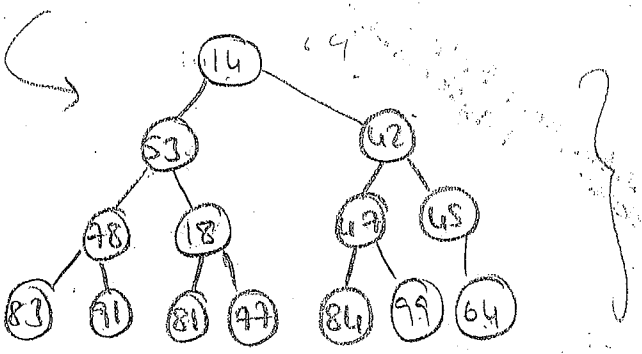
- Kökteki elemanı sil
- En sağdaki elemanı köke taşı
- Tekrar heapi düzelt (heapify)

6, 14, 18

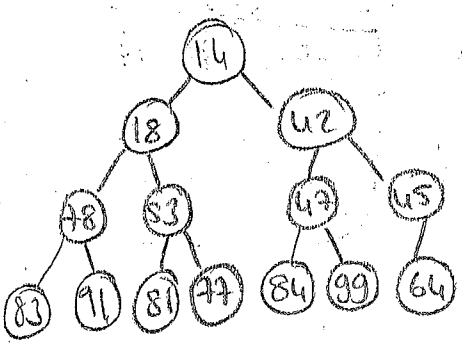
Kökü silip, kökün yerine pelesenkiyi bulacaz.
En sağdaki veya en soldakini getir.



53 pelince yapı bozuldu //
14'ü veya 42'yi almalıyız, küçük olanı almalıyız.



53, 18'den büyük sağlıyor.
Yer değiştirirler



Silme işlemi maliyeti → log N

//

Genel olarak

Kodu

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

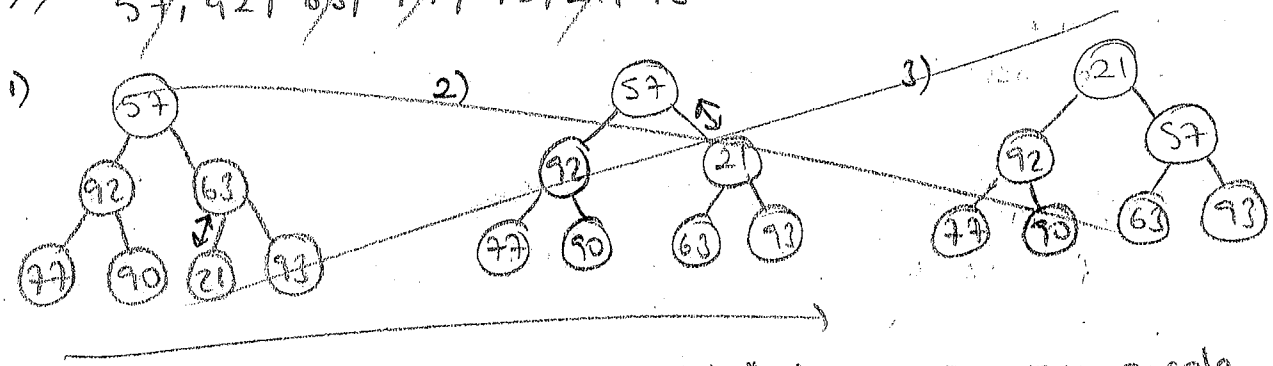
Gayet basarılı bir algoritmadır, çünkü problemi yarıya indirgeyerek düşürü problemin diğer yarısıyla ilgilenmez sadece ilgili taraf ile ilgilenir.

Karmaşıklıklar

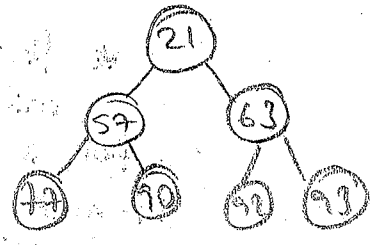
En iyi → O(nlogn)
Ortalama → O(nlogn)
En kötü → O(nlogn)

Ekleme veya silme de seviye olarak N düğüm varsa seviyelere göre N log N işlem yapar.

Ör// 57, 92, 63, 77, 90, 21, 93



En küçüğü köke al, en büyüğü en sağa veya en sola



5) Selection Sort (Seçme Sıralaması)

Mantık → En küçük değerin başa getirildiği sıralamadır.

* Başlangıçta dizinin ilk öğesi en küçük kabul edilir. (tabi pozisi)

Ör// 3, 9, 4, 7, 1

Çözüm

1, 9, 4, 7, 3

1, 3, 4, 7, 9 //

Ör// 1, 4, 7, 8, 3, 5, 2, 6

Çözüm

1, 2, 7, 8, 3, 5, 4, 6

1, 2, 3, 8, 7, 5, 4, 6

1, 2, 3, 4, 7, 5, 8, 6

1, 2, 3, 4, 5, 7, 8, 6

1, 2, 3, 4, 5, 6, 8, 7

1, 2, 3, 4, 5, 6, 7, 8 //

Karmaşıklıkları

En iyi durum → $O(n^2)$

Ortalama → $O(n^2)$

En kötü → $O(n^2)$

Kodu

```
public static void selectionSort (int[] dizi, int n){
 int temp, i
 int enkucuk;
 for (int i=0;  i<n-1; i++){
        enkucuk = i;
    for (int j = i ; j<n ; j++){
     if (dizi[j] < dizi[enkucuk]){
                enkucuk = j;
                }}
               temp = dizi[i];
               dizi[i] = dizi[enkucuk];
               dizi[enkucuk] = temp;
}}
```

* 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])

⑥ Bubble Sort (Kabarcık Sıralama)

- Sıralanacak eleman kümesinden ilk eleman alınır.
- Kendinden sonraki büyükse yer değiştirir.
- Sonraki elemana geçilir ve devam edilir.
- Dizi sonuna varıldığında en büyük elemanda sonda yer alır.

Ör.

3, 8, 6, 5, 4, 10, 1 bubble sort kullanarak sırala.

1. Adım

3, 8, 6, 5, 4, 10, 1
3, 6, 8, 5, 4, 10, 1
3, 6, 5, 8, 4, 10, 1
3, 6, 5, 4, 8, 10, 1
3, 6, 5, 4, 8, 1, 10

2. Adım

3, 6, 5, 4, 8, 1, 10
3, 5, 6, 4, 8, 1, 10
3, 5, 4, 6, 8, 1, 10
3, 5, 4, 6, 1, 8, 10

3. Adım

3, 5, 4, 6, 1, 8, 10
3, 4, 5, 6, 1, 8, 10
3, 4, 5, 1, 6, 8, 10

4. Adım

3, 4, 5, 1, 6, 8, 10
3, 4, 1, 5, 6, 8, 10

5. Adım

3, 4, 1, 5, 6, 8, 10
3, 1, 4, 5, 6, 8, 10

6. Adım

3, 1, 4, 5, 6, 8, 10
1, 3, 4, 5, 6, 8, 10

En iyi durum → Sıralı olması

En kötü durum → Tersine sıralı olması veya karışık dup, en küçük sayının sonda olması

Karmaşıklıklar

En iyi → $O(n)$

Ortalama → $O(n^2)$

En kötü → $O(n^2)$

Bubble Sort kodu

```
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; } }}
```

```
func bubbleSort (var a as array)
  for i from 1 to N
    for j from 0 to N-1
      if a[j] > a[j+1]
        swap (a[j], a[j+1])
      end fun
```

7) Shell Sort (Kabuk Sıralama)

(5, 7, 2, 9, 6, 1) } en basit yat yarından başlanacak

5, 7, 2

9, 6, 1

3

Her kolon kendi arasında sıralar

3, 6, 1
5, 7, 2
9

(3, 6, 1, 5, 7, 2, 9)

1, 2, 3, 5, 6, 7

} Sıralanaya yardımcı olur. İçinde bubble sort da kullanır.

Worsecase $O(n^2)$

Karmaşıklığı $\rightarrow O(n^2)$ dir.

⑧ Radix Sort (Basamağa göre)

Mantık → Sayıları basamaklarına göre sıralar.

↳ En anlamlı basamağa göre / en anlamsız basamağa göre

57, 43, 213, 24, 44, 102, 70, 37, 111, 23

birler basamağına 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

sıralandı //

```
Radix-Sort(A,d)
    for i=1 to d
    do use a stable sort to sort array A on dipit
```

Karmaşıklığı → O(n+k) } Her 3 durumdada //

⑨ Sayarak ((Counting) Sort) Sıralama

Mantık → En büyük sayıya göre, indiste sıralanır. Her sayıdan kaç tane olduğunu farklı bir dizide sayar.

5 7 2 9 6 1 3 7

→ En büyük sayı 9 olduğuna göre 9. indise kadar yapacak.

0	1	2	3	4	5	6	7	8	9
0	1	1	1	0	1	1	2	0	1

→ 1, 2, 3, 5, 6, 7, 7, 9

En iyi durum → Sıralı olarak verilmesi

En kötü durum → Mesela bütün elemanlar tek basamaklı biri 3 basamaklı 3. basamaklı indise kadar gelirse hafızayı boş yere kaplar.

Karmaşıklıkları

En iyi → O(n+k)

Ortalama → O(n+k)

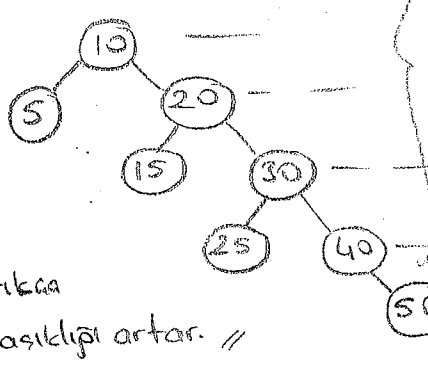
En kötü → O(n+k)

1, 2, 7, 4 36

AVL AĞAÇLARI

* BST (Binary Search Tree)
(İkili Arama Ağaçları)

10, 20, 15, 30, 25, 5



Binary Search'te etkene
Ama derpe sorunu var//

* Derinlik arttıkça
işlem karmaşıklığı artar. //

Level Algoritma karmaşıklığını etkileyen bir unsurdur.

