VISVESVARAYATECHNOLOGICALUNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Aisha Taffazul Chesti (1BM21CS010)

in partial fulfillment for the award of the degree of
BACHELOROFENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019 May-2023 to July-2023
B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Aisha Taffazul Chesti (1BM21CS010),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Antara Roy Choudhry	Dr. Jyothi S Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE

Index Sheet

10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	39
11	Implement "N-Queens Problem" using Backtracking.	43

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

1.Write program to do the following:

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

Solution:

Code:

```
#include<stdio.h>

#define MAX_VERTICES 20

int a[MAX_VERTICES][MAX_VERTICES], q[MAX_VERTICES], visited[MAX_VERTICES];
int n, f, r;

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

int main() {
   int v, i, j;

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

   for (i = 1; i <= n; i++) {
      q[i] = 0;
      visited[i] = 0;
   }
```

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

    f = 0;
    r = -1;
    visited[v] = 1;
    bfs(v);

    printf("The nodes that are reachable are:\n");

    int allNodesReachable = 1;
    for (i = 1; i <= n; i++) {
        if (visited[i]) {
            printf("%d\t", i);
        } else {
            printf("\nBFS is not possible. Not all nodes are reachable.\n");
            allNodesReachable = 0;
            break;
        }
    }

    if (allNodesReachable) {
        printf("\n");
    }

    return 0;
}
```

Output:

```
Enter the number of vertices: 7
Enter graph data in matrix form:
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 0 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Enter the starting vertex: 1
The nodes that are reachable are:
1       2       3       4       5       6       7

Process returned 0 (0x0)   execution time : 177.319 s
Press any key to continue.
```

```
Enter the number of vertices: 5
Enter graph data in matrix form:
1 0 0 0 0
0 0 0 1 0
0 1 0 1 0
0 1 1 0 0
0 0 0 1 0
Enter the starting vertex: 1
The nodes that are reachable are:
1
BFS is not possible. Not all nodes are reachable.

Process returned 0 (0x0)   execution time : 44.208 s
Press any key to continue.
```

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

Solution:

Code:

```
#include<stdio.h>
#include<conio.h>
int graph[20][20];
void DFS(int i,int vis[],int n)
{
   int j;
   printf("%d ",i);
   vis[i]=1;
   for(j=0;j<n;j++)
   {
      if(graph[i][j]==1 && vis[j]==0)
      {
         DFS(j,vis,n);
      }
   }
}
int main()
{
   int n,i,j,top=-1;
   printf("Enter the number of vertices:\n");
   scanf("%d",&n);
   printf("Enter the adjacency matrix representing the graph:\n");

   int vis[n],st[n];
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
         scanf("%d",&graph[i][j]);
      }
   }
   for(int i=0;i<n;i++)
   {
      vis[i]=0;
   }
```

```
    DFS(0,vis,n);
    return 0;
}
```

Output:

```
Enter the number of vertices:
4
Enter the adjacency matrix representing the graph:
0 1 0 0
0 0 1 0
0 1 0 1
1 1 0 0
0 1 2 3
Process returned 0 (0x0)   execution time : 38.994 s
Press any key to continue.
```

