Unit-III

IA-2

1 Searching (2023

Searching is a method to finding an element from data structure with their appropriate location.
Searching is an non-linear data structure.

Two types of Searching.
- Linear Searching.
- Binary Searching.

1. Linear Searching:
In linear Search we Search an element in a given array by traversing the array from the Starting till the desire element is not found.

-Algorithm :-

Step1 : Begin
Step 2 : Set a[3] {10,20,30}
Step3 : Set i
Step4 : input Searching item
Step5 : Repeat step 6 to 7 while i<3

Step 6 : if a[i] = item then,
Print "item found" location = i & exit

Step 7 : i+1
Step 8 : if i>=3 then
Print item "Not found"
Step 9 : exit

Program :

```
#Include < Stdio.h>
void main ()
  { int item, i=0;
    int a[5] = { 34, 56, 32, 78, 12 };
    Printf(" Enter Searching Item");
    Scanf("%d", &item);
    while (i<5)
     {
       if (a[i] == item)
        {
          Printf("Item Found at %d", i);
          exit(0);
        }
       i++;
     }
     if (i>=5)
      {
        Printf("Item Not Found");
        exit(0);
      }
  }
```

2. Binary Searching.

Binary Search is the divide and Conquer Searching technique in which we have to arrange the data in particular order before Searching operation After that we find mid element of the array and Compare with target element.

Algorithm:

Step1: Begin
Step2: Set a[3] {10, 20, 30}
Step3: Set lr, up, f ← 0
Step 4: Input Searching item
Step 5: Repeat step 6 to 8 while lr <= up
Step 6: Set mid ← $\frac{lr+up}{2}$
Step 7: if a[mid] = item then,
Set f = 1 & breals
Step 8: If a[mid] < item then,
Set lr ← mid+1
else
Set up ← mid-1
Step 9: if f=1 then
Print "item found" location = mid
else
Print "Not found"

Program:

```
#include <stdio.h>
void main ()
    int a[5]={10, 20, 30, 40, 50};
    int lr=0, up=4, mid, item; f=0;
    Printf("Entir Searching item:");
    Scanf("%d", &item);
    while (lr<=up)
    {
        mid = (lr+up)/2;
        If (a[mid]==item)
```

```
    {
      f=1;
      break;
    }
}
if (f==1)
{
  Printf("Item found at %d", mid);

if (a[mid] < item)
  {
    lr = mid+1;
  }
else
  {
    up = mid -1;
  }
}
if (f==1)
  {
    Printf("Item found at %d", mid);
  }
  else
  {
    Printf("Item not found");
  }
}
```

2023

2 Sorting

Sorting is the method to arrange an elements of array in a particular order either ascending or descending order.

Types of Sorting:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort

1. Bubble Sort:

Bubble Sort arrange N element of array by placing the biggest element on proper position.

eg: 8 7 5 6 2

Step 1:
7 8 5 6 2
7 5 8 6 2
7 5 6 8 2
7 5 6 2 8

Step 2:
7 5 6 2
5 7 6 2
5 6 7 2
5 6 2 7

Step 3:
5 6 2
5 2 6

Step 4:
5 2
2 5

Result: 2 5 6 7 8

Algorithm :

Step 1: Begin
Step 2: Define a array
Step 3: Compare the 1st element with 2nd ele.
- If the 1st element is greater swap them
- move to the next pair of element.

Step 4: Continue Comparing and Swapping adjacent element until the largest element bubble up to the end.
Step 5: Repeat the same process for the remaining
Step 6: Exit

Program :

```
#include <stdio.h>
int main()
  int i, n, j, temp;
  Printf("Enter Array Size:");
  Scanf("%d", &n);
  int a[n];
  Printf("Enter values in array:");
  for(i=0; i<n; i++)
    {
      Scanf("%d", &a[i]);
    }
  for(i=n; i>0; i--)
    {
      for(j=0; j<i-1; j++)
        {
          if(a[j]>a[j+1])
            {
```

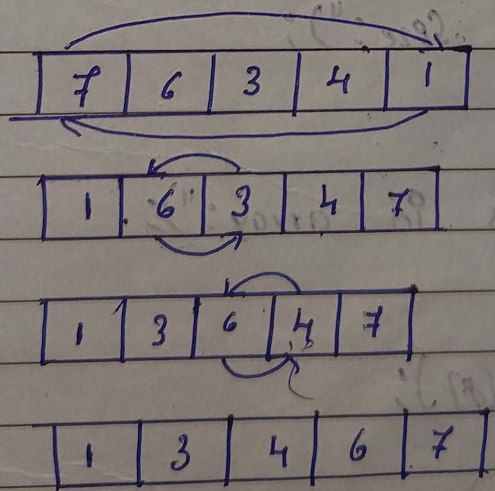
→ n>0 --
0 <i-1 ++

```
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }
  printf("Sorted Result: \n");
  for (i=0; i<n; i++)
    printf("%d", a[i]);
  }
  return 0;
  }
```

2. Selection Sort.

Selection Sort arrange N element of array by ~~planning~~ placing the smallest item in proper position incase of ascending order arrangement.

eg:



Algorithm:

Step 1: Begin
Step 2: Find the Smallest element of array
Step 3: Swap the Smallest number with the 1st number in the list
Step 4: Repeat for the rest
Step 5: Swap again with newly found elements.
Step 6: Keep repeating
Step 7: Done Sorting
Step 8: Exit.

Program:

```
#include <stdio.h>
void main()
{
int i, n, m, j, loc, temp;
Printf("Enter Array Size:");
Scanf("%d", &n);
int a[n];
Printf("Enter values in array:");
for (i=0; i<n; i++)
{
  Scanf("%d", &a[i]);
}
for (i=0; i<n-1; i++)
 {
   m = a[i];
   loc = i+1;
   for (j= i+1; j<n; j++)
    {
```

```
if (m>a[j])
 {
    m = a[j];
    loc = j;
  }
}
if (a[loc] < a[i])
 {
   temp = a[loc];
   a[loc] = a[i];
   a[i] = temp;
 }
}
```

3. Insertion Sort:

Insertion Sort arrange N elements of array by inserting particular item in a particular ~~item~~ ~~in~~ a place such away that the item in sorted order.

eg:

7 6 3 4 1
6 7 3 4 1
6 3 7 4 1
3 6 7 4 1
3 6 4 7 1
3 4 6 7 1
3 4 6 1 7
3 4 1 6 7
3 1 4 6 7
1 3 4 6 7

Algorithm:

Step 1: Begin
Step 2: Start with 2nd element of the array
Step 3: Compare & shift, if the current element is smaller, shift the larger element to the right
Step 4: Place the current element in its current sorted position within the shifted portion
Step 5: Move to the next unsorted element and repeat step 3 and 4 until all elements are sorted.
Step 6: exit.