Derpe sorunu çözmemiz gerektir!
işlem karmaşıklığı nlogn'e yaklaşsın.

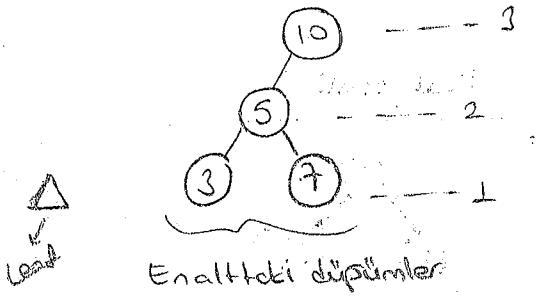
→ Sürekli böyle dizersek karmaşıklık artar.
Burda AVL ağaçları devreye girer.

AVL Ağaçları → Binary Search Tree üzerinde dengeleme yapar.

Amaç: İşlem karmaşıklığını azaltmak

(1) Düğümlerin yüksekliği //
(1) Derpe //

- Bilmemiz gereken en önemli unsur ağacın yüksekliğidir. (Düğümün yüksek

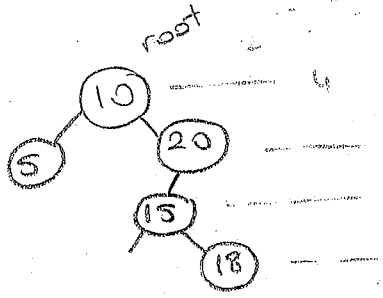


- Sol alt ağaç yüksekliği
- Sağ alt ağaç yüksekliği } Derp

→ Derpesizlik var //

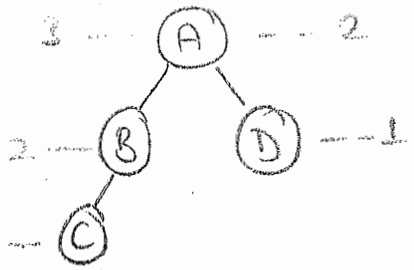
(Yaprak) Leaf olarak adlandırılır.

Yaprak düğümlerin sayısı 1 olarak hesaplanır.



sağ ve sol alt ağaçlarının yükseklikleri arasında 2 fark olmuyor.

SolAltAğaç(h) - SağAğaç(h) ≥ 1
Derpesizlik var
SolAltAğaç(h) - SağAğaç(h) = 1
Derpesizlik yok



$|3-2| = 1$ → Dengesizlik yok //

Döndürme İşlemi

Durumlar

1- Sol - Sol Durumu (Left - Left case) //
2- Sağ - Sağ Durumu (Right - Right case) //
3- Sol - Sağ Durumu (Left - Right case) //
4- Sağ - Sol Durumu (Right - Left case) //

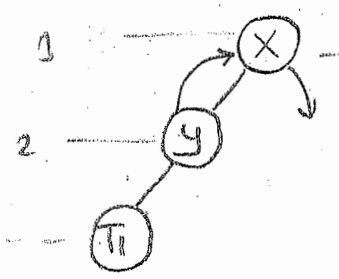
Durum → Dengesizlik Durumu //

İki çeşit çözüm uygulanır.

Rotations (Döndürme işlemi)

1- Sola Döndürme
2- Sağ Döndürme

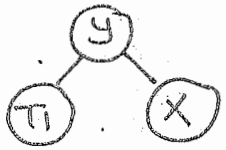
Sol - Sol Dengesizliği (Durumu)

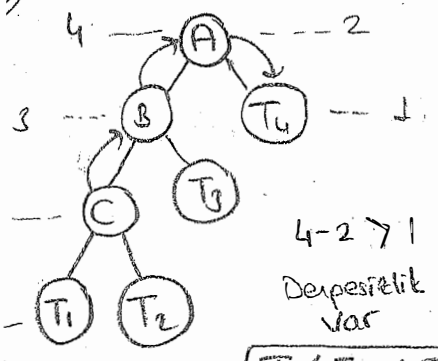


→ Sağa doğru döndürme yapılır.

$|3-1|$
$2 > 1$
Dengesizlik var.

Yeni Hali





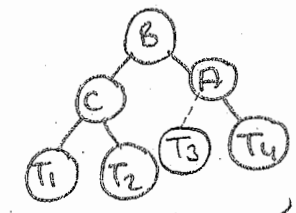
$4-2 > 1$
Dengesizlik var

$T_1 < T_2 < T_3 < T_4$

⇒ Sol - Sol durum
Sol Altağaca doğru bir dengesizlik oluşmuş } Sağa doğru döndürme

Çözüm

1)

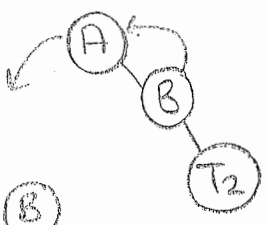


Dengeli hale geldi //

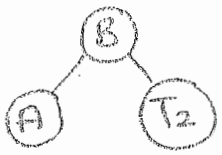
T_3 ne olacak ?

T_3 B'den büyük olduğu için sol alt ağaca yazılmaz ama A'nın sol alt tarafına yazılabildiğine göre A'dan küçüktür.

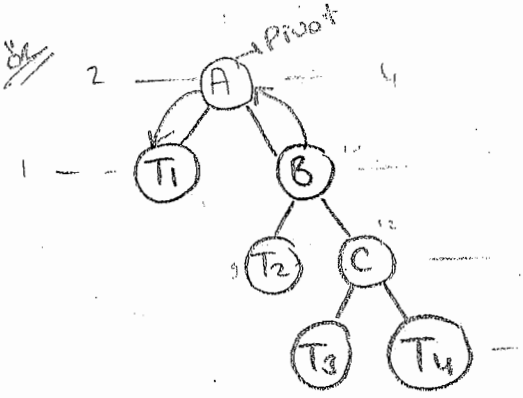
Sağ - Sağ Dengesizliği



Sağa doğru dengesizlik oluştuğu için, sola doğru döndeririz.



Ör

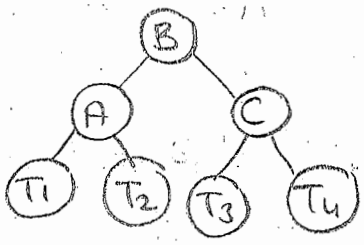


$T_3 < T_4$

$T_2 < T_4$

$4 - 2 > 1$ old. için dengesiz.

Çözüm



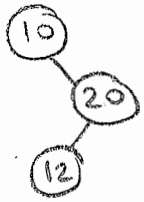
T_2 nereye gelecek?

T_2 normalde B'nin solunda, ama şu an orda A var.

T_2 A'dan büyüktür

↳ Çünkü örnekte T_2 A'nın sağ altopa

BST



12 ekliyim

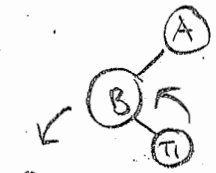
10'dan büyük 20'ye geldim

20'den küçük

- Yani her halükarda Binary Search Tree'de sağ alt ağaçtakiler, düğümden büyüktür.

Sol - Sağ Dengesizliği

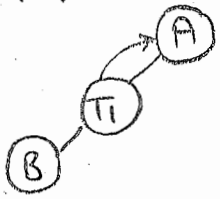
Dengesizlik sol alt ağaçtadır. Ama sağa doğru bir dallanma olusmuş (iki Asamalı bir döndürme uygulanır.)



Sol-sola çevirirsek

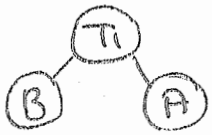
1. Aşama

B'yi pivot alıp sola döndürme yaparız.



sol-sol durumu oldu

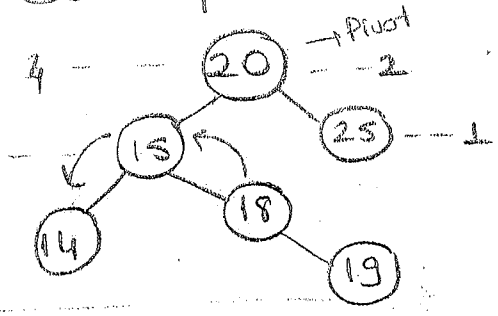
2. Aşama (Sağa döndürme olarak çözerim)



AVL ağaçları, ikili ağaç kuralına uymak zorundadır. Büyüklük, küçüklük ilişkisi olarak..

Ağaç seviye farkı kuralını sağlarsa ancak, ikili ağaç kuralına uymazsa; AVL Tree olmaz.

Sol - Sağ Durumu



4 - - - 2 - - - 1 → 4-2 > 1

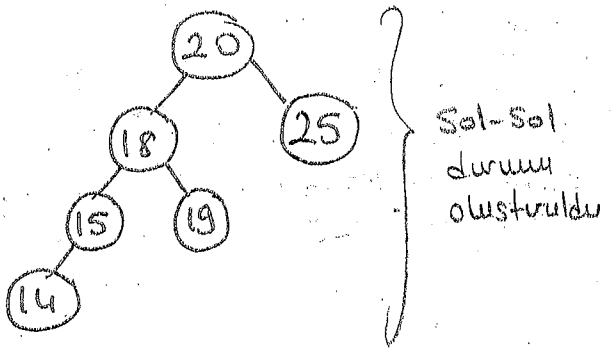
Çözüm

Pivotun solundaki düğüm sola döndürülür.

⇓

Dengesizliği oluşturan denklik Solda - sağa //

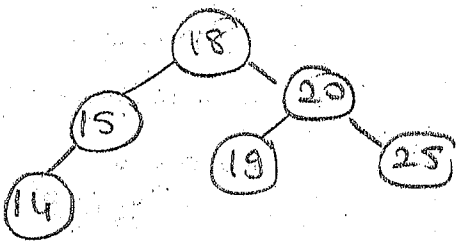
1. Aşama



Sol-Sol durumu oluşturuldu

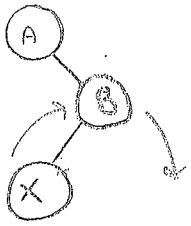
2. Aşama

20 sağa döndürülür.



19 normalde 18'in sağındadır.
18'in sağında 20 olduğu için
19'u 20'nin soluna yerleştirdik

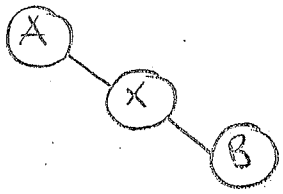
Sağ - Sol Durumu (Right - Left Case)



Dengesizlik sağa doğru oluşmuştur.
Ana sola doğru dallanma gerçekleşmiştir.
Yine 2 aşamalı çözüm uygulanır.

