Algorithm: Abdullah Muhammad bin Musa al-Khwarizmi was the founder of Algorithm. **Algorism** and **algorithm** stem from *Algoritmi*, the Latin form of his name Khwarizmi.

Fig: Founder of algorithm Musa al-Khwarizmi

Algorithm: An algorithm is defined as finite sequence of unambiguous instructions followed to accomplish a given task.

An algorithm must satisfy the following criteria:

Input: Each algorithm should have zero or more inputs. The range of inputs for which algorithm works should be satisfied.

Output: The algorithm should produce correct results. At least one output has to be produced.

Definiteness: Each instruction should be clear and unambiguous.

Effectiveness: The instructions should be simple and should transform the given input to the desired output.

Finiteness: The algorithm must terminate after a finite sequence of instruction.

Analysis framework:

The efficiency of the algorithm depends on two factors:

Space efficiency

Time efficiency

Space efficiency: The space efficiency of an algorithm is the amount of memory required to run program completely and efficiently.

Components that affect space efficiency:

Program space

Data space

Stack space

Time efficiency*:* The Time efficiency of an algorithm is measured purely on how fast a given algorithm is executed. Since the efficiency of an algorithm is measured using time, the word time complexity is often associated with an algorithm.

Components that affect space efficiency:

Speed of the computer

Choice of the programming language

Compiler used

Choice of the algorithm

Number of inputs/outputs

Size of inputs/outputs

Worst case, best case and average case efficiency

An item we are searching for may be present in the very first location itself.

In this case only one item is compared and this is the **best case.**

The item may be present some where in the middle which definitely takes some time. Running time is more when compared to the previous case for the same value of n. Since we do not know where the item is we have to consider the average number of cases and hence this situation is an **average case.**

The item we are searching for may not be present in the array requiring n number of comparisons and running time is more than the previous two cases. This may be considered as the **worst case.**

Asymptotic notations:

The value of a function may increase or decrease as the value of n increases.

The asymptotic behavior of a function is the study of how the value of a function varies for large value of n where n is the size of the input. Using the asymptotic behavior of a function, we can easily find the time efficiency of an algorithm.

Example Algorithm:

ALGORITHM GCD (M, N)

//description: computes GCD (M, N)

// input: M and N should be positive integers

//output: Greatest common divisor of M and N

While N! =0 do

R←M%N //compute the remainder

M←N //Assign N to M

N← R // Assign remainder to N

End while

Return M //Return the GCD

The algorithm design and analysis process is explained by considering the following flow chart:

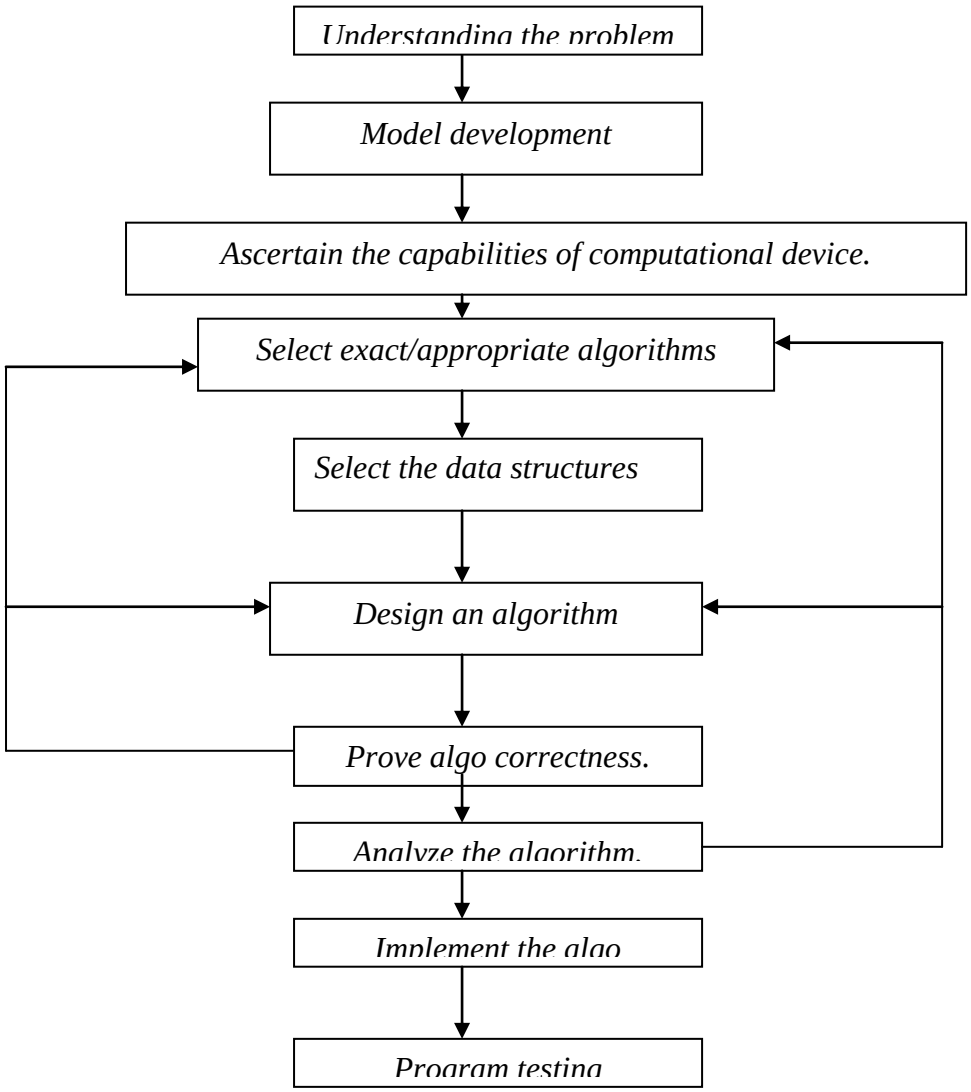

fig: Phases in program development

Parallel Programming:

In parallel programming, the processing is broken up into parts, each of which can be executed concurrently. The instructions from each part run simultaneously on different CPUs. These CPUs can exist on a single machine, or they can be CPUs in a set of computers connected via a network.

