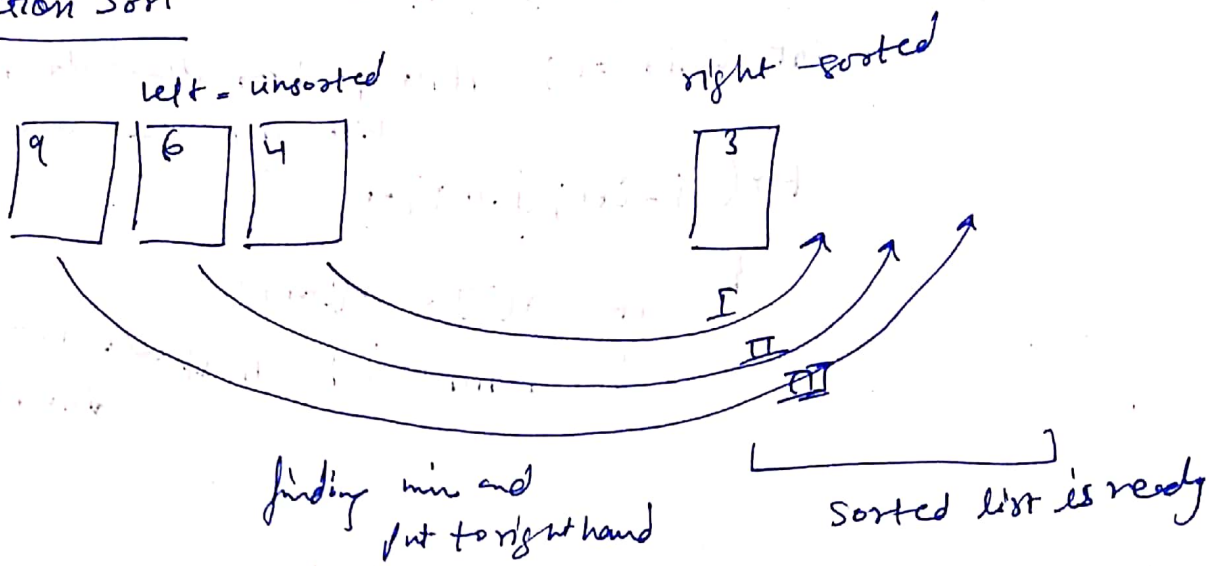
SORTING Algos

Selection Sort



⇒ | 2 | 7 | 4 | 1 | 5 | 3 |

find min ↑

⇒ Now replace it with index [0] element

⇒ | 1 | 7 | 4 | 2 | 5 | 3 |

⇒ find min

⇒ Now swap it with index [1] element.

⇒ | 1 | 2 | 4 | 7 | 5 | 3 |

⋮

⇒ | 1 | 2 | 3 | 4 | 5 | 7 |

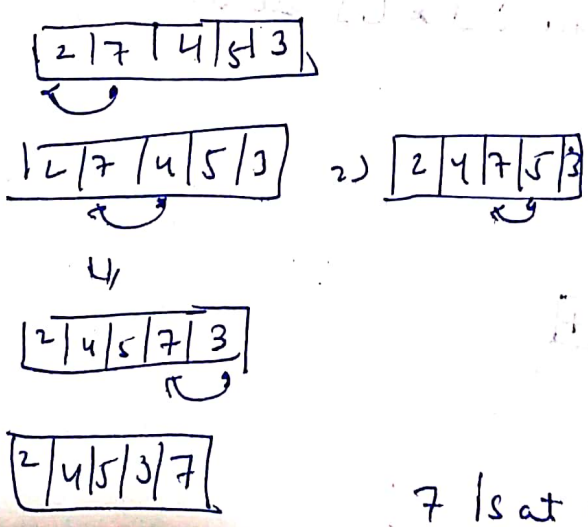
$T = O(n^2)$

```
void SelectionSort( int A[], int n)
{
    for(i=0; i<n-1; i++)   // we need to do n-2 passes
        int imin = i;      // ith position elements from i
                           till n-1 are candidates
        for( j=i+1; j<n; j++)
        {
            if (A[j] < A[imin])
            { imin = j;    // update index of
                           minimum element
            }
        }
        int temp = A[i];
        A[i] = A[imin];
        A[imin] = temp;
    }
}
```

finding min element

swap A[min] & A[i]

BUBBL SORT



7 is at his appropriate position

Time Complexity :-

$T(n) = (n-1) * (n-1) * c$

$= cn^2 - 2cn + 1$

$= O(n^2)$

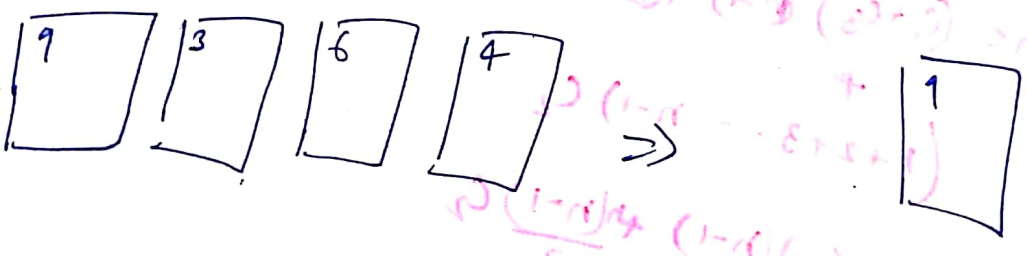
Best case = $O(n)$

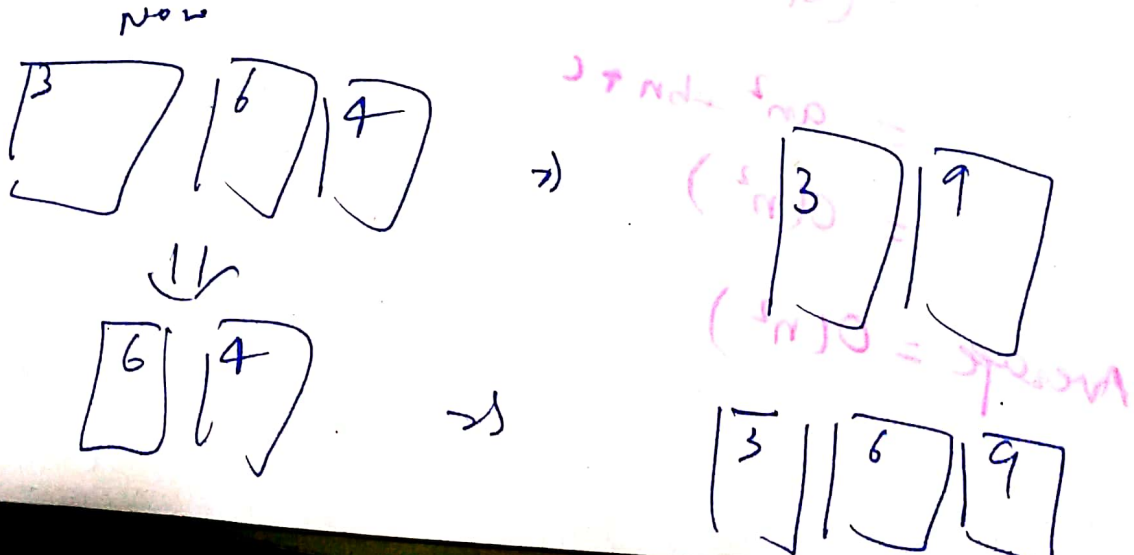
Average case = $O(n^2)$

Worst = $O(n^2)$

```
Bubble sort ( A, n)
{
  for k ← 1 to n-1
  {
    flag ← 0
    for(i ← 0 to n-k-1
    {  if (A[i] > A[i+1]
       { swap [A[i], A[i+1];
         flag ← 1;
       }
    }
    if (flag == 0) break;
  }
}
```

Insertion Sort

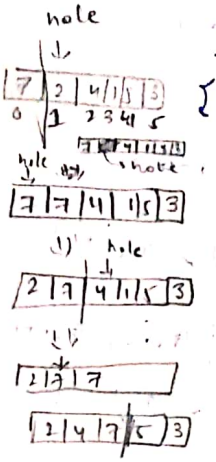




MODIFIED SORT

We have to put it at right position in right hand

Sorted array 1, 2, 3, 4, 5



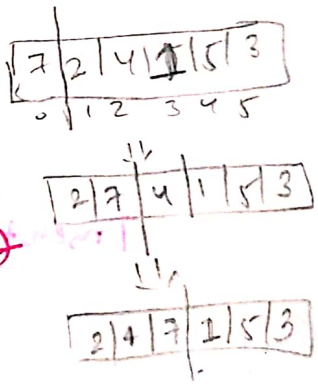

```
InsertionSort(A, n)
{
    A[0] is already sorted
    for i ← 1 to n-1
    {
        value ← A[i]      ] (n-1) times
        hole ← i
        while (hole > 0 && A[hole-1] > value)
        {
            A[hole] ← A[hole-1]      ] C2
            hole ← hole - 1;
        }
        A[hole] ← value      ] C3 (n-1) times
    }
```

i	value	hole
1	2	1
1	2	0
2	4	2
2	4	1
3	1	1

best case $T(n) = (C_1 + C_3)(n-1)$

$= an - b$

$= O(n)$



Worst Case 5, 4, 3, 2, 1

$T_n = (C_1 + C_3)(n-1) + (1 + 2 + 3 \cdots n-1) C_2$

$= (C_1 + C_3)(n-1) + \frac{n(n-1)}{2} C_2$

$= an^2 + bn + c$

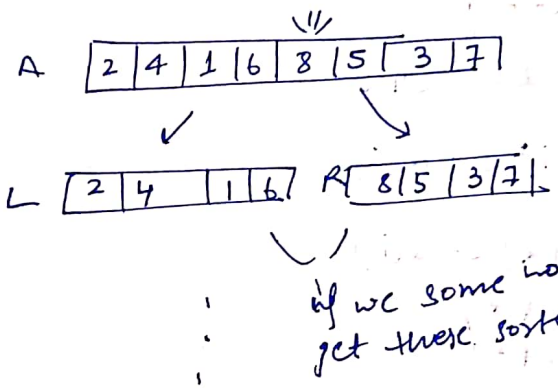
$= O(n^2)$

Average $= O(n^2)$

MERGE SORT

$T(n \log n)$

Break Problem into subproblem




```
Merge Sort ( A)
{
    n ← length(A)
    if (n<2) return
    mid ← n/2                                   } C1  creating element
    Left[] ← array of size (mid)
    Right[] ← array of size (n-mid)
    for ( i=0 ; i< mid ; i++)                   } C2n  filling with element
        Left [i] ← A[i]
    for ( i= mid ; i<n ; i++)
        Right [i - mid] ← A[i]
    T(n/2) ← mergeSort(Left)
    T(n/2) ← mergeSort(Right)
    C3n+C4 ← Merge (Left, Right, A)
}
```

```
Merge (L, R, A)  // there 3 are Arrays
{  nL ← length of(L)      ]  finding length
   nR1 ← length(R)        ]
   i ← j ← k ← 0 ;        ]  initialize to zero
while (i<nL && j<nR)
{  if (L[i] <= R[j]
    {
       A[k] ← L[i]
       k++;
       i++;
    }
else
    {
       A[k++] ← R[j];
       j++;
    }
}
while (i<nL)
{
   A[k] ← L[i]; i++, k++;
}
while (j<nR)
{
   A[k++] ← R[j++];
}
}
```

comparing till while condition is true

Adding remaining elements to Array

MERGE SORT Programme

```
#include <iostream>
#include <stdlib.h>      // for free function
using namespace std;
void Merge (int *A, int *L, int leftcount, int *R, int Rcount);
void MergeSort (int *A, int n);

int main ()
{
    int A[] = {6,2,3,1,9,10,15,13,12,17};
    int n = sizeof(A)/sizeof(A[0]);
    MergeSort (A, n);
    for (int i=0; i<n; i++) cout<< A[i] <<" ";

    return 0;
}

void MergeSort (int *A, int n)
{
    int mid, *L, *R;
    if (n<2) return;
    mid = n/2;
    L = new int[mid];
    R = new int[n-mid];
        or
    L = (int*) malloc (mid * sizeof(int));
    R = (int*) malloc ((n-mid) * sizeof(int));

    for (i=0; i<mid; i++)  L[i] = A[i];
    for (i=mid; i<n; i++)  R[i-mid] = A[i];
```

```
    MergeSort(L, mid);
    MergeSort(R, n-mid);
    Merge(A, L, mid, R, n-mid);
    free(L); free(R);
}

void Merge(int *A, int *L, int LC, int *R, int RC)
{
    int i=0, j=0, k=0;
    while(i<LC && j<RC)
    {
        if(L[i] < R[j]) A[k++] = L[i++];
        else A[k++] = R[i++];
    }
    while(i<LC) A[k++] = L[i++];
    while(j<RC) A[k++] = R[j++];
}
```

