ANALYSIS AND DESIGN OF ALGORITHMS

PRACTICAL FILE

COURSE CODE: CSE303

SUBMITTED TO:
Dr. SUMITA GUPTA

SUBMITTED BY:
V SRI HRUDYA
5CSE6-X
A2305221030

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY
NOIDA

INDEX

S.NO.	OBJECTIVE	ALLOTMENT DATE	SUBMISSION DATE	PAGE NO.	MARKS ALLOTED	SIGN
01						
02						
03						
04						
05						
06						
07						
08						
09						
10						

S.NO	AIM	ALLOTMENT DATE	SUBMISSION DATE	PAGE NO.	SIGN
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

PROGRAM 1

AIM: Write a program to implement LINEAR SEARCH, find its step count and calculate its complexity.

PSUDO CODE:

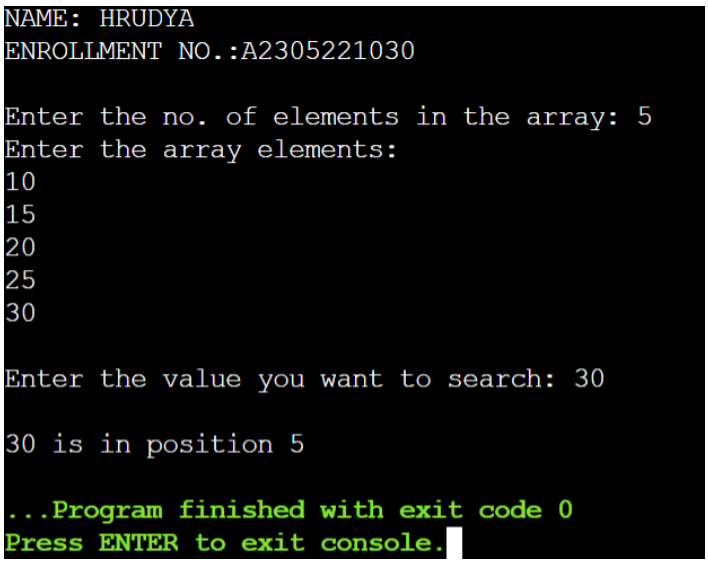
```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for

end procedure
```

CODE:

```
#include <stdio.h>

#define N 100

int main() {
    int arr[N], n, k, i, count=0;

    printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
    printf("Enter the no. of elements in the array: ");
    scanf("%d", &n);

    printf("Enter the array elements: \n");

    for( i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }

    printf("\nEnter the value you want to search: ");
    scanf("%d", &k);

    for(i=0; i<n; i++){
        if (arr[i]==k){
```

```
            count= i+1;
        }
    }
    if(count>=1){
    printf("\n%d is in position %d", k, count);
    }

    else{
        printf("%d does not exist in the array.", k);
    }

}
```

OUTPUT:



COMPLEXITY: O(n)

PROGRAM 2.1

AIM: Write a program to implement BINARY SEARCH (*Without recursion)*, find its step count and calculate its complexity.

PSUDOCODE:

```
Procedure binary_search
  A ← sorted array
  n ← size of array
  x ← value to be searched

  Set lowerBound = 1
  Set upperBound = n

  while x not found
    if upperBound < lowerBound
      EXIT: x does not exists.

    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

    if A[midPoint] < x
      set lowerBound = midPoint + 1

    if A[midPoint] > x
      set upperBound = midPoint - 1

    if A[midPoint] = x
      EXIT: x found at location midPoint
  end while

end procedure
```

CODE:

```
#include <stdio.h>
#define N 100

int binarySearch(int arr[], int l, int r, int x)
{
   while (l <= r)
   {
      int m = l + (r-l)/2;
      if (arr[m] == x){
         return m;
      }
      if (arr[m] < x){
         l = m + 1;
      }
      else{
      r = m - 1;
      }
   }
   return -1;
}


int main(void){
   int i, n, arr[N], x;

   printf("NAME: HRUDYA\nENROLLMENT NO.: A2305221030\n\n");

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

   printf("Enter the array elements: \n");

   for( i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }

   printf("Enter the no you want to search: ");
   scanf("%d", &x);
```

```
    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
    : printf("Element is present at index %d", result+1);
    return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.: A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
10
20
30
40
50
Enter the no you want to search: 20
Element is present at index 2

...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY: O(log(n))

PROGRAM 2.2

AIM: Write a program to implement BINARY SEARCH (*With recursion)*, find its step count and calculate its complexity.

PSUDOCODE:

```
Procedure binary_search
  A ← sorted array
  n ← size of array
  x ← value to be searched

  Set lowerBound = 1
  Set upperBound = n

  while x not found
    if upperBound < lowerBound
      EXIT: x does not exists.

    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

    if A[midPoint] < x
      set lowerBound = midPoint + 1

    if A[midPoint] > x
      set upperBound = midPoint - 1

    if A[midPoint] = x
      EXIT: x found at location midPoint
  end while

end procedure
```

CODE:

```
#include <stdio.h>
#define N 100

int binarySearch(int arr[], int l, int r, int x){
   if (r >= l)
   {
```

```
        int mid = l + (r - l)/2;
        if (arr[mid] == x) return mid;
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);
        return binarySearch(arr, mid+1, r, x);
    }
    return -1;

}

int main(void){
   int i, n, arr[N], x;

    printf("NAME: HRUDYA\nENROLLMENT NO.: A2305221030\n\n");

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

    printf("Enter the array elements: \n");

    for( i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    printf("\nEnter the no you want to search: ");
    scanf("%d", &x);

    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("\nElement is not present in array")
    : printf("\nElement is present at index %d", result+1);
    return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.: A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
10
15
20
25
30

Enter the no you want to search: 25

Element is present at index 4

...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY: O(log(n))

PROGRAM 3

AIM: Write a program to implement QUICK SORT and calculate its complexity.

PSUDO CODE:

```
function quickSort(arr, low, high)
   if low < high
      pivotIndex = partition(arr, low, high)
      quickSort(arr, low, pivotIndex - 1)
      quickSort(arr, pivotIndex + 1, high)

function partition(arr, low, high)
   pivot = arr[high]
   i = low - 1
   for j = low to high - 1
      if arr[j] < pivot
         i = i + 1
         swap arr[i] with arr[j]
   swap arr[i + 1] with arr[high]
   return i + 1
```

CODE:

```
#include <stdio.h>
#define N 100

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);
}
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 arr[N], n, k, i,;

   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the no. of elements in the array: ");
   scanf("%d", &n);

   printf("Enter the array elements: \n");

   for( i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }

        quickSort(arr, 0, n - 1);
        printf("Sorted array: \n");
        for (int i = 0; i < n; i++)
                printf("%d ", arr[i]);
        return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
20
10
40
32
40
Sorted array:
10 20 32 40 40
```

COMPLEXITY: O(nlog(n))

PROGRAM 4

AIM: Write a program to implement MERGE SORT and calculate its complexity.

