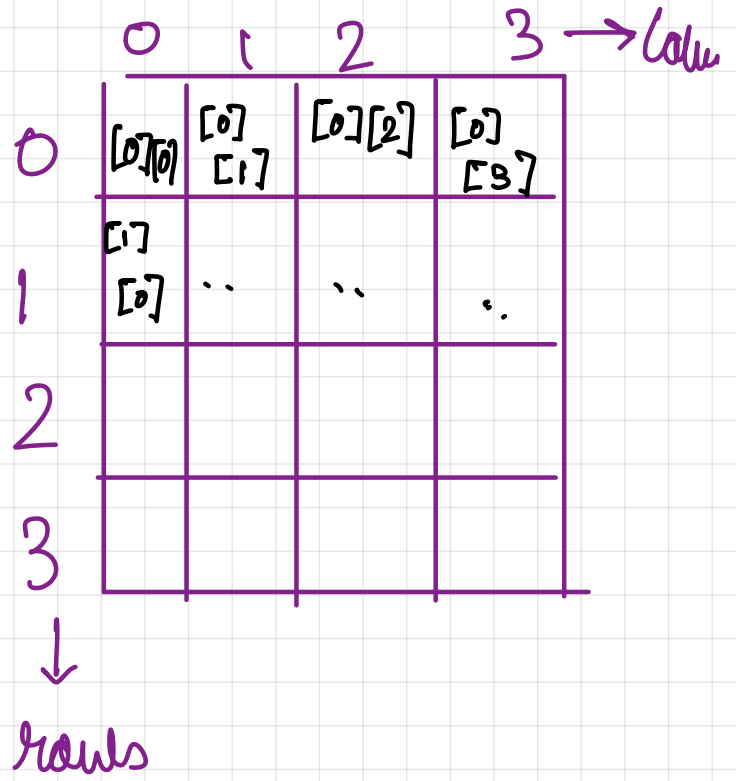
Aditya Dhanaraj Kundu

Data Structures And Algorithms.

```
#include <stdio.h>
int main()
{ int arr2d[5][5], i, j, k, l;
  for(i=0; i<5; i++)
    for(j=0; j<5; j++){
  printf("arr 2D[%d][%d]: ", i+1, j+1);
  scanf("%d ", &arr 2D[i][j]);
    }
    for(k=0; k<6 : k++)
    for(l=0; l<6; l++)
  { printf("arr 2d[%d][%d] = %d ", k, l, arr 2d[i][j]);
  }
    return 0;
}
```



Problem : Write a C program which will take two polynomials and will deliver (i) addition (ii) subtraction (iii) multiplication of them.

$P_1(x) = a + bx + cx^2 + dx^3 \ldots$

$P_2(x) = m + nx^3$

$= m + 0x^1 + 0x^2 + nx^3$

$A[4] = \{a, b, c, d\}$

$B[4] = \{m, 0, 0, n\}$

$Sum(P_1 + P_2) = (a+m) + bx + cx^2 + (d+n)x^3$

$Mul(P_1, P_2) = am + bmx + cmx^2 + dmx^3 + anx^3 + bnx^4 + cnx^5 + dnx^6$

$= am + bmx - cnx^2 + (dm+an)x^3 + bnx^4 + cnx^5 + dnx^6$

$Sub(P_1, P_2) = (a-m) + bx + cx^2 + (d-n)x^3$

25/Aug/2022

① Write a C program to check a matrix is sparse or not.

② Display a sparse matrix in i) row major and ii) column major order

③ Evaluate the transpose of a sparse matrix.

④ Multiply two sparse matrix.

**

Suppose $A_{m \times n}$ matrix. If more than $\left(\frac{M+N}{2}\right)$ are zero elements then the matrix is called sparse matrix.

Eg: For a 3×3 matrix,

$$\begin{matrix} 0 & 0 & 5 \\ 4 & 0 & 0 \\ 0 & 0 & 2 \end{matrix} \Rightarrow \text{Sparse matrix.}$$

- Row Major Order :-

	C_1	C_2	C_3
R_1	1	2	3
R_2	4	5	6
R_3	7	8	9

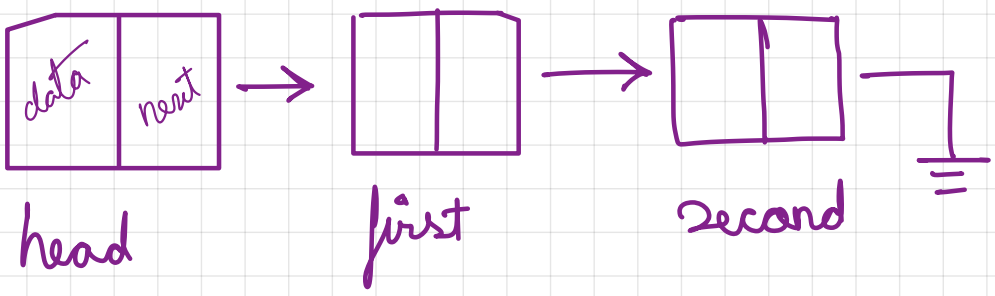
{(1,2,3), (4,5,6), (7,8,9)}

```
for (i=1 to m)
{ for (j=1 to n)
{ printf (matrix[i][j]); }
```