Properties

(i) Divide and Conquer Problems

(ii) Recursive algorithm

(iii) Stable — means it preserves the relative order of elements i.e. example (1,2) (2,5) (4,8) (2,5) — equal keys
=> (1,2,3) can also comes first but order is preserved

(iv) Not - In-place Algo (extra space take) $\theta(n)$ or $O(n)$ [Merge sort]

(v) $T = O(n \log n)$

$$T(n) = \begin{cases} c & \text{, if } n=1 \\ 2T(n/2) + (c_2+c_3)n + (c_1+c_4) \\ \text{or} \\ 2T(n/2) + c'n + c'' & \text{if } n \geq 1 \\ \text{or} \\ 2(n/2) + c'n \end{cases}$$

$$T(n) = 4T(n/4) + 2c'n$$

$$= 8T(n/8) + 3c'n$$

$$= 16T(n/16) + 4c'n$$

$$= \boxed{2^k T(n/2^k) + k \cdot c'n}$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + cn/2\right] + cn$$

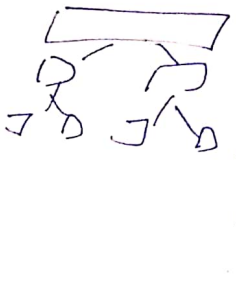
$$= 4T(n/4) + cn + cn$$

$$= 4T(n/4) + 2cn$$

$$\Rightarrow \frac{n}{2^k} = 1 \Rightarrow 2^k = n$$

$$k = \log_2 n$$

$$= 2^{\log_2 n}(T(1)) + \log_2 n \, c'n \Rightarrow nc + c'.n \log n \Rightarrow O(n \log n)$$



Space Complexity

$\frac{n}{2^k} = 1$

$k = \log_2 n$

if we do not clear extra memory for left and right $= n \log n = \theta(n \log n)$

if we delete ~~then~~ in each call

$= n + \frac{n}{2} + \frac{n}{4} \cdots$

$= n \left[1 + \frac{1}{2} + \frac{1}{4} \cdots \right]$

$=$ (taken to ∞)

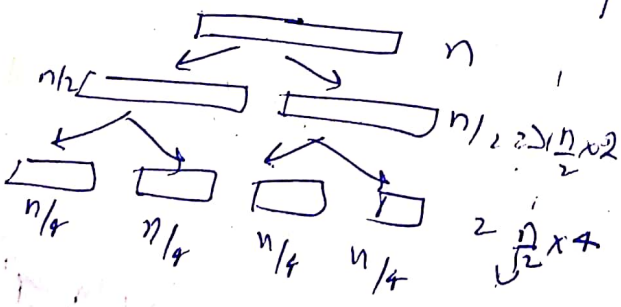
$\frac{1}{1-\frac{1}{2}} = 2$

$= 2n$

space complexity $= \theta(n)$

---

if we don't clear array then



$n/2 + n/2 = \frac{n}{2} \times 2$

$\frac{n}{2} \times 4$

So memory use $= nk$ (k times) $\Rightarrow n \times \log n$

if we clear

$\Rightarrow \frac{n}{2} + \frac{n}{2} = n$

$\frac{n}{4} \times 4 = n$

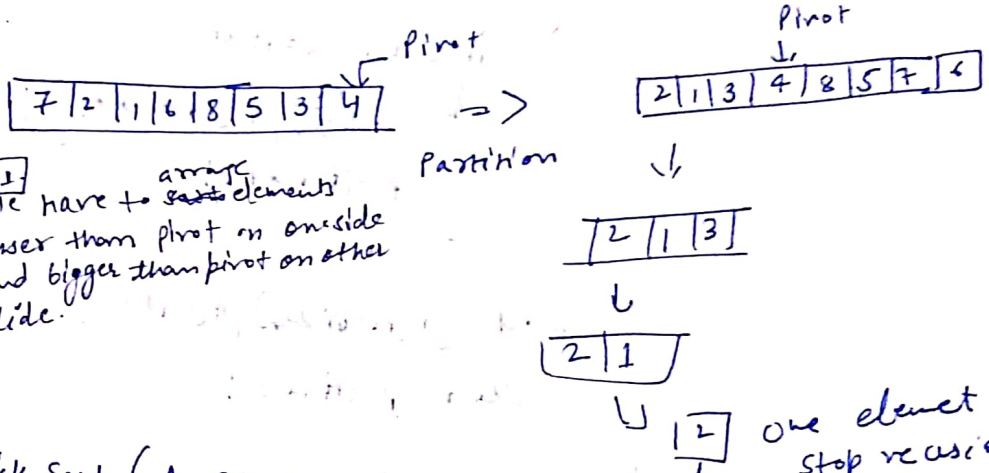
So each time it takes n memory but previous memory is deleted so $O(n)$ is complexity

Quick Sort

T = $O(n \log n)$ → Average case running time

$O(n^2)$ → worst case

but In place so benifit in space complexity



step1 We have to arrange elements lesser than pivot on one side and bigger than pivot on other side.

```
Quick Sort (A, start, end)   T(n)
{
    if( start >= end) return;   // if(start<end) to check equal or return
    Pindex ← Partition (A, start, end);   Tn = an + b
    Quicksort( A, start, Pindex-1 );  → T(n/2)
    Quicksort ( A, Pindex+1, end);  → T(n/2)
}
```

example 1 > 2 false 1 >= 1 true then actu

Worst case when array is already is sorted

$T(n) = T(n-1) + c.n$

To make probability low we use randomize function

$T(n) = 2T(n/2) + an + b + c$

$T(n) = 2T(n/2) + an$

(A, start, end)

7	2	1	6	8	5	3	4

↓ Pivot

↑ Partindex

```
int Partition (int *A, int start, int end)
{
    int pivot = A[end];
    int partindex = start
    for (int i = start ; i < end ; i++)
    {
        if( A[i] <= pivot)
        {
            swap(&A[i], &A[partindex]);
            partindex++;
        }
    }
    swap (A[partindex], A[end]);
    return partindex;
}
```

In this auxiliary array is not created like in merge ~~Insertion~~ Sort

(i) Divide and Conquer

(ii) Recursive algo

(iii) Not stable ⟶ eg (1,2) (4,5) (2,3) (4,5) (5,2)

↑ ↓

x y

Sort by x coordinate

(1,2) (2,3) (4,5) (4,5) (5,2)

or

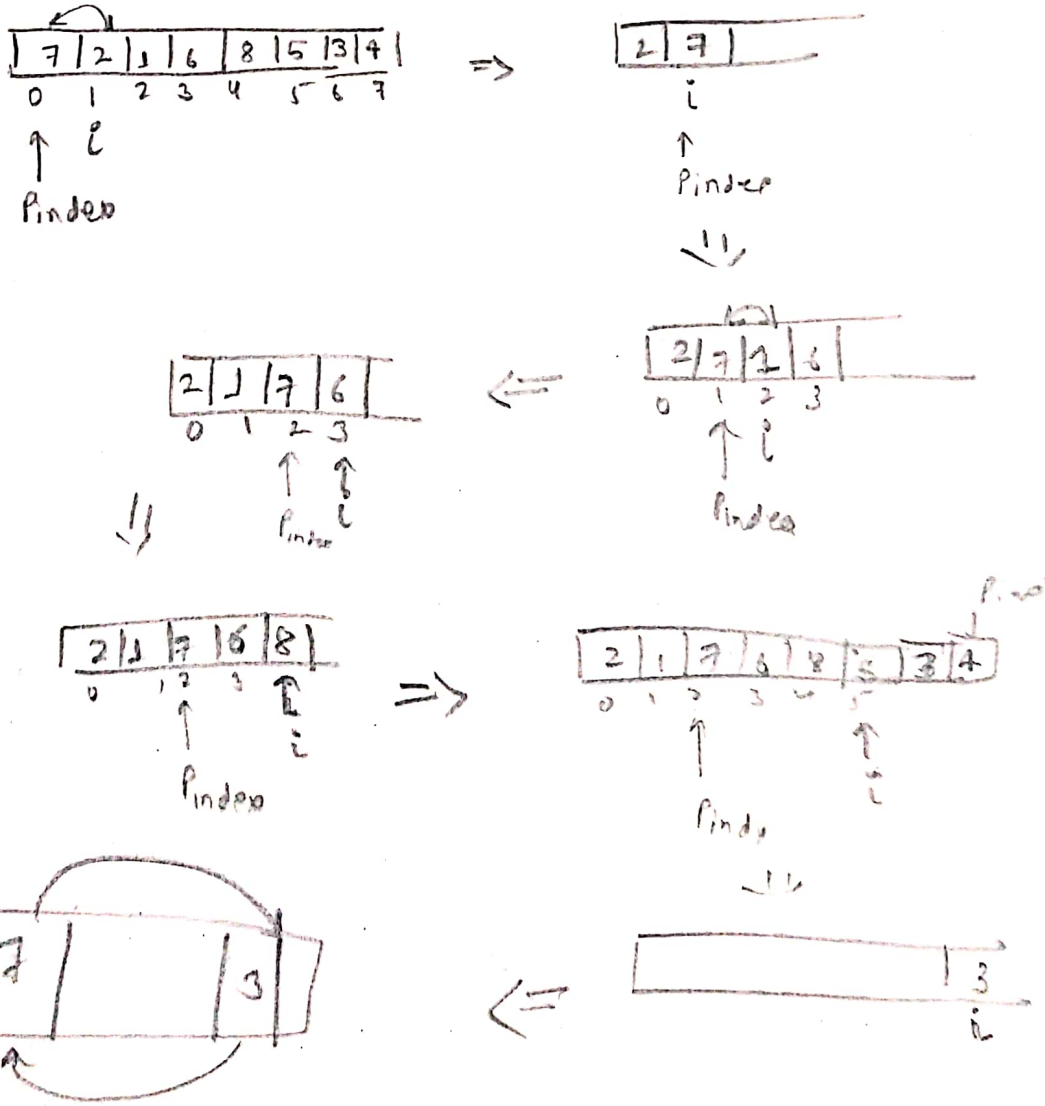
(4,3) (4,5)

(iv) T = n log n

worst = O(n²)

```
Randomize Partition (A, start, end)
{
    Pivot_Index ← Random (start, end);
    swap ( A[PivotIndex], A[end]);
    Partition (A, start, end)
}
```

now Instead of calling Partition in Quick sort we call Randomizepartition function. So that probability of worst case become low



Data Structures

uses → English Dictionary

City Map → position of landmark, lane marks

Daily Cash In, cash out called cashbook

A data structure is a way to store and organize data in a computer, so that it can be used efficiently.