```
Enter the number of vertices:
5
Enter the adjacency matrix representing the graph:
0 1 0 1 0
1 0 0 0 0
1 1 0 0 0
0 1 0 0 1
1 1 0 0 0
0 1 3 4
Process returned 0 (0x0)   execution time : 32.151 s
Press any key to continue.
```

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

Solution:

Code:

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

#define MAX_VERTICES 100

typedef struct Node {
   int data;
   struct Node* next;
} Node;

typedef struct Graph {
   Node* adjList[MAX_VERTICES];
   int numVertices;
} Graph;

Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

Graph* createGraph(int numVertices) {
   Graph* graph = (Graph*)malloc(sizeof(Graph));
```

```
    graph->numVertices = numVertices;

    for (int i = 0; i < numVertices; i++) {
        graph->adjList[i] = NULL;
    }

    return graph;
}

void addEdge(Graph* graph, int src, int dest) {
    Node* newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;
}

void topologicalSortUtil(Graph* graph, int v, int visited[], int* stackIndex, int stack[]) {
    visited[v] = 1;

    Node* node = graph->adjList[v];
    while (node != NULL) {
        int adjVertex = node->data;
        if (!visited[adjVertex]) {
            topologicalSortUtil(graph, adjVertex, visited, stackIndex, stack);
        }
        node = node->next;
    }

    stack[(*stackIndex)] = v;
    (*stackIndex)++;
```

```
}

void topologicalSort(Graph* graph) {
    int visited[MAX_VERTICES] = {0};
    int stack[MAX_VERTICES];
    int stackIndex = 0;

    for (int i = 0; i < graph->numVertices; i++) {
        if (!visited[i]) {
            topologicalSortUtil(graph, i, visited, &stackIndex, stack);
        }
    }

    printf("Topological Sort: ");
    for (int i = stackIndex - 1; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
}

int main() {
    int numVertices = 6;
    Graph* graph = createGraph(numVertices);

    addEdge(graph, 5, 2);
    addEdge(graph, 5, 0);
    addEdge(graph, 4, 0);
    addEdge(graph, 4, 1);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 1);
```

```
    topologicalSort(graph);

    return 0;
}
```

Output:

```
Topological Sort: 5 4 2 3 1 0
Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

```
Topological Sort: 5 2 0 3 4 1
Process returned 0 (0x0)   execution time : 0.111 s
Press any key to continue.
```

3. Implement Johnson Trotter algorithm to generate permutations.

Solution:

Code:

```
#include <stdio.h>
void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void display(int *array, int n) {
   for (int i = 0; i < n; i++) {
      printf("%d ", array[i]);
   }
   printf("\n");
}

void johnsonT(int *array, int *directions, int n) {
   display(array, n);

   int mobileElement, mobileIndex;

   while (1) {
      mobileElement = 0;
      mobileIndex = 0;

      for (int i = 0; i < n; i++) {
         if ((i == 0 && directions[i] == -1) ||
            (i == n - 1 && directions[i] == 1)) {
            continue;
         }

         if ((array[i] > array[i + directions[i]]) &&
            (array[i] > mobileElement)) {
```

```
            mobileElement = array[i];
            mobileIndex = i;
         }
      }

      if (mobileElement == 0) {
         break;
      }

      int neighborIndex = mobileIndex + directions[mobileIndex];
      swap(&array[mobileIndex], &array[neighborIndex]);
      swap(&directions[mobileIndex], &directions[neighborIndex]);

      for (int i = 0; i < n; i++) {
         if (array[i] > mobileElement) {
            directions[i] = -directions[i];
         }
      }

      display(array, n);
   }
}

int main() {
   int n;

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

   int array[n];
   int directions[n];

   for (int i = 0; i < n; i++) {
      array[i] = i + 1;
      directions[i] = -1;
   }

   johnsonT(array, directions, n);

   return 0;
```

}

Output:

```
Enter the n of the array: 3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

Process returned 0 (0x0)   execution time : 7.598 s
Press any key to continue.
```

```
Enter the n of the array: 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

Process returned 0 (0x0)   execution time : 1.656 s
Press any key to continue.
```

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.

Solution:

Code:

```
#include <stdio.h>

void merge(int arr[], int left[], int left_size, int right[], int right_size) {
   int i = 0, j = 0, k = 0;

   while (i < left_size && j < right_size) {
      if (left[i] < right[j])
         arr[k++] = left[i++];
      else
         arr[k++] = right[j++];
   }

   while (i < left_size)
      arr[k++] = left[i++];

   while (j < right_size)
      arr[k++] = right[j++];
}

void merge_sort(int arr[], int size) {
   if (size <= 1)
      return;

   int mid = size / 2;
   int left[mid];
   int right[size - mid];

   for (int i = 0; i < mid; i++)
      left[i] = arr[i];

   for (int i = mid; i < size; i++)
      right[i - mid] = arr[i];
```

```
    merge_sort(left, mid);
    merge_sort(right, size - mid);
    merge(arr, left, mid, right, size - mid);
}

int main() {
    int arr[] = {13, 11, 13, 5, 6, 7};
    int size = sizeof(arr) / sizeof(arr[0]);

    merge_sort(arr, size);