- Column Major Order :-

```
for(j=1 to N) {
for( i=1 to m){
print (matrix[j][i]); }
}
```

Linked List :- [01/Sep/2022]




```
struct node {
int data;
struct node * next;
}
```

initializing nodes :-

```
Struct node * head = "Null" ;
Struct node * first = "Null" ;
Struct node * second = "Null" ;
```

Allocation of memory :-

```
Struct node* head = (struct node*) malloc (size of (struct node));
for first and second, same as head ;
```

Inserting values in the linked list :-

```
head -> data = 100;
head -> next = first;
first -> data = 200;
first -> next = second;
second -> data = 300;
second -> next = "NULL";
```

Assignment: Q. Write a C program to (i) insert an element in linked list (ii) search a particular element (iii) deletion of element from a linked list.

Q① What are the operations normally performed on any linear structure, whether it can be an array or linked list?

Ans: Traversal :-

Proecessing each element in the list.

Search :-

Finding the location of the element with a given value or record of a given key.

Insertion : Inserting a new element into the list.

Deletion : Deleting an element from the list.

Sorting : Arranging data elements in ascending or descending order.

Merging : Combining two lists into one single list.

② An automobile company uses an array auto to record the number of automobile sold each year. From 1932 to 1984. So rather than begining the index at 1 it is more useful to start with 1932.

Ans: Auto[K] = No of automobiles sold in year K

Then LB = 1932 ⇒ Length = UB - LB + 1

UB = 1984

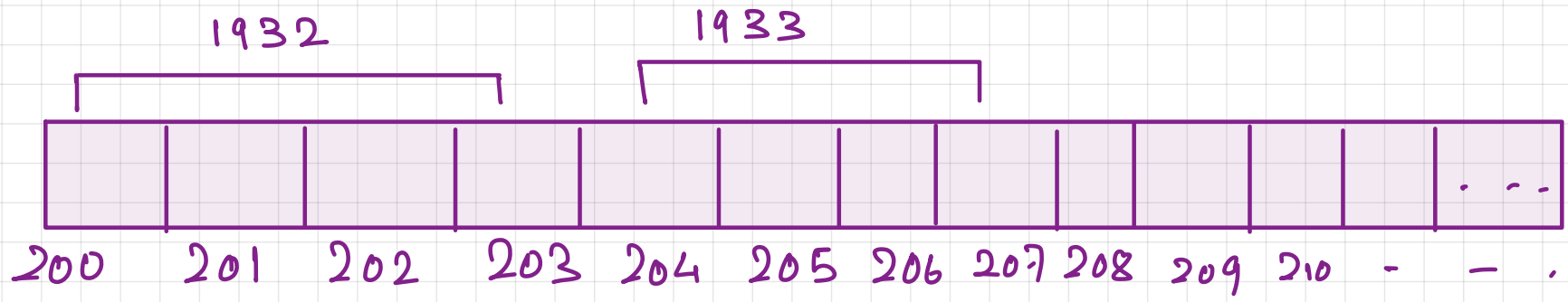
LOC(Auto[1965])

= Base(Auto) + W(1965 - lower bound)

= 200 + 4(1965 - 1932)

= 332

LOC(Auto[1932]) = 200



Traversing Algorithm :-

(Traversing a Linear Array). Here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation process to each element of LA.

① Initialize a counter, set K = LB
② Repeat step ③ and ④ while K ≤ UB
③ Visit element, Apply process to LA[K]
④ Increase counter, set K = K+1,

[End of Step 2 loop]

⑤ Exit

Inserting to a Linear Array :-

Here LA is a linear array with N elements and K is a positive integer such that K≤N

Insert (LA, K, N, ITEM), This algorithm inserts an element ITEM into the K^{th} element in LA.

① [Initialize counter] set J = N
② Repeat step ③ and ④ while J ≥ K

③ [Move J^{th} element downward] set LA[J+1] := LA[J]

④ [Decrease counter] set J : J-1

[End of step 2 loop]

⑤ [Insert element] set LA[K] := ITEM

⑥ [Reset N] set N : N+1

⑦ Exit.

12/Sep/2022

Deletion from a linear array :-

① Set item = LA[K]

② Repeat for J = K to N-1

{ Move J+1st element upward
Set LA[J] = LA[J+1] }

End of loop

③ [Reset the no. of N of elements in LA]
Set N = N-1

④ Exit

LA = linear array.

N = no. of elements

K = any +ve integer such that $K \leq N$

N-1 is the end of the loop.