It is possible to write parallel programming for multiprocessors using MPI and OpenMP

OpenMP:

- OpenMp is an application programming interface (API) for parallel programming on multiprocessors
- It consists of a set of compiler directives and a library of support functions
- OpenMp works in conjunction with Standard Fortran, C or C++

Steps to execute parallel programming:

1. To write the program "vi filename.c"
2. For Compiling "cc –fopenmp filename.c"
3. To run the program ./a.out

Experiment No. 1 **Date: __ /__ / _____**

Quick Sort
(Sorting Problem)

Aim:
Sort a given set of elements using the Quick sort method and determine the time required to sort the 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. The elements can be read from a file or can be generated using the random number generator.

Design of Algorithm:

Algorithm: (Pseudo Code)

Quick Sort:

ALGORITHM Quicksort (A[l….r])

//Sorts a subarray by quicksort

//Input: A subarray A[l…r] of A[0…n-1], defined by its left & right indices l & r

//Output: Subarray A[l…r] sorted in increasing order

```
if l < r
    s ← Partition (A[l…r]) // s is a split position
    Quicksort ( A[l….s-1] )
    Quicksort (A[s+1…r])
```

ALGORITHM Partition (A[l…r])

//Partitions a subarray by using its first element as pivot

//Input: A subarray A[l…r] of A[0…n-1], defined by its left & right indices l & 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
        repeat i ← i +1   until A[i] ≥ p
        repeat j ← j - 1   until A[i] ≥ p
        swap (A[i], A[j])
    until i ≥ j
        //undo last swap when i ≥ j
        swap (A[l], A[j])
    return  j
```

Code: (Implementation)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
int ct,sw;

void quicksort(int a[],int low,int high)
{
   int s;
   //To check for boundary condition
   if(low<high)
  {
       s=partition(a,low,high);
       quicksort(a,low,s-1);
       quicksort(a,s+1,high);
   }
}

int partition(int a[],int low,int high)
{
       int p,i,j,temp;
       p=a[low];
       i=low+1;
       j=high;
       while(1)
       {
          while(a[i]<=p && i<high)
                 i++,ct++;     // count for the comparision
          while(a[j]>p)
                 j--,ct++;    // count for the comparision

          if(i<j)
         {
                sw++  // count for the swap
                 temp=a[i];
                 a[i]=a[j];
                 a[j]=temp;
         }
         else
         {
               sw++  // count for the swap
               temp=a[low];
               a[low]=a[j];
               a[j]=temp;
               return j;
         }
     }
}
```

```
void main()
{
      int a[50],i,n;
      clrscr();

      I:printf("=======================================\n\n");
        printf(" To sort the Integer elements using Quick Sort Method\n\n");
        printf("=======================================\n\n");
        printf("\nEnter the Number of elements to be sorted which is less than the
array size 50\n");
        scanf("%d",&n);

        if(n>0 && n<=50)   // array bound check
       {
                 randomize();
                 for(i=0;i<n;i++)
                {
                       a[i] = random(1000);
                 }
         }
        else
       {
                printf("Invalid input.. please try again...\n");
                clrscr();
                goto I;
       }

        printf("\n~~~~~~~~~~~ OUTPUT ~~~~~~~~~~~~~~~\n");
        printf("The Inputs generated by Random Number Generated are \n");
        for(i=0;i<n;i++)
            printf("%d  ",a[i]);

        quicksort(a,0,n-1);

        printf("\n Sorted Integer Array \n");
        for(i=0;i<n;i++)
                printf("%d  ",a[i]);

        printf("\n\n******* Analysis of Quick Sort Algorithm *******\n\n");
        printf("The Basic Operation of Quick Sort algorithm are Comparision and
        Swap ");
        printf(" \nThe number of Comparisions are : %d units",ct);
        printf(" \nThe number of Swaps are : %d units",sw);
        printf("\n\n ***** Thank You *****\n");
        getch();
  }
```

OUTPUT

```
================================================
    To sort the Integer elements using Quick Sort Method
================================================
Enter the Number of elements to be sorted which is less than the array size 50

5

~~~~~~~~~~ OUTPUT ~~~~~~~~~~~~~~~
The Inputs generated by Random Number Generated are

463 257 12 940 624

Sorted Integer array
12 257 463 624 940

********** Analysis of Quick Sort Algorithm ***************

The Basic Operation of Quick Sort algorithm are Comparision and Swap
The number of Comparisions are: 5 units
The number of Swaps are: 3 units

******************  Thank You ************************
```

Output:

Graph:

Viva Voce:

Evaluation:

Experiment No. 2 **Date: __ /__ / _____**

Merge sort
(Sorting Problem)

Aim:

Usin OpenMP, implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the 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. The elements can be read from a file or can be generated using the random number generator.

Algorithm:(Pseudo Code)

Merge Sort:

```
ALGORITHM  Mergesort (A [0...n-1])
//Sorts array A [0…n-1] by recursive mergesort
//Input: An array A[0…n-1] of orderable elements
//output: Array A[0…n-1] sorted in increasing 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)
```

Merging Two List:

```
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 & 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]←C[j] ;  j ← j+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]
```

Code:

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

void simple_merge(int a[],int low,int mid,int high);
void merge_sort(int a[],int low,int high);

void main()
{
	int a[200],i,n;
	double startTime,endTime;


	printf("============ Parallelized Merge Sort ==========\n");
	printf("Enter the number of elements\n");
	scanf("%d",&n);

	randomize();
	for(i=0;i<n;i++)
	{
		a[i]=random(1000);
	}

	printf("\n The random numbers generated are:\n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);

	startTime=omp_get_wtime();
	merge_sort(a,0,n-1);
	endTime = omp_get_wtime();

	printf("Time taken is %10.9f\n",(double)(endTime-startTime));
	printf("The sorted elements are:\n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);

}

void simple_merge(int a[],int low,int mid,int high)
{
	int i=low,j=mid+1,k=low,c[200];

	while(i<=mid && j<=high)
	{
		if(a[i]<a[j])
		{
			c[k]=a[i];
			i++;
```

```
                k++;
            }
            else
            {
                c[k]=a[j];
                j++;
                k++;
            }
        }
        while(i<=mid)
            c[k++]=a[i++];
        while(j<=high)
            c[k++]=a[j++];

        for(i=low;i<=high;i++)
            a[i]=c[i];
}
void merge_sort(int a[],int low,int high)
{
        int mid;
        if(low<high)
        {
            mid=(low+high)/2;
            #pragma omp parallel sections
            {
                #pragma omp section
                    merge_sort(a,low,mid);
                #pragma omp section
                    merge_sort(a,mid+1,high);
            }
            simple_merge(a,low,mid,high);
        }
}
```

Output:

========== Parallelized Merge Sort ==========

Enter the number of elements

5

The random numbers generated are:

463 257 12 940 624

Time taken is 0.000194626

The sorted elements are

12 257 463 624 940

Output:

Graph:

Viva Voce:

Evaluation:

Experiment No. 3a **Date: __ /__ / _____**

Topological Ordering
(Sorting Problem)

Aim:

Obtain the Topological ordering of vertices in a given digraph.

Algorithm: (Pseudo Code)

Step 1: Start

Step 2: Find the indegree of all the nodes

Step 3: Push all the nodes which is having the indegree zero

Step 4: Pop the element from the stack and delete all the edges outgoing from the deleted node

Step 5: If the indegree of the node is zero, push that node to the stack.

Step 6: Repeat steps 3 to 5 until the stack is empty.

Step 7: Stop

Code:

```
#include<stdio.h>
int a[10][10],indegree[10],v[10],stack[10],n,k,top=-1;
void main()
{
	int i;
	clrscr();
	printf("=======================================\n");
	printf(" To find the topological ordering of the vertices\n");
	printf("=======================================\n");
	readadj();
	topological();
	if(k==n)
	{
		printf("\n topological sequence is");
		for(i=0;i<n;i++)
		printf("\t%d",v[i]);
	}
	else
		printf("\n topological sequence doesnot exist");
	getch();
}
```

```
readadj()
{
        int i,j;
        I:printf("\n Enter the number of vertices of a graph\n");
        scanf("%d",&n);
        if(n>0 && n<50)
        {
          printf("\nEnter the adjacency matrix");
          for(i=0;i<n;i++)
          {
                for(j=0;j<n;j++)
                {
                  scanf("%d",&a[i][j]);
                }
          }
          return;
        }
        else
        {
          printf("Enter valid number of vertices\n");
          goto I;
        }
}

topological()
{
   int i,j,R;
    //To find the indegree
    for(i=0;i<n;i++)
     for(j=0;j<n;j++)
     {
       if(a[i][j]==1)
        indegree[j]++;
     }

    // Push all the nodes with indegree 0
    for(i=0;i<n;i++)
    {
       if(indegree[i]==0)
             stack[++top]=i;
    }

    while(top!=-1)
    {
       R = stack[top--];
       v[k++] = R;

       for(i=0;i<n;i++)
       {
                  if(a[R][i]==1)
              {
                indegree[i]--;
```

```
            if(indegree[i]==0)
               stack[++top] = i;
         }
      }
   }
}
```

Output:

```
==============================================
To find the topological ordering of the vertices
==============================================
Enter the number of vertices of a graph
5
Enter the adjacency matrix
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0

The topological sequence is 2 1 3 4 5
```

Output:

Viva Voce:

Evaluation:

Experiment No. 3b **Date: __ /__ / _____**

Warshall's Algorithm
(Graph Problem)

Aim:

To compute the transitive closure of a directed graph.

Concepts Required:

The transitive closure of a directed graph with n vertices can be define as the n-by-n Boolean matrix T={$t_{i,j}$ },in which the element in the element in the I th row and jth column is 1 if there exists a non trivial directed path from the I th vertex to j th vertex Otherwise, $t_{i,j}$ is 0.

Algorithm:

ALGORITHM Warshall's (A[1..n,1..n])
//Implements Warshall's Algorithm to compute the transitive closure
//Input: The adjacency matrix A of a directed graph with n vertices
//Output: The transitive closure of the digraph
$R^{(0)}$ <-A
For k<- 1 to n do
For i<-1 to n do
For j<- 1 to n do
$R^{(k)}$ [i,j]<- $R^{(k-1)}$ [i,j] or $R^{(k-1)}$ [i,k] and $R^{(k-1)}$ [k,j]
Return $R^{(n)}$

Code:

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

void warshalls(int a[20][20], int n)
{
  int i,j,k;
  for (k=0;k<n;k++)
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
            a[i][j] = a[i][j] || a[i][k] && a[k][j];
   printf("\n The transitive closure is:\n");

   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
         printf("%d\t",a[i][j]);
         printf("\n");
   }
}
```

```
void main()
{
    int a[20][20],n,i,j;
    clrscr();
    printf("=======================================\n");
    printf("  Compute transitive closure using Warshalls Algorithm \n");
    printf("=======================================\n");
    I:printf("\n Enter the number of vertices of the graph:\n");
    scanf("%d",&n);

    if(n>0 && n <90)
    {
        printf("\n Enter the adjacency matrix:\n");
        for(i=0;i<n;i++)
        {
           for(j=0;j<n;j++)
          {
              scanf("%d",&a[i][j]);
          }
        }
        warshalls(a,n);
     }
     else
     {
        printf("Enter the valid number of vertices\n");
        goto I;
     }
     getch();
  }
```

Output:

```
==============================================
Compute transitive closure using Warshalls Algorithm
==============================================

