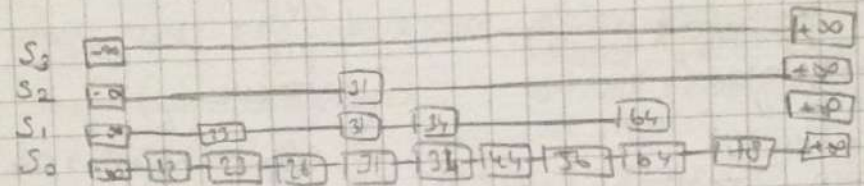
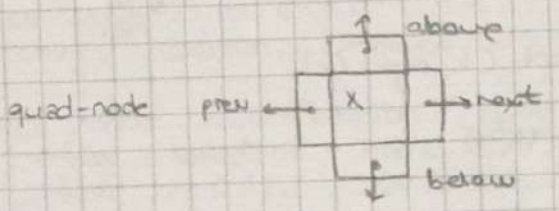
Skip List

A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \ldots, S_h$ such that
- Each list S_i contains the special keys $+\infty$ and $-\infty$.
- List S_0 contains the key of S in nondecreasing order
- Each list is a subsequence of the previous one, i.e.,
$S_0 \supseteq S_1 \ldots \supseteq S_h$

S_h contains only two special keys





Space usage → $O(n)$

Search, Insertion, Deletion → Running Time → $O(\log n)$

Multiset — Multimap

- Set is an unordered collection of elements, without duplicates,
- Multiset (also known as bag) is a set-like container that allows duplicates
- Multimap is similar to a traditional map, in that it associates values with keys; however, in a multimap the same key can be mapped to multiple values,

Set ADT

add(e) : (if not already present)
remove(e) : (if it is present)
contains(e) : boolean
iterator() : Returns an iterator of the element of S.
* addAll(T) : Update S according to set T, replaces S by $(S \cup T)$
* retainAll(T) : replace S by $(S \cap T)$
* removeAll(T) : replacing S by $(S - T)$.

Ch 3 : Arrays, Linked Lists, Double Lists

1 Arrays
An array is a sequenced collection of variables all of the

Abstract Class

```
abstract class Employee {
    ...
    public abstract double computePay();   // abstract method does not have a body
    public String getName() { return name; }   // method with body
```

Exceptions

try-catch → "handles" the problem

```
try {
        guardedBody
} catch (exType1 variable1) {
        remedyBody
} ...
```

throwing exceptions

```
throw new exceptionType(parameters);
```

the throws clause

```
public static int parseInt(String s) throws NumberFormatEx- {
    ... }
```

- Casting

```
CreditCard card = new PredatoryCreditCard();   // widening conversion
PredatoryCreditCard pc = (PredatoryCreditCard) card();   // narrowing conversion
```

GENERICS

The generic framework allows us to define a class in terms of a set of formal type parameters, which can then be used as the declared type for variables, parameters and return values within the class definition.

- Syntax -

```
// Generic add method
static <T> add (T a, T b) {
    T answer = a+b;
    return answer; }
```

} T could be int, float, or double.

Nested Classes

Java allows a class definition to be nested inside the definition of another class.

It could be static or non-static.

```
public class CreditCard {
    [private] [static] class Transaction { ... }
    // instance variable for CreditCard
    Transaction [] history; // keep log of all transaction
                            // for this card
}
```

DATA STRUCTURE

Ch 2 : Object Oriented

- Abstract Data Types (ADTs)

* Abstraction is to distill a system to its most fundamental parts. (ayıştırmak)

- Encapsulation (private)

* Different components of a software system should not reveal the internal details of their respective implementations. (gizlilik, ortaya çıkarmak)

* Encapsulation gives one programmer freedom to implement to details of a component.

- Modularity

* Modern software systems typically consist of several different components that must interact correctly in order for the entire system to work properly. Keeping these interactions straight requires that these different components be well organized.

* Modularity refers to an organizing pricple in which different components of a software system are divided by into separate functional unit.

- Design Patterns (tekrar kullanılabilir, hatasız)

* Object-Oriented design facilitates reusable, robust, adaptable software.

