INDEX

1. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

Aim : To apply Kruskal's Algorithm for computing the minimum spanning tree is directly based on the generic MST algorithm. It builds the MST in forest.

Definition: Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This mean it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest .It is an example of a greedy algorithm**.**

Efficiency:With an efficient sorting algorithm,the time efficiency of Kruskal's algorithm will be in O(|E|log|E|)

Algorithm

Start with an empty set A, and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in graph.

- Initially, each vertex is in its own tree in forest.
- Then, algorithm consider each edge in turn, order by increasing weight.
- If an edge (*u*, *v*) connects two different trees, then (*u*, *v*) is added to the set of edges of the MST, and two trees connected by an edge (*u*, *v*) are merged into a single tree.
- On the other hand, if an edge (*u*, *v*) connects two vertices in the same tree, then edge (*u*, *v*) is discarded.

Kruskals algorithm can be implemented using **disjoint set** data structure or **priority queue** data structure.

I Kruskal's algorithm implemented with disjoint-sets data structure.

MST_KRUSKAL (G, w)

1. A ← {} // A will ultimately contains the edges of the MST
2. for each vertex v in V[G]
3. do Make_Set (*v*)
4. Sort edge of E by nondecreasing weights w
5. for each edge (*u*, *v*) in E
6. do if FIND_SET (*u*) ≠ FIND_SET (*v*)
7. then A = AU{(*u*, *v*)}
8. UNION (*u*, *v*)
9. Return A

Program:

```
#include<stdio.h>
#define INF 999
#define MAX 100

int p[MAX],c[MAX][MAX],t[MAX][2];

int find(int v)
{
        while(p[v])
        v=p[v];
        return v;
}

void union1(int i,int j)
{
        p[j]=i;
}

void kruskal(int n)
{
        int i,j,k,u,v,min,res1,res2,sum=0;
        for(k=1;k<n;k++)
        {
                min=INF;
                for(i=1;i<n-1;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(i==j)continue;
                                if(c[i][j]<min)
                                {
                                u=find(i);
                                        v=find(j);
                                        if(u!=v)
                                        {
                                                res1=i;
                                                res2=j;
                                                min=c[i][j];
                                        }
                                }
                        }
                }
                union1(res1,find(res2));
                t[k][1]=res1;
                t[k][2]=res2;
                sum=sum+min;
        }
```

```
        printf("\nCost of spanning tree is=%d",sum);
        printf("\nEdgesof spanning tree are:\n");
        for(i=1;i<n;i++)
        printf("%d -> %d\n",t[i][1],t[i][2]);
}

int main()
{
        int i,j,n;
        printf("\nEnter the n value:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        p[i]=0;
        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&c[i][j]);
        kruskal(n);
        return 0;
}
```

Input/Output:
Enter the n value:5
Enter the graph data:
0 10 15 9 999
10 0 999 17 15
15 999 0 20 999
9 17 20 0 18
999 15 999 18 0
Cost of spanning tree is=49

Edgesof spanning tree are:
1 -> 4
1 -> 2
1 -> 3
2 -> 5

Program 2 Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

Aim: To find minimum spanning tree of a given graph using prim's algorithm

Definition: Prim's is an algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all edges in the tree is minimized. Prim's algorithm is an example of a greedy algorithm.

Algorithm

MST_PRIM (G, *w*, *v*)

1. Q ← V[G]
2. for each *u* in Q do
3. key [*u*] ← ∞
4. key [*r*] ← 0
5. π[*r*] ← NIl
6. while queue is not empty do
7. *u* ← EXTRACT_MIN (Q)
8. for each *v* in Adj[*u*] do
9. if v is in Q and *w*(*u*, *v*) < key [*v*]
10. then π[*v*] ← *w*(*u*, *v*)
11. key [*v*] ← *w*(*u*, *v*)

Analysis

The performance of Prim's algorithm depends of how we choose to implement the priority queue Q.

Program:

```
#include<stdio.h>
// #include<conio.h>
#define INF 999

int prim(int c[10][10],int n,int s)
{
	int v[10],i,j,sum=0,ver[10],d[10],min,u;
	for(i=1;i<=n;i++)
	{
		ver[i]=s;
		d[i]=c[s][i];
		v[i]=0;
	}
	v[s]=1;
	for(i=1;i<=n-1;i++)
	{
		min=INF;
		for(j=1;j<=n;j++)
		if(v[j]==0 && d[j]<min)
		{
			min=d[j];
			u=j;
```

```
                }
                v[u]=1;
                sum=sum+d[u];
                printf("\n%d -> %d sum=%d",ver[u],u,sum);
                for(j=1;j<=n;j++)
                if(v[j]==0 && c[u][j]<d[j])
                {
                        d[j]=c[u][j];
                        ver[j]=u;
                }
        }
        return sum;
}

void main()
{
        int c[10][10],i,j,res,s,n;
        clrscr();
        printf("\nEnter n value:");
        scanf("%d",&n);
        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&c[i][j]);
        printf("\nEnter the souce node:");
        scanf("%d",&s);
        res=prim(c,n,s);
        printf("\nCost=%d",res);
        getch();
}
```

Input/output:

```
Enter n value:3
Enter the graph data:
0 10 1
10 0 6
1 6 0
Enter the souce node:1
1 -> 3 sum=1
3 -> 2 sum=7
Cost=7
```

Program 3

a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

Definition: The **Floyd algorithm** is a graph analysis algorithm for finding shortest paths in a weighted graph (with positive or negative edge weights). A single execution of the algorithm will find the lengths

(summed weights) of the shortest paths between *all* pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of dynamic programming

Algorithm:

Floyd's Algorithm

Accept no .of vertices

Call graph function to read weighted graph // w(i,j)

Set D[ ] <- weighted graph matrix // get D {d(i,j)} for k=0

// If there is a cycle in graph, abort. How to find?