Bubble sort algorithm :-

Bubble (Data, N)

① Repeat Step 2 and 3 for K = 1 to N-1

② Set Ptr = 1 [initialize pass point PTR]

③ Repeat while PTR $\leq$ N-1 : [execute pass]

a) If Data [PTR] > Data [PTR+1] then,
interchange Data [PTR] and Data [PTR+1]
[End of IF structure]

b) Set PTR := PTR+1
[End of inner loop]
[End of step 1 outer loop]

④ Exit

$$f(n) = (n-1)+(n-2)+\cdots+2+1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n)$$
$$= O(n^2)$$

Q. Explain the steps of Bubble sort using diagram?

Ans: Suppose a list of numbers A[1], A[2], A[3] ... A[N] is in the memory. Then the bubble sort algorithm actually works in

Step-① : Compare A[1] and A[2] and arrange them in the desired order. So that A[1] < A[2]. Then compare A[2] and A[3] and arrange them. So that A[2] < A[3]. Then Compare A[3] and A[4] and arrange them so that A[3] < A[4]. Continue until we compare A[N-1] with A[N] and arrange them so that A[N-1] < A[N]

{ Observe that Step 1 has (n-1) no of comparison, during this the largest element is bubbled up to nth position and sticks there. So in the end A[N] will contain the largest element }

Step -2 : Repeat Step 1 with one less comparison that is now we stop after we compare and possibly rearrange A[N-2] and A[N-1]. Step 2 includes (n-2) no of comparison and when it is completed the second largest element of the array will occupy the A[N-1] position.

Step -3 : Repeat Step 1 with two lesser comparisons that is we stop after we compare and possibly rearrange A[N-3] and A[N-2]

Step N-1 : Compare A[1] and A[2] and arrange them so that A[1] < A[2].

For (N-1) steps, the list/array will be sorted in ascending order.

Linear Search Algorithm :-

[Linear (Data, N, Item, Loc)]

Here Data is a linear array with N elements and item is a given item of information. This algorithm finds the location Loc of item in Data or sets. Loc := 0 if the search is unsuccessful.

① [Insert item at the end of data] set Data[N+1] := ITEM

② [Initialize counter] set LOC := 1

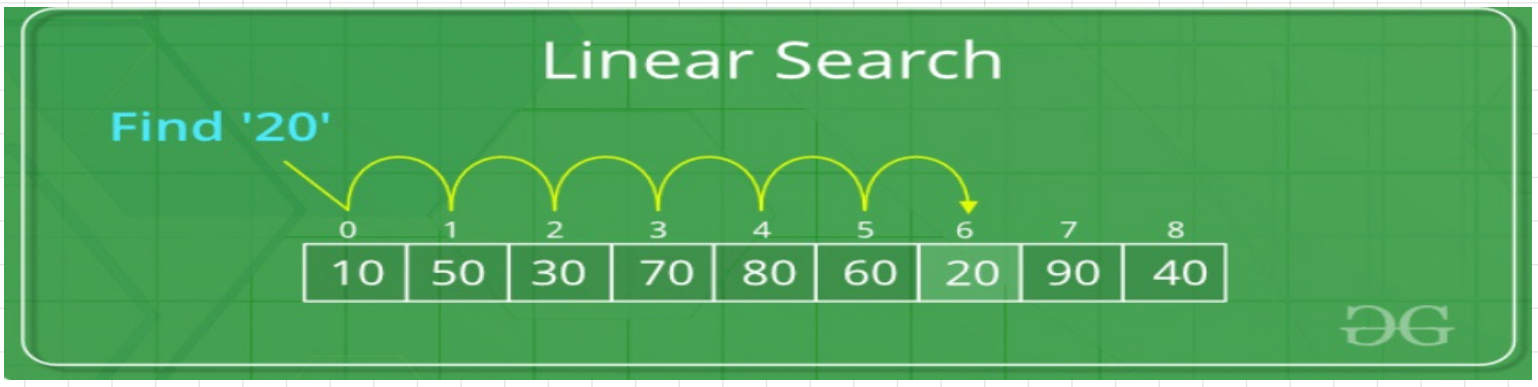
③ [Search for item], Repeat while Data[LOC] ≠ ITEM

Set LOC := LOC+1 :

[End of loop]

④ [Successful] if LOC = N+1, then : set LOC := 0

⑤ Exit



Complexity of an Algorithm :-

14/Sep/2022

An algorithm is a well defined list of steps for solving a particular problem. The time and space it uses, are two major measures of efficiency of algorithms. The Complexity of an algorithm M is the function $f(n)$ which gives the running time and storage space requirements of an algorithm in terms of size n of the input data the storage space required by an algorithm is simply the multiple of the data size n. The term Complexity refers to the running time of an algorithm.

Average Case :-

Here we assume that ITEM does appear in data and that it equally likely to occur at any position in the array. Accordingly the no. of comparisons can be of any of the numbers 1, 2, 3 . . . n and each no. occurs with probability $P = 1/n$, Then -

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \ldots + n \cdot \frac{1}{n}$$

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

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

This agrees with our initiative dealing that the avg no. of comparisons needed to find the location of ITEM is approximately equal to the no. of elements in Data list.