- UML : Unified Modelling Language

- Polymorphism

ex: `Animal[] animal = { new Dog("Fındık"), new Cat("Minnoş") };`
(Animal: Base class, Dog: subclass, Cat: subclass)

~ Interface ~

```
public interface Sellable {
    public String description (); }

public class Photo implements Sellable {
    private String descript;

    public String description () { return descript; }
```

~ Multiple Inheritance

In Java, multiple inheritance is allowed for interfaces but not for classes. The reason for this rule is that interface do not define fields or method bodies, yet classes typically do.

Ex: `class example implements X, Y { ... }`

$1 < \log n < n < n\log n < n^2 < n^3 < 2^n$

Pseudocode Details

Type of Operation	Symbol	Ex.
Assignment	$\leftarrow$ or $:=$	$c \leftarrow 2\pi r$, $c := 2\pi r$
Comparison	$=, \neq, <, >, \leq, \geq$	
Arithmetic	$+, -, \times, /$ mod	
Floor / Ceiling	$\lfloor \rfloor, \lceil \rceil$	$a \leftarrow \lfloor b \rfloor + \lceil c \rceil$
Logical	and, or	
Sums, Product	$\sum, \prod$	$h \leftarrow \sum_{a \in a} 1/a$

- method call
  method (arg [, arg ...])
- return value
  return expression
- Control flow
  if ... then ... [else ...]
  while ... do ...
  repeat ... until ...
  for ... do ...

Ex: initialize p_0 agents, each with energy $E = \frac{\theta}{2}$
loop:
  foreach alive agent a:
    pick link from current document
    fetch new document D
    $E_a \leftarrow E_a - c(D) + e(D)$
    θ - learn with reinforcement signal $-e(D)$
    if ($E_a > 0$)
      a' ← mutate (recombine (clone (a)))
      $E_a, E_{a'} \leftarrow E_a/2$
    elif ($E_a \leq 0$)
      die (a)
  process optional relevance feedback from ...

Big-Oh and Growth Rate

- The big-oh notation gives an upper bound on the growth rate of a function.
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$

$f(n)$ is $O(g(n))$
↓
① no, grows more
② yes, grows more
③ yes, same growth

$f(n)$ is asymptotically less than or equal to $g(n)$

- The SLL occupies less memory than DLL.
- Complexity of insertion and deletion at a given position is O(n).

- Complexity of deletion with a given node is O(n). Because previous nodes needs to be known and travels take O(n). - Mostly used for the execution of stack - Do not need to perform any searching operation, just want to save memory PREFER (SLL)!	- Complexity of deletion with a given node is O(1). Because the previous node can be accessed easily. - Mostly used to execute heaps, and stack, binary trees. - In case of better implementation, while searching, PREFER (DLL)!

Advantages of Linked List

* The size of the arrays is fixed. There is no limit in Linked List.
* Insertion a new element in an array of elements is expencive because a room has to be created for the new element and a create a new room existing, elements have to be shifted.

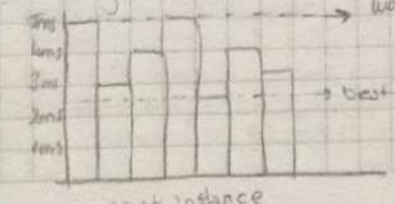
Disadvantages of Linked List

* Random access is not allowed. So we cannot do binary search.
* Extra memory space for a pointer.
* Arrays have a cache locality (ön bellek)
* It takes a lot of time in traversing and changing the pointers.
* It will be confusing when we work with pointers.

Ch 4 : ALGORITHM ANALYSIS

* The primary analysis tool involves characterizing the running times of algorithms and data structure operations, with space usage also being of interest.
* Computer solutions should run as fast as possible.

Measuring operations as a function of input size



Typically, running times are characterized in terms of the WORST CASE !!!

Queues

Recursion

```
public static int factorial ( int n) throws IllegalArgumentException{
    if (n<0) // argument must be non-negative
      throw new IllegalArgumentException ();
    if (n==0)  // base case
      return 1;
    return n* factorial (n-1); // recursive call
```

Binary Search (sorted list)

```
public static boolean binSearch (int[] data, int target, int low, int high) {
  if (low > high) {
     return false; }
  else {
     int mid = (low+high) / 2;
     if (target == mid) {
        return true;
      }
     else if (target < data [mid]) {
        return binSearch (data, target, low, mid-1);
     else {
        return binSearch (data, target, mid+1, high);
     }
  }
```