Repeat for k = 1 to n

Repeat for i = 1 to n

Repeat for j = 1 to n

D[i,j] = min {D[i,j], D[i,k] + D[k,j]}

Print D

Program A:

```
#include<stdio.h>
#include<conio.h>
#define INF 999

int min(int a,int b)
{
	return(a<b)?a:b;
}

void floyd(int p[][10],int n)
{
	int i,j,k;
	for(k=1;k<=n;k++)
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}

void main()
{
	int a[10][10],n,i,j;
	clrscr();
	printf("\nEnter the n value:");
	scanf("%d",&n);
	printf("\nEnter the graph data:\n");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	scanf("%d",&a[i][j]);
	floyd(a,n);
	printf("\nShortest path matrix\n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		printf("%d ",a[i][j]);
```

```
                printf("\n");
        }
        getch();
}
```

Input/Output:

Enter the n value:4
Enter the graph data:
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0
Shortest path matrix
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

Definition:The Floyd-Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph. A single execution of the algorithm will find the lengths of the shortest path between all pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of Dynamic programming.

Algorithm

//Input: Adjacency matrix of digraph
//Output: R, transitive closure of digraph
Accept no .of vertices
Call graph function to read directed graph
Set R[ ] <- digraph matrix // get R {r(i,j)} for k=0
Print digraph
Repeat for k = 1 to n
Repeat for i = 1 to n
Repeat for j = 1 to n
R(i,j) = 1 if
{$r_{ij}^{(k-1)}$ = 1 OR
$r_{ik}^{(k-1)}$ = 1 and $r_{kj}^{(k-1)}$ = 1}
Print R

Program:

```
#include<stdio.h>
void warsh(int p[][10],int n)
{
        int i,j,k;
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        p[i][j]=p[i][j] || p[i][k] && p[k][j];
}
```

```
int main()
{
        int a[10][10],n,i,j;
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        warsh(a,n);
        printf("\nResultant path matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",a[i][j]);
                printf("\n");
        }
        return 0;
        }
```

Input/Output:

Enter the n value:4
Enter the graph data:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Resultant path matrix
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1

Program 4 Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

Aim:To find shortest path using Dijikstra's algorithm.

Definition: Dijikstra's algorithm -For a given source vertex(node) in the graph, the algorithm finds the path with lowest cost between that vertex and every other vertex. It can also be used for finding cost of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

Efficiency:1)$\theta(|V|^2)$-graph represented by weighted matrix and priority queue as unordered array
2)O(|E|log|v|)-graph represented by adjacency lists and priority queue as min-heap

Algorithm: Dijikstra(G,s)

//Dijikstra's algorithm for single source shortest path
//input:A weighted connected graph with non negative weights and its vertex s
//output:The length d_v of a shortest path from s to v and penultimate vertex p_v for every vertex v in V
Initialize(Q)

for every vertex v in V do
d_v<-∞;P_v<-null
Insert(Q,v,d_v)
D_s<-0; Decrease(Q,s,d_s);V_T<-ǿ
for i<- 0 to |V|-1 do
u^*<-DeleteMin(Q)
V_T<-V_T U{u^*}
For every vertex u in V-V_T that is adjacent to u^* do
If d_{u*}+w(u^*,u)<d_u
d_u<- d_{u*}+w(u^*,u); p_u<-u^*
Decrease(Q,u,d_u)

Program:

```
#include<stdio.h>
#define INF 999
void dijkstra(int c[10][10],int n,int s,int d[10])
{
	int v[10],min,u,i,j;
	for(i=1;i<=n;i++)
	{
		d[i]=c[s][i];
		v[i]=0;
	}
	v[s]=1;
	for(i=1;i<=n;i++)
	{
		min=INF;
		for(j=1;j<=n;j++)
		if(v[j]==0 && d[j]<min)
		{
			min=d[j];
			u=j;
		}
		v[u]=1;
		for(j=1;j<=n;j++)
		if(v[j]==0 && (d[u]+c[u][j])<d[j])
		d[j]=d[u]+c[u][j];
	}
}

int main()
{
	int c[10][10],d[10],i,j,s,sum,n;
	printf("\nEnter n value:");
	scanf("%d",&n);
	printf("\nEnter the graph data:\n");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
```

```
        scanf("%d",&c[i][j]);
        printf("\nEnter the souce node:");
        scanf("%d",&s);
        dijkstra(c,n,s,d);
        for(i=1;i<=n;i++)
        printf("\nShortest distance from %d to %d is %d",s,i,d[i]);
  return 0;
}
```

Input/Output

Enter n value:6
Enter the graph data:
0 15 10 999 45 999
999 0 15 999 20 999
20 999 0 20 999 999
999 10 999 0 35 999
999 999 999 30 0 999
999 999 999 4 999 0
Enter the souce node:2
Shortest distance from 2 to 1 is 35
Shortest distance from 2 to 2 is 0
Shortest distance from 2 to 3 is 15
Shortest distance from 2 to 4 is 35
Shortest distance from 2 to 5 is 20
Shortest distance from 2 to 6 is 999

Output:

enter the no. of nodes:

6

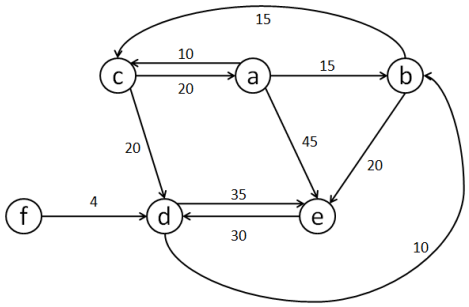
enter the cost adjacency matrix,'9999' for no direct path

0 15 10 9999 45 9999
9999 0 15 9999 20 9999
20 9999 0 20 9999 9999
9999 10 9999 0 35 9999
9999 9999 9999 30 0 9999
9999 9999 9999 4 9999 0

enter the starting vertex:

6

Shortest path from starting vertex to other vertices are



6->1=49
6->2=14
6->3=29
6->4=4
6->5=34
6->6=0

Program 5 Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph

Aim: To find topological ordering of given graph

Definition:**Topological ordering that for every edge in the graph,the vertex where the edge starts is listed before the edge where the edge ends.**

Algorithm:

1.repeatedly identify in a remaining digraph a source which is a vertex with no incoming edges and delete it along with all edges outgoing from it

2.Thje order in which the vertices are deleted yields a solution to the topological sorting.

```
#include<stdio.h>
// #include<conio.h>
int temp[10],k=0;
void sort(int a[][10],int id[],int n)
{
int i,j;
for(i=1;i<=n;i++)
{
if(id[i]==0)
{
id[i]=-1;
temp[++k]=i;
for(j=1;j<=n;j++)
{
if(a[i][j]==1 && id[j]!=-1)
id[j]--;
}
i=0;}}}
void main()
{
int a[10][10],id[10],n,i,j;
// clrscr();
printf("\nEnter the n value:");
```

```
scanf("%d",&n);
for(i=1;i<=n;i++)
id[i]=0;
printf("\nEnter the graph data:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
if(a[i][j]==1)
id[j]++;
}
sort(a,id,n);
if(k!=n)
printf("\nTopological ordering not possible");
else
{
printf("\nTopological ordering is:");
for(i=1;i<=k;i++)
printf("%d ",temp[i]);
}
// getch();
}
```

Input/output:
Enter the n value:6
Enter the graph data:
001100
000110
000101
000001
000001
000000

Topological ordering is:1 2 3 4 5 6

Program 6 Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

Aim: To implement 0/1 Knapsack problem using Dynamic programming

Definition: using **Dynamic programming**

It gives us a way to design custom algorithms which systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency).

We are given a set of *n* items from which we are to select some number of items to be carried in a knapsack(BAG). Each item has both a *weight* and a *profit*. The objective is to choose the set of items that fits in the knapsack and maximizes the profit.

Given a knapsack with maximum capacity *W*, and a set *S* consisting of *n* items , Each item *i* has some weight *wi* and benefit value *bi* (all *wi , bi* and *W* are integer values)

Problem: How to pack the knapsack to achieve maximum total value of packed items?

ALGORITHM

//(n items, W weight of sack) Input: n, $w_{i,,}$, v_i and W – all integers
//Output: V(n,W)
// Initialization of first column and first row elements
Repeat for i = 0 to n
set V(i,0) = 0
Repeat for j = 0 to W
Set V(0,j) = 0
//complete remaining entries row by row
Repeat for i = 1 to n
repeat for j = 1 to W
if (w_i <= j) V(i,j)) = max{ V(i-1,j), V(i-1,j-w_i) + v_i }
if (w_i > j) V(i,j) = V(i-1,j)
Print V(n,W)

PROGRAM:

```
#include<stdio.h>
int w[10],p[10],n;

int max(int a,int b)
{
        return a>b?a:b;
}
int knap(int i,int m)
{
        if(i==n) return w[i]>m?0:p[i];
        if(w[i]>m) return knap(i+1,m);
        return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}
int main()
{
        int m,i,max_profit;
        printf("\nEnter the no. of objects:");
        scanf("%d",&n);
        printf("\nEnter the knapsack capacity:");
        scanf("%d",&m);
        printf("\nEnter profit followed by weight:\n");
        for(i=1;i<=n;i++)
        scanf("%d %d",&p[i],&w[i]);
        max_profit=knap(1,m);
        printf("\nMax profit=%d",max_profit);
        return 0;
}
```

Input/Output:

```
Enter the no. of objects:4
Enter the knapsack capacity:6
Enter profit followed by weight:
```

78 2
45 3
92 4
71 5
Max profit=170

Program 7 Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

This program first calculates the profit-to-weight ratio for each item, then sorts the items based on this ratio in non-increasing order. It then fills the knapsack greedily by selecting items with the highest ratio until the knapsack is full. If there's space left in the knapsack after selecting whole items, it adds fractional parts of the next item. Finally, it prints the optimal solution and the solution vector.

Here's a simplified version of the C program to solve discrete Knapsack and continuous Knapsack problems using the greedy approximation method:

```
#include <stdio.h>
#define MAX 50

int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;


void greedyKnapsack(int n, int w[], int p[], int m) {
   double ratio[MAX];

   // Calculate the ratio of profit to weight for each item
   for (i = 0; i < n; i++) {
      ratio[i] = (double)p[i] / w[i];
   }

   // Sort items based on the ratio in non-increasing order
   for (i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (ratio[i] < ratio[j]) {
            double temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            int temp2 = w[i];
            w[i] = w[j];
            w[j] = temp2;

            temp2 = p[i];
            p[i] = p[j];
            p[j] = temp2;
         }
      }
```

```
    }

    int currentWeight = 0;
    maxprofit = 0.0;

    // Fill the knapsack with items
    for (i = 0; i < n; i++) {
        if (currentWeight + w[i] <= m) {
            x[i] = 1; // Item i is selected
            currentWeight += w[i];
            maxprofit += p[i];
        } else {
            // Fractional part of item i is selected
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }

    printf("Optimal solution for greedy method: %.1f\n", maxprofit);
    printf("Solution vector for greedy method: ");
    for (i = 0; i < n; i++)
        printf("%d\t", x[i]);
}

int main() {
    printf("Enter the number of objects: ");
    scanf("%d", &n);

    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);

    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

    printf("Enter the maximum capacity: ");
    scanf("%d", &m);

    greedyKnapsack(n, w, p, m);

    return 0;
}
```

Program 8 Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

AIM: An instance of the Subset Sum problem is a pair (S, t), where S = {x1, x2, ..., xn} is a set of positive integers and t (the target) is a positive integer. The decision problem asks for a subset of S whose sum is as large as possible, but not larger than t.

