VERİ YAPILARI (ÇALIŞMA)

HELİN YARDIMCI

```
public class ucgen {  // sınıf adında bir üçgen
    int kenar1;
    int kenar2;
    int kenar3;
```

} Parametreleri

→ public yazabilirsin de yazmayabilirsin. Erişim seçer.

```
ucgen (int x, int y, int z) {
    kenar1 = x;
    kenar2 = y;
    kenar3 = z;
}
```

} Sınıf adında kurucu metod

→ Kurucu metodtaki parametreleri, sınıf içindeki parametrelere eşitleme

```
public int cevre() {
    return x+y+z;
}}
```

} Çevre adında bir metod açıp çevreyi hesaplama

```
Psvm() {
    ucgen k = new ucgen (3,5,6);
    int d = k.cevre();
    System.out.println(d);
}
```

(k → Nesne; (3,5,6) → Parametre)

Main kısmında sınıf adında bir nesne oluşturulmuş.
Nesne üzerinden metod çağırılmış
- Kurucu metodta parametreler girildiği için nesne içerisinde parametre de yollarız.

```
class ogrenci {
int no;
String adsoyad;
char cinsiyet;
int yas;
double boy;
public ogrenci (int no, String adsoyad, char cinsiyet, int yas, double boy) {
```

→ Kurucu metod (sınıf adında metod)

```
this.no=no;
this.adsoyad=adsoyad;
this.cinsiyet=cinsiyet;
this.yas=yas;
this.boy=boy;
}}
```

this kullandığımızda kurucu metodtaki parametrelerin ismini değiştirmeyiz.

```
Psvm() {
    ogrenci[] a = new ogrenci[30];
    for (i=0; i<a.length; i++) {
    a[i] = new ogrenci (8, "Ali", "E", 22, 1.80);
}}
```

(a → değişken tipi gibi bir şey)

} Class'ın adında bir dizi açılıyor.

Temel Veri Yapıları ⇒ int, char, double, float, String, byte, boolean

Tanımlamalı Veri Yapıları ⇒ Temel veri yapıları kullanılarak yeni bir veri yapısı oluşturulur.

Veri Modelleri ⇒ Bağlı liste, Kuyruk, Yığıt, Ağaç, Graf, Veritabanı, AP

Veri Yapısı	Artıları	Eksikleri
Dizi	Eğer indis bilinyorsa erişim ve ekleme hızlıdır.	Sabit boyut Arama ve silme yavaş
Sıralı Dizi	Arama hızlıdır	Sabit boyut, silme yavaş
Yığıt	LIFO, Erişim hızlı İlk giren son çıkar	
Kuyruk	FIFO Son giren son çıkar.	Erişim yavaştır.
Bağlı Liste	Ekleme silme hızlı	Arama yavaş
Ağaç	Dengeli ise arama silme ekleme çok hızlı	Silme algoritması karmaşıktır.
Graf	Gerçek dünya sistemleri modelleyebilir.	Bazı algoritmalar yavaş çalışır, Karmaşıklık yüksek

Algoritmik Program Tasarımı

Verilen bir problemin bilgisayarda çözülecek şekilde adım adım yazılıp, bir programlama diliyle çözülmesini ifade eder. Yazılan komutlar uygulanabilir olmalıdır.

* Tasarım (Design) * Doğrulupunu ispat etme * Analiz * Uygulama * Test

Sözde Kod ⇒ Yarı programlama dili Gerçek kod ⇒ Tamamen programlama dili
Yarı Konuşma "

1) Atama operatörü :=
Eşitlik karşılaştırılması =

2) metot
Algoritma adı. ([Parametre Listesi])

3) if sart_ than
else - - - -

4) for - - - do

5) while do

6) Dizi indisleri

7) return

8) do-while
repeat

do {
} while (sart)

repeat
until sart

Sözde Kod //

ÖRNEK

```
toplacarp (dizi, toplam, carpim)
Girdi : n elemanlı dizi
Cikti : toplam ve carpim değişkenleri
for := 1 to n arasında do → bitiş
toplam := toplam + dizi[i] ;
carpim := carpim * dizi[i];
end for
```

Sözde kod

```
int[] sonuc = {0,1};
for (int i=0; i<dizi.length ; i++) {
sonuc[0] = sonuc[0] + dizi[i];
sonuc [i-1] = sonucci] * dizi[i]; }
return sonuc; }
```

Gerçek Kod

Algoritma Analizi

Herhangi bir algoritmanın calisma hızının matematiksel ifadesi //

1) Algoritma performansını ölçmek için
2) Farklı algoritmo karsilastirmalari
3) Daha iyi bir algoritma mümkün mü ?

ÖRNEK

```
int kare_topla (int[] d) {
    int toplam = 0 ;          1
  for( i=0 ; i<d.length; i++) { N+1
    toplam = toplam + dizi[i] ; N
    }
    return toplam; } 1
```

kaç kez	maliyet
1	c_1
N+1	c_2
N	c_3
1	c_4

$T(n) = 1 \times c_1 + c_2 \times (N+1) + N \times c_3 + c_4$
$= 2N+3$ //

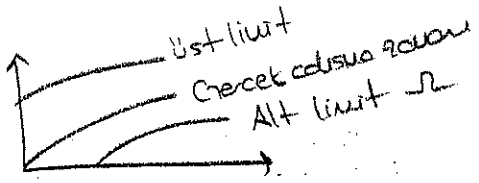
ÖRNEK

```
toplam=0 ; 1
for( i=0 ; i< N ; i++){ N+1
for (j=0 ; j<N ; j++) { N*(N+1)
    toplam++ ; N*N
  } }
```

$T(n) = 1 + N + 1 + N^2 + N + N^2$
$= 2N^2 + 2N + 2$ //

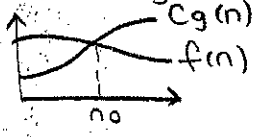
NOTASYONLAR

* En iyi çalışma zamanı → Ω
* Ortalama " " → Θ
* En kötü " " → O



Büyük O Notasyonu

$n \geq n_0$ $f(n) \leq g(n) \cdot c$ bu iki durumu sağlıyor ise $f(n) \in O(g(n))$ dir.



ÖRNEK

$f(n) = \frac{1}{2}n^2 + 3n$ $g(n) = n^2$ $f(n) \in O(n^2)$?

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

$\frac{1}{2}n^2 + 3n \leq cn^2$

$c = 1$ için

$\frac{1}{2}n^2 + 3n \leq n^2$

$\frac{1}{2} + \frac{3}{n} \leq 1$

$\frac{3}{n} \leq \frac{1}{2}$

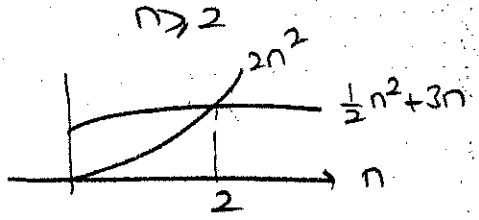
$n \geq 6$;

$c = 2$ için

$\frac{1}{2}n^2 + 3n \leq 2n^2$

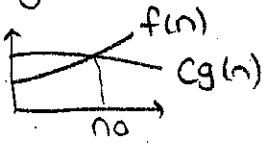
$\frac{3}{n} \leq \frac{3}{2}$

$n \geq 2$



Ω - Notasyonu En iyi durum

$n \geq n_0$ $fn \geq g(n) \cdot c$ bunu sağlıyorsa $f(n) \in \Omega(g(n))$ dir.



ÖRNEK

$f(n) = n^2 + 10n$ $f(n) \in \Omega(g(n))$?

$g(n) = n^2$

a) $n^2 + 10n \geq cn^2$

$c = 1$ için

$n^2 + 10n \geq n^2$

$10n \geq 0$

$n \geq 0$

b) $f(n) \in O(g(n))$

$n^2 + 10n \leq cn^2$

$n^2 + 10n \leq n^2$

$10n \leq 0$

$n \leq 0$

$c = 2$ için

$n^2 + 10n \leq 2n^2$

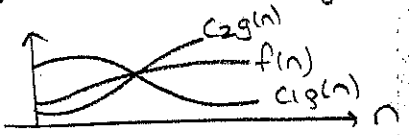
$10n \leq n^2$

$10 \leq n$

$n \geq 10$ // 5

(H) Notasyonu

$n \geq n_0 \quad g(n) \cdot c_1 \leq f(n) \leq g(n)\, c_2$ bunu sağlıyorsa $f(n) \in (H)(g(n))$ dir.



ÖRNEK

$f(n) = \frac{1}{2}n^2 - 3n \in (H)(n^2)$?

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

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

$n_0 = 7 \qquad c_1 \leq \frac{1}{14}$

$c_2 = \frac{1}{2}$

Cevap:

$\frac{1}{14}n^2 \leq \frac{1}{2}n^2 - 3n \leq \frac{1}{2}n^2$

* Çalışma zamanı negatif olmaz !

SORU

```
int ara (int[] d, int aranan) {
for(int i=0; i<d.length; i++) {          N+1
 if (aranan == d[i])                     N
    return i ; }
        return -1 ; }                    1
```

$\Omega = ?$ $\Omega = 1$ $\Omega(1)$

$O = ?$ $O(N) \Rightarrow 2N+2 \in O(N)$

$2N+2 \leq CN$

$C=3$ alırsak

$2N+2 \leq 3N$

$N \geq 2$

SORU

```
toplam = 0;                              1
for (int i=1; i<=N; i++)                 N+1
    toplam += i*i ;                      N
for (i=0; i<N; i++)                      N+1
for (j=0; j<N; j++)                      N*(N+1)
    toplam++;                            N*N
```

En iyi, en kötü ortalama durum?

$T(N) = N^2 + N^2 + N + N+1 + N + N+1+1$

$= 2N^2 + 4N + 3$ → Ortalama durum

En iyi durum → $2N+2$

En kötü durum → $2N^2+2N+1$

SORU

```
int basamak_sayisi (int N){
int i; sayi = 1
while (N>1) {
    sayi++;
    N = N/2;
}
return sayi;
```

$N, \frac{N}{2}, \frac{N}{4}, \frac{N}{6}, \ldots \frac{N}{2^k}$

$\frac{N}{2^k} = 1 \qquad \log_2 2^k = \log_2 N$

$k \cdot \log_2 2 = \log_2 N$

$k = \log_2 N$

Soru

```
toplam = 0;
for(int i=1; i<n; i=i*2)
    toplam++;
```

i = 1
i = 2
i = 4
i = 8
⋮
$i = 2^k$

$2^k = n$

$k = \log_2 n$

Soru (Recursive)

```
int fakt(int N) {
if (N==1)
    return 1;
else
    return N* fakt(N-1);
}
```

$T(1) = 1$

$T(N) = T(N-1) + 1$;

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

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

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

⋮

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

$T(N) = N$ //

NOT

* $O(1)$ → Sabit
* $O^{\wedge}(\log_2 n)$ → logaritmik
* $O^{\wedge}(n)$ → doğrusal
* $O^{\wedge}(n \log(n))$ → doğrusal çarpanlı logaritmik
* $O^{\wedge}(n^2)$ → karesel
* $O^{\wedge}(n^3)$ → kübük
* $O^{\wedge}(2^n)$ → üstel

ÖRNEK

$7n^2+5 \in O(n^2)$?

$7n^2+5 \leq Cn^2$

$C = 8$ //

$7 \cdot 0^2+5 \leq 8n^2$

$5 \leq n^2$

$n \geq 1$

$C = 12$

$7n^2+5 \leq 12n^2$

$5 \leq 5n^2$

$n \geq 1$ //

ÖRNEK

$2n+5 \leq Cn^2$?