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

    return 0;
}
```

Output:

```
Merge Sort
Sorted array: 5 6 7 11 13 13
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
```

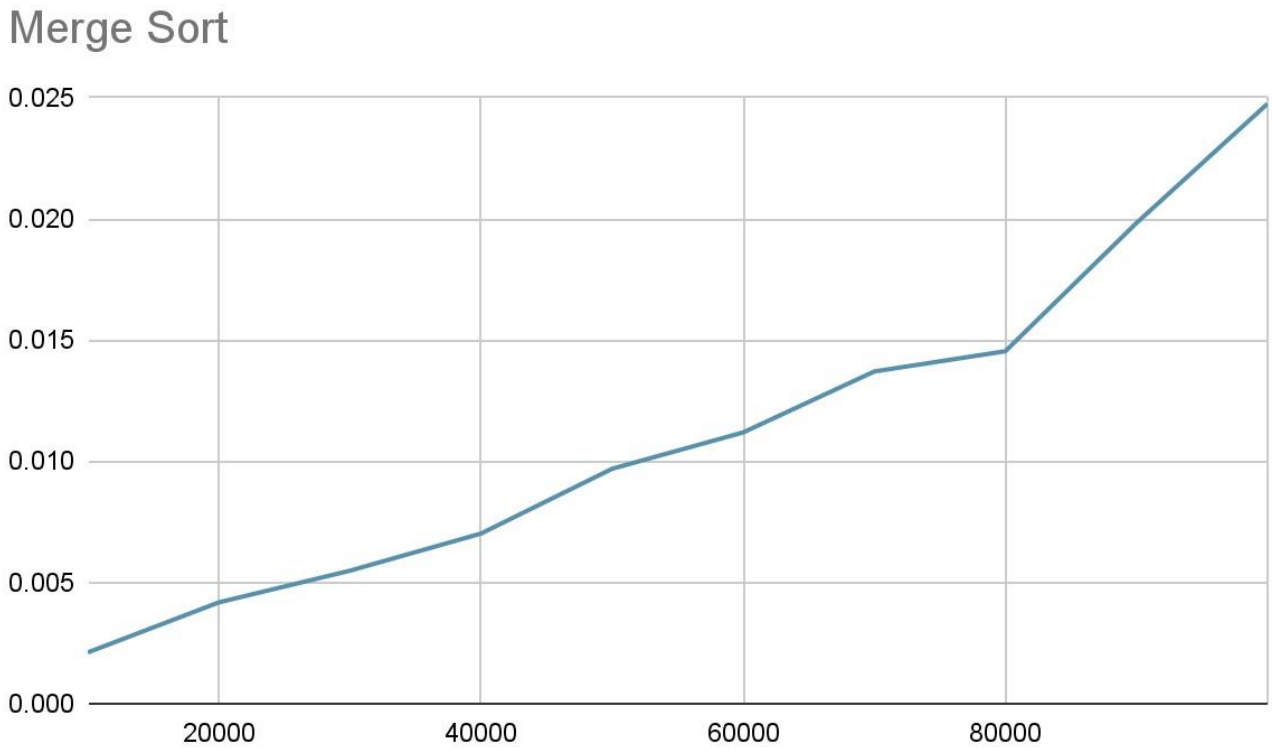
```
Merge Sort
Sorted array: 1 23 24 25 65 345 556
Process returned 0 (0x0)   execution time : 0.073 s
Press any key to continue.
```

Table of values:

Input size(n)	Time taken
10000	0.002114
20000	0.00418

30000	0.005486
40000	0.007019
50000	0.00969
60000	0.011191
70000	0.013704
80000	0.014539
90000	0.019828
100000	0.024749

Graph:



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

Solution:

Code:

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

int partition(int low, int high, int* a) {
   int i = low;
   int j = high + 1;
   int pivot = a[low];
   int temp;

   while (1) {
      do {
         i = i + 1;
      } while (pivot >= a[i]);

      do {
         j = j - 1;
      } while (pivot < a[j]);

      if (i >= j)
         break;

      temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   temp = a[low];
   a[low] = a[j];
   a[j] = temp;
```

```
    return j;
}

void Quicksort(int low, int high, int* a) {
    int j;

    if (low < high) {
        j = partition(low, high, a);
        Quicksort(low, j - 1, a);
        Quicksort(j + 1, high, a);
    }
}

int main() {
    int arr[20], n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    Quicksort(0, n - 1, arr);

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

    return 0;
}
```