PSUDOCODE:

```
function mergeSort(arr):
    if length of arr is 1 or less:
        return arr

    middle = length of arr / 2
    left_half = mergeSort(arr[0 : middle])
    right_half = mergeSort(arr[middle : length of arr])

    sorted_arr = merge(left_half, right_half)
    return sorted_arr

function merge(left_arr, right_arr):
    merged_arr = []
    left_index = 0
    right_index = 0

    while left_index < length of left_arr and right_index < length of right_arr:
        if left_arr[left_index] <= right_arr[right_index]:
            add left_arr[left_index] to merged_arr
            left_index++
        else:
            add right_arr[right_index] to merged_arr
            right_index++

    while left_index < length of left_arr:
        add left_arr[left_index] to merged_arr
        left_index++

    while right_index < length of right_arr:
        add right_arr[right_index] to merged_arr
        right_index++

    return merged_arr
```

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define N 100

void merge(int arr[], int l, int m, int r)
{
	int i, j, k;
	int n1 = m - l + 1;
	int n2 = r - m;
	int L[n1], R[n2];
	for (i = 0; i < n1; i++)
		L[i] = arr[l + i];
	for (j = 0; j < n2; j++)
		R[j] = arr[m + 1 + j];
	i = 0;
	j = 0;
	k = l;
	while (i < n1 && j < n2) {
		if (L[i] <= R[j]) {
			arr[k] = L[i];
			i++;
		}
		else {
			arr[k] = R[j];
			j++;
		}
```

```
            k++;
        }
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
}
void mergeSort(int arr[], int l, int r)
{
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
}
}
void printArray(int A[], int n)
{       int i;
        for (i = 0; i < n; i++)
            printf("%d ", A[i]);
        printf("\n");}
```

```
int main(){
int arr[N], n, k, i;
   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the no. of elements in the array: ");
   scanf("%d", &n);
   printf("Enter the array elements: \n");
   for( i=0; i<n; i++){
      scanf("%d", &arr[i]);   }
        mergeSort(arr, 0, n-1);
        printf("\nSorted array: \n");
        printArray(arr, n);
        return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
12
13
14
15
16

Sorted array:
12 13 14 15 16


...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY: O(nlog(n))

PROGRAM 5

AIM: Write a program to implement SELECTION SORT and calculate its complexity.

PSUDO CODE:

```
procedure selection sort
  list  : array of items
  n     : size of list

  for i = 1 to n - 1
    min = i

    for j = i+1 to n
      if list[j] < list[min] then
        min = j;
      end if
    end for

    if indexMin != i  then
      swap list[min] and list[i]
    end if
  end for

end procedure
```

CODE:

```
#include <stdio.h>
#define N 100

void swap(int *xp, int *yp)
{
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}

void Sort(int arr[], int n)
{
```

```
	int i, j, min_ind;
	for (i = 0; i < n-1; i++)
	{
		min_ind = i;
		for (j = i+1; j < n; j++)
		if (arr[j] < arr[min_ind])
			min_ind = j;

		if(min_ind != i)
			swap(&arr[min_ind], &arr[i]);
	}
}

int main()
{
	int arr[N], n, k, i;

   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the no. of elements in the array: ");
   scanf("%d", &n);

   printf("Enter the array elements: \n");

   for( i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }

	Sort(arr, n);
	printf("\nSorted array: ");

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

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
57
34
59
34
25

Sorted array:   25      34      34      57      59

...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY: $O(n^2)$

PROGRAM 6

AIM: Write a program to implement BUBBLE SORT and calculate its complexity.

PSUDOCODE:

```
procedure bubbleSort( list : array of items )

  loop = list.count;

  for i = 0 to loop-1 do:
    swapped = false

    for j = 0 to loop-1 do:

      if list[j] > list[j+1] then
        /* swap them */
        swap( list[j], list[j+1] )
        swapped = true
      end if

    end for

    if(not swapped) then
      break
    end if

  end for

end procedure return list
```

CODE:

```
#include <stdio.h>

#define N 100

void swap(int *xp, int *yp)

{
```

```
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}
void Sort(int arr[], int n)
{
        int i, j;
        for (i = 0; i < n; i++)
        {
         for(j=0; j<n-1; j++){
            if(arr[j]>arr[j+1]){
                swap(&arr[j], &arr[j+1]);
            }
         }
        }
}

int main()
{
        int arr[N], n, k, i;

   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the no. of elements in the array: ");
   scanf("%d", &n);
   printf("Enter the array elements: \n");

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

```
        scanf("%d", &arr[i]);
    }
        Sort(arr, n);
        printf("\nSorted array: ");
        for (i=0; i < n; i++)
                printf("\t%d", arr[i]);
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the no. of elements in the array: 5
Enter the array elements:
47
38
49
23
1

Sorted array:   1       23      38      47      49

...Program finished with exit code 0
Press ENTER to exit console.
```

COMPLEXITY: $O(n^2)$

PROGRAM 7

AIM: Write a program to implement INSERTION SORT and calculate its complexity.

PSUDO CODE:

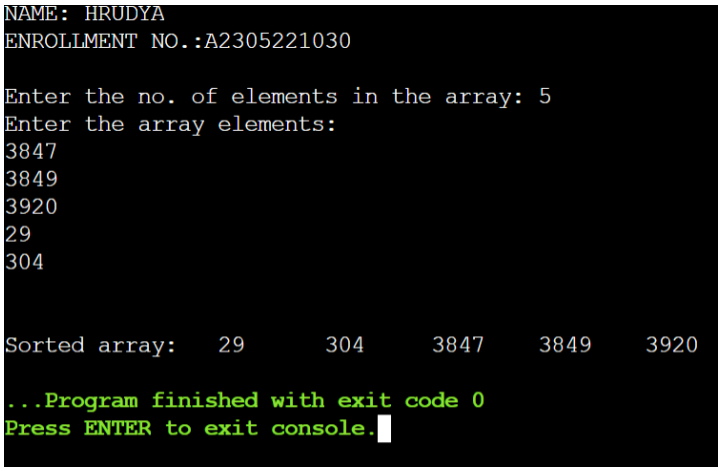
```
InsertionSort(A):
    for i = 1 to length(A) - 1 do
        key = A[i]
        j = i - 1
        while j >= 0 and A[j] > key do
            A[j + 1] = A[j]
            j = j - 1
        A[j + 1] = key
```

CODE:

```
#include <math.h>
#include <stdio.h>
#define N 100
void insertionSort(int arr[], int n)
{
        int i, key, j;
        for (i = 1; i < n; i++)
        {
                key = arr[i];
                j = i - 1;
                while (j >= 0 && arr[j] > key)
                {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
}
```

```
void printArray(int arr[], int n)
{
        int i;
        for (i = 0; i < n; i++)
                printf("\t%d", arr[i]);
}

int main()
{
   int arr[N], n, k, i;
   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the no. of elements in the array: ");
   scanf("%d", &n);

   printf("Enter the array elements: \n");

   for( i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }
        insertionSort(arr, n);
        printf("\n\nSorted array: ");
        printArray(arr, n);
        return 0;
}
```

OUTPUT:



COMPLEXITY: $O(n^2)$

PROGRAM 8

AIM: Write a program to implement FRACTIONAL KNAPSACK.