$2n+5 \leq n^2$

$2+\frac{5}{n} \leq n$

$n = 4$ //

$C = 2$ için

$2n+5 \leq 2n^2$

$2+\frac{5}{n} \leq 2n$

$n_0 = 3$ //

$2n+5 \in O(n)$

$2n+5 \in On$

$C = 3$ için

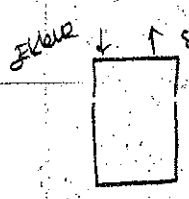
$2n+5 \leq 3n$

$n \geq 5$ //

STATİK YIĞIT (STACK)

* LIFO (İlk giren son çıkar) → son giren ilk çıkar :)

- Eleman ekleme ve çıkarma işlemleri her zaman üstten yapılır.



Stack gerçekleştirmek için
- Sabit dizi kullanılır
- Bağlı liste kullanılır

} iki şekilde kullanılır

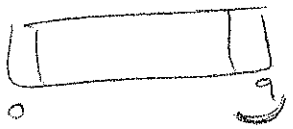
Stack Nerede Kullanılır?

- Tarayıcılarda Back (Geri) tuşu stacktir.
- Parantez kontrolünde kullanılır.
- Yazılımlarda undo (Geri al) tuşu
- Graf algoritmalarında
- Matematiksel ifadelerin hesaplanması ve dönüşümde kullanılır.

```
package yigit;
public class stack {
int[] dizi; // stack uzunluğu
int üst;
int N;
public stack(int N){ // Kurucu metod // Dizinin boyutunu belirtmek için → N
  dizi=new int[N];
this.N=N;
üst=-1; } // stack boştur

boolean bosmu(){
 if (üst==-1)
      return true;
 else
    return false; }

boolean dolumu(){
     return üst==N-1;
}
```

} Değişkenleri

veya → `return ==-1;`



N=10
N-1=9

Devam

```
int top(){
if (üst != -1)
   return dizi[üst];
  else
   return -1 ;}
```

üst = 3 olunca işleme devam

2	〃	〃	〃
1	〃	〃	〃
0	〃	〃	〃

Topla

* -1 olunca boş döndürür.

```
void ekle (int e) {
   if ( !dolumu() ){
    üst = üst +1 ;
     dizi[üst] =e ; }

    else {
  System.out.println("stack dolu");} }
```

e ⟶ eleman

```
int sil() {
   if(!bosmu){
    ust=ust -1 ;
    return dizi[ust +1];
                    }
      return -1
                  }}
```

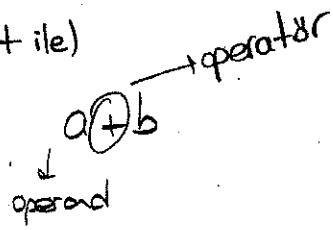
* üst = 5 ise azalttığımızda üst = 4 olacak, neyi azalttığımızı göstermek için dizinin içine ust+1 yazıyoruz.

Matematiksel İfadeleri Hesaplamak (Yığıt ile)

infix ⟶ ara gösterim ⟶ a+b

prefix → ön gösterim → +ab

postfix → arka gösterim → ab+



-Her zaman operatörler yığıta atılır. Operandlar ise bir diziye kaydedilir.

'(' stack boşsada direk ekleyin.

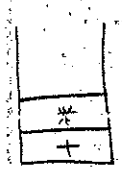
')' en yüksek öncelikli operatördür. '(' görene kadar stack boşaltır.

1) _ : 2) * | / | % 3) + | -

ÖRNEK

a + b * c postfix karşılığını bul.

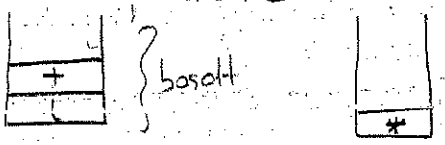
abc*+



ÖRNEK

(a+b) * c

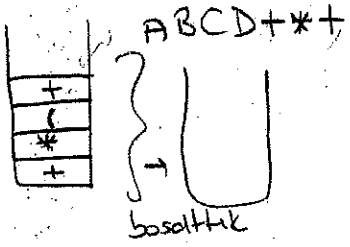
ab+C*



ÖRNEK

A+B * (C+D)

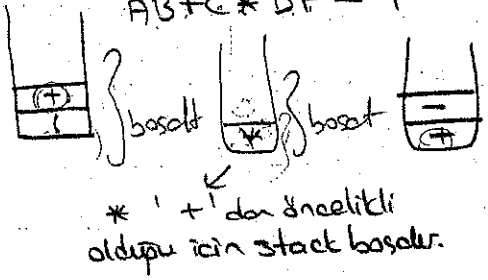
ABCD+*+



ÖRNEK

(A+B) * C+D − F

AB+C*DF−+



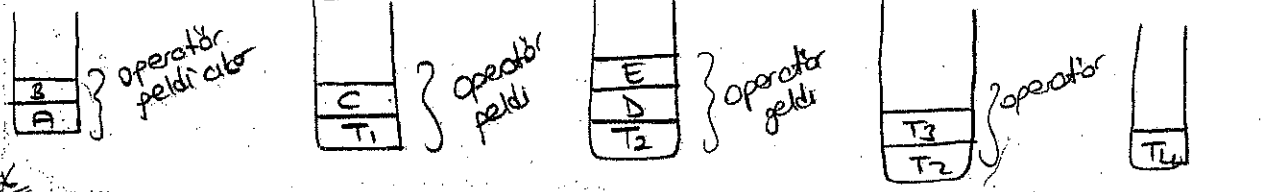
Yığıt ile Postfix Hesaplanması

Operand gelirse yığıta ekle, Operatör gelirse yığıttan iki operand çekip işlemi yap tekrar yığıta ekle.

ÖRNEK

AB+C*DE−+

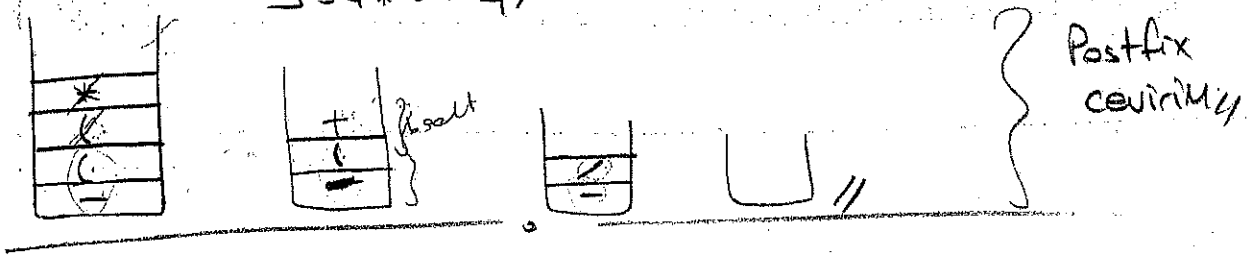
$T_1 = A+B$ $T_2 = T_1 * C$ $T_3 = D-E$ $T_4 = T_2 + T_3$



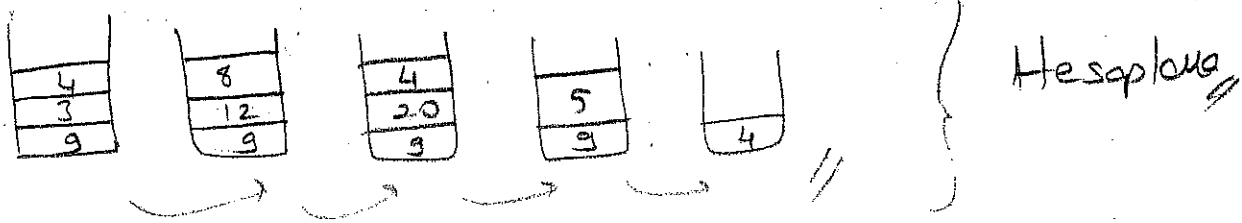
ÖRNEK

9−((3*4)+8)/4 bu ifadeyi önce postfix çevir. Sonrada hesaplama yap.

934*8+4/−



4*3=12



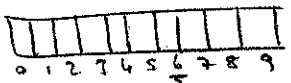
STATİC KUYRUK

* FIFO (Son giren son çıkar)

Ekleme sondan yapılır.
Çıkarma baştan yapılır.

Nerede Kullanılır?
- İş programlama (işletim sistemi)
- Yazıcı kuyruğu
- Graf algoritmalarında
- Kelebek çoğalması seviye seviye dolaşma

```
public class Kuyruk {
 int[] dizi;
 int N;
 int bas, son;                    } Değişkenlerini gizliş
 public Kuyruk (int N){
 dizi = new int[N]; →N elemanlı bir dizi oluşturuluyor
    this.N=N;
     bas=0;
     son=0; }                     } Kuyruk şu an boş

 boolean bosmu(){
  if(bas==son)
        return true;
    else return false; }

 boolean dolumu(){
 if(bas==(son+1)%N)
       return true;
   else return false; }

 void ekle(){
    if (!dolumu()) { int yeni;
      dizi[son]=yeni;
      son=(son+1)%N; }
 else System.out.println("Kuyruk dolu"); }

 int sil(){
     int sonuc;
   if(!bosmu()){
      sonuc=dizi[bas];
    bas=(bas+1)%N;
          return sonuc; }
          return -1; }
```

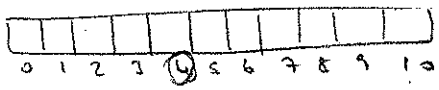


son=6 (+1+7) N=10 ① ② ①

son = (6+1) %. 10 = 7

son=7 oluyor.

baştan 7. sondan 3.



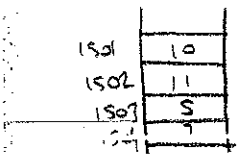
N=11 // bas=4 → silmek istiyorum

(4+1) % 11 = 5 //

Yani 4. indisin yerine 5. indis gelir //

- BAĞLI LİSTELER -

* Dizi — Dizide olduğu gibi elemanlar hafızada ardışıl olarak saklanır.

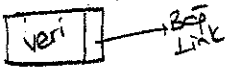


* Araya ekleme maliyetli (kaydırma)
* Araya silme maliyetli (kaydırma)
* Dizi boyutu sabit
* Dizi elemanları fiziksel olarak sıralı

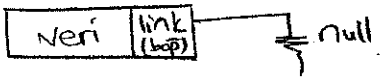
```
int[] diziden Sil ( int [] dizi , int k ) {        → Silinecek elemanın indisi
    int i ;
    for ( i=k ; i< dizi.length - 1 ; i++ ){
        dizi [i] = dizi [i+1] ; }

        return dizi ; }
void ekle (int[] dizi , int k, int yeni ) {
    int i ;
    for (i= dizi.length - 2 ; i>=k ; i--){        → ekleme yapacağı için dizi.length-2
        dizi [i+1] = dizi [i] ; }
          dizi[k]=yeni ; }
```

Bağlı Liste

Fiziksel olarak ardışıl değildir. Her eleman kendisinden sonraki elemanın nerde olduğunu bilir, adresini tutar.

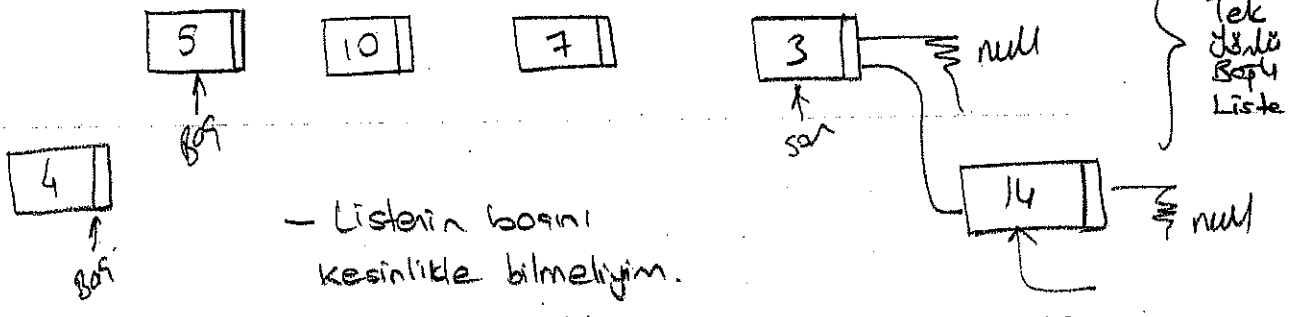


Boyut problemi yok, veriler her bağıl liste elemanı için hafızada yer saplar.
Her düğüm kendisinden sonrakini gösterir.