Output:

```
Enter the number of elements: 5
Enter the elements: 23 434 2 53 25
Sorted array: 2 23 25 53 434

Process returned 0 (0x0)   execution time : 20.820 s
Press any key to continue.
```

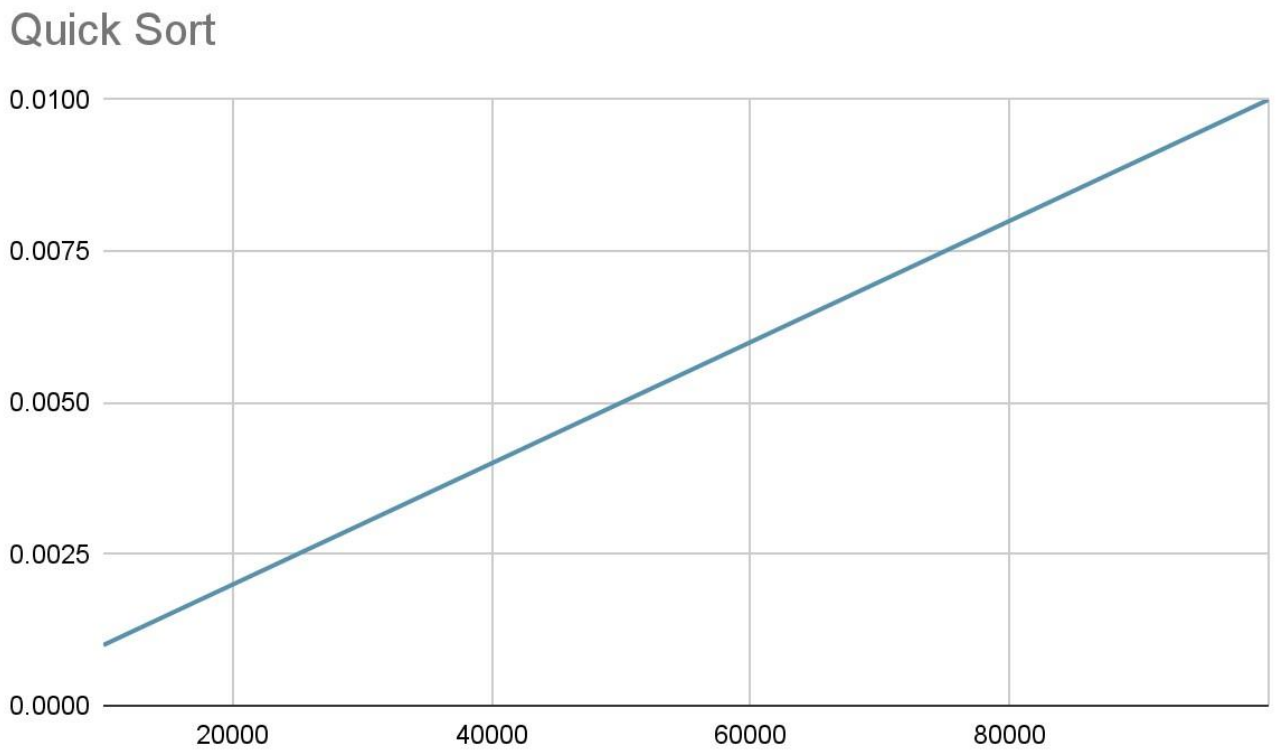
```
Enter the number of elements: 9
Enter the elements: 2 4 6 1 8 5 73 23 98
Sorted array: 1 2 4 5 6 8 23 73 98

Process returned 0 (0x0)   execution time : 17.828 s
Press any key to continue.
```