Reverse Array

```
public static void reverseArray (int [] data, int low, int high)
    if (low < high) {
        int temp = data [low];
        data [low] = data [high];
        data [high] = data [temp];
        reverseArray (data, low+1, high-1);
    }
}
```

run time

~ STACK ~

First-in - last-out (FILO)

O(1) push (Obj.) : insert an element

O(1) pop (): removes and return the last element } if empty

O(1) top (): just returns last element } return null

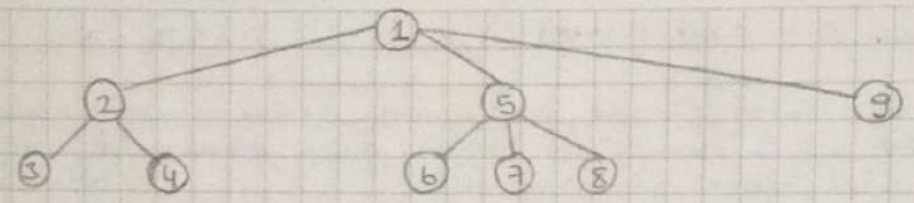
O(1) size(). returns the number of elements stored.

O(1) isEmpty().

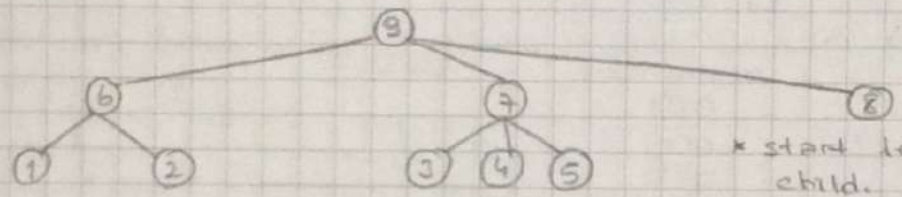
O(n) → spaced usage.

Inorder Traversal

Preorder Traversal



Postorder Traversal



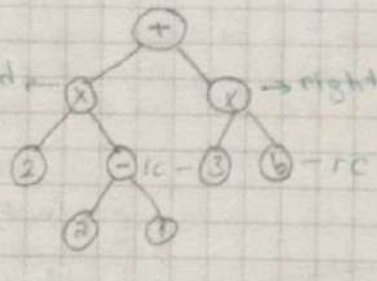
* start left-most child.

Binary Trees

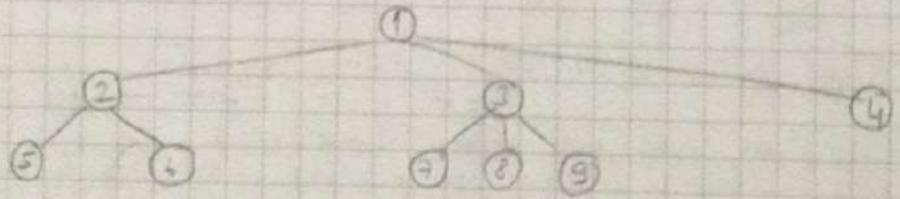
- Each internal node has at most two children
- The children of a node are an ordered pair

Application : Arithmetic expression,
Decion process (yes, no), Searching

Ex $(2 \times (a-1) + (3 \times b))$

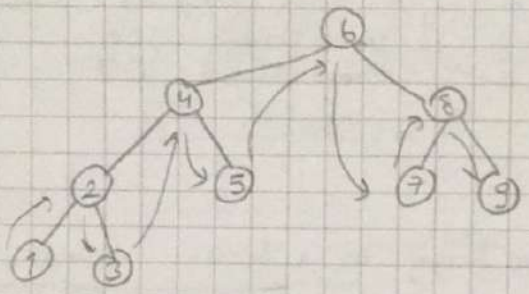


Breadth-First Tree Traversal

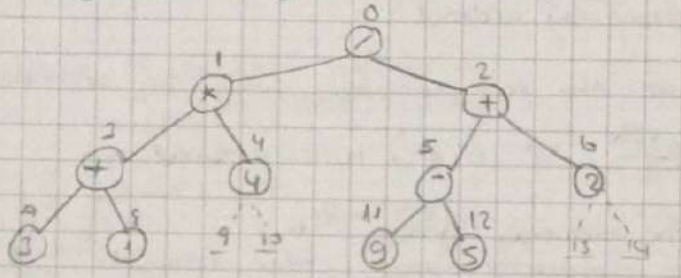


Inorder Traversal

A node is visited after its left subtree and before its right subtree



Ex: Binary Tree Array



/	*	+	+	4	-	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Priority Queue

* Sometimes First-in-first-out (FIFO) is not enough
  ex = hospital
  air-traffic control center
* P.Q. can be used to handle priority situations.
* A P.Q. stores collection of entries.