* Tek Yönlü Bağlı Liste
* Çift " " "
* Dairesel tek yönlü liste
* " çift " "

} çeşitleri //



- Listenin başını kesinlikle bilmeliyim.

Ama sonunu bilmeme gerek yoktur.

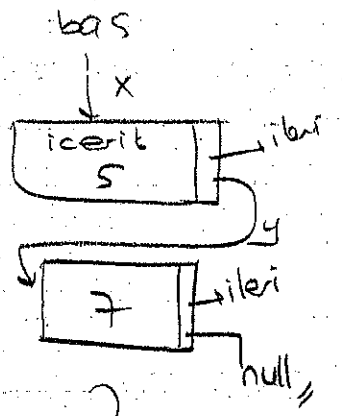
* Ekleme
  - Başa ekleme
  - Sona "
  - Araya "
* Silme
  - Baştan silme
  - Sondan "
  - Aradan "

TEK YÖNLÜ BAĞLI LİSTELER

Listenin basından sonuna doğru olur. Ok yönü bağlı listenin ilerleme yönüdür. Listenin başı kesinlikle tutulmalı. Listenin sonunu tutulması zorunluluktur.

- Elemanlar classda tanımlanır.

```
class Eleman {
    int icerik;
    Eleman ileri;
    Eleman(int icerik){          // icerik parametresi yazılmalı
        this.icerik = icerik;    // icerik değiştirilecek
        ileri = null;            // Başlangıçta bişey olmadığı için ileri = null'dır.
    }
}
```



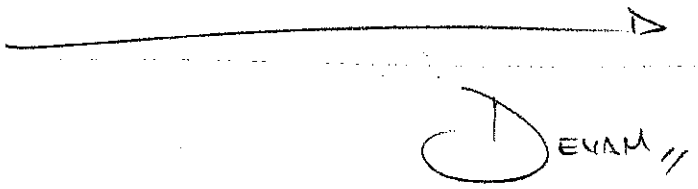

```
Eleman x = new Eleman(5);
Eleman y = new Eleman(7);
x.ileri = y;
Eleman bas = x;
bas = bas.ileri;
```

} İlk eleman x //

} Ama bağlı listeye eleman eklemek.

```
Psvm(){
  Scanner k = new Scanner(System.in);
  int sayi = k.nextInt();
  Eleman bas = new Eleman(sayi);
  for(int i=0; i<9; i++){
    System.out.println("Sayi gir");
    sayi = k.nextInt();
    Eleman yeni = new Eleman(sayi);
    yeni.ileri = bas;
    bas = yeni;
  }
}
```

Bu kısmın mainde yapılışı //



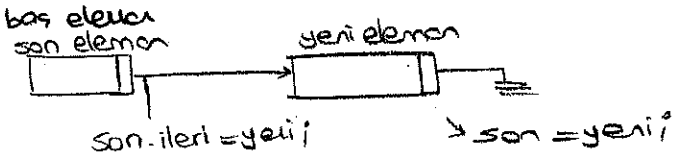
```
public class BagliListe {
  Eleman bas;
  Eleman son;
  public BagliListe(){
     bas=null;
     son=null;
  }
- void basaekle (Eleman yeni){
  if(son==null) {
     bas=yeni;
     son=yeni; }
  else {
     yeni.ileri=bas;
     bas=yeni; }}
```

→ Basa eklenecek eleman parametre olarak geliyor.

// Bas elemanımız yeni eklediğimiz eleman oluyor.

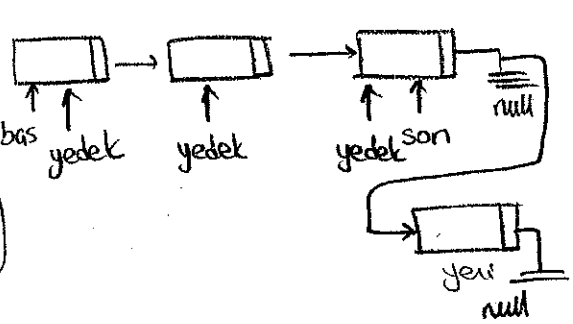
```
- void sonaekle(Eleman yeni){
   if(bas==null){
     bas=yeni;
     son=yeni;}
   else{
     son.ileri=yeni;
     son=yeni;}}
```

// Bu metot bağlı liste bossa veya tek elemanı varsa kullanılır.




```
- void sonEkleme(Eleman yeni){
     if(bas==null){
        bas=yeni;}
     else{
     Eleman yedek=bas;
     while(yedek.ileri!=null){
        yedek=yedek.ileri;
        yedek.ileri=yeni;}}}
```

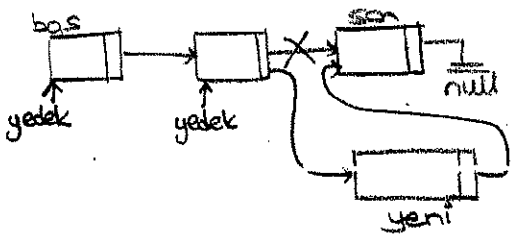
// Bu metot bağlı listede birden fazla eleman varsa kullanılır.



- void arayaekle (Eleman yeni, int sayi) {

```
if (bas==null){
    bas=yeni;}
  else{
   Eleman yedek=bas;
   while(yedek.ileri !=null){
    if(yedek.icerik==sayi)
        break;
    yedek=yedek.ileri;
    }
   yeni.ileri=yedek.ileri;
   yedek.ileri=yeni;
        }}
```

Listdis//



- Eleman arama(int aranan){

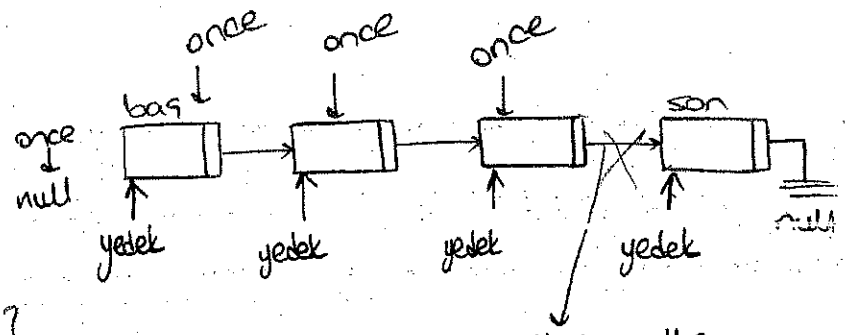
```
Eleman yedek=bas;
while(yedek!=null){
 if(yedek.icerik==aranan){
      return yedek;}
    yedek=yedek.ileri;}
        return null;}
```

- void bastasil(){ // Tek eleman varsa bu metod kullanılır.

```
bas=bas.ileri;
if(bas==null)
     son=null;}
else
Eleman temp=bas;
   bas=bas.ileri;
   temp.ileri=null;
  if(bas==null)
       son=null;}}
```

// Bağıl listede birden fazla eleman varsa bu metod kullanılır.

```
void listeSonuSil(){
Eleman yedek, once;
 yedek = bas;
 once = null;
while(yedek != son){
  once = yedek;
  yedek = yedek.ileri;}
if(once == null){
   bas = null;
   son = null;}
 else{
   once.ileri = null;
   son = once;}
```



ÖRNEK

Bir öğrencinin numarası, adısoyadı, notu tutan bağlı listeyi yazınız.

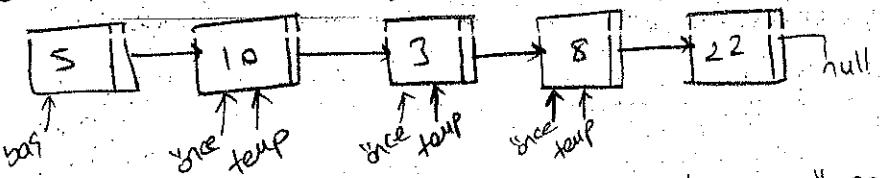
```
class ogrenci{
 int numara;
 String adsoyad;
 int notu;
 ogrenci ileri;
 ogrenci(int numara, String adsoyad, int notu){
 this.numara = numara;
 this.adsoyad = adsoyad;
 this.notu = notu;
 ileri = null;}}

 ogrenci bas;
 bas = null;
 ogrenci K = new ogrenci(123, "Helin", 70);
 bas = K;
 K = new ogrenci(456, "Nesibe", 70);
 bas.ileri = K;
          | → psvm()
 for(i=0; i<5; i++){
 System.out.println("Bilgileri giriniz");
 int numara = t.nextInt();
 int notu = t.nextInt();
 String adsoyad = t.next();
 ogrenci yeni = new ogrenci(numara, notu, adsoyad);
   yeni.ileri = bas;
     bas = yeni;}
```

BAĞLI LİSTE İLE YIĞIT GERÇEKLEŞTİRME

```
class Eleman {
 int icerik;
 Eleman ileri;
 Eleman(int icerik){
 this.icerik = icerik;
 ileri = null; }}

public class stack {
 Eleman ust;
 public stack (){
    ust = null; }

  boolean bosmu(){
    return ust == null; }

  void ekle (Eleman yeni){
  if (bosmu ()) {
      ust = yeni; }
       else {
         yeni.ileri = ust;
           ust = yeni; }}

Eleman sil() {
    Eleman sonuc;
    sonuc = ust;
   if ( !bosmu()){
      ust = ust.ileri;
      }
     return sonuc; }
```

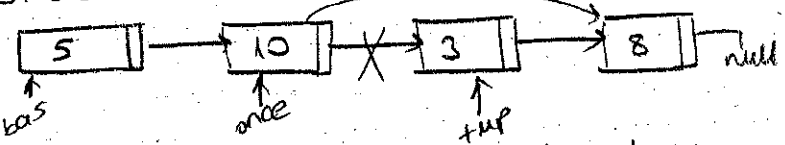
ÇİFT YÖNLÜ BAĞLI LİSTELER

- Ortasına eleman ekleme sadece kendisinden önceki eklenecek elemanı bilindiğinde yapılır.
- Eğer sadece bağlı listenin başını tutuyorsanız. Son elemanı silmek için sondan bir önceki elemana gitmek gerekir



once.ileri=null;

- Eğer ortadan eleman sileceksek silinecek elemanın öncesini bilmek gerekir.

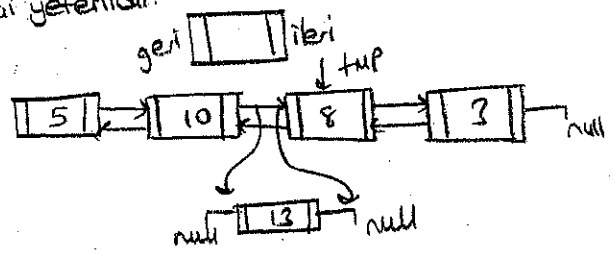


once.ileri = tmp.ileri;
tmp=null;

Tek yönlü...