Program:

```
#include <stdio.h>
int main()
{
  int i, n, j, temp;
  Printf("Enter Array Size");
  Scanf("%d", &n);
  int a[n];
  Printf("Enter Values in array:");
  for(i=0; i<n; i++)
  {
    Scanf("%d", &a[i]);
  }
  for(i=1; i<n; i++)
  {
    for(j=i; j>=1; j--)
    {
      if(a[j-1]>a[j])
```

```
        temp = a[j-1];
        a[j-1] = a[j];
        a[j] = temp;
      }
    }
  }
  printf("Sorted Result: \n");
    for(i=0; i<n; i++)
      {
        printf("%d", a[i]);
      }
  return 0;
}
```

4. Quick Sort: (2023)

Quick sort algorithm ~~item~~ Separate the list of elements into two parts and sort each part recursively. It use devide and conquer method.

In this method the partision of list performed based on pivot element.

The list is devided into two partision such that all elements to the left pivot are smaller and all elements to the right are greater.

eg: (50) 3 1 60 65 45 90 13 67 — pivot element.

13	3	1	60	65	45	90	(50)	67
13	3	1	(50)	65	45	90	60	67
13	3	1	45	65	(50)	90	60	67
13	3	1	45	(50)	65	90	60	67

(13) 3 1 45
1 3 (13) 45

(65) 90 60 67
60 90 (65) 67
60 65 90 67
60 65 67 90

Algorithm :

Step 1 : Begin
Step 2 : Select the start element of array as a pivot element.
Step 3 : ~~exchange~~ ~~both~~ ~~element~~ Scan ~~the~~ find the Smallest element from right side of array
Step 4 : exchange both element
Step 5 : Scan and find the biggest element from left side of list
Step 6 : Repeat the process until all the element of left side are Smaller and right side elements are greater than pivot.
Step 7 : Now two sub-list are available apply Same process until all element of array are not Sorted.
Step 8 : exit

Program :

```
#include <stdio.h>
void quicksort (int a[], int low, int high)
{
  if (low<high)
  {
    int pivot= a[high];
    int i=low-1, temp;
```

```
for (int j=low; j<=high-1; j++)
 {
   if (a[j] < pivot)
    {
      i++;
      temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }
 }
temp = a[i+1];
a[i+1] = a[high];
a[high] = temp;
quicksort(a, low, i);
quicksort(a, i+2, high);
  }
}
int main()
 {
   int i, j, n, low, high, pivot, temp;
   Printf("Enter Array Size:");
   Scanf("%d", &n);
   Int a[n];
   Printf("Enter values in array:");
   for (i=0; i<n; i++)
    {
     Scanf("%d", &a[i]);
    }
    quicksort (a, 0, n-1);
    Printf("Sorted Result:\n");
     for (i=0; i<n; i++)
       Printf("%d", a[i]);
 }
```

2023

5. Merge Sort:

Merge sort is a sort algorithm that split the items to be sorted into two group recursively sort each group and merges them into a final sorted sequence.

eg: [7, 1, 8, 3, 9, 2, 10, 12]

[1, 7, 3, 8, 2, 9, 12, 10]

[1, 3, 7, 8, 2, 9, 10, 12]

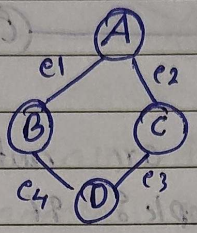
[1, 2, 3, 7, 8, 9, 10, 12]

Algorithm:

Step 1: Begin
Step 2: Divide all the items of array in two item pairs, if single item remain make it single group.
Step 3: Sort each of the pair
Step 4: Merge two pairs into a single pairs
Step 5: Sort the merge pair
Step 6: repeat the above step until all elements of array are not sorted.
Step 7: exit.

3 Graph.

Graph is a collection of data item called node. where nodes are connected to each other by the help of edge.



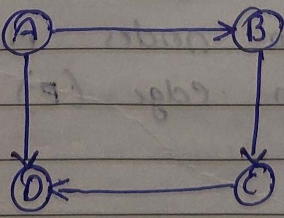
- A graph G can be represented an ordered pair. G(V, E)

- where 'v' indicates no of vertices or node and E indicates no of edges.

Types of Graph.

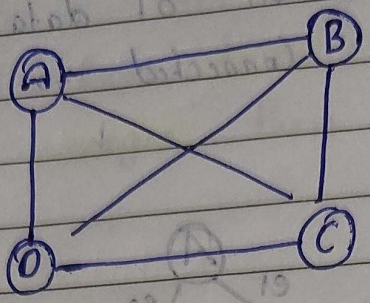
1. Directed Graph
2. Undirected Graph.

1. Directed Graph



it indicates a specific path from one vertex to other vertex.

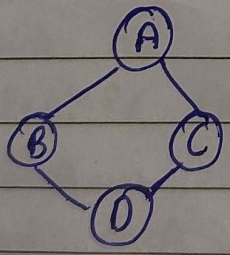
2. Undirected Graph.



in this, graph edges are not associated with the any directions for example: if an edge exists between A & B vertex then the vertices can be traversed from B to A and A to B.

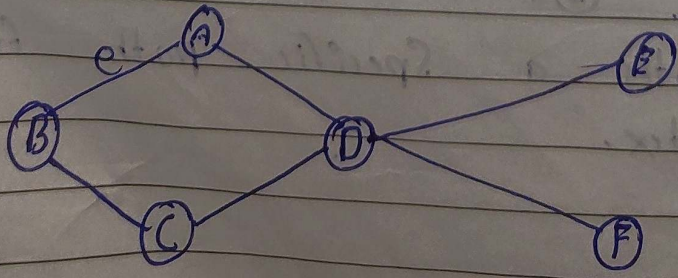
Terminology of Graph.

Path: A path is nothing but a way to reach initial node to terminal node.

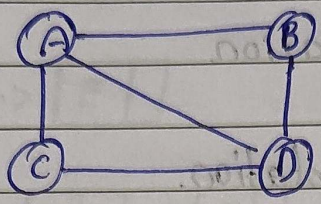


A → B
C → D.