1. Aşama

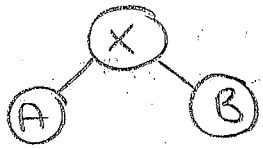
Pivotun sağındaki düğüm sağa döndürülür.



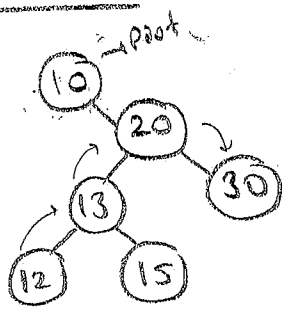
} Sağ - Sağ durum

2. Aşama

A'yı sola döndürürüz.

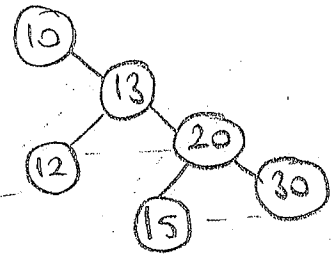


Sağ - Sol Durumu



1. aşama

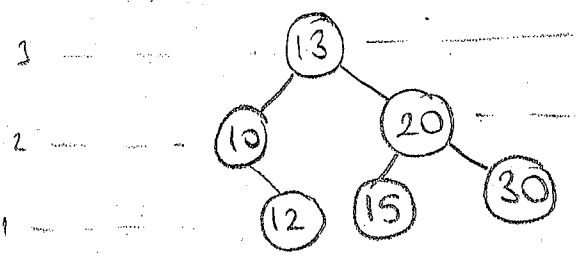
Pivotun sağındaki düğümü sağa döndür
(20'yi sağa döndür)



} Sağ - Sağ Durumu

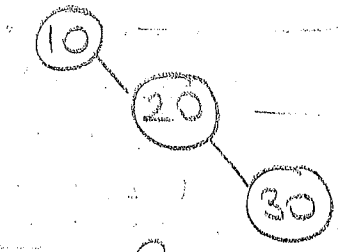
2. Aşama

10'u (Pivotu sola döndür)



} Ağaç dengeli hale geldi.

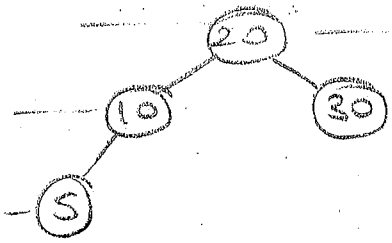
AVL Ağaçlarında Ekleme - Silme İşlemi



Sp - Sp

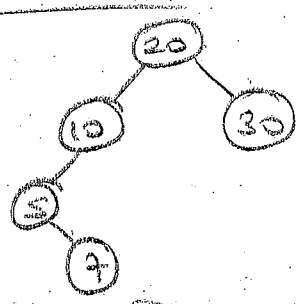
→ Sola döndür

5 ekle



Dengeli

7 ekle

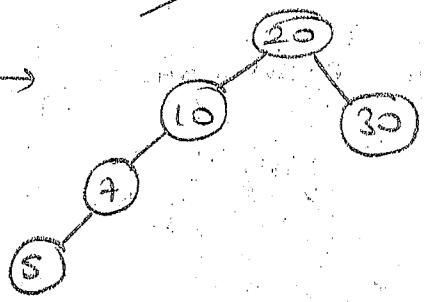


Dengesiz

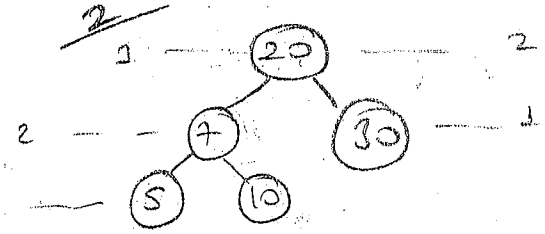
Sol - Sp

1) Sola döndür
2) Sağa döndür

1



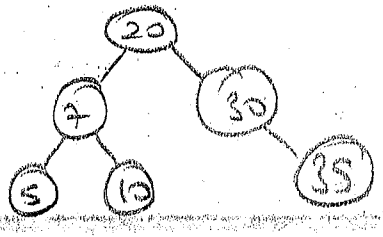
2



3 - 2 = 1

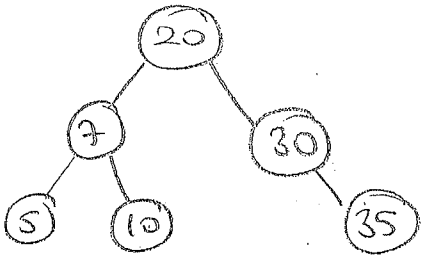
Dengeli hale geldi

35 ekle

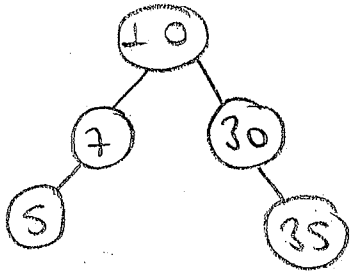


Hâlâ dengeli

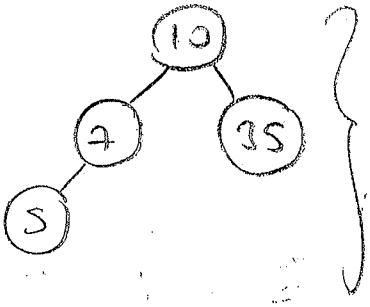
Silme İşlemi



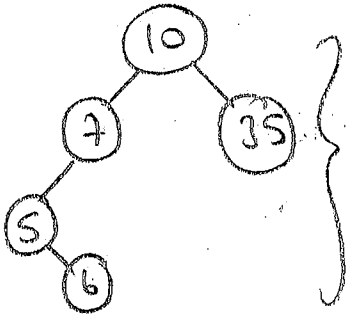
Düğüm silinince, yani 25'yi silince sol alt ağaçta en büyük düğüm 20'nin yeri kopyalanır.



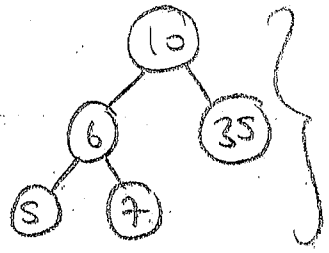
30'u sil →



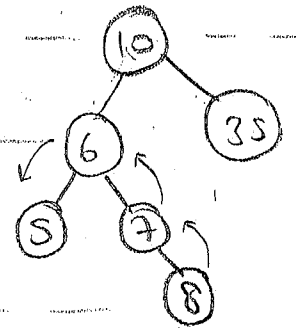
5'in sağına 6 ekle



Derpeyi sağla →

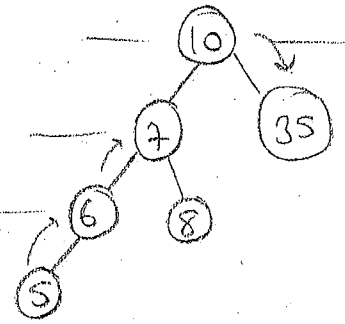


8 ekle



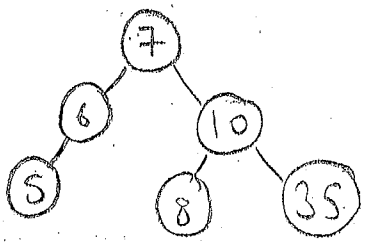
Sol - Sağ

Derpeyi sağla

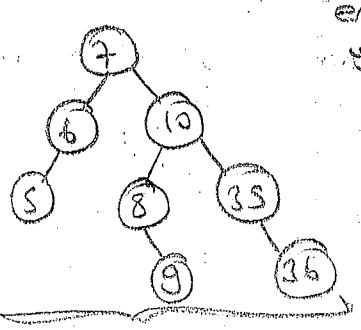


Sola döndür

Sağa döndür

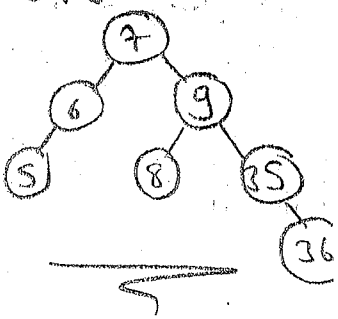


9 ve 36 ekle

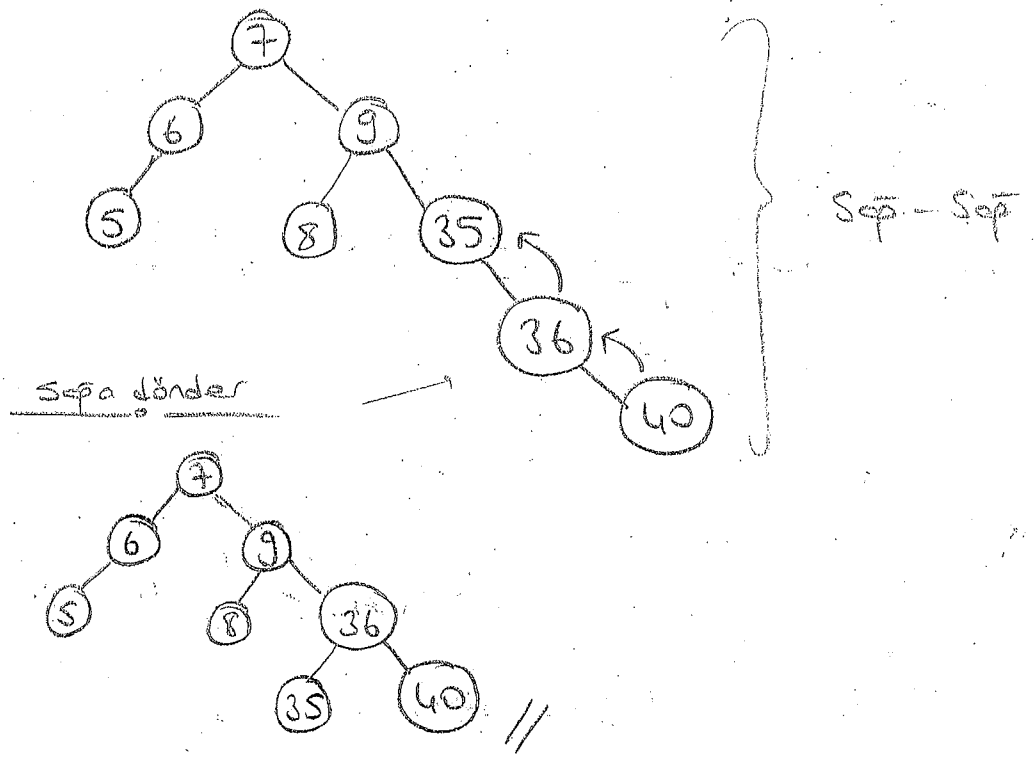


10'u sil

10'u sil dediğimde 10'un sol alt ağacındaki en büyük düğümü 10'un yerine kopyalar



→ D-even 40 ekle //



//

ED-BLACK TREE (KIRMIZI SİYAH AĞAÇLAR) (Binary Search Tree)

- Kırmızı Siyah ağaçlar öz-dengeli ikili arama ağacıdır.
- Ağacın kök düğümü her zaman siyahtır.
- Herhangi bir kırmızı düğümün ne çocukları ne de ebeveyni kırmızı olamaz.
- Kök düğümün bulunduğu konumdan yaprak düğüme kadar olan tüm farklı yollardaki siyah düğüm sayıları eşittir.

