DSA UNIT-3

Topics =
- Graphs
  - Prim's Algorithm
  - Kruskal's Algorithm
  - Dijkstra Algorithm
- Trees (Inorder, Preorder, Postorder)
  - Binary Search Tree
  - Full Binary Tree
  - Almost Complete Binary Search Tree (ACBT)
  - Complete Binary Search Tree
- BFS (Breadth first Search)
- DFS (Depth first Search)
- AVL Trees
- Heap (Max Heap And Min Heap)
- B - Tree
- Hashing
  - Chaining (Open hashing)
  - Open Addressing
    - Linear Probing
    - Quadratic Probing
    - Double Probing

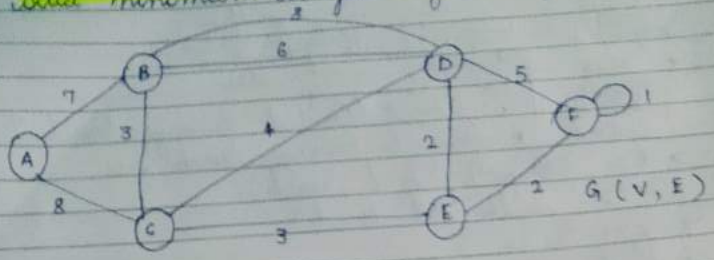
* Graphs = In Graphs each vertices (Nodes) are connected to each other by the help of adjacent vertices edges.
- Non-Linear data structure
- To find Minimum Spanning Tree, there are three methods =
  1. (i) Prim's Algorithm
  2. (ii) Kruskal's Algorithm
  3. (iii) Dijkstra Algorithm

(i) PRIM'S ALGORITHM = Time complexity = $O(V^2)$ where, v is the vertex (Greedy Algo) $O((V+E)\log V)$ because vertex is Inserted in priority Queue only Once. Insertion in the vertex priority Queue takes logarithmic time.

Steps =
1. Remove the looping node.

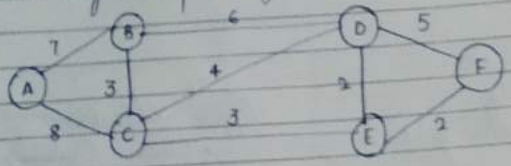
→ We choose any arbitrary node.
→ Used in dense graphs.
→ Condition = No, closed loop should be made.
2. Select any arbitrary node as root node.
3. Select minimum weight edge comparable.

Q.

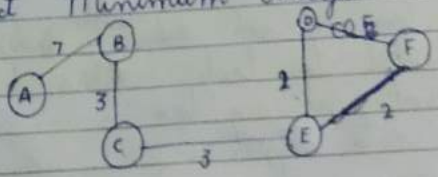


G(V, E)

→ Removing loop Edge



Let's assume A as arbitrary Edge
Select Minimum weight



G(V', E')

$V = V'$

$E' = |V| - 1$

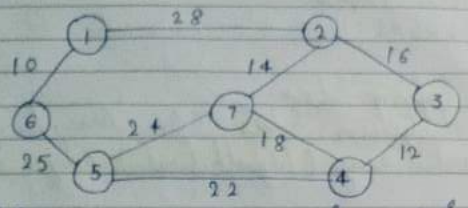
$(7 + 3 + 3 + 2 + 2) = 17$

$\because V = 6, \quad E' = (6 - 1) = 5$

Note = $E' = |V| - 1$

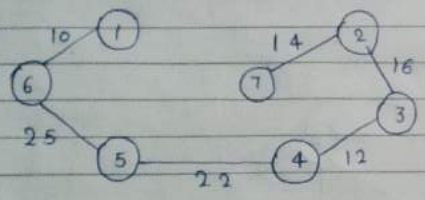
$V = V'$

Q1.



G (V, E)

→ Let's say, 6 as the Arbitrary node



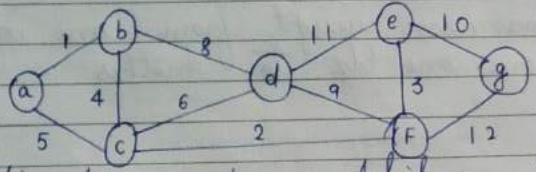
G' (V', E')

Minimum Spanning Tree = $V = V'$

(10 + 25 + 22 + 12 + 16 + 14) $E' = |V| - 1$ → (7 − 1) = E'

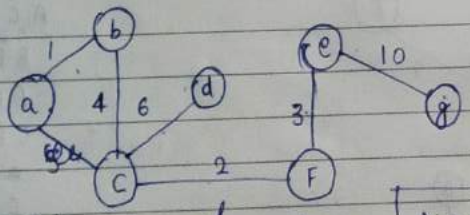
→ 99 ∴ → E' = 6 ∴

Q2.



G (V, E)

Let's Assume b as Arbitrary Node



G' (V', E')

→ Minimum Spanning Tree = (1 + 4 + 6 + 2 + 3 + 10) $V = V'$ $E' = |V| - 1$ → (7 − 1) = E'

→ E' = 6

26 Units

∴

- Choose least weighted node (vertices).
- Used in sparse Graphs.

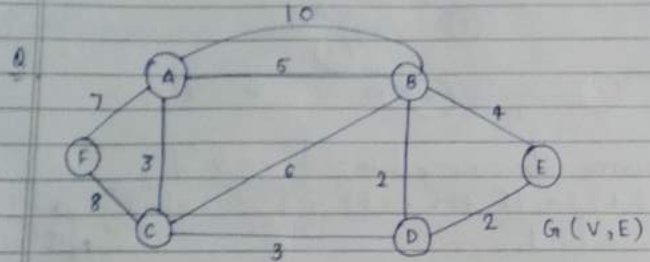
(ii) KRUSKAL'S ALGORITHM = O(E log V)

Steps =

- Remove the looping edges.
- Take weighted distinct edges. (In Ascending Order)
- Take, one arbitrary (hypothetical) edge and compare.