Adjecent node: If two nodes A and B are connected via an edge (E)

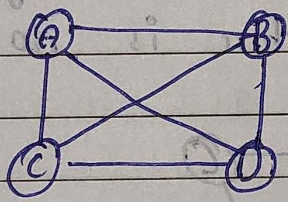


Degree : The degree of nodes indicates total no of edges connected to it



Degree (A) = 3
Degree (B) = 2.

Complete Graph : If all the nodes of a graph connected to each other is called Complete graph.



2023

4 Graph Representation

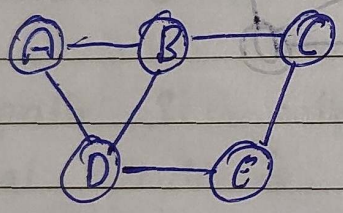
- Sequential Representation
- Linked Representation

1. Sequential Representation.

In this representation a matrix is used to represent the graph called Adjacency matrix in which M is the no of vertices in the graph and value of matrix can be defined as.

1. Vij = 1, if there is edge between Vi to Vj
2. Vij = 0, if there is no edge between Vi to Vj

eg:



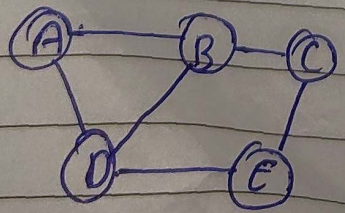
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

2. Linked Representation.

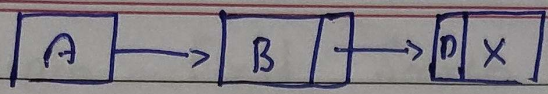
In linked representation at first defined the adjacency list.

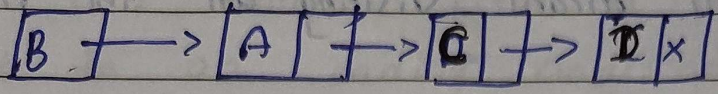
By using adjacency list and linked list we can represent the graph.

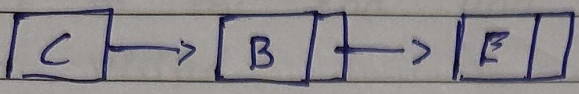
eg:

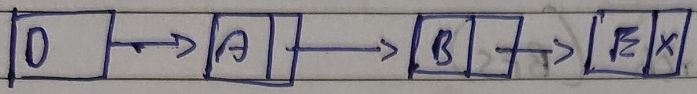


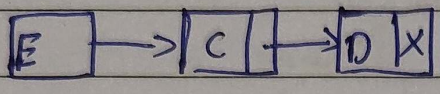
Vertex	Adjacency list
A	B, D
B	A, C, D
C	B, E
D	A, B, E
E	C, D

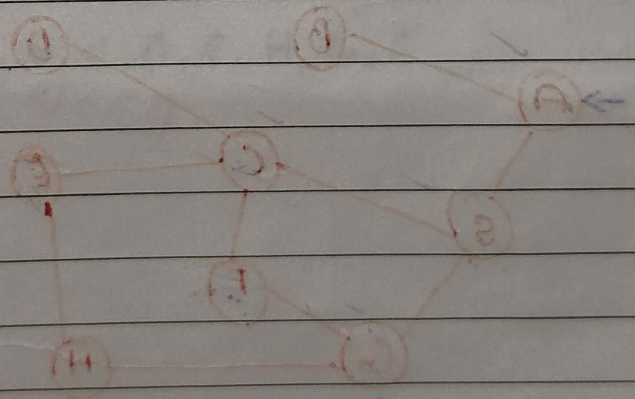

A
B
D
X


B
A
C
D
X


C
B
E


D
A
B
E
X


E
C
D
X



2023

5 Graph Traversals

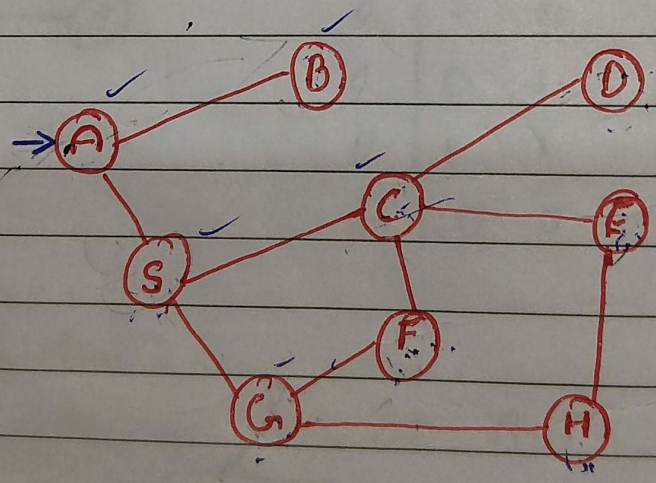
A graph traversals means visiting all the nodes of the graph.

Two types.

1. Breadth First Search (BFS)
2. Depth First Search (DFS)

1. Breadth First Search (BFS)

In breadth first search, one node is selected as start position. It is visited and marked, then all unvisited nodes adjacent of the next node are visited and marked in some sequential order.



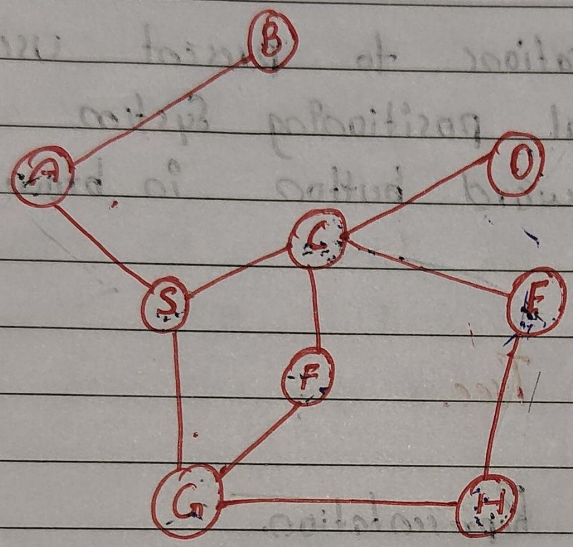
Output : A, B, S, C, G, D, E, F, H.

