KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BIL 205 VERİ YAPILARI
DERS NOTLARI

2013-2014 Güz Dönemi

VERİ YAPILARI : (Data structures)

(117)

$n! = n * (n-1)!$

$f(n) = n * f(n-1)$ ...

```
int f (int n)
{
  f(n == 0) : return 1
  else return n * f(n-1)
}
```

→ hiçbir şeye işaret etmeyen pointer NULL'a eşitlenir.

106 Boşluk oluştur nextler birbirini göstersin.

Bağıl Liste:

(Singly Linked list)

head = head → next

*** Jonathan Shewchuk
Jrs @ cory.ercs

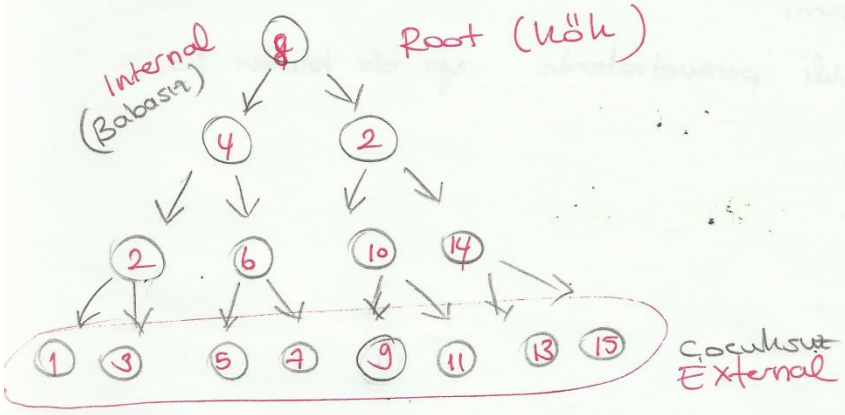
Dairesel Bağlı: Son eleman ilk elemanı işaret ediyor.

Last in first out (Lifo)

Ekleme en sondan silme en baştan başlanır.

✓ Chapter 6 geçen sene işlenmedi!

BINARY TREE:



$\underbrace{2^n - 1}_{\text{Level}}$

{ } (kod grubunu çalıştırmamak için.

STD (Standart Template Library)
(şablon)

```
#include<string>
using std::string;
// ...
string s = "to be";
string t = "not" + s;
string u = s + "or" + t;
if (s>t)
cout<<u;
```

*

```
char ch = 'Q';
char *p = &ch;
cout << *p << endl;
```

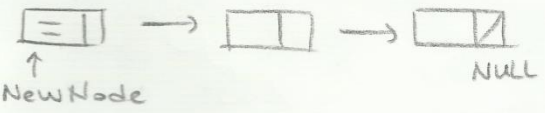
*

```
passenger *p;
p = new passenger;
p -> name = "Pocahontas";
```

pnin işaret ettiği structure name'i;

NOT: p->name ≡ (*p).name

** new... (Bellekte yeri yer açılıyor)



önce p oluşuyor bellekte yer açılıyor sonra işaret ediyor.

Garbage Collection: Bellekte dinamik olarak oluşturulan nesnelerin arasından işe yaramayanların silinip, kullandığı bellek alanının boşaltılması.

- C++ da yok. kütüphanelerle yapılır.

delete p; // pointer'ı siler.

Member indexing: 43 lock.

Argue Passing: (Veri Aktarım)

Overloading: Aynı fonk farklı parametrelerde ya da birden fazla parametre ile tanımlanacak.

Class structure.

private
public

```
class Counter
{
public:
Counter();          // kendisi otomatik olarak koşan bir fonk. (constructor)
int getCount();
void increaseBy(int x);
private:
int count;
};
```

} fonk.

} değişkenler.

```
#include "counter"
void main()
{
counter ctr;   // constructor koşar. Class'ın değişkenlerine ilk değer atar.
cout << ctr.getcount() << endl;
}
```

** Passenger * pp1 = new Passenger

↓

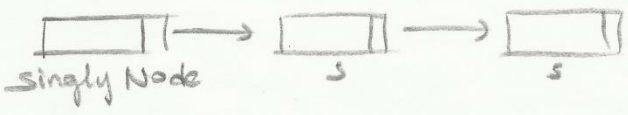
Passenger nesnesine ptr olarak erişebiliriz.

64 - Vect

delete [] → işiniz bitince koşmalı.
// destructor.

```
Vect * p = new Vect;
|
delete p;
```

→ Başka bellek alanları varsa destructure içinden silmek gerekiyor.



} singly Linked list.

V1 [1,2,3]
V2 [4,5,6]
V3 [7,8,9]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

69:
```
#include <vector>
using namespace std;
vector <int> scores 100 ...
scores.resize (scores.size()+10);
```

*) `s.find ("dog", 3)`
`s.substr (7,5)`
a dog is a dog

*) `#ifndef CREDIT_CARD_H` (if not defined)
`Vect *p = new Vect;`
=
`delete p;`

`#pragma once`

* OBJECT ORIENTED DESIGN:

Inheritance: Class, bir class'ı miras alır metotlarını kullanabilm. için.

Miras veren: base class, parent //, super //

Miras alan: child class, derive //, subclass

```
class Student : public Person {
String major;
int grdYear;
```
Student: miras alan. public: en kapsamlı miras.

```
void Person::print
```
Yaptığı iş yazar.

(71)
95.sayf.
NOT: Miras alsan bile private değişkenlere erişemezsin. Name protected yapılırsa erişilebilir.

constructor: Değişkenlere ilk değer atar.

`:name(nm)` ≡ `name = nm`

Polimorfizm: (Çokbiçimlilik)

Birbirlerinden miras alanlar. Nesnelere işaret etmesi. Farklı türden nesnelere onların ortak metotları ile erişebiliriz.

```
Person *pp[100];
pp[0] = new Person(...);
pp[1] = new Student
```

static binding , dynamic binding (↗ tipi değişebilir.)

↳ ilgili fonksiyona virtual yazdığımızda miras alan class nesnesi printini koşabilir.

- Ortak fonk virtual olarak tanımla

abstract: Fonk sadece header'ı var. Miras alan içini doldurur.

Circle, rectangle and Triangle → shape classları

draw = 0; // ortak kullanıkları fonk.

Abstract class.

overload: Farklı türden fonk. peşpeşe tanımlayabiliriz.

Exception: C++ ve C# da hata olma ihtimali olan yerlere.

```
try {
if (divisor == 0)
throw zerodivide ("...")
}
```
→ Hata mesajı fırlatılmış

```
catch (...) {
}
```
→ Hataya dair çözüm.

CHAPTER 3 : JİNAN BURADAN İTİBAREN ...

Name	Mike	Rob	Paul	Anna	Rose	Jack	
Score	1105	750	720	660	590	510	

GameEntry

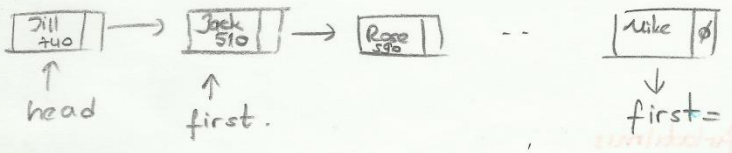
scores

SinglyNode: Bağlı liste elemanlarının herbirini temsil ediyor.

Insertion Sort:

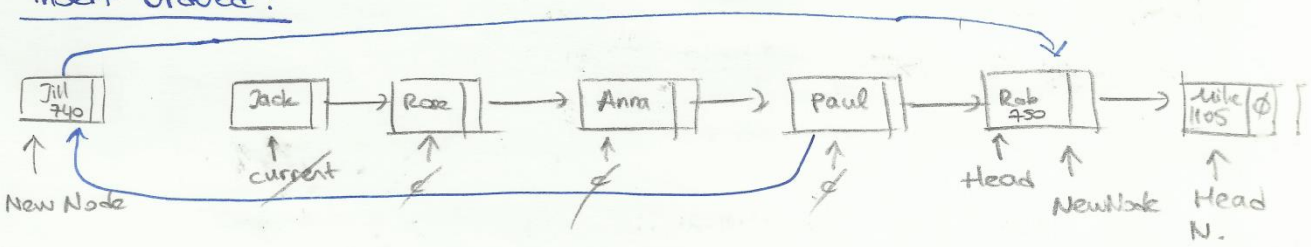
Singly Linked list: Tek yönlü bağlı liste.

- insertorder : sıralı ekler
- addfront : Başına ekler
- removefront : Siler ⟶ Parametre almıyor.



- Sağdaki ptr hangi nesneye işaret ediyorsa soldaki de aynısına işaret eder.

★ **insert Ordered :**

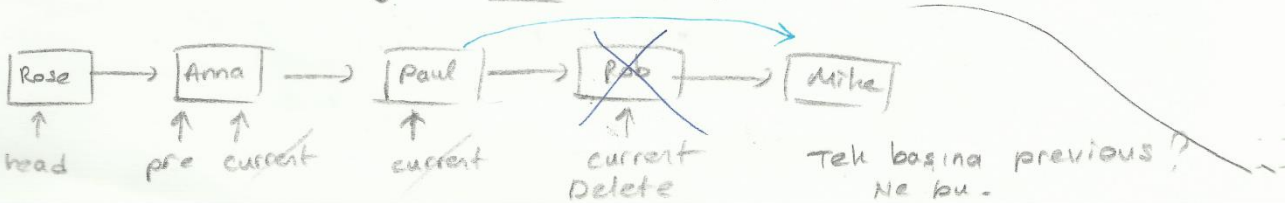


- Head kime işaret ediyorsa current de ona işaret eder.
- Tek yönlü head ptr ⟶ NULL set.

Remove ordered :

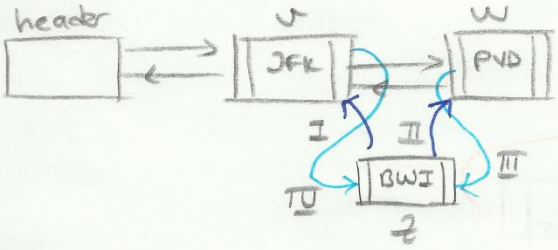
head = NULL ise boş bir listeden bir şey silemezsin.

Jack silinirse başlangıç Rose olur.

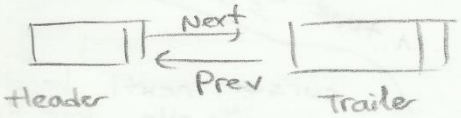


Doubly Linked Lists: (Çift yönlü bağlı liste)

Son eleman —next→ trailer
İlk eleman —next→ header
} işaret eder.

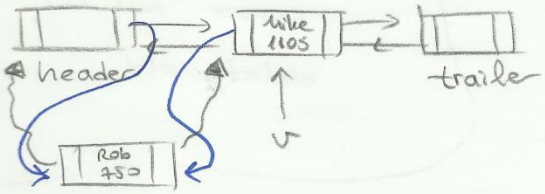


addfront:



⇝ Header ve ilk eleman başına ekler

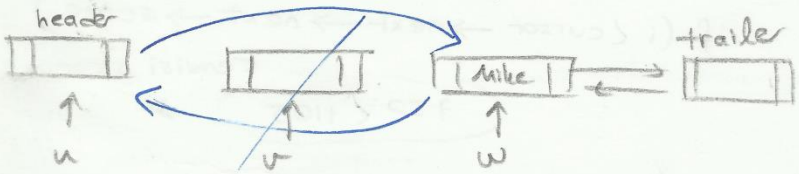
Trailer'dan önceki bağlı listenin son elemanı.



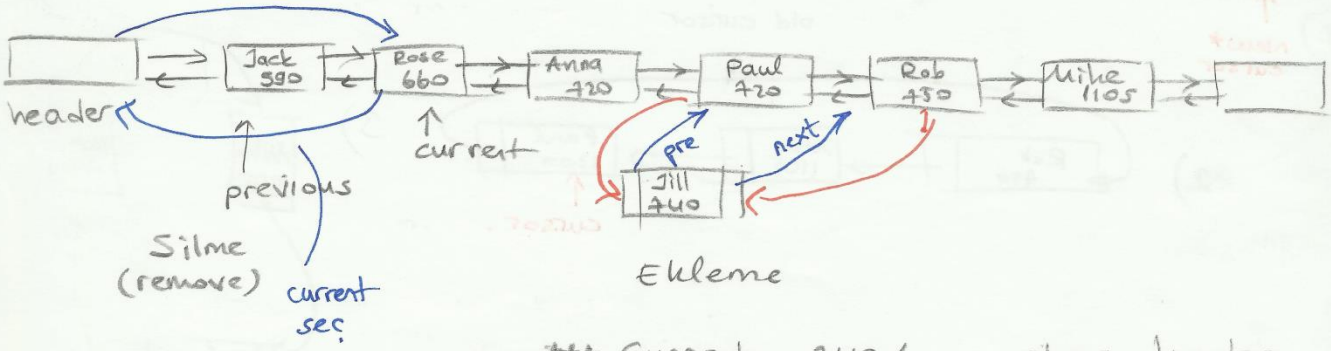
Trailer previousu → Mike'dan önceki

remove front: Look yourself.

remove: Direkt verilen düğümü siler.

