:= Contents.

Source:- programiz.com/DSA.

Graphs:-

Sorting and searching Algorithms:-

Greedy Algorithms:-

- Greedy Algorithm → 37
- Ford-Fulkerson Algorithm → 37
- Dijkstra's Algorithm → 37
- kruskal's Algorithm → 38
- Prim's Algorithm → 39
- Huffman code → 40

Dynamic Programming:-

- Dynamic Programming → 42
- Floyd warshall Algorithm → 42
- Longest common subsequence → 44
- Backtracking Algorithm → 45
- Rabin-Karp Algorithm → 45.

Introduction:- /programiz.com/dsa/

- //Algorithm//

* In computer programming terms, an algorithm is a set of a well-defined "instructions" to solve a particular problem.

An algorithm to "add two" numbers:

1) Take two numbers as inputs.
2) Add numbers using the + operator
3) Display the result.

Qualities of a good algorithm:-

> input and output should be defined precisely.
> Each step in the algorithm should be clear and unambiguous.
> Algorithms should be most effective among many different ways to solve a problem.
> An algorithm shouldn't include computer code. It can be written in English / steps which can be used in any programming languages.

- Data structures and types:-

Data structure:- Data structure is a "storage" that is used to store data in an "organised way".

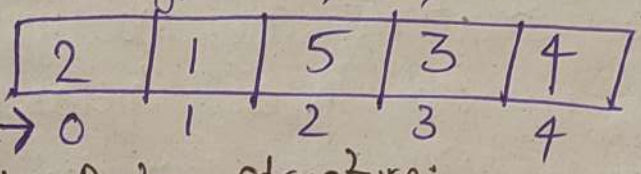
Types:- < Linear DS
Non-linear DS.

Linear data structure:-

* In linear data structures, the elements are arranged in "sequence" one after the other.
* If "length" of linear DS increases then the searching time is increased and it will be "complex".

1. Array Data structure:-

* In an array, "elements" in memory are arranged in "continuous memory". All the elements of an array are same type [int, char, float].



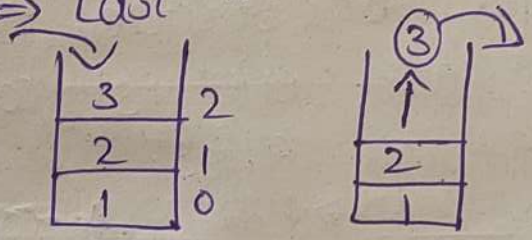
Python Lists
a = ["1", "2", "3"]

2) Stack Data structure:-

In ~~(an)~~ stack data structure, elements are stored in the LIFO principle.

LIFO ⇒ Last in first out.

Ex:-



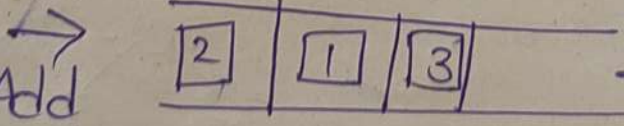
Lunch Box

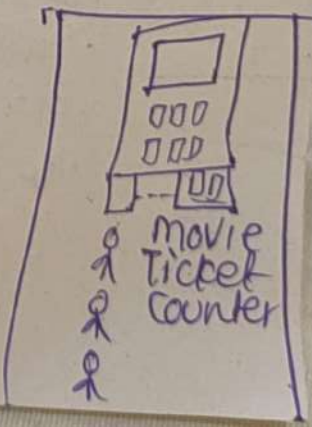
Last in first out.

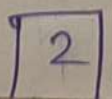
3) Queue Data structure:-

unlike the stack, the Queue data structure works on the FIFO principle.

FIFO → first in first out.





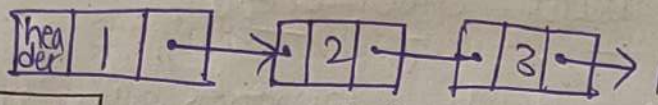


Aton = 4) Linked List Data structure:-

In linked list data structure, data elements are connected through a series of node.
And each node contains the data items and address to the next node.

① Head → [1|next] → [2|next] → [3|next] → Null

②

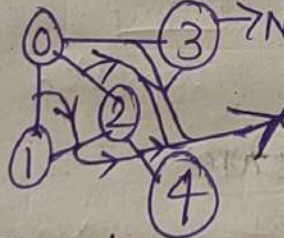


Non-~~linked~~ linear data structures:-

* Unlike linear data structures, elements in non-linear data structures are not in any sequence.
* They are arranged in a hierarchy manner where one element will be connected to one or more elements.

1) Graph data structure:-

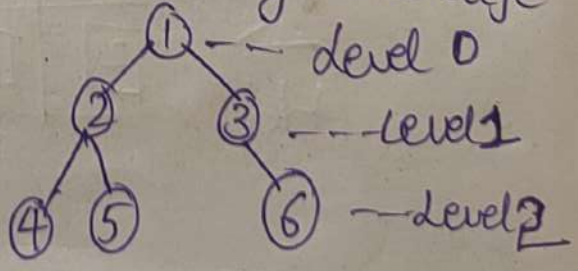
In graph data structure, each node is called vertex. and each vertex is connected to other vertices through edges.



* Connecting lines from Node/vertex to N/V is an edge.

2) Tree data structures:-

Similar to graph, a tree is also a collection of vertices/Nodes and edges. There is only one edge b/w the vertex/Nodes.



— Why learn DSA?

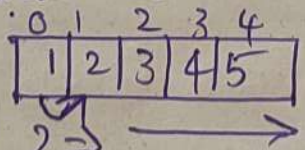
* what are algorithm; is nothing but a mention of steps to solve a problem. They are essentially a solution.
* Programming is all about data structures and algorithms.
* Data structures are used to hold / store the data
* Algorithms are used to solve problems.
* Time is precious; more on scalability;
  → Scalability is the capacity to be changed.
* memory is expensive.

— "Asymptotic Notations:"

Asymptotic Notations are the mathematical notations used to describe the running time of an algorithm when the i/p tends towards a particular value or a limiting value.

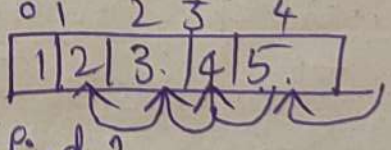
→ In linked list the searching value found at first is best case in forward search.
→ In linked list the searching value found at last is worst case in Reverse search.

Ex:-



find 2 ⇒

Forward two steps
Best Case



find 2

Reverse four steps
worst case.

> Big o notation <
> omega notation <
> Theta notation <

"Big-O Notation (O-notation)"

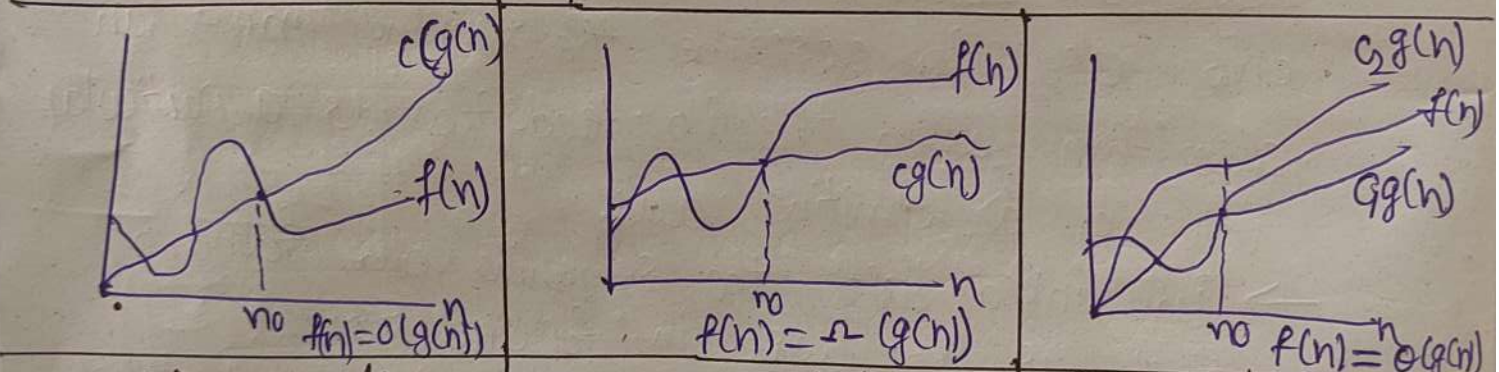
Big-O notation represents the upper bound of the running time of an algorithm. it gives worst-case complexity of an algorithm.

Omega Notation (Ω-notation):-

omega notation represents the lower bound of the running time of an algorithm. It gives best-case complexity of an algorithm.

Theta Notation (Θ-notation):-

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm. It gives the average case.



Big-O Notion	Omega Notation	Theta Notation

Master theorem:-

The master method is a formula for solving recurrence relations of the form:

$$T(n) = aT(n/b) + f(n)$$

n = size of input

a = no. of subproblems in the recursion.

n/b = size of each subproblem. (size is equal for subprobl)

f(n) = Total cost to the problem solution

Recursion: Function calls itself.

$a \geq 1$ & $b > 1$

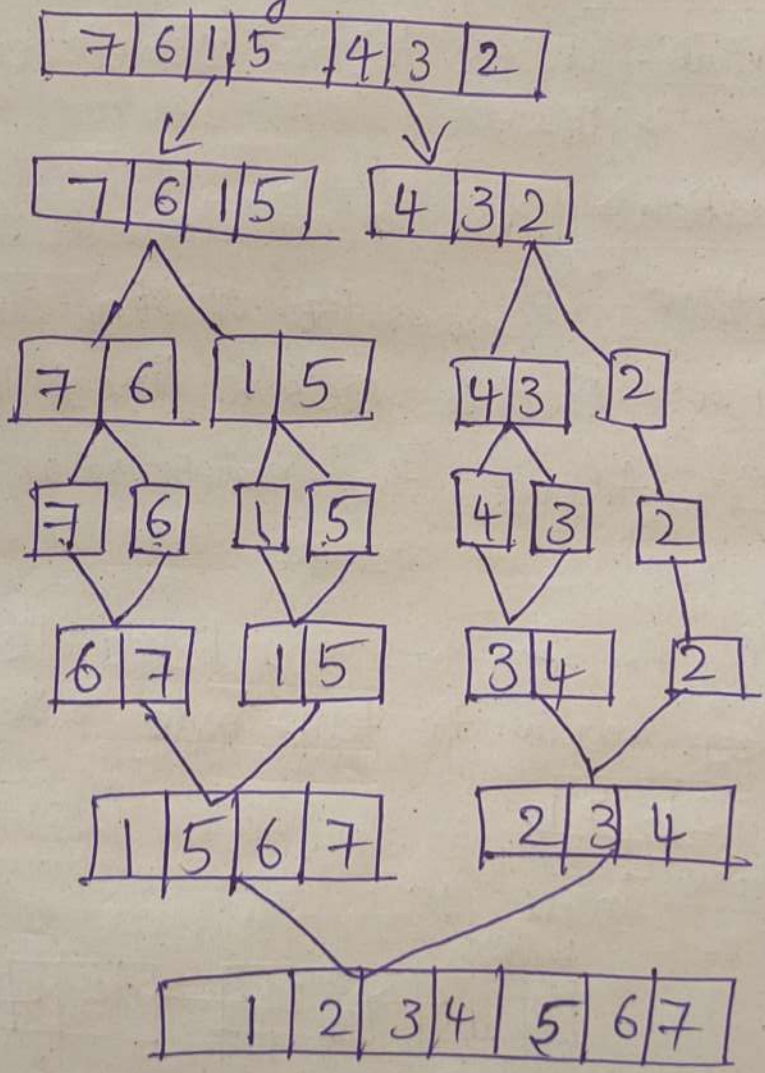
Divide and Conquer Algorithm:-

A divide and conquer algorithm is a strategy of solving a large problem by

1. Breaking the problem into smaller sub problems.
2. Solving the sub problems and
3. Combining them to get the desired output.

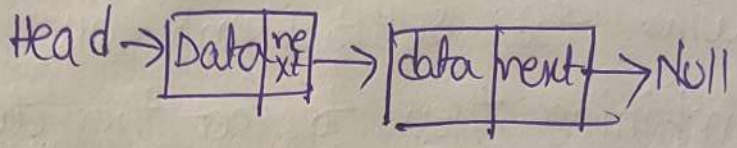
1> Divide --> divide problem into sub Pb using recursion.
2> Conquer --> Solve the small sub problems.
3> combine --> Combine the solutions of sub-problems that are part of the recursive process.

PB:- Sort an array :-



Linked List:- /Programiz.com/dsa/

A linked list is a linear data structure that includes a series of connected nodes.



Representation of Linked List:-

a data item and an address of another node.

Pointer → stores a memory address.

Operations:-

Traversal:- access /visiting each element of the linked list

Insertion:- adds a new element to the linked list.

Deletion:- Removes the existing elements.

Search:- Find a node in linked list

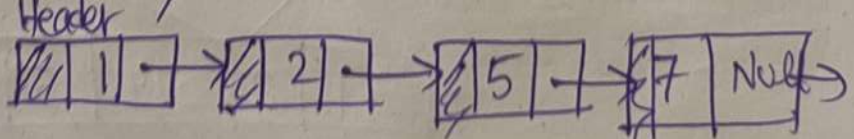
Sort:- Arrange the nodes in an order.

Traverse:-

Display the contents of a linked list is very simple. We keep moving the temp node to the next node.

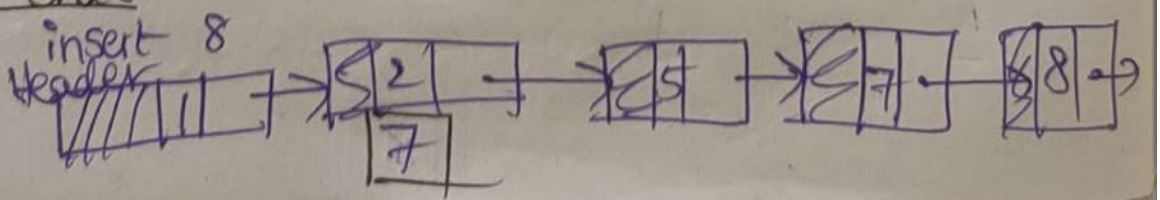
Insert:- 2 5 7

1) At begining:- insert 1;



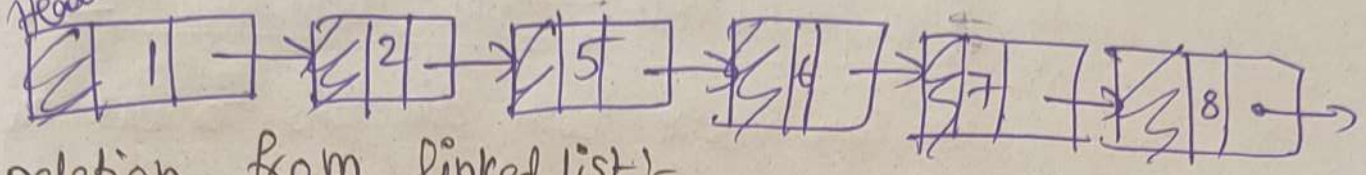
2) At the end:-

insert 8



3) Insert at the middle:-

Insert 6

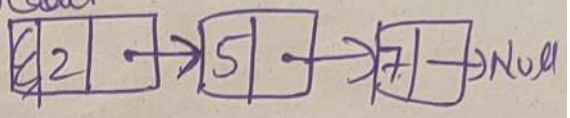


Deletion from linked list:-

Deletion at begining/end/middle

Delete 1, 8, 6

After step by step result:-



Search an element in an linked list:-

By using loop we can search.

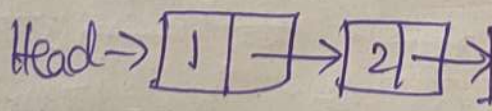
- make (head) as the (current) node
- Run a loop until the (current) node is (Null) because the last element points to null.
- In each iteration, check if the key of node = item. Return true otherwise false.

Sort elements:- (Arranging) in a order.

1) Head as current node and another node inden for later use
2) If head is null return
3) Else, run a loop till the last node. index.
4) follow 5 & 6 5) Store next node of current in
6) check if the data of the current node is greater than next node.

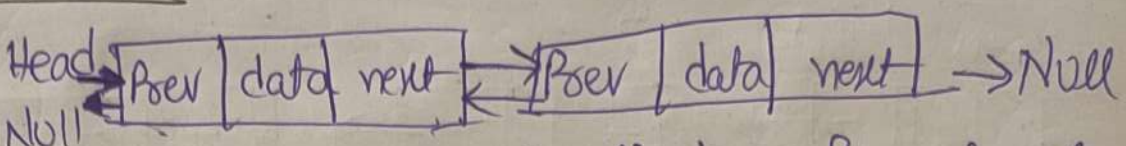
Types of linked list:-

Single linked list:-



we can move only forward direction.