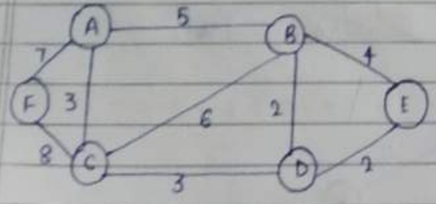
Condition = The edges should never make closed loop.

Q.



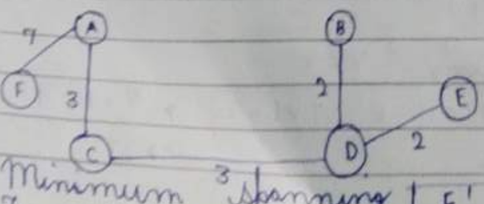
G (V, E)

Remove Maximum weight from one of two weight from one edge to another.



~~AB = 5~~
~~AF = 7~~

BD = 2
DE = 2
AC = 3
CD = 3
BE = 4
AB = 5
CB = 6
AF = 7
CF = 8



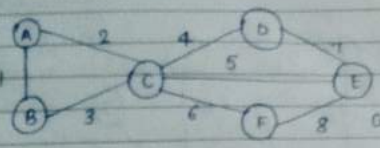
G (V', E')

V' = V

- Minimum Spanning Tree = (7 + 3 + 3 + 2 + 2) = 17

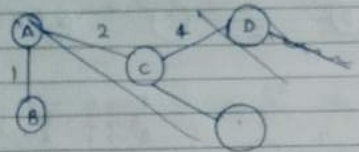
E' = |V| - 1 = (6 - 1) = 5

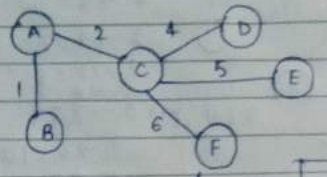
Q.



G(V, E)

AB = 1
AC = 2
BC = 3
CD = 4
CE = 5
CF = 6
DE = 7
FE = 8





G(V', E')

⇒ Minimum spanning Tree

$V' = V$

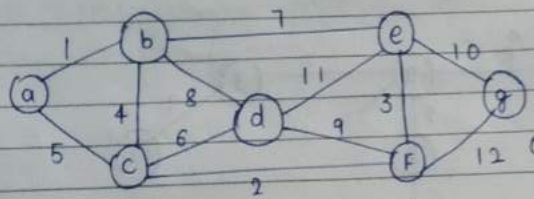
$E' = |V| - 1 = (6 - 1) = E'$

$E' = 5$

$(1 + 2 + 4 + 5 + 6)$

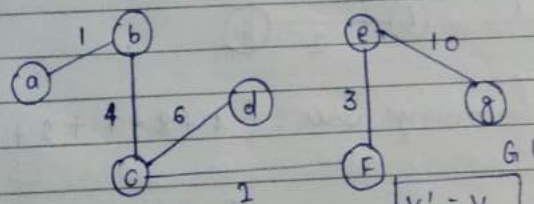
18 ~

Q.



G(V, E)

ab = 1
cF = 2
eF = 3
~~bc = 4~~
ac = 5
cd = 6
be = 7
bd = 8
dF = 9
eg = 10
de = 11
Fg = 12



G(V', E')

$V' = V$

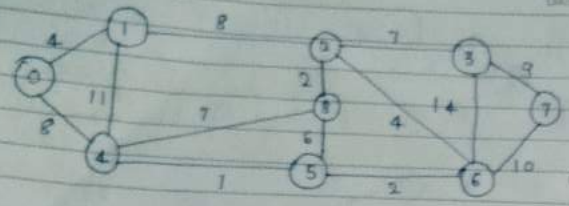
$E' = |V| - 1 = (7 - 1) = 6 = E$

⇒ Minimum spanning Tree

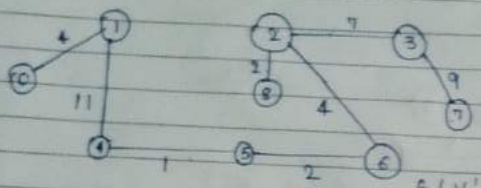
⇒ $(1 + 4 + 6 + 2 + 3 + 10)$

⇒ 26 ~

Q.



G(V, E)



G(V', E')

$4 \to 5 = 1$
$5 \to 6 = 2$, $2 \to 8 = 2$
$0 \to 1 = 4$
$2 \to 6 = 4$
$5 \to 8 = 6$
$4 \to 8 = 7$
$2 \to 3 = 7$
$1 \to 2 = 8$
$8 \to 4 = 8$
$3 \to 7 = 9$
$7 \to 6 = 10$
$1 \to 4 = 11$
$3 \to 6 = 14$

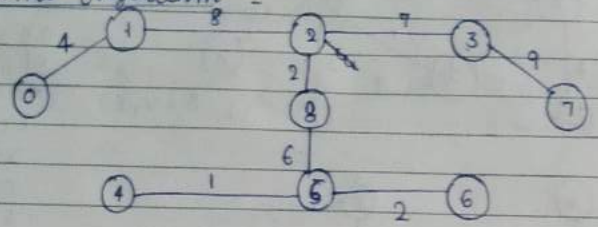
$V' = V$

$E' = |V| - 1$

$E' = 9 - 1 = 8$

⇒ Minimum Spanning tree = $(4 + 11 + 1 + 2 + 2 + 4 + 7 + 9)$
$= 40$

Prim's Algorithm =



⇒ Minimum Spanning Tree = $(4 + 8 + 7 + 2 + 6 + 2 + 14 + 9)$

⇒ 39

Applications = Used to resolve many real-life problems as =

i) Networking (Telephonic networks) in which wired channels will act as Edge and the nodes will be considered as Nodes (Vertices)
ii) Social media networks (Communication) like linkedin, Facebook, Google etc.

Breadth First Search :=
→ Works on concept of Queue.