Algorithm: SumOfSub (s, k, r)

```
//Values of x[ j ], 1 <= j < k, have been determined
//Node creation at level k taking place: also call for creation at level K+1 if possible
// s = sum of 1 to k-1 elements and r is sum of k to n elements
//generating left child that means including k in solution
Set x[k] = 1
If (s + s[k] = d) then subset found, print solution
If (s + s[k] + s[k+1] <=d)
        then SumOfSum (s + s[k],  k+1,  r – s[k])
//Generate right child i.e. element k absent
If (s + r - s[k] >=d) AND  (s + s[k+1] )<=d
THEN { x[k]=0;
      SumOfSub(s, k+1, r – s[k])
```

Program:

```
#include<stdio.h>
// #include<conio.h>
#define MAX 10
int s[MAX],x[MAX],d;
void sumofsub(int p,int k,int r)
{
        int i;
        x[k]=1;
        if((p+s[k])==d)
        {
                for(i=1;i<=k;i++)
                if(x[i]==1)
                printf("%d ",s[i]);
                printf("\n");
        }
        else
        if(p+s[k]+s[k+1]<=d)
        sumofsub(p+s[k],k+1,r-s[k]);
        if((p+r-s[k]>=d) && (p+s[k+1]<=d))
        {
                x[k]=0;
                sumofsub(p,k+1,r-s[k]);
        }
}

int  main()
{
        int i,n,sum=0;
```

```
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the set in increasing order:");
        for(i=1;i<=n;i++)
        scanf("%d",&s[i]);
        printf("\nEnter the max subset value:");
        scanf("%d",&d);
        for(i=1;i<=n;i++)
        sum=sum+s[i];
        if(sum<d || s[1]>d)
        printf("\nNo subset possible");
        else
        sumofsub(0,1,sum);
        return 0;
}
```

Input/output:

```
Enter the n value:9
Enter the set in increasing order:1 2 3 4 5 6 7 8 9
Enter the max subset value:9
1 2 6
1 3 5
1 8
2 3 4
2 7
3 6
4 5
9
```

Program 9 Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Aim: Sort a given set of elements using Selection sort and determine the time required to sort elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

Definition: selection sort is a sorting routine that scans a list of items repeatedly and, on each pass, selects the item with the lowest value and places it in its final position. It is based on brute force approach. Sequential search is a $\Theta(n^2)$ algorithm on all inputs.

Algorithm:

```
SelectionSort(A[0…n-1])
//sort a given array by select5ion sort
//input:A[0…n-1]of orderable elements
Output:Array a[0…n-1] Sorted in ascending order
for i<- 0 to n-2 do
min<-i
for j<-i+1 to n-1 do
if A[j]<A[min] min<-j
```

swap A[i] and A[min]

program:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<dos.h>
void selsort(int a[],int n)
{
        int i,j,small,pos,temp;
        for(j=0;j<n-1;j++)
        {
                small=a[j];
                pos=j;
                for(i=j+1;i<n;i++)
                {
                        if(a[i]<small)
                        {
                                small=a[i];
                                pos=i;
                        }
                }
                temp=a[j];
                a[j]=small;
                a[pos]=temp;
        }
}
void main()
{
        int a[10],i,n;
        clock_t start,end;
        float dura;
        clrscr();
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the array:");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        start=clock();
        selsort(a,n);
        delay(100);
        end=clock();
        dura=(end-start)/CLK_TCK;
        printf("\nTime taken is:%f",dura);
        printf("\nSorted array is:");
        for(i=0;i<n;i++)
        printf("%d ",a[i]);
        getch();
}
```

Input /output:
Enter the n value:5
Enter the array:10 2 3 15 12
Time taken is:0.109890
Sorted array is:2 3 10 12 15

Program 10 Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Aim:

The aim of this program is to sort 'n' randomly generated elements using Quick sort and Plotting the graph of the time taken to sort n elements versus n.

Definition: Quick sort is based on the Divide and conquer approach. Quick sort divides array according to their value. Partition is the situation where all the elements before some position s are smaller than or equal to A[s] and all the elements after position s are greater than or equal to A[s].

Efficiency: $C_{best}(n) \in \Theta(n\log_2 n)$, $C_{worst}(n) \in \Theta(n^2)$, $C_{avg}(n) \in 1.38 n\log_2 n$

Algorithm: Quick sort (A[l….r])

```
// Sorts a sub array by quick sort
//Input : A sub array A[l..r] of A[0..n-1] ,defined by its left and right indices l
//and r
// Output : The sub array A[l..r] sorted in non decreasing order
        if l < r
           s = Partition (A[l..r]) //s is a split position
           Quick sort (A[l …s-1])
           Quick sort (A[s+1…r])