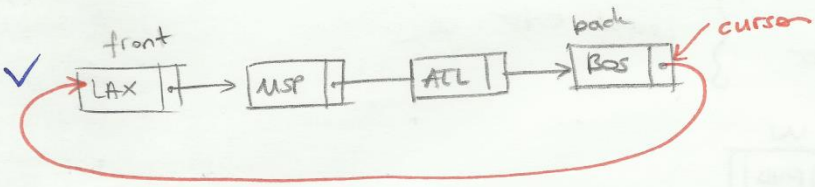


Ptr kaldırıldı ama hala bellekte. (Delete yap)
↳ Artık buna erişemeyiz.



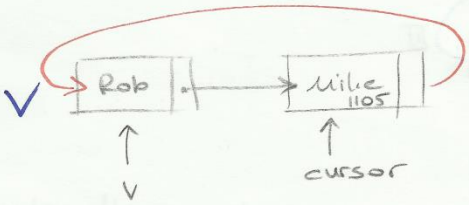
*** Current 440'a kadar olana kadar yer değiştirir.

DAİRESEL BAĞLI LİSTE:

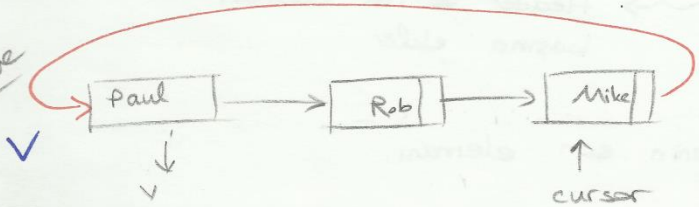


Add ve remove

cursor'un nexti silinir veya nextine eklenir.



Ekleme

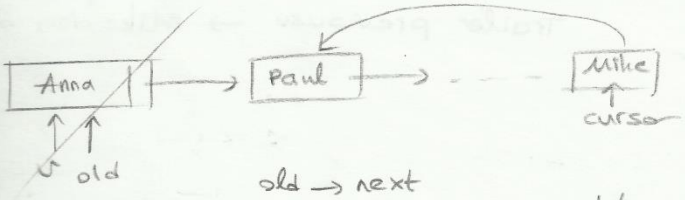


1 tane kalınca silme:

cursor nexti kendi ise sile sile bir tane kalmış.

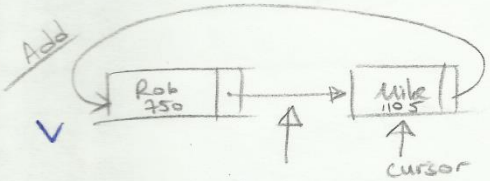
```
if (old==cursor)
    cursor NULL;
delete old;
```

Silme




```
old -> next
cursor -> next -> next
delete old;
```

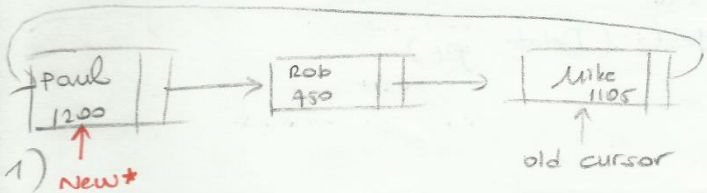
Add




```
if (i < cursor->next->next->score)
```

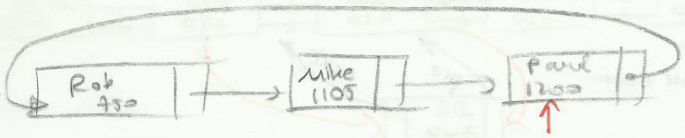
Kendisi skoru

750 < 1105



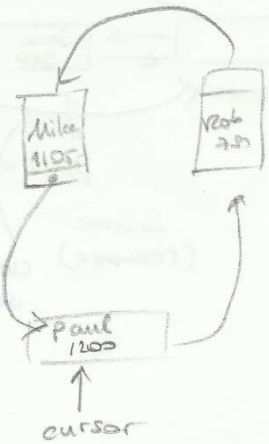

```
if (i > cursor -> score)
```

1) New* cursor

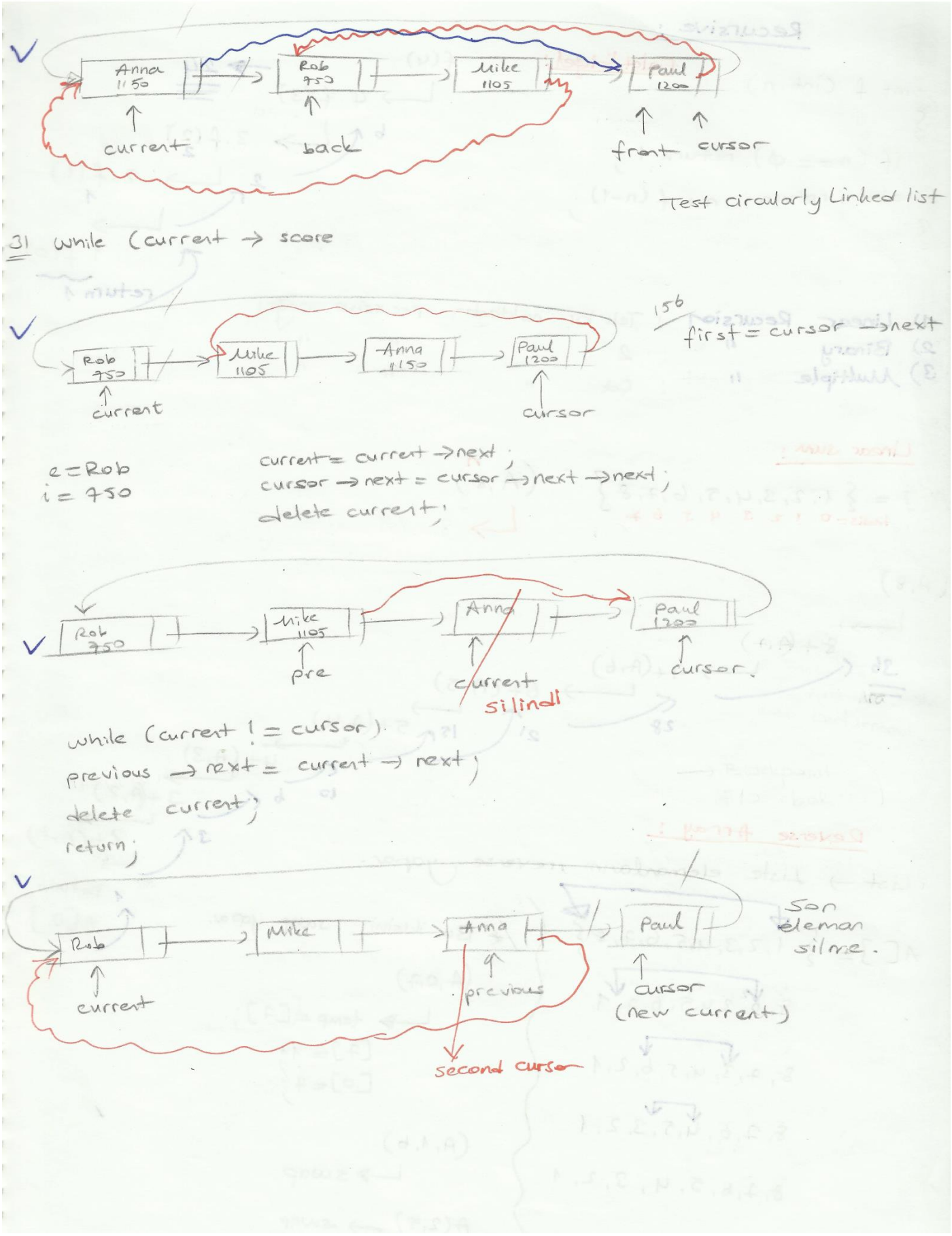


2)

3)



1, 2, 3 Aynıdır


Anna 1150
Rob 750
Mike 1105
Paul 1200
current
back
front
cursor
Test circularly Linked list
31 while (current → score
Rob 750
Mike 1105
Anna 1150
Paul 1200
current
cursor
156
first = cursor → next
e = Rob
i = 750
current = current → next;
cursor → next = cursor → next → next;
delete current;
Rob 750
Mike 1105
Anna
Paul 1200
pre
current
silindi
cursor
while (current != cursor)
previous → next = current → next;
delete current;
return;
Rob
Mike
Anna
Paul
current
previous
cursor
(new current)
second cursor
Son eleman silme.

Recursive :

Faktöriyel:

```
int f (int n)
{
  if (n == 0) return 1;
  else return n * f(n-1);
}
```

f(4) → 24
4 · f(3)
3 · f(2)
2 · f(1)
1 · f(0)
return 1

1) Linear Recursion : Tek bir noktadan recursive geçer
2) Binary " : 2 " " "
3) Multiple " : Çok " " "

Linear sum :

A[] = { 1, 2, 3, 4, 5, 6, 7, 8 }
indis = 0 1 2 3 4 5 6 7

(A,8)

(A,8)
8+(A,7)
7+(A,6)
6+(A,5)
5+(A,4)
4+(A,3)
3+(A,2)
2+(A,1)
return A[0] = 1
36
28
21
15
10
6
3

Reverse Array !

rList → Liste elemanlarını reverse yapar.

A[] = { 1, 2, 3, 4, 5, 6, 7, 8 }
8, 2, 3, 4, 5, 6, 7, 1
8, 7, 3, 4, 5, 6, 2, 1
8, 7, 6, 4, 5, 3, 2, 1
8, 7, 6, 5, 4, 3, 2, 1

/* Bu ikisini swap yapar.
(A,0,7)
→ temp = [7];
[7] = 1;
[0] = 7;

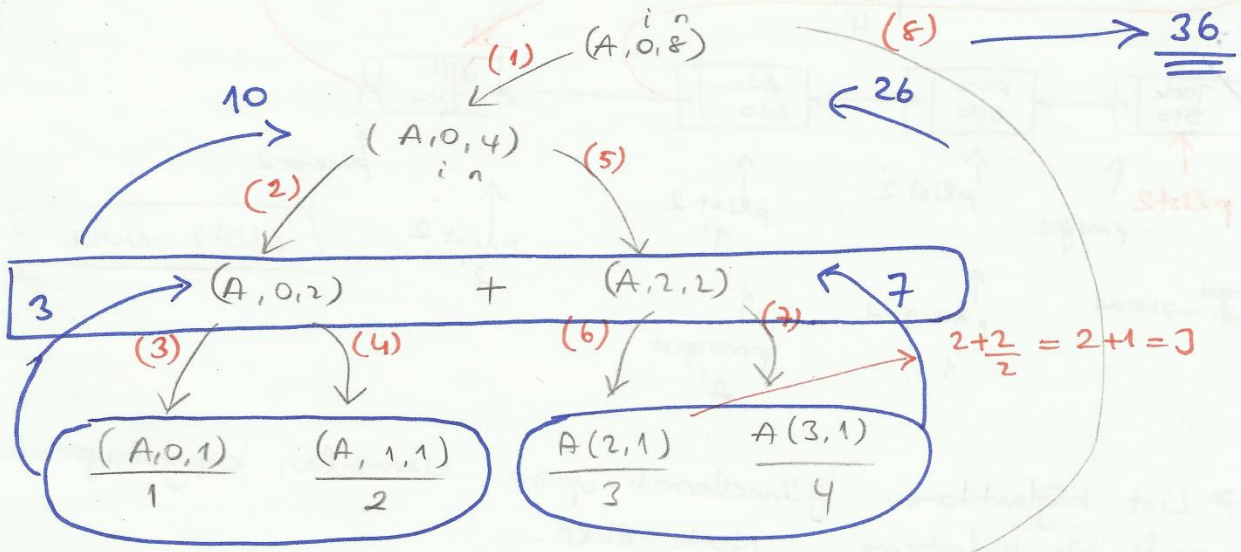
(A,1,6)
→ swap

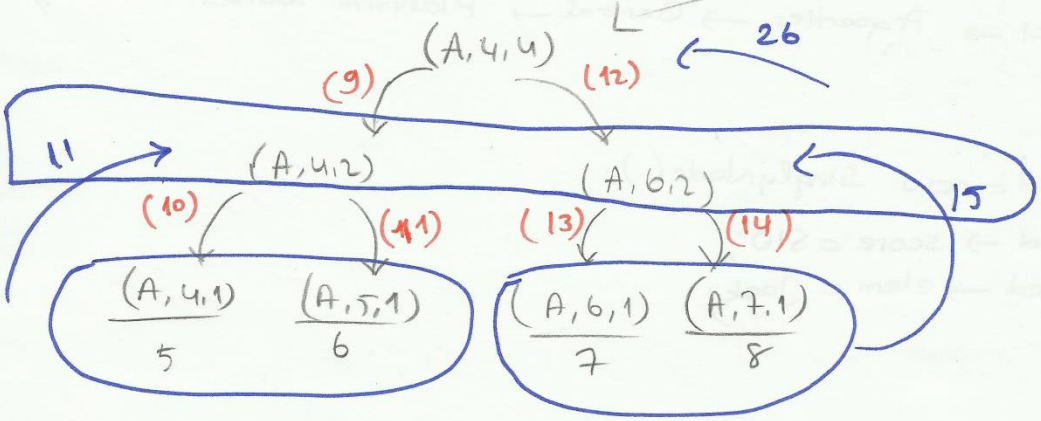
A(2,5) → swap
A(3,4) → swap
A(4,3) → ~~swap~~ */

Tail Recursion : Son kısmında recursive yapılır.

