PAGE NO :
DATE : 19/06/23

Experiment-1

Write program to do the following:

a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

b) Check whether a given graph is connected or not using DFS method.

Program:

(a)
```
# include <stdio.h>
# include <conio.h>

int a[20][20], q[20], visited[20], n, i, j,
f=0, r=-1;

Void bfs (int v)
{ for (i=1; i<=n; i++)
    if (a[v][i] && !visited[i])
      q[++r] = i;
    if (f<=r)
      { visited [q[f]] = 1;
        bfs (q[f++]);
      }
}

void main()
{ int v;
  printf ("\n Enter the number of vertices :");
  scanf ("%d", &n);
```

→

```
for (i=1; i<=n; i++)
{   q[i]=0;
    visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1; i<=n; i++)
  for(j=1; j<=n; j++)
    scanf("%d", &a[i][j]);

printf("\n Enter the starting vertex :");
scanf("%d", &v);
bfs(v);

printf("\n The node which are reachable are:\n");
for(i=1; i<=n; i++)
  if (visited[i])
    printf("%d\t", i);

  getch();
}
```

Output:

```
Enter the number of vertices : 4
Enter graph data in matrix form:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the starting vertex : 1
The node which are reachable are:
2       3       4
```

PAGE NO :
DATE : 19/06/23

(b) Program:

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

int a[20][20], reach[20], n;

void dfs (int v)
{ int i;
  reach[v] = 1;
  for (i=1; i<=n; i++)
    if (a[v][i] && !reach[i])
    { printf("\n %d → %d", v, i);
      dfs (i);
    }
}

int main()
{ int i, j, count = 0;
  printf("\n Enter no. of vertices : ");
  scanf("%d", &n);
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        { reach[i] = 0;
          a[i][j] = 0;
        }
    printf("\n Enter adjacency matrix:\n");
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        scanf("%d", &a[i][j]);
      dfs(1);
    for (i=1; i<=n; i++)
    if (reach[i])
    count++;
```

```
if (count == n)
    printf("\n Graph is connected");
else
    Printf("\n Graph is disconnected");
getch();
return(0);
}
```

Output:

(i) Enter no. of vertices : 4

Enter adjacency matrix :

```
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
```

1 → 2
2 → 4
4 → 3
Graph is connected.

(ii) Enter no. of vertices : 4

Enter adjacency matrix :

```
0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0
```

1 → 2
2 → 3
Graph is Disconnected

O/P saw
26/6/23.

PAGE NO :
DATE : 26/06/2023

Experiment - 2

Write a program to obtain the Topological ordering of vertices in a given digraph

Program:

```
#include <stdio.h>

int main()
{ int i, j, k, n, a[10][10], indeg[10], visited[10],
    count = 0;

  printf("Enter the no. of vertices:");
      scanf("%d", &n);

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

for (i=0; i<n; i++)
{ indeg[i] = 0;
  visited[i] = 0;
}

for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    indeg[i] = indeg[i] + a[j][i];

printf("\n The Topological order is:");
```

→

```
while (count < n)
{ for (k=0; k<n; k++)
  { if ((indeg[k] == 0) && (visited[k] == 0))
    { printf ("%d", (k+1));
      visited[k] = 1;
    }
    for (i=0; i<n; i++)
    { if (a[i][k] == 1)
        indeg[k]--;
    }
  }
  count++;
}
return 0;
}
```

Output:

Enter the no. of vertices:

7

Enter the adjacency matrix:

0000100

1010000

0001000

0000100

0000010

0000001

0000000

The topological order is: 2 1 3 4 5 6 7

PAGE NO :
DATE : 03/07/2023

Experiment - 3

Implement Johnson Trotter algorithm to generate permutations.

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

#define MAX_N 10

void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void printPermutation (int permutation[],
                       int direction[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf ("%d", permutation[i]);
    }
    printf ("\n");
}

void generatePermutations (int n)
{
    int permutation[MAX_N];
    int direction[MAX_N];
    bool mobile[MAX_N];

    for (int i = 0; i < n; i++)
    {
        permutation[i] = i + 1;
        direction[i] = -1;
        mobile[i] = true;
    }
```