ALGORITHM Partition (A[l…r])
//Partition a sub array by using its first element as a pivot
// Input : A sub array A[l…r] of A[0…n-1] defined by its left and right indices l and // r (l < r)
// Output : A partition of A[l…r], with the split position returned as this function's value
p=A[l]
i=l;
j=r+1;
repeat
   delay(500);
   repeat i= i+1 until A[i] >= p
   repeat j=j-1 until A[J] <= p
   Swap (A[i],A[j])
until l >=j
Swap (A[i],A[j]) // Undo last Swap when i>= j
Swap (A[l],A[j])
Return j
```

Program:

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<sys/time.h>
#include<time.h>
void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
int fnPartition(int [], int , int );
void fnQuickSort(int [], int , int );
inline void fnSwap(int*, int*);
inline void fnSwap(int *a, int *b)
{
int t = *a; *a = *b; *b = t;
}
/*********************************************************************
*********
*Function
: main
*Input parameters:
*
int argc - no of commamd line arguments
*
char **argv - vector to store command line argumennts
*RETURNS
:
0 on success
*********************************************************************
********/
int main( int argc, char **argv)
{
FILE *fp;
struct timeval tv;
double dStart,dEnd;
int iaArr[500000],iNum,iPos,iKey,i,iChoice;
for(;;)
{
printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
printf("\nEnter your choice\n");
scanf("%d",&iChoice);
switch(iChoice)
{
case 1:
fp = fopen("QuickPlot.dat","w");
for(i=100;i<100000;i+=100)
{
fnGenRandInput(iaArr,i);
gettimeofday(&tv,NULL);
dStart = tv.tv_sec + (tv.tv_usec/1000000.0);
fnQuickSort(iaArr,0,i-1);
gettimeofday(&tv,NULL);
dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);
fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);
}
fclose(fp);
```

```
printf("\nData File generated and stored in file < QuickPlot.dat >.\n
Use a plotting utility\n");
break;
case 2:
printf("\nEnter the number of elements to sort\n");
scanf("%d",&iNum);
printf("\nUnsorted Array\n");
fnGenRandInput(iaArr,iNum);
fnDispArray(iaArr,iNum);
fnQuickSort(iaArr,0,iNum-1);
printf("\nSorted Array\n");
fnDispArray(iaArr,iNum);
break;
case 3:
exit(0);
}
}
return 0;
}
/*********************************************************************
*********
*Function
: fnPartition
*Description
: Function to partition an iaArray using First element as Pivot
*Input parameters:
*
int a[] - iaArray to hold integers
*
int l
- start index of the subiaArray to be sorted
*
int r
- end index of the subiaArray to be sorted
*RETURNS
: integer value specifying the location of partition
*********************************************************************
********/
int fnPartition(int a[], int l, int r)
{
int i,j,temp;
int p;
p = a[l];
i = l;
j = r+1;
do
{
do { i++; } while (a[i] < p);
do { j--; } while (a[j] > p);
fnSwap(&a[i], &a[j]);
}
while (i<j);
```

```
fnSwap(&a[i], &a[j]);
fnSwap(&a[l], &a[j]);
return j;
}
/*********************************************************************
*********
*Function
: fnQuickSort
*Description
: Function to sort elements in an iaArray using Quick Sort
*Input parameters:
*
int a[] - iaArray to hold integers
*
int l
- start index of the subiaArray to be sorted
*
int r
- end index of the subiaArray to be sorted
*RETURNS
: no value
*********************************************************************
******/
void fnQuickSort(int a[], int l, int r)
{
int s;
if (l < r)
{
s = fnPartition(a, l, r);
fnQuickSort(a, l, s-1);
fnQuickSort(a, s+1, r);
}
}
/*********************************************************************
*********
*Function
: GenRandInput
*Description
: Function to generate a fixed number of random elements
*Input parameters:
*
int X[] - array to hold integers
*
int n
- no of elements in the array
*RETURNS
:no return value
*********************************************************************
********/
void fnGenRandInput(int X[], int n)
{
int i;
```

```
srand(time(NULL));
for(i=0;i<n;i++)
{
X[i] = rand()%10000;
}
}
/*****************************************************************
*********
*Function
: DispArray
*Description
: Function to display elements of an array
*Input parameters:
*
int X[] - array to hold integers
*
int n
- no of elements in the array
*RETURNS
: no return value
*****************************************************************
********/
void fnDispArray( int X[], int n)
{
int i;
for(i=0;i<n;i++)
printf(" %5d \n",X[i]);
}
```

Quick.gpl

```
# Gnuplot script file for plotting data in file "QuickPlot.dat"
# This file is called Quick.gpl
set terminal png font arial
set title "Time Complexity for quick Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "QuickPlot.png"
plot "QuickPlot.dat" t 'Quick Sort' with lines
```

To compile and execute:

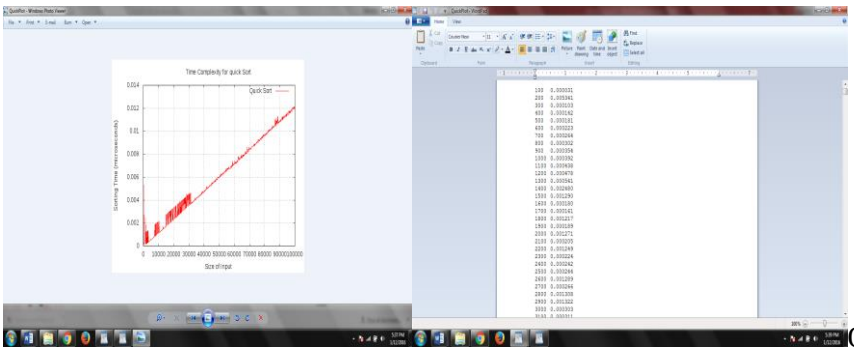
iselab@cselab-B85M-DS3H:~/Desktop/sample$ gcc quicksort2.c