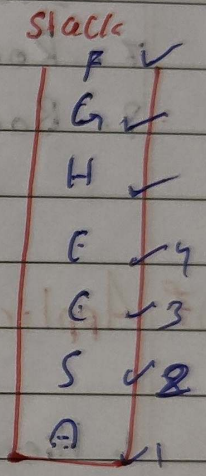
Queue status

A	A ✓1
B	B ✓2
S	S ✓3
C	C ✓4
G	G ✓5
D	D ✓6
E	E
F	F
H	H

2. Depth First Search (DFS)

DFS follows first a path from the starting node to an ending node. then another path from t start to the end and so forth, until all nodes have been visited.





Output: A, B, S, C, D, E, H, G, F

6 Graph Applications.

RENT MB.

1. Travelling Application
2. Mapping Applications.
3. Resource utilization and availability in an organization
4. Shortest path from point one point to another point
5. Neural Networks.
6. Ecommerce applications to present user preferences.
7. Routes in global positioning System.
8. Back or Forward button in browsers.

(Unit IV) # Application of Tree.

1. Hierarchical Data Representation.
2. Database Indexing.
3. Search Algorithms
4. Network Routing
5. Decission-Making Process.
6. Artificial Intelligence
7. Expression Parsing.

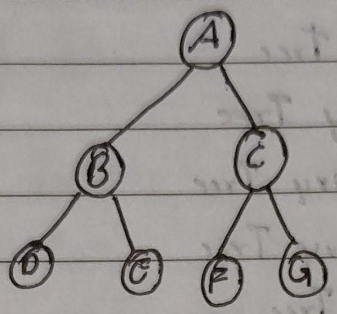
DASH END

Unit - IV

1. Trees.

Tree is a non-linear Collection of data, item called node where each nodes Connected by edges.

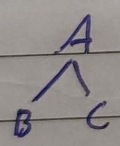
eg:-



- The hierarchical representation of node called tree.
- A Special node through which the tree get started is called root.

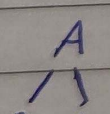
Terminology of tree:

i. Sibling:



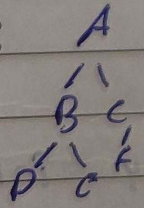
B & C are the sibling.

ii. Terminal:



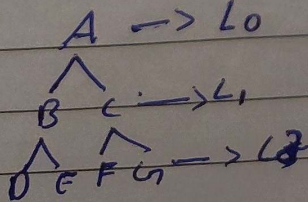
The node which Contain no any child node. (B, C)

iii. Degree:



Degree A = 2
Degree C = 1

iv. Level:



1. Binary Tree.

A tree which one node contains maximum two child node is called binary tree.

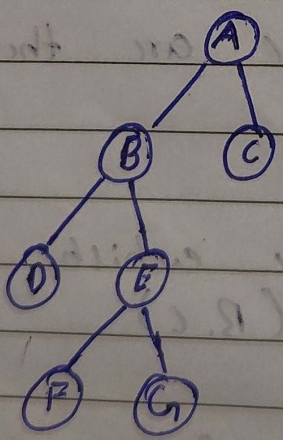
Types of Binary Tree.

- Full Binary Tree
- Perfect Binary Tree
- Complete Binary Tree
- Degenerate Binary Tree
- Expended Binary Tree
- Binary Search Tree
- Heap Tree.

1. Full Binary Tree.

A binary tree said to be full binary tree if it each node contain exactly 2-node or 0-node.

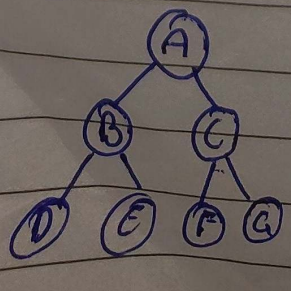
eg:



2. Perfect Binary Tree

A binary tree in which every internal node has exactly two children and all leaf node are at same level

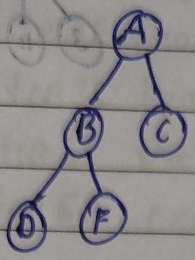
eg:



3. Complete binary tree

A binary tree said to be Complete binary tree if its all the levels are full except the last level

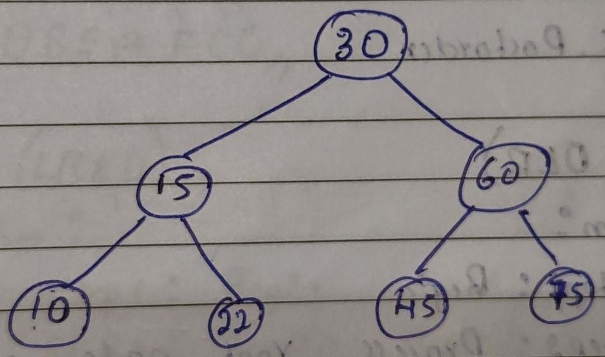
eg:



4. Binary Search Tree

A tree 'T' said to be the binary Search tree if it's all the nodes of left sub tree are Smaller and right Sub-tree are the greater from the root node.

eg:



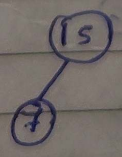
Creating Binary Tree.

15, 7, 3, 11, 19, 16, 25

Solution:

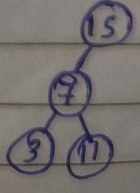
Step1

Step2

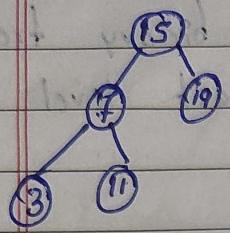


Step3

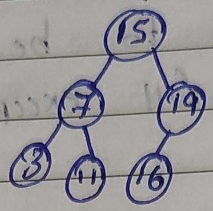
Step 4



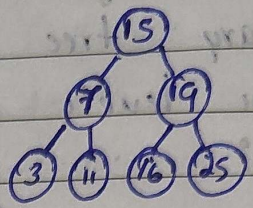
Step 5



Step 6



Step 7



2 Tree Traversal

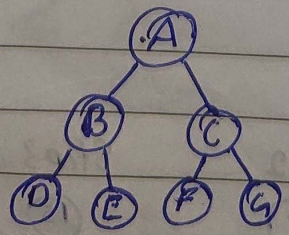
- Tree traversing means visiting each node of a tree
- Tree traversing can be classified into 3-types
  - i. Pre-order
  - ii. Inorder
  - iii. Postorder

1. Pre-order (DLR)

Algorithm:

Step1 : Begin
Step2 : Process root node of given tree
Step 3 : Process the left node.
Step 4 : Process the right node.