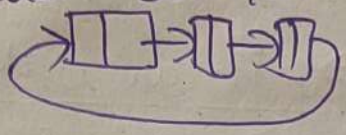
Double linked list:-



we can move both directions forward and Backward.

circular linked list:-

In a circular linked list is a variation of a linked list in which the last element is linked to the first element.



Hash Table:-

The hash table data structure stores elements in key-value pairs where.

Key → unique integer used for index values

Value → data that are associated with keys.

Hashing:- In a hash table, a new index is processed using the keys, And the element corresponding to that key is stored in the index.

Hash collision:- If there is same (values) index for multiple keys.

Collision resolution by Chaining:-

using double linked list.

Collision resolution by open addressing:-

Doesn't store multiple elements into same slot.

Sol → Good Hash functions:- K ⇒ key; m is size of table

> Division → m=22; K=17; h(K) = 17 mod 22 = 01

> multiplication → h(K) = ⌊m (KA mod 1)⌋ → floor

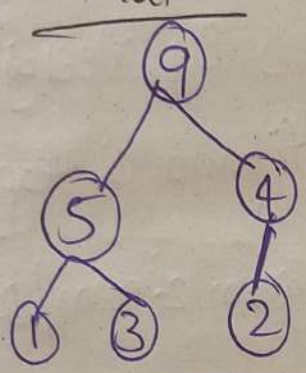
> universal → Random independent keys.

Heap data structures:-

max heap tree:- Alway greaterthan Child node & key of root of node is the largest among allother nodes.

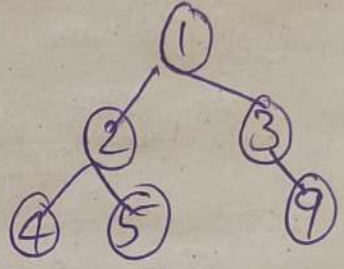
min heap tree:- Always smaller than the child node is and the key of root node is the smallest among all the other nodes.

max



9>5,4
5>1,3
4>2

min

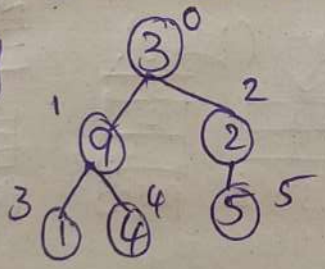


1<2,3
2<4,5
3<9

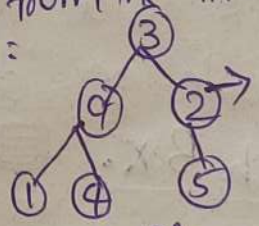
Ex:- | 3 | 9 | 2 | 1 | 4 | 5 |
0 1 2 3 4 5

① as input:

② Binary tree



③ start from the first index of non-leaf n/2-1:



④ Set current element i as largest

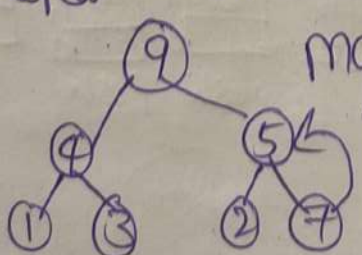
④④ Set current element (i) as largest

⑤ The index of left child is given by 2i+1 & the right child is given by 2i+2.
if left child is > current set left as largest.
If right child is greater than element in largest, set right child index as largest

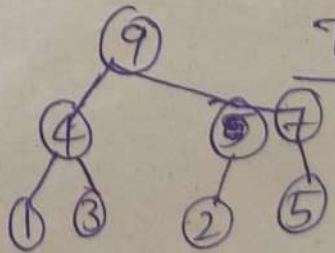
⑥ Swap largest with current element.

⑦ Repeat steps 3 → 7

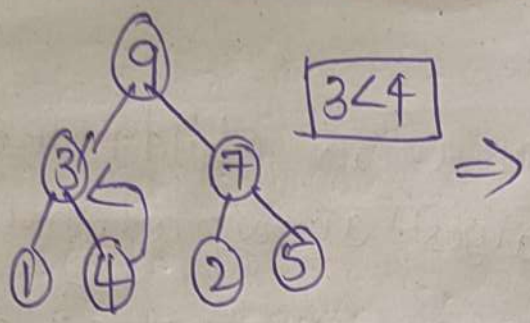
① max heap tree →

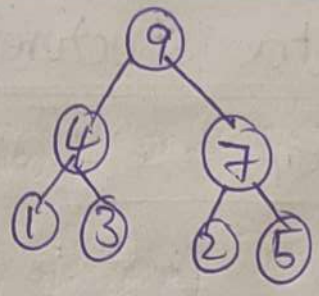


⇒



9>4,7
4>1,3
7>2,5

2) 



Fibanacci Heap:-

Follow min/max heap tree.

(or)

watch in youtube:- TechLearners By Neeraj Saxena

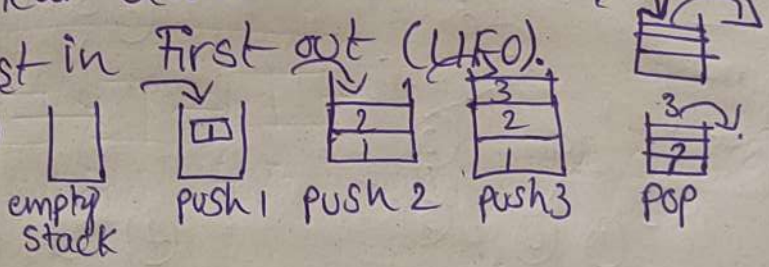
Decrease key & Delete key in fibanacci Heap.

Stack and queues:- /Programiz.com/DSA/

Stack:-

A stack is a linear data structure that follows the principle of last in First out (LIFO).

Ex:- Lunch Box.



Basic operations:-

Push:- Add an element to the top of the stack.

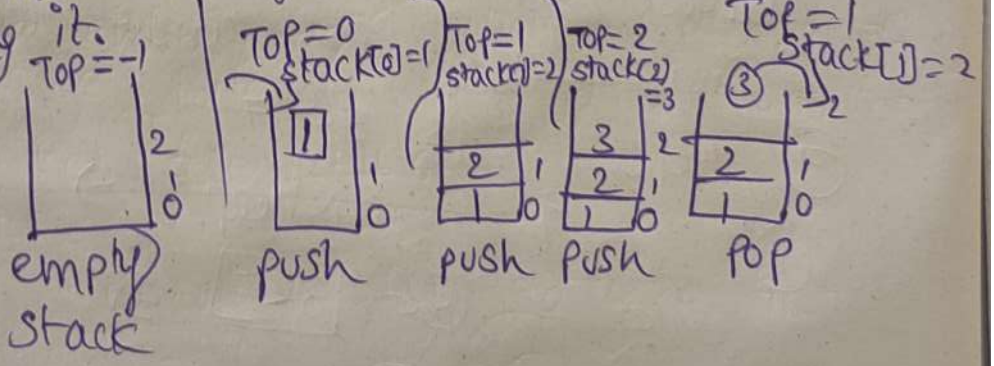
Pop:- Remove an element from the top of the stack.

IsEmpty:- check if the stack is empty.

IsFull:- check if the stack is full.

Peek:- Get the value of the top element without removing it.

Top = (index)



operations:-

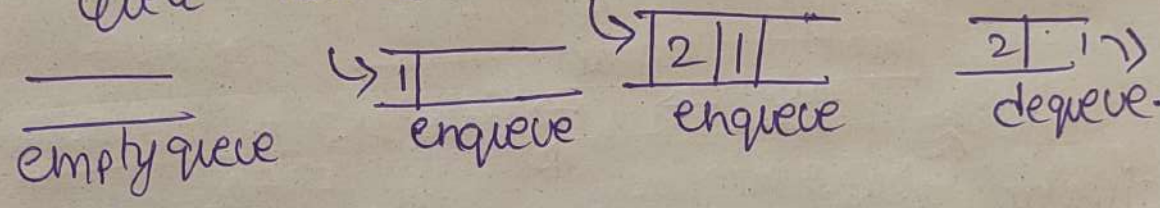
To reverse a word:- Put all the letters in a stack & Pop them out.

In compilers:- 2+4/5*(7-9) expression prefix/postfix.

In Browsers:- when visiting websites in browser saved in stack. when we press back button. the current url is removed.

Queue:-

A queue is useful data structure in programming. It is similar to the ticket queue outside a cinema hall. Queue follows the first in first out.



operations:-

Enqueue:- Add an element to the end of the queue.

Dequeue:- Remove an element from the front of the queue

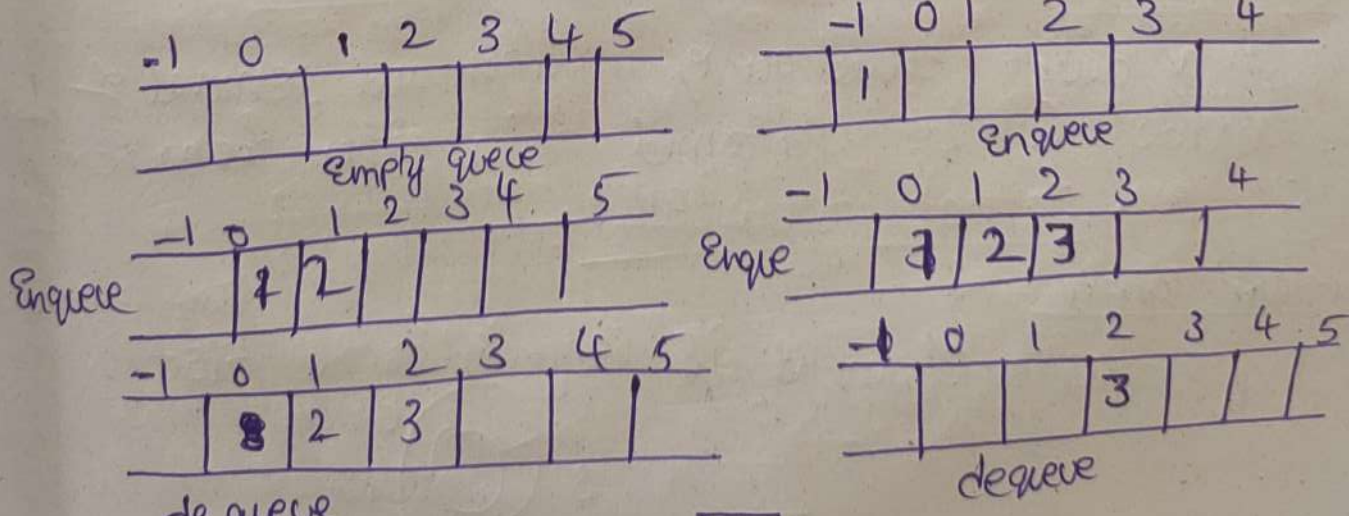
Is empty:- Check if the queue is empty.

Is full:- Check " " " is full.

Peek:- Get the value of front of queue without removing it.

Working of Queue:-

Front & Rear.



Limitations of Queue:-

We can only add indexes 0 and 1 only when the queue is reset.

If the queue is deleted (dequeue) we can use empty use spaces (0&1).

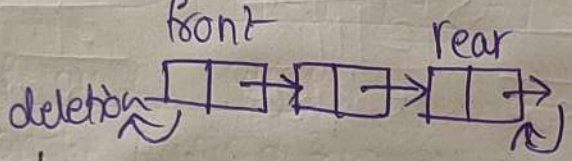
Applications:-

CPU & Disk scheduling; pipes; file; Call center.

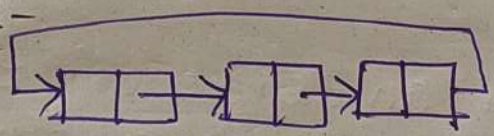
Types of Queue:-

simple queue:- (FIFO).

Insertion at rear, Removal at front.



Circular queue:-



If the last is full and first is empty we can use first index space.

Priority queue:-

A Priority queue is a special type of queue in which each element is associated with a Priority. and is served according to its Priority.

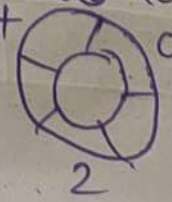
∴ Do not follow FIFO:-

Dequeue:-

In a double ended queue, insertion and removal of elements can be performed from either from front or rear. Do not follow FIFO.

Circular queue:-

Last element is connected to the first element.



* we can use empty space of first index at front index after deque operation.

working:-

when we try to increment the pointer and we reach the end of the queue, we can start from the begining of the queue.

operations:-

Front & Rear; initially front and rear > -1

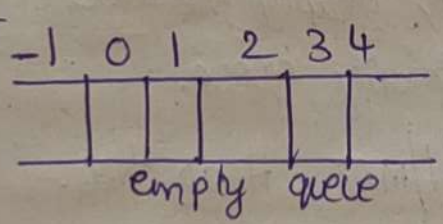
Enqueue:-

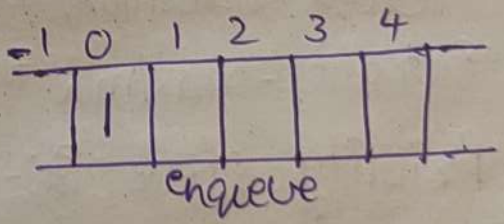
if queue is fully for first element, set value of front to 0 increase it by 1, add the new element.

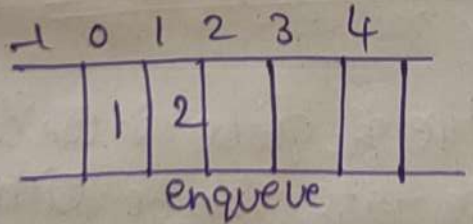
Dequeue:-

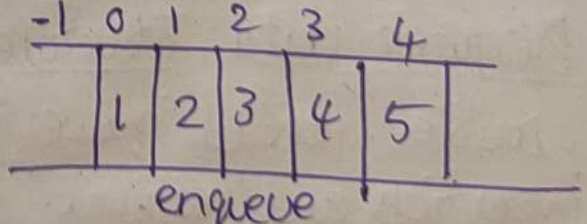
If queue is empty; Return the value of front; increase index by 1, for the last element reset the values front and Rear to -1.

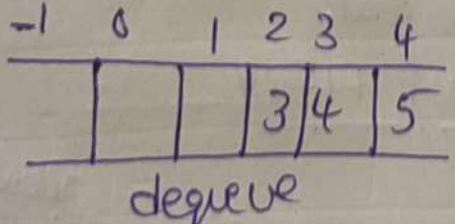
Ex:-

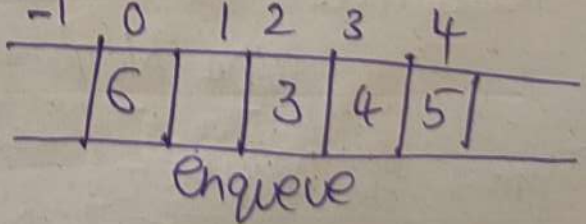












Applications:- CPU Scheduling, memory management, Traffic management.

Priority queue:-

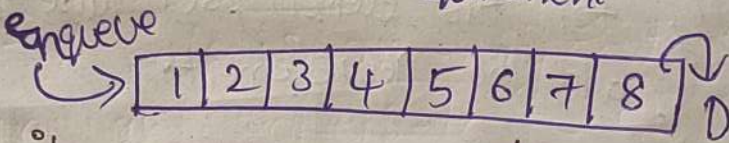
Assigning Priority value:-

Generally, the value of element itself is considered for assigning the priority.

Highest (or) lowest values of element are considered as priority as per our requirement.

we can also set priorities as per our needs

Priority ← highest value / Lowest value / our Requirement — as per our need.



Priority queue v/s Normal queue

// on the Basis of Priority — FIFO

~~FIFO~~ Not Follow FIFO.

Priority queue operations:-

Enqueue/Insertion, Deletion, Heapify the tree, Extract-max/min from Priority queue.

Priority queue is:- MIN/MAX Heap tree.

Deque data structure:-

Double ended queue is a type of queue in which insertion and removal of elements can be either performed front and the rear.

//Do NOT FOLLOW FIFO//

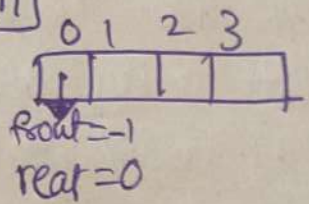
Types of Deque:-

Input Restricted:- i/p is restricted at single end but deletion allows both side

output restricted:- o/p // // // // but insertion allows at both the ends.

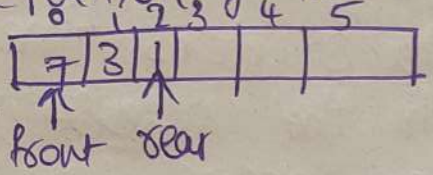
operations on a deque:-

1) Take an array (deque) of size [n]
2) Set front = -1 & rear = 0.