iselab@cselab-B85M-DS3H:~/Desktop/sample$ gnuplot Quick.gpl

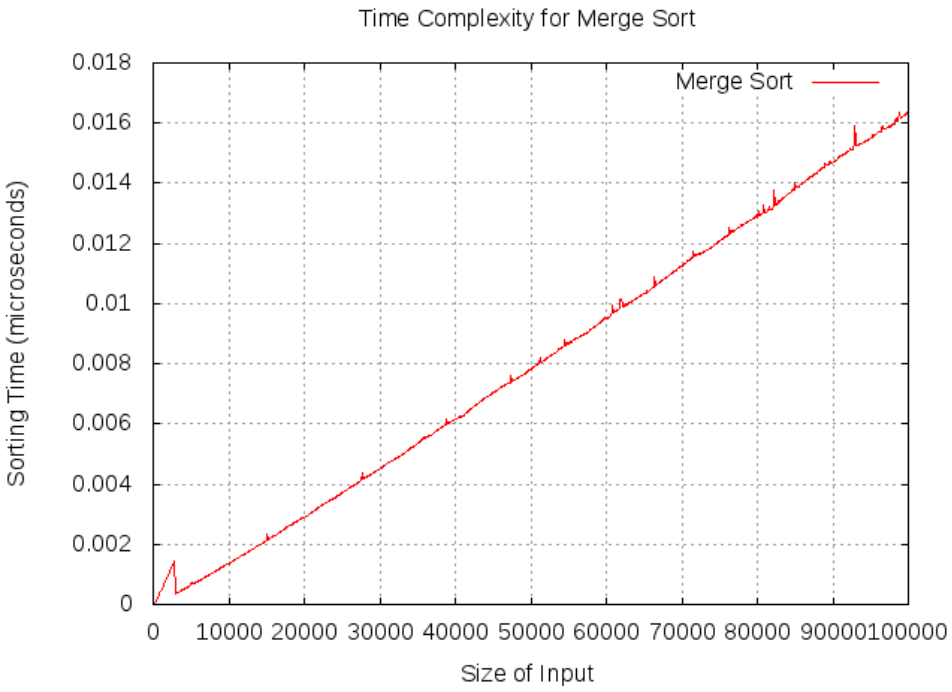
iselab@cselab-B85M-DS3H:~/Desktop/sample$ ls

This will create the QuickPlot.png file

```
iselab@cselab-B85M-DS3H:~/Downloads$ gcc mergesort.c
iselab@cselab-B85M-DS3H:~/Downloads$ ./a.out
1.Plot the Graph
2.MergeSort
3.Exit
Choice 1 shows the QuickPlot.dat file which is created automatically after entering the choice 1, this is
checked by giving ls cmd in the terminal
Enter your choice
1
Data File generated and stored in file < MergePlot.dat >.
 Use a plotting utility
1.Plot the Graph
2.MergeSort
3.Exit
Enter your choice
3
iselab@cselab-B85M-DS3H:~/Downloads$ ls
MergePlot.dat       mergesort.c, quicksort2.c  QuickPlot.dat
iselab@cselab-B85M-DS3H:~/Downloads$ gedit MergePlot.dat // to see the random values generated
iselab@cselab-B85M-DS3H:~/Downloads$ vi MergePlot.dat
Input and output:
Enter your choice
2
Enter the number of elements to sort
10
Unsorted Array
  7506
   741
  7150
  6997
  5247
  2059
  8915
 7327
  9897
  9867
Sorted Array
   741
  2059
  5247
  6997
  7150
  7327
  7506
  8915
  9867
  9897
QuickPlot.png
```



QuickPlot.dat



In the .dat file we have `99900 0.012101 random values incremented for every 100.`

Program 11 Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Aim:

The aim of this program is to sort 'n' randomly generated elements using Merge sort and Plotting the graph of the time taken to sort n elements versus n.

Definition: Merge sort is a sort algorithm based on divide and conquer technique. It divides the array element based on the position in the array. The concept is that we first break the list into two smaller lists of roughly the same size, and then use merge sort recursively on the subproblems, until they cannot subdivide anymore. Then, we can merge by stepping through the lists in linear time. Its time efficiency is Θ(n log n).

Algorithm: Merge sort (A[0…n-1]

// Sorts array A[0..n-1] by Recursive merge sort

```
// Input : An array A[0..n-1]  elements
// Output : Array A[0..n-1] sorted in non decreasing  order
  If  n  > 1
    Copy A[0…(n/2)-1] to B[0…(n/2)-1]
    Copy A[0…(n/2)-1] to C[0…(n/2)-1]
   Mergesort (B[0…(n/2)-1])
    Mergesort (C[0…(n/2)-1])
    Merge(B,C,A)

ALGORITHM Merge (B[0…p-1], C[0…q-1],A[0….p+q-1])
   // merges two sorted arrays into one sorted array
  // Input : Arrays B[0..p-1] and C[0…q-1] both sorted
 // Output : Sorted array A[0…. p+q-1] of the elements of B and C
    I = 0;
    J = 0;
    K= 0;
    While I < p and j < q do
      If  B[i] <=  C[j]
       A[k]= B[I];     I= I+1;
      Else
          A[k] = B[i];   I=i+1
     K=k+1;
     If  I = = p
         Copy  C[ j..q-1] to A[k….p+q-1]
     else
        Copy  B[I … p-1] to A[k …p+q-1
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
void fnMerge(int [], int ,int ,int);
void fnMergeSort(int [], int , int);

int main( int argc, char **argv)
{
    FILE *fp;
    struct timeval tv; // struct def is already there just creating
instance of it.

    double dStart,dEnd;
    int iaArr[500000],iNum,iPos,iKey,i,iChoice;
    for(;;)
    {
        printf("\n1.Plot the Graph\n2.MergeSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d",&iChoice);
```

```
        switch(iChoice)
        {
            case 1:
                fp = fopen("MergePlot.dat","w"); // to find order of
growth for x axis, it create
                for(i=100;i<100000;i+=100)
                {
           fnGenRandInput(iaArr,i);// hold up to 5lac values
          gettimeofday(&tv,NULL);  // precision up to microsec,
available in header file <sys/time.h> with struct in terminal:  give
man gettimeofday,

dStart = tv.tv_sec + (tv.tv_usec/1000000.0);// converting time into
sec, so add tv.tv_sec , if no .0 is given it end up in 0
                    fnMergeSort(iaArr,0,i-1);
                    gettimeofday(&tv,NULL);
                    dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);
                    fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);// i -size
of i/p, diff is time taken
                }
                fclose(fp);
                printf("\nData File generated and stored in file <
MergePlot.dat >.\n Use a plotting utility\n");
                break;
            case 2:
                printf("\nEnter the number of elements to sort\n");
                scanf("%d",&iNum);
                printf("\nUnsorted Array\n");
                fnGenRandInput(iaArr,iNum);
                fnDispArray(iaArr,iNum);
                fnMergeSort(iaArr,0,iNum-1);
                printf("\nSorted Array\n");
                fnDispArray(iaArr,iNum);
                break;
            case 3:
                exit(0);
        }
    }
return 0;
}

void fnMerge(int a[], int low,int mid,int high)
{
    int i,k,j,b[500000];
    i=k=low;
    j=mid+1;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
            b[k++]=a[i++];
        else
            b[k++]=a[j++];
```

```
    }
    while(i<=mid)
        b[k++]=a[i++];
    while(j<=high)
        b[k++]=a[j++];
    for(i=low;i<k;i++)
        a[i]=b[i];
}

void fnMergeSort(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid=(low+high)/2;
        fnMergeSort(a,low,mid);
        fnMergeSort(a,mid+1,high);
        fnMerge(a,low,mid,high);
    }
}

void fnGenRandInput(int X[], int n)
{
    int i;
    srand(time(NULL));      // if u dont use this fn it seeds 0 value,
value keeps changing -to get current time-used in time.h, it takes
different seed value to generate rand no.

    for(i=0;i<n;i++)
    {
        X[i] = rand()%10000;// defined in stdlib, pseuodo random no
generating sequence-random like
    }
}




void fnDispArray( int X[], int n)
{
    int i;
    for(i=0;i<n;i++)
    printf(" %5d \n",X[i]);
}
```