eg:-



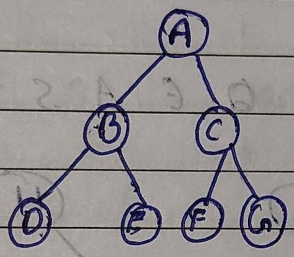
ABDECFG //

2. Inorder (LDR)

Algorithm:

Step 1: Bigin
Step 2: Process the lef Sub tree element
Step 3: Process the root node
Step 4: Process the right Sub-tree content
Step 5: exit

eg:



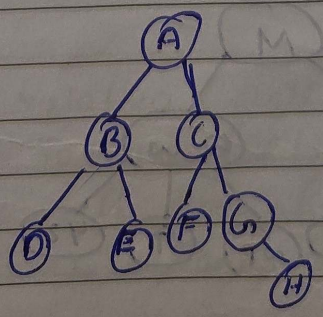
DBEAFCG//

3. Postorder (LRD)

Algorithm:

Step 1: Begin
Step 2: Process the left-Sub-tree element
Step 3: Process the right Sub tree element
Step 4: Process the root node.
Step 5: Exit

Eg:



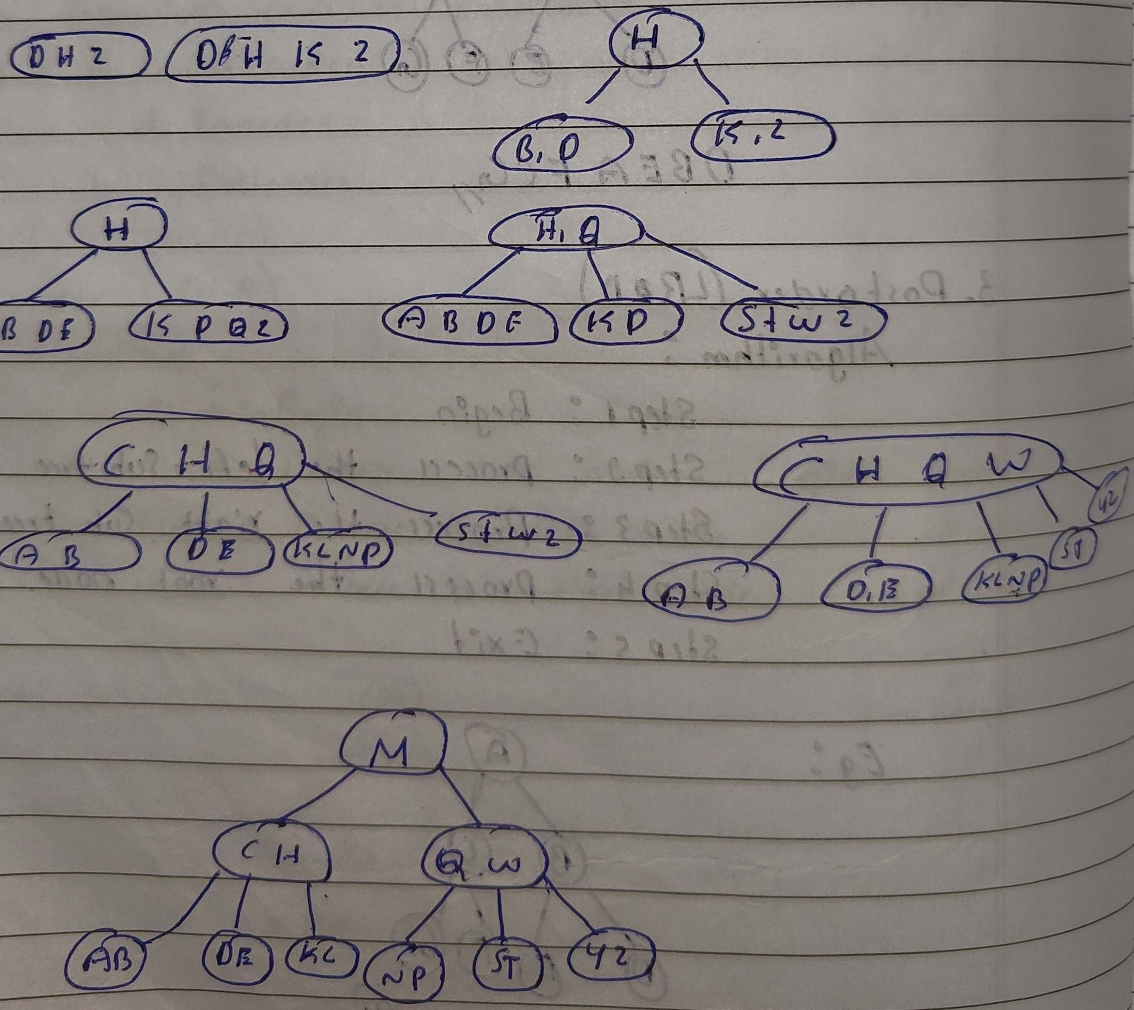
~~DEB F HG C A~~
DEBFHGCA

3 B-Tree.

A tree is known as balanced M-way tree. It is used in external sorting to reduce ~~the~~ disk access, Several Conditions of the tree must be true.

* The height of the tree must be kept minimum
* There must be no empty subtree above the leaves of the tree
* Leaves of the tree must all at the same level.

Eg: D H Z K B P Q E A S W T C L N Y M



AVL Tree

- It is Binary Search tree
- BF → Hieght of left Subtree − hieght of right Subtree = {-1, 0, 1}
- Dublicate elements are not allowed.
- The balance factor plays an important role in the insation and deletion of elments

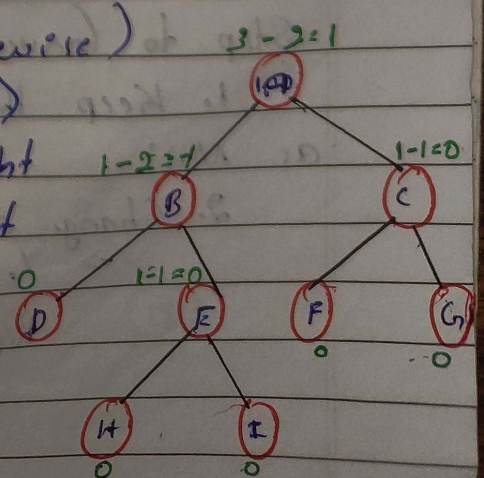
Charecterstics of AVL

1. If the BF of a node is 0, it means that the height of the left Sub-tree and height of the right Sub tree is equal.

2. if the BF of node is 1, it means that the height of the left-Sub tree is greater than the height of the ~~left~~ right Sub-tree.