PSUDO CODE:

```
Fractional Knapsack (Array W, Array V, int M)
 for i <- 1 to size (V)
          calculate cost[i] <- V[i] / W[i]
 Sort-Descending (cost)
 i ← 1
 while (i <= size(V))
          if  W[i] <= M
                    M ← M – W[i]
                    total ← total + V[i];
          if  W[i] > M
                i ← i+1
```

CODE:

```
#include <stdio.h>
void main()
{
   int cap, num, cur_weight, item, i, w[10], value[10], used[10];
   float total_profit;
   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Enter the capacity of knapsack: ");
   scanf("%d", &cap);
   printf("Enter the number of items: ");
   scanf("%d", &num);
   printf("Enter the weight and value of %d item: ", num);
   for (i = 0; i<num; i++)
   {printf("\nWeight[%d]: ", i+1);
      scanf("%d", &w[i]);
      printf("Value[%d]: ", i+1);
      scanf("%d", &value[i]); }
   for (i = 0; i < num; ++i)
   used[i]= 0;
   cur_weight = cap;
   while (cur_weight > 0)
   {item = -1;
```

```
      for (i = 0; i < num; ++i)
         if ((used[i] == 0) &&
            ((item == -1) || ((float) value[i] / w[i] > (float) value[item] / w[item])))
            item = i;
      used[item] = 1;
      cur_weight -= w[item];
      total_profit += value[item];
      if (cur_weight >= 0)
         printf("\nADDED OBJECT %d (%d, %dKg) COMPLETELY. \nSPACE
LEFT: %d.", item + 1, value[item], w[item], cur_weight);
      else
      {int item_percent = (int) ((1 + (float) cur_weight / w[item]) * 100);
         printf("\nADDED %d%% (%d, %dKg) OF OBJECT %d.",
item_percent, value[item], w[item], item + 1);
         total_profit -= value[item];
         total_profit += (1 + (float)cur_weight / w[item]) * value[item];
      } }
   printf("\n\nTOTAL PROFIT %.2f.", total_profit);
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the capacity of knapsack: 60
Enter the number of items: 4
Enter the weight and value of 4 item:
Weight[1]: 40
Value[1]: 280

Weight[2]: 10
Value[2]: 100

Weight[3]: 20
Value[3]: 120

Weight[4]: 24
Value[4]: 120

ADDED OBJECT 2 (100, 10Kg) COMPLETELY.
SPACE LEFT: 50.
ADDED OBJECT 1 (280, 40Kg) COMPLETELY.
SPACE LEFT: 10.
ADDED 50% (120, 20Kg) OF OBJECT 3.

TOTAL PROFIT 440.00.
```

PROGRAM 9

AIM: Write a program to implement KRUSKAL'S ALGORITHM.

PSUDO CODE:

KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
   MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
   if FIND-SET(u) ≠ FIND-SET(v):
   A = A ∪ {(u, v)}
   UNION(u, v)
return A

CODE:

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

struct Edge {
   int src, dest, weight;
};

struct Subset {
   int parent, rank;
};

int find(struct Subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}

void unionSets(struct Subset subsets[], int x, int y) {
   int xroot = find(subsets, x);
   int yroot = find(subsets, y);

   if (subsets[xroot].rank < subsets[yroot].rank)
      subsets[xroot].parent = yroot;
```

```
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int compare(const void* a, const void* b) {
    return ((struct Edge*)a)->weight - ((struct Edge*)b)->weight;
}

void kruskalMST(int V, int E, struct Edge edges[]) {
    int cost=0;
    qsort(edges, E, sizeof(edges[0]), compare);

    struct Subset* subsets = (struct Subset*)malloc(V * sizeof(struct Subset));
    for (int v = 0; v < V; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    printf("Minimum Spanning Tree:\n");
    for (int i = 0, e = 0; e < V - 1 && i < E; i++) {
        struct Edge nextEdge = edges[i];
        int x = find(subsets, nextEdge.src);
        int y = find(subsets, nextEdge.dest);

        if (x != y) {
            unionSets(subsets, x, y);
            printf("%d - %d : %d\n", nextEdge.src, nextEdge.dest,
nextEdge.weight);
            e++;
            cost= cost +nextEdge.weight;
        }
    }

    free(subsets);
    printf("Cost of MST: %d", cost);
}

int main() {
    int V, E;
```

```
    printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
     printf("Enter the number of edges: ");
    scanf("%d", &E);

    struct Edge edges[E];
    for (int i = 0; i < E; i++) {
        printf("Enter edge %d (Source Destination Weight): \n", i + 1);
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }

    kruskalMST(V, E, edges);

    return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the number of vertices: 4
Enter the number of edges: 4

Enter edge 1 (Source Destination Weight): 1 2 3

Enter edge 2 (Source Destination Weight): 2 3 1

Enter edge 3 (Source Destination Weight): 3 4 3

Enter edge 4 (Source Destination Weight): 4 1 2
Minimum Spanning Tree:
2 - 3 : 1
4 - 1 : 2
1 - 2 : 3
Cost of MST: 6

...Program finished with exit code 0
Press ENTER to exit console.
```

PROGRAM 10

AIM: Write a program to implement PRIM'S ALGORITHM.

PSUDO CODE:

MST-PRIM(G, w, r)

```
for each u  V [G]    do key[u] ← ∞
π[u] ← NIL
key[r] ← 0
Q ← V [G]
while Q ≠ Ø
do u ← EXTRACT-MIN(Q)
        for each v  Adj[u]
        do if v  Q and w(u, v) < key[v]
                then π[v] ← u
                key[v] ← w(u, v)
```

CODE:

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

int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()

{
   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\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;
                } } }
```

```
printf("\nMST: ");
visited[1]=1;
printf("\n");
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("Edge %d:(%d %d) cost:%d\n",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;}
        cost[a][b]=cost[b][a]=999;}
printf("\nMinimun cost= %d",mincost);
getch();}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the number of nodes: 4
Enter the adjacency matrix:
0 3 0 0
0 0 1 0
0 0 0 3
2 0 0 0

MST:
Edge 1:(1 2) cost:3
Edge 2:(2 3) cost:1
Edge 3:(3 4) cost:3

Minimun cost= 7

...Program finished with exit code 255
Press ENTER to exit console.
```

PROGRAM 11

AIM: Write a program to implement DIJKASTRA'S ALGORITHM .