** **Binary Sum:** iki kez recursive yapılır.

```
if (n==1) return A[i];
else
{
 int sum = binarySum (A,i,n/2) + binarySum (A,i+n/2,n/2);
 return sum;
}
```





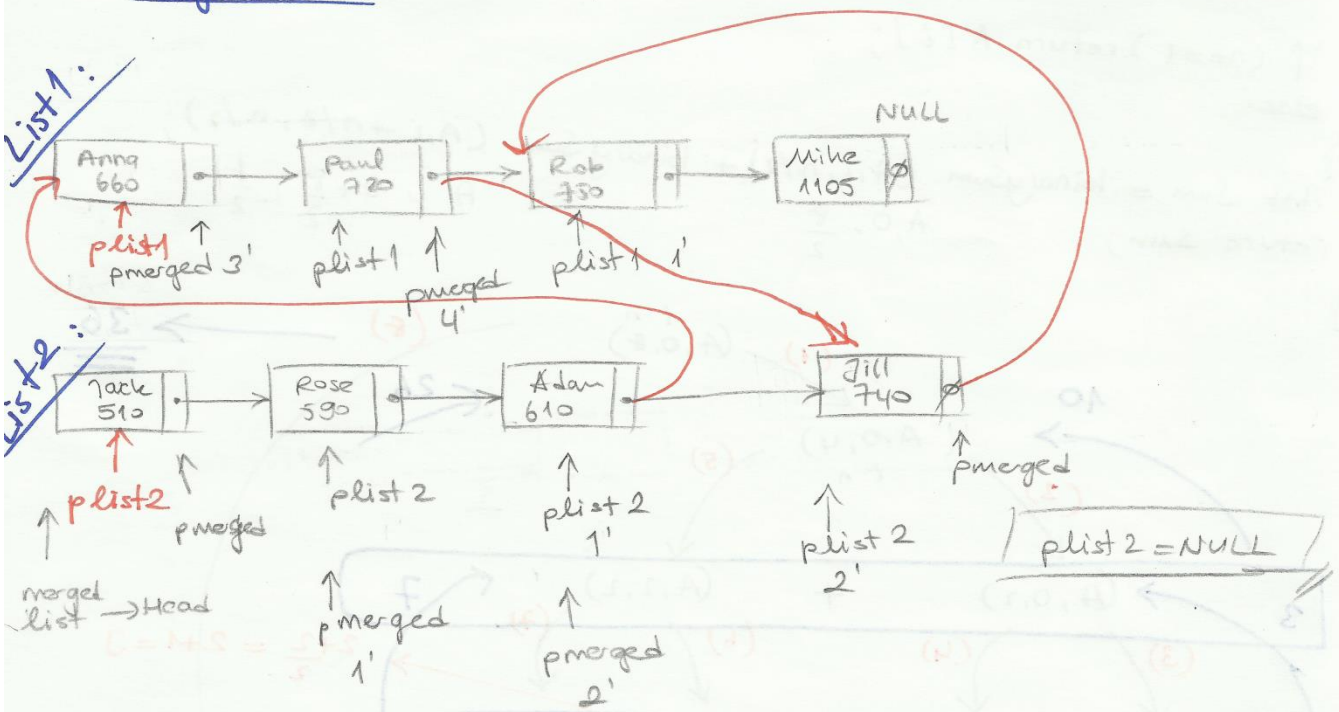
consol output

```
Sum = 3
sum = 7
Sum = 10
Sum = 11
Sum = 15
Sum = 26
Sum = 36
```

Multiple sum look.

Serbest Ödevler:

— **Merged List:** Birleştirilmiş bir liste tanımla.



→ List bağlantılarını güncelleyerek yerine elemanları kopyalayarak sıralı bir birleştirme. Node ekle.

→ Project → Properties → General → Platform Toolset 100

```
pmerged = new SinglyNode()
pmerged → score = 510;
pmerged → elem = Jack;
```

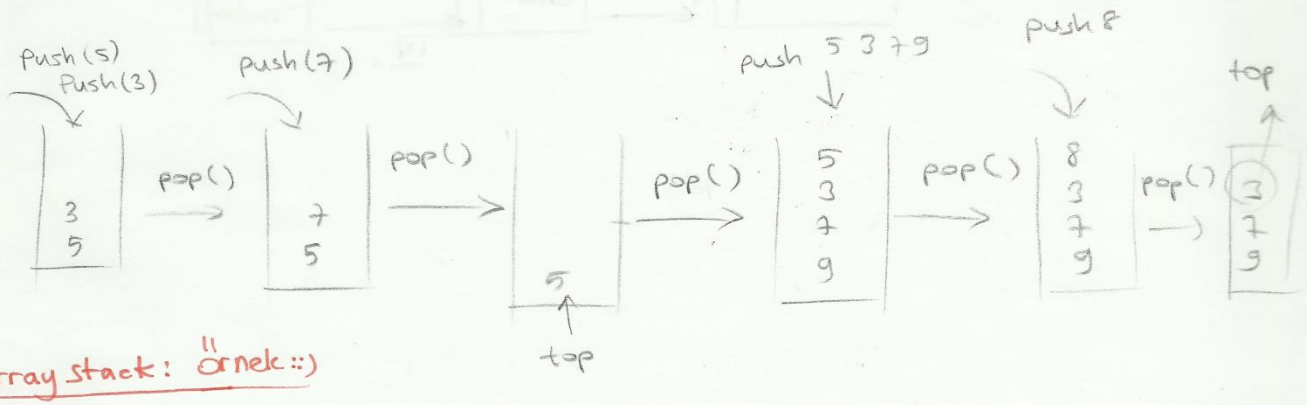
CHAPTER 5 :

STACKS : (YIĞIN)

LIFO (last in first out)

→ Microsoft word'de undo (ctrl+z) bastığımızda önceki yaptık larımız bir yığına atılmış. Yığının son elemanına dönülür.

- pop() yığının son elemanını sile döndürür
- top() " " "
- push() Ekle



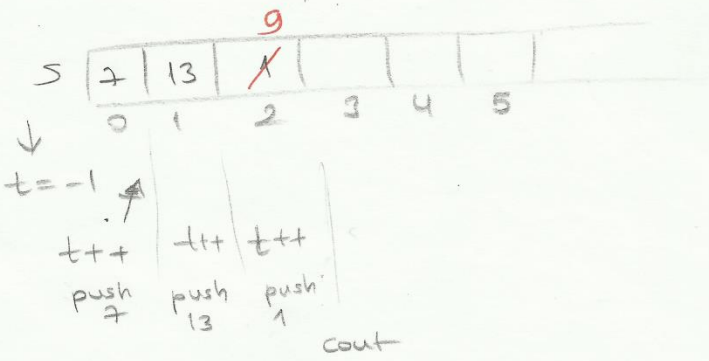
Array stack: örnek :)

```
int * S = new int [100];
```

t(-1) → top ptr top indisini gösteir.

```
A.push(7);
A.push(13);
A.push(1);
cout << A.top() << endl;
A.pop();
A.push(9);
cout << A.top() << endl;
A.pop();
cout << A.top() << endl;
```

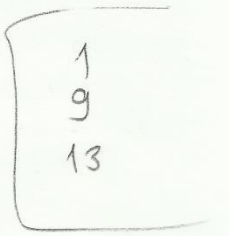
En son tepedeki yazılır.



22. satır yığın boş mu?
Değilse s[t]= 1

36. satır --t
1 silindi 9 yazdı
Ekrana 9

9 u sildirdik
Ekrana 13



Ekran çıktısı

ÖRNEK

LINKED STACK :

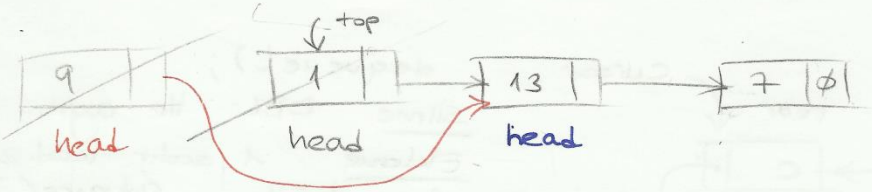
S → Singly Linked List

n=0 Listede eleman var mı?

S.front( ) Bağlı liste ilk elemanı. Head ptr işaret

push() → add front

pop() → removefront



1
13
7

Ekran çıktısı

1
9
13

ÖRNEK

Reversing a Vector Using a Stack :

V[ ] = { 1, 2, 3, 4, 5 }

(i= 0, 1, 2, 3, 4)

→ 5 2 3 4 5
5 2 3 4 1
5 4 3 2 1

5
4 top
3 top
2 top
1

```
for(i=0;i<5;i++)
{ V[i]=S.top();
  S.pop();
}
```

ÖRNEK

HTML :

< b> omer < /b>
< i > omer < /i >

vector < string >tags → stringlei taglarla vector içinde tutar.
↳Hem açma hem kapama var.

"/"dan farklı ise push yap.

kapama tagı ise pop yap.

Açma kapama tag eşitse True Eşit değilse false

matched | Not a match

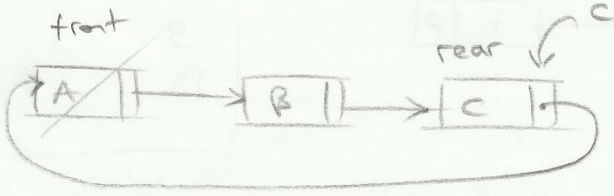
QUEUES (KUYRUKLAR)

FIFO (First in first out) ★ queue.c.print() → cout gibi.

front: kuyruğun başı.

rear: kuyruğun sonu.

Dairesel bağlı listede silme de, ekleme de cursor'un nextine yapılır.



dequeue();
Silme DBL ile aynı.
Ekleme 1 satır kod ekle.
Aynı değil. advance();
kuyruğun başındaki elemanı siler

Cursor dairesel bağlının sonuna işaret ediyor.

Cursor → list ilk elemanı,
Next(kuyruk " ")

2012-2013 BÜT'ünlemede sorulmuş.