Bir düğümün en fazla iki çocuğu var.
İki tane kırmızı arka arkaya gelemez.

Neden Red-Black Tree ?

Bu ağacın dengesiz olduğu durumlar için O(n) e kadar çıkabilir.
Fakat Red-Black Tree tipindeki bir ağacın tüm bu işlemlerdeki maliyeti $O(\log(n))$ ile üstten sınırlıdır ve bu sınır garantidir.

AVL ve RED BLACK KARŞILAŞTIRMA

* AVL ağaçları Red-Black ağaçlarından daha dengelidir.
* Fakat AVL de ekleme ve silme de dengeyi korumak için birçok rotasyon yapılır.
* Eğer uygulamamızda kullanacağımız veri modelinde silme ve ekleme işleri yoğun yapılacaksa AVL yerine REDBLACK Tree seçimi daha mantıklı olacak.
* Fakat ekleme çıkarma az ve arama işlemi çok yapılıyorsa bu sefer AVL tercih edilmelidir.

! Kırmızı siyah ağaçların yüksekliği her zaman $h <= 2 \log_2(n+1)$ ile sınırlıdır.

! Red-Black Tree'de n elemanlı bir ağaç için kökten yaprağa en uzun yoldaki siyah düğüm sayısı $\log_2(n+1)$ //

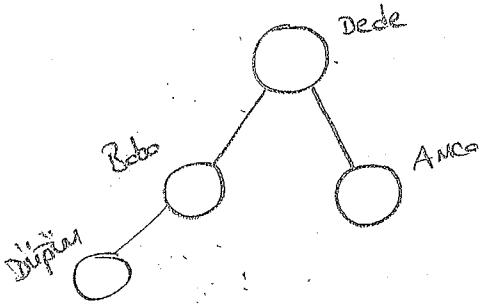
→ n elemanlı bir ağaç için toplam siyah düğüm sayısı en az: $n/2$ //

Red-Black Tree | Düğüm Ekleme

- Bu tip ağaçlarda dengeleme sağlanmak için

1) (Yeniden renklendirme)
2) (Rotasyon)

- şeklinde iki adet operasyon yapılır.
- İlk olarak renklendirme yapılmaya çalışılır.
- Koşullar sağlanmaz ise rotasyona başvurulur. //



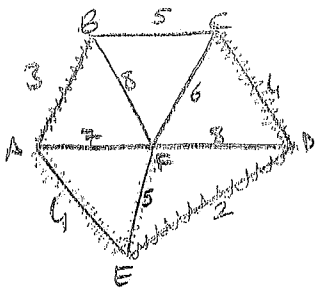
Amca kırmızıysa → yeniden renklendirme
siyahsa → rotasyon
↳ yeniden renklendir

* NULL düğümler siyahtır.

KRUSKAL ALGORİTMASI

- Minimum spanning tree,
- En az maliyetle tüm düğümleri dolaşmak.

exam



En küçük mesafeden, en büyük mesafeye kadar yazarız.

ED = 2
AB = 3
AE = 4, CD = 4
BC = ~~5~~ → alsaydık cycle olurdu.
EF = 5 → son olarak bunu alırız ve tüm düğümleri dolaşmış oluruz.
CF = 6
AF = 7
BF = 8
CF = 8

Toplamda 18 olur.

⚠ Cycle olusmamasına dikkat

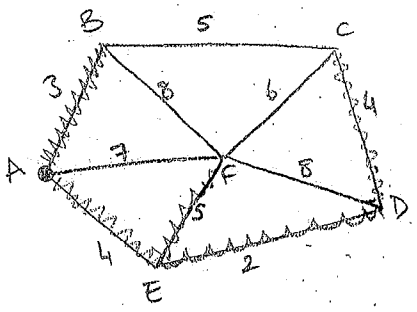
```
MST - KRUSKAL (G,w)
1.  A ← ∅
2.  for each vertex v ∈ V[G]
3.      do MAKE-SET (v)
4.  sort the edges of E into nondecreasing order by weight w
5.  for each edge (u,v) ∈ E, taken in nondecreasing order by weight
6.      do if FIND-SET(u) ≠ FIND-SET (v)
7.          then A ← A ∪ {(u,v)}
8.              UNION (u,v)
9.  return A
```

Karmaşık → $O(n^2 \log n)$

- Kenarları sıralayarak başlıyor
- Eşitsizlikteki sıra önemli değildir.
- Kenarları sırasıyla dolaşır

PRİM ALGORİTMASI

- Minimum spanning tree,



Soruda A noktasından başlanması istenirse

A'nın en yakın komşusunu seçeriz. (B)
A ve B'nin en yakın komşusunu seç (E)
A, B, E'nin en yakın komşusu (D)
A, B, E, D'nin en yakın komşusu (C)
Son olarak en küçük iki tane 5 kalıyor ancak BC yolunu seçmeyiz çünkü cycle oluşur.
EF'yi seçeriz.

Toplayınca → 3 + 4 + 2 + 4 + 5 = 18

Karmaşıklık → Prim → $O(n^2)$
Çünkü her düğüme birer defa bakarak sorunda.

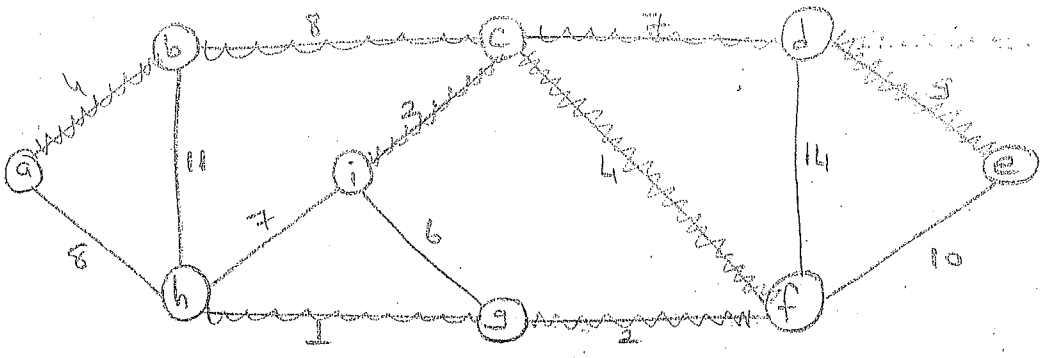
- Sonuç kapsama ağacı farklar
- En küçük değerli kenarda ilerler
- Yerel bir kapsama olarak alınır.
- Bütün düğümleri kapsayacak kadar ilerler.

⚠ Prim, herhangi bir noktadan başlar, kapalı döngü oluşturmadan tüm düğümlere uğrar.

Kruskal ise, en küçük uzunluktaki yerden başlar ve kendi içinden kontrol eder, kısadan başlar

} Fark

Kod



a noktasından başlarsak;
a'ya en yakın b //
a,b'ye en yakın c' yi seçtik //
a,b,c'ye en yakın i //
a,b,c,i'ye en yakın f //
a,b,c,i,f'ye en yakın g //
a,b,c,i,f,g'ye en yakın h //
d ve e kaldı (dolaşılmayan)
a,b,c,i,f,g,h'a en yakın d //
a,b,c,i,f,g,h,d'ye en yakın e //

Cycle oluşmamasına dikkat ettik !

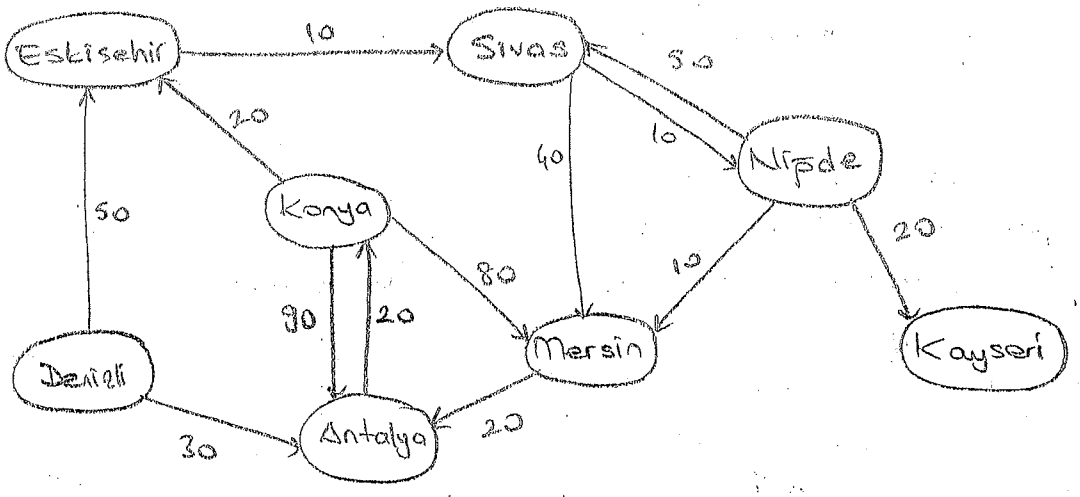
Toplam = 37 //

1. $Q \leftarrow \emptyset$
2. for each $u \in V$
3. do key[u] $\leftarrow \infty$
4. $\pi[u] \leftarrow$ NIL
5. INSERT (Q, u)
6. DECREASE-KEY (Q, r, 0) → key[r] ← 0
7. while $Q \neq \emptyset$
8. do u ← EXTRACT-MIN (Q)
9. for each $v \in$ Adj[u]
10. do if $v \in Q$ and $w(u,v) <$ key[v]
11. then $\pi[v] \leftarrow u$
12. DECREASE-KEY (Q, v, w, (u, v))