Table of values:

Input size(n)	Time taken
10000	0.001
20000	0.002
30000	0.003
40000	0.004
50000	0.005
60000	0.006
70000	0.007

80000	0.008
90000	0.009
100000	0.010

Graph:



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

Solution:

Code:

```
#include<stdio.h>

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

   // Extract elements from the heap one by one
   for (int i = n - 1; i > 0; i--)
   {
      // Move the current root (maximum value) to the end
      swap(&arr[0], &arr[i]);
      // Call max heapify on the reduced heap
      heapify(i, arr, 0);
   }
}

void heapify(int n, int arr[], 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)
   {
      swap(&arr[i], &arr[largest]);
```

```
      heapify(n, arr, largest);
   }
}

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

void main()
{
   int n;
   printf("HEAP SORT \n ");
   printf("\nEnter the number of elements to be sorted: ");
   scanf("%d", &n);
   int arr[n];
   printf("\nEnter the elements: ");
   for (int i = 0; i < n; i++)
      scanf("%d", &arr[i]);

   heapsort(n, arr);

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

Output:

```
HEAP SORT

Enter the number of elements to be sorted: 5

Enter the elements: 6 23 7 523 8

Sorted array in ascending order:
 6 7 8 23 523
Process returned 5 (0x5)   execution time : 11.386 s
Press any key to continue.
```

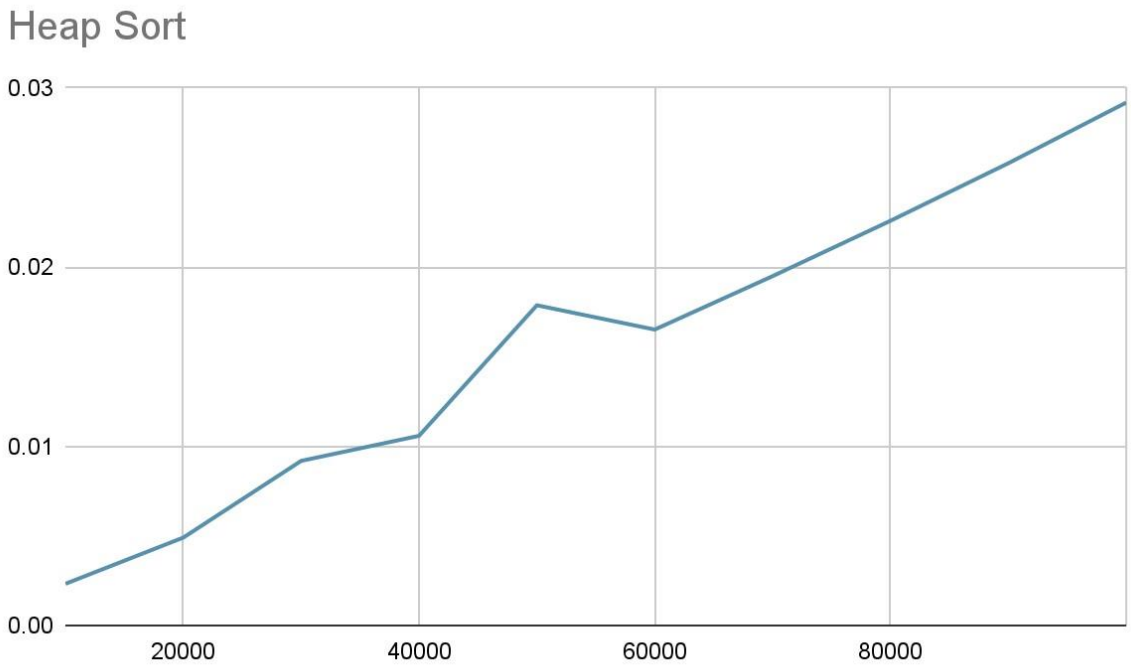
```
HEAP SORT