Queue = linear data structure with the concept of first In first Out

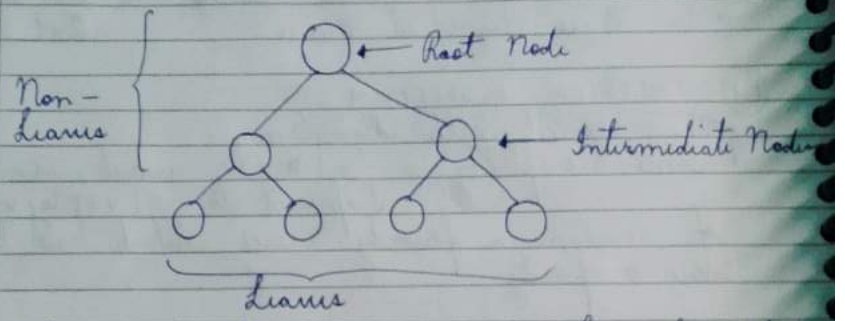
Applications =
→ In movie Queue ticket taking (First In First Out)
→ In Mess for food Queue

(iii) DIJKSTRA ALGORITHM =
→ The shortest path Algorithm.
→ Time complexity = $O(V^2)$, which can be improved to $O(V+E \log V)$ because of minimum priority Queue which has logarithmic time complexity.

DATE / /

* Trees = Trees are non-linear data structures which are not having connected circuit / edges.
Or
Non-linear data structures which does not have connected circuit.

→ A node of tree has three parts =
(i) Data
(ii) Pointer to left Node. (Left Childs)
(iii) Pointer to Right Node (Right Childs)

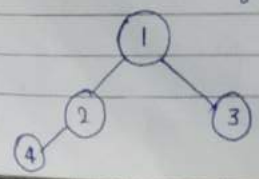


Let's Say there are n number of nodes then, the number of vertices will be

→ Number of Vertices = (n - 1)

(i) Binary Tree =
→ It must be having 0, 1 or 2 child nodes

Eg =



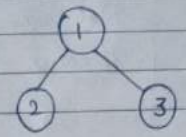
1 → 2 Child Nodes
2 → 1 Child Node
3 → 0 Child Node
Hence, Binary Tree.

DATE ___/___/___

(ii) Full Binary Tree =

→ Must be having 0 or 2 child nodes.

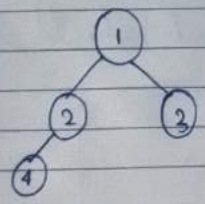
Eg = (a)



1 → 2 Child Nodes
2 → 0 Child Nodes
3 → 0 Child Nodes

Hence, full binary tree because having 0 or 2 child nodes.

(b)



1 → 2 Child Nodes
2 → 1 Child Node
3 → 0 Child Node
4 → 0 Child Node

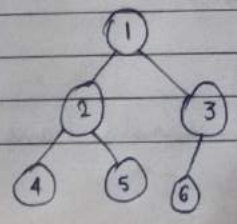
Hence, not full binary tree because 2 is having only one child node.

Note = A full binary tree is always a binary tree but binary tree is not always i.e; (that is) may or may not be full binary tree.

(iii) Almost Complete Binary Tree =

→ Their is scanning from left to right.
If the same last node is having 0 child 1, 2 child then almost complete Binary tree.
Else it would be not almost Complete Binary Tree.

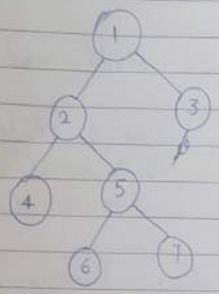
Eg = (a)



Almost Complete Binary Tree.

DATE / /

b)

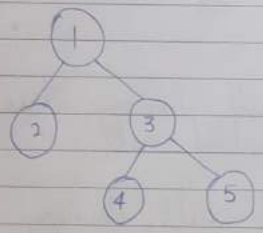


Not an Almost Complete Binary Tree.

ii) Complete Binary Tree =

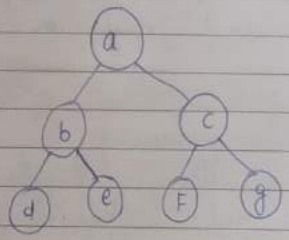
→ The nodes at same level should be having Equal child nodes.

Eg = a)



Not a complete Binary Tree

b)



Complete Binary Tree.

Binary Search Tree =

Properties =

i) The left subtree should have smaller elements

DATE / /

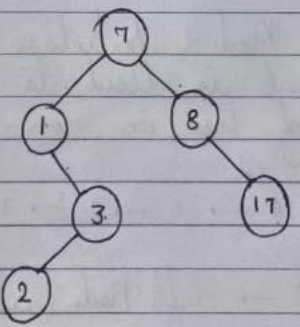
with respect to the root node.
ii) The right subtree should have greater Elements as compared to the root node.

Note = Binary Tree = 0, 1, 2 Childs
Full Binary Tree = 0, 2 Childs
Almost Complete Binary Tree = Left to right scan, having 0 or 2 childs at same level.
Complete Binary Tree = Having same number of child at same level.

* BINARY SEARCH TREE =
→ The binary search Tree is binary Tree which is having left subtree having smaller Elements than root node.
→ Right subtree is having greater subtree.
→ Number of Edges = (Vertices - 1)

Q

7	8	1	3	2	17



Binary Search tree

Edges = (Vertices - 1)

DATE / /

(i) According and with respect to breadth first search would be O(h) where h is the height of the tree.

(ii) According to Breadth First Search the time complexity for Insertion in Binary Search Tree would be O(n).

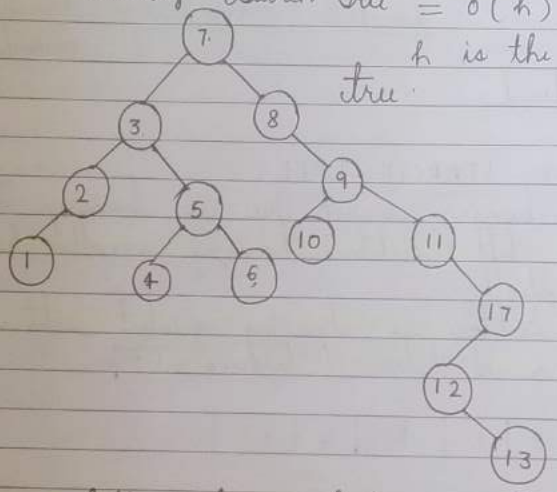
(iii) For deletion O(n), time complexity is O(h).