Worst Case :-

Clearly the worst case when ITEM is the last element in the array Data or is not there at all. In either situation we have,

$$C(n) = n$$

Accordingly, $C(n) = n$ is the worst case completely of the linear search algorithm.

- The big O notation defines an upper bound function $g(n)$ for $F(n)$ which represent the time/space complexity of the algorithm on an input characteristic n. There are other asymptotic notation such as ℓ, Ⓗ, O, which also serves to provide bound for the function $f(n)$.
- To indicate the convinience of this notation, we give the complexity of certain well known searching and sorting algorithms.

→ linear search $O(n)$ → Bubble sort $O(n^2)$

→ Binary Search $O(\log n)$ → Merge sort $O(n \log n)$

Theta Notation (θ) :-

The theta notation is used when the function $F(n)$ is bounded both from above and below by the function $g(n)$.

PTO

Definition : $f(n) = \theta(g(n))$ {read as F on n-theta of g of n} if there exist any two positive constants C_1 and C_2 and a positive integer n_0. such that $C_1|g(n)| \leq |f(n)| \leq C_2|g(n)|$ for all $n \geq n_0$.

From the definition it implies that the function $g(n)$ is both an upper and lower bound for the function $f(n)$ for all the values of n_1, $n \geq n_0$. In other word $F(n)$ is such that,

$F(n) = O(g(n))$ and $F(n) = \Omega(g(n))$

For $F(n) = 15n + 9$, since $F(n) > 18n$ and $F(n) \leq 27n$ for $n \geq 1$, we have $F(n) = \Omega(n)$ and $F(n) = O(n)$ respectively for $n \geq 1$. Hence $F(n) = \theta(n)$. Again, $16n^2 + 30n - 30 = \theta(n^2)$ and $7.2^n + 30n = \theta(2^n)$

Omega Notation :-

19/Sep/2022

The omega notation is used when the function $g(n)$ defines a lower bound for the function $f(n)$.

- **Definition :-**
  - $F(n) = \Omega(g(n))$ {read as F of n is omega of G of n} if there exist a positive integer number and a positive number m. Such that $|F(n)| \geq M(g(n))$, for all $n \geq n_0$
  - For $F(n) = 18n + 9$, $f(n) > 18n$ for all n, hence $F(n) = \Omega(n)$

Also for $F(n) = 90n^2 + 18n + 6$, $F(n) > 90n^2$ for $n \geq 0$ and therefore $f(n) = \Omega(n)^2$.

- For $f(n) = \Omega(g(n))$, $g(n)$ is a lower bound function and there may be several such functions which is almost as large function of n as possible such that the definition of Ω is satisfied is chooren as $g(n)$. For example $f(n) = 5n+1$ lead to both $f(n) = \Omega(n)$ and $f(n) = \Omega(1)$. However we never conrider the latter to be correct since $f(n) = \Omega(n)$ represent the largest possible function of n satisfying the definition of Ω and hence is more informative.

Q. What is Recursion?

26/Sep/2022

Ans:- In C it is possible to function call themselves. A function is called recursive if the statement within the body of the function calls the same function itself. Recursion thus is the process of defining something in term of themselves. Eg: Factorial Calculation.

Conditions: ① The program will not not continue to run indefinately when the following properties hold. There must be certain criteria (Base Case), when the function does not call itself.

② Each time the function calls itself, it should be closer to the base class.

③ Recursive Call (calling ourselves)

TOWER OF HANOI :-

Algorithm : TOWER (N, BEG, AUX, END) [Recursive approach]

Where, n = no of disk

① If N=1 Then:

a) write : BEG → END

b) Return

[End of if structure]

② [Move n-1 disk from pay BEG to pay AUX]

[all TOWER (N-1, BEG, END, AUX)]

③ Write : BEG → END

④ [Move n-1 disks from pay AUX to pay END]

Call TOWER (N-1, AUX, BEG, END)

⑤ Return

TOWER OF HANOI (Without Recursion) :-

Consider again the towers of HANOI problems. Procedure 6.11 is a recursive problem for n disk. We translate the procedure into a non recursive solution. In order to keep the steps analogous, we label the beginning statement.

1) Set TOP = NULL

2) If N=1, then:

(a) Write BEG = END

(b) Go to Step 5

[End of y structure]

3) [Translation of "call TOWER (N-1, BEG, END, AUX)"]

ⓐ [Push current values and new return addresses onto stack]

ⓑ Set TOP := TOP+1

ⓒ Set STN [TOP := N, STBEG (TOP) := BEG,

STAUX [TOP] := AUX, STEND [TOP] := END,

STADD [TOP] := 3

ⓓ [Reset Parameters] Set, N := N+1, BEG := BEG, AUX := END

END = AUX

ⓔ Go to step 1.

④ Write : BEG → END.