We talk about data structures as

(i) Mathematical / Logical models

or

Abstract data types (ADT)

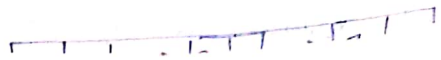
eg  TV. we know it Receives signals, play audio / vedio but we do not concern about what circuit is used, how it is embedded.

eg List → store data number elements

Read, modify elements

Array is concrete implementation

∴ ADT → defines data and operations but no implementation



ii) Implementation :- ex Linked list

ADT

List is collection of objects of same times . i.e is real world entity

basic

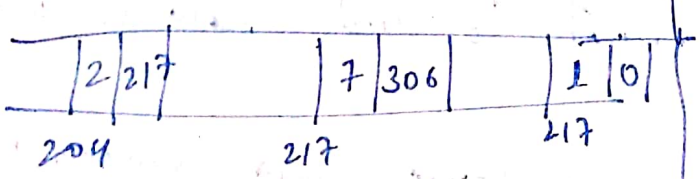
array :- store , modify and access elements

ex

```
int a[10];
A[i] = 2;
Print A[i];
```

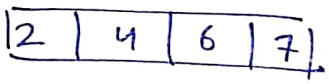
this is static list

Now creating dynamic list which is data structure. which is more efficient in memory



So

```
Struct node {
    int data;      // 4 bytes
    node *next;    // 4 bytes
```



5 ↗

Insertion

deletion

Complexity :-

accessing = O(1)

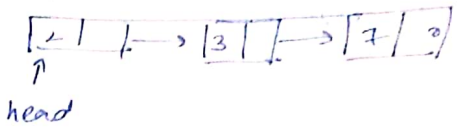
Insertion :

Time ∝ size of array

(n)

O(n)

Deletion = O(n)



* pointer takes n bytes in 4 bit integer system

Complexity : Time of size of list (n)

$O(n)$

Insertion :-

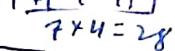
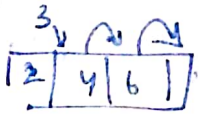
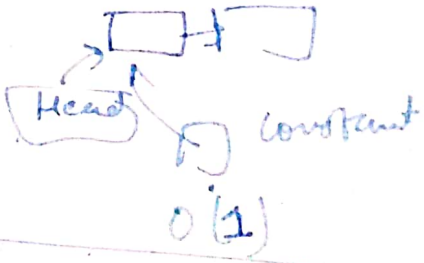
Traversal → $O(n)$
then $O(1)$
So finally $O(n)$

If on head then $O(1)$

Deletion :- $O(n)$

at head then $O(1)$

Access = $O(n)$

	Array	LinkList
Memory requirement :-	Fixed size  $n = 7[0-6]$ $7 \times 4 = 28$ bytes	dynamic size $6 \times 4 = 32$ bytes  extra memory for * variable
Access time	$O(1)$	$O(n)$
Cost of Inserting at beginning	$O(1)$ 	 constant $O(1)$
at end	$O(1)$ if array is not full $O(n)$ if array is full	$O(n)$
at i^{th} position :-	$O(n)$	$O(n)$

struct node *temp;

if array is full then we have to make a new array and copying all the elements of previous array in new array so O(n) is complexity

Easy to use. → Array is easy to use

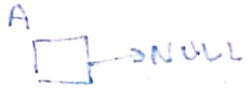
not easy to use

Implementation of Linklist

C → struct node {
    struct node * link;
    int data;
}

C++ struct node {
    node * link;
    int data;
}

node * A;
A = NULL ; // List is empty



// to create a node

node * temp = (node *) malloc (sizeof(Node));
// C

node * temp = new node(); // C++
(*temp).data = 2; ≡ temp -> data = 2;
(*temp).link = NULL; ≡ temp -> link = NULL;
A = temp;

Inserting node at beginning.

```
Struct node {
        int data;
        node * next;
    }
Struct node * head;
int main()
    {
      head = NULL;
      Printf("Enter numbers");
      int n, i; int x;
      Scanf("%d", &n);
      for(i=0; i<n; i++) {
          Print("number");
          Scanf("%d", &x);
          Insert(x);
          Printf();
      }
}
```

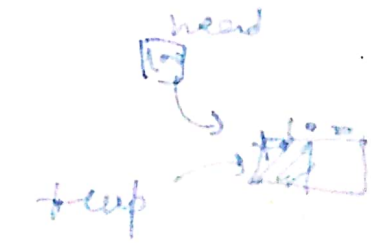
if we use struct node *temp;
then it will create as local variable and store in STACK we want in heap so that we can retain in last of programme. In stack after ending of function stack values are dismissed so we use node * temp = new node;
if head is not global

node * head = NULL;

if we want to reference
Insert(&head, data

head = Insert (head, data);
Printf (head);

```
Insert (int x)
{
    Node * temp = (node *) malloc (sizeof(node));
    temp -> data = x;
    temp -> next = NULL;
    if(head == NULL)
        temp -> next = head;
        head = temp;
    head = temp;
}
```



nth position

```
void Print()
{
    struct node* temp = head;
    while(temp != null)
    {
        Printf("%d", temp->data);
        temp = temp->next;
    }
    Printf("\n");
}
```

traversal

```
Print(node* head)
{
}
```

```
node* Insert(node* head, int data)
{
    |
    return head;
}
```

```
void Insert(node** PointerToHead, int x)
{
    node* temp = new node();
    temp->data = x;
    temp->next = null;
    if(*PointerToHead != NULL)
    {
        temp->next = *PointerToHead;
        *PointerT.hed = temp;
    }
```

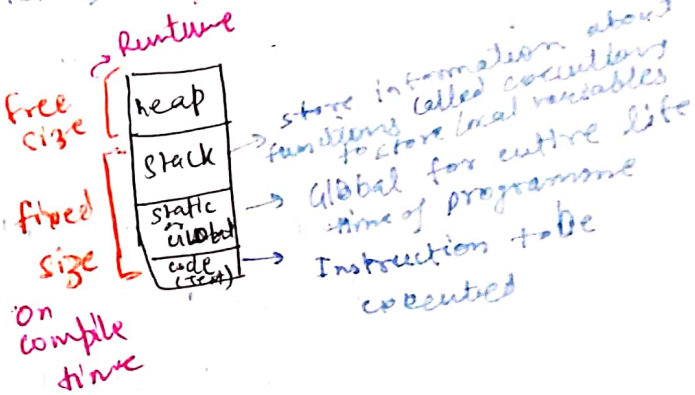
Inserting a node at nth position

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

struct node * head;

int main()
{
    head = NULL;
    Insert(2,1); // 2
    Insert(3,2); // 2,3
    Insert(4,1); // 4,2,3
    Insert(5,2); // 4,5,2,3
    Print();
}

void Insert(int data, int n)
{
    node * temp1 = new node();
    temp1 -> data = data;
    temp1 -> next = NULL;

    if (n == 1) {
        temp1 -> next = head;
        head = temp1;
        return;
    }
    node * temp = head;
    for (i = 0; i < n-2; i++)
    {
        temp = temp -> next;
    }
```

creating node

inserting at beginning

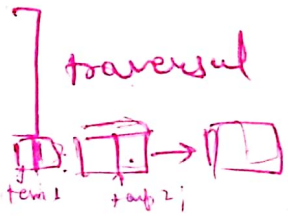
traversal



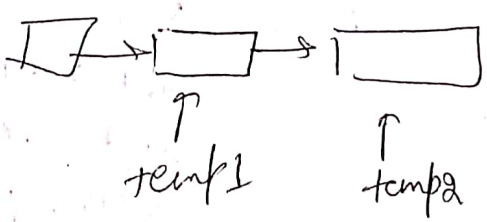
```
    temp1 -> next = temp -> next ;
    temp -> next = temp1 ;
}
```

Delete a node at nth position:

```
void del (int n)
{
    struct node * temp1 = head ;
    if (n == 1)
    {
        head = temp1 -> next ;
        free (temp1);
        return ;
    }
    int i = 0
    for (i = 0 ; i < n-2 ; i++)
        temp1 = temp1 -> next ;
```

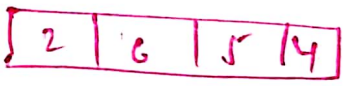



```
    node * temp2 = temp1 -> next ;
    temp1 -> next = temp2 -> next ;
    free (temp2) ;
}
```



Reverse Linklist using Iterative method.

```
node * Reverse (node * head)
{
    node * previous, * current, * next;
    * previous = NULL;
    * current = head;

    while ( current != null)
    {
        next = current -> next;
        current -> next = previous;
        previous = current;
        current = next;
    }
    head = previous;
    return head;
}
```



Reverse using Recursion

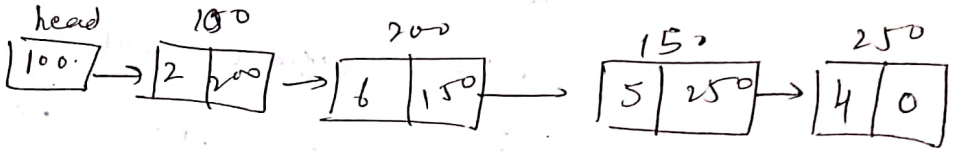
```
// Normal Print via Recursion
void Print (node * P)
{
    if ( P == NULL ) // Exit condition
    {
        return;
    }
    Print f("%d", P -> data); // Print data
    Print ( P -> next); // Recursive call.
}
```

Print (head)
{ // recursion
// 2, 6, 5, 4

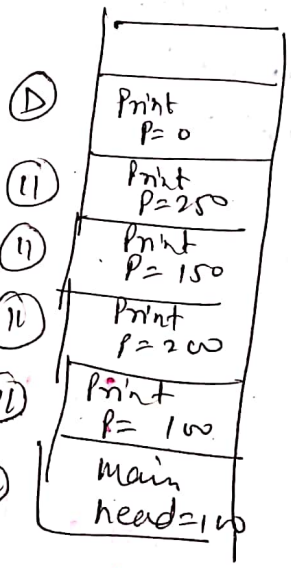
Reverse Print (head)
{ //
recursion
// 4, 5, 6, 2

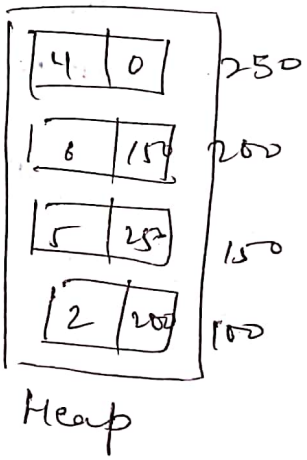
```
void ReversePrint ( node *P)
{
    if ( P == NULL ) return;
    Print ( P -> next );
    Printf ( "%d", P -> data );
}
```

