JEPPIAAR INSTITUTE OF TECHNOLOGY

Self Belief | Self Discipline | Self Respect

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

NAME :

REG NO. :

YEAR :

SEMESTER :

BRANCH :

COURSE CODE :

COURSE NAME :

JEPPIAAR INSTITUTE OF TECHNOLOGY

SELF BELIEF | SELF DISCIPLINE | SELF RESPECT

KUNNAM, SUNGUVARCHATRAM, SRIPERUMBUDUR, CHENNAI - 631 604

BONAFIDE CERTIFICATE

This is a certified Bonafide Record Work of Mr./Ms._____

Register No._____submitted for the Anna University Practical Examination held in **CS3401- ALGORITHMS LABORATORY** during the year 2023-2024

Signature of the Lab In-charge

Head of the Department

Internal Examiner

External Examiner

Date _____

INSTITUTE VISION

Jeppiaar Institute of Technology aspires to provide technical education in futuristic technologies with the perspective of innovative, industrial and social application for the betterment of humanity.

INSTITUTE MISSION

IM1: To produce competent and disciplined high-quality professionals with the practical skills necessary to excel as innovative professionals and entrepreneurs for the benefit of the society.

IM2: To improve the quality of education through excellence in teaching and learning, research, leadership and by promoting the principles of scientific analysis, and creative thinking.

IM3: To provide excellent infrastructure, serene and stimulating environment that is most conducive to learning.

IM4: To strive for productive partnership between the Industry and the Institute for research and development in the emerging fields and creating opportunities for employability.

IM5: To serve the global community by instilling ethics, values and life skills among the students needed to enrich their lives.

DEPARTMENT VISION

To impart futuristic technological education, innovation and collaborative research in the field of Computer Science and Engineering and to develop Quality Professionals for the improvement of the society and industry.

DEPARTMENT MISSION

DM1: Develop the students as professionally competent and disciplined engineers for the benefit of the development of the country.

DM2: Produce excellent infrastructure to adopt latest technologies, industry-institute interaction and encouraging research activities.

DM3: Provide multidisciplinary technical skills to pursue research activities, higher studies, entrepreneurship and perpetual learning.

DM4: Enrich students with professional integrity and ethical standards to handle social challenges successfully in their life.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates can

- Graduates can able to apply their technical competence in computer science to solve real world problems, with technical and people leadership.
- Graduates can able to conduct cutting edge research and develop solutions on problems of social relevance.
- Graduates can able towWork in a business environment, exhibiting team skills, work ethics, adaptability and lifelong learning.

PROGRAM SPECIFIC OUTCOMES

The Students will be able to

- Exhibit design and programming skills to build and automate business solutions using cutting edge technologies.
- Strong theoretical foundation leading to excellence and excitement towards research, to provide elegant solutions to complex problems.
- Ability to work effectively with various engineering fields as a team to design, build and develop system applications.

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. Engineering knowledge: (K3) Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: (K4) Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: (K4) Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: (K5) Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: (K3, K5, K6) Create, select, and apply appropriate techniques, resources, and

modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: (A3) Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: (A2) Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: (A3) Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: (A3) Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: (A3) Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: (A3) Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: (A2) Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ANNA UNIVERSITY SYLLABUS

CS3401 ALGORITHMS L T P C 3 0 2 4

COURSE OBJECTIVES:

- To understand and apply the algorithm analysis techniques on searching and sorting algorithms
- To critically analyze the efficiency of graph algorithms
- To understand different algorithm design techniques
- To solve programming problems using state space tree
- To understand the concepts behind NP Completeness, Approximation algorithms and randomized algorithms.

PRACTICAL EXERCISES: 30 PERIODS

Searching and Sorting Algorithms

1. Implement Linear Search. Determine the time required to search for an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.
2. Implement recursive Binary Search. Determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.
3. Given a text txt [0...n-1] and a pattern pat [0...m-1], write a function search (char pat [ ], char txt [ ]) that prints all occurrences of pat [ ] in txt [ ]. You may assume that n > m.
4. Sort a given set of elements using the Insertion sort and Heap sort methods 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.

Graph Algorithms

5. Develop a program to implement graph traversal using Breadth First Search
6. Develop a program to implement graph traversal using Depth First Search
7. From a given vertex in a weighted connected graph, develop a program to find the shortest paths to other vertices using Dijkstra's algorithm.
8. Find the minimum cost spanning tree of a given undirected graph using Prim's algorithm.
9. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.
10. Compute the transitive closure of a given directed graph using Warshall's algorithm.

Algorithm Design Techniques

11. Develop a program to find out the maximum and minimum numbers in a given list of n numbers using the divide and conquer technique.
12. Implement Merge sort and Quick sort methods to sort an array of elements and determine the time required to sort. 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.

State Space Search Algorithms

13. Implement N Queens problem using Backtracking.

Approximation Algorithms Randomized Algorithms

14. Implement any scheme to find the optimal solution for the Travelling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.
15. Implement randomized algorithms for finding the kth smallest number.