Q.

7	3	2	8	9	5	1	6	4	11	17	10	12	13

Binary Search Tree = O(h)

h is the height of the tree.



(i) Inorder = Left Node (Subclass) → Root → Right Node (Subclass)

(Inorder Traversal is used to get Element of tree in non-decreasing order)

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

(ii) Preorder = Root → Left Node (Subclass) → Right Node (Subclass)

→ Used to arrange the elements and is also used to find prefix expression of the given tree

DATE: / /

Preorder = Root → Left Node (Subclass) → Right Node (Subclass)

7 → 3 → 2 → 1 → 5 → 4 → 6 → 8 → 9 → 10 → 11 → 17 → 12 → 13

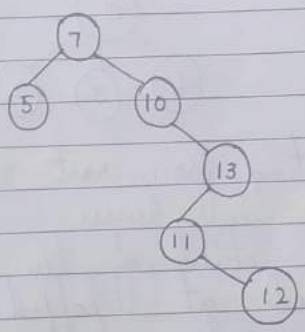
(iii) Postorder = Left Node (Subclass) → Right Node (Subclass) → Root

→ The Postorder traversal is used to delete the tree.

[Postorder = Left → Right → Root]

1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 17 → 13 → 12 → 8 → 9 → 11

Q.



→ Inorder = Non-decreasing Order

Left Node → Root → Right Node

(Subclass) (Superclass) (Subclass)

5 → 7 → 10 → 13 → 11 → 12

Preorder = Used to find prefix Expression.

Root → Left Node → Right Node

(Subclass) (Subclass)

7 → 5 → 10 → 13 → 11 → 12

Postorder = Used to delete the tree.

Left Node → Right Node → Root

(Subclass) (Subclass)

→ 5 → 12 → 11 → 13 → 10 → 7

DATE / /

* Breadth First Search =

→ Works on Queue.

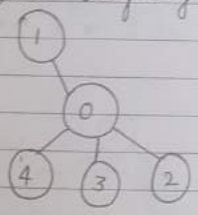
Queue is a linear data structure which works on concept of first In and First Out.

→ There are two ends front and Rear.
Insertion takes place from Rear end and deletion from Front.

→ Eg = A line of movie tickets.

→ Used to represent Adjancy list

Q.



Steps = (i) Starting from root node we have to push In Queue.

(ii) As the child of the root node comes the root node get popped.

(iii) If there are more than one child then concept of Queue is used.

S = X → Ø → X → 8 → X

BFS = 1 → 0 → 4 → 3 → 2

→ Time Complexity =

•) O (E + V) for Adjancy List where, E is Edges and V is vertices

•) O (V^2) for Adjancy Matrix, where V is i.e; nodes. vertices.

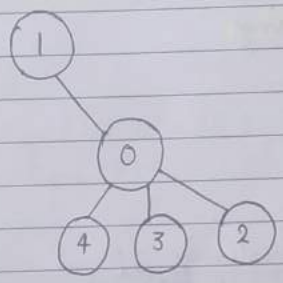
Since, we are dealing with Adjancy matrix, hence the time complexity becomes O(V^2).

DATE / /

* Depth First Search =
→ Works on Stack.
On the concept of Last In First Out
→ It is a vertex based Technique.

Steps = (i) The root element is pushed in the stack
(ii) As the child element arrives, the root element should be popped from the stack and dropped
(iii) If there are more than two child elements then Depth First Search is used.

Q.



S → 1 → 0 → 4 → 3 → 2
Depth First Search → 1 → 0 → 2 → 3 → 4

Note = The time complexities are as follo
(i) O(E+V) = Adjancy List
(ii) O(V2) = Adjancy Matrix

Since, we are dealing in terms of adjancy Li
Matrix, hence, time complexity would be
O(V²) where V is the vertex.

DATE / /

HASHING

→ Hashing is a process of mapping keys in a hash table by use of hash functions.

→ Useful in storing or retrieving data from the table.

Collisions

- Open Hashing
  - Chaining
- (Open Addressing) Handling Collisions: Closed Hashing
  - Linear Hashing
  - Quadratic Probing
  - Double

Linear Probing =

→ It is chain hashing which is also Probing termed as Open Addressing for handling collisions.

→ No need of extra space.

Steps = (i) Take out the modulus of given element to which respect to the given func hashing function.

(ii) Place the element at that Index position

(iii) If in any chance that particular Index position is full then move the element to next empty Index Position.

Q

42	35	12	19	52

k mod 5

0	1	2	3	4
35	52	42	12	19

DATE / /

(ii) Chaining :-

→ It is kind of open hashing which does not controls address.

→ Extra space is required.

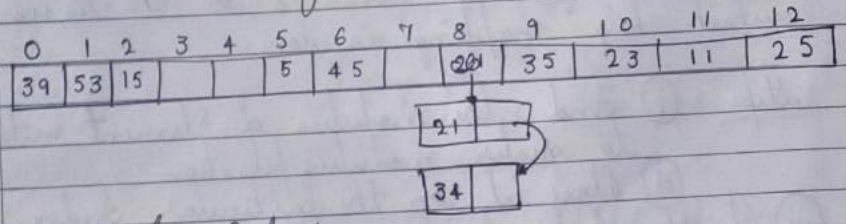
Steps :- (i) Find the modulus of element according to given function.

(ii) Place the elements at particular index position.

(iii) If two or more elements are having same modulus then linked list addressing is followed.

Q. 5, 15, 25, 35, 45, 11, 21, 23, 34, 39, 53

Mod function (13)



(iii) Quadratic Probing :-

→ It is closed hashing collision which deals with open addressing i.e., the address collisions are not occuring.

Steps :- (a) Find the modulus of the element with respect to the given function.

(ii) Place (b) Place the elements on the given index location.

(c) If the index location is matched then add square of probe number which ranges from 1 to n.