→

```
printPermutation (permutation, direction, n);

int mobileElement, mobileIndex, temp;

while (true)
{ mobileElement = -1;
  mobileIndex = -1;

  for (int i = 0; i < n; i++)
  { if (direction[i] == -1 && i > 0 &&
      permutation[i] > permutation[i-1] &&
      mobile[i])
    { if (mobileElement == -1 || permutation[i] >
                                 mobileElement)
      { mobileElement = permutation[i];
        mobileIndex = i;
      }
    }
  if (direction[i] == 1 && i < (n-1) &&
      permutation[i] > permutation[i+1] && mobile[i])
  { if (mobileElement == -1 || permutation[i] > mobileElement
    { mobileElement = permutation[i];
      mobileIndex = i;
    }
  }
}

if (mobileIndex == -1)
  { break;
  }
```

→

```
        if (direction[mobileIndex] == -1)
        {   swap(&permutation[mobileIndex],
                 &permutation[mobileIndex - 1]);
            swap(&direction[mobileIndex],
                 &direction[mobileIndex - 1]);
        }
        else
        {   swap(&permutation[mobileIndex],
                 &permutation[mobileIndex + 1]);
            swap(&direction[mobileIndex],
                 &direction[mobileIndex + 1]);
        }

        for(int i = 0; i < n; i++)
        {   if(permutation[i] > mobileElement)
            {   direction[i] *= -1;
            }
        }
        printPermutation(permutation, direction, n);
    }
}

int main()
{   int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX_N)
    {   printf("Invalid Input!\n");
        return 0;
    }
    generatePermutations(n);
    return 0;
}
```

O/P for n = 3

→

PAGE NO :
DATE :

Output:

Enter the value of n : 4

1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4

PAGE NO :
DATE : 10/07/2023

Experiment - 4

Sort a given set of N integer elements using Merge sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Program :

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

void mergesort(int a[], int i, int j)
void merge(int a[], int i1, int j1, int i2, int j2);

int main()
{ int a[50000], n, i;
  clock_t start_t, end_t;
  double total_t;
  srand(time(NULL));

  printf("Enter the number of elements :\n");
  scanf("%d", &n);

  printf("Enter array Elements :");
  for(i=0; i<n; i++)
  a[i] = rand()%10000;
  start_t = clock();
  printf("Starting of the program, start_t = %ld\n", start_t);
  mergesort(a, 0, n-1);
  end_t = clock();
```

→

```
printf("End of the program, end_t = %ld \n", end_t);
total_t = (double)(end_t - start_t)/CLOCKS_PER_SEC;
printf("Total time taken by CPU: %f \n", total_t);

printf("Sorted array is: ");
for(i=0; i<n; i++)
  printf("%d", a[i]);

  return 0;

}

void mergesort(int a[], int i, int j)
{ int mid;
    if(i<j)
    { mid = (i+j)/2;
      mergesort(a, i, mid);
      mergesort(a, mid+1, j);
    }
}

Void merge(int a[], int i1, int j1, int i2, int j2)
{ int temp[50000];
  int i, j, k;
  i = i1;
  j = i2;
  k = 0;

  while(i<=j1 && j<=j2)
  { if(a[i] < a[j])
      temp[k++] = a[i++];
```

```
        else
        temp[k++] = a[j++];
    }

    while (i<=j1)
    temp[k++] = a[i++];

    while (j<=j2)
    temp[k++] = a[j++];

    for (i=i1, j=0; i<=j2; i++, j++)
    a[i] = temp[j];
}
```

Output:

Enter the number of elements = 25000

Enter array elements : (By Random)

Starting of the program, start-t = 1331

End of the program, end-t = 10091

Total time taken by CPU = 0.008760

Sorted array :

PAGE NO :
DATE : 17/7/2023

Experiment - 5

Sort a given set of N integers elements using quick sort technique and compute its time taken.

Program:

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

void swap (int *a, int *b)
{ int t = *a;
   *a = *b;
   *b = t;
}

int partition (int arr[], int low, int high)
{ int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++)
  { if (arr[j] <= pivot)
    { i++;
      swap (&arr[i], &arr[j]);
    }
  }
  swap (&arr[i+1], &arr[high]);
  return (i+1);
}
```