3. if the BF of a node is -1, it means that the height of the left Sub-tree is lesser than the height of the right Sub tree.

4. A non AVL tree can be converted to an AVL tree by performming diffrent types of rotation on a given tree.
   1. Left rotation (Anti-clockwise)
   2. Right rotation (clockwise)
   3. Rotate left and Rotate right
   4. Rotate right and Rotate left



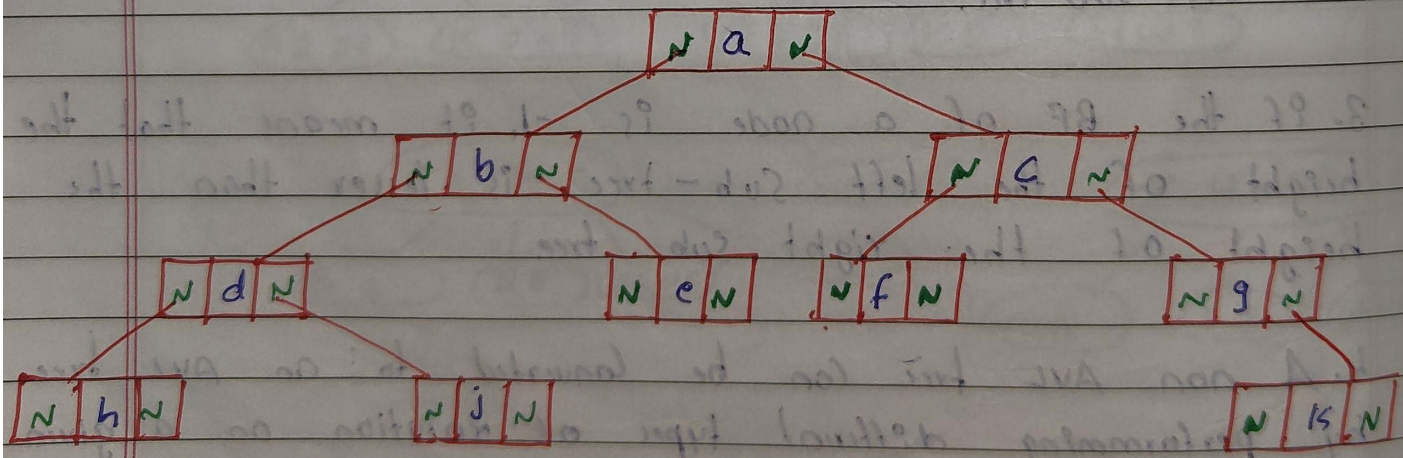
5 Threaded Binary Tree.

A binary tree is threaded by making all right child pointers that would normally be Null point to the Inorder Successor of the node and left child pointers that would normally be Null point to the Inorder Predecessor of the node

Threads are pointers to the predecessor and Successor of the node according to the Inorder traversal and the tree whose nodes use threads are called T.B.T.

Types of Threaded Binary Tree

- Single Threaded
- Double Threaded.



Step to Convert:

1. Keep the leftmost and the rightmost Null pointers as Null
2. Change all other Null Pointers as
   => left Pointer = Inorder predecessor
   Right Pointer = Inorder Successor.

6 Red-Black Tree

-> In this tree, every node is either Red or Black.

-> In this red-black tree, root node and leafnodes are Blacks.

-> In this Tree, if a node is red, then its Childrens are blacks.

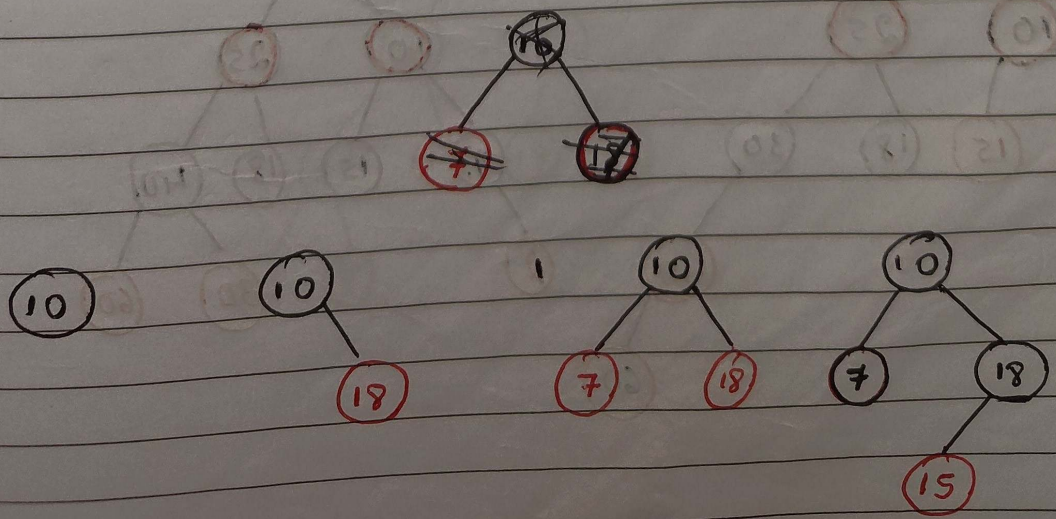
-> Every path from root to leaf has the same number of Blacks nodes.