```
main()
↓
RP(100)
↓
RP(200)
↓
RP(150) -> RP(250)
            ↓
          RP(NULL)
```



for Print function





For normal traversal use iterative approach because in Recursion lots of functions are called and they are stored in stack.

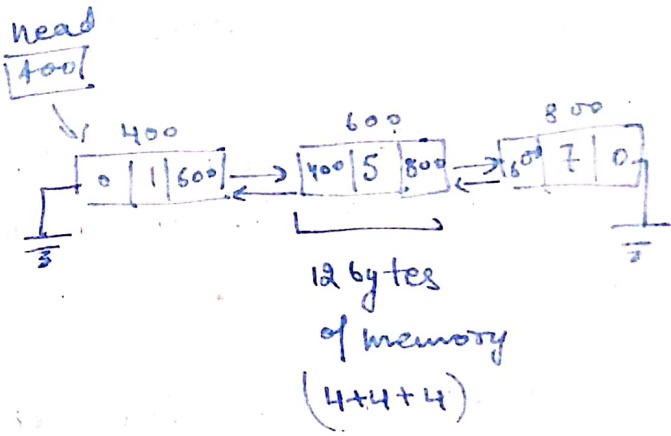
Now Reversed the LinkList

```
void Reverse (struct *p)
{
    if (p->next == NULL)
    {
        head = p;
        return;
    }
    Reverse (p->next);
    struct node *q = p->next;
    q->next = p;
    p->next = NULL;
}
```

or we can do `p->next->next = p;`

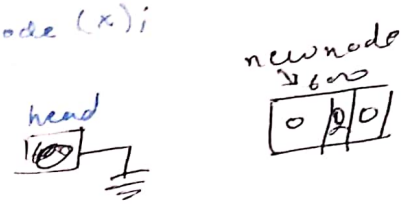
Doubly LinkList

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




```
struct node *GetNewNode(int x)
{
    struct node *temp = new node();
    temp->data = x;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
```

```
// inserting at head;
void InsertAtHead (int x)
{
    node * newnode = GetNewNode (x);
    if (head == NULL)
    {
        head = newnode;
        return;
    }
    head -> prev = newnode;
    newnode -> next = head;
    head = newnode;
}
```



STACK

we talk about STACK as ADT
means features and operations not implementation.

ex:
① stack of disks = dinner plates
② Tower of Hanoi

LIFO
Last In First out

STACK :- A List with restriction that insertion and deletion can be performed only from one end, called the top

operations :-

(i) Push (x); // inserting element
(ii) Pop // removing element
(iii) Top() // returns the element on stack
(iv) IsEmpty() // check whether stack is empty or not

} constant time or O(1)

```
top = -1        // if array is empty or stack is empty

Push(x)
{
    top ← top+1;
    A[top] = x;
}

Pop()
{
    top ← top-1;
}

Top()
{
    return A[top];
}

IsEmpty()
{
    if (top == -1)
    { return True
    }
    else false;
}
```

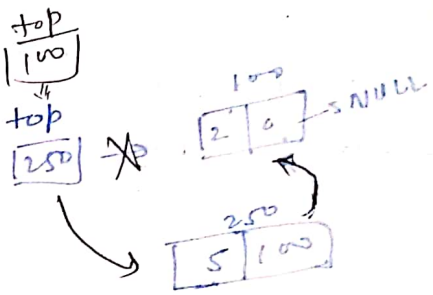
Program:

```
#include <stdio.h>
#define MAX_SIZE 101
int A[max];
int top = -1;
void Push (int x) {
    if (top == max-1)
    {
        Print "stack overflow";
        return
    }
    A[++top] = x;
}
void Pop ()
{
    top--;
    if (top == -1)
        Print "stack is empty"
}
int Top()
{
    return A[top];
}
void print ()
{
    int i;
    Printf("stack");
    for (i=0;i<=top;i++){ Print A[i]; }
    Printf("\n");
}
int main()
{
    Push(2); Push(5); Push(10); Print(); Pop(); Print();
}
```

Stack using Linklist :

```
Struct node {
        int data; struct node * link; }
```

```
Struct node *top = NULL;
void push (int x)
  {
    node *temp = new node();
    temp -> data = x;
    temp -> link = top;
    top = temp;
  }
```




```
void Pop()
  {
    Struct Node * temp;
    if (Top == NULL) return;
    temp = Top;
    top = top -> link;
    free (Temp);
  }
```

used in heresial of string, Linklist

In C++

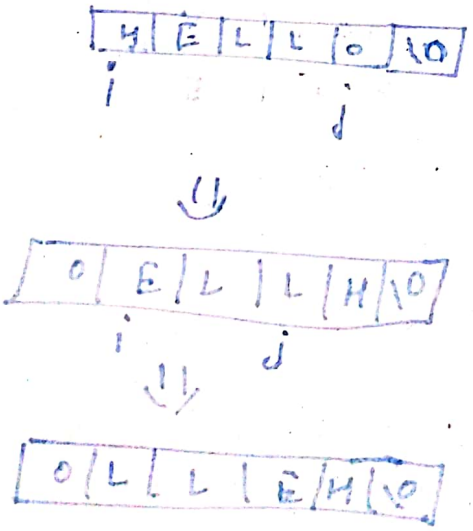
```
class stack
  {
    Private char a[101];
            int top;
    Public:
            void Push (int x);
            void Pop ();
            Int Top(); bool IsEmpty();
  };
```

```
void Reverse (char *c, int n)        // no. of characters in string
{
    stack <char> s;                  // include <stack>
    for(i=0 ; i<n ; i++)             // loop for push
    {
        s.push(c[i]);
    }
    // loop for pop
    for( int i=0 ; i<n ; i++)
    {
        c[i] = s.top();              // overwrite the character at index i
        s.pop();
    }
}

int main()
{
    char c[51];
    gets(c);
    Reverse (c, strlen(c));
    Printf ("output = %s", c);
}
```

O(n) = Time
O(n) = space for this function

Time Complexity = O(n)
space = O(n)

Another method for Reverse is Swapping



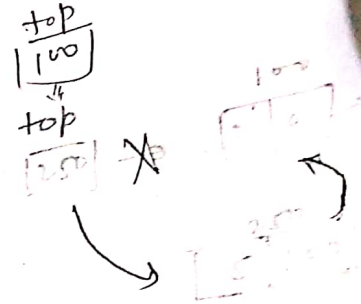

```
while (i<j)
{
    swap [ a[i], a[j] )
}
```

space complexity = O(1)

Time complexity = O(n)

Stack using linked list

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

void push (int x)
{
    struct node *temp;
    temp = (struct node *) malloc (sizeof (struct node));
    temp->data = x;
    temp->next = top;
    top = temp;
}

void Pop()
{
    struct Node *temp;
    if (Top == NULL) return;
    temp = Top;
    Top = Top->next;
    free (Temp);
}
```

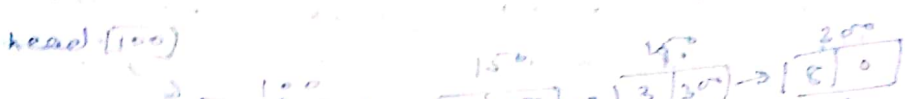


Used in Reversal of String

In C++ class stack

```
{
    Private: char a[10];
             int top;
    Public:  void Push(int x);
             void Pop();
};
```

Link list



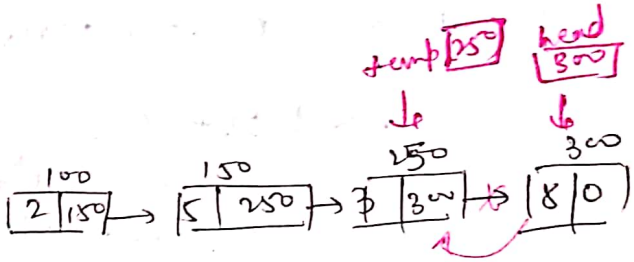
To push all references

```
node * temp = head;
while (temp != NULL)
{
    s.push (temp);
    temp = temp -> next;
}
```


```
stack< struct node *> s;
struct node {
    int data;
    node * next;
}
```

Poping

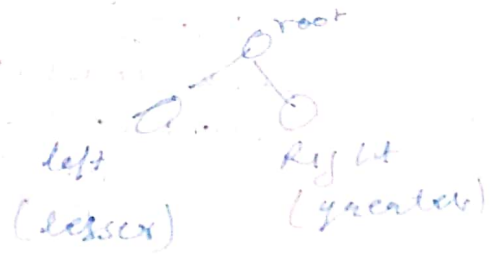
```
node * temp = s.top();
head = temp;
s.pop();
while ( !s.empty() )
{
    temp -> next = s.top();
    s.pop();
    temp = temp -> Next;
}
temp -> next = NULL;
```



```
void Reverse ()
{
    if (head == NULL) return;
    stack < struct node * > s;
    // To push all references
    temp = s.top(); head = temp;
    // poping function
}
```

BINARY SEARCH TREE
(Implementation)

A binary tree in which for each node value of all the nodes, values of all the nodes in left subtree is lesser or equal and value of all the nodes in right subtree is greater




```
#include <iostream>
struct Bstnode {
        int data;
        Bstnode * left;
        Bstnode * Right;
    }
Bstnode * root;
Bstnode * Getnewnode (int data)
  {
     Bstnode * newnode = new Bstnode();
     newnode -> data = data;
     newnode -> left = newnode -> Right = NULL;
    return newnode;
  }
```

```
BstTnode * node Insert ( Bstnode * root , int data)
{
    if (root == NULL)
    {
      root = getnewnode (data);
    }
   else if ( data <= root -> data)
     {
       root -> left = Insert (root -> left, data);
     }
    else
      {
        root -> right = Insert (root -> right, data);
      }
    return root;
}

Bool search ( Bstnode * root , int data)
{
    if (root == NULL) return false;
    elseif ( root -> data == data) return True;
    elseif ( data <= root -> data) {
                return search (root -> left , data);
      }
    else
      {
        return search ( root -> right , data);
      }
}
```

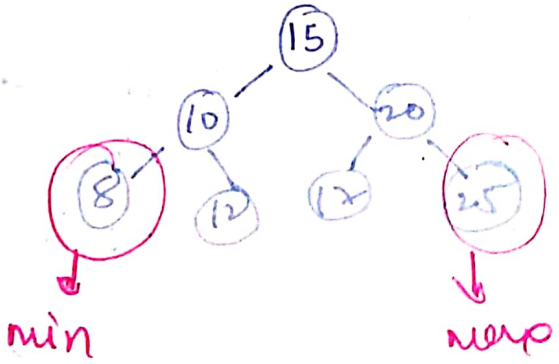
// Recursion.

```
int findMin ( Bst * root)
{
    if ( root == NULL)
```

```
int main ( )
{
    Bstnode * root = NULL;
    root = Insert (root, 15);
    root = Insert (root, 10);
    root = Insert (root, 3);
    int n;
    cin>>n;
if (search (root, n) == true) cout << "found";
else
  { cout << "not found";
  }
}
```

finding min and max

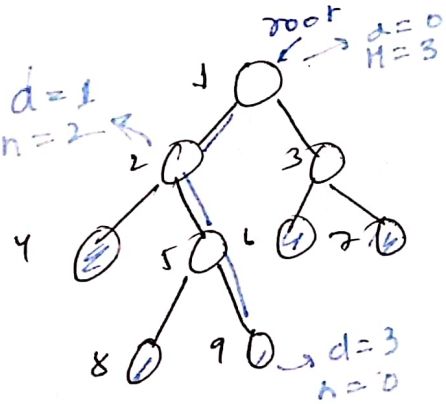
```
//iterative Solution
int findmin ( Bst * root)
{
  if ( root == NULL)
    {
      cout << "emty";
      return -1;
    }
  while (root->left != NULL)
    { root = root->left;
    }
  return root->data;
}
```



// Recursion.

```
int findMin( BST * root)
{
    if ( root == NULL)
    {
        Print empty
        return -1;
    }
    else if ( root->left == NULL)
    {
        return root->data;
    }
    else
        return FindMin( root->left);
}
```