⑤ [Translation of "call TOWER (N-1, AUX, BEG, END)"]

ⓐ [Push current values and new return addresses onto stack]

(i) Set TOP := TOP+1

(ii) Set STN [TOP] := N, STBEG [TOP] := END

STADD [TOP] := 5

ⓑ [Reset Parameters]

Set, N := N+1, BEG := AUX, AUX := BEG, END := END

ⓒ Go to step 1.

⑥ [Translation of "return"]

a) If TOP := NULL, then : Return

b) [Restore top values on stack]

(i) Set, N := STN [TOP], BEG := STBEG [TOP]

ADD : STADD [TOP]

(ii) Set, TOP := TOP-1

c) Go to step ADD.

Deletion Algorithm :-

(1) Set ITEM := INFO [START] [This takes the data in the first node]

(2) Delete first node from the list.

(3) Process ITEM.

(4) Exit

Addition Algorithm :-

(1) Transverse the one way list until finding a node X whose priority number exceeds N. Insert in front of node X.

(2) If no such node is found, insert ITEM, in the last element of the list.

Infix, Prefix, Postfix :-

• Application of Stack Data Structure :-

1. Page visited history in a web browser
2. Undo sequence in a text editor.
3. Conversion of one form of expression to another form.
4. Use in systematic memory management.
5. Auxillary data structure for algorithms
6. Component of other data structures.

Direct Use

indirect Use

• Application of Queue data Structure :-

1. Waiting list
2. Access to shared resources.
3. Multiprogramming
4. Auxilary data structure for algorithms.
5. Component of other data structure.

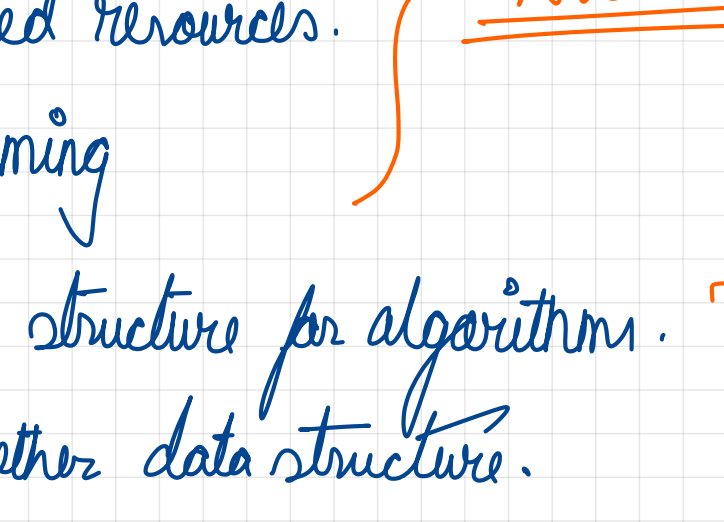
Direct application

Indirect application

Tree :-

02/Nov/2022

A binary tree is defined as a finite set of elements called nodes. An empty tree is called null tree. Tree contain distinguished node R called root of tree.