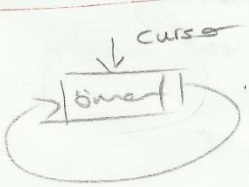
(DBL)
✓ Eklediğimiz eleman cursor → nextine eklenir.
✓ Kuyruktaki yeri elemana cursor işaret etmeli. Cursor'ı bir ilerlet

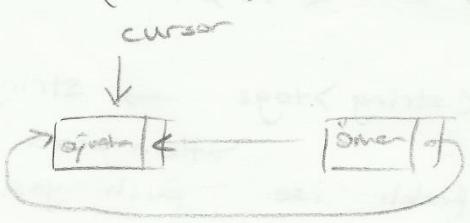
advance(): cursor = cursor →next

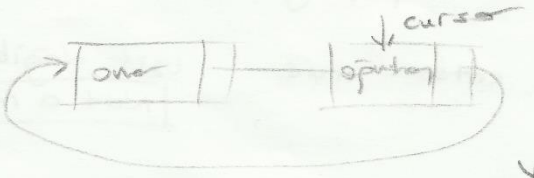
Circularly Linked Queue:

✓ dequeue: kuyruk başından veri siler.
C.remove: silme (DBL)
enqueue kuyruk sonuna veri eklenir.
✓ Ekleme C.add(e);
C.advance(); // fazladan (Advance)

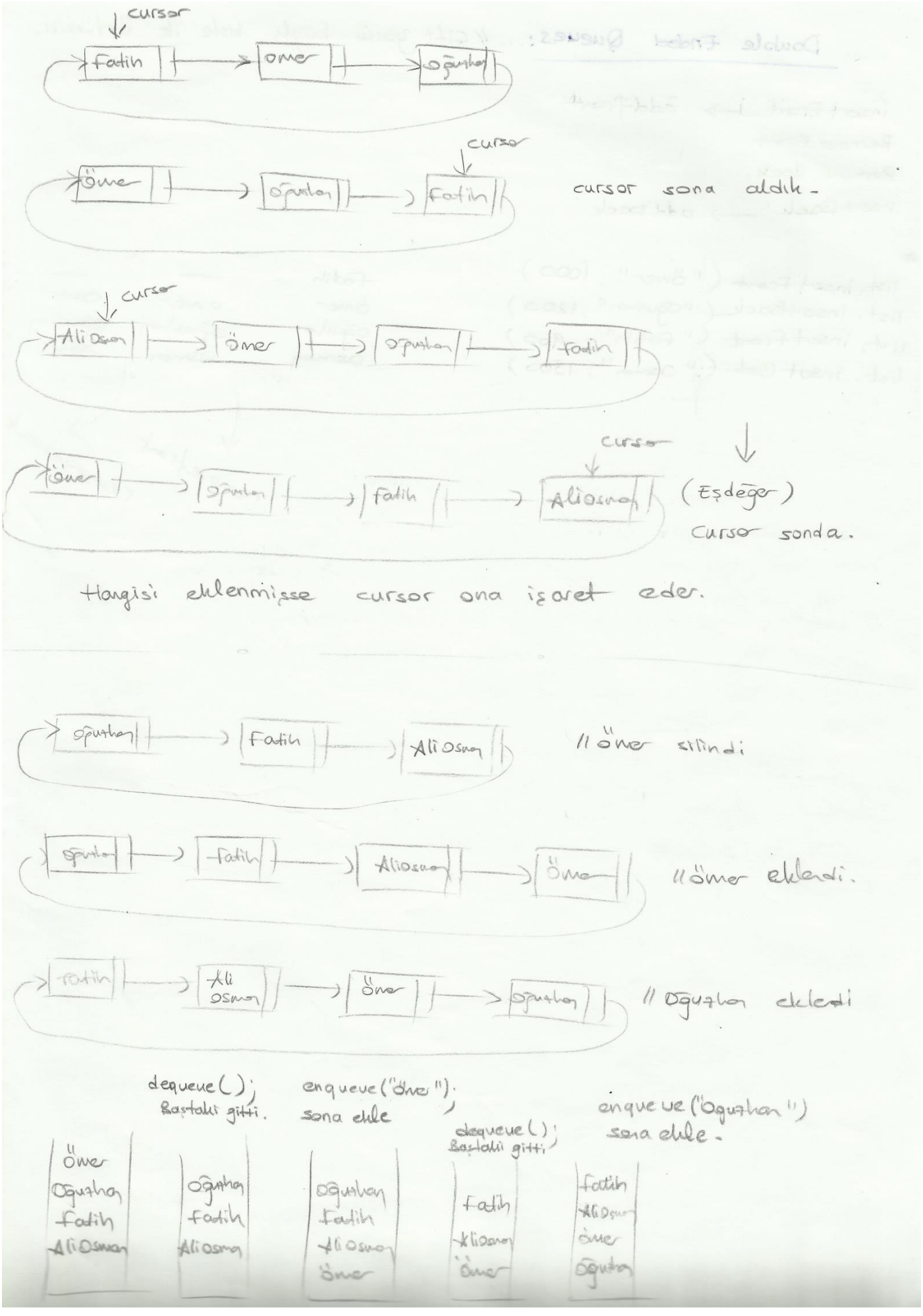
Test circ. Link-9







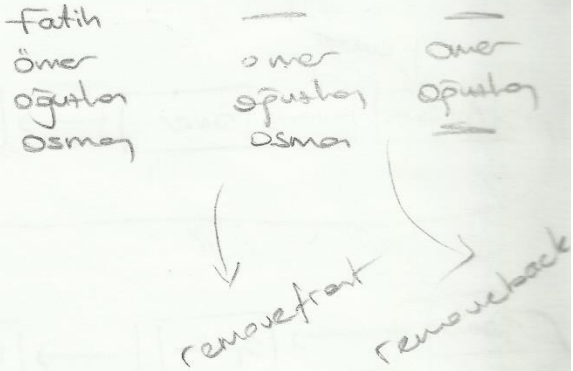
cursor sona aldık.
Yukarıdaki aynı.


cursor
Fatih
Ömer
Oğuzhan
cursor
Ömer
Oğuzhan
Fatih
cursor sona aldık.
cursor
Ali Osman
Ömer
Oğuzhan
Fatih
cursor
Ömer
Oğuzhan
Fatih
Ali Osman
(Eşdeğer)
Cursor sonda.
Hangisi eklenmişse cursor ona işaret eder.
Oğuzhan
Fatih
Ali Osman
// Ömer silindi
Oğuzhan
Fatih
Ali Osman
Ömer
// Ömer eklendi.
Fatih
Ali Osman
Ömer
Oğuzhan
// Oğuzhan eklendi
dequeue();
Baştaki gitti.
enqueue("Ömer");
sona ekle
dequeue();
Baştaki gitti
enqueue("Oğuzhan")
sona ekle.
Ömer
Oğuzhan
Fatih
Ali Osman
Oğuzhan
Fatih
Ali Osman
Oğuzhan
Fatih
Ali Osman
Ömer
Fatih
Ali Osman
Ömer
Fatih
Ali Osman
Ömer
Oğuzhan

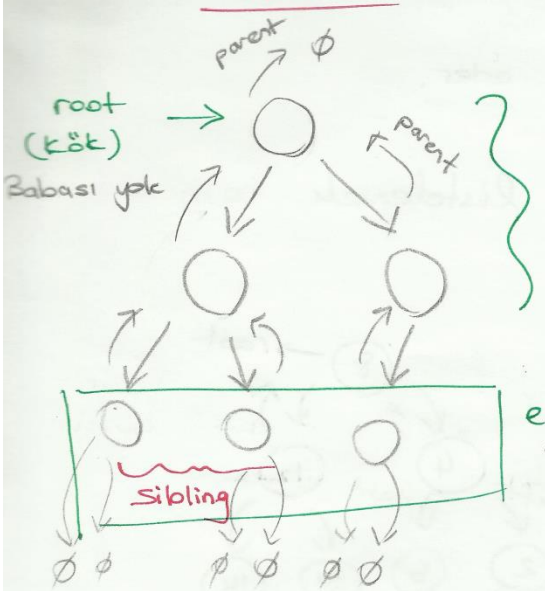
Double Ended Queues: // Çift yönlü bağlı liste ile örtüşür.

insertFront → addfront
RemoveFront
RemoveBack
insertBack → addback

list.insertFront("ömer", 1000)
list.insertBack("oğuzhan", 1200)
list.insertFront("fatih", 900)
list.insertBack("osman", 1500)



TREES : (AĞAÇLAR)

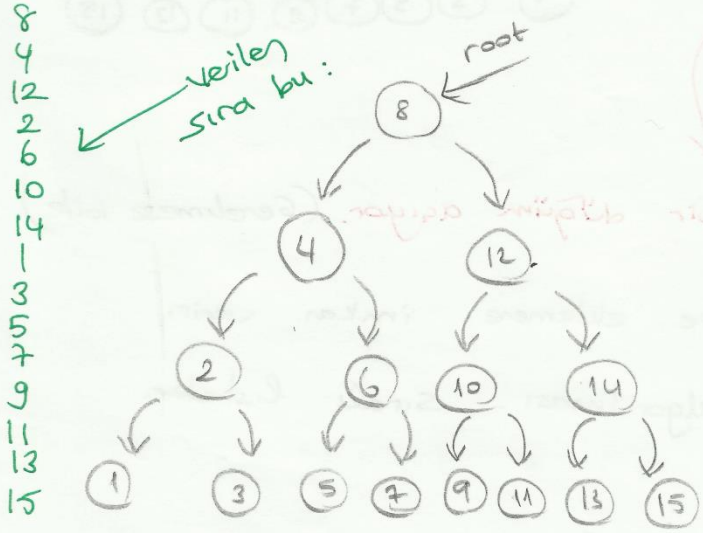


(Düğüm)

Level	Nodes
n	2^n

$2^n - 1$

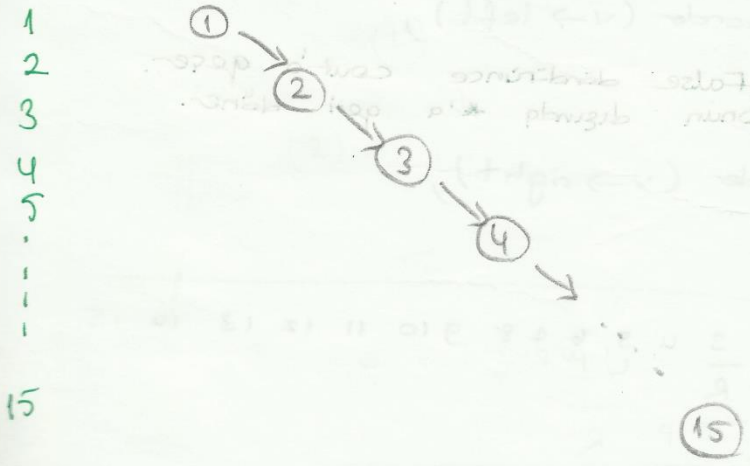
Max nodes



8 den { Küçük sola / Büyük sağa

8 − 4 } Büyük
8 − 12 } Küçük

8 − 4 − 2 : 8 − 12 − 10 } >
8 − 4 − 6 : 8 − 12 − 14 } <



Dengeli bir ağaç.

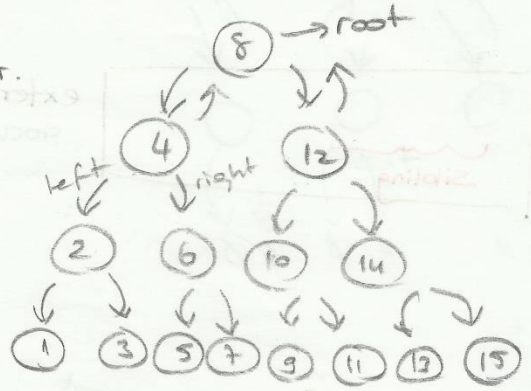
** Splay Tree : Ağacı oluştururken bir yandan da dengeliyor.

LinkedBinaryTree(); // Ağaç boş.

Node * root; ilk düğüme root işaret eder.

preorder
inorder
postorder
} Ağaç elemanlarını gezinip listelemek için. (Recursive fonk.)

addroot; Yeni bir düğüm oluşturur.




```
nuTree.addroot();
nuTree.root->elt = 8;
nuTree.expandExternal(nuTree.root)
nuTree.root->left->elt = 4;
nuTree.root->right->elt = 12;
```

Biti kısıtlıyor.
Hem sağa hem sola yeni bir düğüm açıyor. (Gerekmese bile)

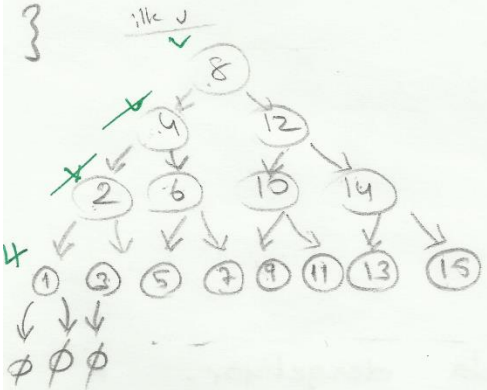
addBelowRoot → istediğin yere eklemene imkan verir.
(Binary Search Tree içinde)

inorder Traversal : Bir gezinme algoritması. Sıralı listeler.

L P R

```
void LinkedBinaryTree::inorder(Node *v) const
{
  if (v->left != NULL) inorder(v->left);
  cout << v->elt << " ";
  if (v->right != NULL) inorder(v->right);
}
```

→ False döndürünce cout'a geçer. Onun dışında *'a geri döner.