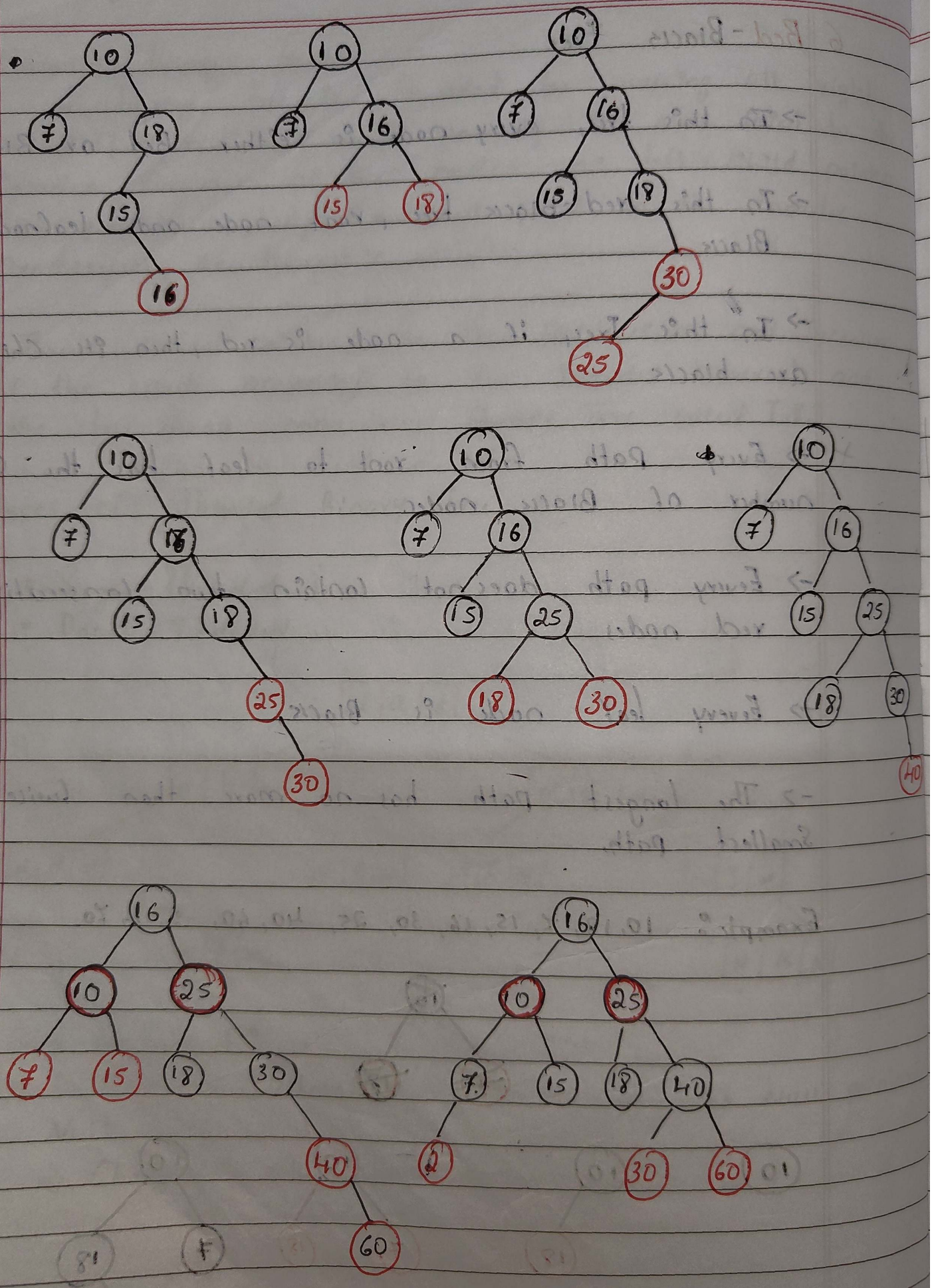
-> Every path does not contain two consecutive red nodes.

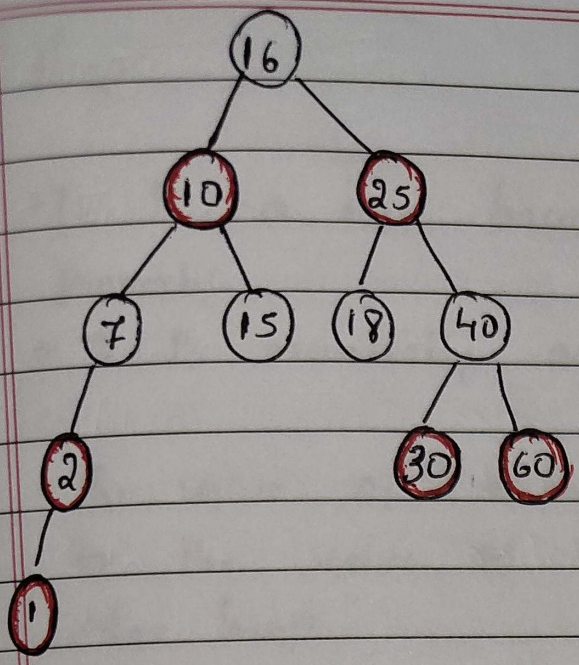
-> Every leaf node is Blacks.

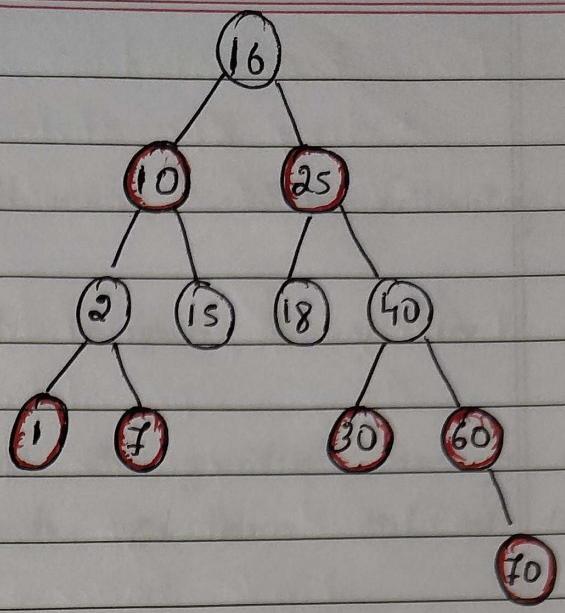
-> The longest path has no more than twice the smallest path.

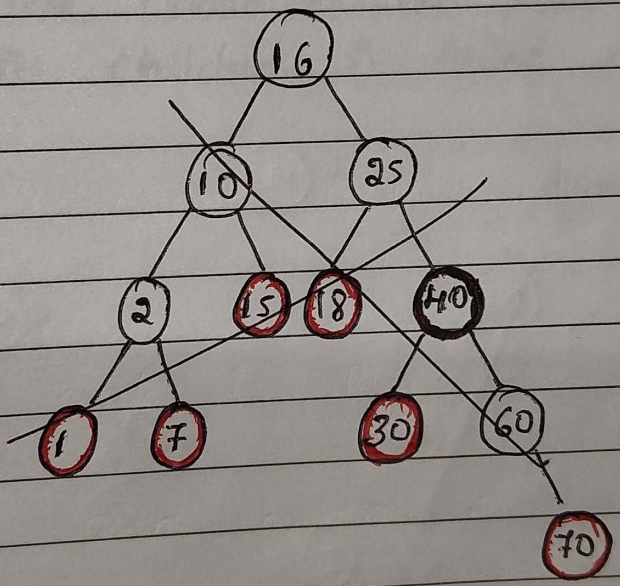
Example: 10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, 70.