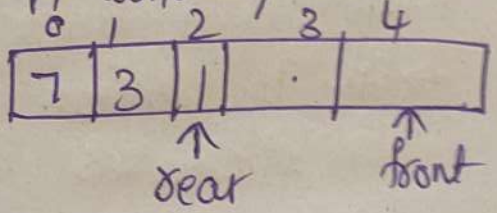


Insert at the front:-

① check the position of front

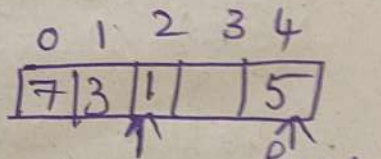


② if front < 1, then front = n-1



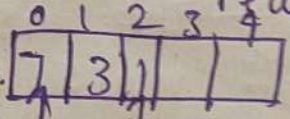
③ Else, decrease front by 1.

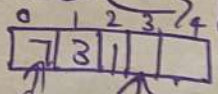
④ Add the new key [5] into array [front].

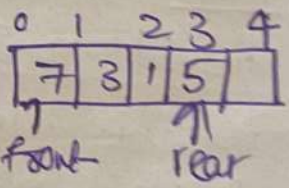


Insert at the Rear:-

1) check if the array is full. →
2) If the deque is full, reinitialize
3) Else, increase [rear] by 1.
4) Add the new key [5] into array [rear].







Delete from the front:-

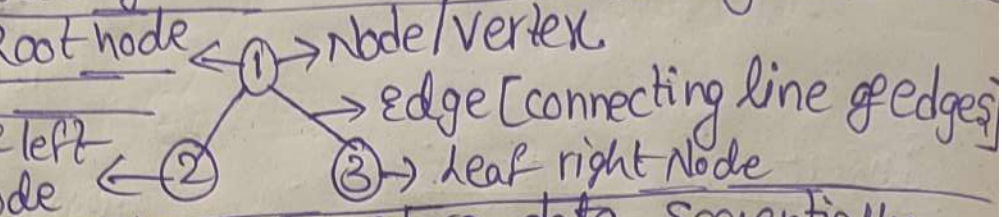
Delete from the end; check empty; check full.

Applications of deque Data structure:-

1) In undo operations, on software
2) To store history in Browsers.
3) For Both Stacks & Queues.

"Tree data Structure" /programiz.com/dsa/

* A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



* Linear data structures store data sequentially.
* In linear data structures time complexity increases with the increase of size of data.
* In trees we can access data quickly.

Node:- A node is an entity that contains a key or value and pointers to its child nodes.

—The last nodes of each path are called leaf nodes or external nodes.

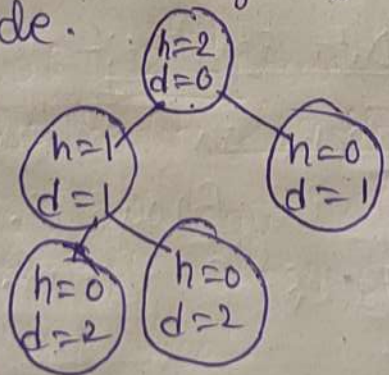
Edge:- It is the link between any two nodes.

Root:- It is the topmost node of a tree.

Height of a Node:- The longest path from the root node to the leaf node.

Depth of node:- The no. of edges from root to the node.

Height of tree:- The height of root node or depth of the deepest node.

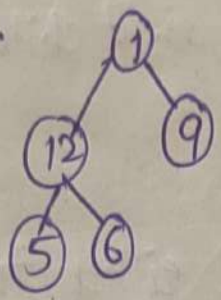


d = from root to x node
h = from x node to leaf node.

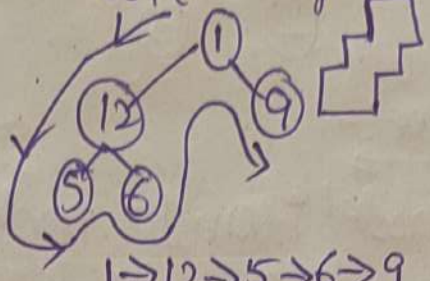
Degree of a node:- Total no. of branches of that node.

Tree traversal:-

Traversing a tree means visiting every node in the tree.

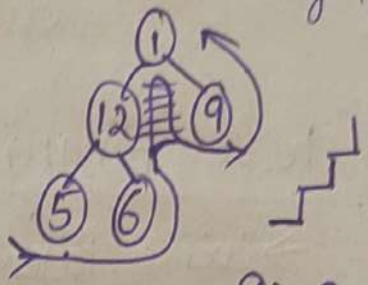


start from top left to right



1→12→5→6→9

start from bottom left to right



5→6→12→9→1

Inorder traversal:-

1) First visit all the nodes in the left subtree 2) Then root node 3) visit all the nodes in the right subtree.

Ex:- www.Education4u.youtube.com ⇒ ~~root, left, right~~ left, root, right.

Preorder traversal:-

1) vist root node 2) visit all left 3) visit all right nodes

root, left, right

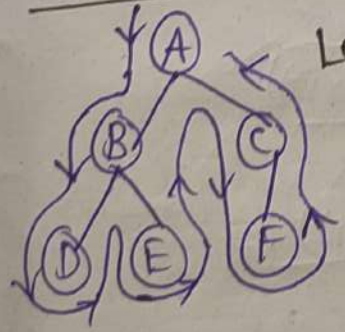
Post order:-

1) Left subtree 2) Right node 3) Visit root node.

Logic:- Left to right is common for all.

In ⇒ 2nd middle | Pre ⇒ First root | Post ⇒ Last root.

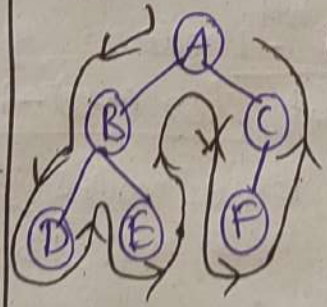
In order



Left to right Root in middle

D→B→E→A→F→C

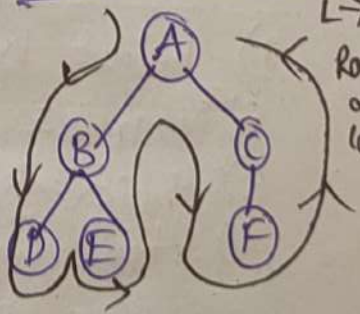
Pre order



Left to right Root at First

A→B→D→E→C→F

Post order.

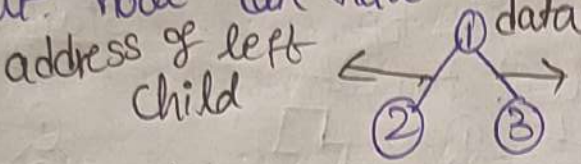


L→R Root at last

D→E→B→F→C→A

Binary tree:-

Binary tree is a tree data structure in which each Parent node can have at most two children.



Full Binary tree:-

→ Every Parent node/ internal node has either two or no children.

FBT Theorem:-

1) The no. of leaves $i+1$
2) The total no. of nodes $2i+1$
3) No. of internal nodes $(n-1)/2$
4) No. of leaves $= (n+1)/2$
5) Total no. of nodes $2l-1$
6) The total internal nodes is $l-1$
7) No. of leaves is at most $2^{\lambda-1}$

let i = the no. of internal nodes / λ = no. of levels.
n = be the total no. of nodes / l = no of leaves

Perfect Binary tree:-

→ Every internal node has exactly two child nodes & all leaf nodes at same level.

PBT Theorem:-

1) height h has $2^{h+1}-1$ node
2) N nodes has height $\log(n+1)-1 = O(\ln(n))$
3) height h has 2^h leave nodes
4) Avg depth of a node in a is $O(\ln(n))$.

Complete Binary tree:-

All levels are completely filled except possibly by the lowest one which is filled from the left.

1) All the leaf elements must lean left.
2) The last leaf element might not have right sibling.

Balanced Binary tree:-

Height balanced tree, height of left subtree - height of right subtree < 1.

1)

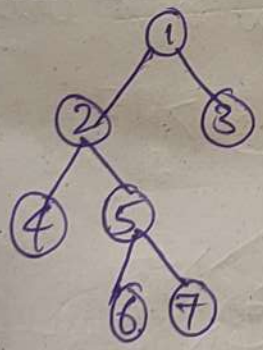
Binary Search Tree (BST):-

Binary search tree is a data structure. that quickly allows us to maintain a sorted list of numbers.

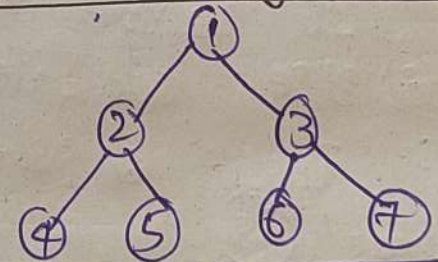
AVL Tree:-

AVL tree is a self-balancing binary search tree.
Balance factor -1, 0, +1.
Balance factor = Height of leftsub tree - Height of rightST.

Full Binary tree:-

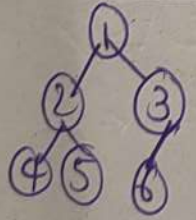


Perfect Binary tree:-



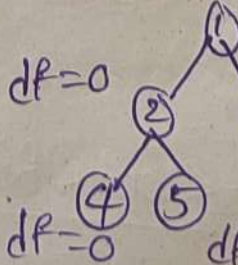
Level
Parent must have two child nodes
Leaf nodes at same level.

complete Binary tree



Level → fill
Leaf → Lean Left
No right sibling

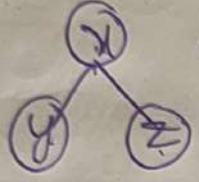
Balanced Binary tree:-



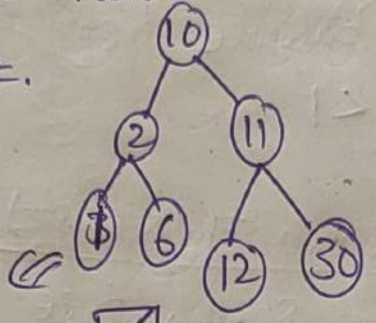
height of the left - height of right subtree
0 [or] 1

Binary Search Tree:-

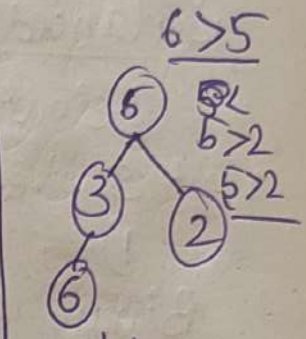
In BST, nodes left child must have values less than its root/Parent value & nodes right child must have a greater than its Parent root.



$\therefore y < x;\ x < z.$



Left < Root < right $2 < 6$



Not a Binary tree

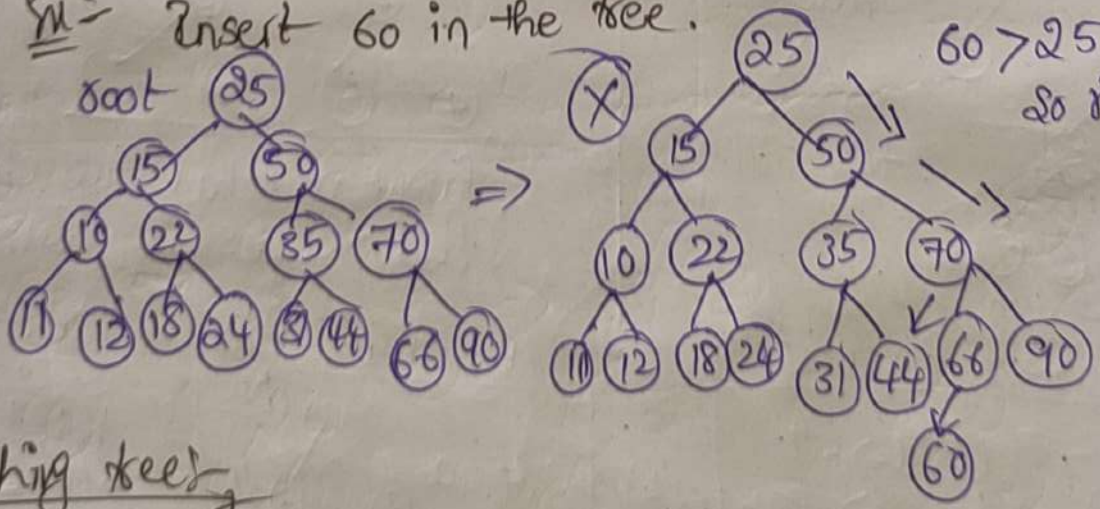
operations in Binary Search tree:-

Insertion:-

Educational. youtube.com

Always insert new node as a leaf node.
Start root node as current node.
If new node < current key
1) If current node has a left child, search left
2) else add new node as current left child.
If new node > current key.
1) If current node has right child, search right
2) else add new node as current right child.

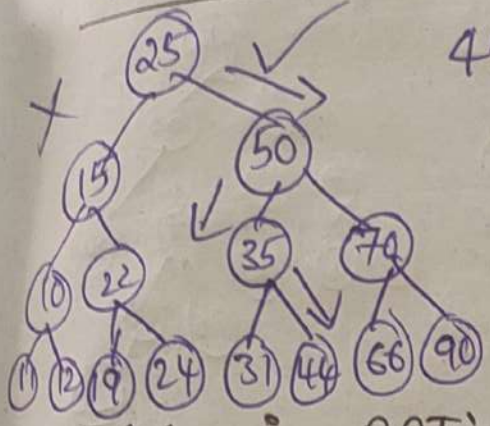
Ex:- Insert 60 in the tree.



Searching tree:-

1) Left < root < Right

Search 45 in the tree:-

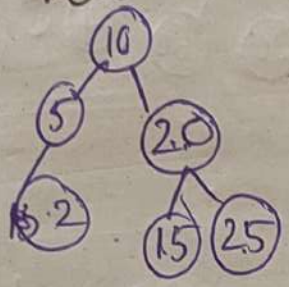


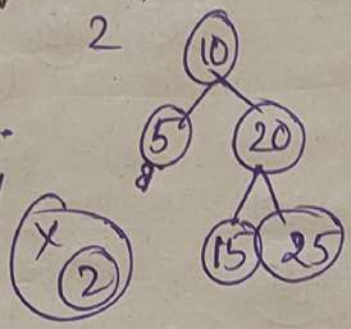
45 > 25 so right side.
45 < 50 so, again left.
35 < 45 so - right
so, 45 not present.

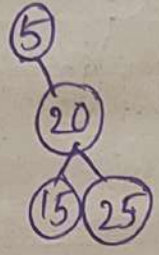
Deletion in BST:-

Algorithm; Search and delete if its leaf node else.
Replace largest value on right & small in left

Ex:- delete 2 & ~~25~~, 10,







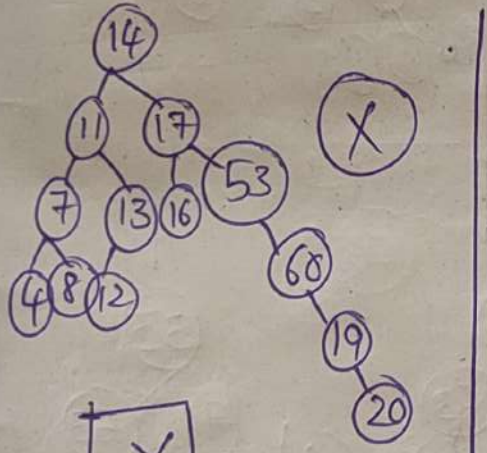
AVL Tree:-

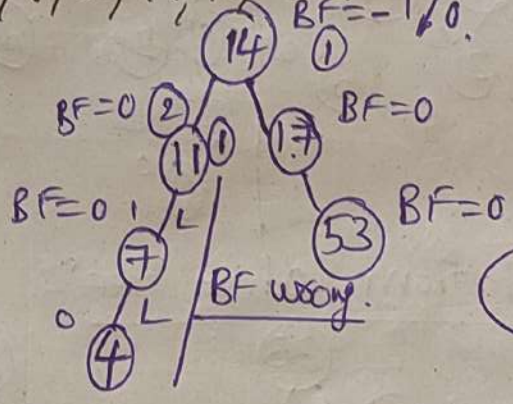
(Balance factor 0, 1, -1;

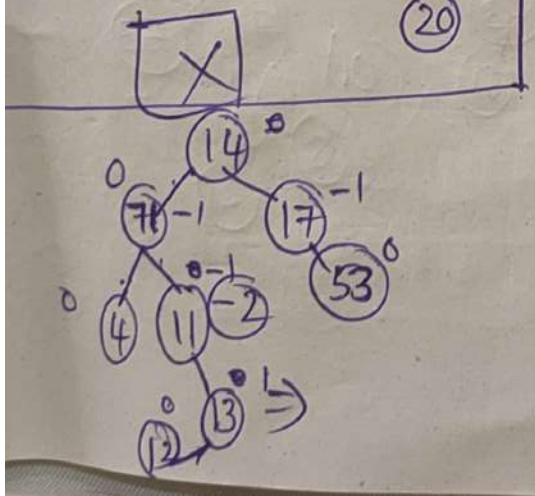
14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20

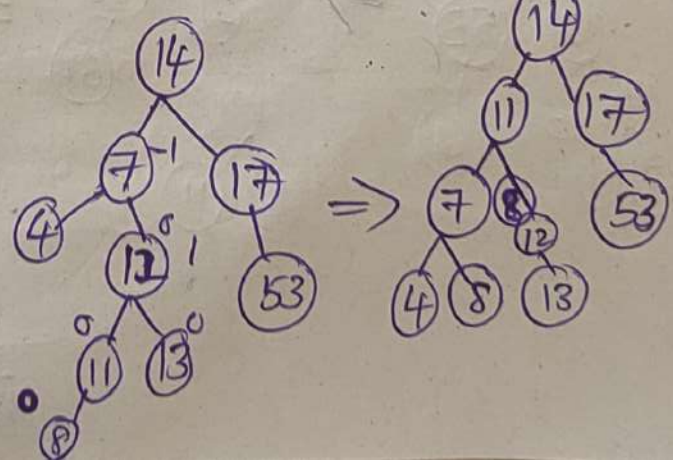
Left < root < Right

Always check BF.