The programs can be implemented in C/C++/JAVA/ Python.

TOTAL: 30 PERIODS

COURSE OUTCOMES: At the end of this course, the students will be able to:

CO1: Analyze the efficiency of algorithms using various frameworks
CO2: Apply graph algorithms to solve problems and analyze their efficiency.
CO3: Make use of algorithm design techniques like divide and conquer, dynamic programming and greedy techniques to solve problems
CO4: Use the state space tree method for solving problems.
CO5: Solve problems using approximation algorithms and randomized algorithms

CS3401 ALGORITHMS LABORATORY

(REGULATIONS-2021)

List of Experiments

S.No	Name of the Experiment
1	Linear Search Implementation
2	Recursive Binary Search Implementation
3	Pattern Search Implementation
4	Insertion sort and Heap sort
5	Breadth First Search Graph Traversal Implementation
6	Depth First Search Graph Traversal Implementation
7	Dijkstra's algorithm Single Source Shortest Path
8	Prim's algorithm Minimum Spanning Tree
9	Floyd's algorithm All-Pairs- Shortest-Paths problem
10	Warshall's algorithm
11	Maximum and minimum numbers in a given list of n numbers using the divide and conquer technique
12	Implement Merge sort and Quick sort
13	Implement N Queens problem using Backtracking
14	Travelling Salesperson problem using any approximation algorithm
15	Implement randomized algorithms for finding the kth smallest number

CS3401 ALGORITHMS LABORATORY

INDEX

S.No.	Date	Name of the Experiment	Pg.No.	Marks	Signature

Signature of the Staff

COURSE OUTCOMES

Course Outcome No.	Course Outcome	Highest Cognitive Level
CO213.1	CO1: Analyze the efficiency of algorithms using various frameworks	K3
CO213.2	CO2: Apply graph algorithms to solve problems and analyze their efficiency.	K3
CO213.3	CO3: Make use of algorithm design techniques like divide and conquer, dynamic programming and greedy techniques to solve problems	K3
CO213.4	CO4: Use the state space tree method for solving problems.	K3
CO213.5	CO5: Solve problems using approximation algorithms and randomized algorithms	K3

CO's- PO's & PSO's MAPPING

CO's	PO's												PSO's		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	2	1	3	2	-	-	-	-	2	1	2	3	2	1	1
2	2	1	1	1	1	-	-	-	1	3	3	3	2	3	3
3	1	3	3	3	1	-	-	-	1	2	1	2	2	1	2
4	1	2	2	3	-	-	-	-	2	3	3	1	3	1	3
5	1	2	3	2	3	-	-	-	3	1	3	3	1	3	3
AVg.	1	2	2	2	2	-	-	-	2	2	2	2	2	2	2

1 - low, 2 - medium, 3 - high, '-"- no correlation

Ex. No. 1 LINEAR SEARCH IMPLEMENTATION

DATE :

AIM: To implement Linear Search. Determine the time required to search for an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

ALGORITHM:

Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value to search

Step 1: set pos = -1
Step 2: set i = 1
Step 3: repeat step 4 while i <= n
Step 4: if a[i] == val
set pos = i
print pos
go to step 6
[end of if]
set ii = i + 1
[end of loop]
Step 5: if pos = -1
print "value is not present in the array "
[end of if]
Step 6: exit

PROGRAM : LINEAR SEARCH

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<timer.h>
int a[20000],ele,pos;
int lin()
{
   if(pos<0)
   {
      return pos+1;
   }
   else if(a[pos]==ele)
   {
      return pos+1;
   }
   else     {
      pos=pos-1;
```

```
        return lin();
    }}
void main()
{
    int n,temp;
    Timer t;
    clrscr();
    printf("\nEnter the number of elements ");
    scanf("%d",&n);
    randomize();
    printf("\nEnter the search element ");
    scanf("%d",&ele);
        for(i=0;i<n;i++)
        {
            a[i]=rand();
        }
        pos=n-1;
        t.start();
        temp=lin();
        t.stop();
    printf("\nElement found at position = %d and time taken = %lf",temp,(double)t.time());
    getch();
}
```

RESULT :

Ex. No. 2 **BINARY SEARCH IMPLEMENTATION**

DATE :

AIM: To implement Binary Search. Determine the time required to search for an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

ALGORITHM :

Binary_Search(a, lower_bound, upper_bound, val)
// 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search
Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
Step 2: repeat steps 3 and 4 while beg <=end
Step 3: set mid = (beg + end)/2
Step 4: if a[mid] = val
set pos = mid
print pos
go to step 6
else if a[mid] > val
set end = mid - 1
else
set beg = mid + 1
[end of if]
[end of loop]
Step 5: if pos = -1
print "value is not present in the array"
[end of if]
Step 6: exit

PROGRAM : BINARY SEARCH

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<timer.h>
int a[20000],n,low,high,ele,pos;

    void sort(int a[], int n)
    { int i, j, temp;
    for(i=0;i<n-1;i++)
    for (j=i+1; J<n;j++)
    {
    If (a[i]>a[j])
     {    temp=a[i];
```

```
a[i]=a[j];
a[j]=temp; }
}
}
int bin()
{
   sort(a,n);
    int mid=(low+high)/2;
    if(low>high)
    {
       return 0;
    }
    if(a[mid]==ele)
    {
       return mid+1;
    }
    else if(a[mid]>ele)
    {
       high=mid-1;
       return bin();
    }
    else if(a[mid]<ele)
    {
       low=mid+1;
       return bin();
    } }
void main()
{
    int temp;
    Timer t;
    clrscr();
    printf("\nEnter the number of elements ");
    scanf("%d",&n);
    randomize();
    printf("\nEnter the search element ");
    scanf("%d",&ele);

     for(i=0;i<n;i++)
        {
           a[i]=rand();
        }
```

```
        pos=n-1;
        low=0;
        high=n-1;
        t.start();;
        temp=bin();
        t.stop();
    }
    printf("\nElement found at position = %d and time taken = %lf",temp,(double)t.time());
    getch();
}
```