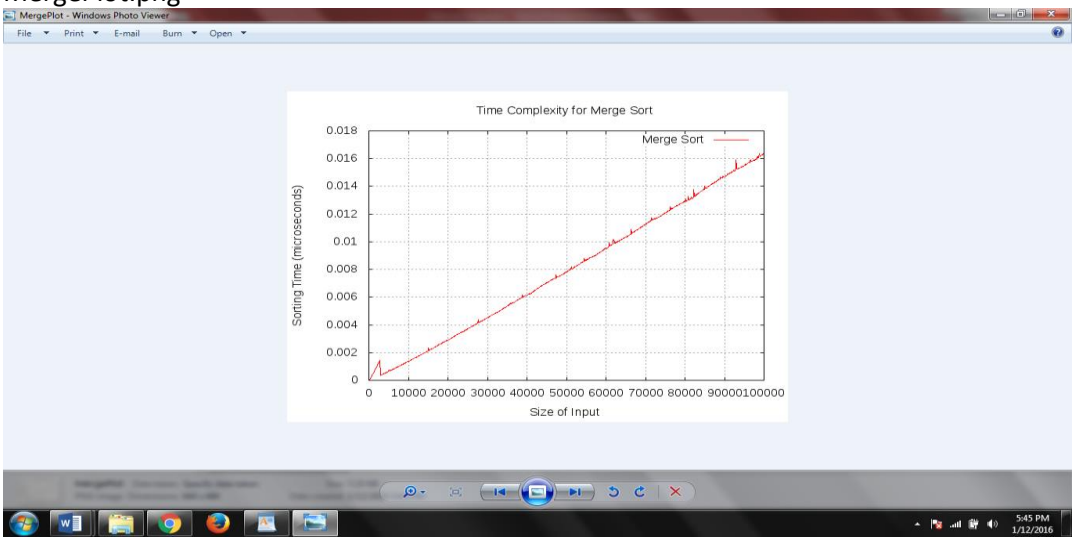
MergePlot.gpl

```
# Gnuplot script file for plotting data in file "MergePlot.dat"
# This file is called MergePlot.gpl
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
```

```
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "MergePlot.png"
plot "MergePlot.dat" t 'Merge Sort' with lines

Follow same command as above program quicksort.
```

Execution see the above program for procedure:
MergePlot.png



github.com/fsmk/CS-VTU-Lab-Manual
git clone https://github.com/fsmk/CS-VTU-Lab-Manual.git

sudo apt-get install git-core (give in terminal)
For help:
iselab@cselab-B85M-DS3H:~/Downloads$ man
What manual page do you want?
iselab@cselab-B85M-DS3H:~/Downloads$ man gettimeofday

Program 12 Design and implement C/C++ Program for N Queen's problem using Backtracking.
Aim: To implement N Queens Problem using Back Tracking
Definition:

The object is to place queens on a chess board in such as way as no queen can capture another one in a single move
Recall that a queen can move horz, vert, or diagonally an infinite distance
This implies that no two queens can be on the same row, col, or diagonal
We usually want to know how many different placements there are
Using Backtracking Techniques

Algorithm:
/* outputs all possible acceptable positions of n queens on n x n chessboard */
// Initialize x [ ] to zero

// Set k = 1 start with first queen
Repeat for i = 1 to n // try all columns one by one for k^{th} queen
if Place (k, i) true then
{
x(k) = i // place k^{th} queen in column i
if (k=n) **all queens placed and hence print output (x[ ])**
else NQueens(K+1,n) //try for next queen
}
Place (k,i)
/* finds if k^{th} queen in k^{th} row can be placed in column i or not; returns true if queen can be placed */
// x[1,2, . . . k-1] have been defined
//queens at (p, q) & (r, s) attack if |p-r| = |q-s|
Repeat for j = 1 to (k-1)
if any earlier j^{th} queen is in i^{th} column (x[j]= i)
or in same diagonal (abs(x[j] - i) = abs(j - k))
then k^{th} queen cannot be placed (return false)
return true (as all positions checked and no objection)

Program: correct program in ubuntu

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 50

int can_place(int c[],int r)
{
        int i;
        for(i=0;i<r;i++)
        if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))
        return 0;
        return 1;
}

void display(int c[],int n)
{
        int i,j;
        char cb[10][10];
        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        cb[i][j]='-';
        for(i=0;i<n;i++)
        cb[i][c[i]]='Q';
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                printf("%c",cb[i][j]);
                printf("\n");
```