Enter the number of elements to be sorted: 8

Enter the elements: 5 26 18 32 43 243 54 24

Sorted array in ascending order:
 5 18 24 26 32 43 54 243
Process returned 8 (0x8)   execution time : 20.375 s
Press any key to continue.
```

Table of values:

Input size(n)	Time taken
10000	0.002324
20000	0.004903
30000	0.009185
40000	0.010584
50000	0.017871
60000	0.016515

70000	0.019496
80000	0.022587
90000	0.025799
100000	0.029185

Graph:



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

Solution:

Code:

```
#include<stdio.h>
int max(int a, int b) {
   return (a > b) ? a : b;
}
int knapsack(int W,int weights[],int values[],int n)
{
   int matrix[n+1][W+1];
   int i,j;
   for(i=0; i<n+1; i++)
   {
      for(j=0; j<W+1; j++)
      {
         if(i==0 || j==0) matrix[i][j]=0;
         else if(weights[i]>j) matrix[i][j]=matrix[i-1][j];
         else {
            matrix[i][j]= max(matrix[i-1][j], matrix[i-1][j-weights[i]]+values[i]);
         }
      }
   }
   return matrix[n][W];
}

int main() {
   int W, n;

   printf("Enter the capacity of the knapsack: ");
   scanf("%d", &W);

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

   int weights[n];
   int values[n];
```

```
    printf("Enter the weight and value of each item:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &weights[i], &values[i]);
    }

    int max_value = knapsack(W, weights, values, n);
    printf("Maximum value in the knapsack: %d\n", max_value);

    return 0;
}
```

Output:

```
Enter the capacity of the knapsack: 5
Enter the number of items: 4
Enter the weight and value of each item:
2 12
1
10
3 20
2 15
Maximum value in the knapsack: 35

Process returned 0 (0x0)   execution time : 31.309 s
Press any key to continue.
```

```
Enter the capacity of the knapsack: 7
Enter the number of items: 4
Enter the weight and value of each item:
1 8
2 6
3 12
4 20
Maximum value in the knapsack: 32

Process returned 0 (0x0)   execution time : 30.041 s
Press any key to continue.
```

8.Implement All Pair Shortest paths problem using Floyd’s algorithm.

Solution:

Code:

```
#include <stdio.h>

void main() {
   int adj[10][10], n, i, j, k;
   int result[10][10];

   printf("Floyd's algorithm\n");
   printf("Enter the number of vertices\n");
   scanf("%d", &n);

   printf("Enter the distance matrix for %d vertices\n", n);
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &adj[i][j]);
         result[i][j] = adj[i][j];
      }
   }

   for (k = 0; k < n; k++) {
      for (j = 0; j < n; j++) {
         for (i = 0; i < n; i++) {
            result[i][j] = result[i][j] < (result[i][k] + result[k][j]) ? result[i][j] : (result[i][k] + result[k][j]);
         }
      }
   }

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

```
  }
}
```

Output:

```
Floyd's algorithm
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Result
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

```
Floyd's algorithm
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Result
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

9.Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

Prim's algorithm:

Code:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5

int minKey(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;
}
int printMST(int parent[], int graph[V][V])
{
printf("Edge \tWeight\n");
for (int i = 1; i < V; i++)
printf("%d - %d \t%d \n", parent[i], i,
graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{
int parent[V];
int key[V];
bool mstSet[V];
for (int i = 0; i < V; i++)
key[i] = INT_MAX, mstSet[i] = false;
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
int u = minKey(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];
}
printMST(parent, graph);
}


int main()
{
   int v;
   printf("enter the no of vertices :");
   scanf("%d",&v);
   int graph[V][V];
   printf("enter the adjacency matrix\n");
   for(int i=0;i<v;i++)
      for(int j=0;j<v;j++)
         scanf("%d",&graph[i][j]);
primMST(graph);

return 0;
}
```

Output:

```
enter the no of vertices :5
enter the adjacency matrix
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

Process returned 0 (0x0)   execution time : 61.191 s
Press any key to continue.
```

```
enter the no of vertices :5
enter the adjacency matrix
0 1 0 4 0
1 0 3 0 2
0 3 0 5 0
4 0 5 0 6
0 2 0 6 0
Edge    Weight
0 - 1   1
1 - 2   3
0 - 3   4
1 - 4   2

Process returned 0 (0x0)   execution time : 12.989 s
Press any key to continue.
```

Kruskal's algorithm:

Code:

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

int find(int v, int parent[10]) {
   while (parent[v] != v) {
      v = parent[v];
   }
   return v;
}

void union1(int i, int j, int parent[10]) {
   if (i < j)
      parent[j] = i;
   else
      parent[i] = j;
}

void kruskal(int n, int a[10][10]) {
   int count, k, min, sum, i, j, t[10][10], u, v, parent[10];
   count = 0;
   k = 0;
   sum = 0;

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

   while (count != n - 1) {
      min = 999;
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (a[i][j] < min && a[i][j] != 0) {
               min = a[i][j];
               u = i;
               v = j;
            }
         }
      }

      i = find(u, parent);
```

```
      j = find(v, parent);

      if (i != j) {
         union1(i, j, parent);
         t[k][0] = u;
         t[k][1] = v;
         k++;
         count++;
         sum = sum + a[u][v];
      }
      a[u][v] = a[v][u] = 999;
   }

   if (count == n - 1) {
      printf("Spanning tree\n");
      for (i = 0; i < n - 1; i++) {
         printf("%d %d\n", t[i][0], t[i][1]);
      }
      printf("Cost of spanning tree = %d\n", sum);
   } else {
      printf("Spanning tree does not exist\n");
   }
}

void main() {
   int n, i, j, a[10][10];
   clrscr();

   printf("Enter the number of nodes\n");
   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]);

   kruskal(n, a);
   getch();
}
```

Output:

```
enter the number of nodes
5
enter the adjacency matrix
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 2
999 999 999 2 0
spanning tree
0 1
0 3
3 4
2 3
cost of spanning tree=8
```

```
enter the number of nodes
5
enter the adjacency matrix
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 2
999 999 999 2 0
spanning tree
0 1
0 3
3 4
2 3
```

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

Solution:

Code:

```
#include <stdio.h>

#define INFINITY 9999

#define MAX 10



void dijkstra(int G[MAX][MAX], int n, int startnode);

int main() {

   int G[MAX][MAX], i, j, n, u;

   printf("Enter the number of vertices:");

   scanf("%d", &n);

   printf("\nEnter the adjacency matrix:\n");

   for (i = 0; i < n; i++)

      for (j = 0; j < n; j++)

         scanf("%d", &G[i][j]);

   printf("\nEnter the starting node:");

   scanf("%d", &u);

   dijkstra(G, n, u);

   return 0;
```

```
}
void dijkstra(int G[MAX][MAX], int n, int startnode) {
   int cost[MAX][MAX], distance[MAX], pred[MAX];
   int visited[MAX], count, mindistance, nextnode, i, j;
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         cost[i][j] = (G[i][j] == 0) ? INFINITY : G[i][j];
   for (i = 0; i < n; i++) {
      distance[i] = cost[startnode][i];
      pred[i] = startnode;
      visited[i] = 0;
   }
   distance[startnode] = 0;
   visited[startnode] = 1;
   count = 1;
   while (count < n - 1) {
      mindistance = INFINITY;


      for (i = 0; i < n; i++) {
         if (distance[i] < mindistance && !visited[i]) {
            mindistance = distance[i];
```

```
            nextnode = i;
        }
    }
    visited[nextnode] = 1;
    for (i = 0; i < n; i++) {
        if (!visited[i] && mindistance + cost[nextnode][i] < distance[i]) {
            distance[i] = mindistance + cost[nextnode][i];
            pred[i] = nextnode;
        }
    }
    count++;
}
for (i = 0; i < n; i++) {
    if (i != startnode) {
        printf("\nDistance of node %d = %d", i, distance[i]);
        printf("\nPath = %d", i);
        j = i;
        do {
            j = pred[j];
            printf("<-%d", j);
        } while (j != startnode);
```

```
    }
  }
}
```