RESULT :

Ex. No. 3 **PATTERN SEARCHING**

Date :

AIM :

Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search (char pat [], chartxt[]) that prints all occurrences of pat[] in txt[] using naïve string algorithm? (assume that n > m)

T[] = "THIS IS A TESTTEXT" P[] = "TEST"
Output Pattern found at index 10

ALGORITHM :

NAIVE-STRING-MATCHER (T, P)

1. n ← length [T]
2. m ← length [P]
3. for s ← 0 to n -m
4. do if P [1.....m] = T [s + 1....s + m]
5. then print "Pattern occurs with shift" s

PROGRAM : PATTERN SEARCH

```
#include <stdio.h>
#include <string.h>
void search(char* pat, char* txt)
{int M = strlen(pat);
int N = strlen(txt);
for (int i = 0; i <= N - M; i++) {
int j;
for (j = 0; j < M; j++)
if (txt[i + j] != pat[j])
break;
if (j == M) //  if pat[0...M-1] = txt[i, i+1, ...i+M-1]
printf("Pattern found at index %d \n", i);}}
int main()
{
char txt[] = "AABAACAADAABAAABAA";
char pat[] = "AABA";
 search(pat, txt);
  return 0;
}
```

RESULT :

Ex. No 4. **INSERTION SORT AND HEAP SORT**

Date :

AIM : To Sort a given set of elements using the Insertion sort and Heap sort methods 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.

ALGORITHM

INSERTION SORT

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.
Step2 - Pick the next element, and store it separately in a **key.**
Step3 - Now, compare the **key** with all elements in the sorted array.
Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.
Step 5 - Insert the value.
Step 6 - Repeat until the array is sorted.

HEAP SORT

1. HeapSort(arr)
2. BuildMaxHeap(arr)
3. for i = length(arr) to 2
4. swap arr[1] with arr[i]
5. heap_size[arr] = heap_size[arr] ? 1
6. MaxHeapify(arr,1)
7. End

BuildMaxHeap(arr)

1. BuildMaxHeap(arr)
2. heap_size(arr) = length(arr)
3. for i = length(arr)/2 to 1
4. MaxHeapify(arr,i)
5. End

MaxHeapify(arr,i)

1. MaxHeapify(arr,i)
2. L = left(i)
3. R = right(i)
4. if L ? heap_size[arr] and arr[L] > arr[i]
5. largest = L
6. else

7. largest = i
8. if R ? heap_size[arr] and arr[R] > arr[largest]
9. largest = R
10. if largest != i
11. swap arr[i] with arr[largest]
12. MaxHeapify(arr,largest)
13. End

PROGRAM : INSERTION SORT

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

void main()
{
        int a[1000],n,i,item,j;
        clock_t start, end;
        float time;
        clrscr();
        printf("Enter the size of the array:");
        scanf("%d",&n);

        printf("\nRandomly generated array elements are:\n");
        for(i=0;i<n;i++)
        {
                a[i]=(int)rand()%100;
                printf("%d  ",a[i]);
        }

        start=clock();

        for(i=1;i<n;i++)
        {
                item=a[ i ];
                delay(10);

                for( j=i-1; j>=0 && item<a[ j ]; j--)
                        a[ j+1]=a[ j ];

                a[ j+1]=item;
        }

        end=clock();

        printf("\nSorted list is:\n");
```

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

time= (float)(end-start)/CLK_TCK;
printf("\nTime taken = %f", time);
getch();
```