PSUDO CODE:

dist[S] ← 0

Π[S] ← NIL

for all v ∈ V - {S}

do dist[v] ← ∞

Π[v] ← NIL

S ← ∅

Q ← V

while Q ≠ ∅

do u ← mindistance (Q, dist)

S ← S ∪ {u}

for all v ∈ neighbors[u]

do if dist[v] > dist[u] + w(u,v)

then dist[v] ← dist[u] + w(u,v)

return dist

CODE:

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#define MAX_VERTICES 100
int minDistance(int dist[], bool sptSet[], int vertices)
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < vertices; v++) {
      if (!sptSet[v] && dist[v] < min) {
         min = dist[v];
         min_index = v;
      } }
```

```
        return min_index;
}

void printPathAndDistance(int parent[], int target) {
   if (parent[target] == -1) {
      printf("%d ", target+1);
      return; }
      printPathAndDistance(parent, parent[target]);
      printf("-> %d ", target+1);
}
void printSolution(int dist[], int parent[], int src, int vertices) {

printf("\n_____
_____\n");
   printf("\nTO VERTEX      COST                    PATH\n");
   for (int v = 0; v < vertices; v++) {
      printf("%d \t\t %d \t\t\t ", v+1, dist[v]);
      printPathAndDistance(parent, v);
      printf("\n");
   }
}

void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int
vertices) {
   int dist[MAX_VERTICES];
   bool sptSet[MAX_VERTICES];
   int parent[MAX_VERTICES];
   for (int i = 0; i < vertices; i++) {
      dist[i] = INT_MAX;
      sptSet[i] = false;
      parent[i] = -1;
   }
   dist[src] = 0;
   for (int count = 0; count < vertices - 1; count++) {
      int u = minDistance(dist, sptSet, vertices);
      sptSet[u] = true;
      for (int v = 0; v < vertices; v++) {
      if (!sptSet[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, vertices);
```

```
}
int main() {
   int vertices;
      printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   printf("Number of vertices: ");
   scanf("%d", &vertices);
   int graph[MAX_VERTICES][MAX_VERTICES];
   printf("\nAdjacency matrix:\n");
   for (int i = 0; i < vertices; i++) {
      for (int j = 0; j < vertices; j++) {
         scanf("%d", &graph[i][j]);
   } }
   int source, sourcek;
   printf("\nSource vertex: ");
   scanf("%d", &sourcek);
   source= sourcek-1;
   dijkstra(graph, source, vertices);
   return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Number of vertices: 5

Adjacency matrix:
0 3 1 0 0
3 0 1 3 1
1 1 0 0 4
0 3 0 0 2
0 1 4 2 0

Source vertex: 1

TO VERTEX       COST                    PATH
1                0                     1
2                2                     1 -> 3 -> 2
3                1                     1 -> 3
4                5                     1 -> 3 -> 2 -> 4
5                3                     1 -> 3 -> 2 -> 5
```

PROGRAM 12

AIM: Write a program to implement Strassen's Matrix Multiplication and calculate its complexity.

PSEUDO CODE:

Algorithm: Matrix-Multiplication (X, Y, Z)

```
for i = 1 to p do
  for j = 1 to r do
    Z[i,j] := 0
    for k = 1 to q do
      Z[i,j] := Z[i,j] + X[i,k] × Y[k,j]
```

CODE:

```
#include<stdio.h>
int main(){
   printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
   int a[2][2],b[2][2],c[2][2],i,j;
   int m1,m2,m3,m4,m5,m6,m7;
   printf("Enter the 4 elements of first matrix:\n");
   for(i=0;i<2;i++){
     for(j=0;j<2;j++){
     printf("A[%d][%d]: ", i+1, j+1);
     scanf("%d",&a[i][j]);
     }}
  printf("Enter the 4 elements of second matrix:\n");
  for(i=0;i<2;i++){
     for(j=0;j<2;j++){
     printf("B[%d][%d]: ", i+1, j+1);
         scanf("%d",&b[i][j]);
```

```
    }}
printf("\n_____
__\n");
  printf("FIRST MATRIX:  \n");
  for(i=0;i<2;i++){
     printf("\n");
     for(j=0;j<2;j++)
         printf("%d\t",a[i][j]);
  }
printf("\n_____
__\n");
  printf("SECOND MATRIX: \n");
  for(i=0;i<2;i++){
     printf("\n");
     for(j=0;j<2;j++)
         printf("%d\t",b[i][j]);
  }
  m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
  m2= (a[1][0]+a[1][1])*b[0][0];
  m3= a[0][0]*(b[0][1]-b[1][1]);
  m4= a[1][1]*(b[1][0]-b[0][0]);
  m5= (a[0][0]+a[0][1])*b[1][1];
  m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
  m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
  c[0][0]=m1+m4-m5+m7;
  c[0][1]=m3+m5;
  c[1][0]=m2+m4;
  c[1][1]=m1-m2+m3+m6;
printf("\n_____
```

```
__\n");
    printf("RESULT MATRIX:\n");
    for(i=0;i<2;i++){
      printf("\n");
      for(j=0;j<2;j++)
          printf("%d\t",c[i][j]);
    }
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the 4 elements of first matrix:
A[1][1]: 1
A[1][2]: 2
A[2][1]: 3
A[2][2]: 1
Enter the 4 elements of second matrix:
B[1][1]: 2
B[1][2]: 1
B[2][1]: 3
B[2][2]: 1


FIRST MATRIX:

1       2
3       1

SECOND MATRIX:

2       1
3       1

RESULT MATRIX:

8       3
9       4
```

COMPLEXITY: O(n^3).

PROGRAM 13

AIM: Write a program to implement Longest Common Subsequence Problem using Dynamic Programming and calculate its complexity.

PSEUDO CODE:

```
LCS-Length(X, Y){
    1. m <- length[X]
    2. n <- length[Y]
    3. for i <- 1 to m
    4. c[i,0] <- 0
    5. for j <- 1 to n
    6. c[0,j] <- 0
    7. for i <- 1 to m
    8. for j <- 1 to n
    9. if (x_i == y_j) {
   10.c[i,j] <- c[i-1,j-1] + 1
   11.b[i,j] <- NW}
   12.else if (c[i-1,j] >= c[i,j-1]) {
   13.c[i,j] <- c[i-1,j]
   14.b[i,j] <- N}
   15.else {
   16.c[i,j] <- c[i,j-1]
   17.b[i,j] <- W
       }
       }
```

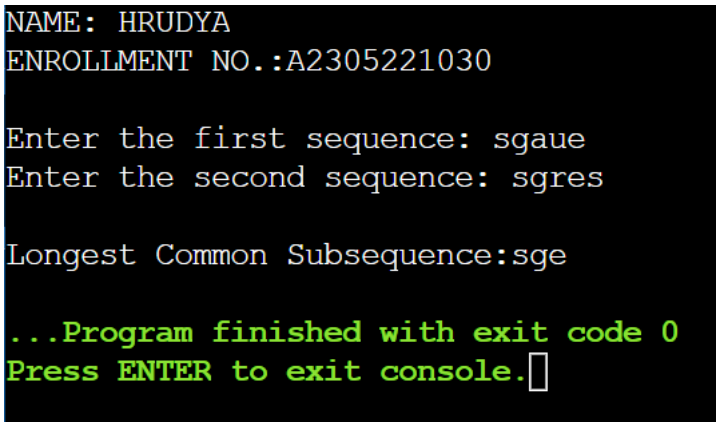
CODE:

```
#include<stdio.h>
#include<string.h>
 int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];
 void print(int i,int j)
{if(i==0 || j==0)
   return;
   if(b[i][j]=='c')
   {
      print(i-1,j-1);
      printf("%c",x[i-1]);
   }
   else if(b[i][j]=='u')
   print(i-1,j);
   else
   print(i,j-1);
}
 void lcs()
{m=strlen(x);
   n=strlen(y);
   for(i=0;i<=m;i++)
   c[i][0]=0;
   for(i=0;i<=n;i++)
   c[0][i]=0
```

```
    for(i=1;i<=m;i++)
    for(j=1;j<=n;j++)
    {
       if(x[i-1]==y[j-1])
       {
          c[i][j]=c[i-1][j-1]+1;
          b[i][j]='c';
       }
    else if(c[i-1][j]>=c[i][j-1])
    {
       c[i][j]=c[i-1][j];
       b[i][j]='u';
    }
    else
    {c[i][j]=c[i][j-1];
       b[i][j]='l';
    }}
}
int main()
{printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
    printf("Enter the first sequence: ");
    scanf("%s",x);
    printf("Enter the second sequence: ");
    scanf("%s",y);
    printf("\nLongest Common Subsequence:");
```

```
    lcs();
    print(m,n);
}
```

OUTPUT:



COMPLEXITY: O(mxn).

m and n are the lengths of the strings.

PROGRAM 14

AIM: Write a program to implement 0/1 Knapsack using Dynamic programming.

PSEUDO CODE:

```
dp[N+1][W+1][F+1] // memo table, initially filled with -1
 int solve(n,w,f)
{
   if(n > N)return 0;
   if(dp[n][w][f] != -1) return dp[n][w][f];
   dp[n][w][f] = solve(n+1,w,f); //skip item
   if(w + weight(n) <= W && f + isFragile(n) <=F)
   dp[n][w][f] = max(dp[n][w][f], value(n) + solve(n+1, w + weight(n), f + isFragile(n)));
   return dp[n][w][f]
}
print(solve(1,0,0))
```

CODE:

```
 #include <stdio.h>
 #define N 100
 int max(int a, int b) { return (a > b)? a : b; }
 int knapsack(int W, int wt[], int prof[], int n)
 {
    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(prof[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
            }
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
int main(){
    int prof[N], wt[N], n, i, W;
    printf("NAME: HRUDYA\nENROLLMENT NO.:A2305221030\n\n");
    printf("Enter the max capacity of Knapsack: ");
    scanf("%d", &W);
    printf("Enter the no of items: ");
    scanf("%d", &n);
    printf("Enter the profit and weight of item[P W]: \n");
    for(i=0; i<n; i++){
      printf("Item %d: ", i+1);
      scanf("%d %d", &prof[i], &wt[i]);
    }
    printf("\nTotal Profit = %d", knapsack(W, wt, prof, n));
    return 0;
}
```

OUTPUT:

```
NAME: HRUDYA
ENROLLMENT NO.:A2305221030

Enter the max capacity of Knapsack: 6
Enter the no of items: 3
Enter the profit and weight of item[P W]:
Item 1: 10 1
Item 2: 12 2
Item 3: 25 4

Total Profit = 37

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT-15

AIM Implementation of breadth first search by using c/c++ and write it's complexity.

Pseudo Code:

```
BFS(G, s)
  for each vertex u ∈ G.V -
    {s} u.color = WHITE
    u.d = ∞ u.π =
    NIL
  s.color = GRAY
  s.d = 0 s.π =
  NIL Q = ∅
  ENQUEUE(Q, s)
  while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u] if
      v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1 v.π
        = u
        ENQUEUE(Q, v)
    u.color = BLACK
```

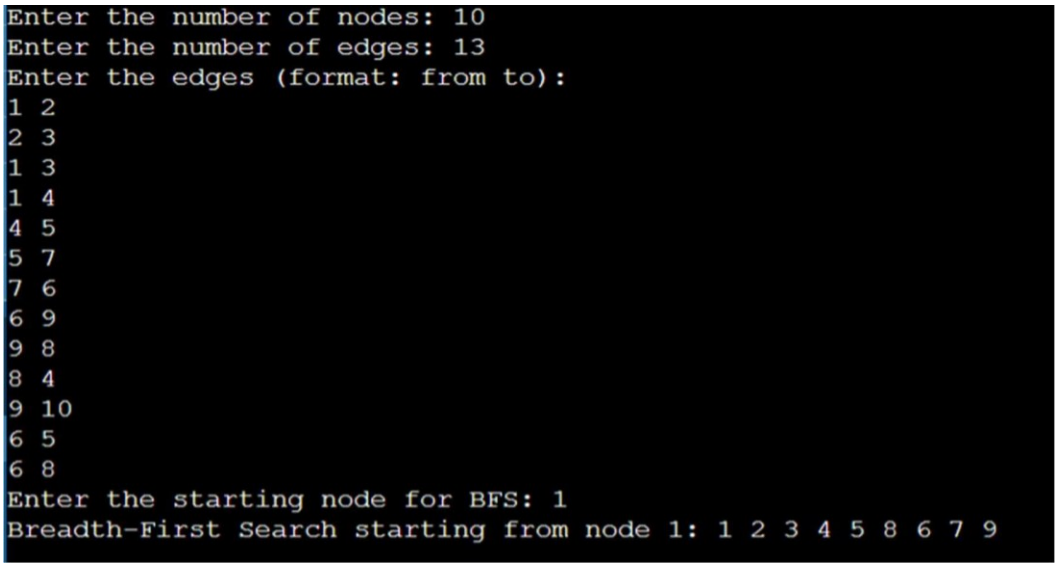
Input:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100
int queue[MAX_NODES];
int front = -1, rear = -1; void
enqueue(int node) {
    if (rear == MAX_NODES - 1) {
      printf("Queue is full. Cannot enqueue %d.\n", node);
    } else {
      if (front == -1) {front
        = 0;
      }
      rear++;
      queue[rear] = node;
    }}
  int dequeue() {int
    node;
```