[Index = $h(x) + i^2$]

DATE / /

Q. 5, 15, 25, 35, 45, 11, 21, 23, 34, 39, 53
k Mod 13

0	1	2	3	4	5	6	7	8	9	10	11	12
39	34	15		53	5	45		21	35	23	11	25

$5+i^2$ =	$1+i^2$
$5+1=6$	$1+1=2$
$5+4=9$	$1+4=5$
$5+9=14\ \%\ 13=1$	$1+9=10$
	$1+16=17\ \%\ 13=4$

Double Hashing =

→ It is closed hashing collision which is also termed as Open Addressing which restricts add ressing collisions.

→ It is having two functions on the basis of which key mapping is done.

Steps = (i) Find the modulus of element with respect to given modulus function.
(ii) Place it on the particular Index location.
(iii) If the Index position (Address) is not free i.e (that is) collisions is occuring, then find the modulus with respect to other modulus function, add them.
(iv) If then Also collision is occuring then multiply the modulus of second fuction with i ranging from 1 to n and then add till the time, the empty Index location is found.

DATE / /

Q. 5, 15, 25, 35, 45, 11, 21, 23, 34, 39, 53
k mod 13, k mod 11

0	1	2	3	4	5	6	7	8	9	10	11	12
39	53	15			5	45	34	21	35	23	11	25

34 % 13 = 5 , 34 % 11 = 1
5 + 1 x 1 = 6
5 + 1 x 2 = 7

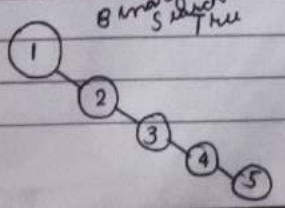
AVL Tree -

→ AVL Tree is binary used to control the hight of binary search Tree.
→ As the time complexity of the binary search tree is O(h) where h is the hight of the tree, if the hight becomes fully skewed then the worst case becomes O(n).
→ Hence, on the basis of balanced factor, we will be able to balance the hight of binary search tree.

Balanced Factor - (Maximum hight of left subtree - Maximum Height of right subtree)

If it ranges or it's domain is between (-1, 0, 1) then the tree is balanced, else we have to balance it on the given conditions.

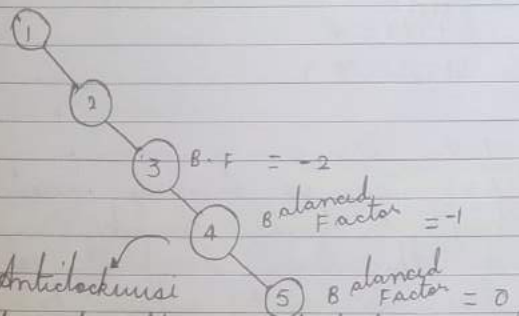
Q. 1, 2, 3, 4, 5



DATE / /

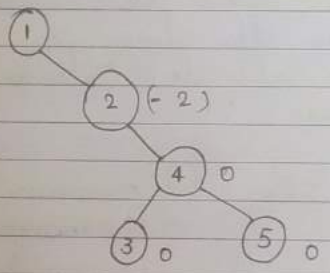
Balance Factor of Root Node = 0 - 4 = -4
Hence, Unbalanced.

→ Now, we have to check balanced factor of each subnode and the node which is Unbalanced, have to balance it.



Anticlockwise

Case (i) = If Right skewed, then Anticlockwise rotation



DATE / /

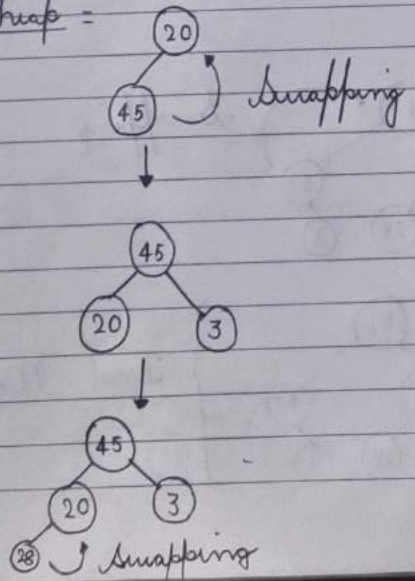
Case (i) = If Unbalanced node is following right skewed mechanism then Anticlockwise rotation.
Case (ii) = If Unbalanced node is following left skewed mechanism then clockwise rotation.
Case (iii) = If none of these first then also Unbalanced node then according to given condition two rotations.

HEAP TREE

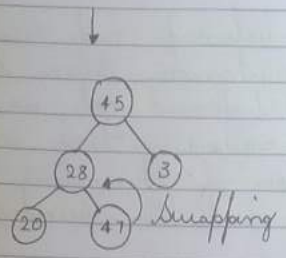
→ Construct a ~~binary~~ heap tree for a tree using heap tree condition must be followed is =
(i) The tree must be almost complete binary tree i.e. that is from scanning left to right the nodes of same ~~tree~~ must be having 0 or 2 child.
(ii) Order → Max heap (Parent > Child)
→ Min heap (Child > Parent)

Q 20, 45, 3, 28, 47, 51

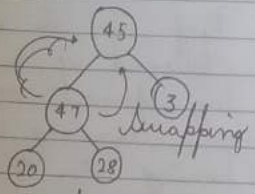
→ Max heap =



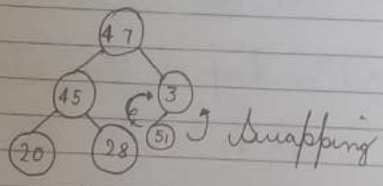
DATE / /



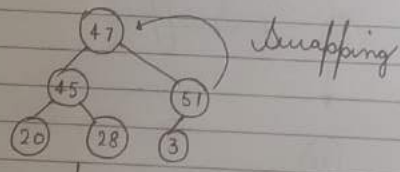
Swapping



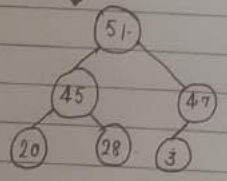
Swapping



Swapping



Swapping

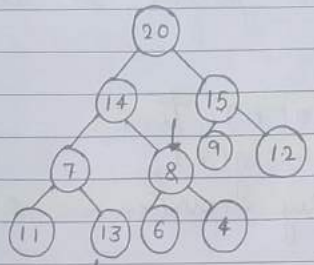


Final Max heap tree

DATE / /

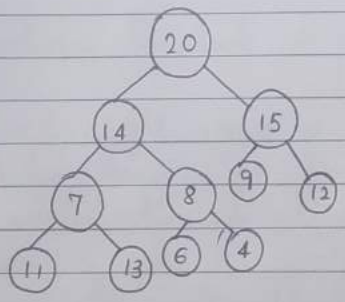
Note = If the binary Almost Complete binary tree is given And we have to make as maximum heap tree, then find the middle element and check the child node then scan from left to right, the nodes and their child.