Enter the number of vertices of the graph

4

Enter the adjacency matrix

0 1 0 0

0 0 0 1

0 0 0 0

1 0 1 0
```

The transitive closure is

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

Output:

Viva Voce:

Evaluation:

Experiment No. 4 **Date: __ /__ / _____**

Knapsack problem:
(Combinatory Problem)

Aim:

To Implement 0/1 Knapsack problem using dynamic programming.

Problem Statement:

We are given n objects and a knapsack. Each object has a weight, w and the maximum weight or capacity of the knapsack is M. Each object is associated with a value or profit is carried when it is places in the knapsack.

Objective:

We must fill the knapsack which maximizes the profit control and the output is subset of object number.

Algorithm:(Pseudo Code)

Algorithm Knapsack(i,j)

If j<weights[i]

Value= Knapsack(i-1,j)

Else

Value=max(knapsack(i-1,j),vaues[i]+ knapsack(i-1,j-weights[i]))

V[I,j]=value

Return v[i,j]

Code:

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

int w[10],v[10],a[10][10],t[10];
int c,i,n,j;

void knapsack(int c,int n)
        {
           int i,j;
           for(i=0;i<=n;i++)
           for(j=0;j<=c;j++)
           {
               if(i==0 || j==0)
                       a[i][j]=0;
               else if(w[i]>j)
                       a[i][j]=a[i-1][j];
               else
                       a[i][j]=max(a[i-1][j],(a[i-1][j-w[i]]+v[i]));
            }
```

```
            i=n;j=c;
            while(i>0 && j>0)
            {
                if(a[i][j] != a[i-1][j])
                {
                        t[i]=1;
                        j=j-w[i];
                }
                i--;
            }
        }


void read_capacity()
        {
           printf("\n\n Enter the capacity of the bag:");
               scanf("%d",&c);
           if(c>0)
               read_numofitems();
           else
               {
                       printf("\n\n Invalid capacity,please try again...");
                       read_capacity();
               }
        }


int read_numofitems()
        {
               I:printf("\n\n Enter the number of items:\n");
                       scanf("%d",&n);
               if(n>0)
                       read_weights();
               else
                       {
                               printf("\n\n Invalid items,please try agian...");
                               goto I;
                       }
               return 1;
        }


int read_weights()
        {
         J: printf("\n\n Enter the corresponding weights:\n");
               for(i=1;i<=n;i++)
                       scanf("%d",&w[i]);
               for(i=1;i<=n;i++)
               {
               if(w[i]>0 && w[i]<=999)
               {
                       read_values();
```

```
            }
            else
            {
                    printf("\n\n Invalid weights,please try again..");
                    goto J;
            }
        }
        return 0;
    }


int read_values()
{
     K: printf("\n\n Enter the corresponding values:\n");
        for(i=1;i<=n;i++)
              scanf("%d",&v[i]);
        for(i=1;i<=n;i++)
        if(v[i]>0 && v[i]<=999)
        {
            knapsack(c,n);
        }
        else
        {
            printf("\n\n Invalid values,please try again...");
            goto K;
        }
        return 0;
    }


    void main()
    {
            clrscr();
            printf("\t\t    ***THIS    PROGRAM    DEMONSTRATES    KNAPSACK
            PROBLEM***");
            read_capacity();
            printf("\n\n Resultant matrix is:\n");
            for(i=0;i<=n;i++)
            {
                    for(j=0;j<=c;j++)
                            printf("\t%d",a[i][j]);
                            printf("\n");
            }
            printf("\n\n The optimal solution is:%d",a[n][c]);
            printf("\n\n Items selected are:\n");
            for(i=1;i<=n;i++)
                    if(t[i]==1)
                       printf("\t%d",i);
            printf("\n\n Thankyou for using program,have a nice day...");
            getch();
    }
```

Output:

***** THIS PROGRAM DEMONSTRATES KNAPSACK PROBLEM*****

enter the capacity of the knapsack
5

enter no. of items
4
enter the wieght of each item
2
1
3
2
enter the corresponding value of each item
12
10
20
15
Resultant matrix is
0 0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37

The optimal solution is:37
Items selected are:
1 2 4

Thank you

Output:

Viva Voce:

Evaluation:

Experiment No. 5 **Date: __ /__ / _____**

Dijkstra's algorithm
(Graph Problem)

Aim:

To find shortest paths from a given starting vertex to other vertices in a weighted connected graph using Dijkstra's algorithm.

Algorithm: (Pseudo Code)

ALGORITHM Dijkstra(G,s)

// Dijkstra's algorithm for single-source shortest paths

// Input: A weighted connected graph G={V,E} with nonnegative weights & its vertex s

// Output:The length dv of a shortest path from s to v & its penultimate vertex pv for a vertex v in V

```
Intialiize(Q)   //initialize vertex priority queue to empty
for every vertex v in V do
        dv ← ∞;pv ← null
        Insert (Q, v, dv)       //initialize vertex priority in the priority queue
ds ← 0 ;     Decrease(q,s,ds)     //update priority of s with ds
Vt  ← Ø
for i ← 0 to |V| - 1 do
    u* ← Deletemin(Q)            //delete the minimum priority element
    Vt ← Vt U {u*}
       for every vertex u in V – Vt that is adjacent to u* do
           if du* + w(u*,u)<du
              du ← du* + w(u*,u);      pu ← u*
              Decrease (Q,u,du)
```

Code:

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

int d[10],p[10],visited[10];
void dijktras(int a[10][10],int s,int n)
{
	int u,v,i,j,min;
	for(v=0;v<n;v++)
	{
		d[v]=99;
		p[v]=-1;
	}
	d[s]=0;
	for(i=0;i<n;i++)
	{
		//select u which is the minimum among the remaining
		min=99;
		for(j=0;j<n;j++)
		{
			if(d[j]<min && visited[j]==0)
			{
				min=d[j];
				u=j;
			}
		}
		visited[u]=1;    // Vt=Vt+u
		//select v in the remaining vertices(V-Vt) that is adj to u
		for(v=0;v<n;v++)
		{
			if((d[u]+a[u][v]<d[v]) && (u!=v) && visited[v]==0)
			{
				d[v]=d[u]+a[u][v];
				p[v]=u;
			}
		}
	}