insert(k, v) : inserts an entry with key k and value V
removeMin : removes and returns the entry with smallest key (or null)
min() : returns but not remove smallest key (or null)
size()
isEmpty()

~ unsorted ~

! min() O(n)
insert(k,v) (kolay) O(1)
! removeMin() O(n)
size() } O(1)
isEmpty() }

~ sorted ~

min() (kolay) O(1)
! insert(k,v) O(n)
removeMin() O(1)
} O(1)

— Sorting —

Holds the elements of an Array in a Priority Queue.
the elements of the array will be sorted

! keys → occurs from values.
! getMin, getMin... → display on the screen by ordered.

1 Selection Sort : Chose the min element from an unsorted list.

$1+2+3+\dots+n$
$n \cdot (n+1)/2 = \frac{n^2+n}{2} \rightarrow O(n^2)$ running-time

1- inserting the elements into P-Q
2- remove the element in sorted order from the P-Q

2 Insertion Sort : Chose the min element from a sorted list

$1+2+3+\dots+n$
$n(n+1)/2 = \frac{n^2+n}{2} \rightarrow O(n^2)$ running-time

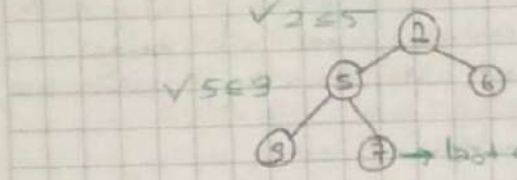
1- Inserting the element into a P-Q
2- Removes the element in sorted order from a sorted list

Recall P-Q Sorting
Running-time $O(\log(n))$ using trees

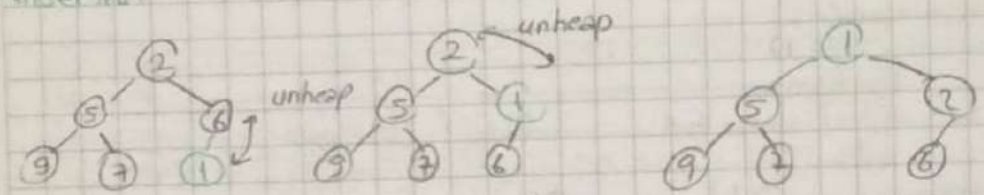
HEAPS

A more efficient realization of a priority queue is possible with using a data structure called a binary heap.
Insert + Remove → $O(\log(n))$
Heap → last node (known which is the last node)
→ rightmost node

parent ≤ child (the only condition)

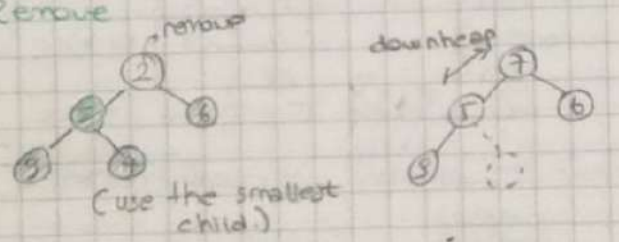


Insertion



Unheap → height O(logn)
→ running-time O(logn)

Remove



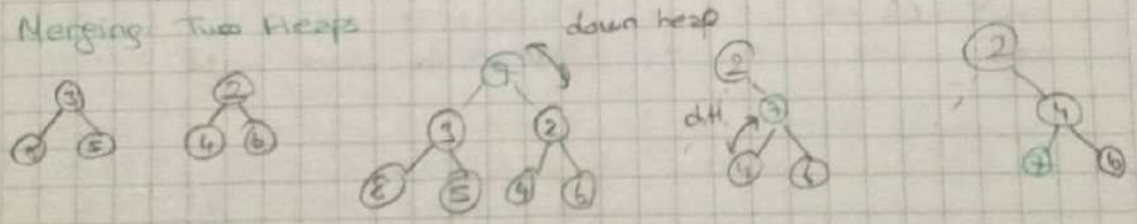
Heap Sort

≠ Space usage → O(n)

insert	O(logn)
removeMin	O(logn)
size	O(1)
isEmpty	O(1)
min	O(1)