Q.

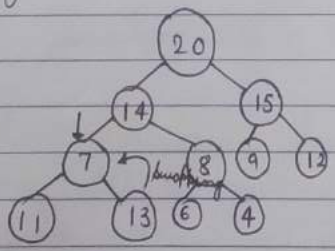


Number of Nodes = 11

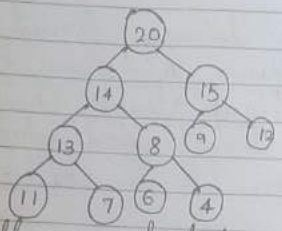
Middle Node = $\frac{11}{2} = 5.5 \sim 5$



Now, if the middle node is at perfect position move leftwards on same level node.



DATE / /



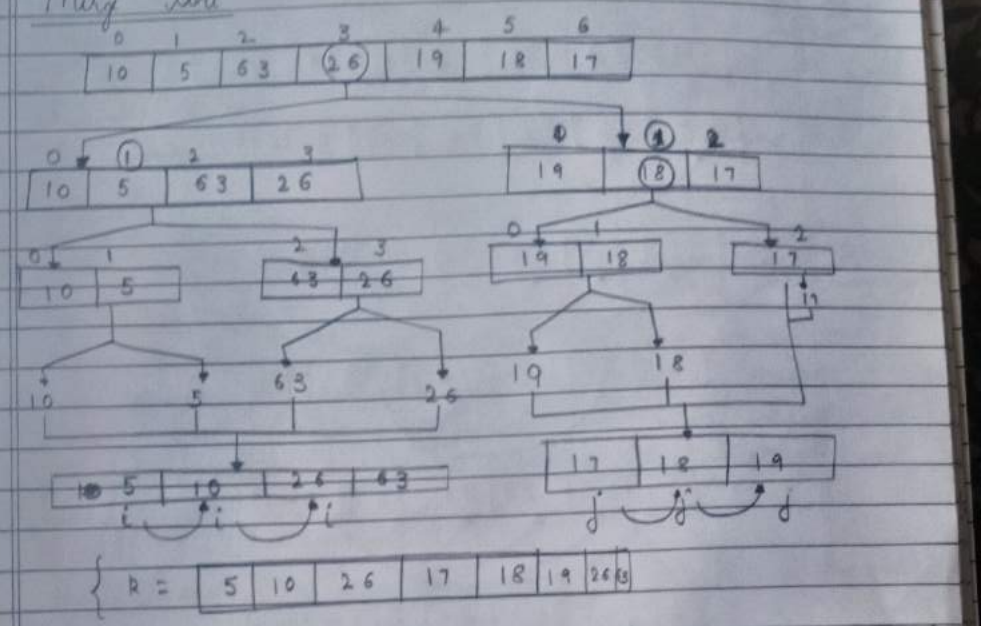
Final Max Heap.

Now, all are perfect so no need to move leftwards.

→ Create heap $O(n)$ time.
Heapify Method = $O(n)$
One by One key Insertion = $O(n \log n)$

DATE / /

Q Merg Sort



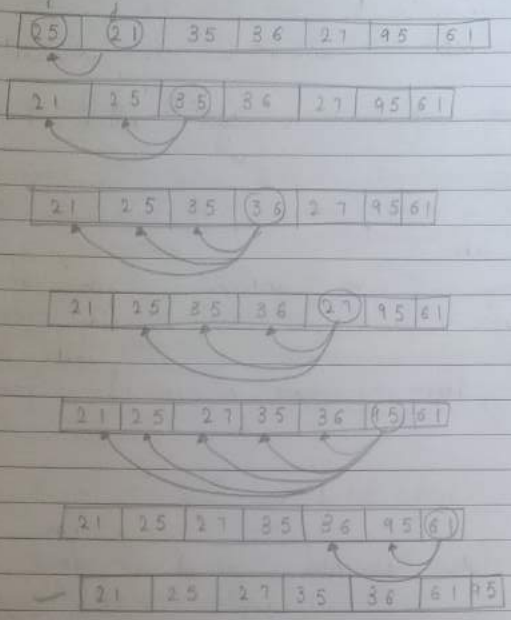
→ Three functions Required = First to sort left side of the middle element.
→ Then to sort right side of the middle element.
→ Merg function having i for left sorted elements, j for right sorted elements and k which stores after comparision
→ Worst case time complexity = O(n log n)

O(n) + O(log n)
↑ Spreading till n elements — Joining or Merging elements

Merg Sort always divides into two halves and then merge all elements in linear time.
Hence, for Spreading O(log n) and for merging O(n)
∴ O(log n) + O(n) = O(n log n) | Best, Worst, Average Case

DATE / /

Insertion sort = i = 0th Index , j = i+1



Worst Case time Complexity = $O(n^2)$
and Average Case

→ Always assume first Element to be at correct position.
→ Then compare it with next Index location Element and if greater then leave as it is, but if smaller then swap.
→ Now that particular part is already sorted then compare further.
→ If the next Indexed Element is larger than last sorted Element, then no need to check.

DATE / /

Q AVL Tree =

→ It is used to balance the binary tree without it's height as not losing it left skewed or right skewed

→ The domain of balance factor must rang between {-1, 0, 1}.

→ Balance Factor = (Left Subtree - Right Subtree)

→ Right skewed = Unbalanced } Apart from this in combination according to given condition.