}


void path(int v,int s)
{
	if(p[v]!=-1)
		path(p[v],s);
	if(v!=s)
	     printf("->%d",v);
}
```

```
void display(int s,int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		if(i!=s)
		{
			printf("%d",s);
			path(i,s);
		}
		if(i!=s)
		   printf("=%d\n",d[i]);
	}
}

void main()
{
	//declaration of variables
	int a[10][10],i,j,n,s;
	clrscr();
	printf("Program for Single Source Shortest Path using Dijktras
		Algorithm\n\n");

	// Read the weighted graph as cost matrix
	printf("Enter the number of vertices (size of the graph)\n");
	scanf("%d",&n);

	printf("Enter the %d * %d cost matrix\n",n,n);
	for(i=0;i<n;i++)
	  for(j=0;j<n;j++)
	     scanf("%d",&a[i][j]);

	printf("Enter the source vertex\n");
	scanf("%d",&s);

	//Calling Dijktras function by passing a graph and source
	dijktras(a,s,n);

	//Display the result
	printf("\t\tOUTPUT\t\t\n");
	printf("The Shortest path between %d to remaining vertices are:\n\n",s);
	display(s,n);
	printf("\t\tThank You\t\t\n");
	getch();
}
```

OUTPUT:

Program for Single Source Shortest Path using Dijktras Algorithm

```
 Enter the number of vertices of the graph:
5

 Enter the weights of the edges of the graph:
0  3  99  7  99
3  0  4   2  99
99 4  0   5   6
7  2  5   0   4
99 99 6   4   0

 Enter the source vertex to find the shortest path:
1
The shortest path from vertex 1 to all other vertices is:
1->2=3
1->2->3=7
1->2->4=5
1->2->4->5=9
```

Output:

Viva Voce:

Evaluation:

Experiment No. 6 **Date: __ /__ / _____**

Krushkal's Algorithm
(Graph Problem)

Aim:

To Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm

Algorithm:(Pseudo Code)

ALGORITHM Kruskal(G)

// Kruscal's algorithm to find a minimum spanning tree

// Input: A weighted connected graph G

//Output:E_T, the set of edges composing the minimum spanning tree of G

Sort E in nondecresing order of the edges weight

$E_T \leftarrow \emptyset$; ecounter ← 0

k ←0

while ecounter < |V|-1 do

k ← k+1

if E_T U $\{e_{ik}\}$ is acyclic

$E_T \leftarrow E_T$ U $\{e_{ik}\}$; ecounter ← ecounter+1

return E_T

Code:

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


int parent[20];


int find(int m)
{
  int p = m;
  while(parent[p]!=0)
  p = parent[p];
  return(p);
 }
```

```
void union_ij(int i,int j)
{
  if(i<j)
    parent[i] = j;
  else
    parent[j]=i;

}

void kruskal(int a[10][10], int n)
{
   int u,v,min,k=0,i,j,sum=0;
   while(k<n-1)
   {
      min =999;
      for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
            if(a[i][j]<min && i!=j)
            {
                  min = a[i][j];
                  u=i;
                  v=j;
            }
       i = find(u);
       j=find(v);
       if(i!=j)
      {
          union_ij(i,j);
          printf("(%d,%d)=%d",u,v,a[u][v]);
          sum = sum + a[u][v];
          k++;
       }
      a[u][v] = a[v][u]=999;
    }
    printf("\n The minimum cost spanning tree is = %d",sum);
}

void main()
{
  int a[10][10],n,i,j;
  clrscr();
  printf("=========================================\n");
  printf("  Find minimum cost spanning tree using Kruskal Algorithm \n");
  printf("=========================================\n");
  I:printf("\n Enter the number of vertices of the graph:\n");
  scanf("%d",&n);
  if(n>0)
 {
    printf("\n Enter the cost adjacency matrix\n");
    printf(" (0 for loops and 999 if there is no direct edge):\n");
    for(i=1;i<=n;i++)
```

```
        for(j=1;j<=n;j++)
             scanf("%d",&a[i][j]);
    kruskal(a,n);
    getch();
   }
   else
  {
    printf("Enter valid number of vertices");
    goto I;
   }
 }
```

Output:

```
==============================================
 Find minimum cost spanning tree using Kruskal Algorithm
==============================================

Enter the number of vertices  of the graph
4

 Enter the cost adjacency matrix
(0 for loops and 999 if there is no direct edge)
0 20 2 99
20 0 15 5
2 15 0 25
99 5 25 99

(1,3)=2
(2,4)=5
(2,3)=15
The minimum cost spanning tree is =22
```

Output:

Viva Voce

Evaluation:

Experiment No. 7a **Date: __ /__ / _____**

Breadth-First Search (BFS)

(Graph Problem)

Aim:

To find all the nodes reachable from a given starting node in a digraph using BFS method

Algorithm: (Pseudo Code)

Algorithm bfs (v)
//v: is the starting vertex.
//Visited [1...n] : to remember the visited information.
//Qinsert () and Qdelete (): sub algorithms to add and remove vertices from the queue //respectively.
//T: Contains the edge details of the output.

```
{
        Visited [v] = 1;      //mark the starting vertex as visited
        T=0;                        //initialize spanning tree.
        InitializeQueue();       //set front and rear pointers
        while (true) do
        {
                for all vertices w adjacent to v do
                        If visited [w] = 0
                        {
                                Qinsert (w);
                                Visited [w] = 1;
                                T=Union (T, <v,w>) ; //add edge to the spanning
tree.
                        }
                        if Qempty() return T;
                        v = Qdelete ();
        }
}
```

Code:

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

int visited[10];

void bfs(int n,int a[10][10],int source)
{
        int i,q[10],u,front=0,rear=-1;
        q[++rear] = source;
        visited[source]=1;
        printf("\n The reachable vertices are:\n");
        while(front<=rear)
```