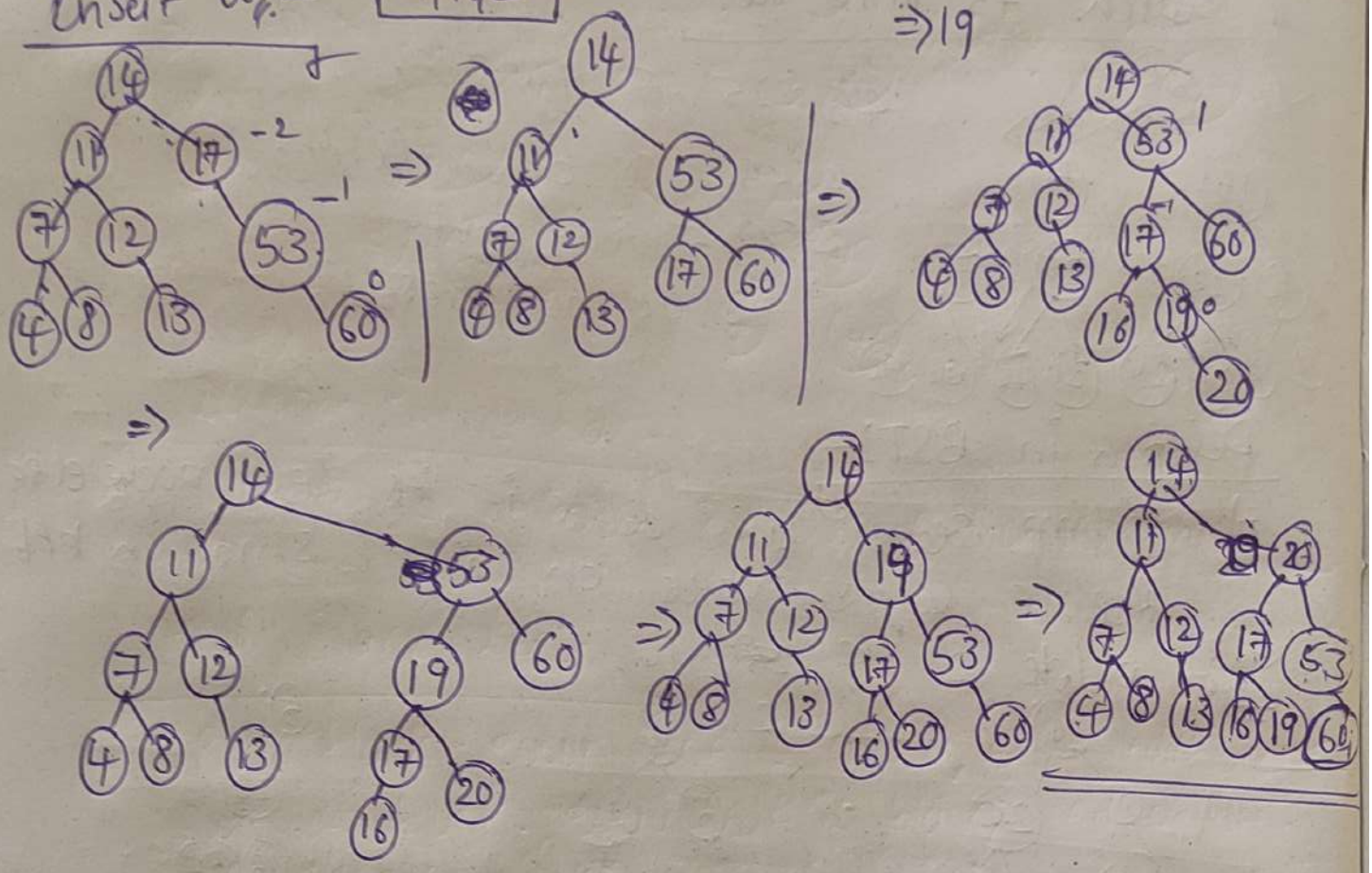




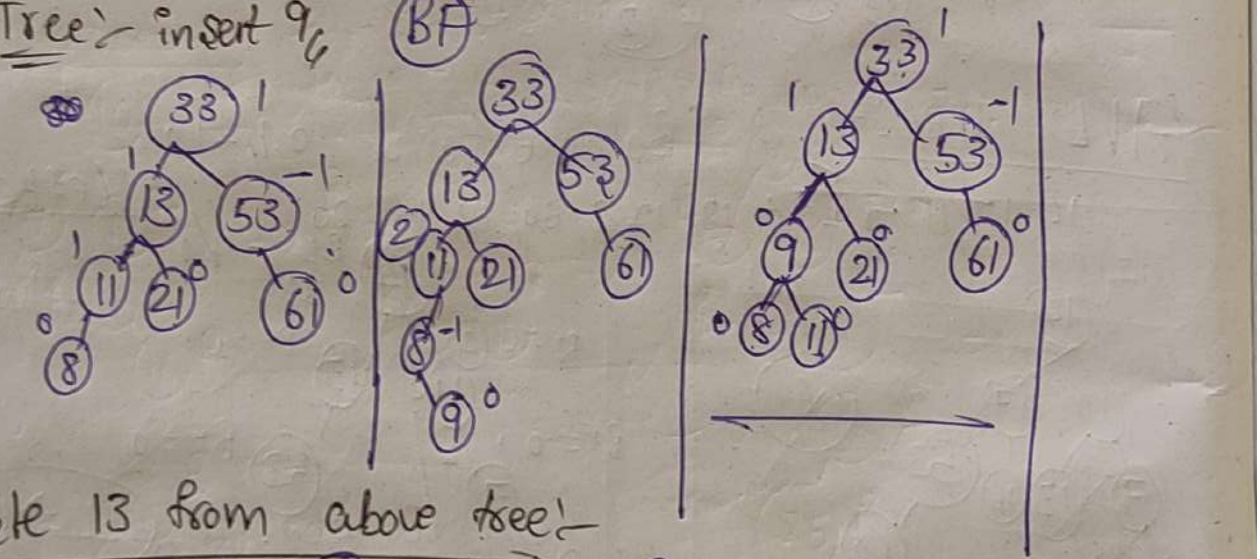


19, 14, 20

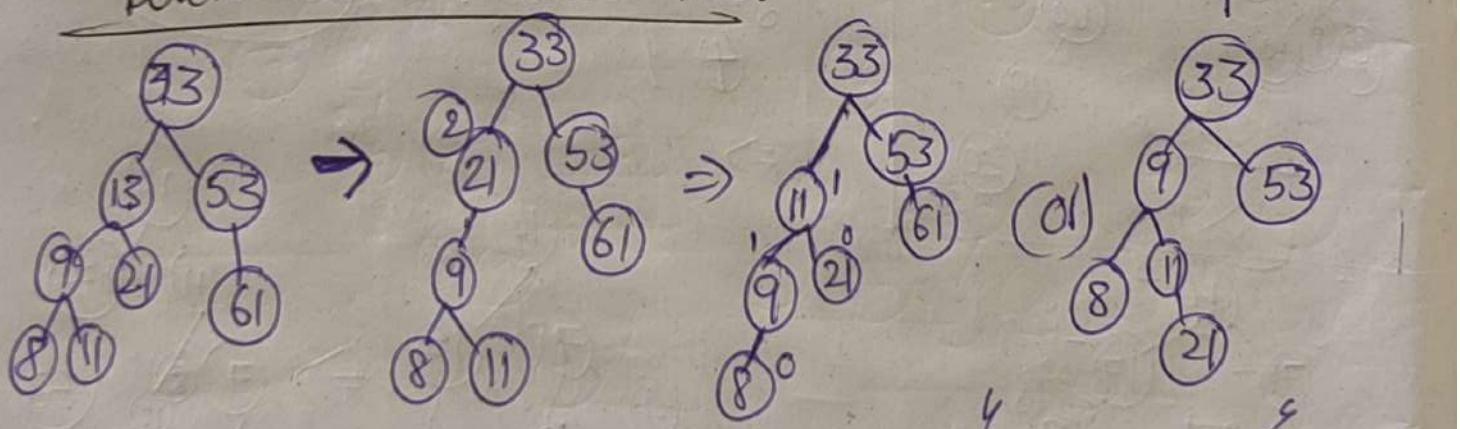
Insert 60:-



Tree:- insert 9 (BF)



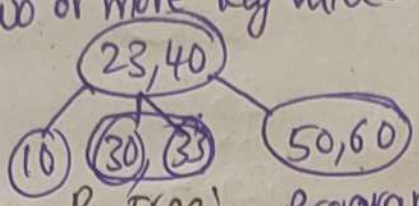
delete 13 from above tree:-



B. Tree:-

have two or more key values and children.

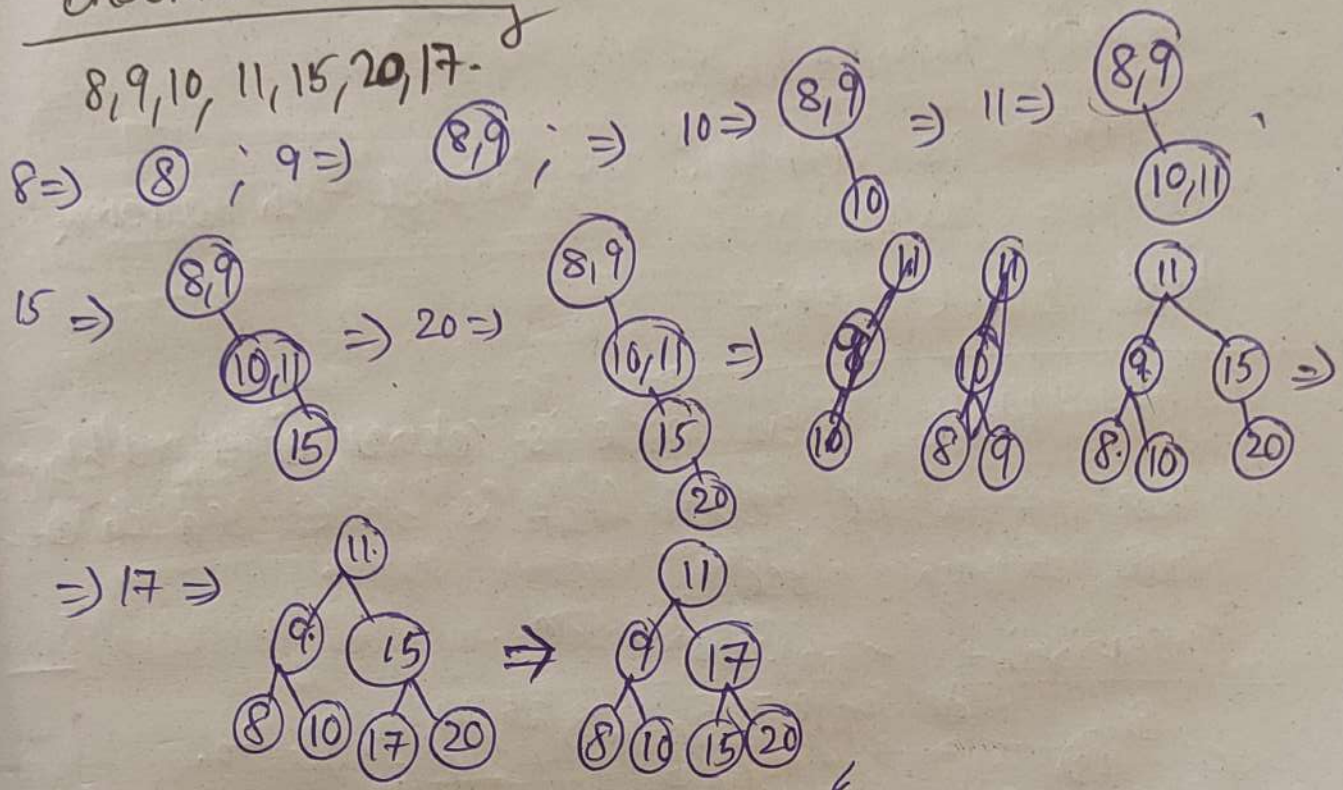
Ex:-



Searching in B-Tree:- Programiz.com/DSA

Insertion in B-Tree:-

8, 9, 10, 11, 15, 20, 17.



Deletion from a B Tree:-

If leaf node; neglect. If internal node, rearrange.

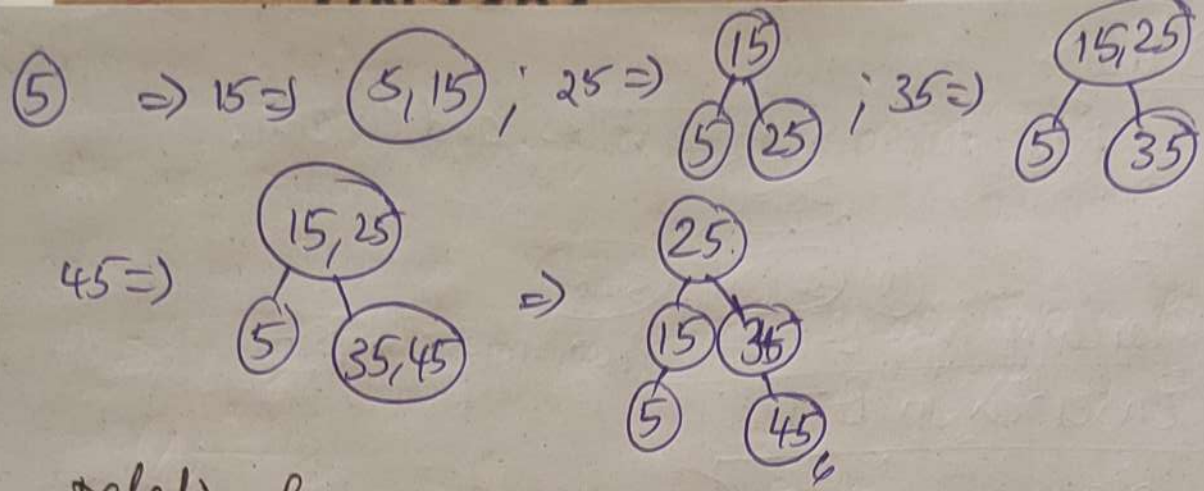
B+ tree:-

All the values are present in Leaf nodes.

1) All leaves are at same level.
2) The root has at least two children.
3) node have m or m/2 children.
4)

Searching:-

Insertion:- 5, 15, 25, 35, 45.