Height of Binary Tree



Height = number of edges in longest path from root to leaf node

Height of tree = Height of root

if node = 1 then height is zero

Depth of a node = no. of edges in path from root to that node

DFS

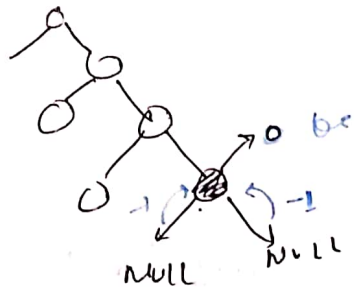
```
FindHeight (root)
{
  if (root is null)
  {
    return -1;
  }
}
```

if you want to count no. of nodes then return 0;

```
leftHeight = findHeight(root->left);
rightHeight = findHeight(root->right);
return max(leftHeight, rightHeight) + 1;
}
```

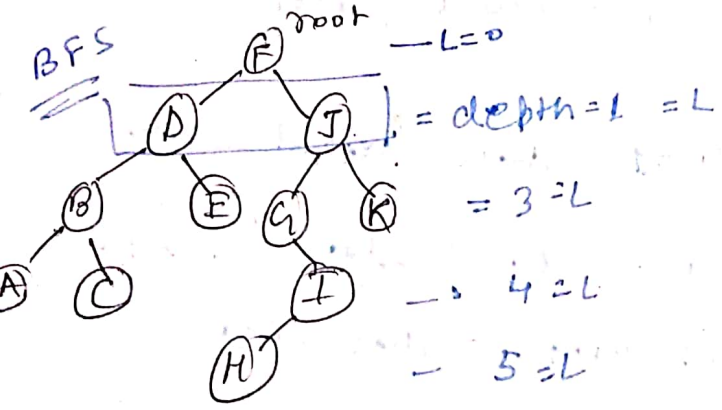
or run

```
return max(FindH(root->left), FH(root->rg)) + 1;
```



because height of left node is zero so we have to return -1 at null;

B Traversal → ~~B~~Binary Breadth-first and depth first Traversal



L=0

= depth = 1 = L

= 3 = L

→ 4 = L

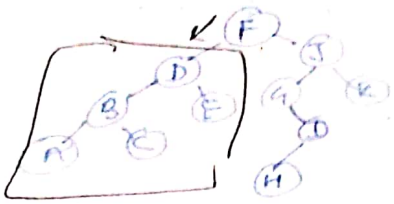
— 5 = L

BFS

F D J B E G K A C I H

called level order traversal

~~BFS~~



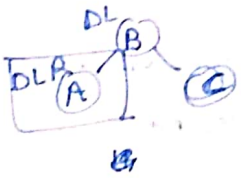
DFS
if we move toward left than we have to go all child of left then right comes and vice versa or subtree

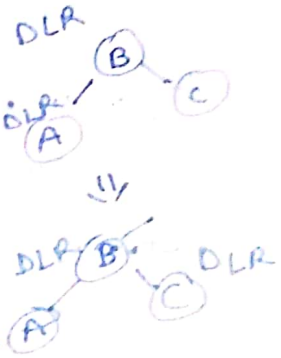
orders →

Data	Left	Right	
<root>	<left>	<right>	Preorder
<left>	<root>	<right>	Inorder
<left>	<right>	<root>	Postorder

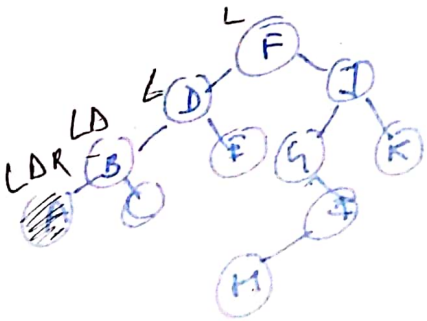
Preorder (D, L, R)

F D, B, A C E I, G, I, H, K





Inorder (L, D, R)



A, B, C, D, E, F, G, H, I, J, K

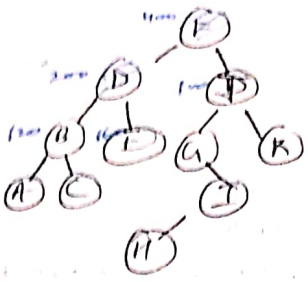
It gives sorted list

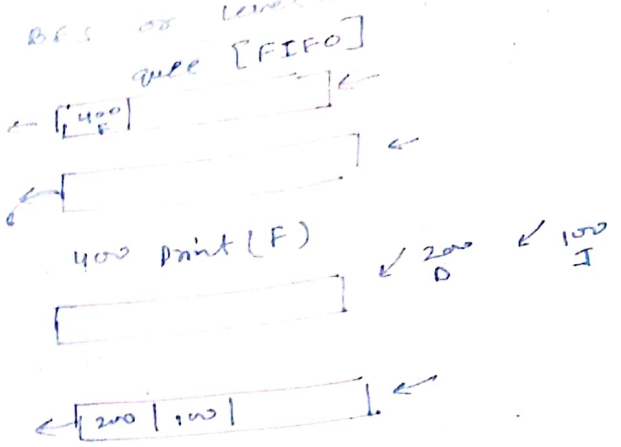
Post order (LR D)

A, C, B, E, D, H I, G, K, J, F

Time Complexity ∝ no. of nodes.

Programme for BFS or Level order Traversal



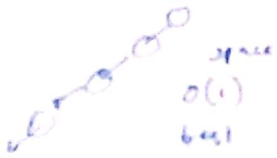


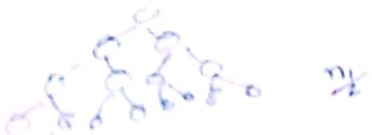

```
#include <iostream>
#include <queue>

struct node {
        char data;
        node *left;
        node *Right;
};

void levelorder (node *root)
{
    if (root == NULL) return;
    queue <Node *> Q;
    while (!Q.empty())
    {
        node * current = Q.front();
        cout << current -> data << " ";
        if (current -> Left != NULL)
                Q.Push (current -> Left);
        if (current -> right != NULL)
                Q: Push (current -> Right);
    }
    Q.Pop();
}
```

Time Complexity ∝ no of nodes

= O(n)

space complexity : rate of growth of extra memory used with input size





Time Complexity = O(n)
∝ nodes

DFS → Pre, Post, Inorder

```
void Preorder (Node * root)
{
    if (root == null) return;
    cout << root->data;
    Preorder (root->left);
    Preorder (root->Right);
}
```

Space complexity
= O(n)
↓
height of tree

best case : O(log n)

```
Void Inorder (node * root)
{
    if (root == NULL) return;
    Inorder (root->left);
    cout << root->data;
    Inorder (root->right);
}
```

```
void Postorder( node * root)
{
    if (root == NULL) return;
    Postorder (root -> left);
    Post order (root -> right);
    cout<< root -> data;
}
```

Check if BT is not "B"T or not

int c we use int instead of bool because there is no bool in c;

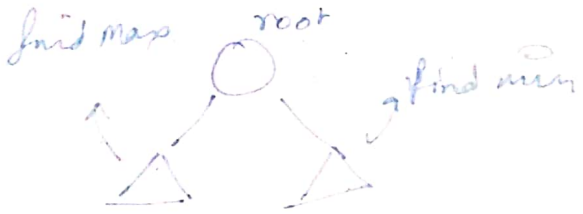
```
bool IsBinarySearchTree( node* root)
{
    if (root == NULL) return True;

    if (Issubtreeless (root -> left, root -> data)
        && Issubtreegreater( root -> right, root->data)
        && IsBinarySearchTree( root -> left)
        && IsBinarySearchTree( root -> right))
    {
        return true;
    }
    else
        false;
}
```

```
bool Issubtreeless( node * root, int value)
{
    if( root == NULL) return true;
    if( root -> data <= value)
        && Issubtreeless( root -> left, value)
        && Is subtree less( root -> Right
```

return true;
else
return false;
}

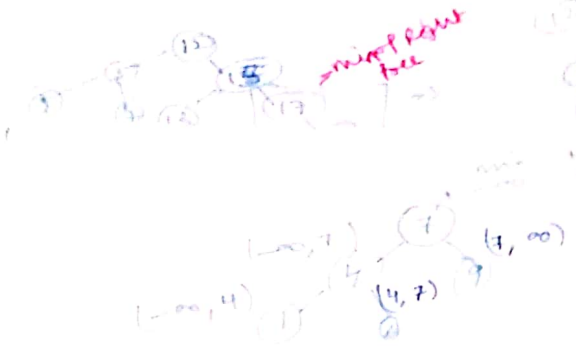
Solution 2

other than find less or greater we can find max of left of subtree and compare with root if max of left of subtree is less than root than all elements are lesser than root and same similar case with right (findmin)



Solution 3

```
bool IsBstUtil (Node* root, int min, int max)
{
  if (root == NULL) return true;
  if (root->data > minvalue && root->data < maxvalue
      && IsBSTUtil (root->left, min, root->data)
      && IsBstUtil (root->right, root->data, max))
      return true;
  else false;
```



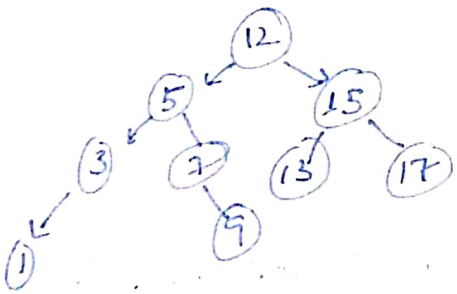
```
bool IsBinaryTree(node * root)
{
    return IsBSTUtil(root, Int min, Int max);
}
```

Solution 4 -

Print tree in Inorder, if it is in sorted order than it is B Search Tree else no

---

Deleting a node from BT

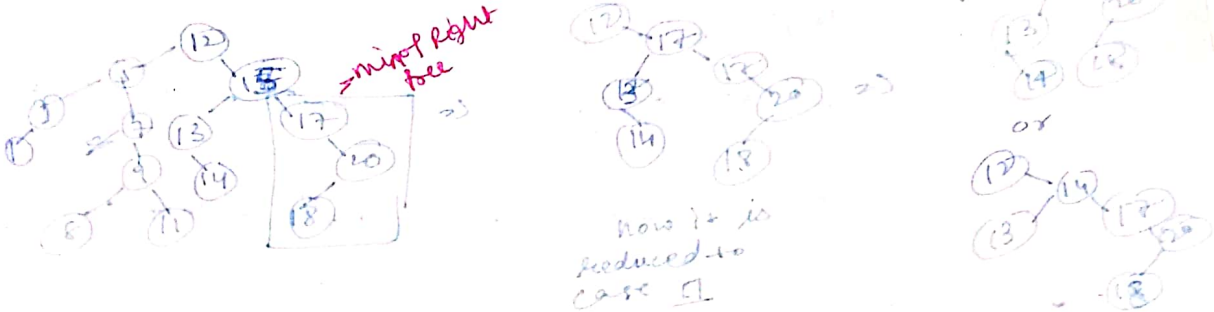


after deleting 7 having only one child

Case I

Case II having 2 children like 5

Case I deleting leaf node like 1, 9, 13, 17 is not so complicated



Case 1 → find the min of right subtree, copy the value to targeted node, Delete duplicate from right subtree.