*sort a sequence of n elements
⇒ O(nlogn) time

Merging Two Heaps



height = log(n)
running-time = log(n)

Entry and Priority Queue

An entry stores a (key, value) pair

getKey(): returns the key associated with this entry.
getValue(): returns the value pair with the key associated with this entry.
remove(e): remove from P and returns entry e
replaceKey(e,k): replace with k and return the key of entry e of P
(if it is not valid (Error occurs))
replaceValue(e,v): ..

Performance

	Unsorted	Sorted	Heap	
size, isEmpty	O(1)	O(1)	O(1)	
insert	O(1)	O(n)	O(log n)	
min	O(n)	O(1)	O(1)	
removeMin	O(n)	O(1)	O(log n)	
remove	O(1)	O(1)	O(log n)	entries
replaceKey	O(1)	O(n)	O(log n)	
replaceValue	O(1)	O(1)	O(1)	

MAPS

key-value entries
searching*, inserting, deleting.
key → unique!
unsorted

O(n) get(k) → key → return value
O(n) put(k,v) → insert entry(k,v) into the map, and return null
if key is already exist → return old value
O(n) remove(k) → remove entry with key k, (or return null)
size(), isEmpty()
entrySet() → return an iterable collection of the entries in map
keySet() → " " " " " keys in map
value() → " " iterator of the values in M.

Ch 3 : Arrays, Linked Lists, Double Lists

1 ARRAYS

An array is a sequenced collection of variables all of the same type.

Declaring

1. elementType[] arrayName = {initialValue$_0$, ..., initialValue$_{n-1}$};
2. declare = new elementType[length]; (returns a reference to the new array; create; positive)

Methods

equals(A, B)

fill(A, x) : Stores value x in every cell of array A

copyOf(A, n) : Returns an array of size n such that the first k elements of this are copied from A. If n > A.length, then the last n-A.length elements in this array will be padded with default values (0, null)

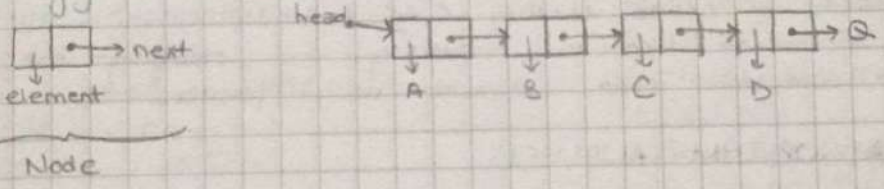
copyOfRange(A, s, t) : size = t-s, copied in order from A[s] to A[t-1]

toString(A) : Returns a string representation of the array A.

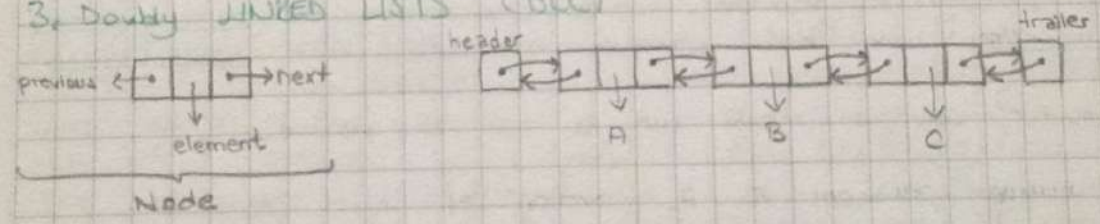
sort(A) : elements must be comparable

binarySearch(A, x) : Searches the sorted array A for value x, returning the index where it is found.

2. Singly LINKED LISTS (SLL)



3. Doubly LINKED LISTS (DLL)



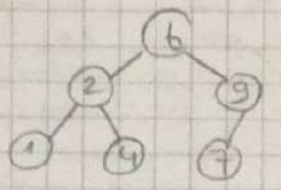
- SLL nodes contains 2 field data field, next link field.
- The traversal is possible in one direction only.

- DLL nodes contains 3 field data, a next link and a prev. link field.
- The traversal is possible in both direction.