Output:

```
Enter no. of vertices:6

Enter the adjacency matrix:
0 25 100 35 9999 9999
9999 0 9999 27 14 9999
9999 9999 0 50 9999 48
9999 9999 9999 0 29 9999
9999 9999 9999 9999 0 21
9999 9999 48 9999 9999 0

Enter the starting node:0

Distance of node1 = 25
Path = 1<-0
Distance of node2 = 100
Path = 2<-0
Distance of node3 = 35
Path = 3<-0
Distance of node4 = 39
Path = 4<-1<-0
Distance of node5 = 60
Path = 5<-4<-1<-0
```

11. Implement “N-Queens Problem” using Backtracking.

Solution:

Code:

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

int board[20], count = 0;
void print(int n);
int place(int row, int column);
void queen(int row, int n);
int main() {
   int n, i, j;
   printf(" - N Queens Problem Using Backtracking -");
   while (1) {
      printf("\n\nEnter number of Queens:");
      count = 0;
      scanf("%d", &n);
      if (n <= 3) {
         printf("No solution\n");
      } else {
         queen(1, n);
      }
   }
   return 0;
}
```

```
void print(int n) {
   int i, j;
   printf("\n\nSolution %d:\n\n", ++count);
   for (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)
            printf("\tQ");
         else
            printf("\t-");
      }
   }
}
int place(int row, int column) {
   int i;
   for (i = 1; i <= row - 1; ++i) {
      if (board[i] == column)
         return 0;
      else if (abs(board[i] - column) == abs(i - row))
         return 0;
   }
   return 1;
}
```

```
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 number of Queens:4

Solution 1:
        1       2       3       4
1       -       Q       -       -
2       -       -       -       Q
3       Q       -       -       -
4       -       -       Q       -
Solution 2:
        1       2       3       4
1       -       -       Q       -
2       Q       -       -       -
3       -       -       -       Q
4       -       Q       -       -
```

```
Enter number of Queens:6

Solution 1:
        1       2       3       4       5       6
1       -       Q       -       -       -       -
2       -       -       -       Q       -       -
3       -       -       -       -       -       Q
4       Q       -       -       -       -       -
5       -       -       Q       -       -       -
6       -       -       -       -       Q       -
Solution 2:
        1       2       3       4       5       6
1       -       -       Q       -       -       -
2       -       -       -       -       -       Q
3       -       Q       -       -       -       -
4       -       -       -       -       Q       -
5       Q       -       -       -       -       -
6       -       -       -       Q       -       -
Solution 3:
        1       2       3       4       5       6
1       -       -       -       Q       -       -
2       Q       -       -       -       -       -
3       -       -       -       -       Q       -
4       -       Q       -       -       -       -
5       -       -       -       -       -       Q
6       -       -       Q       -       -       -
Solution 4:
        1       2       3       4       5       6
1       -       -       -       -       Q       -
2       -       -       Q       -       -       -
3       Q       -       -       -       -       -
4       -       -       -       -       -       Q
5       -       -       -       Q       -       -
6       -       Q       -       -       -       -
```