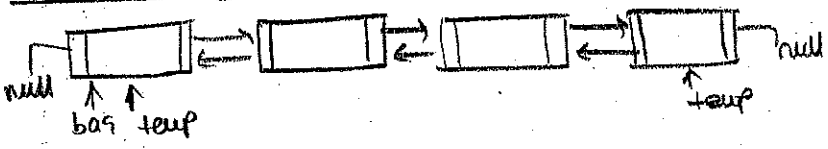
ÇİFT YÖNLÜ

Listenin ortasına eleman eklemek için önüne ekleyeceğimiz elemanı gösteren işaretçi yeterlidir.



tmp.geri.ileri = yeni;
yeni.ileri = tmp;

Sondan eleman silmek



temp.geri.ileri = null;
temp.geri = null;
temp = null;

Ortasından eleman silmek için sadece sileceğimiz elemanın elimizde olması yeterlidir.

```
class cEleman{
 int icerik;
 cEleman geri;
 cEleman ileri;

cEleman(int icerik){
 this.icerik = icerik;
   ileri = null;
   geri = null;
}
```

```
cEleman yeni = new cEleman(10);
cEleman yeni2 = new cEleman(15);

yeni.ileri = yeni2;
yeni2.geri = yeni;
```

yeni2, yeni elemanın ardında ekleme yaptık.

```
public class cBliste {
    cEleman bas;
    cEleman son;
public cBliste () {
     bas = null;
     son = null; }
```

Bağlı liste oluşturuyo

```
Void basaekle (cEleman yeni){
  if (bas == null){
     bas = yeni;
     son = yeni; }
   else {
        bas.geri = yeni;
        yeni.ileri = bas;
          bas = yeni; }}
```

// Hiç eleman yoksa if şartı

// Bir veya birden fazla eleman varsa else şartı

```
void sonaekle (cEleman yeni){
  if (son == null){
      son = yeni;
      bas = yeni; }

   else {
      son.ileri = yeni;
      yeni.geri = son;
        son = yeni; }}
```

// Bir veya birden fazla eleman yoksa else şartı

```
void arayaekle (int x, cEleman yeni){
   if(bas == null){
     bas = yeni;
     son = yeni; }
```

=> int x bu elemanın içeriği

```
    else {
      boolean bulundu = false;
       cEleman tmp = bas;
      while(tmp != null && bulundu == false){
```

Boolean ve temp tutuyoruz boolean true olana kadar ve temp boş olana kadar ilerliyor.

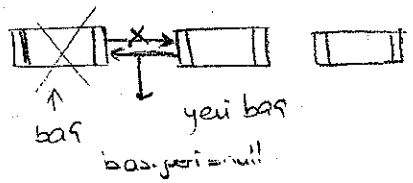
```
   if (tmp.icerik == x){
        bulundu = true; }
      else {
         tmp = tmp.ileri; }}
```

Eğer temp'in içeriği parametre olarak gelen x'in değerine eşitse boolean değeri true olur. Değilse temp ileriye gidiyor.

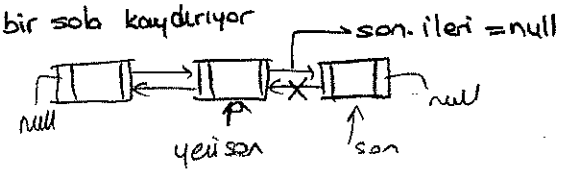
```
        if(bulundu) {
             yeni.ileri = tmp.ileri;
             tmp.ileri.geri = yeni;
             tmp.ileri = yeni;
              yeni.geri = tmp;
        }
```

Bulundu değerimiz true ise araya ekleme gerçekleşir.

```
void bastansil (){
   bas = bas.ileri ;  → Bas silinince kayıyor / Başı bir kaydırıyor
   if(bas==null){
      son=null; }
    else{
      bas.peri=null;}}
```




```
void sondansil(){
   son = son.geri ; → sonu bir sola kaydırıyor
  if(son==null){
      bas=null;}
       else{
       son.ileri=null;
                  }}
```




```
void terstenyaz(){
 cEleman tmp=bas;
 while(tmp.ileri!=null){
    tmp=tmp.ileri; }
 while(tmp.geri!=null){
 System.out.println(tmp.icerik);
    tmp=tmp.geri; }}

   VEYA

 String terstenyaz(){
  cEleman tmp=son;
  String S=" ";
 while(tmp!=null){
 S=S+" "+tmp.icerik;
 }
 tmp=tmp.geri ;
        return S;
                  }}
```

ÇİFT YÖNLÜ DAİRESEL BAĞLI LİSTE

```
public class dBagliliste{
cElemon bas;
cElemon son;
dBagliliste(){
 bas=null;
 son=null;}
 void ekle(cElemon yeni){
  if(bas==null){
   bas=yeni;
   bas.geri=bas;
   bas.ileri=bas;}
  else{
   yeni.ileri=bas;
   yeni.geri=bas.geri;
   bas.geri.ileri=yeni;
   bas.geri=yeni;
   bas=yeni;}}

void sil(){
 if (bas.ileri==bas){
      bas=null;}
  else{
   bas.geri.ileri=bas.ileri;
   bas.ileri.geri=bas.geri;
    bas=bas.ileri;}}
```

—BÜTE HAZIRLIK— (VERİ YAPILARI)

Arama Algoritmaları

* Ardışık Arama
* İkili Arama
* Jump Search
* Interpolation Search

Sıralama Algoritmaları

* Seçmeli (Selection) Sort
* Bubble (Kabarcık) Sort
* Insertion (Eklemeli) Sort

} $O(n^2)$

* Merge (Birleştirmeli) Sort
* Quick (Hızlı) Sort
* Heap Sort

} $O(n(\log n))$

* Sayma Sıralama
* Radix (Taban) Sıralama

} $O(n)$

1) ARDIŞIK ARAMA

0	1	2	3	4	5	6	7
5	7	8	13	20	2	4	19

En iyi: 1. Adım //

En kötü: (n+1). Adım // [Dizide Eleman olmaması]

Ortalama Durum ⇒ $\frac{n+1}{2}$. Adım //

* Dizinin sıralı olmasına gerek yok
* Elemanı tek tek bulana kadar ilerler
* Bulunduğu elemanın indisini gönderir
  Bulamazsa "-1" gönderir

- Java Kodu -

```
int ardisikarama (int[] dizi, int aranan){
for (int i=0 ; i< dizi.length ; i++){
    if (aranan == dizi[i])
        return i ;   // En iyi durum
        return -1 ;  // En kötü durum
    }}
```

2) İKİLİ ARAMA

* Sadece sıralı diziler üzerinde çalışır. Çünkü temel amacı her defasında küçültüp daha dar bir alanda arama yapmaktır.

0	1	2	3	4	5	6	7
5	10	15	23	42	54	65	70

bas=0, orta, bas, orta, bas, son, son=0

* Dizinin ortanca elemanını bulup ona göre küçültüp buluyor.

(int) orta = $\frac{(bas+son)}{2}$

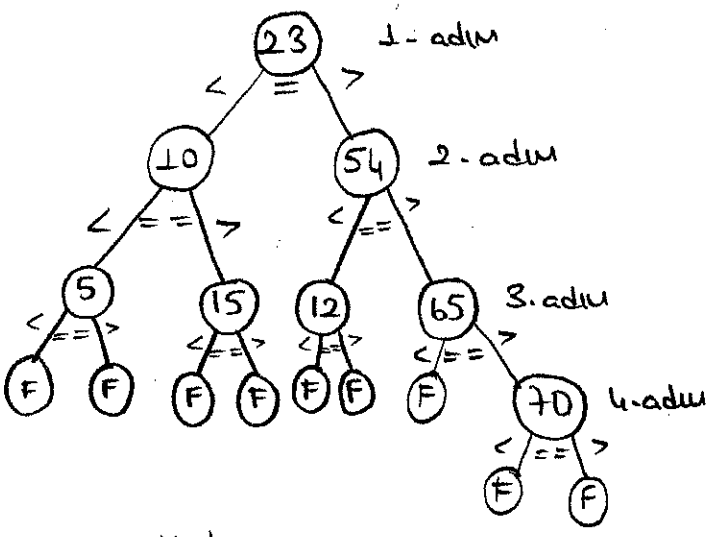
Aranan = 54

orta = $\frac{0+7}{2} = 3$ // orta = $\frac{4+7}{2} = 5$ //

Aranan = 78

orta = $\frac{0+7}{2} = 3$ // orta = $\frac{4+7}{2} = 5$ // $\frac{6+7}{2} = 6$ // Bulamadık...

① bas > son ise (6. adıma git)
② orta = (bas+son) / 2
③ Eğer dizi[orta] = aranan bulundu ve cık
④ Değilse eğer aranan > dizi[orta] , bas = orta+1 (1. adıma git)
⑤ Değilse eğer aranan < dizi[orta , son = orta-1 (1. adıma git)
⑥ return -1 (Çıkış)



Aranan = 78
Aranan = 47

* Aranan bulunmadığında
En kötü 4. adımda //

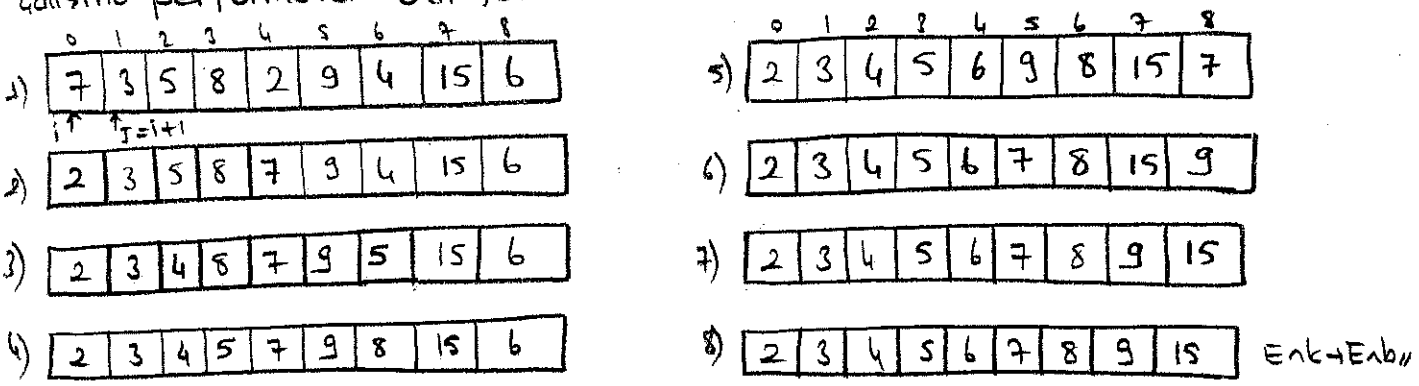
- Java Kodu -

```
int ikiliarama (int[] dizi , int aranan){
int bas, son;
bas=0;
son= dizi.length-1;
while ( bas<=son){
 int orta = (bas+son)/2;
 if (aranan = dizi[orta]) {
   return orta;}
  else{
  if (aranan> dizi[orta]){
      bas=orta+1;}
   else{
      son=orta-1;}
                    }
        return -1;
                    }
                    }
```

SIRALAMA ALGORİTMALARI

* SELECTION SORT

Enk bul başa getir, enb bul başa getir mantığıyla çalışır. Dizideki enb ya da enk elemanı bulup başa getiren sıralama algoritmasıdır. En iyi, en kötü ve ort. çalışma performansı $O(n^2)$'dir.



- Java Kodu -

```
void selectionsort(int[]A){
 int indis;
 for(int i=0; i<A.length-1; i++){
   indis=i;
 for(int j=i+1; j<A.length; j++){
    if (A[j] < A[indis])
        indis=j;
    }
   if (indis != i)
     int ara = A[indis];
      A[indis]=A[i];
       A[i] = ara; }}
```

SR//

27	78	45	8	32	56
8	78	45	27	32	56
8	27	45	78	32	56
8	27	32	78	45	56
8	27	32	45	78	56
8	27	32	45	56	78
8	27	32	45	56	78

* KABARCIK SIRALAMA (BUBBLE SORT)

- Bu sıralama alg. ardışık iki elemanı karşılaştırarak değişim yapar. Her adımda en büyük elemanını dizinin sonuna götürür.