PAGE NO :
DATE :

```
void quickSort (int arr [], int low, int high)
{ if (low < high)
    { int pi = partition (arr, low, high);
      quickSort (arr, low, pi-1);
      quickSort (arr, pi+1, high);
    }
}

int main ()
{ int n;
  double total_t;
  clock_t start_t, end_t;

  printf ("Enter the number of elements : ");
  scanf ("%d", &n);
  int arr[n];

  printf ("Enter the max value of the elements : ");
  int max;

  scanf ("%d", &max);
  for (int i=0; i<n; i++)
    { arr [i] = rand % max;
    }
  printf ("\n Unsorted array : \n");
  for (int i=0; i<n; i++)
    { printf ("%d", arr[i]);

  start_t = clock();
    printf ("\n\n Starting of the program : %ld\n",
            start_t);

  quickSort (arr, 0, n-1);
```

```
printf("\n\n\n Sorted array : \n");
for(int i=0; i<n; i++)
{ printf("%d", arr[i]);
}

end_t = clock();
printf("\n\nEnd of the program: %ld\n", end_t);

total_t = (double)(end_t - start_t)/CLOCKS_PER_SEC;
printf("\n\n Total time taken by CPU: %f\n", total_t);

return 0;
}
```

Output:

Enter the number of elements = 10

Enter the max value of the elements = 10

Unsorted array:

1 7 4 0 9 4 8 8 2 4

Starting of the program: 3093

Sorted array:

0 1 2 4 4 4 7 8 8 9

End of the program: 3093.

Total time taken by CPU: 0.000000.

O/P seen
1/1/23.

PAGE NO :
DATE : 24/07/23

Experiment - 6

Implement all pairs shortest path using Floyd's algorithm.

Program:

```
#include<stdio.h>
#define INFINITY 999

int nV;
Void printMatrix(int matrix[][nV]);
void floyd(int graph[][nV])
{  int matrix[nV][nV], i, j, k;

   for(i=0; i<nV; i++)
      for(j=0; j<nV; j++)
         matrix[i][j] = graph[i][j];

   for(k=0; k<nV; k++)
   {  for(i=0; i<nV; i++)
      {  for(j=0; j<nV; j++)
         {  if(matrix[i][k] + matrix[k][j] < matrix[i][j])
               matrix[i][j] = matrix[i][k] + matrix[k][j];
         }
      }
   }
   printMatrix(matrix);
}

void printMatrix(int matrix[][nV])
{  printf("\n All pairs Shortest Path is: \n");
```

→

```
        for(i=0; i<nV; i++)
        {  for(j=0; j<nV; j++)
           {  if(matrix[i][j] == INFINITY)
                 printf("%4s", "INF");
              else
                 printf("%4d", matrix[i][j]);
           }
           printf("\n");
        }
}

int main()
{  printf("Enter the number of vertices in the graph: ");
   scanf("%d", &nV);

   int graph[nV][nV];

   printf("Enter the weight of edges in the graph :\n");
   for(int i=0; i<nV; i++)
   {  for(int j=0; j<nV; j++)
      {  scanf("%d", &graph[i][j]);
      }
   }
   floyd(graph);
}
```

DATE :

Output:

Enter the number of vertices in the graph : 4

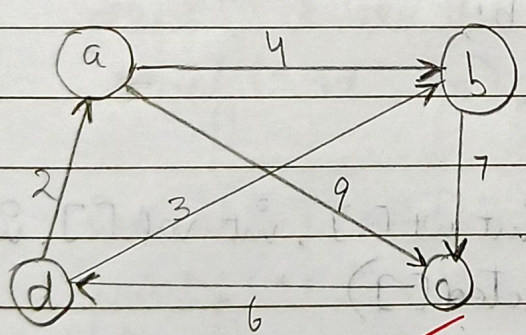
Enter the weight of edges in the graph :

```
0   4   9   999
999 0   7   999
999 999 0   6
2   3   999 0
```

All pairs shortest path is :

```
0   4   9   15
15  0   7   13
8   9   0   6
2   3   10  0
```