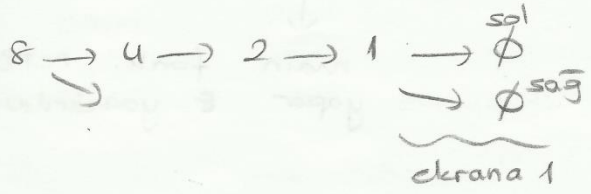


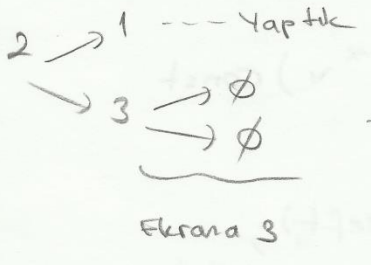
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
L P R
L P R
L P R

Consol

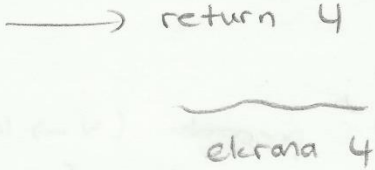
Sol çocuk baba sağ çocuk

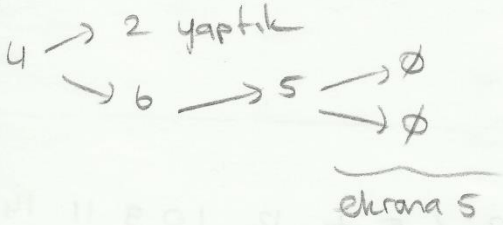
NOT: Sol çocuk varsa ona git bu çocuk ∅ olana kadar git. Çocuksuz olanı yaz. 5 yazdık. Return yapıldıktan sonra bu kez sağ çocuklara bak 7 return 6, 6 4 right olarak çağrıldığı için 4'e gider. 4'ü nereden çağırdık. V=8 iken left olarak çağırdık 4 return yapar 8 çağrılmamış 8 yazdırır. 8'in right çocuklarına bakarız.



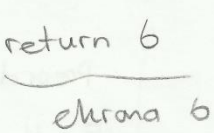


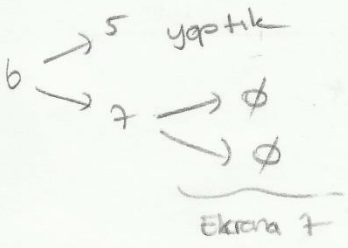
2 Bitti 2'yi kim çağırdı



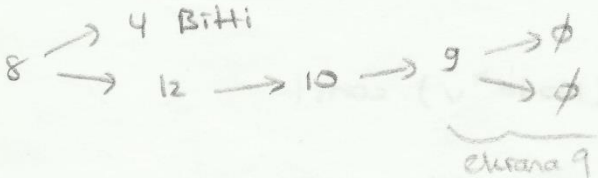


5 nereden 6 sol çocuk -

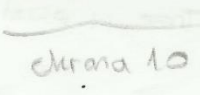


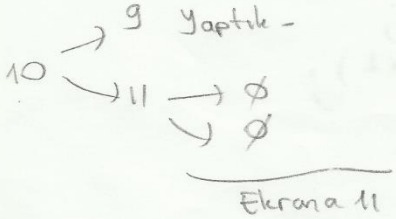


7 nereden 6 bitti 4 de bitti
4 nereden return 8

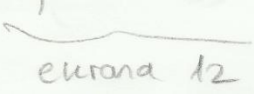


9 nereden 10 sol çocuk 90. satır





11 i kim çağırdı 10 Bitti.
10u kim çağırdı 12



12 → 10 Bitti
12 → 14 → 13 → ∅
13 → ∅
Ekrana 13

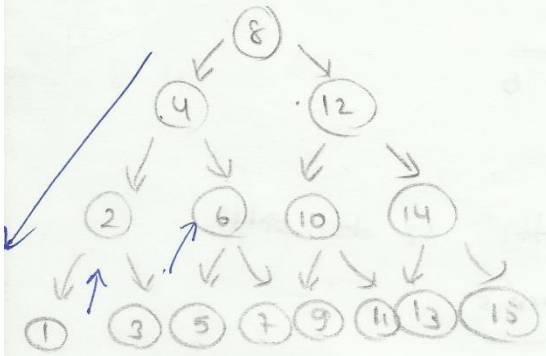
13'ü kim çağırdı return 14
Ekrana 14

14 → 13 Bitti
14 → 15 → ∅
15 → ∅
Ekrana 15

15 — 14 — 12 — 8
↓
main fonk. return yapar & yazdırılmıştı.

Preorder: (PLR)

```
void LinkedBinaryTree::preorder(Node* v) const
{
  cout << v->elt << " ";
  if (v->left != NULL) preorder(v->left);
  if (v->right != NULL) preorder(v->right);
}
```

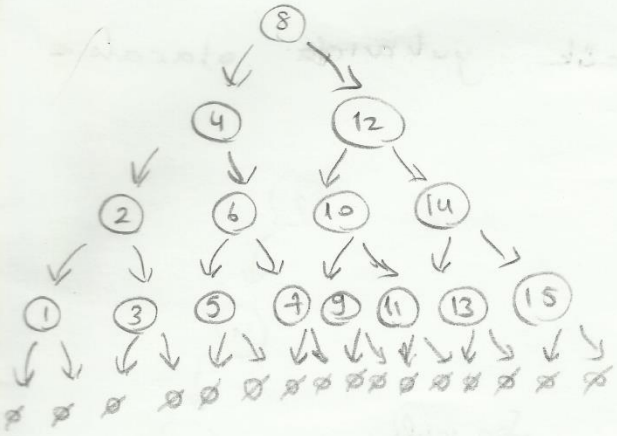


Preorde
8 4 2 1 3 6 5 7 12 10 9 11 14 13 15
P L R

Parent Left Right

Post Order: (LRP)

```
void LinkedBinaryTree::postorder(Node* v) const
{
  if (v->left != NULL) postorder(v->left);
  if (v->right != NULL) postorder(v->right);
  cout << v->elt << " ";
}
```



Post Order

13 3 2 5 7 6 4 9 11 10 13 15 14 12 8

L R P L R P L R P L R P L R P L R P L R P L R P

CHAPTER 8

— PRIORITY QUEUE (öncelikli kuyruk)

int değeri küçükse en önce, sonra büyüklüğüne göre ensona eklenir.

removeMin(); → En küçüğü önce siler.

insert(e); → Yeni e elemanını P kuyruğuna ekle.

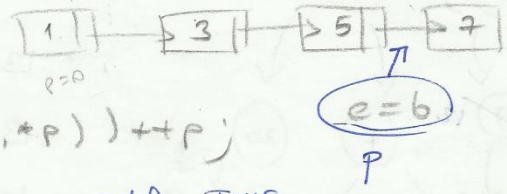
silme aynı ekleme farklı. (Queue ile)

Silme:

L.pop_front(); Yığının ilk elemanını siler. Removefront ile aynı

Ekleme:

```
void ListPriorityQueue::insert(const int &e)
{ std::list<int>::Iterator p;
  p = L.begin();
  while (p != L.end() && !isless(e, *p)) ++p;
  L.insert(p,e);
}
```



değil ↓ e, f karşılaştırır e<f True

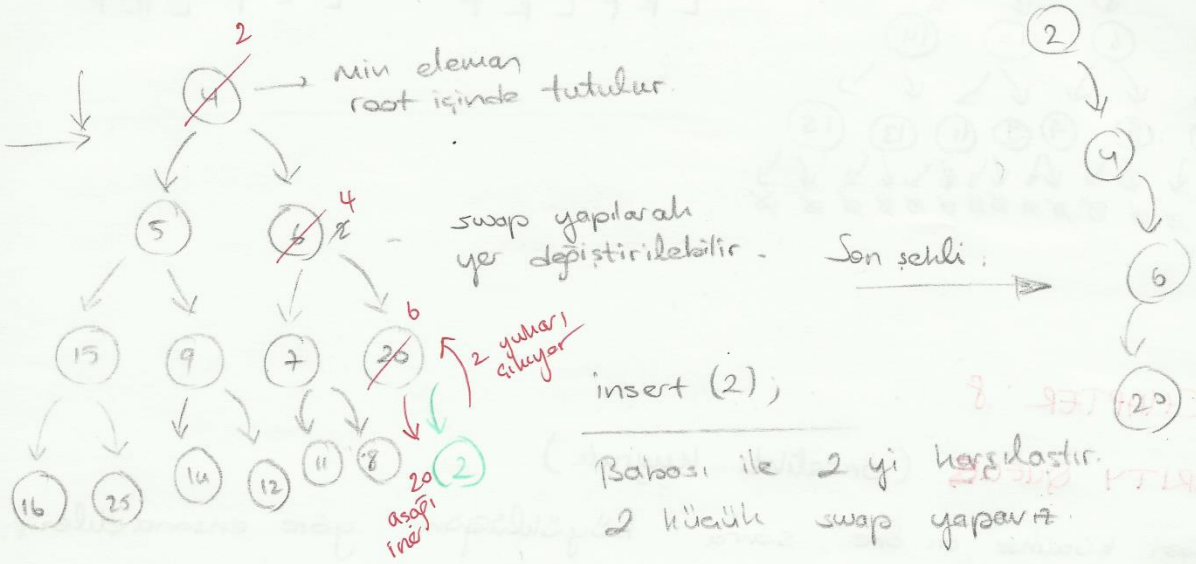
↓ e, p'nin önüne ekle. isless(3,4) → true
isless(4,3) → false

/* insert order gibi */ → Eklemek istediğimiz düğümden büyük old. sürece isless(6, *p) → false döndürdüğü sürece ++p yapılır.

isless(6, 1) → (false) = true
(6, 3) → " "
(6, 5) → " "
(6, 7) → (true) = false

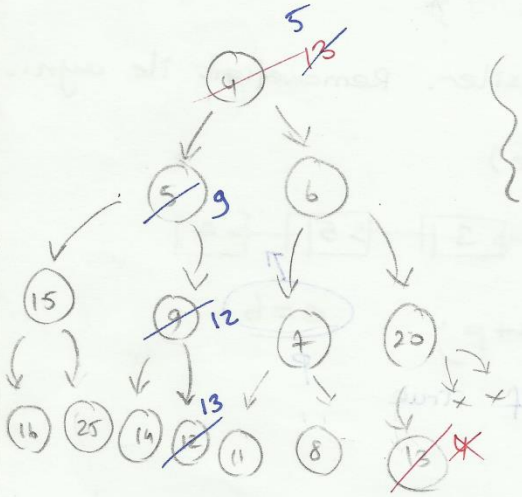
HEAPS:

★ En küçük yukarıda olacak -

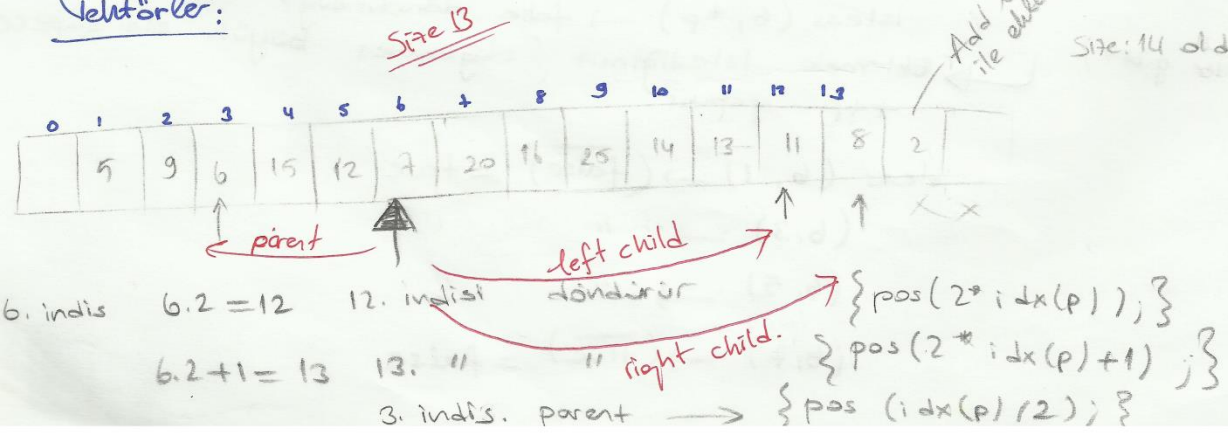


removeMin() rootu siler (e.k. eleman)

Klasik kuyrukta priority öncelik ya da ilk elemanı siliyorduk.