PROGRAM : HEAP SORT

```
#include<stdio.h>
#include<timer.h>
#include<stdlib.h>
#include<conio.h>
void heap(int a[],int n)
{
   int i,j,k,temp;
   for(i=2;i<=n;i++)
   {
      j=i;
      k=j/2;
      temp=a[j];
      while(k>0&&a[k]<temp)
      {
         a[j]=a[k];
         j=k;
         k=k/2;
      }
      a[j]=temp;
   }
}
void heap1(int a[],int n)
{
   int i,j,k,temp;
   for(i=n/2;i>0;i--)
   {
      k=i;
      temp=a[k];
      j=2*k;
      while(j<=n)
      {
         if(j<n&&a[j]<a[j+1])
         {
            j=j+1;
         }
```

```
        if(temp<a[j])
            {
               a[k]=a[j];
               k=j;
               j=2*k;
            }
            else
            {
               break;
            }
        }
        a[k]=temp;
    }
}
void adjust(int a[],int n)
{
    int i=2,temp=a[1];
    while(i<=n)
    {
        if(i<n&&a[i]<a[i+1])
        {
            i=i+1;
        }
        if(a[i]>temp)
        {
            a[i/2]=a[i];
            i=i*2;
        }
        else
        {
            break;
        }
    }
    a[i/2]=temp;
}
void main()
{
    int a[10000],n,i,temp;
    Timer t;
    clrscr();
    printf("\nEnter the value of n ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        a[i]=n-i;
    }
```

```
    t.start();
    heap1(a,n);
    for(i=n;i>=2;i--)
    {
       temp=a[1];
       a[1]=a[i];
       a[i]=temp;
       adjust(a,i-1);
    }
    t.stop();
    printf("\nTimetaken is %lf ",t.time());
    getch();
}
```

RESULT :

Ex. No. 5 IMPLEMENT GRAPH TRAVERSAL USING BREADTH FIRST SEARCH

Date :

AIM : To develop a program to implement graph traversal using Breadth First Search

ALGORITHM :

Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2
[END OF LOOP]
Step 6: EXIT

PROGRAM : BREADTH FIRST SEARCH

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int q[20],a[20][20],r[20],st=0,ed=0,start,n,i,j;
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
         r[i]=0;
      }
   }
   printf("\nEnter the start node ");
   scanf("%d",&start);
   q[ed++]=start-1;
   r[start-1]=1;
   printf("\nNodes reachable from the origin(%d) are %d ",start,start);
   while(st!=ed)
   {
      for(i=0;i<n;i++)
      {
         if((r[i]==0)&&a[q[st]][i]==1)
         {
            q[ed++]=i;
            r[i]=1;
```

```
                printf("%d ",i+1);
            }
        }
        st++;
    }
    if(ed!=n)
    {
        printf("\nAll nodes are not reachable from origin!!");
    }
    getch();
}
```

RESULT :

Ex. No. 6 IMPLEMENT GRAPH TRAVERSAL USING DEPTH FIRST SEARCH

Date :

AIM : To develop a program to implement graph traversal using Depth First Search

ALGORITHM :

Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)
Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2
[END OF LOOP]
Step 6: EXIT

PROGRAM : DEPTH FIRST SEARCH

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[20][20],n,i,j,st[20],tot=1,top=-1,r[20],flag;
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      r[i]=0;
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   st[++top]=0;
   r[0]=1;
   while(top!=-1)
   {
      flag=0;
      for(j=0;j<n;j++)
      {
         if(r[j]==0&&a[st[top]][j]==1)
         {
            st[++top]=j;
```

```
                tot++;
                r[j]=1;
                flag=1;
                break;
            }
        }
        if(flag==0)
        {
            top=top-1;
        }
    }
    if(tot==n)
    {
        printf("\nAll nodes are reachable from the origin!!");
    }
    else
    {
        printf("\nAll nodes are not reachable from the origin!!");
    }
    getch();
}
```

RESULT :

Ex. No. 7 DIJIKSTRA'S ALGORITHM FOR FINDING SHORTEST PATH

Date :

AIM : To develop a program to find the shortest paths to other vertices using Dijkstra's algorithm.

From a given vertex in a weighted connected graph. :

ALGORITHM :

1. Method Dijkstra(G, s): // G is graph, s is source
2. distance[s] -> 0 // Distance from the source to source is always 0
3. for every vertex vx in the Graph G: // doing the initialization work
4. {
5. if vx ? s
6. {
7. // Unknown distance function from source to each node set to infinity
8. distance[vx] -> infinity
9. }
10. add vx to Queue Q // Initially, all the nodes are in Q
11. }
12.
13. // The while loop
14. Untill the Q is not empty:
15. {
16. // During the first run, this vertex is the source or starting node
17. vx = vertex in Q with the minimum distance[vx]
18. delete vx from Q
19. }
20. // where the neighbor ux has not been deleted yet from Q.
21. for each neighbor ux of vx:
22. alt = distance[vx] + length(vx, ux)
23. // A path with lesser weight (shorter path), to ux is found
24. if alt < distance[ux]:
25. distance[ux] = alt // updating the distance of ux
26.
27. return dist[]
28. end Method

PROGRAM : DIJKSTRA'S ALGORITHM

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

void dijkstra(int n, int v, int cost[10][10],int dist[10])
{
        int count, u, i, w, visited[10], min;
```