Deletion from BT tree:-

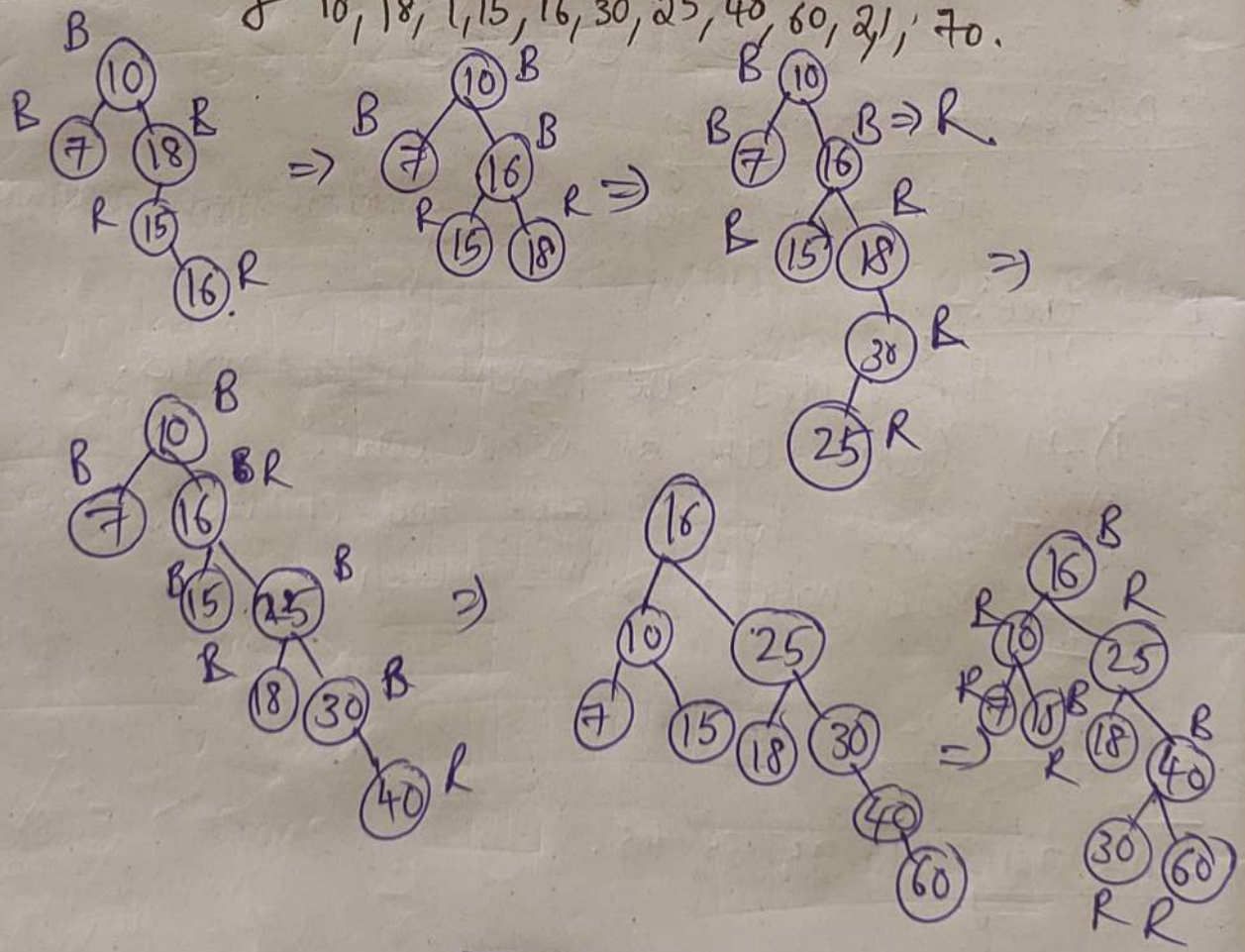
Leaf is neglected; & if internal then rearrange.

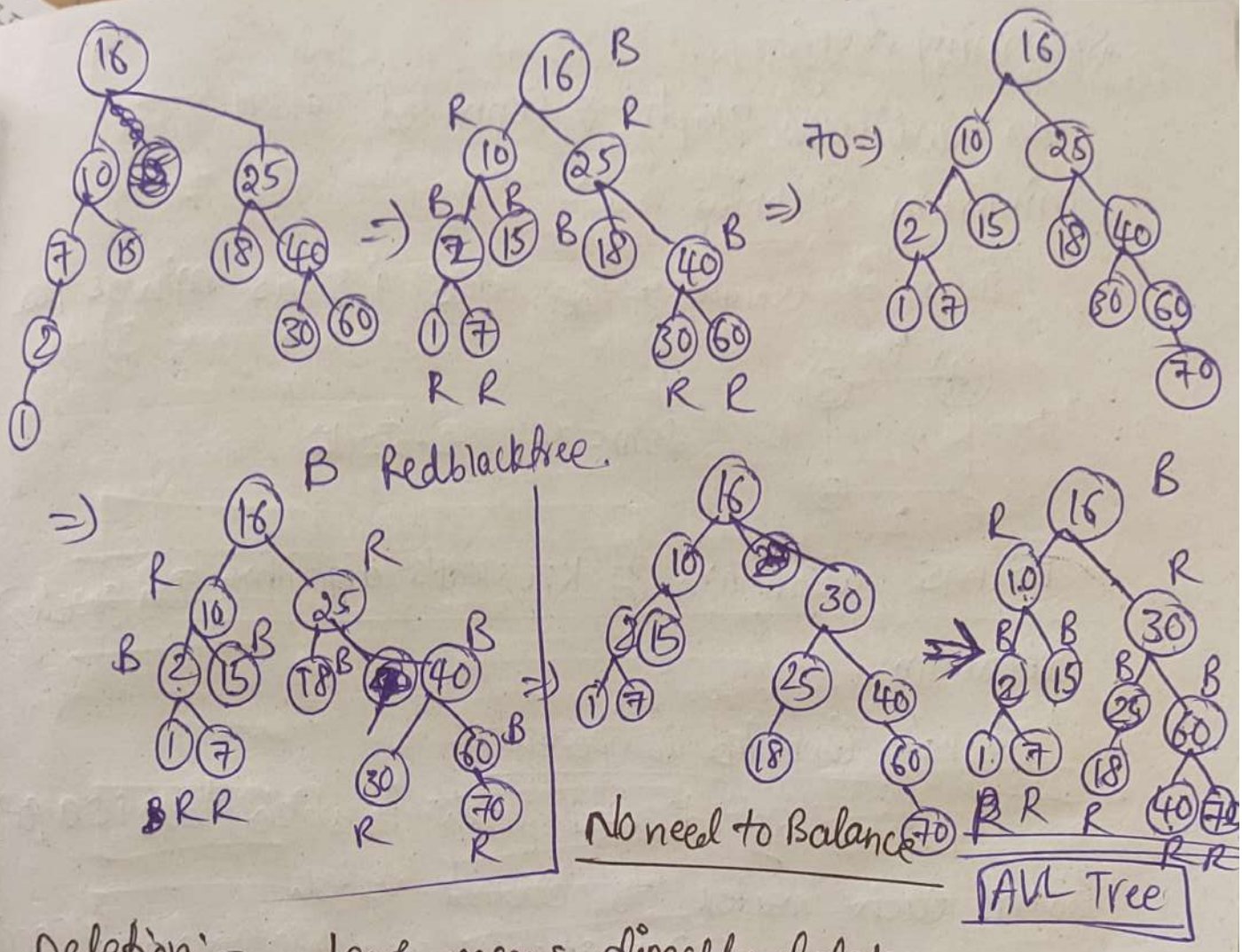
Red Black tree:-

Red black; every node is colored, either red/black. Root is black.; every leaf is black; red children are always black;

Insertion:- Jenny.youtube.com.

10, 18, 7, 15, 16, 30, 25, 40, 60, 21, 70.

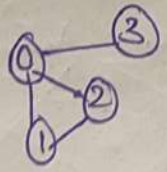




Deletion:- Leaf means directly delete.
Internal node delete and rearrange.

Graph data structure:-

A graph data structure is a collection of nodes that have data and are connected to other nodes.



V(~~vertex~~ values) = {0, 1, 2, 3}.

E = {(0,1) (0,2) (0,3), ~~(3,0) (2,1)~~ (1,2)}.

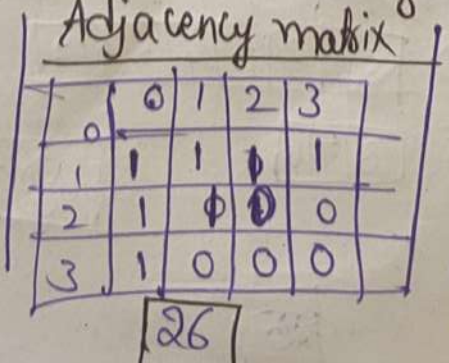
G = {V, E}

Adjacency:-

Path;

Directed graph:-

Sol:-

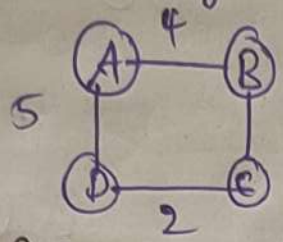


Spanning tree:-

An undirected graph & connected graph.

Minimum Spanning tree:-

sum of weight of the edges is as min as possible.

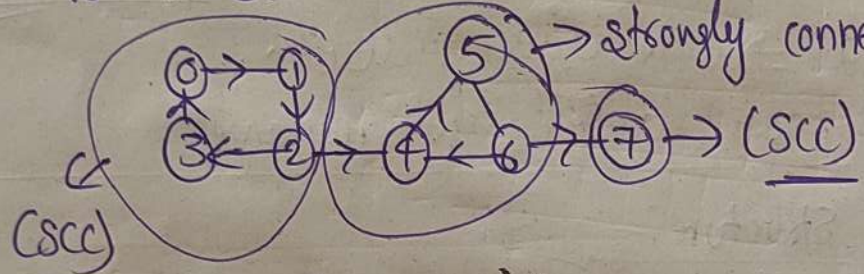


Sum = 4+1+2+5 = 12

Prims algorithm & Kruskal's algorithm in greedy algorithm.

Strongly Connected Components:-

portion of directed graph in which there is a path from each vertex to another vertex.



Kosaraju's algorithm:-

visited ⇒ | 0 | 1 | 2 | 3 |

stack ⇒ | 3 |

After 3 it is 0; 0 is already visited.

Sequentially

visited ⇒ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

stack ⇒ | 3 | 7 |

visited ⇒ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Stack ⇒ | 3 | 7 | 6 |

youtube.com

27

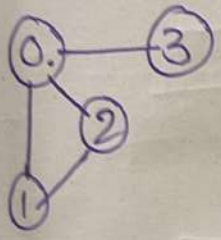
Adjacency Matrix:- Boolean [0 & 1]

If there is a path then (1) else (0)

Adjacency List:-

Array of linked list; Path from start to End.

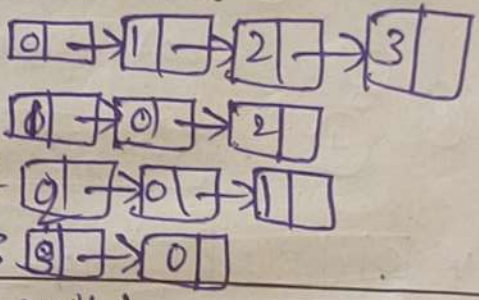
Adjacency tree Example:-



Adj. matrix

	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

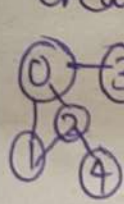
Adjacency list



DFS Algorithm:- (Stack). (depth ↓ stack.)

visited or not visited.

1) Start by putting any one of the graph's vertices on the top of stack.
2) Take top item & add as visited list.
3) Create a list of which not visited. 4) Repeat 2,3.



visited
stack

visited [0]
stack [1 2 3]

visited [0 1]
stack [2 3]

visited [0 1 2]
stack [3]

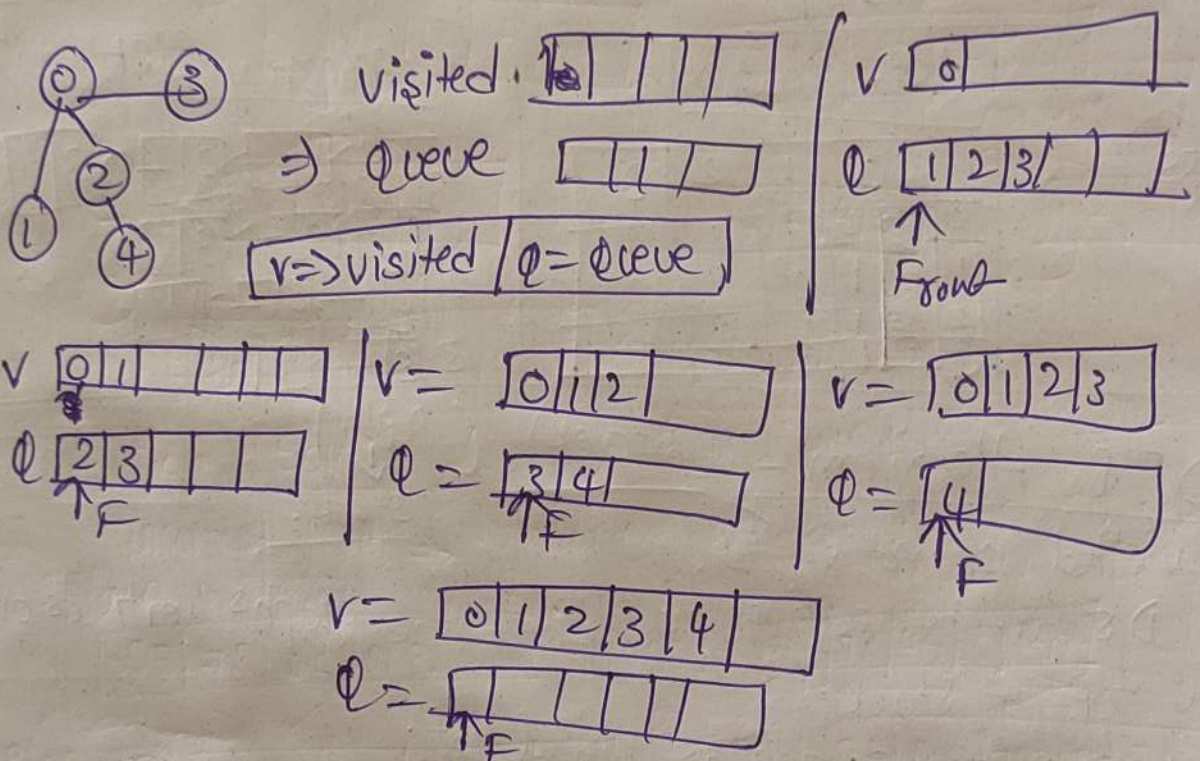
visited [0 1 2 3]
stack [4]

visited [0 1 2 3 4]
stack

Breadth first search (BFS):- (Queue) (Breadth ⟷ Queue)

Recursive algorithm for searching all vertices of a graph visited or Not visited:-

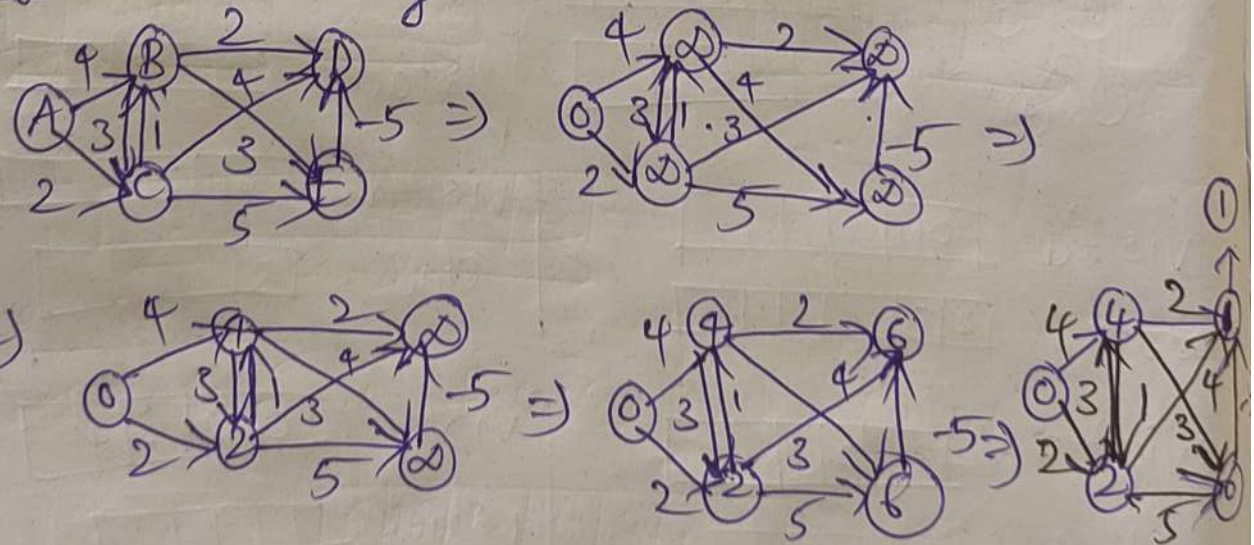
1) one value into queue 2) front add to queue 3) Adjacent list add to queue 4) Repeat 2,3.