DİJKSTRA ALGORİTMASI

Shortest path, En kısa yol

Amaç: Bulunduğumuz noktadan en yakın yolları, en az maliyetli yolları bulmak



Başlangıç Noktası → Konya, Başlangıç noktası → 0 maliyet (sıfır)

Tablo oluşturarak yapabiliriz.

ziyaret edildi	Düğüm	Konya (0)	Eskişehir (∞)	Niğde (∞)	Mersin (∞)	Denizli (∞)	Sivas (∞)	Antalya (∞)	Kayseri (∞)
*	Konya	0	20 (Konya)	∞	80 (Konya)	∞	∞	90 (Konya)	∞
✓	(20) Eskişehir	0	20 (Konya)	∞	80 (Konya)	∞	30 (Eskişehir)	90 (Konya)	∞
✓	(30) Sivas	0	20	40 (Sivas)	70 (Sivas)	∞	30	90	∞
✓	(40) Niğde	0	20	40	50 (Niğde)	∞	30	90	60 (Niğde)
✓	(50) Mersin	0	20	40	50	∞	30	70 (Mersin)	60
✓	(60) Kayseri	0	20	40	50	∞	30	70	60
	Kayseri'den bir yere gidebildiğimiz için hepsi aynı kaldı.								
✓	(70) Antalya	0	20	40	50	∞	30	70 Güncelleme olmadı	60

Hepsi ziyaret edildi, Denizli ziyaret edilmedi.

Denizli'ye zaten ok bile yok :)

Konya – Eskişehir → 20
Konya – Niğde → 40
Konya – Mersin → 50
Konya – Denizli → ∞
Konya – Sivas → 30
Konya – Antalya → 70
Konya – Kayseri → 60

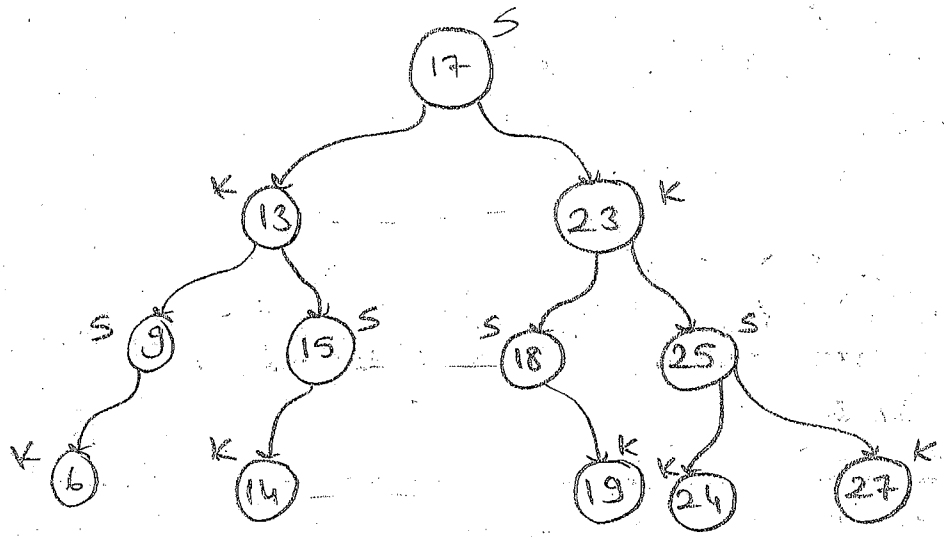
RED BLACK TREES (KIRMIZI-SİYAH AĞAÇLARI)

Veriyi ağaçta tutarken, ağacın dengeli olmasını sağlayan bir algoritmadır.

Worst case → O(log n)dir.

Kırmızı - siyah ağaçlar, ikili arama ağaçlarıdır. Herhangi bir düğümün solunda kendisinden küçük ve sağında ise büyük verilerin durması beklenir.

- Düğümler siyah - kırmızı renktedirler.
- Kök düğüm her zaman için siyahtır.
- Bütün yaprak düğümler siyahtır.
- Herhangi bir kırmızı düğümün bütün çocukları siyahtır.
- Herhangi bir düğümden, yaprak düğüme kadar gidilen bütün yollarda eşit sayıda siyah düğüm bulunur.



Her düğümün çocukları kendine ters renkte olur.

Arama İşlemi (Search)

Arama işlemi tıpkı binary search'teki gibi yapılır - Kökten başlanarak aranan sayı kökten küçük ise sola, büyükse sağa bakılarak, sonuca dönüş edilir.

Ekleme İşlemi (Insertion)

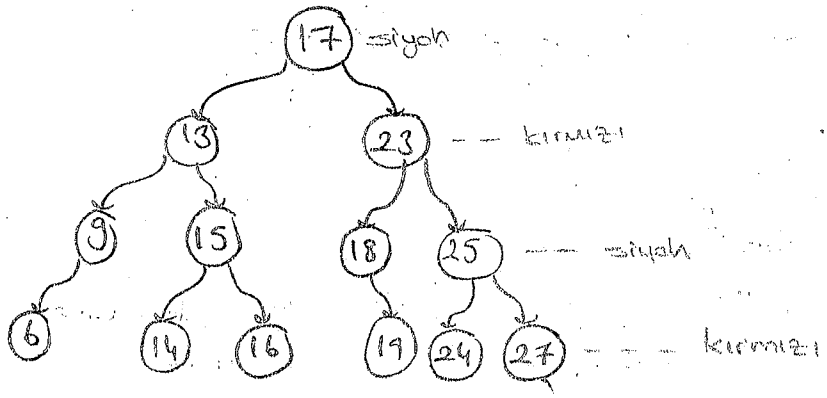
Ekleme işlemi, normal bir binary search tree'deki gibi yapılır. Bu işlem sırasında yeni düğümün kırmızı olacağını kabul ederiz.

Ekleme sırasında 3 temel kural:

1) Kök düğüm her zaman için siyahtır.
2) Herhangi bir düğümden, yapraklara kadar uzanan bir yolda eşit sayıda siyah düğüm bulunur.
3) Kırmızı düğümün, kırmızı çocuğu bulunmaz.

- Bu kurallara uymayan bir durum oluştuğunda ağaçtaki düğümle renki değiştirilir ya da değiştirileniyorsa ağaçta dengelemek için döndü (rotation) işlemi yapılır.

- Örneğin 16 sayısını ağaca eklemek isteyelim.
- 16 değeri, 17'den küçük olduğu için ağacın sol tarafında devam eder ve 13 ile karşılaşır. 13'ten büyük olduğu için sağa bakılır ve 15 ile karşılaşır. Ve son olarak 15'ten büyük olduğu için, 15'in sağ tarafına eklenir. 15 siyah olduğu için ve eklenen 16 kırmızı olduğu için sorun olmaz. (İki ardışık kırmızı olusmamıştır.



- Şimdi 30 ve 32 sayılarını ekleyelim.

Ağaca 30 değeri eklenince, ağacın en sağına yerleşmekte ve istenmeye bir durum olan iki kırmızı arka arkaya gelmektedir. Çözüm olarak ağaçtaki düğümlerin renki değiştirilmelidir.

30'un hemen üzerinde bulunan 27 düğümü, iki kırmızı arka arkaya gelemeyeceği için siyaha çevrilir.

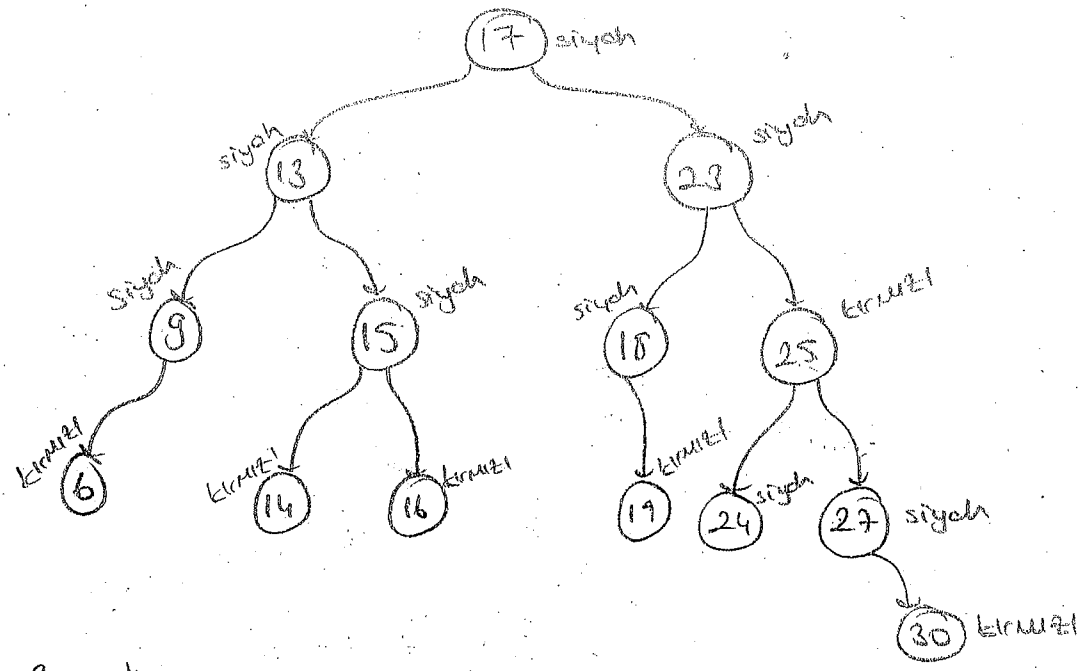
27 düğümünün büyükbabası 23 tür. Yani 27 düğümünün amcası 18'dir

27 siyah, 26 siyah olunca düğümden yapraga kadarki siyah düğüm sayısı eşit olsun diye 25'in kırmızıya çevrilmesi, 23 için sıkıntı oluyor. Çünkü iki kırmızı arka arkaya gelemez. Çünkü bir kırmızı düğümün çocukları siyah olmalıdır.