```
	for(i=0;i<n;i++)
	{
		visited[i]=0;
		dist[i]=cost[v][i]
		;
	}

	visited[v]=1;
	dist[v]=1;
	count=2;

	while(count<=n)
	{
		min=999;
		for(w=0;w<n;w++)
			if((dist[w]<min) && (visited[w]!=1))
			{
				min=dist[w];
				u=w;
			}

		visited[u]=1;
		count++;
		for(w=0;w<n;w++)
			if((dist[u]+cost[u][w]<dist[w]) && (visited[w]!=1))
				dist[w]=dist[u]+cost[u][w];
	}
}



void main()
{
	int n, v, cost[10][10], dist[10], i, j;
	clrscr();

	printf("Enter number of vertices:");
	scanf("%d",&n);
	printf("\nEnter cost matrix (for infinity, enter 999):\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&cost[i][j]);

	printf("\nEnter source vertex:");
	scanf("%d",&v);

	dijkstra(n,v,cost,dist);
```

```
        printf("\nShortest path from \n");
        for(i=0;i<n;i++)
                if(i!=v)
                        printf("\n%d -> %d = %d", v, i, dist[i]);
         getch();
}
```

Output:

```
Enter number of vertices:7
Enter cost matrix (for infinity, enter 999):
0       2       999     3       999     999     999
2       0       9       999     1       4       999
999     9       0       999     999     3       999
3       999     999     0       5       999     7
999     1       999     5       0       999     4
999     4       3       999     999     0       6
999     999     999     7       4       6       0

Enter source vertex:
0
Shortest path from
0 -> 1 = 2
0 -> 2 = 9
0 -> 3 = 3
0 -> 4 = 3
0 -> 5 = 6
0 -> 6 = 7
```

RESULT :

Ex. No. 8 **PRIMS ALGORITHM FOR FINDING MST**

Date :

AIM : To find Minimum Cost Spanning Tree of a given undirected graph using Prim'salgorithm.

ALGORITHM :

1. Step 1: Select a starting vertex
2. Step 2: Repeat Steps 3 and 4 until there are fringe vertices
3. Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight
4. Step 4: Add the selected edge and the vertex to the minimum spanning tree T
5. [END OF LOOP]
6. Step 5: EXIT

PROGRAM : PRIMS ALGORITHM

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n, v, u,cost[10][10], visited[10]={0}, i, j;
        int count=1, mincost=0, min, a, b;

        rintf("Enter number of vertices:");
        scanf("%d",&n);
        printf("\nEnter cost matrix (For infinity, put 999):\n");

        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }

        visited[1]=1;
        printf("\nThe edges of spanning tree are: \n");

        while(count<n)
        {
                min=999;
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                if(cost[i][j]<min)
                                        if(visited[i]==0)
                                                continue;
```

```
                                else
                                {

                                }
min=cost[i][j];a=u=i;
b=v=j;

                        if(visited[u]==0 || visited[v]==0)
                        {
                                count++;
                                printf("\nEdge(%d, %d) = %d", a, b, min);
                                mincost+=min;
                                visited[b]=1;
                        }
                        cost[a][b]=cost[b][a]=999;
                }
                printf("\nMinimum cost = %d", mincost);
 }
```

Output:

```
Enter number of vertices: 6

Enter cost matrix (For infinity, put 999):
0       2       999     3       999     999
2       0       9       999     1       4
999     9       0       999     999     3
3       999     999     0       5       999
999     1       999     5       0       2
999     4       3       999     2       0

The edges of spanning tree are:

Edge(1, 2) = 2
Edge(2, 5) = 1
Edge(5, 6) = 2
Edge(1, 4) = 3
Edge(6, 3) = 3
Minimum cost = 11
```

RESULT :

Ex. No. 9 FLOYD'S ALGORITHM FOR FINDING ALL PAIR SHORTEST PATH

Date :

AIM : To Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.

ALGORITHM :

- As a first step, initialize the solution matrix to be identical to the input graph matrix.
- The solution matrix is then updated by treating each vertex as an intermediate vertex.
- The plan is to select each vertex one at a time and update any shortest routes that use the selected vertex as an intermediate vertex.
- Vertices 0, 1, 2,.., k-1 are already taken into consideration when vertex number k is chosen as an intermediate vertex.
- There are two potential outcomes for every pair of source and destination vertices (i, j), respectively.
- The shortest path from i to j does not include k as an intermediary vertex. We maintain the current dist[i][j] value.
- k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j].
- The aforementioned optimum substructure property in the all-pairs shortest route issue is depicted in the following image.