or

we can get max of left subtree

Code

```
#include<iostream>
struct node {

        3

node * findmin(node *root)
  {
      while(root->left!=NULL) root=root->left;
       return root;
  }

struct node * Delete (struct node *root, int data)
  {   if(root==NULL) return root;
      else if(data<root->data)
          root->left = Delete(root->left, data)
```

```
else if (data > root->data)
    root->right = Delete(root->right, data);
// Now we find element ready to delete.
else
{
    if (root->left == NULL && root->right == NULL)
    {
        delete root;
        root = NULL;
    }
    else if (root->left == NULL)
    {
        struct node *temp = root;
        root = root->right;
        delete temp;
    }
    else if (root->right == NULL)
    {
        struct node *temp = root;
        root = root->left;
        delete temp;
    }
    else
    {
        struct node *temp = FindMin(root->right);
        root->data = temp->data;
        root->right = Delete(root->right,
                             temp->data);
    }
return root;
}
```

```
// Inserting node
Node * Insert (node * root, int data)
{
    if (root == NULL)
    {
        root = new node();
        root -> data = data;
        root->left = root -> left = NULL;
    }
    else if (data <= root -> data)
    {
        root -> left = Insert (root->left, data);
    }
    else
    { root -> right = Insert ( root -> right, data);
    }
return root;
}
int main () {
    node * root = NULL;
    root = Insert (root, 5); root = Insert (root, 10);
```

3

1

4

11

```
// Deleting
    root = Delete (root, 5);
    cout << "Inorder: ";
    Inorder (root); }
}
```

finding successor in Inorder with complexity o(11)

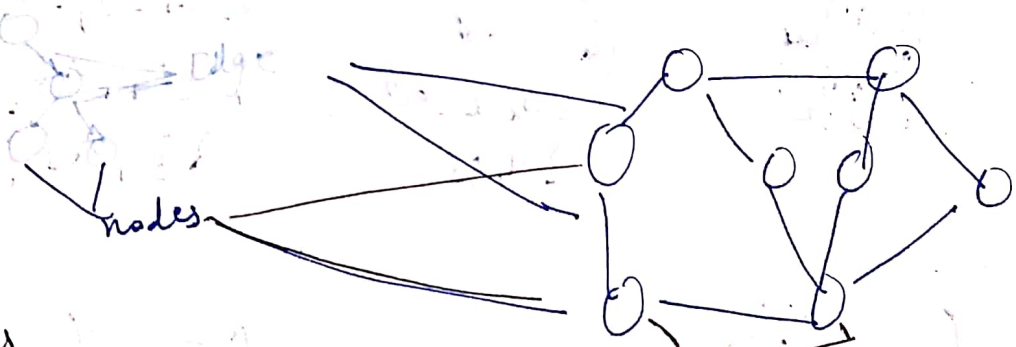
```
Node * Find( node * root , int data)
{
  if (root == NULL) return NULL;
  else if (root->data == data) return root;
  else if (root->data < data) return (Find(root->right,data));
  else return Find( root->left , data);
}

Node * findMin( node * root)
{
  if (root == NULL) return NULL;
  while (root->left != NULL)
      root = root->left;

  return root;
}

Node * Successor ( node * root , int data)
{
  node * current = Find (root, data);
  if (current == NULL) return NULL;
// case I  Node has right subtree
  if ( current->right != NULL) {
      return FindMin( current->right);
  }
// case 2  No right subtree
  use {
      struct node * successor = NULL;
      ancestor = root;
      while( ancestor != current)
      {  if ( current->data < ancestor->d
         {
```

— 1≠n

the deepest node for which current node is left

Graphs

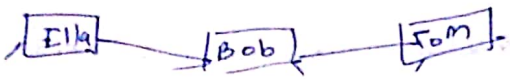
Non linear data structure



if N nodes
then (n-1) edges
one edge for
each parent child
relationship.

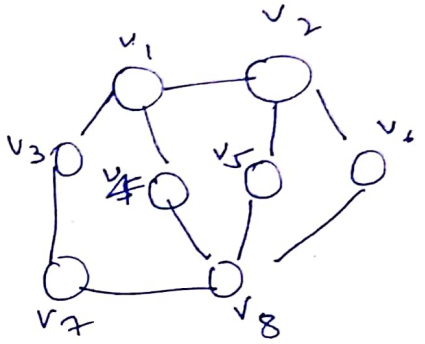
no rules for connections

→ Tree is special type of graph.



Graph :- A graph 'G' is an ordered pair of set 'V' of vertexts and a set 'E' of edges

$$G = (V, E)$$



ordered pair :

$(a,b) \neq (b,a)$ if $a \neq b$

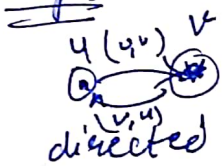
unorder pair

$\{a,b\} = \{b,a\}$

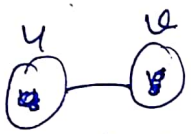
$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$

$E = \{\{v_1,v_2\}, \{v_1,v_3\}, \{v_1,v_4\}, \{v_2,v_5\}, \{v_2,v_6\}, \{v_3,v_7\}, \{v_4,v_8\}, \{v_7,v_8\}, \{v_3,v_8\}, \{v_6,v_8\}\}$

Edges

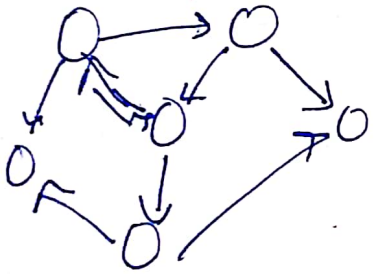


directed

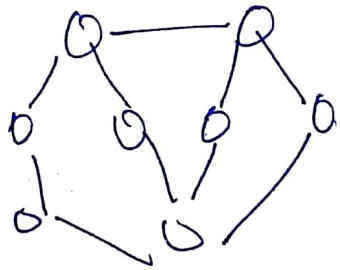


undirected

$\{u, v\}$

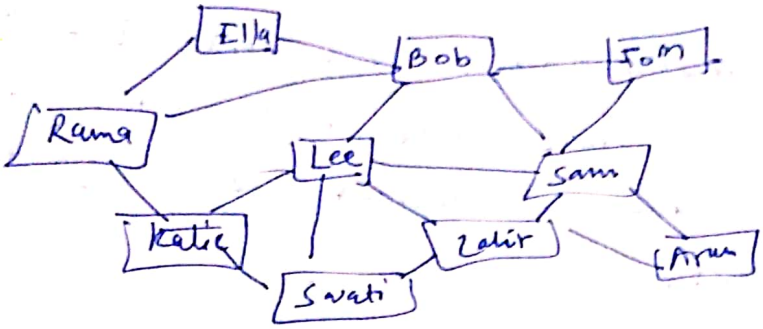


a directed graph or Digraph



undirected graph

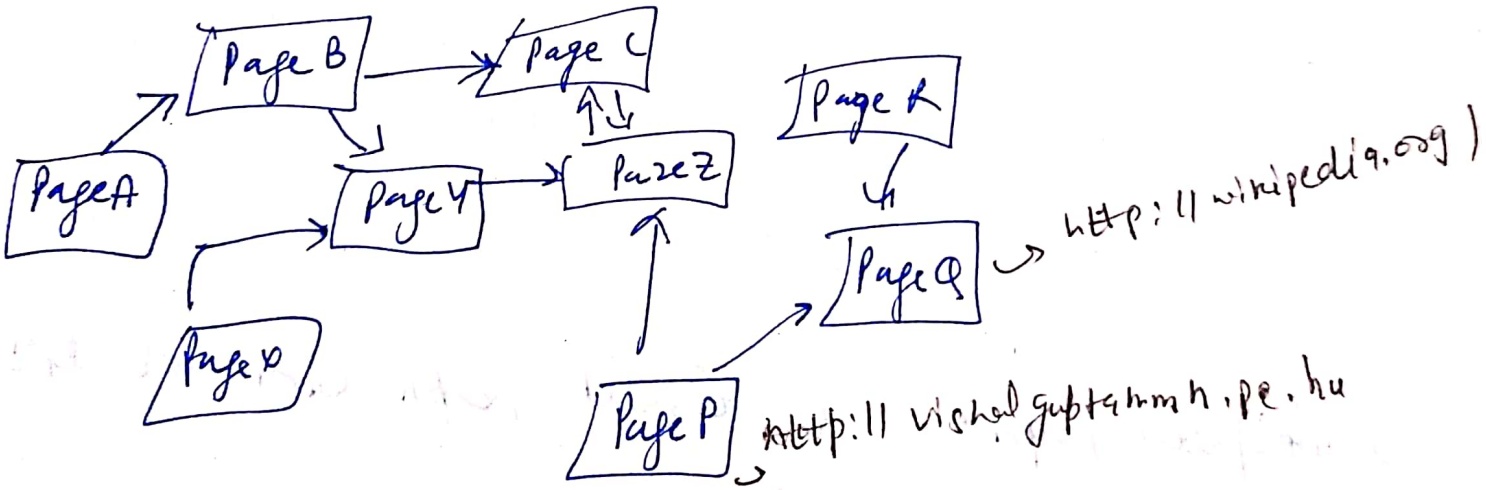
- facebook (social network)



Social network (facebook)

eg we have to suggest friend to Rama.
So we can go to → Tom, Sam, Lee, Swati

In algorithm we say find all nodes having length of ~~sort~~ shortest path from Rama equal to 2



World wide web

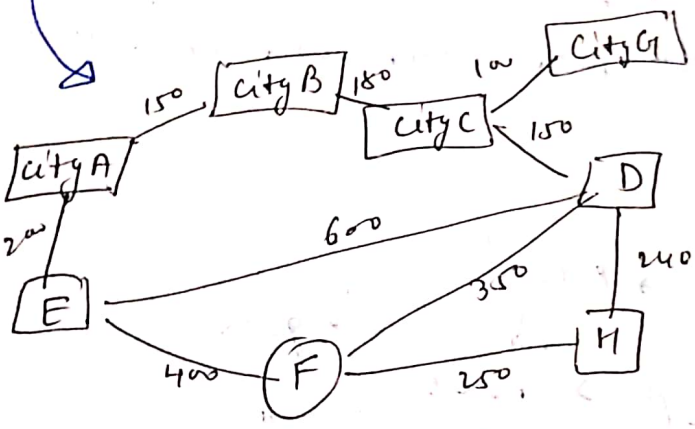
My page has link to Page Q but Page Q does not have link to my page so it is it problem of directed graph

Properties of GRAPH

search engines use programmes called
eg web crawling
that systematically browses the world wide web
to collect and store data about web pages
to provide quick and accurate results against
search queries.

web-crawling is Graph Traversal

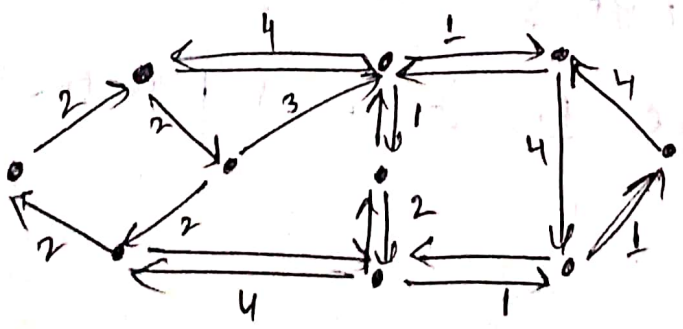
Weighted and Unweighted graph



Intercity road network

Unweighted graph :- a weighted graph with all edges having weight = 1 unit

eg social network



Intra city road network

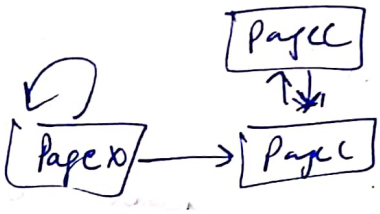
eg of directed weighted graph

Properties of GRAPH

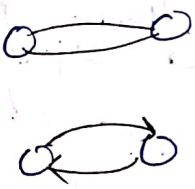
Special Edges

self loop :-

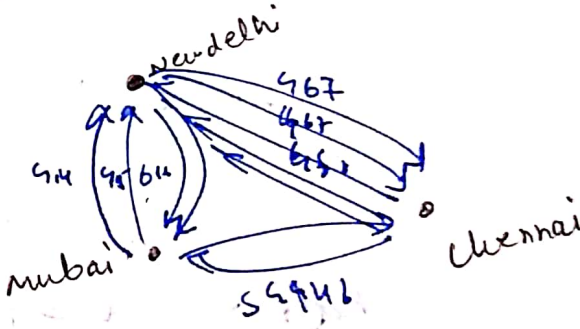
why we have self loop : if we have same source and destination



Multi Edge :-



eg flight route b/w cities



root for each flight

* If a graph have no self loops and multi Edge. then it is called as simple graph

... simple path = walk.