```
int visited[MAX_NODES] = {0};
enqueue(start);
    visited[start] = 1;
    printf("Breadth-First Search starting from node %d: ",
    start);while (!isEmpty()) {
      int current = dequeue();
      printf("%d ", current);
      for (int i = 0; i < nodes; i++) {
        if (adjList[current][i] && !visited[i])
          {enqueue(i);
          visited[i] = 1;
        }    }  }
    }
 int main() {
    int nodes, edges;
    printf("Enter the number of nodes:
    ");scanf("%d", &nodes);
    printf("Enter the number of edges:
    ");scanf("%d", &edges);
    int adjList[MAX_NODES][MAX_NODES] = {0};
    printf("Enter the edges (format: from
    to):\n");for (int i = 0; i < edges; i++) {
      int from, to;
      scanf("%d %d", &from, &to);
```

```
    adjList[from][to] = 1;
    adjList[to][from] = 1; // For undirected graph
  }
  int startNode;
  printf("Enter the starting node for BFS:
  "); scanf("%d", &startNode);
  BFS(adjList, nodes, startNode);
  return 0;
}
```

Output:



Complexity:

O(V+E), where V is the number of nodes and E is the number of edge

EXPERIMENT-13

Program:

Implementation of Depth first search by using c/c++ and write the it's complexity.

Pseudo Code:

```
DFS(G, u)
  u.visited = true
  for each v ∈ G.Adj[u] if
    v.visited == false
      DFS(G,v)

init() {
  For each u ∈ G
    u.visited = false
   For each u ∈ G
    DFS(G, u)
}
```

Input:

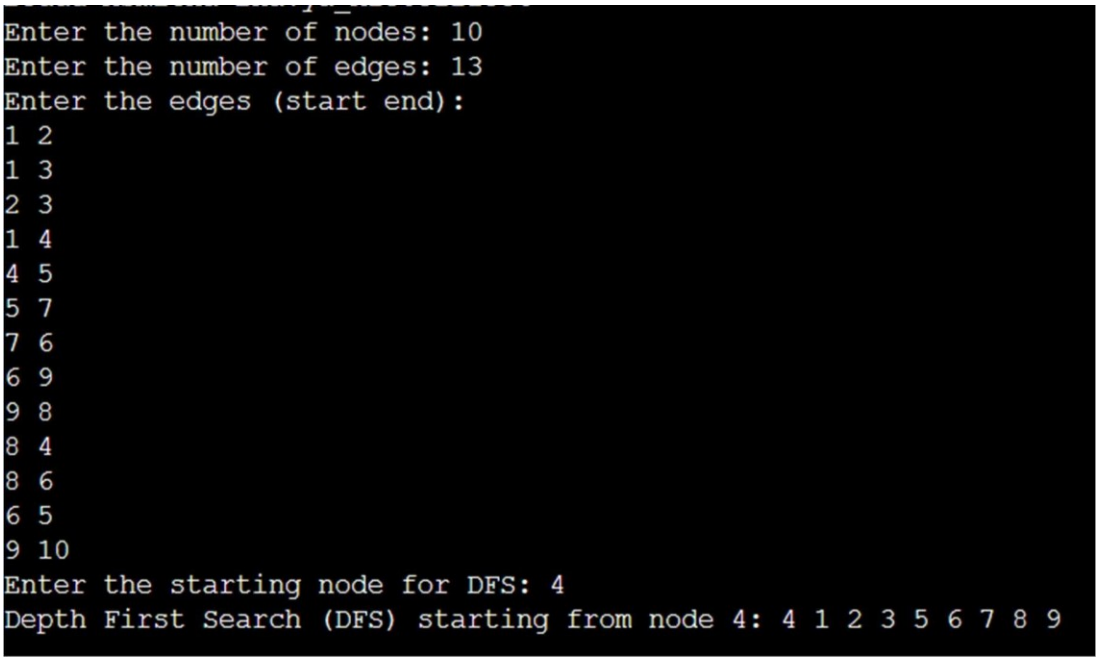
```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
bool visited[MAX_NODES];
int
```

```
adjacency_matrix[MAX_NODES][MAX_NOD
ES];int num_nodes;
void initialize() {
  for (int i = 0; i < MAX_NODES;
    i++) {visited[i] = false;
    for (int j = 0; j < MAX_NODES;
      j++) {adjacency_matrix[i][j] = 0;
    }
  }
}
void addEdge(int start, int end) {
  adjacency_matrix[start][end] =
  1;
  adjacency_matrix[end][start] = 1;
}
void dfs(int node) {
  printf("%d ", node);
  visited[node] = true;
    for (int i = 0; i < num_nodes; i++) {
    if (adjacency_matrix[node][i] &&
      !visited[i]) {dfs(i);
    }
  }
}
```

```
int main() {
  int num_edges; int
  start, end;
  printf("\nEnter the number of nodes: ");
  scanf("%d", &num_nodes);
  initialize();
  printf("Enter the number of edges:
  "); scanf("%d", &num_edges);
  printf("Enter the edges (start
  end):\n"); for (int i = 0; i <
  num_edges; i++) {
    scanf("%d %d", &start,
    &end); addEdge(start, end);
  }
  int start_node;
  printf("Enter the starting node for DFS: ");
  scanf("%d", &start_node);
  printf("Depth First Search (DFS) starting from node %d: ", start_node);
  if (front == -1) {
    printf("Queue is empty.\n");
    return -1;
  } else {
    node = queue[front];
    front++;
```

```
    if (front > rear) { front
      = rear = -1;
    }
    return node;
  }}
int isEmpty() { return
  front == -1;
}
// BFS algorithm
  void BFS(int
  adjList[][MAX_NODE
  S], int nodes, int start
  dfs(start_node); return
  0;
}
```

Output:



Complexity:

O(V + E), where V is the number of vertices and E is the number of edges in the graph.

EXPERIMENT-14

Program:

Implementation of n-Queen problem using c/c++ and write it's complexity.

Pseudo Code:

```
check_row(nxnarray, row, n){
  for(j=0;j<n;j++){//this should be same for most
    languages if(nxnarray[row][j]==1){
      return false;
    }  }
  return true;
}bool placeQueens(int current column, int current row) {
  // Base case.
  if all columns are filled, then return
  true for each row of the board, do
    if isValid (board, i, col), then set queen at place (i,
    col) in the board if solveNQueen (board, col+1) =
    true, then return true
    otherwise remove queen from place (i, col)
  from board. done
  return false
}Algorithm NQueens (k, n) //Prints all Solution to the n-
```