8/ 9, 5, 8, 3, 1

1. Tur

9 5 8 3 1
5 9 8 3 1
5 8 9 3 1
5 8 3 9 1
5 8 3 1 9

2. Tur

5 8 3 1 9
5 3 8 1 9
5 3 1 8 9
5 3 1 8 9

3. Tur

5 3 1 8 9
3 5 1 8 9
3 1 5 8 9
3 1 5 8 9

4. Tur

3 1 5 8 9
1 3 5 8 9

5. Tur

1 3 5 8 9
1 3 5 8 9

1 3 5 8 9
1 3 5 8 9

1 3 5 8 9
1 3 5 8 9

1 3 5 8 9
1 3 5 8 9

— Java Kodu —

```
void kabarciksirala(int[]A){
 boolean degisim=true;
 while(degisim){
 degisim=false;
 for(int i=0; i<A.length-1; i++){
   if(A[i+1]<A[i]){
      degisim=true;
      int ara=A[i+1];
      A[i+1]=A[i];
        A[i]=ara;
                    }}}
```

9/ Sıralı bir dizi
1, 5, 10, 15, 20

1. Tur

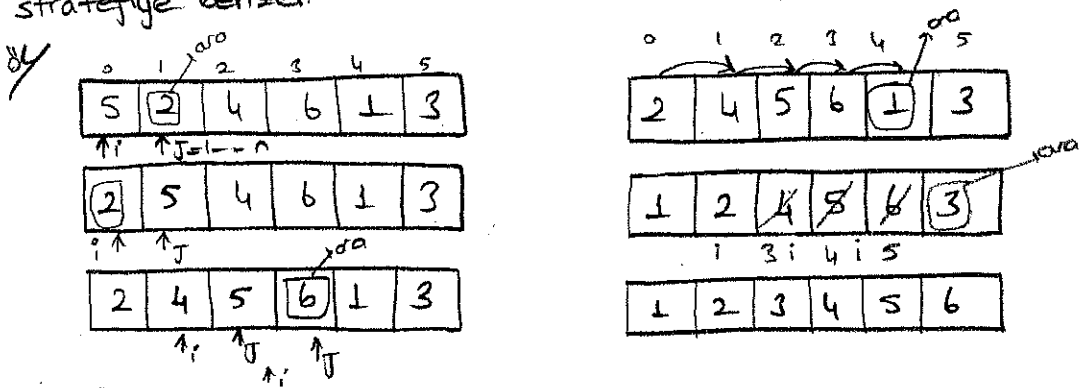
değişim=false

1 5 10 15 20}
1 5 10 15 20}
1 5 10 15 20}

1 5 10 15 20}
1 5 10 15 20}

* EKLEMELİ SIRALAMA (INSERTION SORT)

Yeni gelen bir sayıyı kendisinden önceki sayılardan uygun yere yerleştirir. Bu sıralama algoritması oyun kapıtlarını sıralarken istediğimizde kullandığımız stratejiye benzer.



Eğer $i >= 0$ && $A[i] > ara$

$A[i+1] \leftarrow A[i]$

$i = i - 1$;

$A[i+1] \leftarrow ara$.

- Java Kodu -

```
void eklemelisiralama (int[] A) {
  int i, J, ara;
for (J=1 ; J<A.length ; J++){
     i=J-1;
     ara = A[J];
while (i>=0  && A[i]>ara){
     A[i+1] = A[i];
      i=i-1;
     }
      A[i+1] =ara;
                    }}
```

HASHING

* Hem arama, hem ekleme algoritması olarak kullanılır.
* 1 adımda bulmayı garanti eder.
* Sıralı olmak zorunda değil

*** Elimizdeki veriyi kullanarak, elden geldiği kadar o veriden bir tamsayı elde etme işlemidir. Bu elde edilen tam sayı dizi şeklinde tutulan verilerin indisi olarak kullanılarak verilere tek bir adımda ulaşmamızı sağlar.
- Hashing Fonk.
- Hash Tablosu
- Çakışma

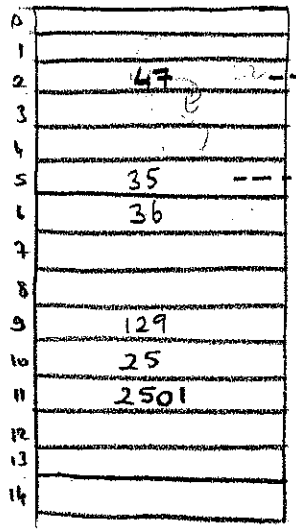
HASH FONKSİYONLARI
- Rakam Seçme
- Katlama
- Bölme
- Orta Kare
- Çıkarım
- Radix Dönüşüm

ÇAKIŞMA
- Doğrusal sınama ∝
- Çift Hashing
- Quadritic Hashing
- Chain - Zincir Kuralı ∝

ÖRNEK

N boyutlu bir dizi hashFonksiyonunun ↳ 0 ____ N-1.

X →	25	129	35	2501	47	36	N=15 ?
h(x) →	10	9	5	11	2	6	h(x) = X % 15



47 → [62] * Burada kullanılan hash. fonk. - bölme yöntemidir.

35 → [65] [95] h(65) = 65 % 15 = (5) → Mor ile yazılı yerler çakışmayı zincir kuralı örneğidir.

Rakam Seçme

Anahtar üzerinde belirlenmiş bazı basamaklara birleştirecek bir pozisyon oluşturur.

Ör// Rakam Seçme : 2. ve 5.

h (0(3)3 4(7)5 6 7 8) = 37. noluncu indise 03347568 değerini yaz.

h (02345568) = 25. " " 02345568 " "

Katlama

- Anahtar birkaç parçaya bölünür, parçalar birkaç parçayı ayrılarak bulunur.

Ör// SSN = 123-45-6789

123 + 456 + 789 = 1368. indise 123-45-6789 yaz.

Bölme (* *)

Ör// SSN = 123-45-6789
Tablo boyutu (N) = 1000
h(SSN) = 123456789 %1000
= 789 //

* Bölme yöntemi kolaydır. Ancak çakışma olma ihtimali yüksektir. Ancak tablo boyutunu asal seçersek çakışma ihtimalini azaltmış oluruz.

Orta Kare

Anahtarın karesi alınır, sonucun orta karesi seçilerek tablodaki kısmı seçilir. N = 1000

Ör// anahtar = 3121
3121^2 = 9740641

Tablo B.	Alacağı değer
1000	3 rakam
100	2 rakam
10	1 rakam

Çıkarım

Anahtar değerinin bazı kısımları seçilerek tablodaki değerleri bulunur. Rakam seçmede farklı toplu / yaygara seçim yapabiliyoruz.

Ör// anahtar = 123456789
Çıkarım = 1234 → Tablo boyutu (1000)
1289
3456
6789

Radix Dönüşümü

Anahtar değeri başka bir tabana dönüştürülür.

Ör// anahtar = 1238

$(1238)_{10} = (2326)_8$

hash (2326) = 2326 % 1000 ← Tablo boyutu

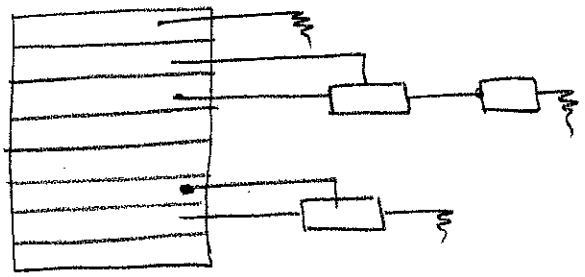
= 326

* 326. indise 2326 yaz. //

ÇAKIŞMA

→ ZİNCİR HASHING

Aynı pozisyona gelen kayıtlar bağlı listelerle yapılır.
Ekleme → Liste başı
Silme → Uygun listede arama

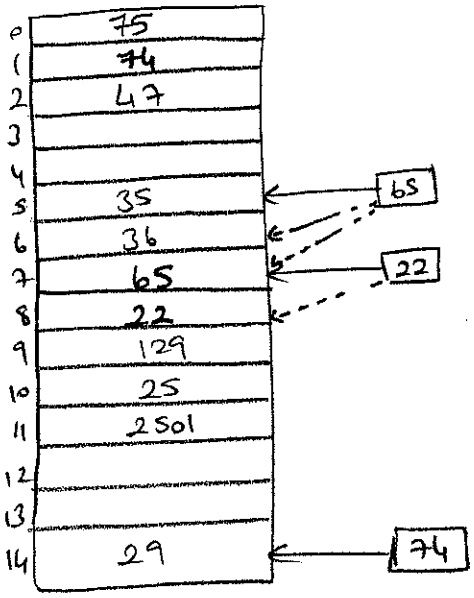


- Aynı değerler çok fazla üretilirse ardışık aramaya dönüşür.
- Aynı değer üretilmemesi için tablo boyutu asal seçilir.

→ DOĞRUSAL SINAMA

Aynı pozisyona gelen ikinci kayıt, ilk gelen kayıttan sonra yerleştirilir.

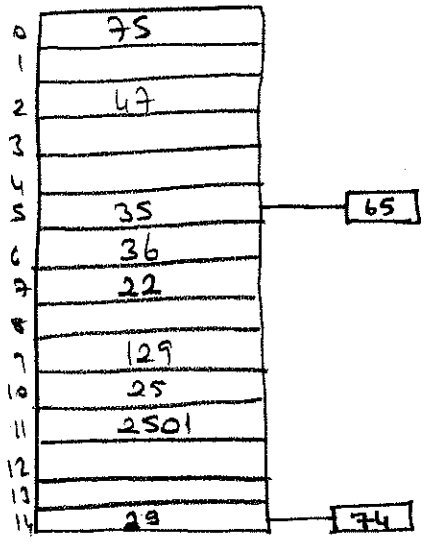
ÖR: N=15
X = {75, 47, 35, 36, 65, 22, 129, 25, 2501, 29, 74}



* Arama yaparken 65%15=5. elemana bakarız. Sonra bir adım daha ilk boşluğa baktım 65 varsa döngü bulup biter yoksa bulamaz alır.

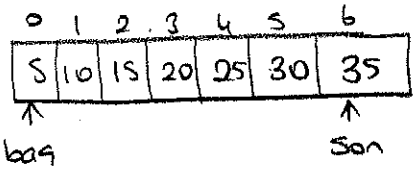
* 74 %15 = 14.
(14+1) mod 15 → indisin başına gider

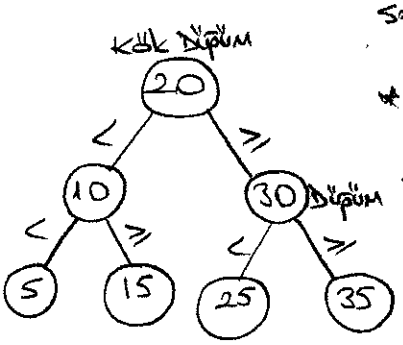
} Doğrusal Sınama



} Zincir Kuralı
(Bağlı liste uyguladuk)

AĞAÇLAR (TREES)

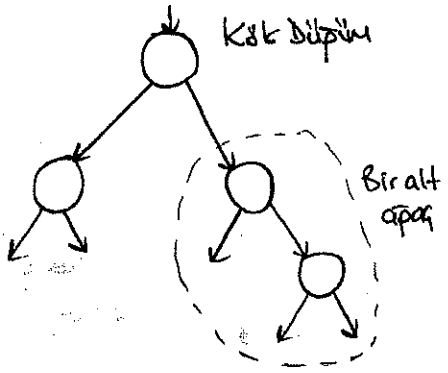




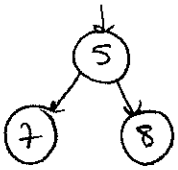
Sol küçük
Sağ büyük

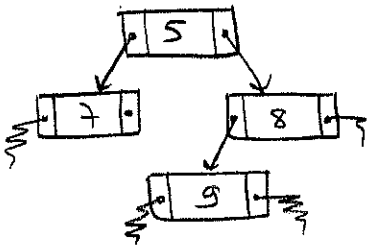
* İkili Arama Ağacı (Binary Search Tree)
* Ekleme - Arama çok kolay

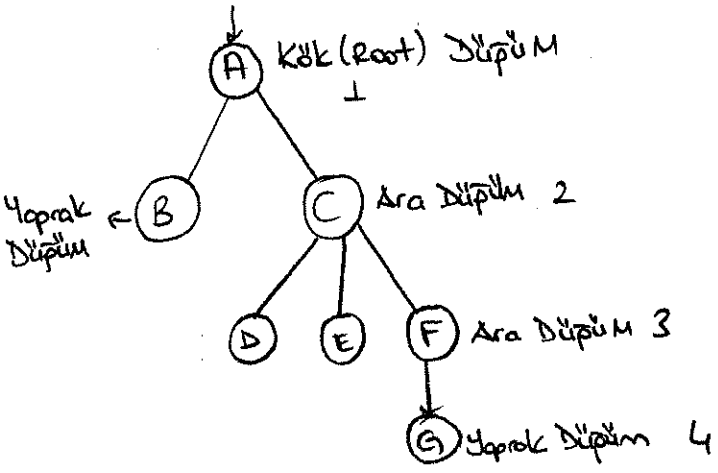
- Kök işaretçisi ve sonlu sayıda düğümleri ve onları birbirine bağlayan veri modelidir. Bu veri modeli soyağacına benzemektedir. (Ata, torun--)
[20'nin torunu 15, 5-15 atası 10],,
- Bağlı liste, Kuyruk, Yığıt ⟶ Doğrusal Veri Yapısı,,
Ağaç ise ⟶ Doğrusal Olmayan veri yapısı,,
- Dosya sis., organizasyon şemaları, programlama ortamlarında kullanırız.