```
        {
            u=q[front++];
            for(i=0;i<n;i++)
            if(a[u][i]==1 && visited[i]==0)
           {
              q[++rear]=i;
              visited[i]=1;
              printf("%d\n",i);
           }
        }
}

void main()
{
        int a[10][10],n,i,j,source;
        clrscr();
        printf("=========================================\n");
        printf("  Find whether the nodes are rechable using BFS method \n");
        printf("=========================================\n");
     I: printf("\n Enter the number of vertices of the digraph:\n");
        scanf("%d",&n);
        if(n>0 && n<90)
        {
           printf("\n Enter the adjacency matrix of the graph:\n");
           for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                    scanf("%d",&a[i][j]);
         }
        else
        {
           printf("Enter the valid number of vertices\n");
           goto I;
        }
        printf("Enter the source vertex to find other nodes rechable or not:\n");
        scanf("%d",&source);
        bfs(n,a,source);
        for(i=0;i<n;i++)
            if(visited[i]==0)
              printf("\n The vertex that is not rechable %d\n",i);
        getch();
   }
```

output:

```
 ==========================================
      Find wether the nodes are rechable using BFS method
 ==========================================
 enter the number of vertices of the digraph:
5
```

```
enter the adjacency matrix of the graph
0 1 1 1 0
0 0 0 0 1
0 0 0  0 0
0 0 0 0 0
0 0 1 1 0

 Enter the source vertex to find other nodes rechable or not
0

the reachable vertices are
1
2
3
4
```

Output:

Viva Voce:

Evaluation:

Experiment No. 7b **Date: __ /__ / _____**

Depth First Search (DFS)
(Graph Problem)

Aim:

To Check whether a given graph is connected or not using DFS method

Algorithm:

Algorithm dfs (v)

//v: the starting vertex.

//Visited [1...n]: is array to remember the visited information.

{

Visited [v] =1; //mark the starting vertex as visited

for each vertex w adjacent to v **do**

if Visited[w] = 0 //vertex not yet finished

dfs (w); //continue to explore

}

Code:

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

int a[10][10],n,s[10],top=-1,q[10];

void dfs(int source)
{
        int v;
        s[++top] = 1;
       q[source]=1;
        for(v=0;v<n;v++)
          if(a[source][v]==1 && q[v]==0)
          {
              printf("%d->%d\n",source,v);
              dfs(v);
          }
}

void main()
{
       int i,j,source,m;
       clrscr();
       printf("=====================================\n");
```

```
        printf("Find whether the graph is connected or not using DFS method");
        printf("=====================================\n");
        I:printf("\n Enter the number of vertices of the graph:\n");
        scanf("%d",&n);
        if(n>0 && n<90)
        {
            printf("\n Enter the adjacency matrix of the graph:\n");
            for(i=0;i<n;i++)
               for(j=0;j<n;j++)
                       scanf("%d",&a[i][j]);
        }
        else
       {
            printf("Enter valid number of vertices\n");
           goto I;
       }
        printf("\n Enter the source vertex to find the graph connectivity:\n");
        scanf("%d",&source);
        m=1;

        dfs(source);
        for(i=0;i<n;i++)
              if(q[i]==0)
                 m=0;
        if(m==1)
             printf("\n The graph is connected:\n");
        else
             printf("\n The graph is not connected: \n");
       getch();
}
```

output :

```
 ==========================================
       Find wether the graph is connected or not using DFS method
 ==========================================
enter the no. of vertices in the graph
5
enter the adjacency matrix
0 1 1 1 0
0 0 0 0 1
0 0 0 0 0
0 0 0 0 0
0 0 1 1 0

enter the source vertex
0
```

DFS order is
0->1
1->4
4->2
4->3
graph is connected

Output:

Viva Voce:

Evaluation:

Experiment No. 8 **Date: __ /__ / _____**

Sum of subset problem
(Combinatory Problem)

Aim:

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. For example, if S= {1,2, 5, 6, 8} and d = 9 there are two solutions{1,2,6}and{1,8}.

Algorithm:

Step 1: Start

Step 2: Find the subset for given set

Step 3: Add the elements of the subset

Step 4: If the sum of the subset is equal to the given positive integer d, solution is found, otherwise solution is not found.

Step 5: Stop

Code:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

 void subset(int num, int n, int x[])
 {
   int i;
   for(i = 1;i<=n;i++)
   x[i]=0;
   for(i=n;num!=0;i--)
   {
     x[i] = num%2;
     num = num/2;
   }
 }


void main()
{
   int a[50],x[50],n,j,i,d,sum,present = 0;
   clrscr();
   printf("=========================================\n");
   printf("Find subset of the given set whose sum is equal to a given integer \n");
   printf("=========================================\n");
   I:printf("\n Enter the number of elements of the set\n");
   scanf("%d",&n);
```

```
if(n>0 && n<90)
{
        printf("\n Enter the elements of the set \n");
        for(i=1;i<=n;i++)
          scanf("%d",&a[i]);
}
else
{
      printf("Enter the valid number of elements\n");
      goto I;
}
J:printf("\n Enter the positive integer sum\n");
scanf("%d",&d);
if(d>0)
{
        for(i=1;i<=pow(2,n)-1; i++)
       {
            subset(i,n,x);
            sum=0;
            for(j=1;j<=n;j++)
                if(x[j]==1)
                    sum = sum +a[j];
            if(d==sum)
            {
                    printf("\n subset={");
                    present = 1;
                    for(j=1;j<=n;j++)
                         if(x[j]==1)
                              printf("%d ",a[j]);
                    printf("}=%d",d);
            }
         }
  }
 else
 {
      printf("Enter the valid positive integer\n");
      goto J;
 }
 if(present==0)
         printf("solution does not exist");
 getch();
}
```

output:

```
==========================================
Find the subset of the given set whose sum is equal to a given integer
==========================================

enter number of elements of the set
5
enter the elements of the set
```

```
1
2
5
6
8

Enter the positive integer sum
9

subset={1 8 }=9
subset={1 2 6 }=9
```

Output:

Viva Voce:

Evaluation:

Experiment No. 9 **Date: __ /__ / _____**

Travelling Salesperson Problem (Graph Problem)

Aim:

Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

ALGORITHM:

Step.1. Start
Step 2. Find the optimal solution for the travelling salesperson problem
Step 3. Find the solution using approximation algorithm
Step 4. Find the error in the approximation
Step 5. Stop

Code:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define MAX 100
#define INFINITY 999

int sum,visi[10],n,a[MAX][MAX],current,tour[100];
float error;

int tsp_dp (int c[][MAX], int tour[], int start, int n)
{
int i, j, k; /* Loop counters. */
int temp[MAX]; /* Temporary during calculations. */
int mintour[MAX]; /* Minimal tour array. */
int mincost; /* Minimal cost. */
int ccost; /* Current cost. */

/* End of recursion condition. */
if (start == n - 2)
return c[tour[n-2]][tour[n-1]] + c[tour[n-1]][0];

/* Compute the tour starting from the current city. */
mincost = INFINITY;
for (i = start+1; i<n; i++)
{
        for (j=0; j<n; j++)
        temp[j] = tour[j];
```

```
        /* Adjust positions. */
        temp[start+1] = tour[i];
        temp[i] = tour[start+1];

        /* Found a better cycle? (Recurrence derivable.) */
    if (c[tour[start]][tour[i]] +(ccost = tsp_dp (c, temp, start+1, n)) < mincost)
    {
        mincost = c[tour[start]][tour[i]] + ccost;
        for (k=0; k<n; k++)
                mintour[k] = temp[k];
    }
 }

        /* Set the minimum-tour array. */
        for (i=0; i<n; i++)
                tour[i] = mintour[i];

        return mincost;
}

void tsp(int source)
{
 int min,u,i,j,p,q;
      min = 999;
     u = source;
     visi[u] = 1;
     for(i=0;i<n;i++)
    {
          if(a[u][i]< min && visi[i]==0 && u!=i)
          {
                min = a[u][i];
                p = u;
                q = i;
           }

    }
    sum = sum + a[p][q];
    printf("(%d,%d) = %d",p,q,a[p][q]);

if((n-1)==q)
{
  sum=sum+a[q][current];
  printf("(%d,%d) = %d",q,current,a[q][current]);
  printf("\nCost using approximation algorithm = %d",sum);

}
else
    tsp(q);
}
```

```
void main()
{
  int i,j;
  int source,cost;
  clrscr();
  printf("=========================================\n");
  printf("\t\t Travelling Salesman Problem\n");
  printf("=========================================\n");
  printf("Enter the number of vertices in the graph\n");
  scanf("%d",&n);

  printf("Enter the cost adjacency matrix\n");
  for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);

  printf("Enter the source vertex\n");
  scanf("%d",&source);
  current=source;

  for(i=0;i<n;i++)
  tour[i]=i;

   // TSP using Dynamic programmimg (Optimal Solution)
   cost = tsp_dp (a, tour, source, n);

  // TSP using Nearest Neighbor algorithm (Approximation)
  tsp(source);

  printf ("\n Minimum cost: %d.\nTour using Dynamic Programming: ", cost);
  for (i=0; i<n; i++)
  printf ("%d ", tour[i]+1);
  printf ("1\n");

  printf("The sum %d and cost is %d \n",sum,cost);
  error = (float)sum/cost;
  // Error Factor
  printf("The error factor is %f",error);
  getch();
}
```

OUTPUT:

```
=========================================
            Travelling Salesman Problem
=========================================

Enter the number of vertices in the graph

4
```

Enter the cost adjacency matrix
0 1 3 6
1 0 2 3
3 2 0 1
6 3 1 0

Enter the source vertex

0
(0,1) =1(1,2) = 2(2,3) = 1(3,0) = 6
Cost using approximation algorithm = 10
Minimum cost: 8.
Tour using Dynamic Programming: 1 2 4 3 1
The sum is 10 and cost is 8
The error factor is 1.250000

Output:

Viva Voce:

Evaluation:

Experiment No. 10 **Date: __ /__ / _____**

Prim's Algorithm
(Graph Problem)

Aim:

To construct a minimum spanning tree of a weighted connected graph.

Algorithm:

ALGORITHM prim(G)

// Prim's Algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph G=(V,E)

//Output: E_T, the set of edges composing a minimum spanning tree of G

V_T <-- {V0}

E_T <-- Ø

for i <-- 1 to |V| − 1 do

find a minimum-weight edge e* = (v*, u*) among all the edges (v,u) such that v is in VT and u is in V − V_T

V_T <-- V_T U {u*}

E_T <-- E_T U {e*}

return E_T

Code:

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

void prims(int n,int a[10][10],int source)
{
 int s[10],i,j,min=999,sum=0,u,v,k,t=0;
 for(i=0;i<n;i++)
       s[i]=0;
 s[source]=1;
 k=1;
 while(k<=n-1)
 {
       min=999;
       for(i=0;i<n;i++)
         for(j=0;j<n;j++)
               if(s[i]==1 && s[j]==0)
                 if(i!=j && min>a[i][j])
                 {
                       min=a[i][j];
                       u=i;
                       v=j;
                 }
```

```
        printf("\n(%d,%d)=%d",u,v,min);
        sum=sum+min;
        s[u]=1;
        s[v]=1;
        k++;
 }
 for(i=0;i<n;i++)
 {
       if(s[i]!=1)
             t=1;
 }
 if(t==0)
       printf("\ncost of min spanning tree is %d",sum);
 else
       printf("\nminimum spanning tree not possible");
}
void main()
{
  int a[10][10],n,i,j,source;
  clrscr();


printf("=============================================\n");
printf("    Find minimum cost spanning tree using Prim's Algorithm \n");
printf("=============================================\n");
  printf("\n Enter the number of vertices");
  scanf("%d",&n);
  printf("Enter the cost matrix 0 for self loop and 99 for no edge \n");
  printf("Enter the n*n matrx\n");

  for(i=0;i<n;i++)
       for(j=0;j<n;j++)
             scanf("%d",&a[i][j]);

  printf("Enter the source node\n");
  scanf("%d",&source);
  prims(n,a,source);
  getch();
}
```

output:

```
=============================================
 Find minimum cost spanning tree using Prim's Algorithm
=============================================
Enter the number of nodes
6
Enter the cost matrix 0 for self loop and 99 for no edge
```

```
Enter the n*n matrx
0 3 99 99 6 5
3 0 1 99 99 4
99 1 0 6 99 4
99 99 6 0 8 5
6 99 99 8 0 2
5 4 4 5 2 0

Enter the source node
1

1->2 =3

2->3 =4

2->6 =8

6->5 =10

6->4 =15
   cost of min spanning tree is=15
```

Output:

Viva Voce:

Evaluation:

Experiment No. 11 **Date: __ /__ / _____**

Floyd's Algorithm
(Graph Problem)

Aim:

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved

Algorithm:

ALGORITHM Floyd (W [1…n, 1…n])

// Implements Floyd's algorithm for all-pair shortest-paths problem

//Input: The weight matrix W of a graph with no negative-length cycle

//Output: The distance matrix of the shortest paths' lengths D ← W is not

//necessary if W can be overwritten

for k ← 1 to n do

for i ← 1 to n do

for j ← 1 to n do

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

return D

Code:

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


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

void floyd(int w[10][10],int n)
{
 int i,j,k;
 #pragma omp parallel for private(i, j, k) shared(w)
 for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
```

```
                w[i][j]=min(w[i][j],w[i][k]+w[k][j]);
}
void main()
{
  int a[10][10],n,i,j;
  double startTime,endTime;

  printf("\n Enter the no. of vertices:");
  scanf("%d",&n);
  printf("\n Enter the cost matrix, 0-self loop and 999-no edge\n");
  for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
                scanf("%d",&a[i][j]);

  startTime=omp_get_wtime();
  floyd(a,n);
  endTime = omp_get_wtime();

  printf("\n Shortest path matrix:\n");
  for(i=1;i<=n;i++)
  {
     for(j=1;j<=n;j++)
                printf("%d\t",a[i][j]);
                printf("\n");
  }
       printf("Time taken is %10.9f\n",(double)(endTime-startTime));
}
```

Output:

Enter the no. of vertices:

4

Enter the cost matrix, 0-self loop and 999-no edge

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

Time taken is 0.001107683

Output:

Viva Voce:

Evaluation:

Experiment No. 12 **Date: __ /__ / _____**

N-queen's problem

Aim:

To Implement N Queen's problem using Back Tracking

Problem Statement:

The problem is to place n queens on an n by n matrix so that no two queens attack each other by being in same row or in the same column or on the same diagonal.

Algorithm:

ALGORITHM NQueen(K,n)

// Using backtracking

// possible placements of 'n' queen on n*n chessboard

```
For i:= 1 to n do

    If Place(k,i) then
        x[k]=I;
        if k=n then write x[1:n]
        else NQueen(k+1,n)
    end if
end for
```

ALGORITHM Place(k,i)

// return true if a queen can be placed in kth row and ith column

// otherwise return false

```
For j=I to k-1 do
    If (x[j]=I ) or (abs(x[j]=i) = abs(j-k) then //two in the same
        return false                    //column or in the same diagonal
Return true
```

Code:

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

int b[20],count;

int place(int row,int col)
{
  int i;
  for(i=1;i<row;i++)
       if(b[i]==col || abs(b[i]-col)==abs(row-i))
             return 0;
       return 1;
}


void dis(int n)
{
  int i,j;
  printf("\n Solution number:%d\n",++count);
    for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
     {
         if(b[i]==j)
             printf("\tq");
         else
             printf("\t*");
     }
     printf("\n");
   }
  printf("\n\n");
}

void queen(int row, int n)
{
    int col;
    for(col=1;col<=n;col++)
   {
       if(place(row,col))
      {
         b[row]=col;
         if(row==n)
            dis(n);
         else
            queen(row+1,n);
      }
    }
}
```

```
void main()
{
    int n;
    clrscr();
    printf("==========================================\n");
    printf("\t\t\t n - queens problem\n");
    printf("==========================================\n");
   I:printf("\n Enter the number of queens to be placed on a n*n chessboard\n");
   scanf("%d",&n);

    if(n>0 && n<90)
   {
        queen(1,n);
         printf("\n Total no of solution:%d", count);
    }
    else
    {
         printf("Enter the valid number of queens\n");
         goto I;
    }
     getch();
}
```

Output:

```
==========================================
     N Queens
==========================================
Enter the number of queens to be placed on a n*n chessboard
4


 solution:1

 * q * *


 * * * q


 q * * *


 * * q *


 solution:2

 * * q *
```

q * * *

* * * q

* q * *

Total number of solutions are 2

Output:

Viva Voce:

Evaluation:

References

1. " Introduction to Analysis and Design of Algorithms" by Anany levitin, second edition.
2. "Fundamentals of Computer Algorithm" by Horowitz,Sahni,and Rajasekaran.
3. "Algrithms" by Berman,and Paul , India Edition.

VIVA questions

1. What is Algorithm?
2. Name the design techniques.
3. Which is efficient Sorting technique?
4. Which sorting technique has the lowest worst case efficiency?
5. Which sorting technique is space efficient?
6. Which sorting technique is Time efficient?
7. Define order of growth.
8. How binary search is advantageous over linear search?
9. What is CLK_TCK?
10. What is divide &conquer technique?
11. Give few problems which can be solved using divide &conquer technique.
12. What is dynamic programming?
13. Give few problems which can be solved using dynamic programming.
14. What is Back Tracking?
15. Define N-Queens problem.
16. What is Brute force Methodology?
17. What is Greedy Technique?
18. Give few problems which can be solved using Greedy Technique.
19. Which is the efficient method for finding MST?
20. What is MST?
21. What is DFS& BFS?
22. What is a heap?
23. What is Clock ( ) function?
24. Which is the header to include Clock function?
25. What are P NP problems?
26. Which are the String Matching algorithms?
27. What is Transitive closure?
28. Define Knapsack problem.
29. Define topological sorting?
30. Which algorithm used for checking graph is connected or not?