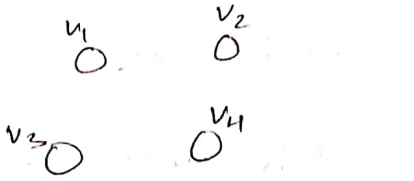
Number of edges :-
[if no self loop and multi edges]

if $|V| = n$

then

$0 \le |E| \le (n-1)n$, if directed

$0 \le |E| \le \frac{n(n-1)}{2}$, if undirected



$V = \{v_1, v_2, v_3, v_4\}$

min $E = \{\phi\}$

max E (directed) :-

$4 \times 3 = 12$

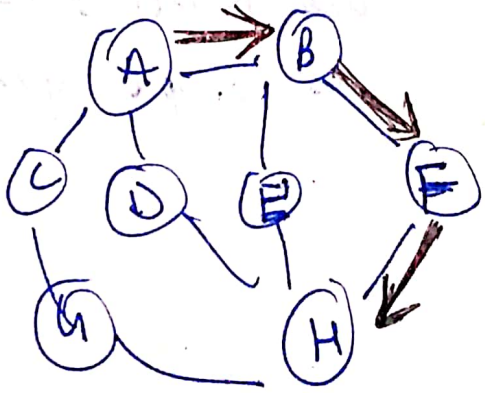
Dense → too many edges means near to max no. of edges then graph is called dense graph.
eg adjacency matrix

sparse :- too few edges.
eg adjacency list

Path or walk :- a sequence of vertexs where each adjacent pair is connected by an edge.

Simple path :- a path or walk in which no vertexs (and thus no edges) are repeated.

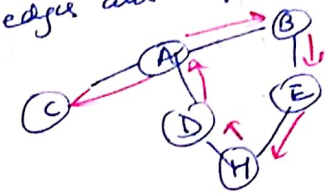
eg <A, B, F, H>



we can say simple path to walk.

Trail : a walk in which no edges are repeated

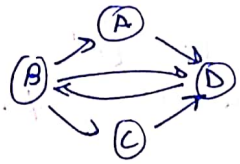
ex



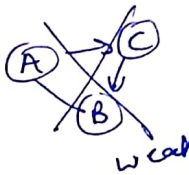
Strongly connected ~~path~~ Graph :-

if there is a path from any vertex to any other vertex

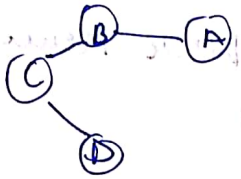
ex



strongly connected



weakly connected



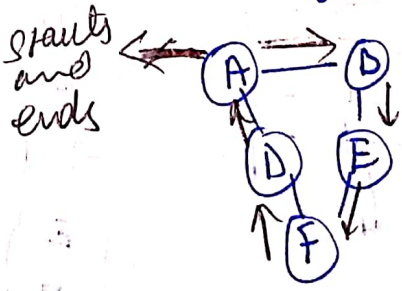
Directed graph so it is called as connected not strongly connected

Closed walk :- starts and ends at same vertex

Simple cycle :- no repetation other than start and end

or

cycle.



```
struct Edge
{
    char * startvertex;
    ... * endvertex;
```

In

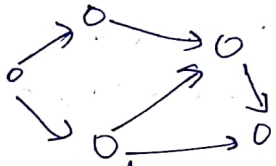
Acyclic graph :- a graph with no cycle

closed walk

In C, I

undirected acyclic graph

there is no simple cycle in tree



Directed acyclic graph DAG

GRAPH Representation.

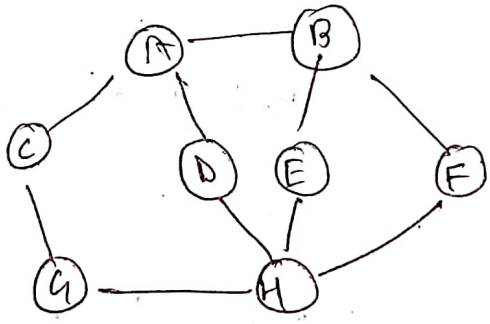
To store graph on Computer Memory we make two lists i) for vertexs
ii) for edges

vertexlist

A
B
C
D
G
E
G
H

Edge list

A	B
A	C
A	D
B	E
B	F
C	G
D	H
E	H
F	H
G	H



this is undirected graph so we do not ~~to~~ need to make entry of GA ~~of~~ AC

we use dynamic list. In C++ like vector and Arraylist in Java

In C

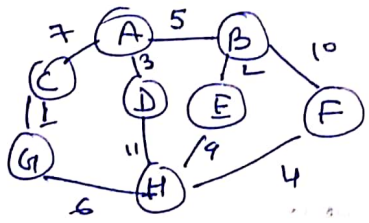
```
struct Edge
{
    char * startvertex;
    char * endvertex;
};
```

or

In C++ or Java

```
class Edge
{
    string startvertex;
    string endvertex;
};
```

If.



Edge List

A	B	5
A	C	7
A	D	3
B	E	2

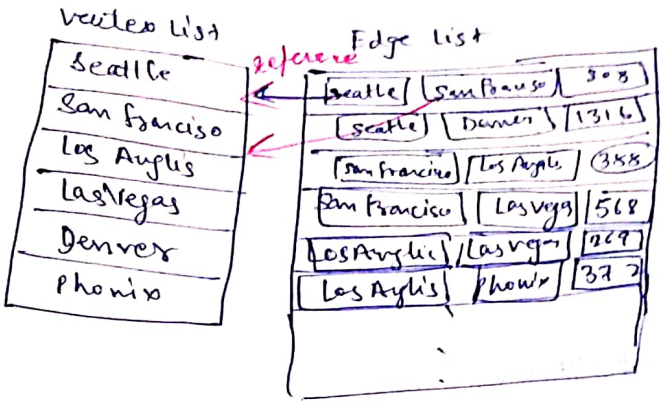
```
class Edge {
    string startvertex;
    string endvertex;
    int weight;
}
```

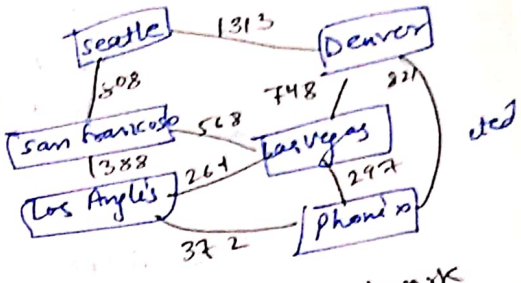
```
string vertex_list [max_size]
Edge edge_list [max_size];
```

Time Complexity:- Rate of growth of time taken with size of input or data

Space Complexity:- Rate of growth of Memory consumed with size of input or data

Time Complexity :-

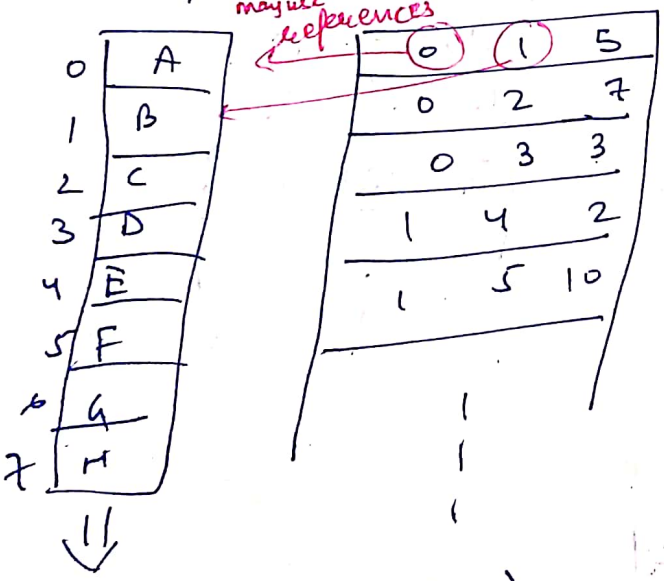




Intercity road network

O(no. of vertexs) if length is constant like A, B, C in previous example

Reference helps to save memory



O(|V|) O(|E|)

Memory usage

Space = O(|V| + |E|)

```
class Edge
{
    int startVertex;
    int endVertex;
    int weight;
};
```

Time Complexity :-

operation → How much time will you take to find all nodes adjacent to a given node?

Soln we have linear search over edge list

Runtime < O(|E|)

operation → find whether two given nodes are connected or not ?

Soln Linear search so O(|E|)

if |V| = n

then o≤|E| ≤ n(n-1) if directed.

$\frac{n(n-1)}{2}$ if undirected

n	E	
10	90	$O(V^2)$
100	9900	
1000	999000	

O[E) or O(|V|x|V|) - costly.

O(|V|) → ok

it is not efficient

so we store it in 2D Array.

Memory is high now $\Rightarrow O(V^2)$

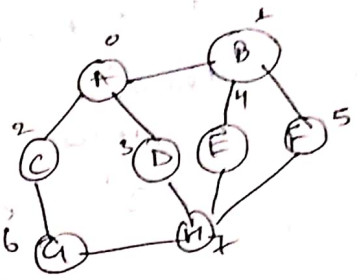
Adjacency matrix is good when graph is dense

Vertex list

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I

Adjacency Matrix

	0	1	2	3	4	5	6	7
0	0	1	1	1	0	0	0	0
1	1	0	0	0	1	1	0	0
2	1	0	0	0	0	0	1	0
3	1	0	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	0	1	1	1	1	0

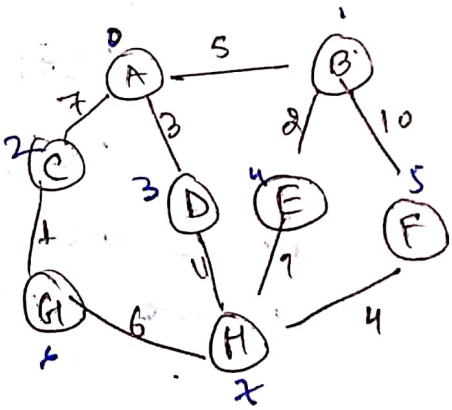


$A_{ij} = A_{ji}$

operation	Time cost	best
find adjacent nodes	$O(V) + O(V) \Rightarrow O(V)$	$O(1)$
find if two nodes are connected or not	$O(1) + O(V) = O(V)$	

use hash table to avoid this

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞



Memory is high now $\Rightarrow O(V^2)$

Adjancey matrix is good when graph is dense otherwise there is wastage of memory and Most world example graph are Sparse