PROGRAM: FLOYD'S ALGORITHM.

```
#include<stdio.h>
#include<conio.h>
void floyd(int cm[10][10], int n)
{
        int i, j, k;

        for(k=0;k<n;k++)
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                                if((cm[i][k]+cm[k][j])<cm[i][j])
                                                cm[i][j] = cm[i][k]+cm[k][j];
}
void main()
{
        int n, i, j, cm[10][10];
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);
```

```
        printf("\nEnter adjacency matrix (for infinity, put 9999):\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&cm[i][j]);

        floyd(cm,n);

        printf("\nThe all pair shortest path is :\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%d\t",cm[i][j]);
                printf("\n");
        }
        getch();
}
```

Output:

Enter number of vertices:4

Enter adjacency matrix (for infinity, put 999):

0	5	999	999
999	0	3	999
999	999	0	4
1	8	999	0

The all pair shortest path is :

0	5	8	12
8	0	3	7
5	10	0	4
1	6	9	0

RESULT

Ex. No. 10 WARSHALL's ALGORITHM FOR FINDING ALL PAIR SHORTEST PATH

Date :

AIM : To **c**ompute the transitive closure of a given directed graph usingWarshall's algorithm.

ALGORITHM :

Warshall(A[1...n, 1...n]) // A is the adjacency matrix
R(0) ← A
for k ← 1 **to** n **do**
for i ← 1 **to** n **do**
for j ← **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)

PROGRAM: WARSHALL'S ALGORITHM

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

void warshall(int a[10][10], int n)
{
        int i, j, k;

        for(k=0;k<n;k++)
                for(i=0;i<n;i++)
                        if(a[i][k]==1)
                                for(j=0;j<n;j++)
                                        a[i][j] = a[i][j] || a[k][j];
}
void main()
{
        int n, i, j, a[10][10];
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);

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

        warshall(a,n);
```

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

Output:

Enter number of vertices:4

Enter adjacency matrix :

0	1	0	1
0	0	1	0
0	0	0	1
0	1	0	0

The transitive closure is :

0	1	1	1
0	1	1	1
0	1	1	1
0	1	1	1

RESULT :

EX. NO. 11 MAXIMUM AND MINIMUM USING DIVIDE AND CONQUER TECHNIQUE

Date :

AIM : To develop a program to find out the maximum and minimum numbers in a given list of n numbers using the divide and conquer technique.

ALGORITHM :

Step 1 : Divide the problem into sub-problems and find the max and min of each group, now max.

Step 2 : Of each group will compare with the only max of another group and min with min.

PROGRAM :

```
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
 {
  max = min = a[i];
 }
 else
 {
  if(i == j-1)
  {
   if(a[i] <a[j])
   {
    max = a[j];
    min = a[i];
   }
   else
   {
    max = a[i];
    min = a[j];
   }
  }
  else
  {
   mid = (i+j)/2;
   maxmin(i, mid);
```

```
   max1 = max; min1 = min;
   maxmin(mid+1, j);
   if(max <max1)
    max = max1;
   if(min > min1)
    min = min1;
  }
 }
}
int main ()
{
 int i, num;
 printf ("\nEnter the total number of numbers : ");
 scanf ("%d",&num);
 printf ("Enter the numbers : \n");
 for (i=1;i<=num;i++)
  scanf ("%d",&a[i]);

 max = a[0];
 min = a[0];
 maxmin(1, num);
 printf ("Minimum element in an array : %d\n", min);
 printf ("Maximum element in an array : %d\n", max);
 return 0;
}
```

OUTPUT :

RESULT:

EX. NO. 12 IMPLEMENT MERGE SORT AND QUICK SORT

Date :

AIM : To implement Merge sort and Quick sort methods to sort an array of elements and determine the time required to sort. 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.

ALGORITHM : MERGE SORT

ALGORITHM-MERGE SORT

1. If p<r
2. Then q → (p+ r)/2
3. MERGE-SORT (A, p, q)
4. MERGE-SORT (A, q+1,r)
5. MERGE (A, p, q, r)

FUNCTIONS: MERGE (A, p, q, r)

1. n 1 = q-p+1
2. n 2= r-q
3. create arrays [1.....n 1 + 1] and R [1.....n 2 +1]
4. for i ← 1 to n 1
5. do [i] ← A [p+ i-1]
6. for j ← 1 to n2
7. do R[j] ← A[q + j]
8. L [n 1+ 1] ← ∞
9. R[n 2+ 1] ← ∞
10. I ← 1
11. J ← 1
12. For k ← p to r
13. Do if L [i] ≤ R[j]
14. then A[k] ← L[i]
15. i ← i +1
16. else A[k] ← R[j]
17. j ← j+1

ALGORITHM : QUICK SORT

```
QUICKSORT (array A, start, end)
{
 1 if (start < end)
 2 {
3 p = partition(A, start, end)
4 QUICKSORT (A, start, p - 1)
5 QUICKSORT (A, p + 1, end)
6 }
}
```

```
PARTITION (array A, start, end)
{
    1 pivot ? A[end]
    2 i ? start-1
    3 for j ? start to end -1 {
    4 do if (A[j] < pivot) {
    5 then i ? i + 1
    6 swap A[i] with A[j]
    7  }}
    8 swap A[i+1] with A[end]
    9 return i+1
}
```

PROGRAM : MERGE SORT