```
queens problem {for i := 1 to n do {
    if Place (k, i) then {x
      [k] := i;
      if (k = n) then write (x [1 : n]
      else NQueens (k+1, n);
    }  }}
Algorithm Place (k, i) {
  for j := 1 to k-1 do
    if ((x [ j ] = // in the same column or (Abs (x [ j ] - i) =Abs (j – k))) // or in
      the same diagonal then return false;
  return true;}
```

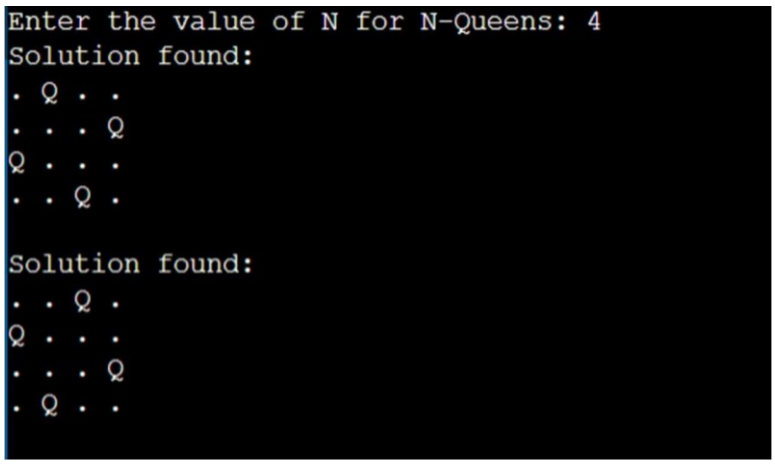
Input:

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <math.h> int
x[10];
bool canPlace(int k, int i) {
  for (int j = 1; j < k; j++) {
    if (x[j] == i || abs(x[j] - i) == abs(j -
      k)) {return false;
    }  }
```

```
  return true;
}void printSolution(int n) {
  printf("Solution found:\n");for
  (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (x[i] == j) {
        printf("Q ");
           } else {
        printf(". ");
             }    }
         printf("\n");
  }  printf("\n");
}void placeQueens(int k, int n)
  {for (int i = 1; i <= n; i++) {
    if (canPlace(k, i)) {
      x[k] = i;
      if (k == n) {
        printSolution(n);
      } else {
        placeQueens(k + 1, n);
```

```
    }   }  }}
void solveNQueens(int n) {
  placeQueens(1, n);
}int main() {int n;
  printf("\nEnter the value of N for N-
  Queens: ");scanf("%d", &n);
  if (n <= 0) {
    printf("Please enter a positive
    integer.\n");return 1;
  }
  if (n > 10) {
    printf("This program is designed for N <= 10 due to performance
    limitations.\n");return 1;
  }
  solveNQueens(n); return
  0;}
```

Output:



Complexity:

Time complexity is O(N!)

OPEN ENDED EXPERIMENT

Problem Statement:

Lukas is a civil engineer who loves designing road networks to connect n cities numbered from 1 to n. A road network can be considered as a graph with positive weights. The nodes represent cities, and each edge of the graph is associated with a road segment between two cities. The weight of an edge corresponds to the length of the associated road segment. Using directed edges, it is also possible to model one-way streets. This property has been formalized using the notion of highway dimension. There are a great number of algorithms that exploit this property and are therefore able to compute the shortest path a lot quicker than would be possible on general graphs. Develop a program to find the shortest path from each city to solve the road network problem.

Constraints:

I. Cost of the fuel is 64/- per litre. Cost of the fuel consumed should be less than 400/-
II. It must be possible to reach any city from any other city by traveling along the network of roads.
III. No two roads can directly connect the same two cities.
IV. A road cannot directly connect a city to itself.

Road Network:

Approach – I: GREEDY METHOD- DIJKASTRA'S ALGORITHM

Input:

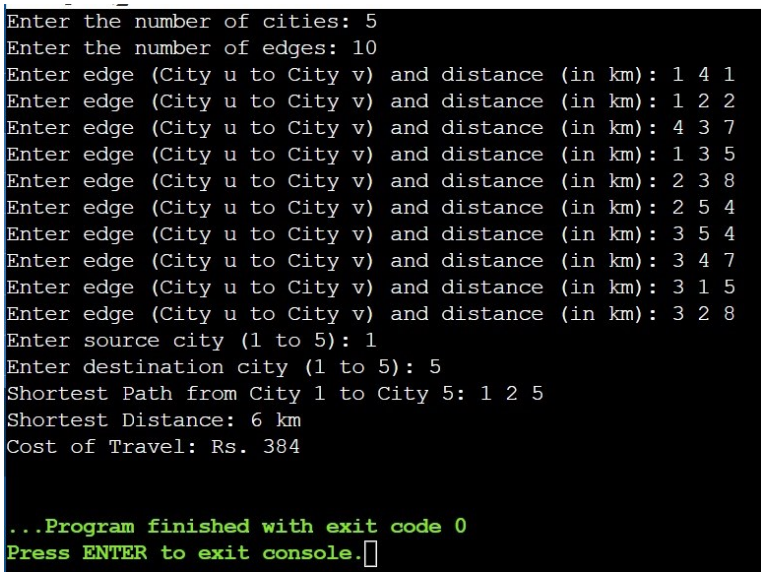
```
#include <stdio.h>
```

```
#include <limits.h>
#define MAX_CITIES 100
#define INF INT_MAX
int graph[MAX_CITIES][MAX_CITIES];
int distance[MAX_CITIES];
int visited[MAX_CITIES];
int path[MAX_CITIES];
int minDistance(int n) {
    int min = INF, min_index;
    for (int i = 0; i < n; i++) {
        if (!visited[i] && distance[i] < min) {
            min = distance[i];
            min_index = i;
        }
    }
    return min_index;
}void printPath(int n, int source, int dest) {
    if (dest == source) {
        printf("%d ", source + 1);
       return;
    }
    printPath(n, source, path[dest]);
    printf("%d ", dest + 1);
}
void dijkstra(int n, int source, int dest) {
    for (int i = 0; i < n; i++) {
        distance[i] = INF;
```

```
        visited[i] = 0;
    }
    distance[source] = 0;
    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(n);
        visited[u] = 1;
        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] && distance[u] != INF &&
                distance[u] + graph[u][v] < distance[v]) {
                distance[v] = distance[u] + graph[u][v];
                path[v] = u;
            }
        }   }

    printf("Shortest Path from City %d to City %d: ", source + 1, dest + 1);
    printPath(n, source, dest);
    printf("\nShortest Distance: %d km\n", distance[dest]);
    printf("Cost of Travel: Rs. %d\n", distance[dest] * 64);
}
int main() {
    int n, source, dest;
    printf("Enter the number of cities: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            graph[i][j] = 0;
        }   }
```

```
    printf("Enter the number of edges: ");
    int num_edges;
    scanf("%d", &num_edges);
    for (int i = 0; i < num_edges; i++) {
        int u, v, w;
        printf("Enter edge (City u to City v) and distance (in km): ");
        scanf("%d %d %d", &u, &v, &w);
        graph[u - 1][v - 1] = w;
    }
    printf("Enter source city (1 to %d): ", n);
    scanf("%d", &source);

    if (source < 1 || source > n) {
        printf("Invalid source city.\n");
        return 1;
    }
    printf("Enter destination city (1 to %d): ", n);
    scanf("%d", &dest);
    if (dest < 1 || dest > n) {
        printf("Invalid destination city.\n");
        return 1;
    }
    dijkstra(n, source - 1, dest - 1);
    return 0;
}
```

Output:



Complexity: O(E+V*logV)