→ Left skewed = Unbalanced

Q Linear Search =

→ The Identified Element which we have to search, we check Element at each Index location starting from first Index location and will return the Index position's value where the Element is found.

→ Best Case time complexity = O(1) → It is possible that the Element at first Index location is only the Element which is to be searched. Hence, no need to check further.

→ Worst Case time Complexity = O(n) → The Element which is to be searched may be the Element at last Index position and hence, loop will run from 0th Index position to nth Index position.

Q Hashing =

→ It is non-linear datastructure used to allocate memory key to the Elements in the hash table according to hash functions.

→ There are two types of Collisions in hashing =

① Open Hashing = Consists of Chaining

DATE / /

ii) Closed hashing Or Open Addressing = It handle Address collisions and is further divided into three parts

(a) Linear hashing
(b) Quadratic Probing
(c) Double Probing

(a) Linear Hashing =
→ Find the modulus of Element with respect to given hash function.
→ Allocate the Index position according to the modulus
→ If the Index position is not Empty then place the Element to further Empty memory location (Index).

(b) Quadratic Probing =
→ First find the modulus of the Element with given hash function
→ Place the Element at particular Index location
→ If the particular Index location is not free or Empty then check for Empty position according to given formula =

$$I = (h + i^2) \quad \text{(Probe Number } i = 0 \text{ to } n\text{)}$$

h → Index position (which is not free)
I → Index Position where the Element is to be placed.

(c) Double Probing =
→ Find the modulus of the Element according to given hash function.

DATE / /

→ Place the element at particular Index position, if the Index position is free.
→ If not free then apply the given formula As :-

$$I = I(h1) + i \cdot I(h2)$$

- Modulus of with respect to first hash function (Greater Number among both functions is given the priority)
- i : Put Number according to second Index position
Range from domain of 0 to n, till the time empty location is not found.

Q Binary Tree :-
→ A part of non-linear tree datastructure, which is having 0, 1 or 2 (Atmost) child element or subnodes
→ Do not have any condition for left subtree or Right subtree

Q Binary Search Tree = A binary tree having condition that with respect to the root node the left subtree must contain smaller elements
→ The right subtree must contain greater elements
→ The main purpose of Using Binary Search Tree concept in this is that searching of the elements will become easier and is termed as Searching Sequence.

Q Complete Binary Tree = The binary tree in which all nodes (Vertices) at same

DATE / /

Level has Equal Amount of child nodes. (Either 2 or 0)

Q What is Linked List? → Single Linked List / Doubly Linked List / Circular Linked List

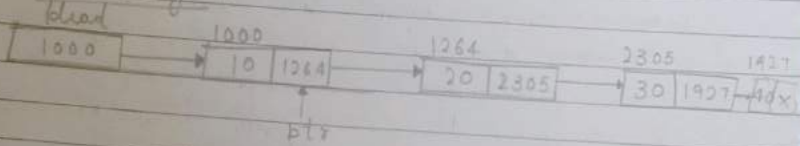
→ Linked List is linear data structure in which Elements are stored at non-contiguous memory locations.

→ Node → data / → Pointer (Address to Next Node)

Advantages =
- → Free Usage of memory.
- → Insertion and deletion operation becomes Easier.
- → Random Access of Elements.

Disadvantages =
- → due to Usage of pointers the Extra memory is Utilized
- → Random Access to the Element is not possible.

(i) Traversing =

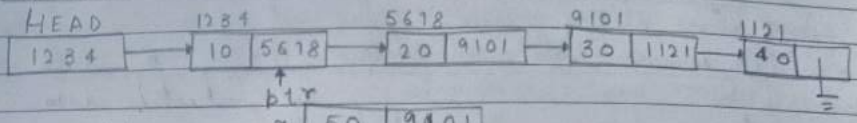



```
    Head = ptr ;
{   while (ptr != NULL)
    cout << ptr → data ;
     ptr → next ;
}
```

→ Time Complexity = O(n).
But for Each Node, it becomes O(1) which in combination of n nodes becomes O(n).

DATE / /

Insertion =




```
while (ptr != NULL)
  { HEAD = PTR;
    PTR → x
    x → next = PTR;
    cout << x → next;
  }
```

For Insertion at first node = O(1)
Insertion at last = O(n)

3

Q Already having Arrays then why Linked List?
→ (i) Random Access of Elements according or without memory Allocations (Random Access is not Allowed)
(ii) Insertions and dilution at particular node takes constant time i.e., (that is) O(1).
which in combination of All nodes becomes O(n)

Q What is data structures?
→ The physical and logical representation of Information (processed data) in sequence, is termed as data structures.
→ The physical representation means time complexity and space complexity.
→ The logical representation means the techniques and logics required for the Implementation of program.

DATE / /

Q Types of data structures?

→ There are two types of data structures

(i) Linear Data Structure - Type of data structure in which data is arranged in Sequential Order
Eg = Arrays, linked-list etc

(ii) Non-linear Data Structure - Type of data structure in which elements (data) is not arranged sequentially
Eg = Graphs, trees etc.

Q Difference between trees and graph?

→ Graphs are non linear data structures having cyclic structure which means the edges are connected to vertices and all are joined logically

→ In Graph the parent node can have many such nodes or child nodes.

→ The traversal of graph can be done by Breadth First Search or Depth First Search

→ For finding Minimum Spanning tree, these algorithms are used
(i) Prim's Algorithm
(ii) Dijkstra's Algorithm

→ For finding the shortest path, Dijkstra Algorithm is used.

→ Trees are also non-linear data structures which vertex (node) can have as many subnodes but in Binary Trees can have 0, 1 or 2 Subnodes.

→ Trees does not make cyclic structure.

→ Further divided into

DATE / /

(a) Binary Tree = Having 0, 1, 2 child nodes.
(b) Full Binary Tree = Having 0, 2 child nodes.
(c) Almost complete Binary Tree = Having same 0 or 2 child at same level nodes when scanning from left to right.
(d) Complete Binary Tree = Having 2 child nodes at each level.

Q Heap Tree?
→ Conditions for a tree to become a heap Tree is
(i) It must be almost complete Binary Tree.