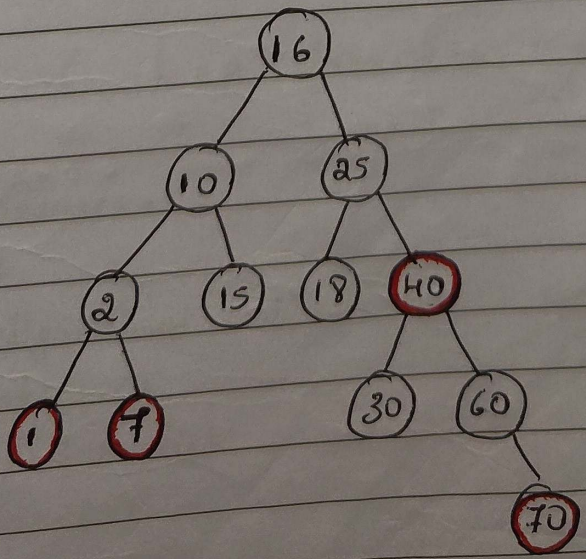


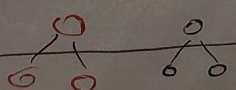

10
7
18
15
16
10
7
16
15
18
10
7
16
15
18
30
25
10
7
18
15
18
25
30
10
7
16
15
25
18
30
10
7
16
15
25
18
30
40
16
10
25
7
15
18
30
40
60
16
10
25
7
15
18
40
2
30
60


16
10
25
7
15
18
40
2
30
60
1


16
10
25
2
15
18
40
1
7
30
60
70


16
10
25
2
15
18
40
1
7
30
60
70

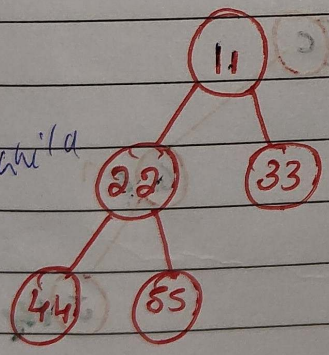

16
10
25
2
15
18
40
1
7
30
60
70



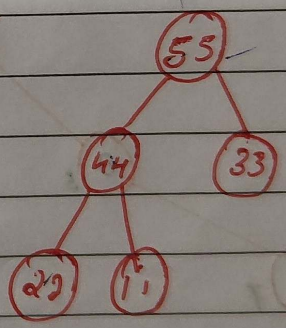
7 Heaps

- It is a tree based Data structure with following property.
- It is essentially an almost Complete Binary tree.
- The value of each Parent node N is greater than or equal to the value ~~at each~~ of the children ~~of N~~ is called Max heap.
- Value of Parent node is less than or equal to either of its children. is called Min heap.

Min :



Max :



Application of heap data structure.

1. Priority Queues.
2. Heap Sort
3. Graph Algorithms
4. Order Statistics
5. Merging Sorted Lists
6. Load Balancing.
7. Data Compression.
8. Median Maintainance.

2023

8 Tree Representation.

- Array Representation.
- Linked Representation.

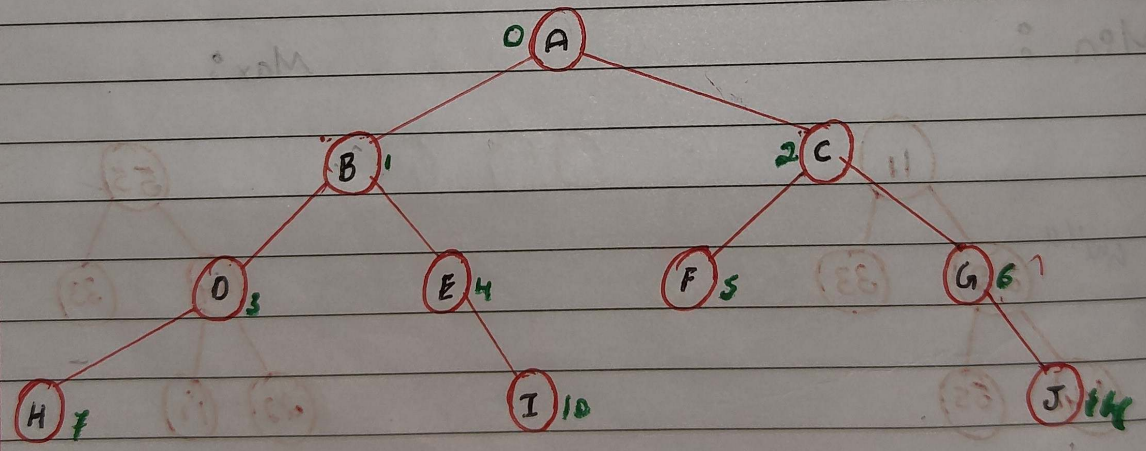
1. Array Representation.

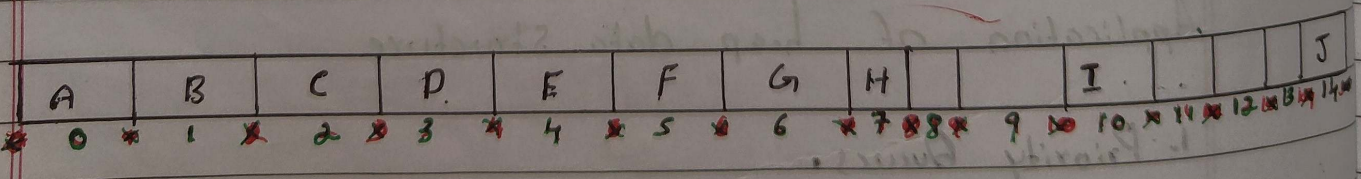
Array representation of a tree stores the tree nodes in an array such that parent : child relationships are maintained through index calculations.

Node index = i

left child index = $2*i+1$

Right child index = $2*i+2$





Declare array of size $2^{3+1}-1 = 15$ $\left(2^{d+1}-1\right)$ (d = depth)

Char a[15]

2. Linked Representation.

In Linked Representation. Each element is represented by a node that has two fields (left child & right child) plus an element field.