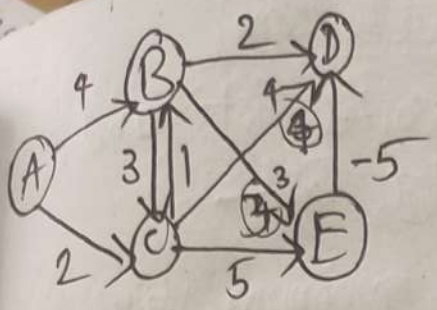


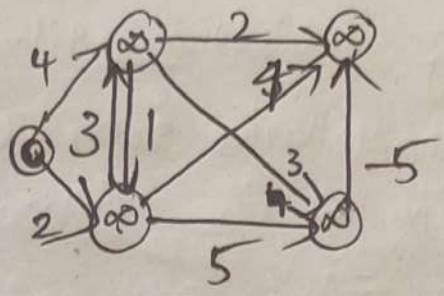
Bellman Ford's Algorithm:-

Negative weight:-

Ex:-







Bellman ford Alg
Adjacency matrix

	B	C	D	E
0	∞	∞	∞	∞
0	4	2	6	∞
0	3	2	5	6
0	3	2	1	6
0	3	2	1	6

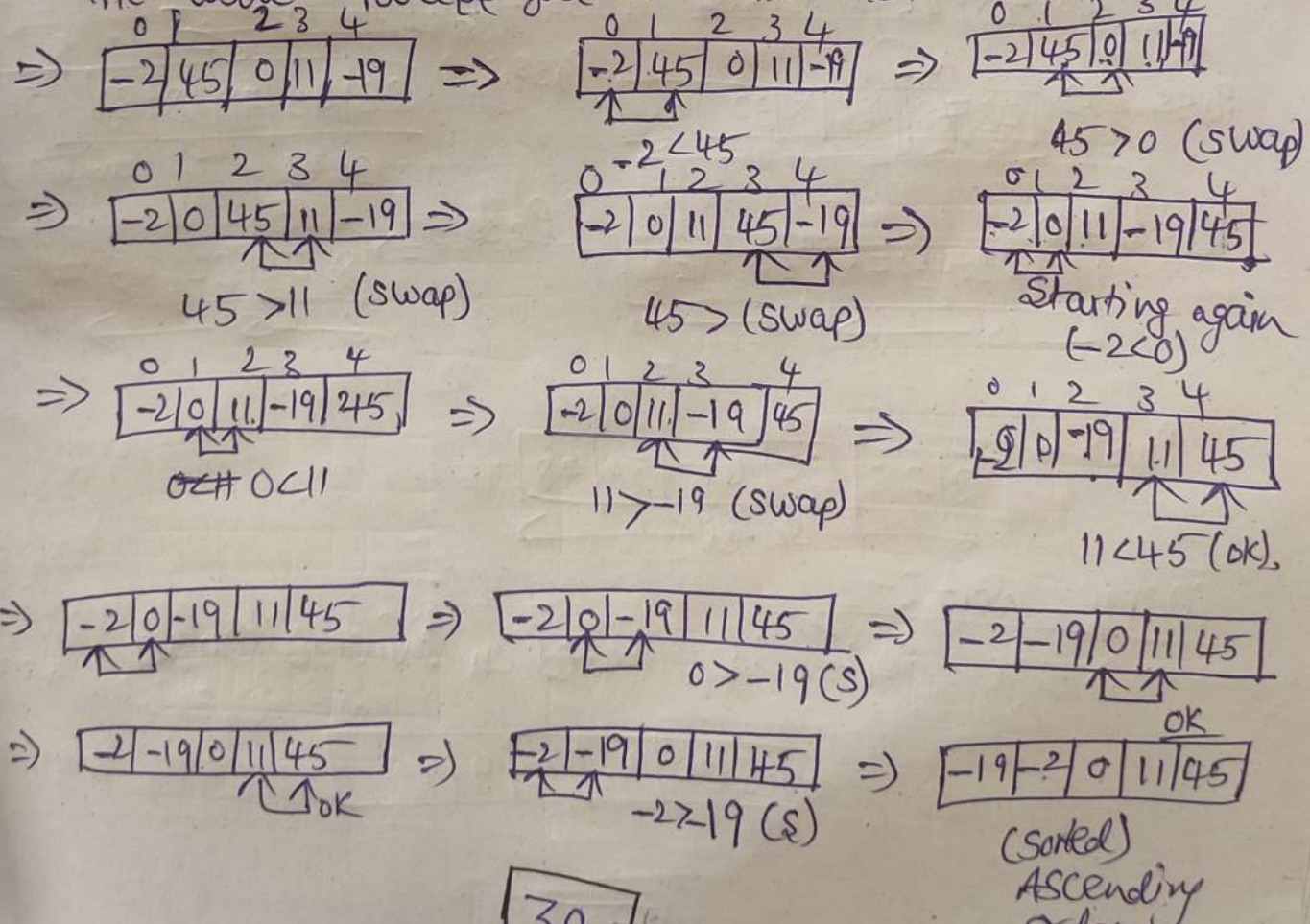
Sorting and Searching algorithms:-

Bubble Sort:- ooo (air bubbles in water).
oo

compares two adjacent elements and swaps them until they are in the intended order.

method:-

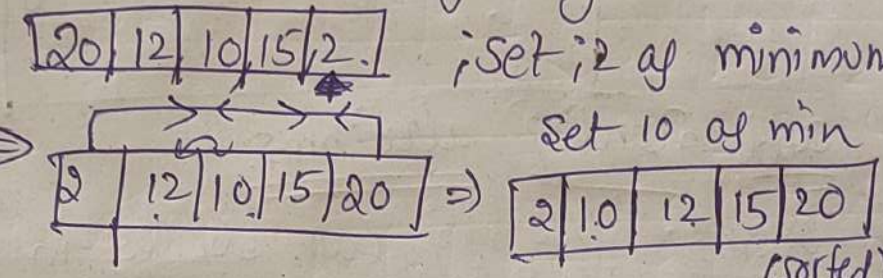
1) Start with index, compare 1 & 2 elements
2) if first element > Second element swap
3) Now, compare 3,4 element, swap if 3(v) > 4(v).
4) The above process goes on until the last element.



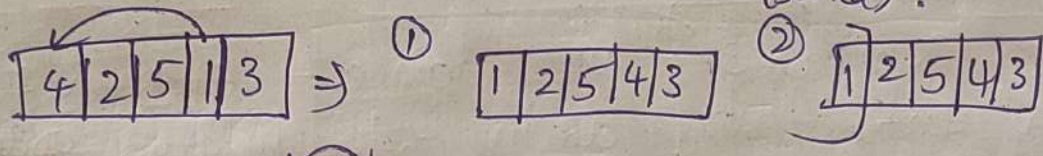
Selection Sort:- (Education4u.youtube.com)

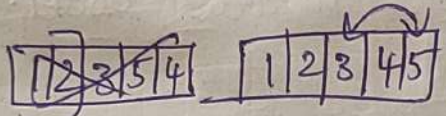
Select small value from unsorted list and places at the begining of the unsorted list

→ we can select ~~any~~ only minimum value.

Eg:- 

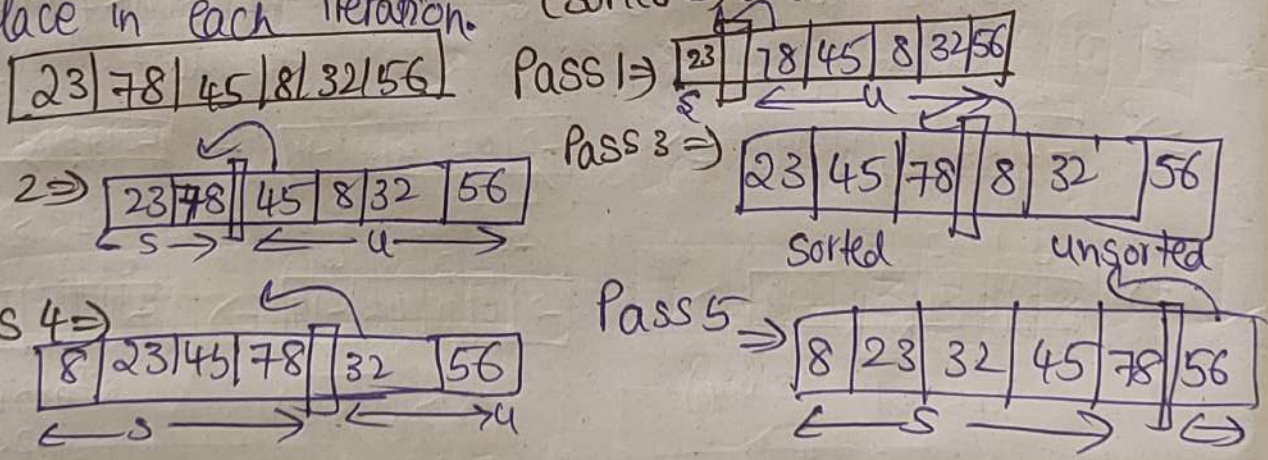
; set i2 as minimum place at beg

Eg:- 

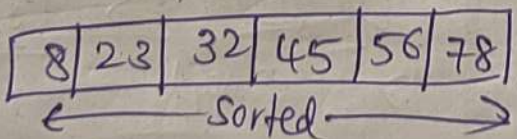
③ 

Insertion Sort:- (cards) Education4u.youtube.com :-

Insertion sort ⇒ unsorted element at its suitable Place in each iteration. (sorted □) & (unsorted) Passes

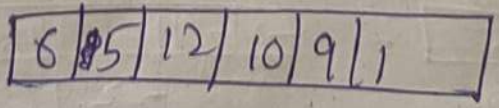


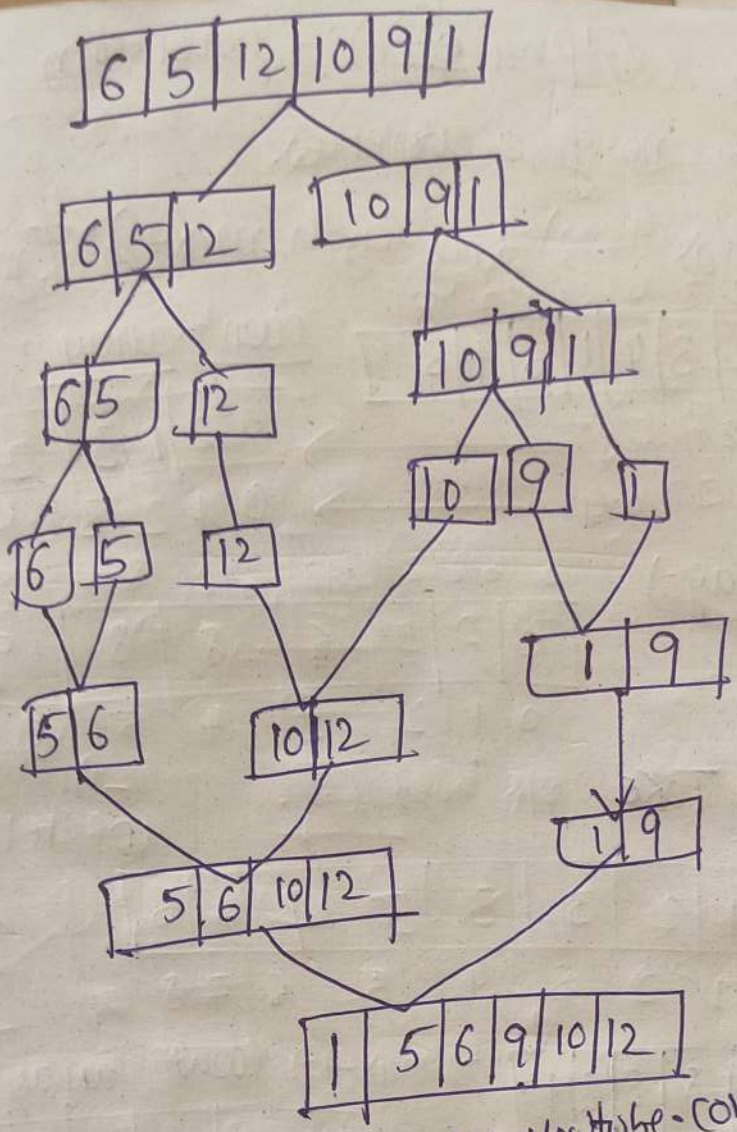
Pass 6



Merge sort:-

Merge sort is based on divide & conquer.





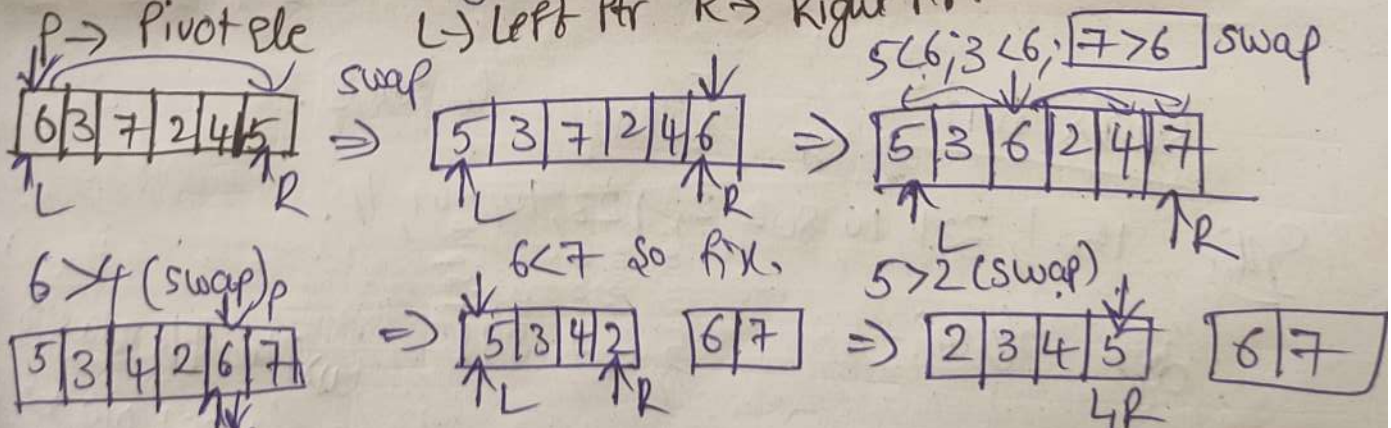
Quick sort:- (Education4u.youtube.com)

An array is divided into subarrays by selecting a pivot element (PE ; less/PE/great values).

1) Select the Pivot Element 2) Rearrange the array (values < PE)
3) Divide and rearrange 4) Sorted.

*LV < PE < RV *

P → Pivot ele L → Left Ptr R → Right Ptr.



(6) No swap possible

L and R are at same index when there is no swap

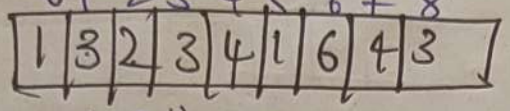
L&R move (on) comparision. 32

Counting sort (Apna college): - Youtube.com

Counting no. of occurrences.

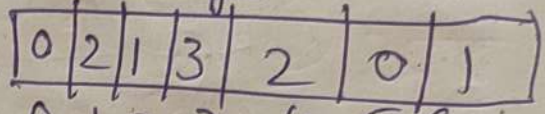
1) Find out max 2) Array length (max+1) [max no. => 6 6+1 (index)

orig :-

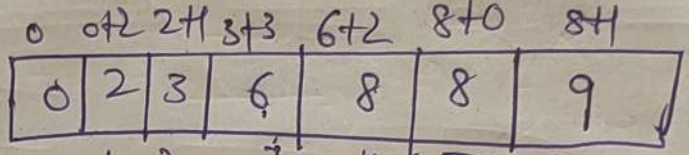


1 is 2 times
3 is 3 times

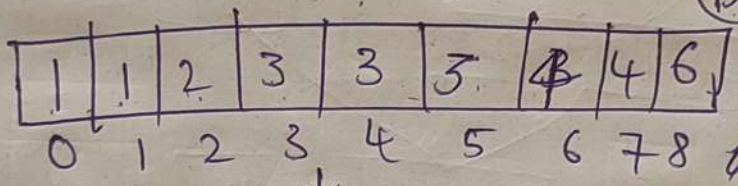
count array



sum array =>

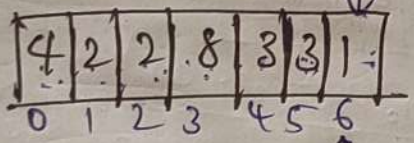


Final o/p => Take o/s index (8). (-1) (Right to left of o/s)
(decrement)

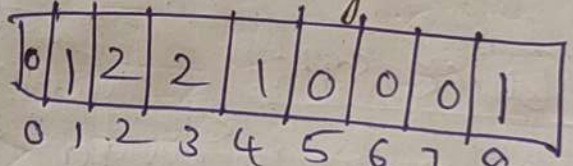


2 => index (3)
index (3) V = 6

2) 8x1

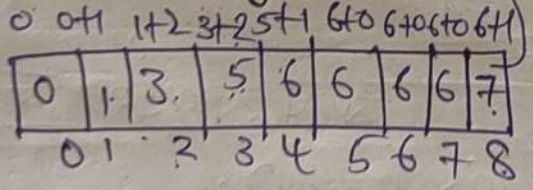


=> max+1 => count array.

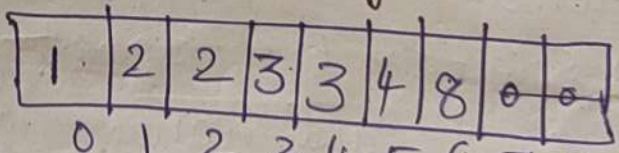


7 elements

Sum array



Decrement array: -



Radix sort :- (Jenny's lectures).