Adjancey List Representation is better solution

TIME COMPLEXITY

eg
n prime ?
2
3
5
7
11

Ram

for i=2 to n-1
if i divides n
n is not prime

Shyam

for i=2 to $\sqrt{n}$
if i dives n
n is not prime

(n-2) times division

(√n-1) divisions

Lets take 1ms for 1 division

n=11 — 9 ms — (3-1) = 2 ms

n=101 — 99 ms — 9 ms

n = 1000003 — ≈ 10^6 ms = 10^3 sec = 16.66 min — $(\sqrt{10^6+3} - 1)$ ≈ 10^3 ms = 1 sec.

n = $10^{10}+19$ — ≈ 10^{10} ms ⇒ 10^7 sec ≈ 115 days — ≈ 10^5 ms = 100 sec = 1.66 min

T ∝ n so O(n)

T ∝ √n
so O(√n)

Running time depends on :-

(i) Single vs multi Processor
(ii) Read/Write speed to memory
3) 32 bit or 64 bit architecture
4) Input to algorithm

When calculate we only consider of Input
* rate of growth of time with respect to Input

Mode machine (we assume)
- single processor
- 32 bit
- sequential execution
- 1 unit time for arithmetic and logical operation
- 1 unit for assignment and return

```
Sum(a,b)
{
    return a+b;      T = 2 (irrespective of input)
}
```

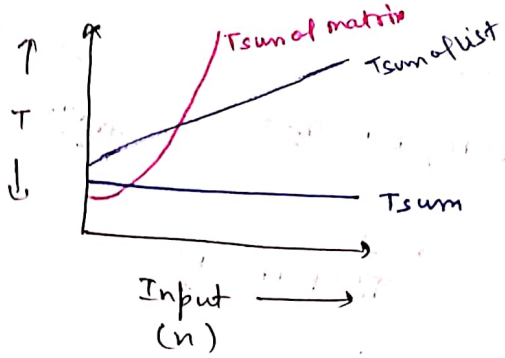
```
Sum (A,n)                          Cost    no. of times
{
1   total=0                        1 (c1)    1
2   for (i=0 to n-)                2 (c2)    n+1
3       total = total+A;           2 (c3)    n
4   return total                   1 (c4)    1
}
```

$$T_{sum} = 1 + 2(n+1) + 2n + 1$$

$$= 4n + 4$$

$$T(n) = cn + c' \qquad O(n)$$

$T_{sum\ of\ matrix} = an^2 + bn + c$ $\quad \Theta(n^2)$

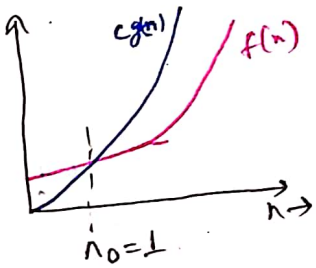


Asymptotic notations

O - "big-oh" notation [it gives upper bound]

$O(g(n)) = \{ f(n)$: there exist constants c and n_0

$f(n) \leq cg(n)$, for $n \geq n_0 \}$



ex $f(n) = 5n^2 + 2n + 1 = O(n^2)$

$g(n) = n^2$

$c = 8 = [5+2+1]$, $f(n) \leq 8n^2$, $n \geq 1$

$n_0 = 1$

Ω - omega notation

$\Omega(g(n)) = \{ f(n)$: their exist constants c and n_0

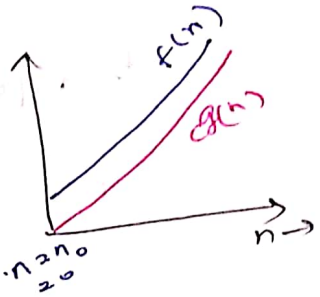
$cg(n) \leq f(n)$, for $n \geq n_0 \}$

ex $f(n) = 5n^2 + 2n + 1$

$g(n) = n^2$

$c = 5 \quad n_0 = 0$

$5n^2 \leq f(n), \quad n \geq 0$



Θ - Theta notation (Tight bound)

$\Theta(g(n)) = \{ f(n)$: there exist constants c_1, c_2 and n_0

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

for $n \geq n_0$

$\}$

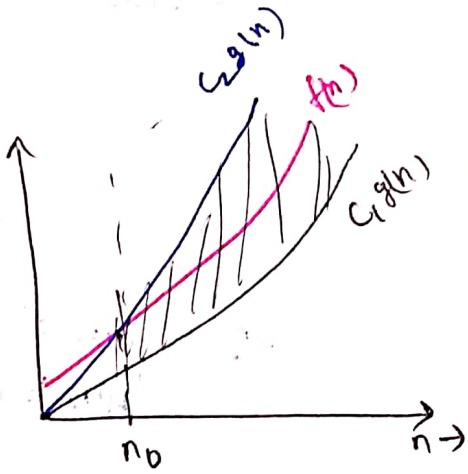
ex $f(n) = 5n^2 + 2n + 1$

$g(n) = n^2$

$c_1 = 5 \quad c_2 = 8 \quad , n_0 = 1$

$5n^2 < f(n) \leq 8n^2$ at $n \geq 1$

So $\Theta(n^2)$



We analyze time complexity for
(a) very large input size
(b) Worst case scenario

$T(n) = n^3 + 3n^2 + 4n + 2$
$\simeq n^3 \quad (n \to \infty)$
$O(n^3)$

```
Int a ;
a = 5
a++ ;
```

$O(1)$

```
for(i=0; i<n; i++)
{
}
```

$O(n)$

```
for( ) -> n
{
  for( ) -> n
  {
  }
}
```

$O(n^2)$

$T(n) = T(n-1) + 3 \quad$ if $n > 0$
$T(0) = 1$

$T(n) = T(n-1) + 3$
$= T(n-2) + 6$
$= T(n-3) + 9$
$= T(n-k) + 3k$

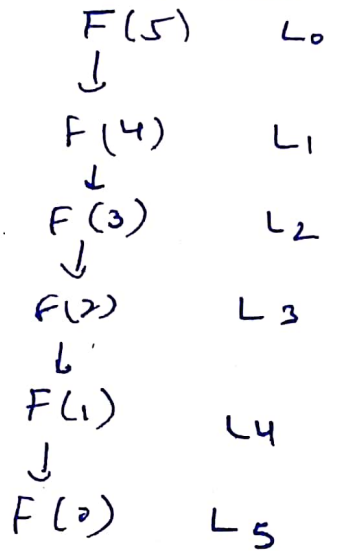
$n - k = 0 \Rightarrow k = n$

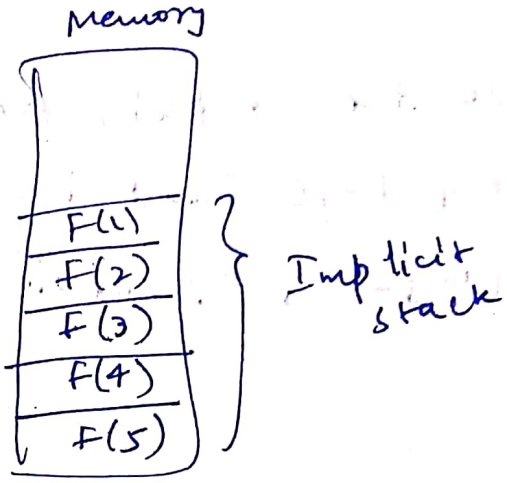
$T(n) = T(0) + 3n$
$= 3n + 1 \Rightarrow O(n)$

for factorial

```
F(n)
{
  if n==0
    return 1
  else
    n * f(n-1)
}
```

Recursion tree





max depth = 5

Space required $\propto$ input n

$O(n)$

HASH TABLE IMPLEMENTATION

```
#include <iostream>
#include <map>
#include <string.h>
using namespace std;
int main()
{
    map<string, string> phonebook;
    int n;
    cin>>n;
    string name, phone;

  for(int i=0; i<n; i++)
  {
      cin >> name >> phone;
     phonebook[name] = phone;
}

map<string, string>::iterator it;
string query;
while(cin>>query)
{
    it = phonebook.find(query);
    if(it != phonebook.end())
    {
        cout << it -> first << "=" << it -> second << '\n';
    }
    else{
        cout << "not found";
    }
}
```

Vishal Gupta

Android Developer

Android Developer with over 1+ year of software and web development experience, including Health Care , Taxi, E Commerce, Crypto Trading type applications . Understanding of Android software development life cycle. Dedicated to continuously developing and adopting new technologies to maximize development efficiency and produce innovative applications

Personal Info

Address
U-46,Dlf phase 3, Block U ,
Gurugram , Haryana

Phone
8218058029 / 9760294877

E-mail
vishalguptahmh@gmail.com

Date of birth
28-07-1995

GitHub
https://github.com/vishalguptahmh

LinkedIn
https://www.linkedin.com/in/vishalguptahmh/

Education

05-2013 - 05-2017
DIT University,Dehradun
B-Tech in Computer Science CGPA-7.89

Certificates

Certified Information Security Specialist V2.0(15dec13 - 01Jan14)

'Core java & Advance Java' at RCPL (28Jan15 - 12Sep15)

Cambridge English Entry level certificate in ESOL International

National Youth summit (Vision 2k35) (12Apr2014)

Experience

06-2018 - present
Talentopedia
Android Developer

Talentopedia Working on Product with latest technologies and still in Development Phaze
https://play.google.com/store/apps/details?id=com.moode&hl=en

Roles and Responsibilities
1)Building Complete App from Scratch

01-2017 - 06-2018
Charpixel Technologies
Android Developer

KidCircle :E-commerce Android application where parents can track their children schedules and kids extracurricular activities and suggest activities that are convenient to them
https://play.google.com/store/apps/details?id=com.product.kidcircle

Roles and Responsibilities.
1)Building and Editing of the Profile section
2)'Kids Activity' Booking Flow.
3)Integration of Google Payment Gateway

Medrexa : Social networking app for health users and professionals. It also provides free health data sharing and health data management service. It gives significant opportunity to health users to follow and get connected with the best health professionals.
https://play.google.com/store/apps/details?id=com.medrexa.user&hl=en
https://play.google.com/store/apps/details?id=com.medrexa.professional&hl=en

Roles and Responsibilities:
1) Integration Of APIs with the help of retrofit,dagger2.
2)Creating A feature of Tasks assigned to Doctors.
3)Building and Editing of Profile Section.
4)Editing of onGoing Appointment.

Taxi App: It is an application which is similar to uber or ola. I developed Driver part of Taxi App. It includes Accept/Reject/Cancel Request, starts the trip, ends the Trip, payment gateways, Location tracking. (It is part of company product)
https://drive.google.com/open?id=0B5kUF-_o9QNpLWFKb0V5cEdhakk

Roles and Responsibilities:
1)Develop Driver part of Taxi App
2)Integration of APIs using retrofit,dagger2.
3)Drawing a path on Map from pickup to drop point.
4)Building the process of Accepting and Rejecting of Bookings.

Cryptongy: Trade your Crypto coins and tokens in your preferred exchange. It includes customized watch list, views your wallet, or put buy and sell limit orders. Use the power of conditional orders to configure stop loss, trailer stop loss, and profit limit, you may also set alerts on the Crypto prices.
https://apkpure.com/cryptongy/crypto.soft.cryptongy

Roles and Responsibilities:
1)Create and developed the app in MVP
2)Store Data using Realm Database
3)Integration of Bittrex and Binnace Socket APIs.

GITHUB : Android Frequently used Stuffs
https://github.com/vishalguptahmh/AndroidCheatSheet

05-2016 - 07-2016
PHP Developer
Editsoft Solutions

- Learned the basics of PHP development
- Learned the basics of GIT
- Implementation of Google and Facebook Login
- Learned to complete the project with in the strict time line