Approach – II DYNAMIC PROGRAMMING-BELLMAN FORD

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
// Constants
#define MAX_CITIES 100
#define MAX_ROADS 1000
#define FUEL_COST_PER_LITRE 65
#define MAX_FUEL_COST 400
// Structure to represent a road segment
typedef struct {
```

```
    int from, to, distance;
} Road;
// Structure to represent a city
typedef struct {
    int distance, fuel_cost;
} City;
// Function to find the shortest path using Bellman-Ford algorithm
void findShortestPath(int n, int m, Road roads[], City cities[], int source) {
    // Initialize the source city
    cities[source].distance = 0;
    cities[source].fuel_cost = 0;
    // Relax edges for (n-1) times
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int from = roads[j].from;
            int to = roads[j].to;
            int distance = roads[j].distance;
            if (cities[from].distance != INT_MAX && cities[from].fuel_cost <
MAX_FUEL_COST) {
                if (cities[from].distance + distance < cities[to].distance) {
                    cities[to].distance = cities[from].distance + distance;
                    cities[to].fuel_cost = cities[from].fuel_cost + (distance *
FUEL_COST_PER_LITRE);
                    // Check if the fuel cost exceeds the limit
                    if (cities[to].fuel_cost > MAX_FUEL_COST) {
                        printf("Fuel cost exceeds 400. The road network violates
constraints.\n");
                        exit(1);
```

```
                }
            }
        }
    }
    // Check for negative cycles
    for (int i = 0; i < m; i++) {
        int from = roads[i].from;
        int to = roads[i].to;
        int distance = roads[i].distance;
        if (cities[from].distance != INT_MAX && cities[from].distance +
distance < cities[to].distance) {
            printf("Negative cycle detected. The road network violates
constraints.\n");
            exit(1);
        }
    }
}
int main() {
    int n, m;
    printf("Enter the number of cities (n) and roads (m): ");
    scanf("%d %d", &n, &m);
    Road roads[MAX_ROADS];
    City cities[MAX_CITIES];
    // Initialize cities and roads
    for (int i = 1; i <= n; i++) {
        cities[i].distance = INT_MAX;
        cities[i].fuel_cost = INT_MAX;
```

```
    }
    for (int i = 0; i < m; i++) {
        printf("Enter road details (from, to, distance): ");
        scanf("%d %d %d", &roads[i].from, &roads[i].to, &roads[i].distance);
        // Check if a road connects a city to itself
        if (roads[i].from == roads[i].to) {
            printf("Error: A road cannot connect a city to itself.\n");
            return 1;
        }
        // Check if two roads connect the same two cities
        for (int j = 0; j < i; j++) {
            if ((roads[i].from == roads[j].from && roads[i].to == roads[j].to) ||
                (roads[i].from == roads[j].to && roads[i].to == roads[j].from)) {
                printf("Error: Two roads cannot directly connect the same two
cities.\n");
                return 1;
            }
        }
    }
    int source;
    printf("Enter the source city: ");
    scanf("%d", &source);
    findShortestPath(n, m, roads, cities, source);
    // Display the shortest distances and fuel costs
    printf("Shortest distances and fuel costs from city %d:\n", source);
    for (int i = 1; i <= n; i++) {
        if (i != source) {
```

```
            printf("City %d: Distance = %d, Fuel Cost = %d\n", i,
cities[i].distance, cities[i].fuel_cost);
            if (cities[i].fuel_cost > MAX_FUEL_COST) {
                printf("Fuel cost exceeds 400. The road network violates
constraints.\n");
                return 1;
            }
        }
    }
    return 0;
}
```

Output:

```
Enter the number of cities (n) and roads (m): 5 7
Enter road details (from, to, distance): 1 2 2
Enter road details (from, to, distance): 1 3 5
Enter road details (from, to, distance): 1 4 1
Enter road details (from, to, distance): 4 3 7
Enter road details (from, to, distance): 2 3 8
Enter road details (from, to, distance): 2 5 4
Enter road details (from, to, distance): 3 5 4
Enter the source city: 1
Shortest distances and fuel costs from city 1:
City 5: Distance = 6, Fuel Cost = 390
```

Complexity: O(E+V)

E: EDGES

V: VERTICES

Approach– III Back Tracking

Input:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_CITIES 100
#define INF 999999
int n; // number of cities
int m; // number of edges
int network[MAX_CITIES][MAX_CITIES]; // adjacency matrix to
represent the road network
int path[MAX_CITIES]; // array to store the shortest path
bool visited[MAX_CITIES]; // array to track visited cities
int sourceCity, destinationCity; // Variables to store source and
destination cities
void initialize() {
    int i, j;
    for (i = 1; i <= n; i++) {
        visited[i] = false;
        for (j = 1; j <= n; j++) {
            network[i][j] = INF; // initialize the road network with a reasonable
value
        }
    }
}void addRoad(int city1, int city2, int weight) {
    network[city1][city2] = weight;
    network[city2][city1] = weight; // assuming it's an undirected network
}
void backtrack(int currentCity, int totalDistance, int pathLength) {
```

```
    int i;
    visited[currentCity] = true; // mark the current city as visited
    path[pathLength] = currentCity; // add the current city to the path
    if(totalDistance<100){
    if (currentCity == destinationCity) {
        // Found the destination city, print the path and distance
        printf("Path: ");
        for (i = 1; i <= pathLength; i++) {
            printf("%d ", path[i]);
        }
        printf("Distance: %d\n", totalDistance);
        visited[currentCity] = false; // backtrack
        return;
    }    // Explore all unvisited neighboring cities
    for (i = 1; i <= n; i++) {
        if (!visited[i]) {
            backtrack(i, totalDistance + network[currentCity][i], pathLength +
1);
        }
    }    visited[currentCity] = false; // backtrack
    }
}
int main() {
    int i, j, city1, city2, weight;
    printf("Enter the number of cities: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &m);
```

```
    initialize();
    printf("Enter the road segments (city1, city2, weight):\n");
    for (i = 1; i <= m; i++) {
        scanf("%d %d %d", &city1, &city2, &weight);
        addRoad(city1, city2, weight);
    }
    printf("Enter source city: ");
    scanf("%d", &sourceCity);
    printf("Enter destination city: ");
    scanf("%d", &destinationCity);
    printf("Shortest path from city %d to city %d:\n", sourceCity,
destinationCity);
   // Initialize path array before the backtrack call
    for (j = 1; j <= n; j++) {
        path[j] = 0; }
    backtrack(sourceCity, 0, 1);
    return 0;}
```

Output:

```
Enter the number of cities: 4
Enter the number of edges: 5
Enter the road segments (city1, city2, weight):
0 1 3
0 2 5
2 3 7
1 2 3
1 3 6
Enter source city: 1
Enter destination city: 3
Shortest path from city 1 to city 3:
Path: 1 3 Distance: 6
Fuel Consumption: 384
```

Complexity:

O(K^N) where K is the number of times the function calls itself.