Eg :- 15, 1, 321, 10, 802, 2, 123, 90, 109, 11.

digit place :-

1000's place, 100's place, 10's place, 1's place

5, 6, 7, 8, 9, 1

the (max no.) write o/s values to

Pass 1 :- 1st place

Pass 2 :- 10th place

Pass 3 :- 100th place

015, 001, 321, 010, 802, 002
123, 090, 109, 011

33

(1's)

Pass 1

0 - 010, 090,
1 - 001, 321, 011
2 - 802, 002
3 - 123
4 - ----
5 - 015,
6 - ---
7 - ---
8 - --
9 - 109

Pass 2 (10's)

0 - 001, 802, 002, 109
1 - 010, 011, 015.
2 - 321, 123.
3 - ---
4 - ---
5 - ---
6 - ---
7 - ---
8 - ---
9 - 090

Pass 3 (100's)

0 - 001, 002, 010, 011, 015, 090.
1 - 109, 123
2 - ---
3 - 321
4 - ---
5 - ---
6 -
7 -
8 - 802
9 -

(add 0's)

Pass 0 ⇒ 015, 001, 321, 010, 802, 002, 123, 090, 109, 011.

Pass 1: ⇒ 010, 090, 001, 321, 011, 802, 002, 123, 015, 109.

Pass 2) ⇒ 001, 802, 002, 109, 010, 011, 015, 321, 123, 090

Pass 3 :- 001, 002, 010, 011, 015, 090, 109, 123, 321, 802.

Bucket Sort)- (Easy way. youtube.com):

0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.57.

multiply X10 ⇒ 4.2, 3.2, 2.3, 5.2, 2.5, 4.7, 5.7 — Not required.

0 → . ---
1 → . ---
2 → 23 → 0.25
3 → 0.32
4 → 0.42 → 0.47
5 → 0.52 → 0.57
6 →
7 →
8 →
9 →

(Ascending the digits place).

0.23	0.25	0.32	0.42	0.47	0.52	→ 0.57

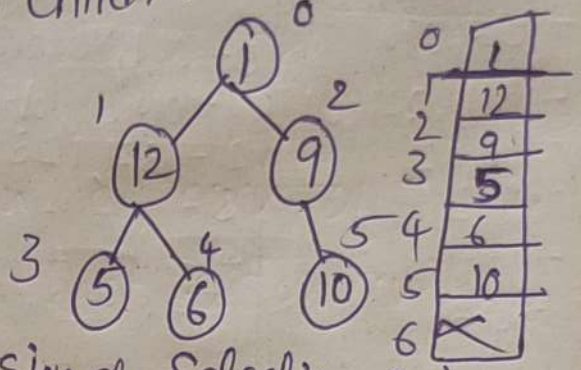
Heap Sort:- (MIN/MAX) (Education 4U.youtobe.com)

MIN ⇒ Parent < child MAX ⇒ child > Parent

Ex:-

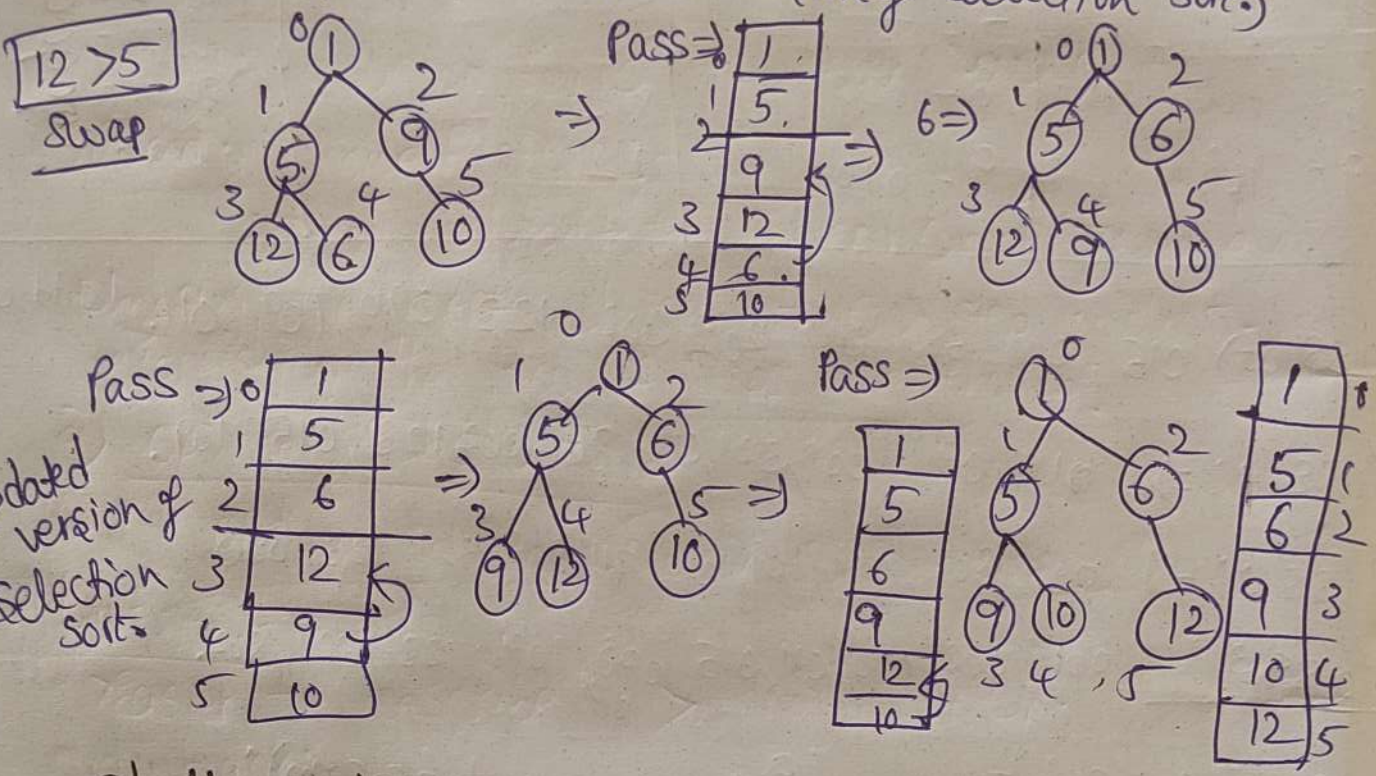
0	1	2	3	4	5
1	12	9	5	6	10

⇒



MIN Heap Sort:- (updated version of Selection sort.)

12 > 5 Swap



Shell Sort:- (Insertion sort algorithm)

77, 62, 14, 9, 30, 21, 80, 25, 70, 55.

(Education 4U.yt.com)

N = 10 :- gap = (floor $\frac{N}{2}$) = $\frac{10}{2}$ = 5;

Pass 1 ⇒

0	1	2	3	4	5	6	7	8	9
77	62	14	9	30	21	80	25	70	55

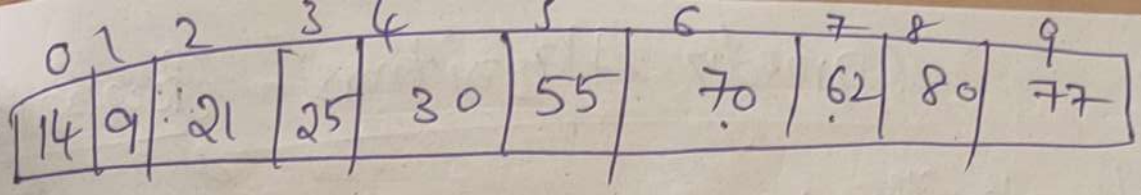
Pass 2 ⇒ (compare) all

0	1	2	3	4	5	6	7	8	9
21	62	14	9	30	77	80	25	70	55

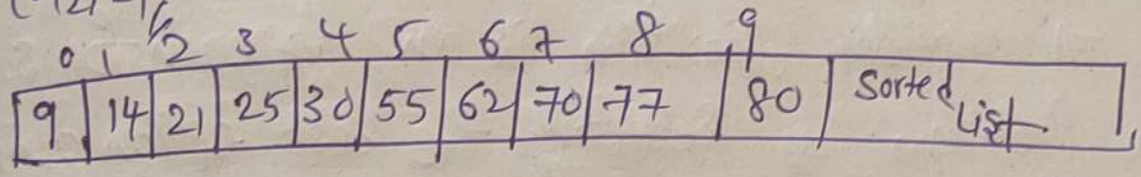
gap = $\frac{5}{2}$ = 2 ⇒ 21 62 14 9 30 77 80 25 70 55

0	1	2	3	4	5	6	7	8	9
14	21	30	70	80	9	25	55	62	77

(gap)



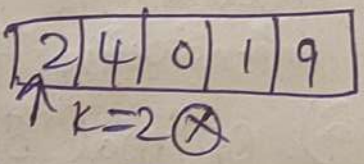
gap(2/2) = 1/2

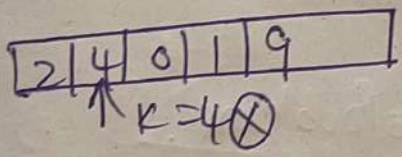


Linear Search:-

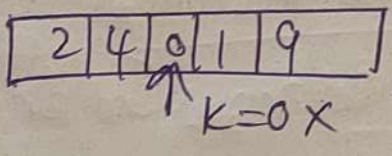
Sequential searching alg start from one end and check every element of the list until the desired element is found.

Ex:- 2 4 0 1 9 find k=1;

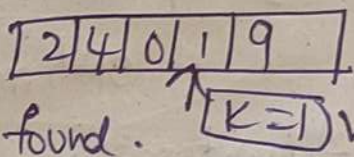




;



;



found.

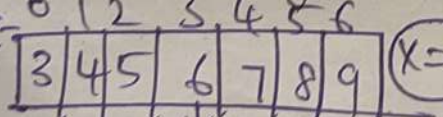
Binary Search:-

Binary search is a searching alg; for finding an elements position in a sorted array.

1) Iterative method 2) Recursive method.

mid = (low + high)/2; x > mid or x < mid;

Ex:1 :-

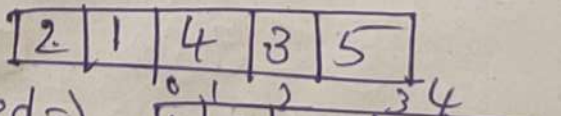


mid = 0+6/2 = 3; 4<6

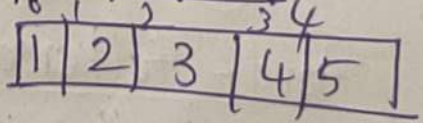
3 4 5 6 ⇒ 3 4 5 (0 1 2)

0+2/2 = 1; x=4 // found.

Ex:2 :-



Sorted ⇒



Find x=3

mid = 0+4/2 = 2;

a[2] = 3 // found.

Greedy Algorithms:- (Programiz.com)

Greedy algorithm:-

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment.

If decision fails its never reverse.

1) Greedy choice Property:-

optimal solution to the problem can be found by choosing the best choice at each step without reconsidering the previous steps once chosen.

2) optimal Substructure:-

optimal solution used for sub problems.

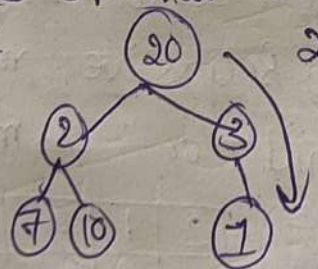
Advantages:-

> Easier to describe > Perform better.

Drawback:-

sometimes optimal solution can't found.

Eg:-



20; 2<3, so right

right path.

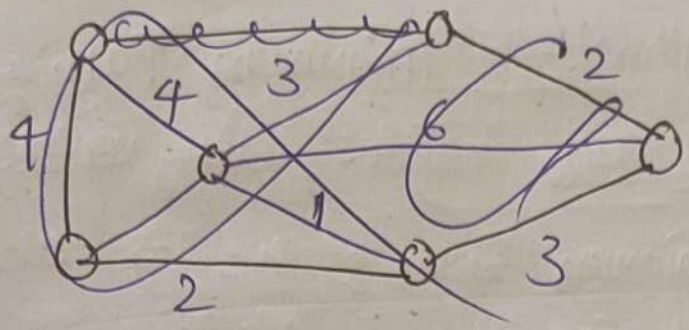
Ford-Folkerson Algorithm:- (Let's try).yt.com

(↳ letstry.youtube.com).

Dijikstra's algorithm:-

Dijikstra's allows us to find the shortest path between any two vertices of a graph.

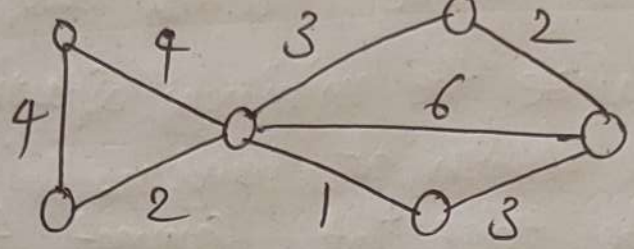
Ex:-

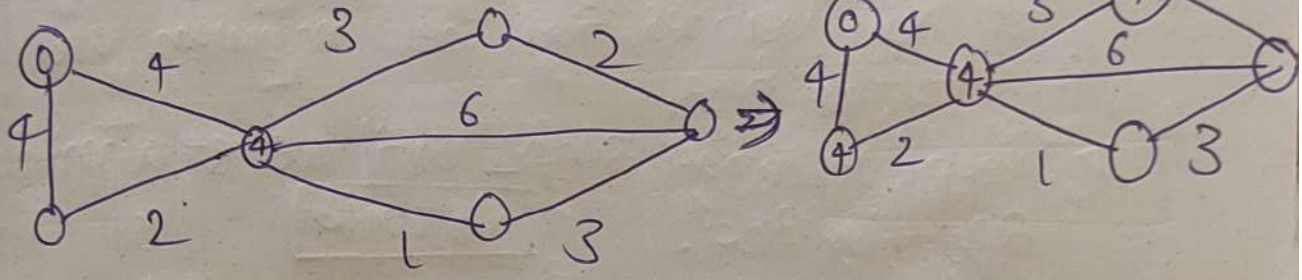


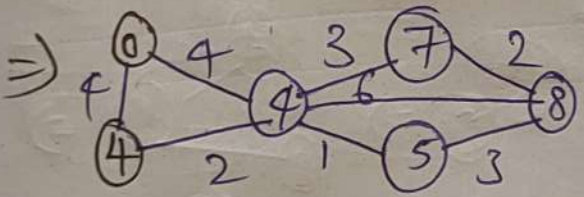
Applications:-

maps; shortest path; social networking; telephone network.

Ex:-





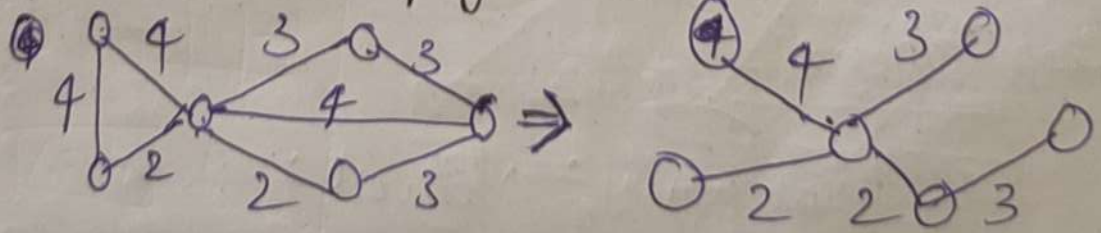


Kruskal's algorithm:-

(minimum spanning tree:-)

- Form a tree that includes every vertex.
- has the minimum sum of weights, among all the trees that can be formed the graph.

* Every vertex has path to all vertex and without loop / cycle. [3,3,4 ⇒ weight ascending]



Applications:- LAN & Electrical wiring.

2;2;3,3;4 (Path).

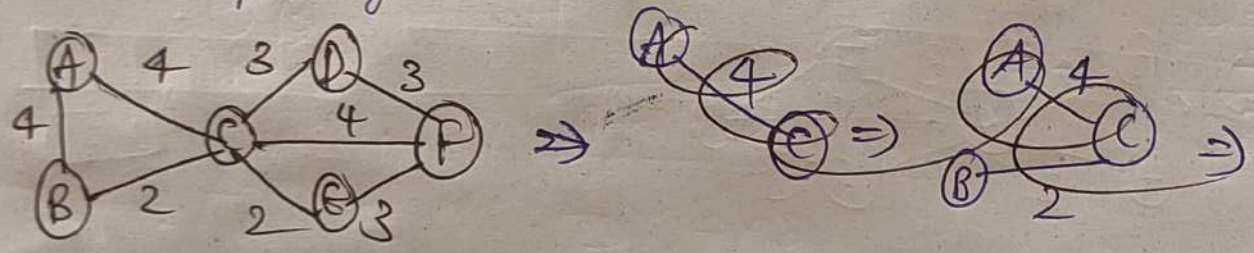
Prims Algorithm:- (minimum spanning tree).

- Form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph.