Vektörler:

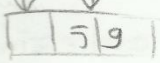


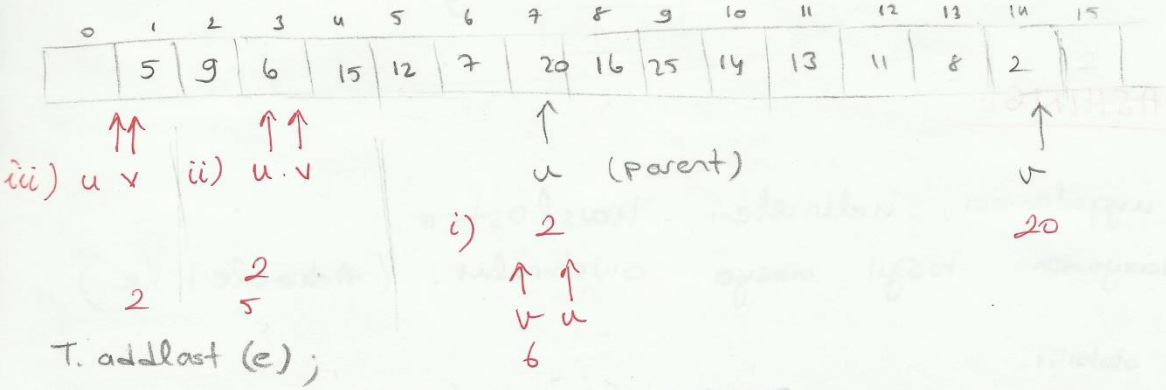
V. push_back() ⟶ sona ekler. Gerekirse swap yapar.

V. pop_back() ⟶ sondakini sil

V. size() - 1 ; // ilk kutu boş old. için.

v(1) push_back.





T.addlast(e);

Position v = T.last();

* Sola doğru kayar

while (!T.isRoot(v))

{

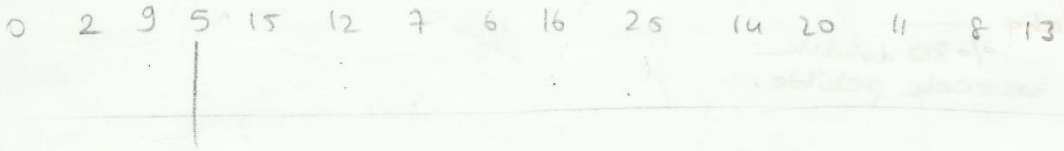
position u = T.parent(v);

if (!isless(*v, *u)) break;

T.swap(v,u);

v=u;

}



if (size()==1)

T.removeLast();

else

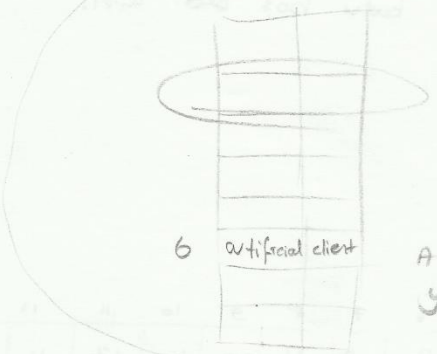
{

position u=T.root();

T.swap(u, T.last());

T.removeLast();

```
Position v = T.left(u);
if (T.hasRight(u) && isless(*(T.right(u)), *v))
  v = T.right(u);
if (isless(*v, *u),
  T.swap(u,v);
  u=v;
}
else break;
}
}
```



Boş ise fotladan bellek

Bucket addressing.

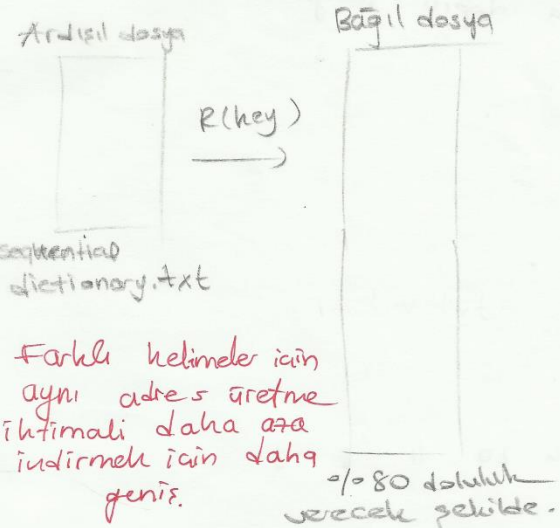
Aynı adresi itade edenle yanyana olur.

HASHING:

~~Sıralı~~ uygulaması, kelimeleri karşılaştırır.

Ardışıl dosyadan bağıl dosya oluşturulur. (Adresleri ile)

Alfabetik sıraya göre olabilir.



Farklı kelimeler için aynı adres üretme ihtimali daha az indirmek için daha geniş.

ÖRNEK: array kelimesi için adres üretirken a → ASCII karşılığını 10 ile çarpıp r → ASCII karşılığı ile topluyor. Tüm harfler için sırayla yapıp topluyorum. Hesaplanan bu bağıl adresin %/ modunu alırız. Mod işleminde genelde asal sayılar kullanılır.

array kelimesi için

sum=0 verildi. a=89

r=70

0 + 89.10 + key[r] = 960 sum

(r=70)

960 + 70.10 + key[a] = 1749

(a=89)

```
for (int j=0; j<4; j+=2)
  sum = (sum + 10^4 key[j]+k[j+1]);
  sum = sum %/ 11;
return sum;
}
```

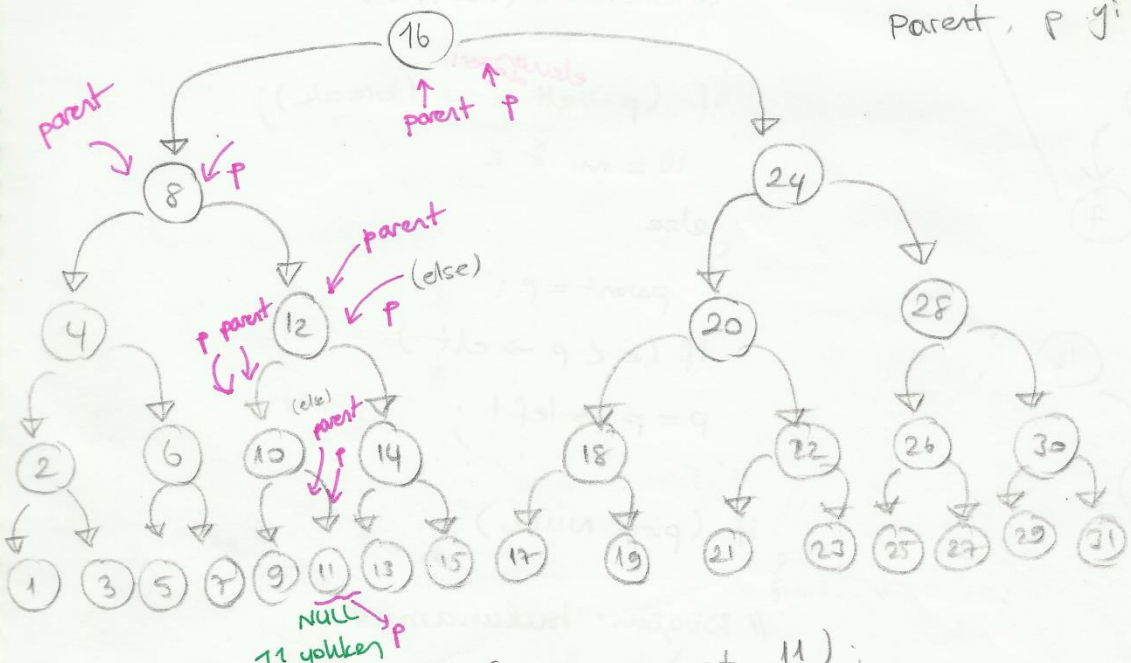
relative .txt teki indis sayısı kadar

1749 | 11

2 satır.

BINARY SEARCH TREE :

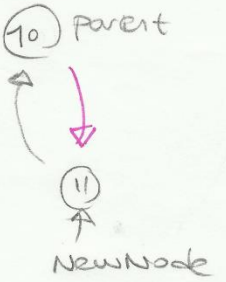
1) Add Node
2) Delete "
3) Search "



nuTree.addBelowRoot(nuTree.root, 11);

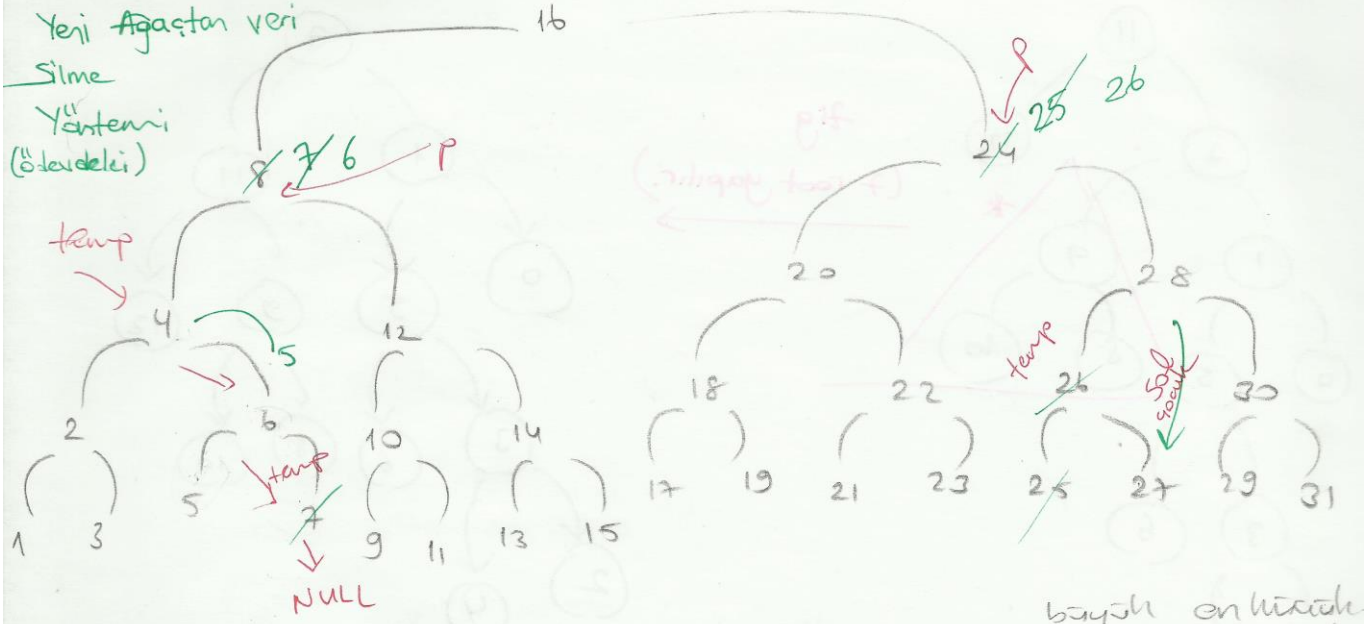
```
LinkedBinaryTree::addBelowRoot(Node* p, int e)
while (p != NULL)
{
  parent = p
  if (e < p->elt)
    p = p->left;
  else
    p = p->right;
}
```

p → NULL old while dan alınır.




```
if (newNode->elt < parent->elt)
  parent->left = newNode;
else
  parent->right = newNode;
```

Yeni Ağaçtan veri Silme Yöntemi (öğrendiği)



7 yi silmek istediğimizde yerine kendinden büyük en küçük (6) gelecek ama çocuğu var 5, 4'ün çocuğu olur.

küçük en büyük

8 yerine 9 getirmek istesek çocuğu yok direkt olur.

9 yerine 10 getirmek " çocuğu var 11, 12'nin çocuğu olur.

Linked Binary Tree

parent pointer silindi. (parent'a erişmek istesek p->par.)

```
if (p->left != NULL)
{
  temp = p->left;
  while (temp->right != NULL) temp = temp->right;
  p->elt = temp->elt;
    if (temp->left != NULL)
  {
    temp->left->par = temp->par;
   if (temp == temp->par->right)
      temp->par->right = temp->left;
   else
   temp->par->left = temp->left;
  }
else { if (temp == temp->par->right) temp->par->right = NULL;
       else temp->par->left = NULL;
```

Hashing ağaca ekleme;

Sol sağ çocuk sözlük sırasına göre bulunur.

strcmp (str1, str2) // string compare farkı için.

str1 == str2 returns 0

str1 < str2 returns -1 (str1 sözlükte, önce) sola

str1 > str2 returns +1 (str1 sözlükte sonra) sağa.