23'ü de siyah yapamayız çünkü 17'den yapraklara gidene sağ ve sol yollarda siyah düğüm sayısı eşit olmalıdır.

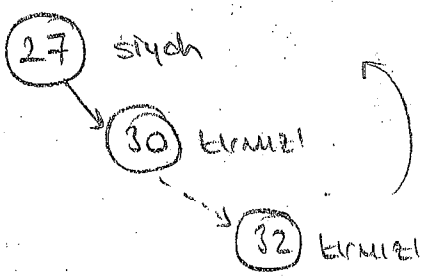
17'yi kesinlikle kırmızı yapamayız çünkü kök siyah olmalıdır!

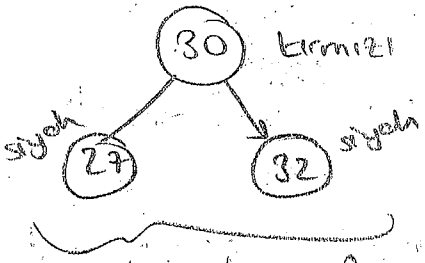
Dolayısıyla bu adımda 13 ve 23'ü siyah yapacak sorunu halledebiliriz.



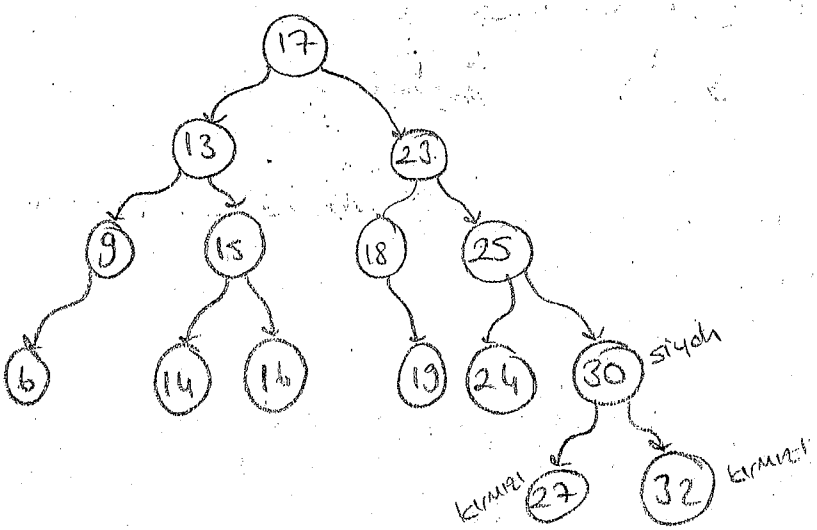
- 32 değerini eklemek istersek, 30'un sağına gelir. Yine sorun: arka arkaya iki kırmızının gelmesidir. 30'un veya 32'nin siyah yapılması problemi çözmez çünkü yollardaki siyah düğüm sayısı eşit olmaz.

Çözüm olarak sola döndürme yaparak çocukları siyah yaparız.





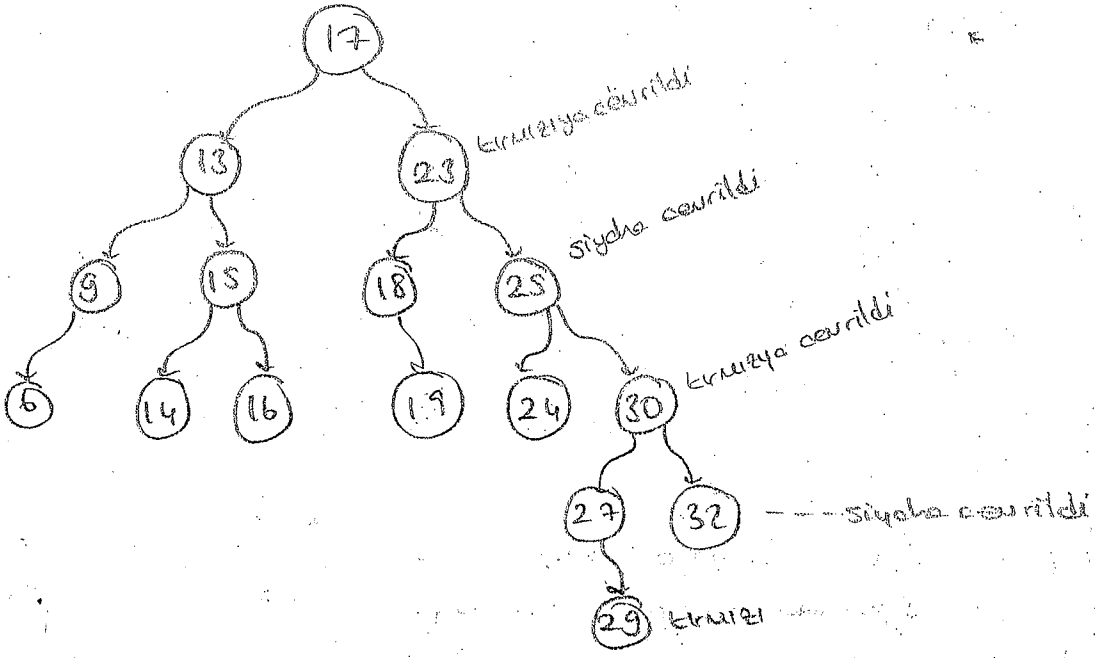
Ama bu sefer yine 25 ve 30 arka arkaya gelecek (kırmızı-kırmızı) Olmaz!



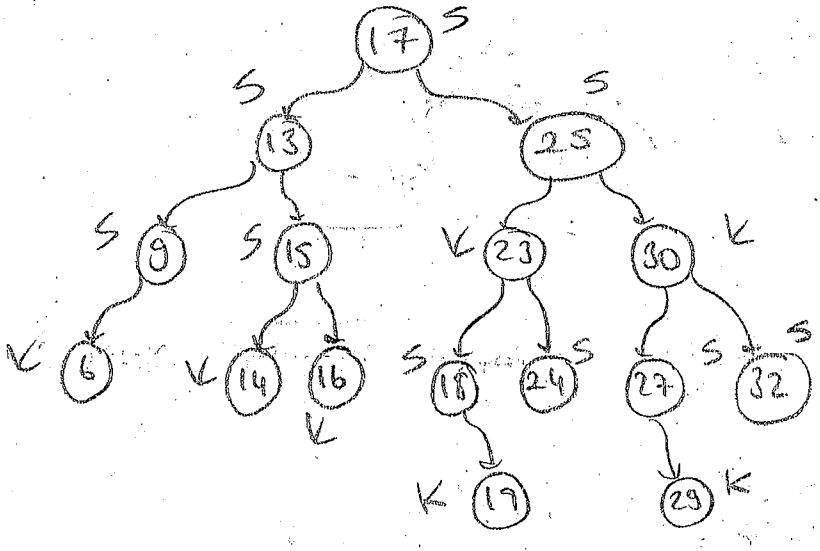
30 ve 27 renk takasında bulunursa hiç bir problem olmaz :)

- 29 sayısını ekleyelim.

29, 27'nin sağına eklene açısından yine aynı sorun oluşuyor.



Şimdi problem 23'ün sağında ve sol yolundaki siyah düğüm sayısı eşit olmaması, 23'ün solundaki problem düzeltilirse 17 için bir problem yok. Döndürme yapalım.



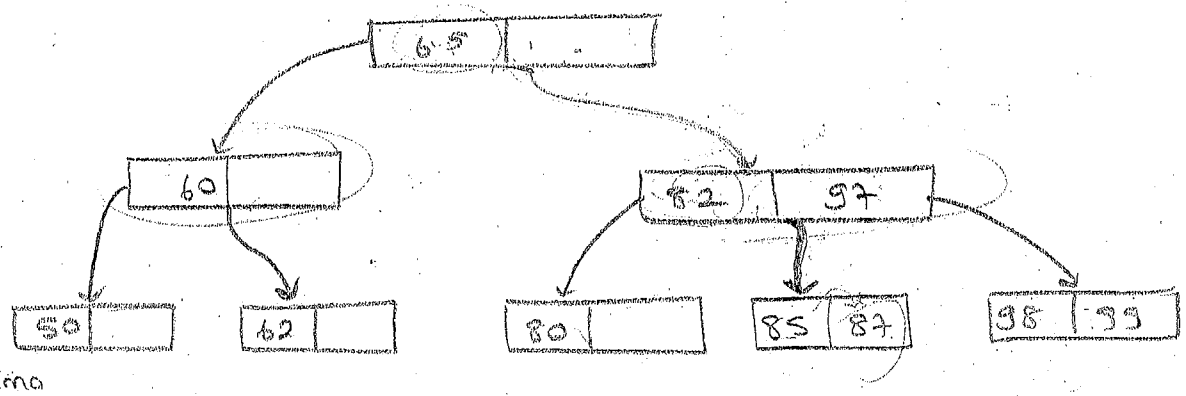
⇒ Her düğüm için yapraklara giden sağ ve sol yolunda siyah düğüm sayısı eşit olmalı !!!

~~Huffman Ağacı Oluşturma Algoritması~~

B Ağacı (B-Tree)

Amaç: Arama zamanını kısaltmaktır.

- Her düğümün en fazla m çocuğu bulunmalıdır.
  (Bu sayının üzerinde eleman bulunursa düğümün çoğaltılması gerekir)
- Kök ve yaprak düğümleri haricindeki her düğümün en az $m/2$ adet elemanı bulunmalıdır.
  (Bu sayının altında eleman bulunursa düğüm kaldırılır)
- Bütün yapraklar aynı seviyede olmak zorundadır.
  (Böyle değilse yapısal değişiklik gerekir)
- Herhangi bir düğümde (k) çocuk bulunuyorsa k-1 elemanı gösteren anahtar bulunmalı



Arama

87 aranıyor olsun:

1) Kök düğüme bak, 87 65'ten büyük, Kök düğümde tek anahtar olduğu için 65'in sağındaki göstericiyi takip et.
2) 65'in sağındaki düğüme gittin, ilk anahtar 82, 87 82'den büyüktür. O zaman ikinci anahtarla karşılaştır (97 ile)
   87, 97'den küçük olduğu için bu düğümde 82 ile 97 arasında bulunan göstericiyi izlenir.
3) Bir sonraki düğümde ilk anahtarla karşılaştır. 87 85'ten büyük. İkinci anahtarla karşılaştırınca 87'yi bulmuş oluruz.