Traversal (question):



	a	b	c	d
a	0	4	9	∞
⇒ b	∞	0	7	∞
c	∞	∞	0	6
d	2	3	∞	0

qp 1st
24/7/23

PAGE NO :
DATE : 31/07/2023

Experiment - 7

1. Implement 0/1 Knapsack problem using dynamic programming.

2. Find Minimum Cost Spanning Tree of a given undirected graph using:
a) Prim's Algorithm
b) Kruskal's Algorithm.

Program:

1. Knapsack:

```
#include<stdio.h>

int max(int a, int b)
{ return (a>b)? a:b;
}

int knapSack(int W, int wt[], int val[], int n,
             int selected[])
{ int i, w;
  int K[n+1][W+1];

  for(i=0; i<=n; i++)
    { for(w=0; w<W; w++)
      { if(i==0 || w==0)
          { K[i][w]=0;
        else if(wt[i-1]<=w)
          K[i][w]=max(val[i-1]+K[i-1][w-wt[i-1]],
                      K[i-1][w]);
        else
          K[i][w]=K[i-1][w];
      }
  }
```

```
    int res = K[n][W];
    w = W;
    for (i = n; i > 0 && res > 0; i--)
    { if (res == K[i-1][w])
        continue;
      else
      { selected[i-1] = 1;
        res = res - val[i-1];
        w = w - wt[i-1];
      }
    }
    return K[n][W];
}
int main()
{ int n, W, i;
  printf("Enter the number of items : ");
  scanf("%d", &n);

  int val[n], wt[n], selected[n];
  printf("Enter the values of items : ");
  for(i = 0; i < n; i++)
      scanf("%d", &val[i]);

  printf("Enter the weights of the items: ");
  for(i = 0; i < n; i++)
      scanf("%d", &wt[i]);

  printf("Enter the Knapsack Capacity : ");
  scanf("%d", &W);

  int max_profit = knapSack(W, wt, val, n, selected);
  printf("The maximum profit is %d\n", max_profit;
```

→

```
    printf("The objects selected for the optimal
            solution are: ");
    for (i=0; i<n; i++)
    { if (selected[i] == 1)
        printf("%d", i+1);
    }
    return 0;
}
```

Output:

Enter the number of items: 4
Enter the values of items: 3 4 5 6
Enter the weights of items: 2 3 4 5
Enter the knapsack capacity: 5

The maximum profit is: 7

The objects selected for the optimal solution are:
1 2.

Item	weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

PAGE NO :
DATE :

2. Prim's Algorithm:

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

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0}, cost[10][10], mincost=0,min;

void main()
{ printf("Prim's Algorithm:\n");
  printf("Enter the number of nodes:");
  scanf("%d",&n);

  printf("Enter the adjacency Matrix:\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("\n");

  printf("\n The edges of Minimum Cost Spanning
          tree are:");
  while(ne<n)
  { for(i=1;min=999;i<=n;i++)
      for(j=1;j<=n;j++)
        if(cost[i][j]<min)
          if(visited[i]!=0)
          { min=cost[i][j];
            a=u=i;
            b=v=j;
          }
```

→

```
    if (visited[u] == 0 || visited[v] == 0)
    { printf("\n Edge %d: (%d,%d) weight: %d",
                ne++, a, b, min);
      mincost += min;
      visited[b] = 1;
    }
    cost[a][b] = cost[b][a] = 999;
  }
  printf("\n\n Minimum Cost = %d", mincost);
}
```

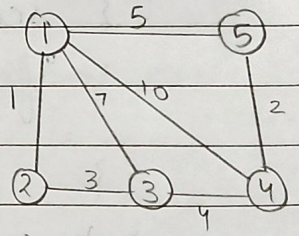
Output:

Prim's Algorithm:

Enter the number of nodes: 5

Enter the adjacency matrix:

0	1	7	10	5
1	0	3	∞	∞
7	3	0	4	∞
10	∞	4	0	2
5	∞	∞	2	0



The edges of Minimum Cost Spanning Tree are:

Edge 1: (1, 2) Weight: 1

Edge 2: (2, 3) Weight: 3

Edge 3: (3, 4) Weight: 4

Edge 4: (4, 5) Weight: 2

Minimum Cost = 10

PAGE NO :
DATE :

3. Kruskal's Algorithm:

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[10][10], parent[10];

int find(int);
int uni(int, int);

void main()
{
  printf("Kruskal's Algorithm :\n");
  printf("Enter the number of vertices:\n");
  scanf("%d", &n);
  printf("Enter the adjacency matrix :\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;
    }
  }

printf("\n The edges of the Minimum Cost
        Spanning Tree are :\n");
while (ne < n)
  {
    for (i = 1; min = 999; i <= n; i++)
    {
      for (j = 1; j <= n; j++)
      {
        if (cost[i][j] < min)
          a = u = i;
          b = v = j;
      }
    }
  }
```