(::strcpy (nuTree → root → keline.turkce, keline.turkce)

↓

kelimenin Türkçesini stringe kopyalıyor.

searchTree: Aradığımız kelimeyi yollarız. Ağaç içinde gezinerek kelimeyi bulur.

generateTree: dosyadan ağaca ekliyor.

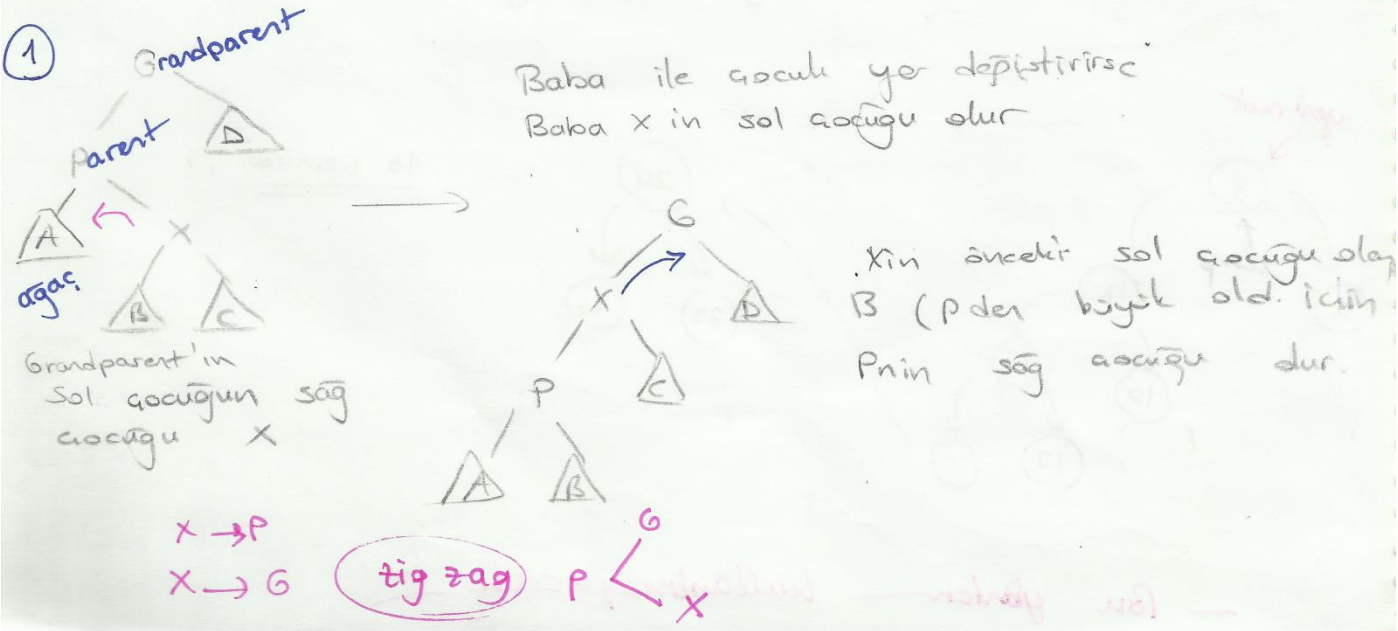
↳ Programdan çıkılırken güncel hali dosyaya yazılır. Böylece her açılışımızda ağacı yeniden oluşturmamıza gerek kalmaz.

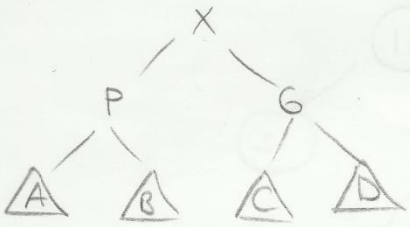
SPLAY TREES: Aşağıdan yukarıya doğru.

X ağaca eklediğimiz veri varsayalım.

X root olana kadar rotation (işlemler) devam ede.

(1)





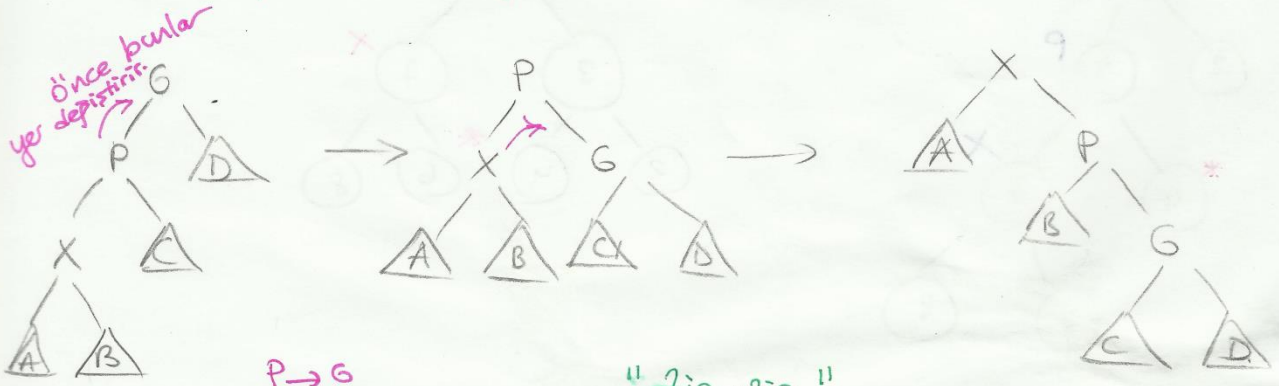
C G'nin sol çocuğu oldu.

"zig-zag" Case

önce sola sonra sağa gidilir.

önce sağa sonra sola olursa mirror'i alınır.

② Sol çocuğun sol çocuğu, sağ çocuğun sağ çocuğu



P → G

X → P

"zig-zig"

G

P

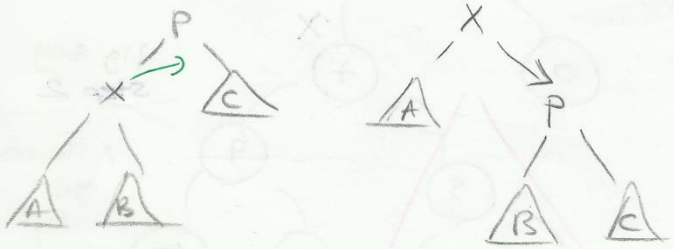
X

Repeat ① & ②

Ne zamana kadar → doğrudan roota da gidebilirsin

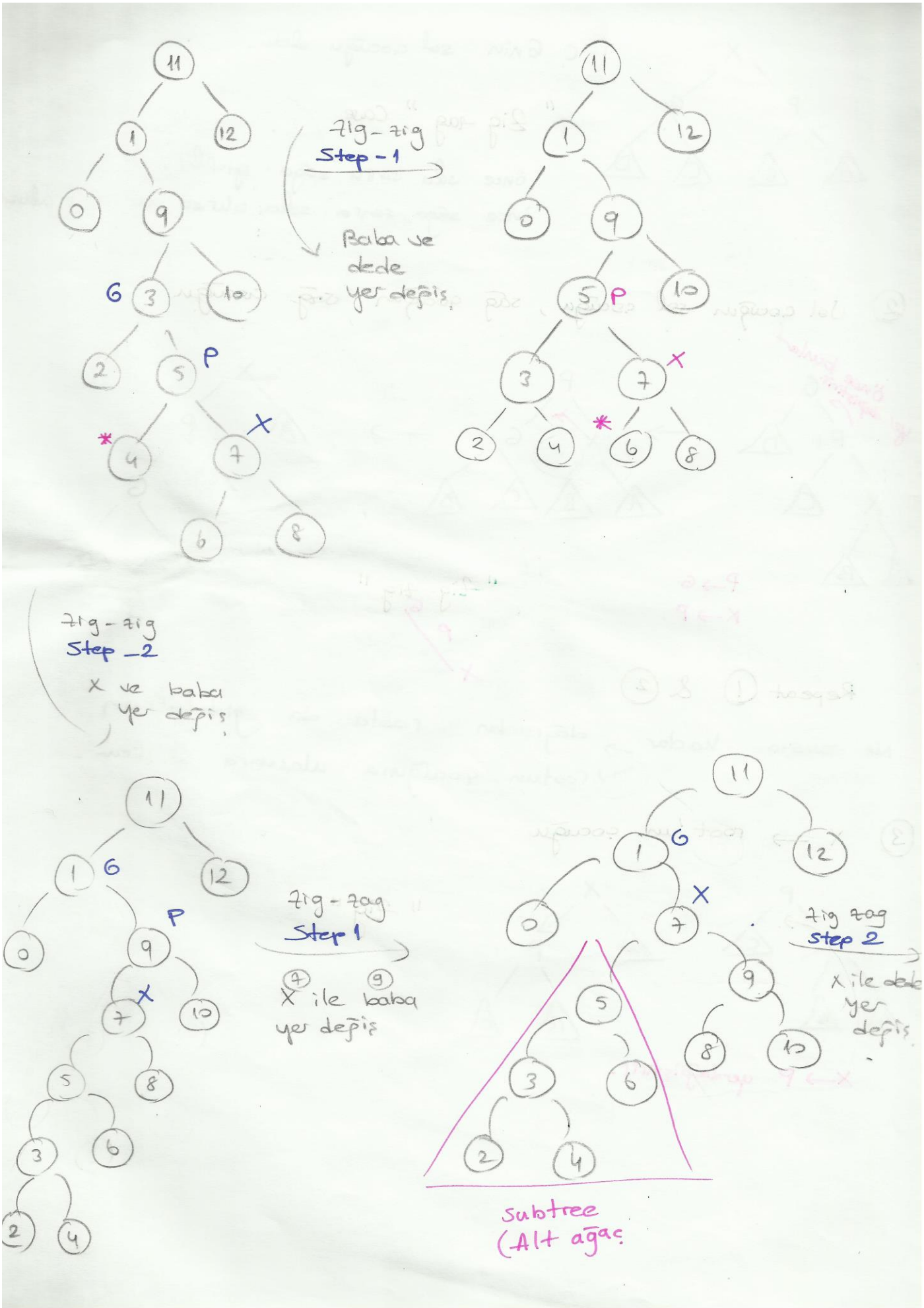
→ root'un çocuğuna ulaşınca 3. işlem.

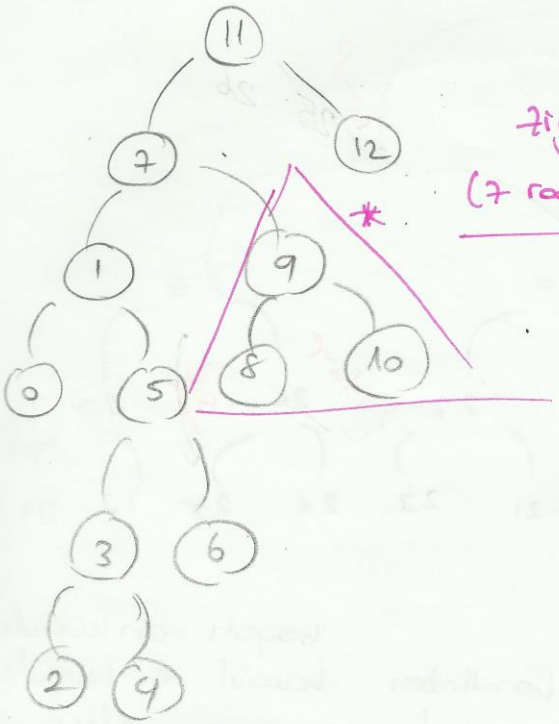
③ X → root'un çocuğu



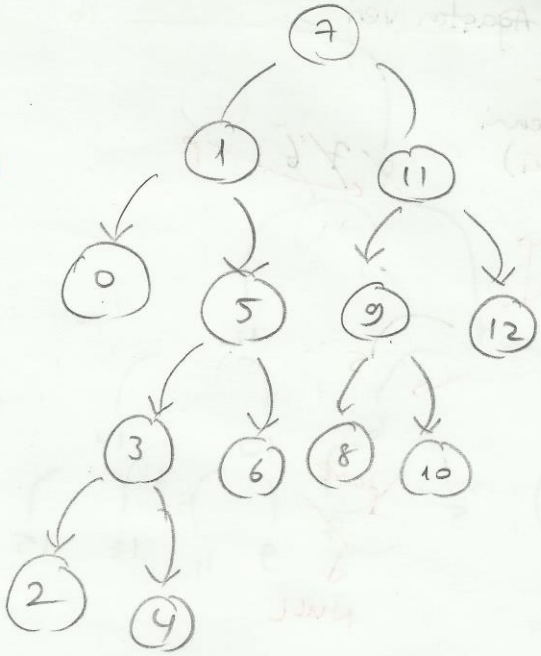
"zig"

X → P yerdeğiştirir.


zig-zig
Step-1
Baba ve dede yer değiş.
G
P
X
*
zig-zig
Step-2
X ve baba yer değiş.
zig-zag
Step 1
X ile baba yer değiş
zig zag
step 2
X ile dede yer değiş
subtree
(Alt ağaç



zig
(7 root yapılır.)