There are two types of heap tree =
(i) Max = Parent > Child (For descending order traversal of tree)
(ii) Min = Child > Parent (for ascending order traversal of tree)

Sorting Algorithms:
(i) Bubble Sort = $O(n^2)$, $\Omega(n)$
(ii) Selection Sort = $O(n^2)$, $\Omega(n^2)$ Worst case
(iii) Insertion Sort = $O(n^2)$ ← Average case, $\Omega(n)$
(iv) Merge Sort = $O(n \log n)$
(v) Quick Sort = $O(n \log n)$
(vi) Heap Sort = $O(n \log n)$

What is the best Sorting Algorithm?
→ The best sorting Algorithm is Quick Sort because we choose the pivot Element and swapping is done from start and end without requirement of Extra space or memory allocation.
Time Complexity = $O(n \log n)$

Time Complexity of Bubble Sort?

DATE / /

8. → $O(n^2)$ because two loops are running.

9. Best case time complexity of Quick sort?
→ Best case = $O(n \log n)$
Average case = $O(n \log n)$
Worst case = $O(n^2)$

10. Bubble sort vs Selection sort?
→ Between bubble sort and selection sort, selection sort is more efficient because in bubble sort as compared to selection sort may swapping are taking place and more memory space is also required

	Best Case	Average Case	Worst Case
Bubble sort	$\Omega(n)$	$\Theta(n)$	$O(n^2)$
Selection sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$

11. When bubble sort will perform the best?
→ Bubble sort will perform the best when the elements in the list is already sorted. It will get terminated within one iteration only.

12. If Binary tree then why binary search tree?
→ Binary tree root node can have atmost two child nodes or sub nodes in which there is random arrangement of elements in left subtree and right subtree.
→ But in binary search tree is having smaller element at left subtree and greater

DATE / /

Element at right subtree with respect to the root node due to which traversal, searching and insertion of nodes in the tree becomes efficient.
Inorder = Left Root Right
Preorder = Root Left Right
Postorder = Right Left Root

Q Advantages of data structure?
→ (i) Physical and Logical Representation.
(ii) Optimality of the program in terms of efficiency
(iii) Can perform searching, sorting, work with Random memory allocations i.e. linked list, stack, Queues, Graphs, trees, Hashing and many more dynamic memory allocation concepts.

Q Function used while Allocating Linked List?
→ By using structure or class we can make the node consisting data and pointer (which is address to next node).
→ Malloc for memory allocation having single argument.

Q What is binary search?
→ It is a kind of searching technique in which searching is done by finding middle element
→ Then at particular at again checking and repeating the steps
→ Ex If greater than middle element then changing first element to mid+1.
→ If smaller than middle element then changing

DATE / /

last Element as mid-1.
→ Time Complexity = Worst Case = O(log n)
But Case = O(1)

Q Different functions performed on Array?
→ (i) Searching
(ii) Sorting
(iii) Traversal
(iv) Insertion
(v) Deletion

Q Algorithms for Traversal of Graph?
→ (i) BFS (Breadth First Search) = Based on concept of Queue
(a) The root Element is pushed in Queue and as soon as the next level child Elements are pushed, the root Element is popped to the final set
(b) If there are more than two child nodes of a root node then popping is done according to Queue i.e., first In and first Out

(ii) DFS (Depth First Search) = Popping is done according to concept of stacks

Q Tree Traversal =
→ Inorder = L P R
→ Preorder = P L R
→ Postorder = R L P

Q Explain Stack and Queue?
→ Stack = The linear datastructure in which the phenomena of Last In First Out

DATE / /

occurs.
→ The Insertion of Elements is termed as pushing the Elements.
→ The extraction is termed as popping.
Traversing = O(n)
Insertion = O(1) If done at the head of the stack.
Deletion = O(n) O(1) if done at head of the stack
O(n)

Queue = A non linear data structure which works on the phenomena of FIFO FIRST IN FIRST OUT

→ Used in various Real life problems like
(a) Ticket Counter
(b) Mess Counter
(c) Queue of rotis
→ Having two ends, Front and Rear.
→ Insertion is done from Rear End and deletion is done from Rear End.

Insertion = Enqueue → O(1)
Deletion = Dequeue

Traversing = O(n)

Q ADT ?
→ Abstract data types.
The Abstract data types are the ~~package~~ used for only showing necessary details and hiding the Implementation of the same.
Eg = Array, List, Binary tree &.

DATE / /

Time Complexities =

i) Array

Traversing
O(n) ⇒ Worst Case
O(1) ⇒ Best Case

Insertion
O(n)

Deletion
Searching

Deletion
O(n)

Linear Search =
Best Case ⇒ O(1)
Worst Case ⇒ O(n)

Binary Search
Best Case
O(log(n))

ii) Sorting

Bubble Sort
Best Case = O(n)
Average Case = O(n)
Worst Case = O(n²)

Insertion Sort
Best Case = O(n)
Average Case = O(n)
Worst Case = O(n²)

Quick Sort =
Best Case = O(n log n)
Average Case = O(n log n)
Worst Case = O(n²)

Heap Sort =
Best Case = O(n log n)
Average Case = O(n log n)
Worst Case = O(n log n)

Selection Sort =
Best Case = O(n²)
Average Case = O(n²)
Worst Case = O(n²)

Merg Sort =
All three Cases = O(n log n)
Best Case = O(n log n)
Average Case = O(n log n)
Worst Case = O(n log n)

DATE / /

Stack =
Push = O(1)
Pop = O(1)

Queue =
Insertion = O(1)

Graphs =
Prim's Algorithm = $O(V^2) \longrightarrow O((V+E) \log V)$
Kruskal's Algorithm = $O(E \log V)$
Dijkstra's Algorithm = $O(V^2) \longrightarrow O(V + E \log V)$

Trees =
BST = $O(h) \longrightarrow O(n)$
BFS = $O(E+V)$ → Adjancy List
$O(V^2)$ → Adjancy Matrix
DFS = $O(E+V)$ → Adjancy List
$O(V^2)$ → Adjancy Matrix