Binary Search Trees

key(u) ≤ key(v) < key(w)

(inorder traversal)
left, node, right



Search

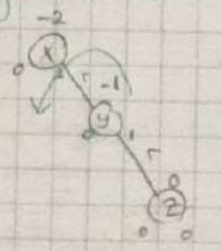
Algorithm TreeSearch(k, v)
  if T.isExternal(v)
    return v
  if k < key(v)
    return TreeSearch(k, left(v))
  else {k > key(v)}
    return TreeSearch(k, right(v))

get
put
remove
} O(h)

O(log n) ← height → O(n)
best case worst case

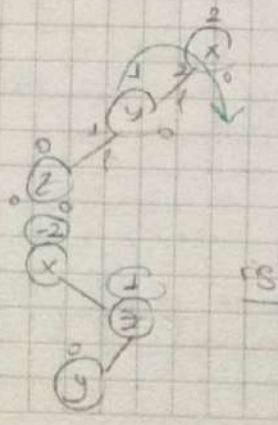
Rotation Operation

Single Rotation

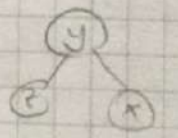


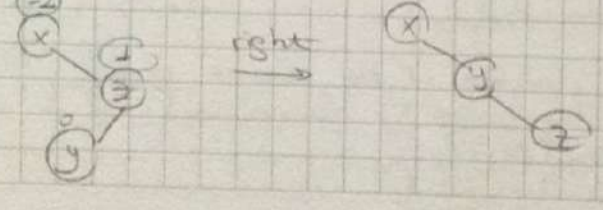
X balance = -2
-2 < -1
left rotation
r ← r → left t.

balance = left (height) - right (height)



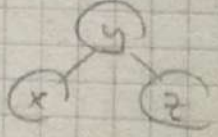
X balance = 2
2 > 1
right rotation
t ← t → right t.





right →

left →



QUEUES

First-in - first-out (FIFO)

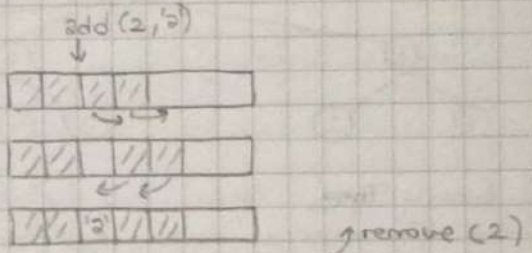
run-time

O(1) enqueue (obj) : inserts an element at the end of the queue
O(1) dequeue () : removes and returns first element? If empty ( ) return null
O(1) first () : returns first element
O(1) size () : return the number of elements of the queue
O(1) isEmpty ()

LISTS

Array List

	running-time
size ()	O(1)
isEmpty ()	O(1)
get (i)	O(1)
set (i, e)	O(1)
add (i, e)	O(n)
remove (i)	O(n)

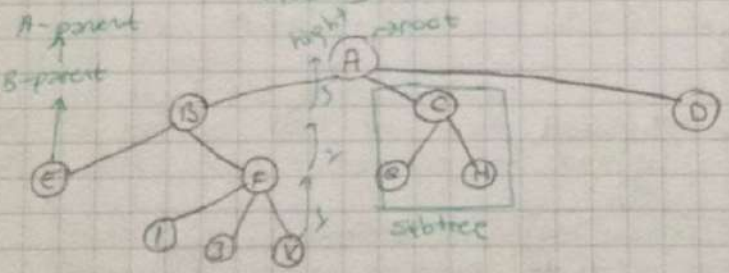


Iterators

An iterator is a software design pattern that abstract the process of scanning through a sequence of element, one element at a time.