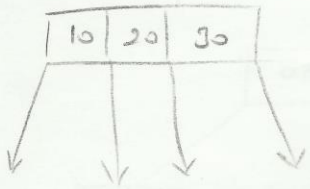
7 rootun sol çocuğu oldu

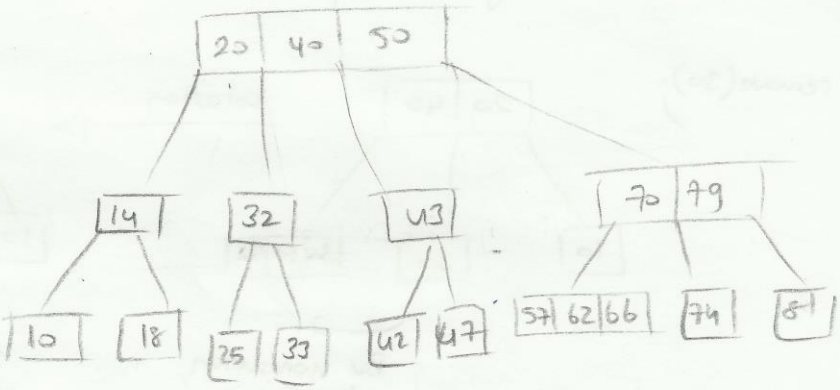
NewNode → x oluyor-

2011 – 2013 son haline 3 ekle.

2-3-4 TREES :

Her bir düğüm 2, 3 veya 4 çocuğa sahiptir. Her bir düğümde 1, 2 veya 3 eleman tutulur.

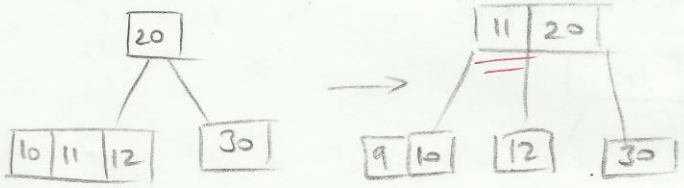




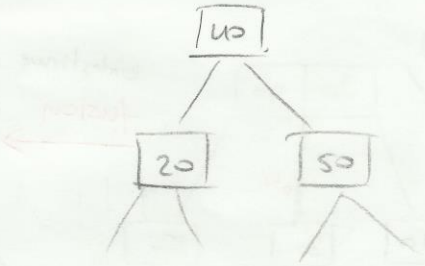
insert (k) = (Ekleme)

insert (9)

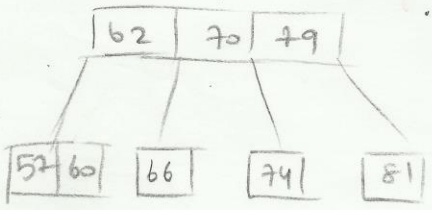
3 tane yanyana olanı parçalıyoruz.



Ortadaki roota çıkar.



insert (60)



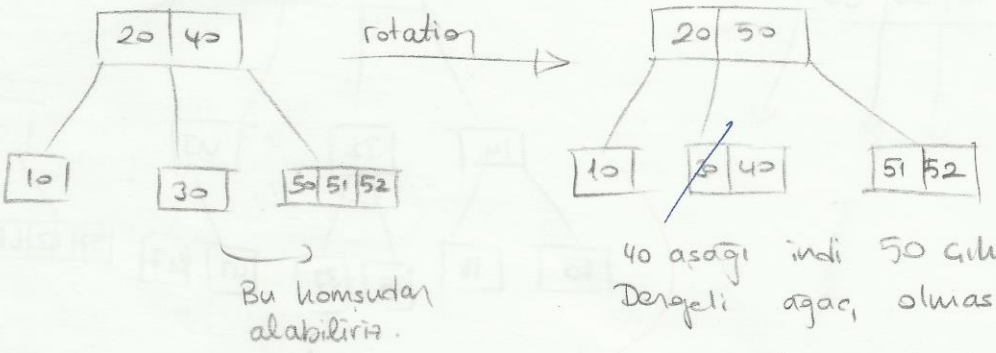
Burada yeni bir yer açmak için yukarı gönderiyoruz.

remove (k) : kendisinden büyük en.k. elemanı kendi yerine getiriyoruz.

Kurallar :

1) Eğer 1 elemanlı düğüme rastlanırsa, komşuların birinden 1 eleman almaya çalış

remove(30);



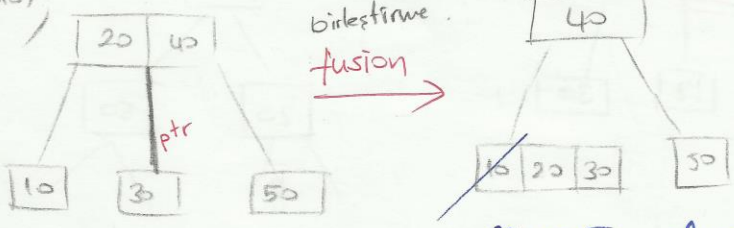
Bu komşudan alabiliriz.

40 aşağı indi 50 çıktı Dengeli ağaç olması için

remove (30) dersek direkt silinir.

2) (0 düğümde 1 eleman varsa)
Eğer komşularda 1'den fazla eleman yoksa babadan 1 eleman al.

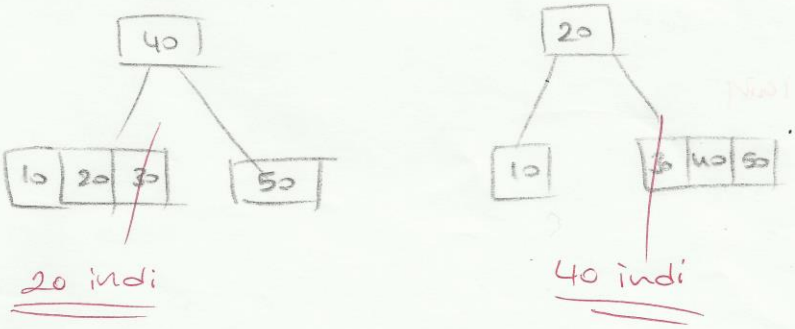
remove (10);



20 aşağı alınınca birleştirerek düğüm oluşturuyoruz.

ptr bir düğüme işaret etmesi gerekiyor.

remove (30); // Bunda 2 seçeneğe sahibiz.

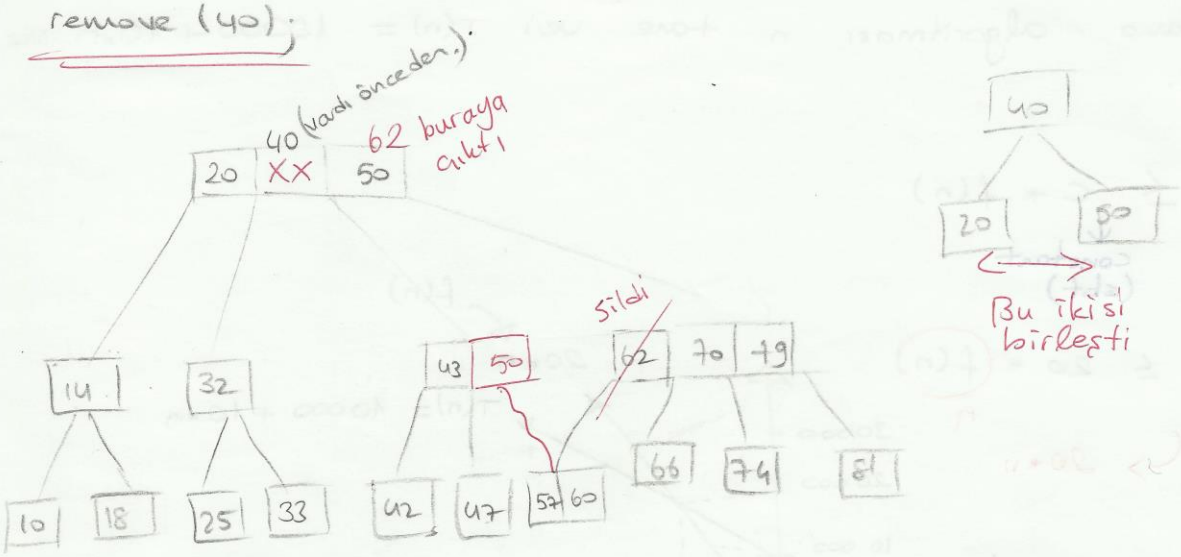


20 indi

40 indi

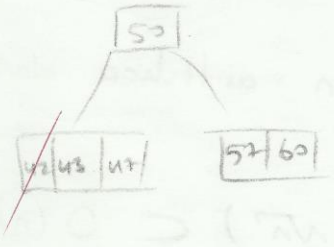
3) Eğer baba root ve 1 elemana sahipse ve komşuda 1 elemana sahipse root, silinecek düğüm ve komşu düğüm birleştirilir. Bu durumda ağaç seviyesi 1 azalır.

remove (40);



62, 50 nin yerine çıktı
50 aşağı alındı, 62 nin çocuklarının babası oldu.

→ 42 yi roota çekmek -
(Rule 2) Yan düğümde 1 eleman var. Babadan eleman al
(47) 47 roota çekilecek olsa 57, 60 bakabilirdik.



→ sonra 42 sil yukarı çıkar.

O (f(n)) NOTASYONU (Order Notasyon) – 19. Video –

- Sıralanacak verilerin diskten okunup diziye kopyalanması, 10.000 ms alsın.
- Sıralanmış her bir verinin yerinin bulunması 10 ms alsın.
- Sıralama algoritması n tane veri $T(n) = 10000 + 10*n$ ms hızar.

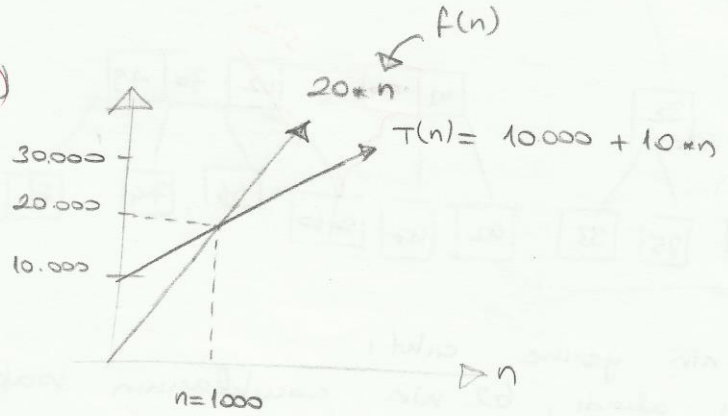
$T(n) \leq c * f(n)$

↓ constant (sbt)

$T(n) \leq 20 * f(n)$

$f(n) = n$

↳ $20 * n$



$n \to \infty$ iken $20*n > T(n)$ $f(n) = O(n)$ çıkıyor.

$T(n) = 10000 + 10*n$

Bu sabit direkt atılır.

↳ Algoritma karmaşıklığı verilen kaç kere işlendiği ile ilgilenir. Mimari ile ilgilenmez.

$T(n) = n^3 + n^2 + n$ $O(n^3)$

Baskın fonk. dikkate alınır. Hangisi n arttıkça daha çok artıyor ona bakılır.

$O(1) \subset O(\log n) \subset O(\log^2 n) \subset O(\sqrt{n}) \subset O(n \log n)$
$\subset O(n^2) \subset O(n^3) \subset O(n^4) \subset O(2^n) \subset O(e^n)$

$O(n) \subset O(n^2)$

n^+ veya daha yavaş gereksiz.

$n \log n$ veya daha hızlı en verimli

```
for i ← 0 to n-1 do
    a ← 0
    for j ← 0 to i do
    a ← a + x[j]
    A[i] ← a / (i+1)
return array A ;
```

$1+2+3+4 \dots +n = \frac{n(n+1)}{2}$

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

$= \frac{n^2+n}{2}$

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

$f(n) = n^2 \quad O(n^2)$

★ iç içe iki tane for döngüsü varsa cevap $\Theta(n^2)$

· Upper bound → En yavaş bu kadar koşar.

lower bound → (Ω) 20. video.