left successor of node A

Successor & Precedessor

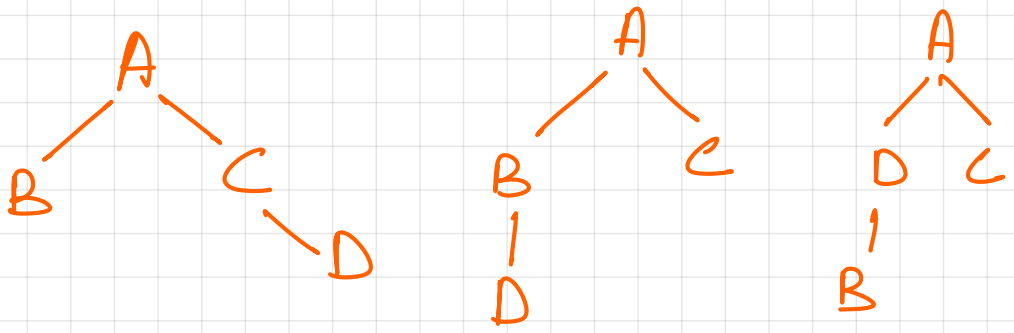
BDEF → left subtree

CGHIJK → right subtree

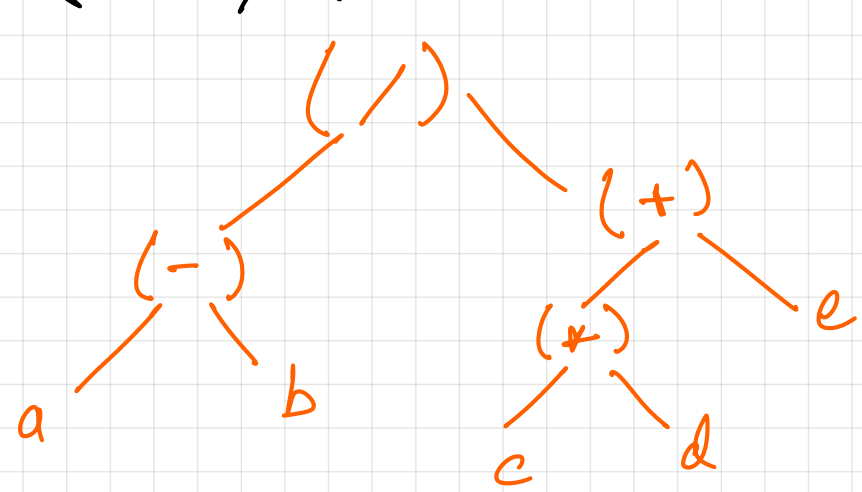
In a binary tree every node can have atmost 2 children. Therefore any node N in a binary tree has either 0, 1, 2 as successor.

Q. Consider A, B, C, D try to make binary tree?

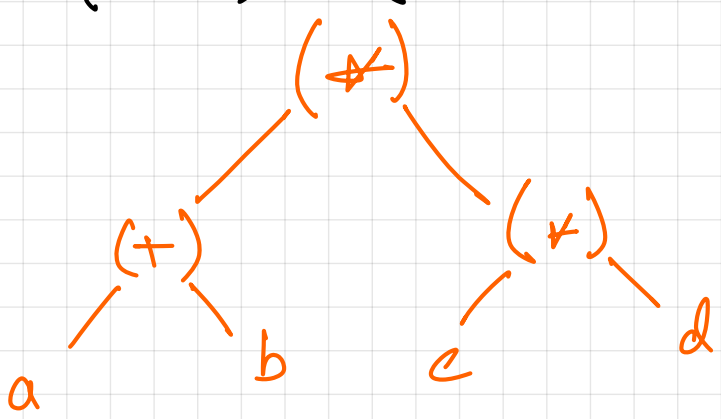
⇒



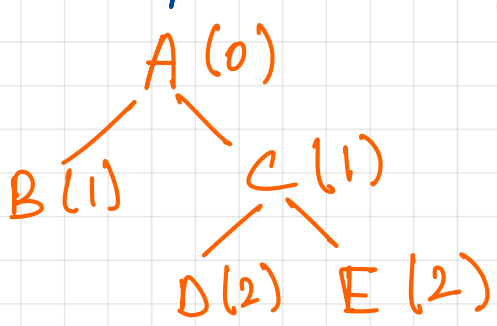
Q. $E = (a-b)/((c*d)+e)$



$E = (a+b) - (c*d)$



- **Branches And Edges** :—
  - The line drawn from a node of tree 'T' to a successor is called an edge.
  - A sequence of consecutive edges is called a path.
  - Terminal node of a tree is called a leaf (L).
  - A path ending in a leaf is called branch.



Level number: The root of a tree is assigned to a level number '0'

A is called parent. In B & C, B is left child and C is right child & B and C are siblings to each other.

• Binary Search Tree :-

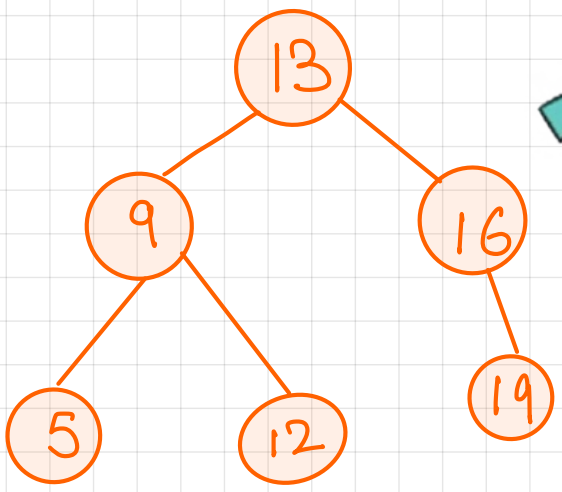
An important special kind of binary tree is the binary search tree. (BST).

In a BST each node stores some information including a unique key value and some associated data. A binary tree is a binary search tree if and only if for every node 'n' in the tree —

- All keys in n's left subtree < key in n right
- All keys in n's right subtree > key in n left.

The reason binary search tree are important is because the following operation can be implemented efficiently using a Binary Search tree —

(i) Insert a key value in the tree
(ii) determine whether a key value is present in the tree. (searching)
(iii) Removing a key value from the tree.
(iv) Print all of the key value in a sorted order.



Q. In this given Binary Search tree, perform a searching operation for 12.

Soln: Searching for 12. Let the node n = 13.
12 < 13, so go to the left subtree of 13. Now node n = 9, and 12 > 9. So go to the right subtree. And 12 is present there.

Q. What if we search for 15?

Soln: 15 > 13. (n = 13). So go to right subtree. (n = 16). Now 15 < 16, so go to left sub tree, and it does not exist, thus search fails and returns false.