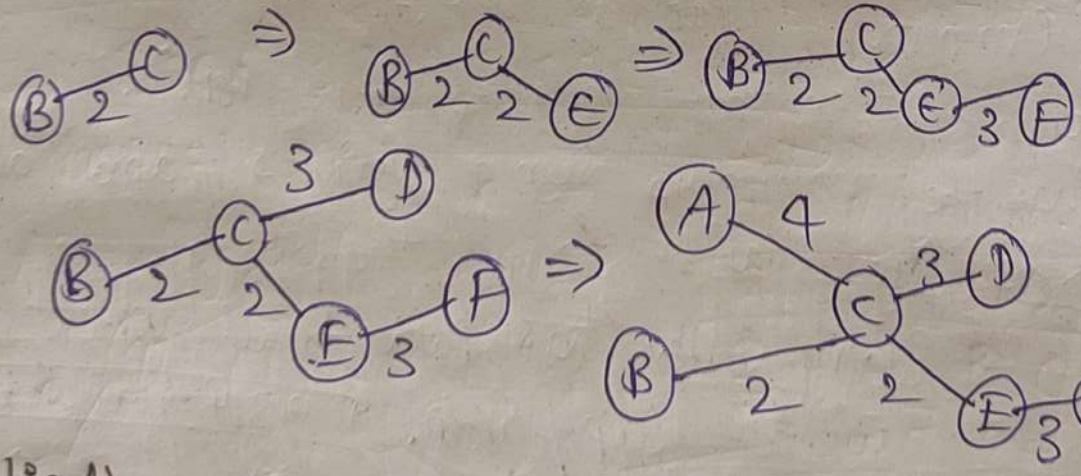
method:- Initialize the minimum spanning tree with a vertex chosen at random.

> Find all the edge that connect the tree to new vertices, find the minimum and add it to the tree.

> keep repeating step 2 until we get a minimum spanning tree. **No cycle**



⇒ weight ascending order



Applications:-

electrical wiring; Network;
N/w. Protocols

Huffmans Coding:-

Compressing data to reduce its size without losing any of the details.

→ mainly used to Compress the data. of frequently occuring.

0→14.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B	C	A	A	D	D	D	C	C	A	C	A	C	A	C

(15)

Each character occupies 8 bits; 8*15 = 120 bits.

Encoding and Decoding:-

Huffmans coding prevents ambiguity → duplicate values → Repeated ||

1. Calculate the frequency of each character in the string.

A	B	C	D
1	6	5	3

2) Sort the characters in increasing order of frequency.

B	D	A	C
1	3	5	6

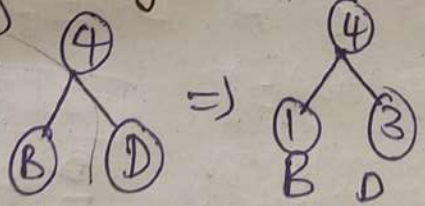
3) make each unique character as a leaf node

left < Z < Right

4) create an empty node Z

Z = 4 (1+3)

*	A	C
4	5	6



5) Remove these two minimum frequencies from Q and add the sum into the list of frequencies:-

1	3	5	6

4+5

*		
4	5	6

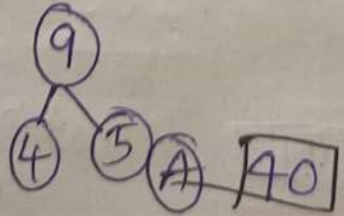
4+5 = 9

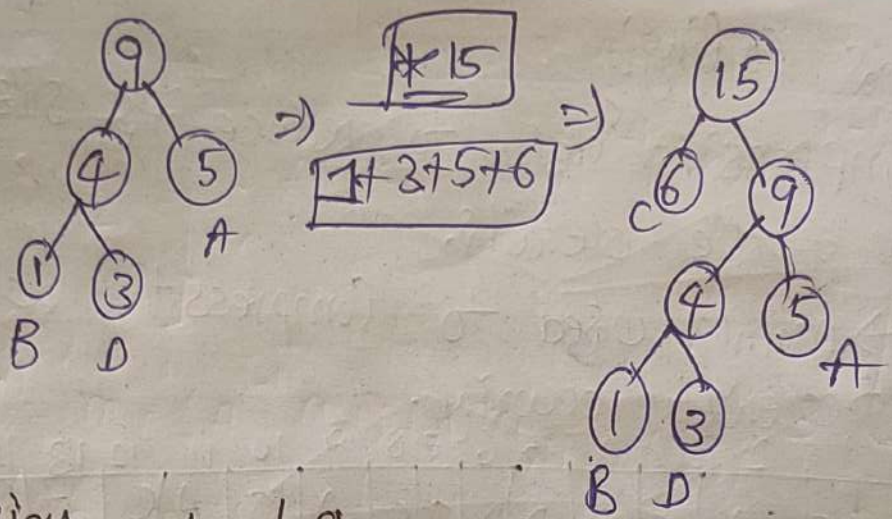
*	
6	9

9>6

6) insert Z = 9 of tree. 7) Repeat steps to 3 to 5

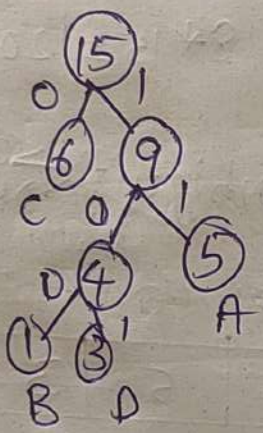
*	
6	9





assign 0 to left node & 1 to right node.

Path weight from root to character



Character	Frequency	Code	Size
A	5	11	5*2=10
B	1	100	1*3=3
C	6	0	6*1=6
D	3	101	3*3=9
4*8=32			28

Applications:- Zip files.

Huffman Coding:- (Easy and Perfect example). Abdul Bari. youtube. com

Q1s ⇒ BCCABB DDA ECC BB A E DCC.

Pass 0 ⇒

B	C	A	D	E
5	6	3	4	2

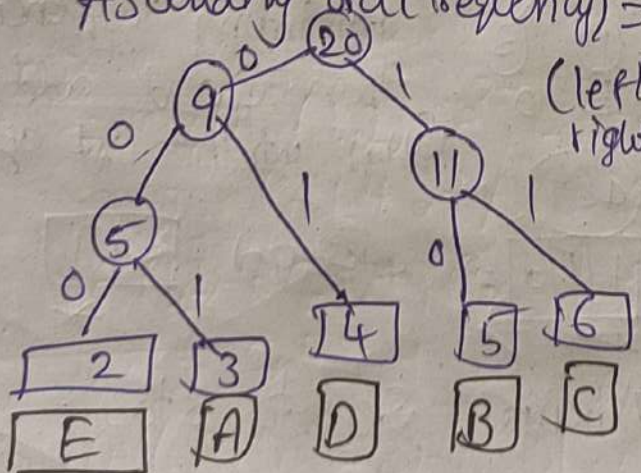
→ frequency

→ Q1s order.

Ascending ord(frequency) ⇒

E	A	D	B	C
2	3	4	5	6

(left 0, right 1)



Char	Frequency	Code	Size
E	2	000	2×3 = 6
A	3	001	3×3 = 9
D	4	01 ⇒	4×2 = 8
B	5	10 ⇒	5×2 = 10
C	6	11 ⇒	6×2 = 12

Code ⇒ path weight of root to char

Dynamic Programming:- (Programiz.com/dsa)

> Solve a class of Problems that have overlapping subproblems and optimal Substructure Property.

Ex:- Fibanaci Series > 5th term > 0,1,1,2,3.
Here computer need to store Previous value,

Working:-

> storing the result of subproblems so that when their solutions are required.

The technique of storing the value of subproblems is called memorization.

Var m = map (0→0, 1→1)

⇒ Allshort.

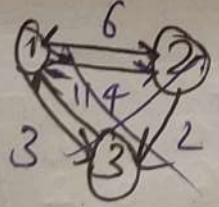
Floyd-Warshall Algorithm:- [Education4u.yt.com]

Finding the shortest path b/w all the pairs of vertices in a weighted graph. Works for Both (weighted Directed & Non Directed) Not for Negative weight.

General formula:-

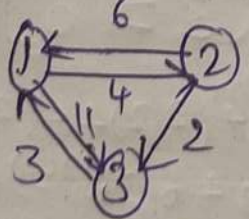
$$\min_{1 \le k \le n} \{A^{k-1}(i,k) + A^{k-1}(k,j)\}, c(i,j)\}$$

Ex:-



⇒

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 6 & 3 \\ \cdot & 0 & \cdot \\ \cdot & \cdot & 0 \end{bmatrix} \end{array}$$



$$\Rightarrow A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

let; $k=1; (i,j) \Rightarrow$

$$\min_{1 \le k \le n} \{A^{k-1}(i,k) + A^{k-1}(k,j); c(i,j)\}$$

(k,j)

K is 1 because we need to take A^0.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

$A^{(i,j)} = \min\{A^{k-1}(i,k) + A^{k-1}(k,j), c(i,j)\}$

(k)

$$A^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$A^{(1,1)} = \min\{A^{1-1}(1,1) + A^{1-1}(1,1), c(1,1)\}$

$A^{(1,1)} = \min\{0, 0\} = 0$

$A^{(1,2)} = \min(A^{1-1}(1,1) + A^{1-1}(1,2), c(1,2))$

$= \min(0+4, 4) = 4$

$A^{(1,3)} = \min\{A^0(1,1) + A^0(1,3), c(1,3)\}$

$= \min(0+11, 11) = 11$

$A^{(2,3)} = \min(A^0(2,1) + A^0(1,3), c(2,3))$

$= \min(6+11, 2) = 2$

$A^{(2,1)} = \min\{A^0(2,1) + A^0(1,1) + c(2,1)\}$

$= \min\{6+0, 6\} = 6$

$A^{(2,2)} = \min(A^0(2,1) + A^0(1,2), c(2,2))$

$= \min(6+4, 0) = 0$

$A^{(3,1)} \Rightarrow \min\{A^0(3,1) + A^0(1,1), c(3,1)\}$

$\Rightarrow \min(3+0, 3) \Rightarrow 3$

$A^{(3,2)} = \min(A^0(3,1) + A^0(1,2), c(3,2))$

$A^{(3,2)} = \min(3+4, \infty) \Rightarrow 7$

$A^{(3,3)} = \min\{A^0(3,1) + A^0(1,3), c(3,3)\} \Rightarrow (3+11, 0) \Rightarrow 0$

(i,j) (k=2)

$A^{(1,1)} = \min(A^{2-1}(1,2) + A^{2-1}(2,1), c(1,1))$

$= \min(A^1(4) + A^1(6), c(0) = 0$

$$A^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$A^{(1,2)} \Rightarrow \min(A^1(1,2) + A^1(2,2), c(1,2))$

$\Rightarrow \min(4+0, 4) = 4$

$A^{(1,3)} \Rightarrow \min(A^1(1,2) + A^1(2,3) + c(1,3))$

$\Rightarrow \min(4+2, 11) = 6$

$A^1(2,1) = \min(A^1(2,2) + A^1(2,1) + c(2,1))$

$= \min(0+6, 6) \Rightarrow 6$

$A^1(2,2) = 0$

$A^1(2,3) = \min(A^1(2,2) + A^1(2,3) + c(2,3))$

$= (0+2, 2) \Rightarrow 2$

$A^1(3,1) = \min\{A^1(3,2) + A^1(2,1), C(3,1)\}$
$= 7 + 6; 3 \Rightarrow 3$

$A^1(3,2) \Rightarrow A^1(3,2) + A^1(2,2); C(3,2)$
$\Rightarrow 7; 7 \Rightarrow 7$

$A^3(1,1)$ $\quad k=3$

$A^3 \Rightarrow \min\{A^{k-1}(i,k) + A^{k-1}(k,j); c(i,j)\}$

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

$\Rightarrow$

$A^3(1,2) \Rightarrow A^2(1,3) + A^2(3,2) + C(1,2)$
$\Rightarrow 6 + 7; 4 \Rightarrow 4$

$A^3(1,3) \Rightarrow \min(A^2(1,3) + A^2(3,3) + C(1,3)$
$\Rightarrow [6+0; 6] \Rightarrow 6$

$A^3(2,1) \Rightarrow A^2(2,3) + A^2(3,1) + C(2,1)$
$\Rightarrow 2 + 3; 6 \Rightarrow 5$

$A^3(2,3) \Rightarrow A^2(2,3) + A^2(3,3) + C(2,3)$
$\Rightarrow 2 + 0; 2 \Rightarrow 2$

$A^3(3,1) \Rightarrow \min(A^2(3,3) + A^2(3,1); C(3,1)$
$\Rightarrow 0 + 3; 3 \Rightarrow 3$

$A^3(3,2) \Rightarrow \min(A^2(3,3) + A^2(3,2); C(3,2)$
$\Rightarrow 0 + 7; 7 \Rightarrow 7$

Final $\Rightarrow A^3 = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 6 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{matrix}$ ✓

"Longest Common Sequence:-" Programiz.com/DSA

LCS :- longest subsequence i.e common to all the given sequences; provided that the elements of the subsequence are not required to occupy consecutive positions.

Ex:- X = a b a a b a Y = b a b b a b

(1)

X \ Y		b	a	b	b	a	b
	0	0	0	0	0	0	0
a	0	0	1	1	1	1	1
b	0	1	1	2	2	2	2
a	0	1	2	2	2	3	3
a	0	1	2	2	2	3	3
b	0	1	2	3	3	3	4
a	0	1	2	3	3	4	4

Jenny's lectures youtube.com

If match (a,a) = Add otherwise (max value). Arrow is on max value. if same any side.

From last of table backward diagonal is the longest common seq = b a b b a 44

logic:- if matching add diagonal top value else max of left & right value.

Backtracking algorithm:- Programiz.com;

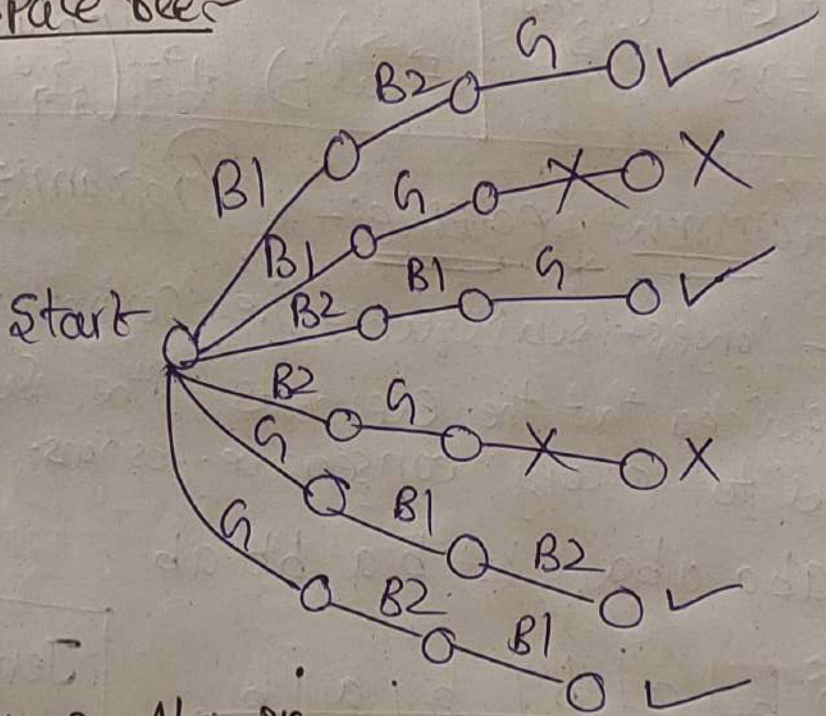
From All possible Solutions choose desired / best Solutions.

Ex:- You want to find all the possible ways of arranging 2 boys & 1 girl on 3 benches. Girl should not be on the middle bench.

Sol:- There are a total of 3! = 6 Posssibilities.

All possibilities [B1 | B2 | G] [B1 | G | B2] [B2 | B1 | G] [B2 | G | B1] [G | B1 | B2] [G | B2 | B1]

State space tree:-



Rabin Karp Algorithm:- Programiz.com

used for searching/matching Patterns in text using a hash function.

Working:-

A sequence of characters is taken and checked for the possibility of the presence of the presence of the required string. 45 If the possibility is found then char match performed

Ex:- abcdabce (Abdul Bari.youtube.com).

Assign:
a - 1
b - 2
c - 3
d - 4
e - 5

Two methods

Sum:- /abcdabce/

Find bcd ⇒ bcd ⇒ 2+3+4 = 9 //

⇒ abcdabce ⇒ abcdabce ⇒ abcdabce

123 ⇒ 6 ✗ (1+2+3)

234 ⇒ 9 ✓ (2+3+4)

341 ⇒ 8 (3+4+1)

and so on ⇒ Until found a string

Multiplication:-

abcdabce and find cda

cda ⇒ 341

abcdabce

$1\times10^2+2\times10^1+3\times10^0 = 100+20+3 = 123$ ⊗

abcdabce

$2\times10^2+3\times10^1+4\times10^0 = 200+30+4 = 234$ ⊗

abcdabce ⇒ $3\times10^2+4\times10^1+1\times10^0 = 300+40+1 = 341$

found:-