```
#include<stdio.h>
#include<conio.h>
#include<timer.h>
#include<stdlib.h>
void copy(int a[],int b[],int st,int end)
{
   int i;
   for(i=st;i<end;i++)
   {
      b[i-st]=a[i];
   }
}
void merge(int a[],int n)
{
   int *b,*c,i,j;
   if(n==1)
   {
      return;
   }
   b=(int*)malloc(n/2);
   copy(a,b,0,n/2);
   merge(b,n/2);
   c=(int*)malloc(n-(n/2));
   copy(a,c,n/2,n);
   merge(c,n-(n/2));
   j=0;
   for(i=0;(i+j)<n;)
   {
```

```
        if(j==n-(n/2))
        {
           a[i+j]=b[i];
           i=i+1;
        }
        else if(i==(n/2))
        {
           a[i+j]=c[j];
           j=j+1;
        }
        else if(b[i]>c[j])
        {
           a[i+j]=c[j];
           j=j+1;
        }
        else
        {
           a[i+j]=b[i];
           i=i+1;
        }
    }
}
void main()
{
    int *a,n,i;
    Timer t;
    clrscr();
    printf("\nEnter the value of n ");
    scanf("%d",&n);
    a=(int*)malloc(n);
    randomize();
    for(i=0;i<n;i++)
    {
       a[i]=rand();
    }
    t.start();
    merge(a,n);
    t.stop();
    printf("\nTime taken = %lf",t.time());
    getch();
}
```

PROGRAM : QUICK SORT

```
#include<stdio.h>
#include<conio.h>
#include<TIMER.H>
#include<alloc.h>
#include<stdlib.h>
int n;
int quick(int a[],int l,int h)
{
   int piv,temp,i,j;
   if(l==h)
   {
      return l;
   }
   piv=a[l];
   i=l+1;
   j=h;
   while(i<=j)
   {
      for(;i<=h&&a[i]<=piv;i++);
      for(;j>=(l+1)&&a[j]>=piv;j--);
      if(j==l)
      {
         return l;
      }
      else if(i==h+1)
      {
         a[l]=a[h];
         a[h]=piv;
         return h;
      }
      else if(i>j)
      {
         a[l]=a[j];
         a[j]=piv;
         return(j);
      }
      else
      {
         temp=a[i];
         a[i]=a[j];
         a[j]=temp;
      }
```

```
    }
}
void quicksort(int a[],int st,int end)
{
    int pos;
    if(st<end)
    {
        pos=quick(a,st,end);
        quicksort(a,st,pos-1);
        quicksort(a,pos+1,end);
    }
}
void main()
{
    int *a,i;
    Timer t;
    clrscr();
    printf("\nEnter the value of n ");
    scanf("%d",&n);
    a=(int *)malloc(n);
    randomize();
    for(i=0;i<n;i++)
    {
        a[i]=rand();
    }
    t.start();
    quicksort(a,0,n-1);
    t.stop();
    for(i=0;i<n-1;i++)
    {
        if(a[i]>a[i+1])
        {
            printf("%d %d",a[i],i);
        }
    }
    printf("\nTime taken = %lf",t.time());
    getch();
}
```

RESULT :

EX. NO. 13 N QUEENS PROBLEM USING BACKTRACKING

Date :

AIM : To implement N Queens problem using Backtracking.

ALGORITHM :

1. N - Queens (k, n)
2. {
3. For i ← 1 to n
4. do if Place (k, i) then
5. {
6. x [k] ← i;
7. if (k ==n) then
8. write (x [1....n));
9. else
10. N - Queens (k + 1, n);
11. }
12. }
13. Place (k, i)
14. {
15. For j ← 1 to k - 1
16. do if (x [j] = i)
17. or (Abs x [j]) - i) = (Abs (j - k))
18. then return false;
19. return true;
20. }

PROGRAM : N QUEENS PROBLEM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   int n,arr[20][20],i,j,k,tot=0,x[20],top;
   char ch;
   clrscr();
   printf("\nEnter the total number of queens ");
   scanf("%d",&n);
   top=0;
   x[top]=-1;
```

```
printf("\nThe solution to the %d queens problem is ",n);
while(top>=0)
{
   x[top]=x[top]+1;
   while(x[top]<n)
   {
      for(i=0;i<top;i++)
      {
         if(x[i]==x[top]||(abs(x[top]-x[i])==abs(top-i)))
         {
            break;
         }
      }
      if(i==top)
      {
         break;
      }
      x[top]=x[top]+1;
   }
   if(x[top]==n)
   {
      top=top-1;
   }
   else if(top==n-1)
   {
      tot=tot+1;
      ch=getch();
      if(ch==0)
      {
         ch=getch();
      }
      printf("\n");
      for(i=0;i<n;i++)
      {
         printf("\n");
         for(j=0;j<n;j++)
         {
            if(x[i]==j)
            {
               printf("Q ");
            }
            else
            {
               printf("X ");
```

```
                }
            }
        }
        top=top-1;
    }
    else
    {
        top=top+1;
        x[top]=-1;
    }
}
if(tot==0)
{
    printf("not possible!!");
}
else
{
    printf("\n\nThe total number of solutions is %d.",tot);
}
getch();
}
```