Properties & Operations of Binary Search Tree :-

- Each node contains 1 Key.
- The Key in the left subtree are always less than it's parent node.
- The Key in the right subtree are always greater than it's parent node.
- Duplicate node Keys are not allowed.

Inserting a node :-

A naive algorithm for inserting a node in a Binary Search Tree is that we start from the root node, if the node to be inserted is less than the root node we go to the left child, otherwise we go to the right child. We continue this process until the first null pointer.
We then insert the node as a left or right child of the leaf node depending on greater than or less than leaf node.

Algorithm

Insert (N, T) = N if T is empty
= insert (N, T left), if N < T
= insert (N, T right), if N > T

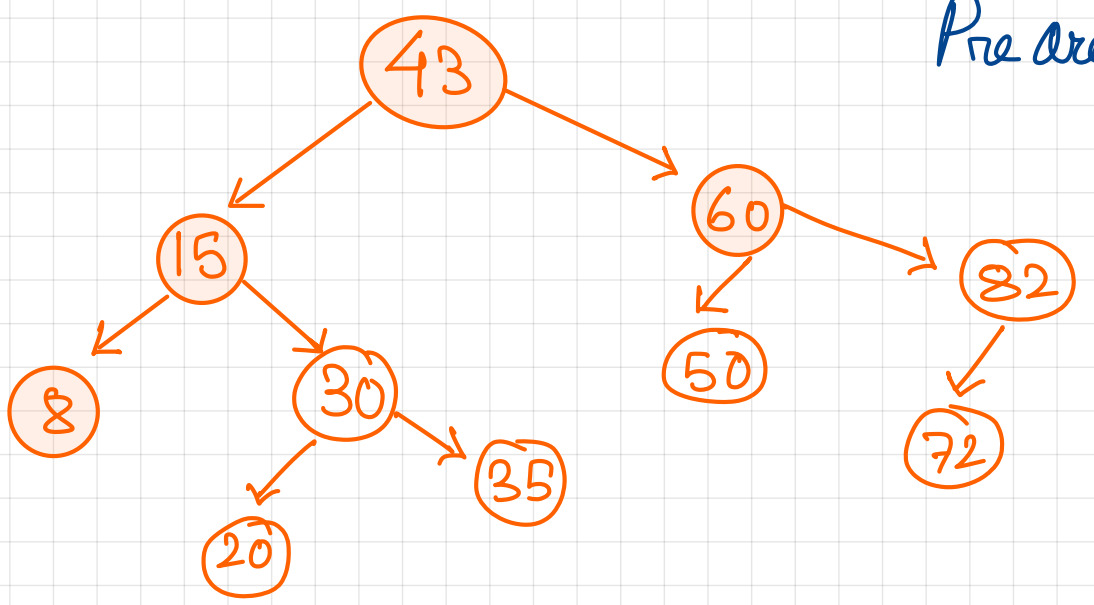
- A recursive algo for inserting a node in a BST.

Search (N, T) = false if T is empty
= true if T = N
= search (N, T left) if N < T
= search (N, T right) if N > T

To traverse a non empty tree, the following steps are in order followed recursively.

In order

(i) Visit the root
(ii) Traverse the left subtree
(iii) Traverse the right subtree.



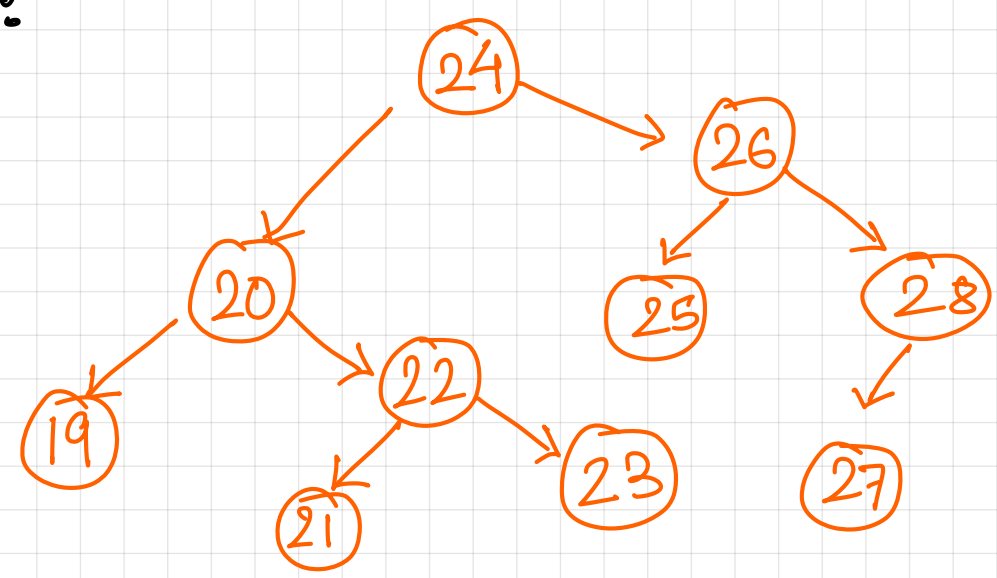
Pre order: 43, 15, 8, 30, 20, 35, 60, 50, 82, 72