→

```
        u = find(u);
        v = find(v);

        if(uni(u,v))
        { printf("Edge %d:(%d,%d) weight:%d\n",
                ne++, a, b, min);
          mincost += min;
          cost[a][b] = cost[b][a] = 999;
        }

        printf("\n Minimum cost: %d\n", mincost);
}

int find(int i)
{ while(parent[i])
      i = parent[i];
      return i;
}

int uni(int i, int j)
{ if( i != j )
    { parent[j] = i;
      return 1;
    }
  return 0;
}
```

Output:

Kruskal's Algorithm:

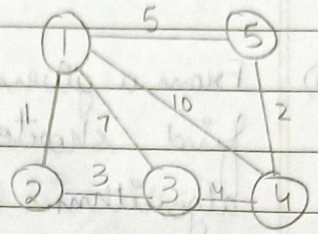
Enter the number of vertices: 5

→

PAGE NO :
DATE :

Enter the adjacency matrix:

0	1	7	10	5
1	0	3	∞	∞
7	3	0	4	∞
10	∞	4	0	2
5	∞	∞	2	0



The edges of Minimum Cost Spanning Tree are:

Edge 1: (1,2) Weight: 1
Edge 2: (4,5) Weight: 2
Edge 3: (2,3) Weight: 3
Edge 4: (3,4) Weight: 4

Minimum Cost = 10

Ans
1/8/23

PAGE NO :
DATE : 14/09/2023

Experiment-8

a) From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

b) Implement "N-Queen Problem" using Backtracking.

Program:

(a) Dijkstra's Algorithm:

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

#define V 20

int minDistance (int dist[], bool visited[])
{ int min = INT_MAX, min_index;
    for(int v=0; v<V; v++)
    { if(!visited[v] && dist[v]<=min)
        { min = dist[v];
          min_index = v;
        }
    }
    return min_index;
}

void printSolution(int dist[], int parent[], int src)
{ printf("Vertex \t Distance \t Path \n");
    for(int i=0; i<V; i++)
    { printf("%d \t %d \t \t %d -> ", i, dist[i], src);
```

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

void dijkstra(int graph[V][V], int src)
{ int dist[V];
  bool visited[V];
  int parent[V];

  for(int i=0; i<V; i++)
  { dist[i] = INT_MAX;
    visited[i] = false;
    parent[i] = -1;
  }

  dist[src] = 0;

  for(int count=0; count<V-1; count++)
  { int u = minDistance(dist, visited);
    visited[u] = true;

    for(int v=0; v<V; v++)
    { if(!visited[v] && graph[u][v] &&
         dist[u] != INT_MAX &&
         dist[u] + graph[u][v] < dist[v])
      { dist[v] = dist[u] + graph[u][v];
        parent[v] = u;
      }
    }
  }
  printSolution(dist, parent, src);
}
```

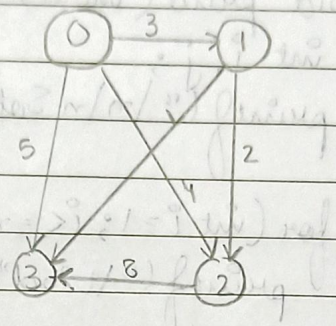
→

```
int main()
{ int graph[V][V];
  int vertices, edges;

printf("Enter the number of vertices (<=%d):", V);
scanf("%d", &vertices);

printf("Enter the number of edges: ");
scanf("%d", &edges);

for (int i=0; i<V; i++)
{ for(int j=0; j<V; j++)
  { graph[i][j] = 0;
  }
}

printf("Enter the edges and weights:\n");
for (int i=0; i< edges; i++)
  { int u,v,w;
    scanf("%d %d %d", &u, &v, &w);
    graph[u][v] = w;
  }

int source;
printf("Enter the source vertex (0 to %d): ",
       vertices-1);
scanf("%d", &source);

dijkstra(graph, source);

return 0;

}
```