```
        }
}

void n_queens(int n)
{
        int r;
        int c[MAX];
        c[0]=-1;
        r=0;
        while(r>=0)
        {       c[r]++;
                while(c[r]<n && !can_place(c,r))
                c[r]++;
                if(c[r]<n)
                {       if(r==n-1)
                        {       display(c,n);
                                printf("\n\n");
                        }
                        else
                        {       r++;
                                c[r]=-1;
                        }
                }
                else
                r--;
        }
}

void main()
{
        int n;
        clrscr();
        printf("\nEnter the no. of queens:");
        scanf("%d",&n);
        n_queens(n);
        getch();
}
```

Input/Output:

```
Enter the no. of queens:4
-Q--
---Q
Q---
--Q-

--Q-
Q---
---Q
-Q—
```

II)Sample Viva Questions and Answers

1) Explain what is an algorithm in computing?

An algorithm is a well-defined computational procedure that take some value as input and generate some value as output. In simple words, it's a sequence of computational steps that converts input into the output.

2) Explain what is time complexity of Algorithm?

Time complexity of an algorithm indicates the total time needed by the program to run to completion. It is usually expressed by using the **big O notation.**

3) The main measure for efficiency algorithm are-Time and space

4. The time complexity of following code:

```
int a = 0;
for (i = 0; i < N; i++) {
   for (j = N; j > i; j--) {
      a = a + i + j;
   }
}Ans O(n*n)
```

5) What does the algorithmic analysis count?

6) The number of arithmetic and the operations that are required to run the program

7) Examples of O(1) algorithms are_____

AMultiplying two numbers.

B assigning some value to a variable

C displaying some integer on console

8) Examples of O(n2) algorithms are_____.

AAdding of two Matrices

B Initializing all elements of matrix by zero

9) The complexity of three algorithms is given as: O(n), O(n2) and O(n3). Which should execute slowest for large value of n?

All will execute in same time.

10) In quick sort, the number of partitions into which the file of size n is divided by a selected record is 2.

The three factors contributing to the sort efficiency considerations are the efficiency in coding, machine run time and the space requirement for running the procedure.

11) How many passes are required to sort a file of size n by bubble sort method?

N-1

12) How many number of comparisons are required in insertion sort to sort a file if the file is sorted in reverse order?

A. N2

13) How many number of comparisons are required in insertion sort to sort a file if the file is already sorted? N-1
14) In quick sort, the number of partitions into which the file of size n is divided by a selected record is 2
15) The worst-case time complexity of Quick Sort is_____.O(n2)

16)The worst-case time complexity of Bubble Sort is_____.O(n2)

17) The worst-case time complexity of Merge Sort is_____.O(n log n)

18)The algorithm like Quick sort does not require extra memory for carrying out the sorting procedure. This technique is called _____.in-place

19)Which of the following sorting procedures is the slowest?
A. Quick sort
B. Heap sort
C. Shell sort
D. Bubble sort
20) The time factor when determining the efficiency of algorithm is measured by Counting the number of key operations

21)The concept of order Big O is important because
A. It can be used to decide the best algorithm that solves a given problem

22) The running time of insertion sort is
A.O(n^2)

23) A sort which compares adjacent elements in a list and switches where necessary is ____.
A. insertion sort

24) The correct order of the efficiency of the following sorting algorithms according to their overall running time comparison is
bubble>selection>insertion
25) The total number of comparisons made in quick sort for sorting a file of size n, is
A. O(n log n)

26)Quick sort efficiency can be improved by adopting
A. non-recursive method

27)For the improvement of efficiency of quick sort the pivot can be _____.
"the mean element"

28) What is the time complexity of linear search?**Θ(n)**

29) What is the time complexity of binary search?**Θ(log2n)**

30) What is the major requirement for binary search?

The given list should be sorted.

31). What are important problem types? (or) Enumerate some important types of problems.
1. Sorting 2. Searching
3. Numerical problems 4. Geometric problems
5. Combinatorial Problems 6. Graph Problems
7. String processing Problems

32). Name some basic Efficiency classes
1. Constant 2. Logarithmic 3. Linear 4. nlogn
5. Quadratic 6. Cubic 7. Exponential 8. Factorial

33) What are algorithm design techniques?
Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmatically, applicable to a variety of problems from different areas of computing. General design techniques are:(i) Brute force (ii) divide and conquer
(iii) decrease and conquer (iv) transform and concquer
(v) greedy technique (vi) dynamic programming
(vii) backtracking (viii) branch and bound

34). How is an algorithm's time efficiency measured?
Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation(running time) is executed. Basic operation is the most time consuming operation in thealgorithm's innermost loop.

35)Explain the greedy method.
Greedy method is the most important design technique, which makes a choice that looks best atthat moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraintsthat is the feasible solution. A greedy method suggests that one candevice an algorithm thatworks in stages considering one input at a time.

36). Define feasible and optimal solution.
Given n inputs and we are required to form a subset such that it satisfies some given constraintsthen such a subset is called feasible solution.A feasible solution either maximizes or minimizes the given objective function is called asoptimal solution

37). What are the constraints of knapsack problem?
To maximize $\sum pixi$
The constraint is : $\sum wixi \geq m$ and $0 \leq xi \leq 1$ $1 \leq i \leq n$
where m is the bag capacity, n is the number of objects and for each object i wiand pi are the weight and profit of object respectively.

38). Specify the algorithms used for constructing Minimum cost spanning tree.
a) Prim's Algorithm
b) Kruskal's Algorithm

39). State single source shortest path algorithm (Dijkstra's algorithm).
For a given vertex called the source in a weigted connected graph,find shotrtest paths to all its other vertices.Dijikstra's algorithm applies to graph with non-negative weights only.

40). State efficiency of prim's algorithm.
O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)
O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

41) State Kruskal Algorithm.
The algorithm looks at a MST for a weighted connected graph as an acyclic subgraph with |v|-1 edges for which the sum of edge weights is the smallest.

42) State efficiency of Dijkstra's algorithm.

O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)