⚠ Her düğümdeki anahtarlar sıralıdır. Bu yüzden bir düğümde istenen anahtar aranırken, düğümde bulunan sayıda teker teker bakılır.
(Linear arama, doğrusal arama)

B Ağaca Ekleme

Adımlar

1- Ağaca eklenecek doğru yaprak düğümü bulur.

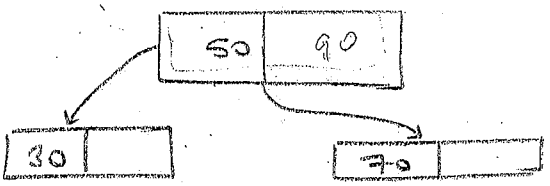
2- Bulunan yaprak düğümde azami anahtar sayısından daha az eleman varsa ekleme işlemi yapılır.

3- Yer yoksa bu durumda bulunan bu yaprak düğüm iki düğüme bölünür ve aşağıdaki adımlar izlenir

1) Yeni eleman eklendikten sonra düğümde bulunan anahtarlar sıralanır ve ortadaki elemandan bölünür. (medyan)

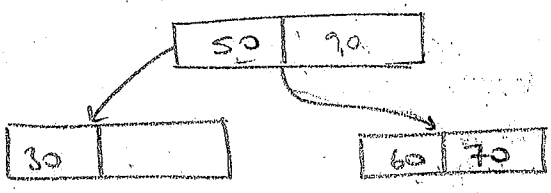
2) Ortanca değerden büyük elemanlar yeni oluşturulan sağ düğüme ve küçük elemanlar da sol düğüme konulur.

3) Ortanca eleman bir üst düğüme konulur.

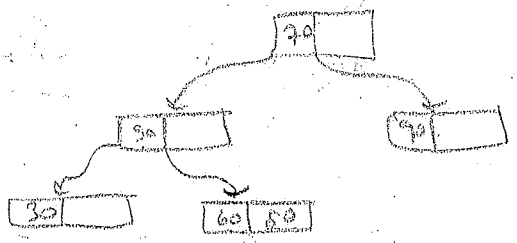


Örnepin azami anahtar sayısı 2 olan örnek;
60 ekleyelim.

⇒ Yer var



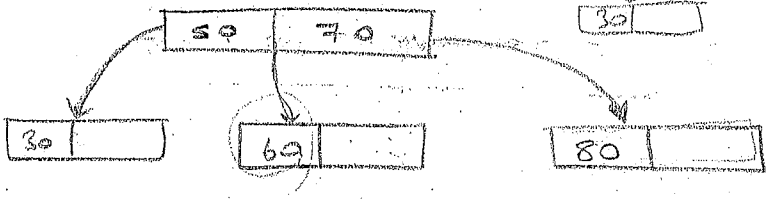
50, 90, 90



60 70 80

* 80 eklersek;

Eklenecek düğü

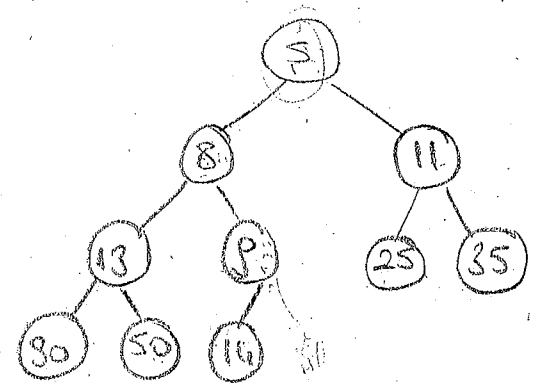


HEAP TREE (Yığın)

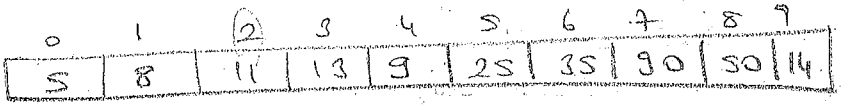
- Tekli bir ağaç türüdür.
- Çocuklar arasında bir ilişki yoktur.
- Değerli bir ağaçtır.

Min Heap

Bir düğümün çocukları daima düğümden büyük olur.



- Düğüm ekleme işlemi soldan sağa eklenir.



8'in kardeşi 1. index sol çocuğu 3. index
11'in kardeşi 2. " sol çocuğu 5. index
9'un kardeşi 4. indeks sol çocuğu 9. index

1 -- 3
2 -- 5
4 -- 9

(Ebeveyn index x 2) + 1 = sol çocuk indeksi

(Ebeveyn index x 2) + 2 = sağ çocuk indeksi

(Düğüm index - 1) / 2 = ebeveyn

Amaç

En yüksek performansta, bir diziden en küçük ve en büyük değerleri teker teker sırasıyla çekebilmektir. Bu dizi sıralı değildir. Ancak ilk eleman (kök) ya en büyük ya da en küçüktür.

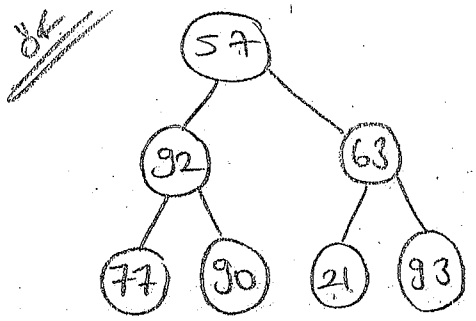
* Yeni eklenen düğüm, her zaman ağacın en alt sol tarafına eklenir. Yani dizinin en sonuna 3)

Ekleme

Değer dizinin sonuna, en sağına ağacın ise en alt soluna eklenir.
Eklenen değer ebeveyniyle kıyaslanır.
Ebeveyninden küçük ise (min heap) onunla yerdeğiştirir.
Bu işlemin aynısı ebeveynlerinin ebeveynleriyle de yapılır.

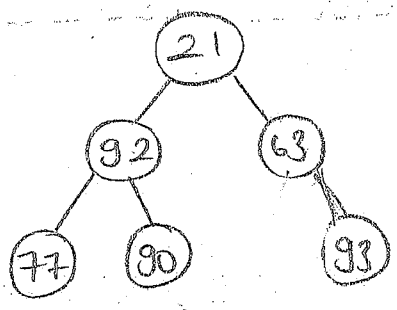
Çıkarma

Heap'de, daima kök değer çıkartılır.
Yani dizinin ilk elemanı çıkartılır.
Çıkartılan bu değer ile yeni ağaç oluşur.
Bu ağaçta kökten başlayan, çocuğu olmayan düğüme kadar, en küçük değere sahip çocuk ile kıyaslanır, eğer en küçük çocuktan büyük ise yer değiştirir.
Yer değiştiren çocukla aynı işlem ağacın o dalı boyunca tekrarlanır.

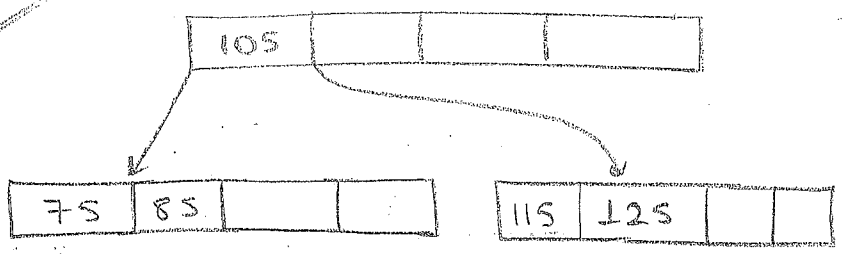


Silme yaparsak eğer 57'yi silersek

En küçük çocuk 21'dir.
21'de büyük olduğuna göre, 21 - 57'nin yerine geçebilir.

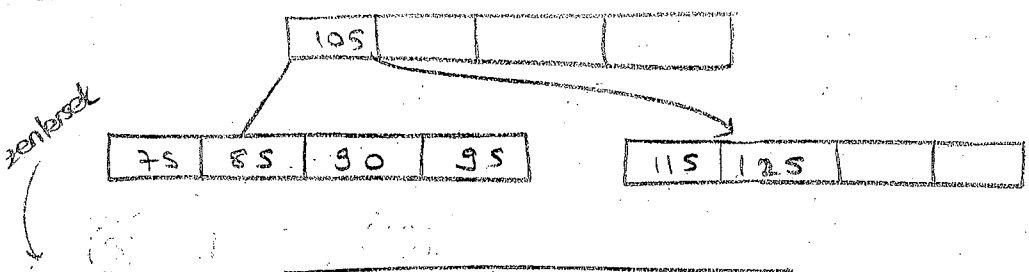


- Tree Örneği

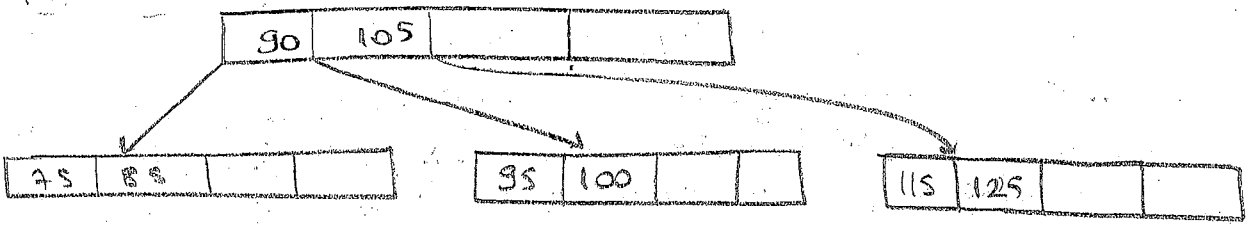


Elimizde bu durumda bir ağacımız var;