PAGE NO :
DATE :

Output:

Enter the number of vertices (<=20) : 4
Enter the number of edges : 7
Enter the edges and weights :
0 1 3
0 2 4
0 3 5
1 2 2
1 3 1
2 3 8
3 2 3



Enter the source vertex (0 to 3) : 0

Vertex	Distance	Path
0	0	0
1	3	0 → 1
2	4	0 → 2
3	4	0 → 1 → 3

(b) N-Queen Problem

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

int board[20], count;

int main()
{ int n, i, j;
  void queen(int row, int n)
  printf("Enter the number of queens : ");
  scanf("%d", &n);
```

```
    queen(1,n);
    return 0;
}

void print(int n)
{ int i,j;
  printf("\n\n Solution %d:\n\n", ++count);

  for(int i=1; i<=n; ++i)
    printf("\t%d", i);

  for(i=1; i<=n; ++i)
  { printf("\n\n%d", i);
    for(j=1; j<=n; ++j)
    { if( board[i]==j )
        print("\tQ");
      else
        print("\t-");
    }
  }
}

int place(int row, int column)
{ int i;
  for(int i=1; i<=row; ++i)
  { if(board[i]==column)
      return 0;
    else
      if(abs(board[i]-column)==abs(i-row))
      return 0
  }

  return 1
}
```

PAGE NO :
DATE :

```
void queen (int row, int n)
{ int column;
  for (column = 1; column <= n; ++column)
  { if (place (row, column)
    { board[row] = column;
      if (row == n)
      print (n);
      else
      queen (row + 1, n);
    }
  }
}
```

Output:

Enter the number of queens : 4

Solution - 1 :

	1	2	3	4
1	-	0	-	-
2	-	-	-	0
3	0	-	-	-
4	-	-	0	-

Sw 16/8/23.

Solution - 2 :

	1	2	3	4
1	-	-	0	-
2	0	-	-	-
3	-	-	-	0
4	-	0	-	-

PAGE NO :
DATE : 21-09-2023

Experiment - 9

Sort a given set of N integer elements using Heap-Sort technique and compute its time taken.

Program:

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

void heapify(int arr[], int n, int i)
{ int largest = i;
  int left = 2*i + 1;
  int right = 2*i + 2;

  if( left<n && arr[left] > arr[largest])
    largest = left;

  if (right<n && arr[right] > arr[largest])
    largest = right;

  if(largest != i)
  { int temp = arr[i];
    arr[i] = arr[largest];
    arr[largest] = temp;
    heapify(arr, n, largest);
  }
}

void heapsort(int arr[], int n)
{ for(int i = n/2 - 1; i >= 0; i--)
    heapify(arr, n, i);
```

```
for (int i = n-1 ; i>0; i--)
{ int temp = arr[0];
  arr[0] = arr[i];
  arr[i] = temp;
  heapify (arr, i, 0);
}
}

int main()
{ int n;

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

  int arr[n];

  srand (time[NULL]);
  printf("\n\n Randomly generated elements: \n");
  for (int i = 0; i<n; i++)
    arr[i] = rand() % 1000;

  printf("original array: \n");
  for (int i=0; i<n; i++)
    printf("%d", arr[i]);

  clock_t start_time = clock();

  heapSort(arr, n);

  clock_t end_time = clock();

  →
```

21/8/23

```
printf("\n\n Sorted array: \n");
for (int i=0; i<n; i++)
    printf("%d", arr[i]);

double time_taken = (double)(end_time - start_time) /
                                    CLOCKS_PER_SEC;

printf("\n\n Time Taken: %f seconds\n", time_taken);

return 0;

}
```

Output:

Enter the number of elements: 20

Randomly generated elements:

Original array:

307 592 675 816 440 301 524 733 505 899
772 822 943 400 913 152 138 620 350 6

Sorted array:

6 138 152 301 307 350 400 440 505 524 591
620 675 733 772 816 822 899 913 943

Time taken: 0.000003