Post order: 8, 20, 35, 30, 15, 50, 72, 82, 60, 43

In Order: 8, 15, 20, 30, 35, 43, 50, 60, 72, 82

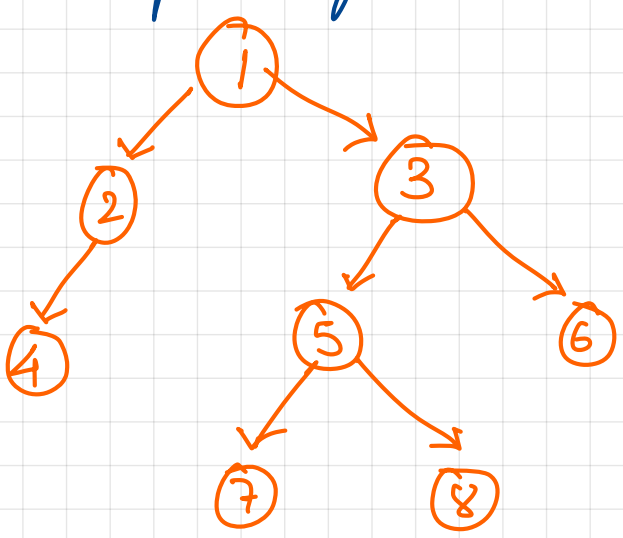
Q. In Order: 19, 20, 21, 22, 23, 24, 25, 26, 27, 28

Soln:



Pre-order :-

(i) Visit left sub tree before right sub tree.



Pre-order: 1, 2, 4, 3, 5, 7, 8, 6

- **Post-Order :-**

(i) Traverse the left sub tree.

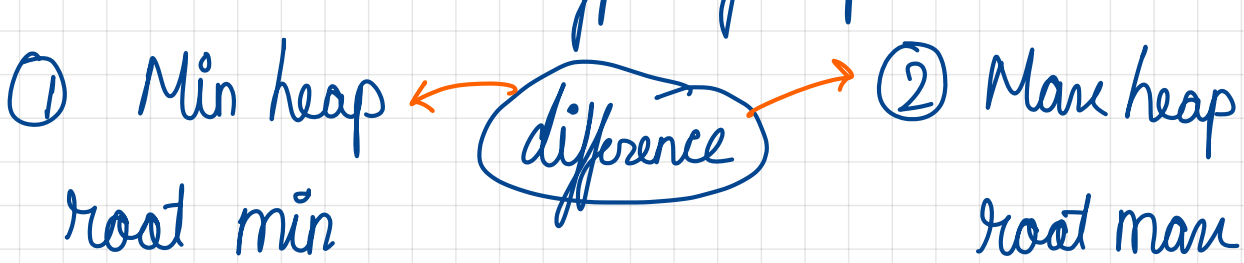
(ii) Traverse the right sub tree.

(iii) Visit Root.

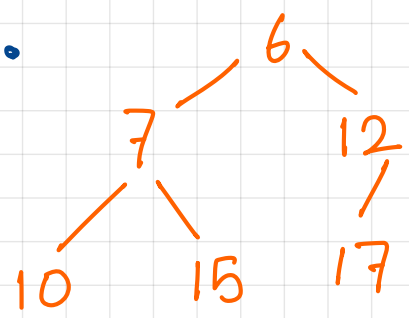
21/Nov/2022

- **Heap :-**

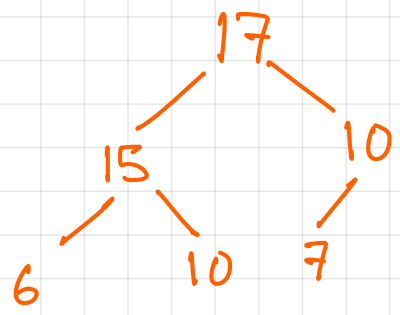
A heap is a special tree based data structure in which tree is complete binary tree. There are two types of heaps →



Eg.



insertion
deletion
complexity



- **Radix Sorting :-**

Application of stack and queue.

Push pop algorithm

Delete element of a queue.

ADT

Dynamic memory allocation.

- **Collision in hash table :-**

A collision occurs when more than one value to be hashed by a particular hash function.

- **Hash function :** A collision occurs when more than one value to be hashed by a particular hash function in the same slot or data structure being generated by the hash function.

- **Hash table :-**

It is an abstract data type that maps keys to values. A hash table uses a hash function to compute an index also called a hash code. into an array of buckets or slots from which the desired value can be found during lookup the key is hashed and the resulting hash indicates where the corresponding value is stored.

- **AVL tree :-**

An AVL tree is a self balancing binary search tree, where the difference between the left and right subtree can't be more than 1 node.