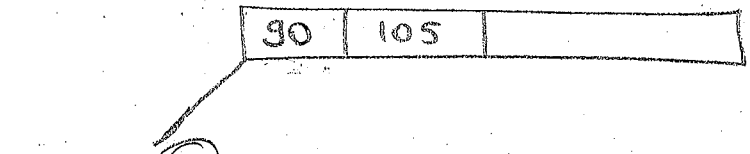
: 90 ve 95 eklemek istersek; 100 eklersek



zenbrook

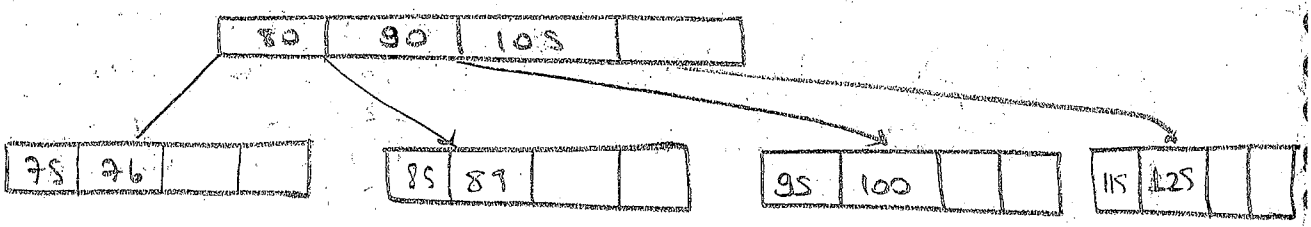


* 80, 89, 76 eklemek istersek;



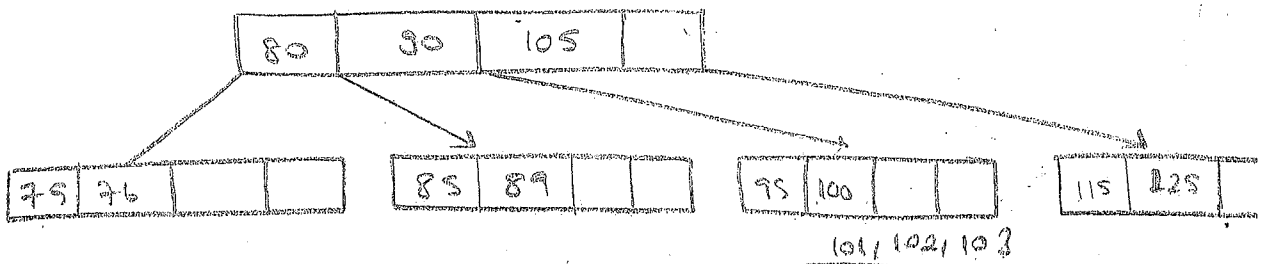
75, 76, 80, 85, 89

5 değer sığmayacağı için
Ortaca değeri (80) i yukarıya taşırız.



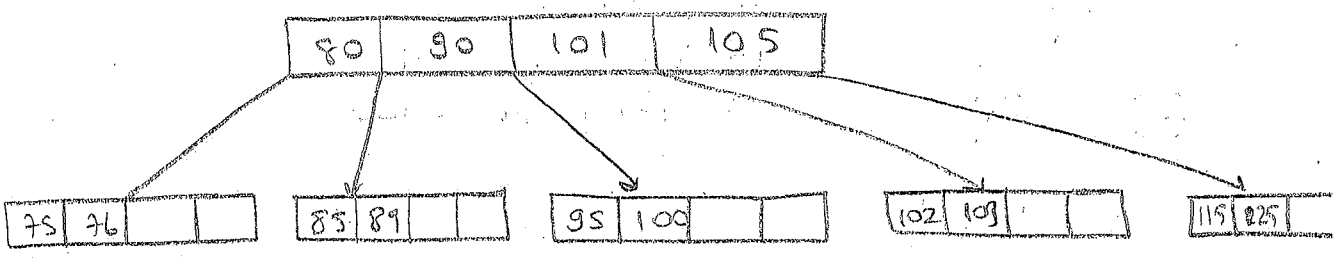
Unutma! Bir düğümde k elemanı varsa k+1 tane çocuğu olmalıdır.

* 101, 102, 103 ekleyelim.

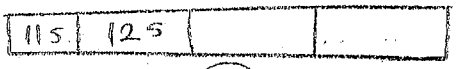


101, 102, 103

Sıralıdır için ortaca elemanı yukarıya gönderiyoruz.



* 225, 325, 175 ekleyelim.

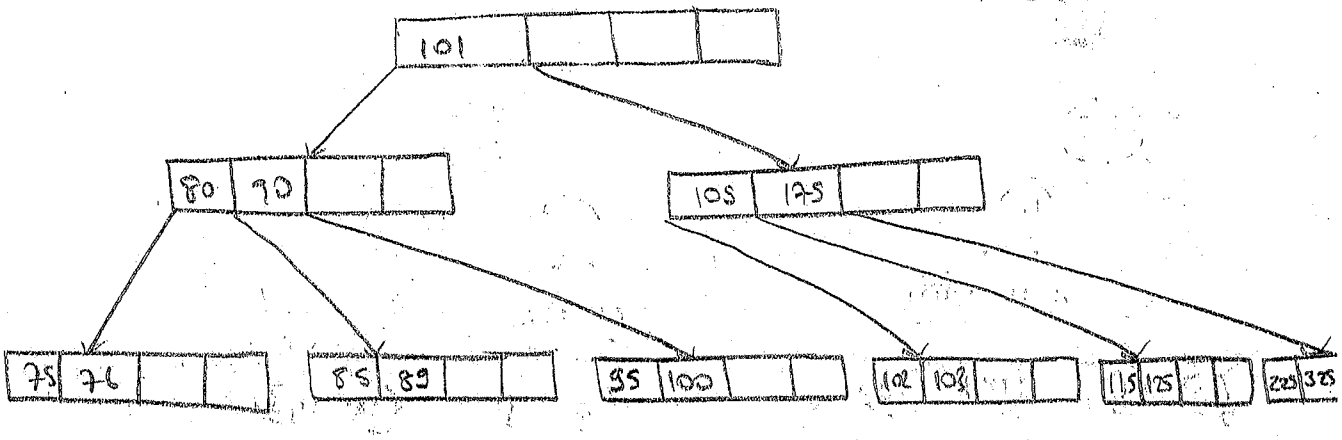


175, 225, 325

Median yukarıya gönderilir.

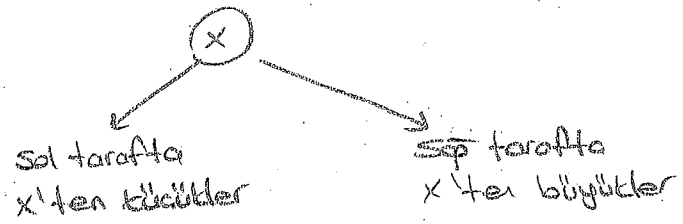
80, 90 | 101 | 105, 175

101 yukarıya

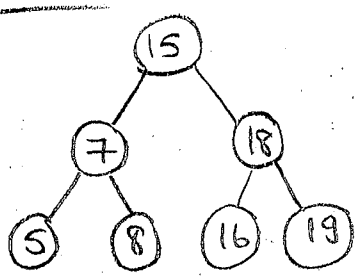


BINARY SEARCH TREE

İkili Arama ağacıdır.

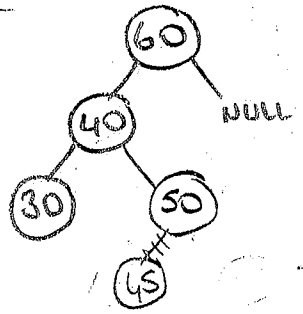


Arama:



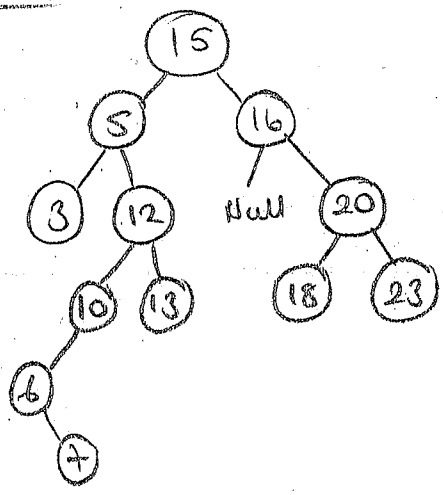
16'yı ara sak
15'ten büyük sağa,
18'den küçük sola,

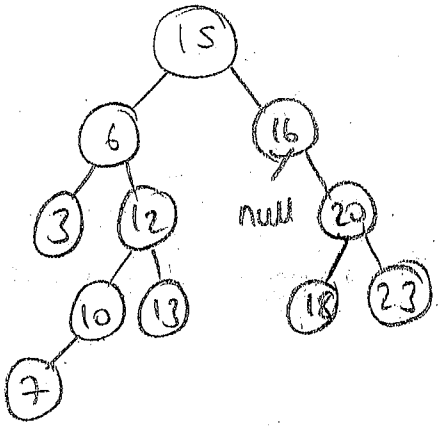
Ekleme:



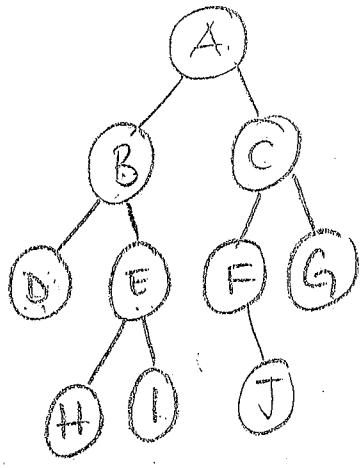
45 ekle

Silme:





5'i sil



Preorder (kök, sol, sağ)

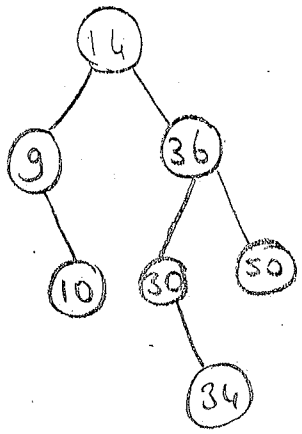
A - B - D - E - H - I - C - F - J - G

Inorder (sol, kök, sağ)

D - B - H - E - I - A - F - J - C - G

Postorder (sol, sağ, kök)

D - H - I - E - B - J - F - G - C - A



Preorder (Kök-Sol-Sağ)

14 - 9 - 10 - 36 - 30 - 34 - 50

Inorder (sol, kök, sağ)

9 - 10 - 14 - 30 - 34 - 36 - 50

Postorder (sol, sağ, kök)

10 - 9 - 34 - 30 - 50 - 36 - 14