OUTPUT :

RESULT :

EX. NO. 14 TRAVELLING SALESPERSON PROBLEM

Date :

AIM : To implement any scheme to find the optimal solution for the Travelling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

ALGORITHM :

Step 1: Firstly, we will consider City 1 as the starting and ending point. Since the route is cyclic, any point can be considered a starting point.
Step 2: As the second step, we will generate all the possible permutations of the cities, which are (n-1)!
Step 3: After that, we will find the cost of each permutation and keep a record of the minimum cost permutation.
Step 4: At last, we will return the permutation with minimum cost.

PROGRAM :

```
int minimum_key(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;
     for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
     return min_index;
}
 vector<vector<int>> MST(int parent[], int graph[V][V])
{
   vector<vector<int>> v;
   for (int i = 1; i < V; i++)
   {
      vector<int> p;
      p.push_back(parent[i]);
      p.push_back(i);
      v.push_back(p);
      p.clear();
   }
   return v;
```

```
}
// getting the Minimum Spanning Tree from the given graph using Prim's Algorithm
vector<vector<int>> primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    // to keep track of vertices already in MST
    bool mstSet[V];
    // initializing key value to INFINITE & false for all mstSet
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    // picking up the first vertex and assigning it to 0
    key[0] = 0;
    parent[0] = -1;
    // The Loop
    for (int count = 0; count < V - 1; count++)
    {
        // checking and updating values wrt minimum key
        int u = minimum_key(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    vector<vector<int>> v;
    v = MST(parent, graph);
    return v;
}
```

RESULT :

EX. NO. 15 **RANDOMIZED ALGORITHMS FOR FINDING THE KTH SMALLEST NUMBER**

Date :

AIM : To implement randomized algorithms for finding the kth smallest number.

ALGORITHM :

ALGORITHM *LomutoPartition*$(A[l..r])$

//Partitions subarray by Lomuto's algorithm using first element as pivot
//Input: A subarray $A[l..r]$ of array $A[0..n-1]$, defined by its left and right
// indices l and r $(l \le r)$
//Output: Partition of $A[l..r]$ and the new position of the pivot
$p \leftarrow A[l]$
$s \leftarrow l$
for $i \leftarrow l+1$ **to** r **do**
  if $A[i] < p$
    $s \leftarrow s+1$; swap$(A[s], A[i])$
swap$(A[l], A[s])$
return s

ALGORITHM *Quickselect*$(A[l..r], k)$

//Solves the selection problem by recursive partition-based algorithm
//Input: Subarray $A[l..r]$ of array $A[0..n-1]$ of orderable elements and
// integer k $(1 \le k \le r-l+1)$
//Output: The value of the kth smallest element in $A[l..r]$
$s \leftarrow$ *LomutoPartition*$(A[l..r])$ //or another partition algorithm
if $s = k-1$ **return** $A[s]$
else if $s > l+k-1$ *Quickselect*$(A[l..s-1], k)$
else *Quickselect*$(A[s+1..r], k-1-s)$

PROGRAM :

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

int N = 20;
int A[20];

void swap(int dex1, int dex2) {
    int temp = A[dex1];
    A[dex1] = A[dex2];
    A[dex2] = temp;
}

int partition(int start, int end) {
    int i = start + 1;
    int j = i;
    int pivot = start;
    for (; i < end; i++) {
        if (A[i] < A[pivot]) {
            swap(i, j);
            j++;
        }
    }
    if (j <= end)
        swap(pivot, (j - 1));

    return j - 1;
}

void quick_sort(int start, int end, int K) {
    int part;
    if (start < end) {
        part = partition(start, end);
        if (part == K - 1)
            printf("kth smallest element : %d ", A[part]);
        if (part > K - 1)
            quick_sort(start, part, K);
        else
            quick_sort(part + 1, end, K);
    }
    return;
}

int main(int argc, char **argv) {
    int i;
```

```
    time_t seconds;
    time(&seconds);
    srand((unsigned int) seconds);

    for (i = 0; i < N; i++)
        A[i] = rand() % (1000 - 1 + 1) + 1;

    printf("The original sequence is:  ");
    for (i = 0; i < N; i++)
        printf("%d ", A[i]);

    printf("\nEnter the Kth smallest you want to find: ");
    int k;
    scanf("%d", &k);
    quick_sort(0, N, k);
}
```

OUTPUT:

```
The original sequence is:
909 967 552 524 735 383 616 718 904 945 730 173 143 954 482 307 228 35 224 703
Enter the Kth smallest you want to find: 3
kth smallest element : 173
```

RESULT :