* Her ağaç düğüm ve kenarlardan oluşur.
* Arama işlemi Bağlı listeye göre hızlı.
* Ağacı göstermek için kullandığımız yapı?







4 seviyeli ağaç

Düğüm (Node): Ağacın herbir elemanına düğüm denir.
Kök Düğüm (Root Node): Ağacın başlangıç düğümüdür.
Çocuk: Bir düğüme doğrudan bağlı dan düğümlere o düğümün çocukları denir.
Kardeş düğüm: Ataları aynı olan veya aynı düğüme bağlı düğümlere denir.
Derece: O düğümün çocuk sayısını gösterir.
Düzey ve Derinlik: İki düğüm arasındaki yolun üzerinde bulunan düğümlerin sayısına düzey denir. Bir düğümün köke olan uzaklığına derinlik denir.
Yükseklik: Bir düğümün en uzak yaprak düğüme dan mesafesidir.
Yaprak: Ağacın en altında bulunan ve çocuğu olmayan düğümlerdir.

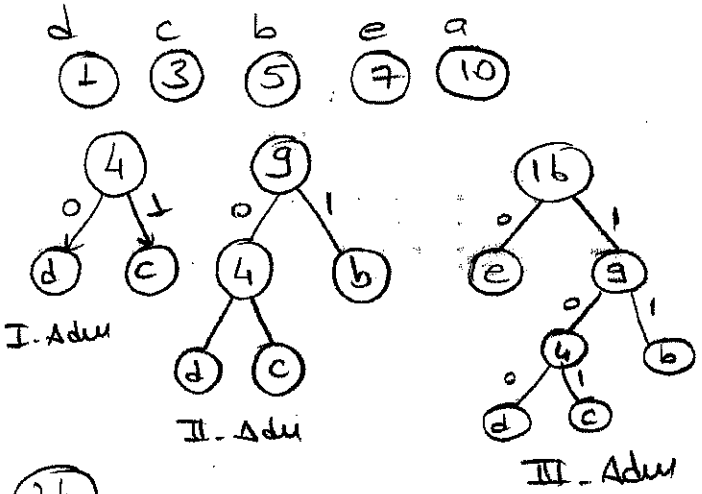
AĞAÇ TÜRLERİ

* İkili Arama Ağacı (Binary Search Tree)
- Her düğümün iki çocuğu vardır.
- Sol küçük, Sağ büyük olmalı.

* Kodlama Ağacı
- Bir kümedeki karakterlere kod atanası yapmak için kullanılan bir ağaçtır.

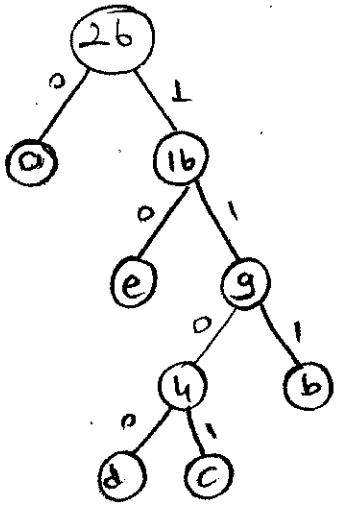
Huffman Kodlama Ağacı

Frekans	Bit Sayısı
a: 10	000 → 30
b: 5	001 → 15
c: 3	010 → 9
d: 1	011 → 3
e: 7	100 → 21
	78 //



Sıkıştırılmış kod

a: 10	0
b: 5	111 → 15
c: 3	101 → 12
d: 1	1100 → 4
e: 7	10 → 14
	55 //



IV. Adım.

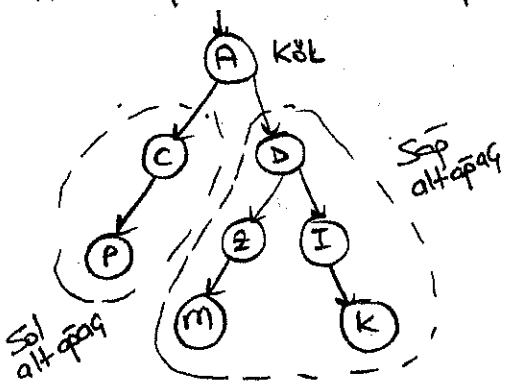
55/78 oranında veri sıkıştırılabilir.

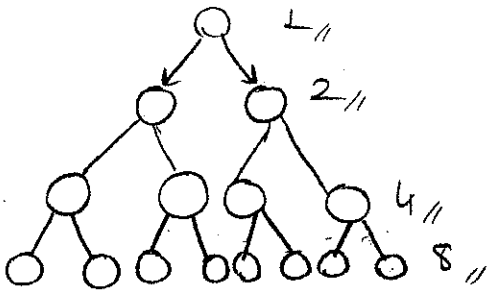
* Sözlük Ağacı

Bir sözlükte bulunan sözcüklerinin tutulması için kurulan bir ağaç şeklidir.

* Heap Tree
  - Max Heap
  - min Heap

* İkili Ağaç (Binary Tree)
  - Sol çocuk ve sağ çocuk olmak üzere iki çocuğu vardır.
  - En alttan yukarı çıktıkça kök düğüme ulaşırız.
  - Her düğümün iki çocuğu olmak zorunda





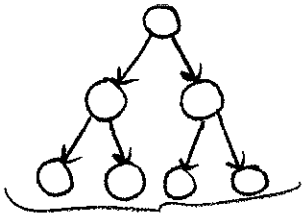
Düğüm $S = 2^0+2^1+2^2+2^3$ → Derinlik Sayısı

Yükseklik $= 2^0+2^1+2^2+\dots+2^k = \frac{2^{k+1}-1}{2-1}$ → Düğüm Sayısı

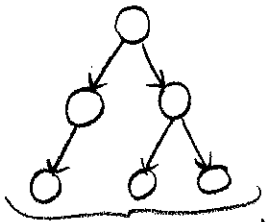
k → derinlik

* Full Binary Tree (Tam İkili Ağaç)

Her yaprağı aynı derinlikte olan ve yaprak olmayanların hepsinin iki çocuğu olan ağaca denir.



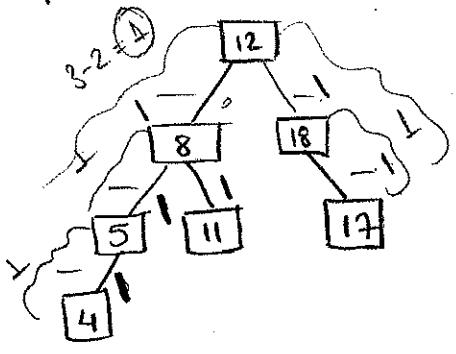
Tam ikili Ağaç //



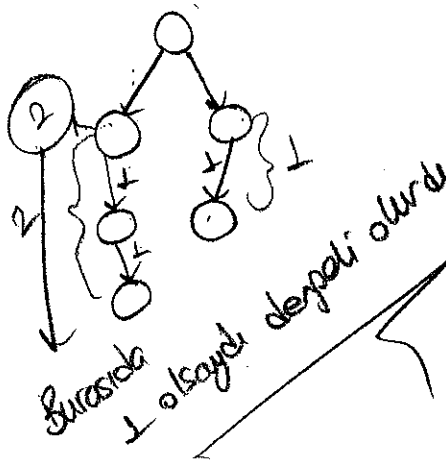
Tam ikili ağaç değil //

* Dengeli Ağaç

Yükseklikleri ayrılmış ağaçlardır. Bütün düğümler için sol alt ağaç ile sağ alt ağaç arasındaki yükseklik farkı 1 ise dengeli ağaçtır.



Dengeli //



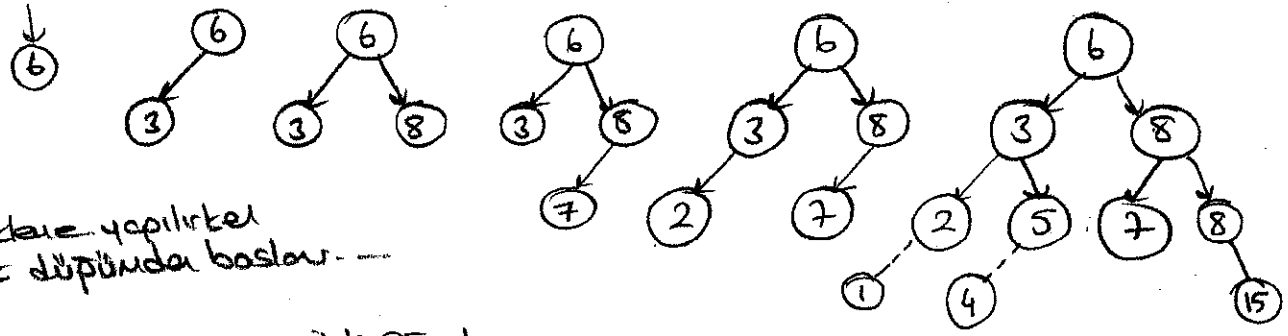
Burada 1 olsaydı dengeli olurdu //

! Tüm düğümlere bakmak gerekir.

BİNARY SEARCH TREE (İKİLİ ARAMA AĞACI)

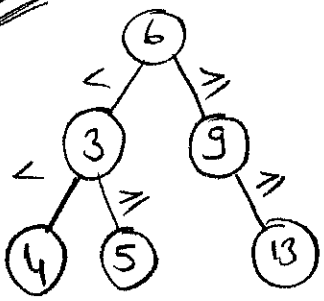
- Her düğümün max. iki çocuğu olur ve her düğümdeki değer sol alt ağaçtaki değerlerden büyük, sağ alt ağaçtaki değerlerden küçüktür.

ÖR 6, 3, 8, 7, 2, 5



- Ekleme yapılırken kök düğümden başlanır.

ÖR



* İkili Ağaçtır
* İkili Arama Ağacı değildir çünkü 4 < 3 olmadığından yanlış

- Ağaçta oraya ekleme yoktur.