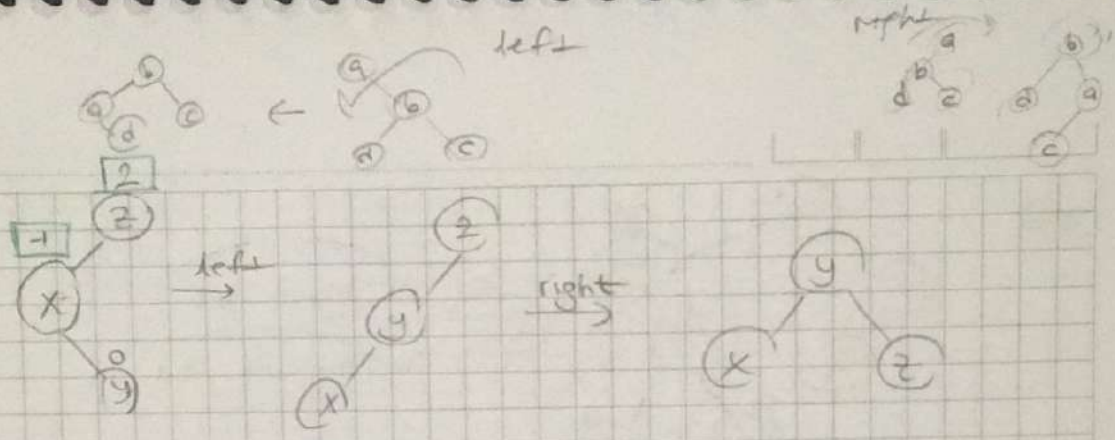
hasNext () : Returns true if there is at least one additional element in the sequence, and false otherwise.
next () : Returns the next element in the sequence.

~ TREES ~



depth of a node
depth E = 2
number of ascestor
hight of a tree
max. depth of any node (3)

internal node : at least one child (A, B, C, F)
external node : (leaf) without any children. (D, E, I, J, K, G, H)
ancestor = parent, grandparent ...
descendent = child, grand-child ...



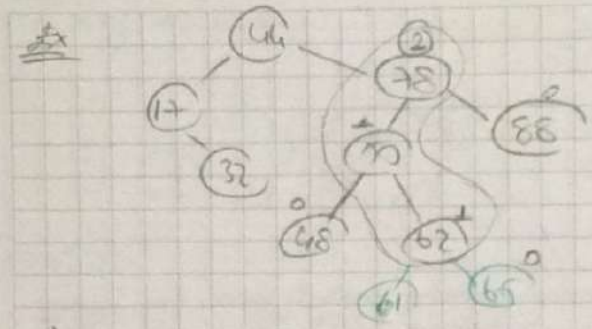
AVL Tree

AVL Trees are balanced, binary search tree. for every internal node v of t, the heights of the children of v can differ by at most 1.

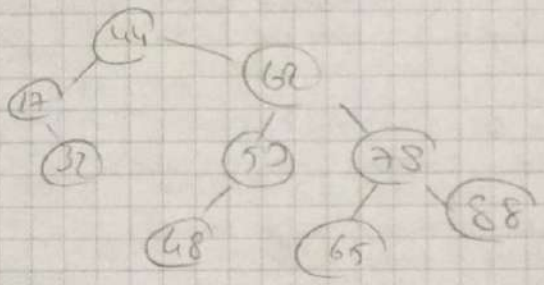
The height → O(logn)

Usage space → O(n)
Searching, Insertion, Deletion → O(logn)

```
public Node rightRotation(Node node){
    Node leftChild = node.left;
    Node rcOflc = leftChild.right;

    leftChild.right = node;
    node.left = rcOflc;

    leftChild.height = Math.max(getHeight(leftChild.left), (
                                getHeight(leftChild.right);
    node.height = Math.max(getHeight(node.left), getHeight(node.right))
                  +1;

    return leftChild;
}
public Node leftRotation(Node node){
    Node rightChild = node.right;
    Node lcOfrc = rightChild.left;

    rightChild.left = node;
    node.right = rcOflc;

    rightChild.height = Math.max(getHeight(rightChild.left), ----);
    node.height = Math.max(getHeigh(--), getHeight(--))+1;

    return rightChild;
```



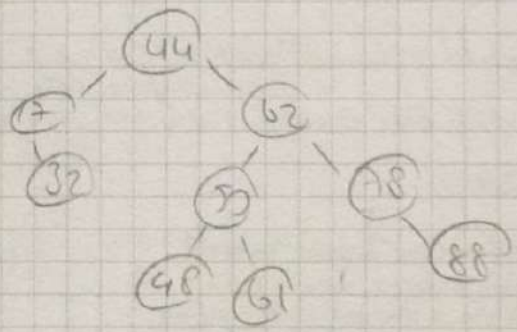
a)



right'ı, right'ının left'i olur!

65 = 62 right
62 right → 62. right. left

b)



left'i = left'inin right'ı olur!