- Java Kodu -

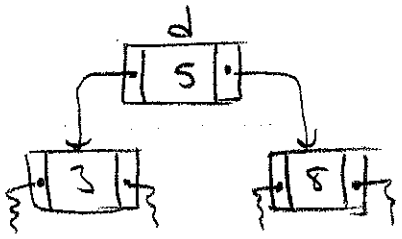
```
class Dugum {
int icerik;
Dugum sol, sag;
Dugum (int icerik){
this.icerik = icerik;
sol = null;
sag = null;
```

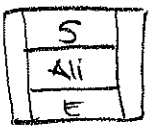
ÖR Öğrencilerin bilgilerini tutan bir ağaç

```
class Dugum {
int no;
String adsoyad;
char cinsiyet;
Dugum sol, sag;
Dugum(int no, String adsoyad, char cinsiyet){
this.no = no;
this.adsoyad = adsoyad;
this.cinsiyet = cinsiyet;
sol = null;
sag = null; } }
```

Nesne Oluşturduk

```
Dugum d = new Dugum(5);
Dugum d1 = new Dugum(8);
  d.sag = d1;
Dugum d2 = new Dugum(3);
  d.sol = d2;
Dugum d3 = new Dugum(5, "Ali", 'E');
Dugum d4 = new Dugum(8, "Ayse", 'K');
```

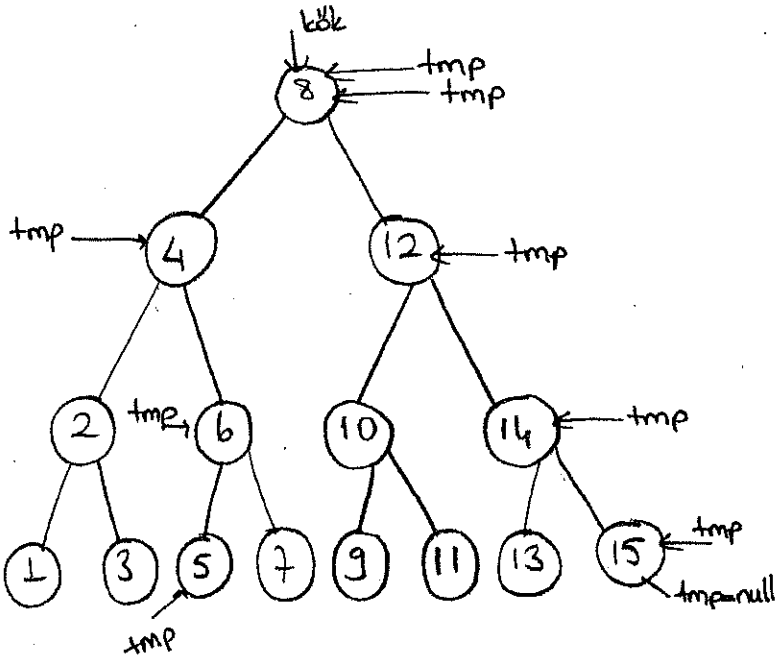




- Nereye ekleyeceğimiz seçtiğimiz duruma göre değişir. Bu yüzden bir dal seçmeliyiz.

-Java Kodu Devamı-

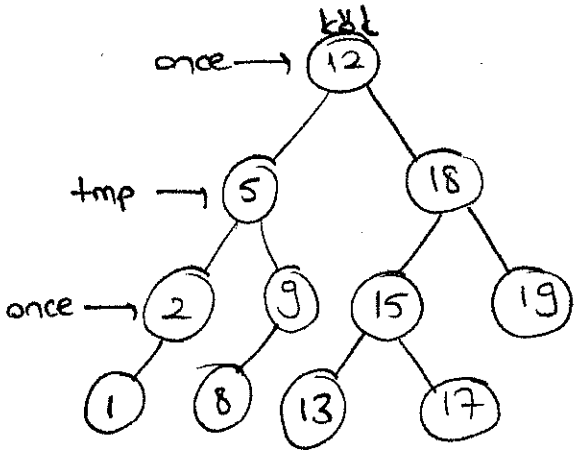
```
public class Agac{
Dugum kok;
public Agac(){
  kok=null;
        }
Dugum Ara (int aranan){
Dugum tmp=kok;
while (tmp!=null){
if (tmp.icerik==aranan){
  return tmp;
}
else if (aranan<tmp.icerik){
  tmp.=tmp.sol;
                  }
 else {
    tmp.=tmp.sag;
   }}
    return null;}}

Dugum enkucuk(){      //Agactaki en küçük eleman bulur
Dugum tmp=kok;
while (tmp.sol!=null)
   tmp.=tmp.sol;
    return tmp;
              }

Dugum enbuyuk(){      //Agactaki en büyük eleman bulur
Dugum tmp.=kok;
while (tmp.sag!=null)
  tmp=tmp.sag;
   return tmp;
              }
```



(5) — elemanını arıyoruz

(17) — elemanını arıyoruz

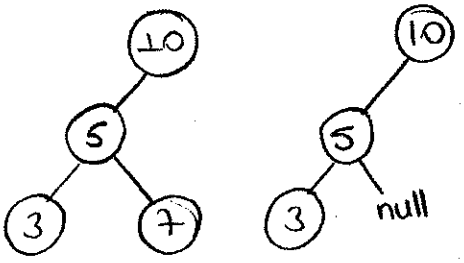
```
void ekle (Dugum yeni) {
Dugum once = null;
Dugum tmp = kok;
while (tmp != null) {
once = tmp;
if (yeni.icerik < tmp.icerik) {
  tmp = tmp.sol; }
else {
   tmp = tmp.sag;
}
if (once == null) {
   kok = yeni; }
 else if (yeni.icerik < once.icerik) {
 once.sol = yeni; }
 else {
   once.sag = yeni; } }
```



1 eklenek için //
13 eklenek için //
8 eklenek için //

AĞAÇTAN ELEMAN SİLME (*)

1. Silinecek eleman yaprak ise (Hiç çocuğu yok ise)
2. Silinecek elemanın 1 çocuğu var ise
3. Silinecek elemanın 2 çocuğu var ise




```
void Sil (int anahtar) {
Dugum tmp = kok;
Dugum ata = kok;
boolean solmu = true;
while (tmp.icerik != anahtar) {
  ata = tmp;
 if (anahtar < tmp.icerik) {
     solmu = true;
     tmp = tmp.sol;
   }
 else {
     solmu = false;
     tmp = tmp.sag;
   }
  if (tmp == null)
      return;
   }
```

Temel kod
- elemanı bulmak için

```
if (tmp.sol == null && tmp.sag == null) {
if ( tmp == kok)
  kok = null;
else if (solmu)
 ata.sol = null;
 else
 ata. sag = null; }
```

Yaprak ise silme kodu //

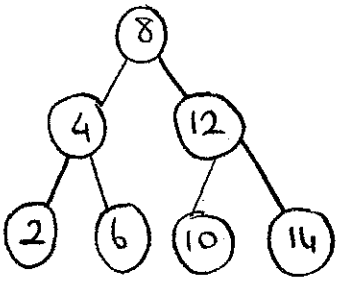
```
else if ( tmp. sag == null) {
  if ( tmp == kok)
   kok = tmp. sol;
  else if (solmu)
   ata.sol = tmp.sol;
   else
    ata.sag = tmp.sol;
```

Sağ cocuğu yok ise //

```
else if ( tmp.sol == null) {
  if ( tmp == kok)
    kok = tmp.sag. ;
  else if (solmu)
    ata.sol = tmp.sag;
  else
    ata.sag = tmp.sag; }
```

Sol cocuğu yok ise //

- AĞAÇ DOLAŞIM ALGORİTMALARI -



Preorder (Kök-Sol-Sağ) 8-4-2-6-12-10-14
inorder (Sol-Kök-Sağ) 2-4-6-8-10-12-14
Postorder (Sol-Sağ-Kök) 2-6-4-10-14-12-8

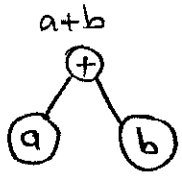
```
void Preorder (Dugum d.){
System.out.println (d.icerik); Kök
if ( d.sol != null)
Preorder (d.sol)
if (d.sag != null)
Preorder (d.sag)
}
```

```
void inorder (Dugum d){
if (d.sol != null){
 inorder (d.sol);
System. out. println (d.icerik); kök
if (d.sag != null)
inorder (d.sag);
}
```

```
void Postorder (Dugum d){
if ( d.sol != null)
Postorder (d.sol)
if (d.sag != null)
Postorder ( d.sag);
System.out.println (d.icerik); Kök
}
```

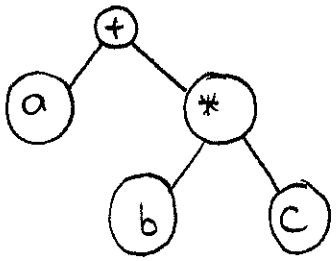
İFADE AĞAÇLARI

Yaprakları operand, ara düğümleri operatör olan ağaçlardır.



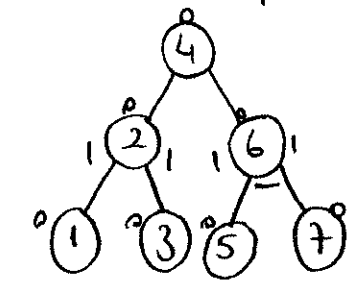
+ab → Preorder → Prefix
a+b → Inorder → Infix
ab+ → Postorder → Postfix

Ör: a + b * c

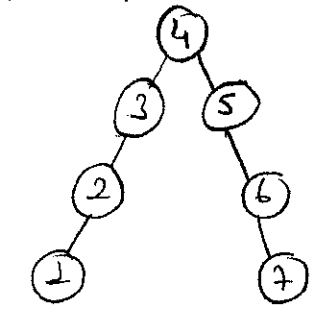


-DENGELİ AĞAÇ-

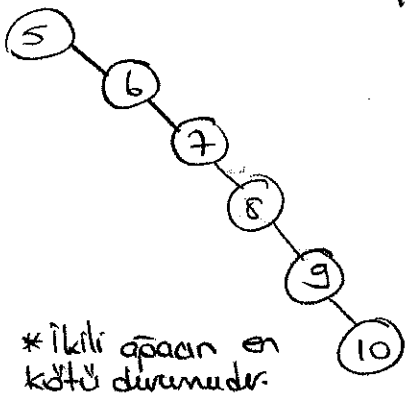
* log n zamanda aramayı garanti eder.
* Sol alt ağaç ile sağ alt ağaç arasındaki yükseklik max 1 olan ağaçlardır.



* Tam ikili ağaçtır ve her ikili ağaç, dengelidir.



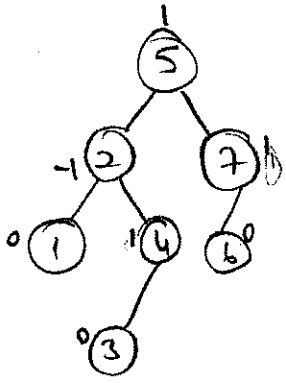
* Seviye farkı 3 ve 5 olduğunda (2) yüksekliğiyle bozuyor. Dengesiz.



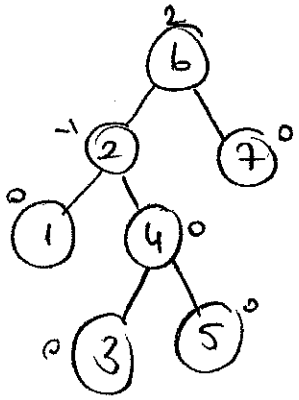
* İkili ağacın en kötü durumudur. Dengesizdir.

AVL AĞACI (Adelsen – Velskıl – Londrs)

* Dengeli bir ikili arama ağacıdır. Denge koşulu oldukça basittir. Ve ağacın derinliğinin log n'de kalmasını garanti eder.
* AVL ağacında her düğümün sol ve sağ alt ağaçlarının boy farkı en az 1'dir.



* Dengeli
* AVL ağacın özelliklerini taşıyor



* 6 denklik koşulunu bozdu.

AVL Ağacı Rotasyonlar ile Denpeyi Sağlar.

Tek Rotasyon

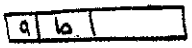
① d düğümünün sol çocuğunun sol alt ağacının soluna
② " " sağ " sağ " " sağına

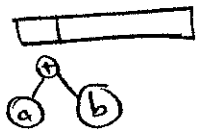
Çift Rotasyon

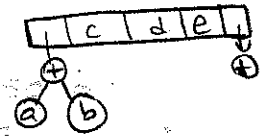
① d düğümünün sol alt ağacının sağına
② d " sağ " " soluna

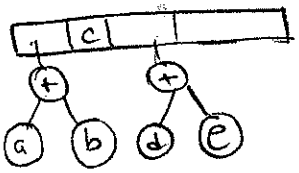
-Postfix bir ifadeden ifade ağacı oluşturken;
* Eğer operand ise yığıta ekle,
* Eğer gelen sembol operatör ise yığıttan iki operand al, sonra sonucu yığıta ekle.

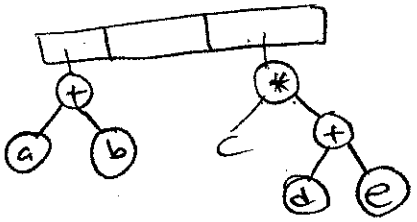
Ör ab+cde+** Postfix - infix ifade ağacı kullan.

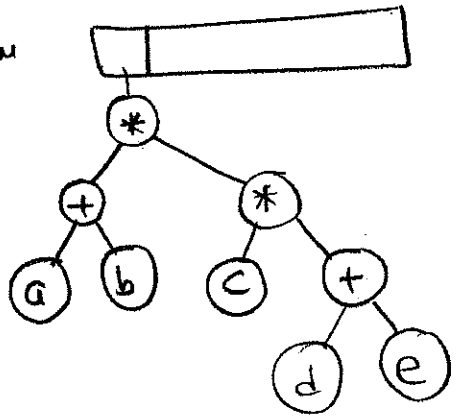
1. adım 

2. adım 

3. adım 

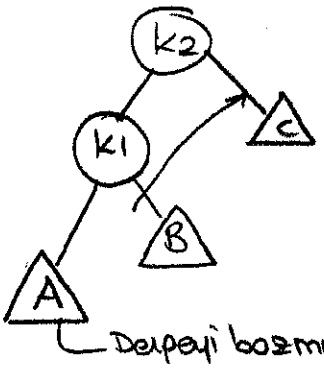
4. adım 

5. adım 

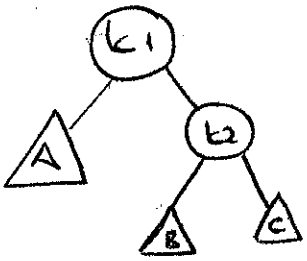
6. adım 

} a+b*c*(d+e) // } infix ifade //

TEK ROTASYON

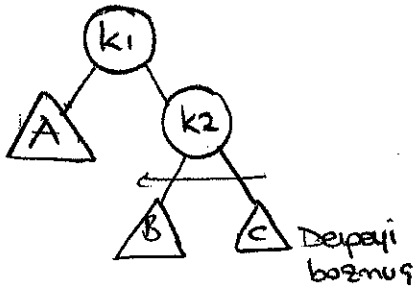


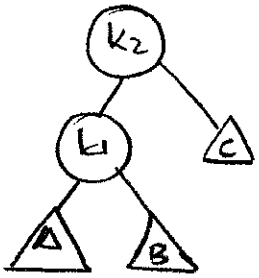
① k2'nin sağına – k1'in sağını bağla.



k2.sol = k1.sag
k1.sag = k2

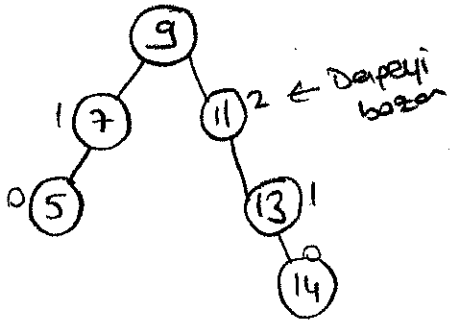
②

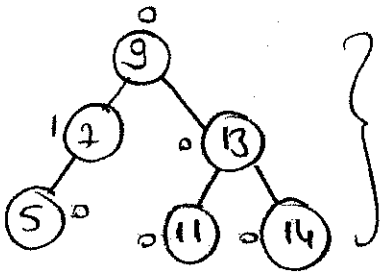




k1.sag = k2.sol
k2.sol = k1

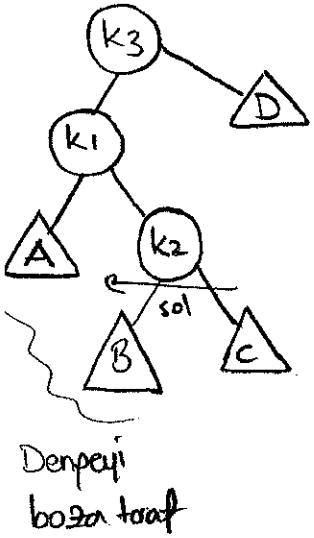
Tek Rotasyon



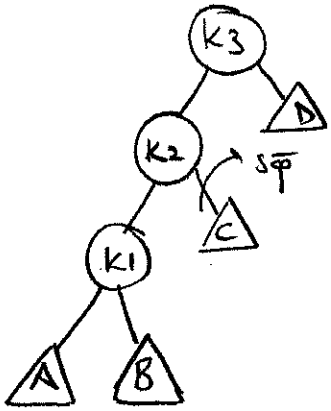


Denpeli oldu//

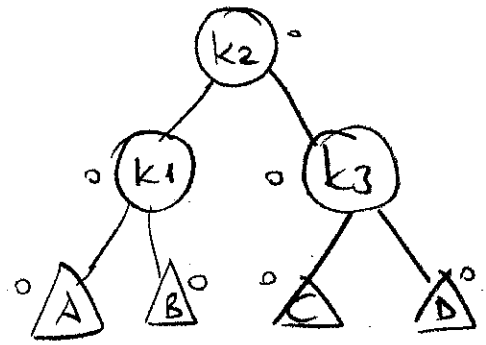
ÇİFT ROTASYON



1. adım

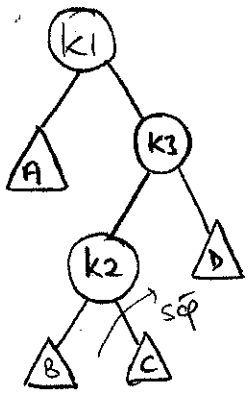


2. adım

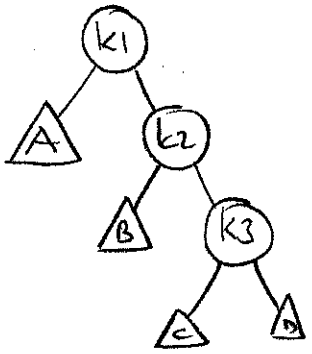


– Denpeli iktili Ağaca Aşağı//

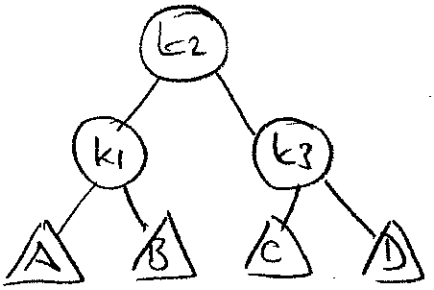
(2)



1. adım



2. adım



2017-2018 VERİ YAPILARI ÇIKMIŞ VİZE SORULARI

1) A)

```
for (int i=0; i<N-1 ; i++){
    min = i ;
    for ( J = i+1; J <N-1; j++)
    if ( A[J] . < A [min])
    min = J ;
    temp. = A[i] ;
    A[i] = A[min] ;
    A[min] = temp; }
```

Cost	Times
C_1	N
C_2	N-1
:	(N-1)(N-1)
:	(N-1)(N-2)
:	(N-1)(N-2)
	N-1
	N-1
C_8	

N^2 //

B)

```
i = 0
while (i<N * N*N) {
sum = sum + N/2 i
i++ ;
} return sum;
```

Bip. $O(n^3)$ //

C)

```
i=0; sum=0;
while (i< log(m)) {
sum + = i * log(m) ;
i++ ;
}
```

$O(\log(m))$ //

D)

```
sum=0;
for (i=0; i<L; i++.) {
sum = func1(L) + func2 (L);
}
```

$O(L^4)$ //

E)

```
int sum = 0 ;
for (r=0; i<L ; i++){
  sum = sum + L* L;
  L = L/2 ;
}
```

$O(\log L)$ //

2) Düğüm ve yığıtı kullanarak öğrencinin adını, soyadını, numarasını ve notunu tutan ve ekleme, silme metodunu yazınız.

```
class Eleman {
int numara;
String adsoyad;
int not;
Eleman ileri;
Eleman(int numara, String adsoyad, int not){
this.numara = numara;
this.adsoyad = adsoyad;
this.not = not;
ileri = null;
}}
```

(Düğüm)

```
public class Yigit {
Eleman ust;
public Yigit(){
ust = null;
}
boolean bosmu(){
return ust == null;
}
void ekle(Eleman yeni){
if(!bosmu){
yeni.ileri = ust;
ust = yeni;
}
else{
ust = yeni;
}}
void sil(){
Eleman sonuc;
if(!bosmu()){
sonuc = ust;
ust = ust.ileri;
return sonuc;
else{
return null;
}}
```

* Yığıttaki not ort bulan metod

```
double ort(){
int top = 0; sayac = 0;
while(!bosmu()){
top = top + ust.not;
sayac++;
ust = ust.ileri;
}
double ort = top/sayac;
return ort;
}
```

Sabit dizi ile tanımlanmış bilgileri kuyruk dizisinde verin ve ekle metodunu yazın.

```
class il{
int plakaNo;
String ad;
int nüfus;
il(int plakaNo, String adi, int nufus){
this.plakaNo = plakaNo;
this.Ad = Ad;
this.nufus = nufus;
public class Kuyruk{
int N, bas, son;
il[] dizi;
public Kuyruk(int N){
this.N = N;
dizi = new il[N];
bas = 0;
son = 0;}
void ekle(il e){
if(!dolumu()){
dizi[son] = e;
son = (son+1)%N;
}}
```

-AĞAÇ KODU-

```
class Dugum {
int icerik;
Dugum sol, sag;
Dugum (int icerik) {
this.icerik = icerik;
sol=null;
sag=null;
public class Agac {
Dugum kok;
public Agac () {
kok = null; }

Dugum Ara (int aranan){
Dugum tmp=kok;
while( tmp != null) {
if (tmp.icerik == aranan){
return tmp; }
else if (aranan < tmp.icerik){
tmp=tmp.sol; }
else {
tmp=tmp.sag;
}}
return null; }}

Dugum enkucuk() {
Dugum tmp=kok;
while (tmp.sol != null) {
tmp= tmp.sol; }
return tmp;
}

Dugum enbuyuk() {
Dugum tmp = kok;
while (tmp.sag != null) {
tmp = tmp.sag; }
return tmp;
}
```

```
void ekle (Dugum yeni) {
Dugum once = null;
Dugum tmp = kok;
while (tmp != null) {
once = tmp;
if (yeni.icerik < tmp.icerik) {
tmp = tmp.sol;
}
else {
tmp = tmp.sag; }
if (once == null) {   // eleman yoksa
kok = yeni; }
else if (yeni.icerik < once.icerik) {
once.sol = yeni;
}
else {
once.sag = yeni; }}
```

Eleman Bulan Temel kod

```
void Sil (int anahtar) {
Dugum tmp = kok;
Dugum ata = kok;
boolean solmu = true;
while (tmp.icerik != anahtar) {
ata = tmp;
if (anahtar < tmp.icerik) {
solmu = true;
tmp = tmp.sol; }
else {
solmu = false;
tmp = tmp.sag; }
if (tmp == null)
return; }
```

Yaprak ise silme kodu

```
if (tmp.sol == null && tmp.sag == null) {
if (tmp == kok) kok=null;
else if (solmu)
ata.sol = null;
else
ata.sag = null;
```

Sağ çocuğu yalnız ise

```
else if (tmp.sag == null) {
if (tmp == kok)
kok = tmp.sol;
else if (solmu)
ata.sol = tmp.sol;
else
ata.sag = tmp.sol;
```