VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

ANALYSIS AND DESIGN OF ALGORITHMS

Submitted by

Harbakhshish Singh Arora (1BM23CS104)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019 Feb 2025 to Jun 2025

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(23CS4PCADA)**" carried out by **Harbakhshish Singh (1BM23CS104) ,** who is a 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 year 2025. The Lab report has been approved as it satisfies the academic requirements in respect of **ANALYSIS AND DESIGN OF ALGORITHMS(23CS4PCADA)** work prescribed for the said degree.

Prof Srushti CS
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	LeetCode Problem- Remove Nth Node from end of a List	5-6
2	LeetCode Problem- Reveal Cards in Increasing Order	6-9
3	LeetCode Problem: Longest Absolute File Path.	9-11
4	Topological ordering of vertices in a given digraph	11-15
5	LeetCode Problem- Course Scheduling	15-17
6	Merge Sort	17-21
7	Quick Sort	21-23
8	LeetCode Problem- 3SUM	23-26
9	MST of a given undirected graph using Prim's algorithm.	26-29
10	MST of a given undirected graph using Kruskal's algorithm.	29-33
11	Shortest paths to other vertices using Dijkstra's algorithm	33-36
12	Johnson Trotter algorithm to generate permutations	36-39
13	Fractional knapsack problem using Greedy technique	39-42
14	LeetCode Problem- Largest Odd Number	42-43
15	0/1 Knapsack problem using dynamic programming	44-46
16	LeetCode Problem-Fibonacci Number	46-47
17	Heap Sort	47-50
18	All Pair Shortest paths problem using Floyd's algorithm	50-53

19	LeetCode Problem-Shortest Path	53-56
20	“N-Queens Problem” using Backtracking	56-59

Course Outcome

CO1	Analyze time complexity of recursive and non-recursive algorithms using asymptotic notations
CO2	Apply various algorithm 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.

INDEX

Name Harbakshish Singh Std 4th Sem Sec B

Roll No. Subject A.D.A School/College

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	28.3.25	Implementation of Stack Using array	1-4	
2	28.3.25	Implementation of Stack Using Linked List	5-9	
3	21.3.25	Merge Sort Program	10-14	10 4/4/25
4	4.4.25	Quick Sort Program	15-18	4/4/25 10
5	4.4.25	Prim's algorithm	19-20	
6	4.4.25	Kruskal's algorithm	21-23	
7	16.5.25	Topological Sort Using DFS	24-26	6
8	16.5.25	0/1 Knapsack Using DP	27-28	6
9	16.5.25	Floyd algorithm Shortest Path	29-32	16/5
10	17.5.25	Dijkstra algorithm	32-34	10
11	17.5.25	Leetcode Program - 1	35-36	18/5/25
12	17.5.25	Leetcode Program - 2	37-38	
13	17.5.25	Leetcode Program - 3	39-40	23/5/25
14	17.5.25	Leetcode Program - 4	41	
15	17.5.25	Leetcode Program - 5	42-43	
16	17.5.25	Leetcode Program - 6	44	
17	23.5.25	N Queens Problem	45-46	10
18	23.5.25	Johnson trotter Problem	47-48	10
19	23.5.25	Heap Sort Problem	49-50	10 23/5/25

LAB-1

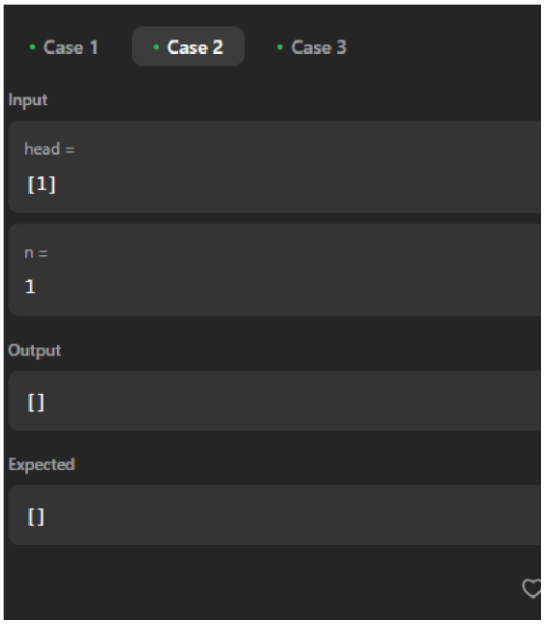
Question- 1:

LeetCode Problem - Remove Nth Node from end of a List **Code:**

```
    struct ListNode* removeNthFromEnd(struct ListNode*
head, int n) {     struct ListNode* temp1=(head);
struct ListNode* temp2=(head);
          for (int
i=0;i<n;i++){
temp1=temp1->next;
    }     if (head-
>next==NULL){
return NULL;
    }
if(temp1==NULL){
head=head->next;
return head;


    }
```

```
            while(temp1-
>next!=NULL){
temp1=temp1->next;
temp2=temp2->next;
    }      temp2->next=temp2->next-
>next;      return head;


}
```

Result:



Question -2:

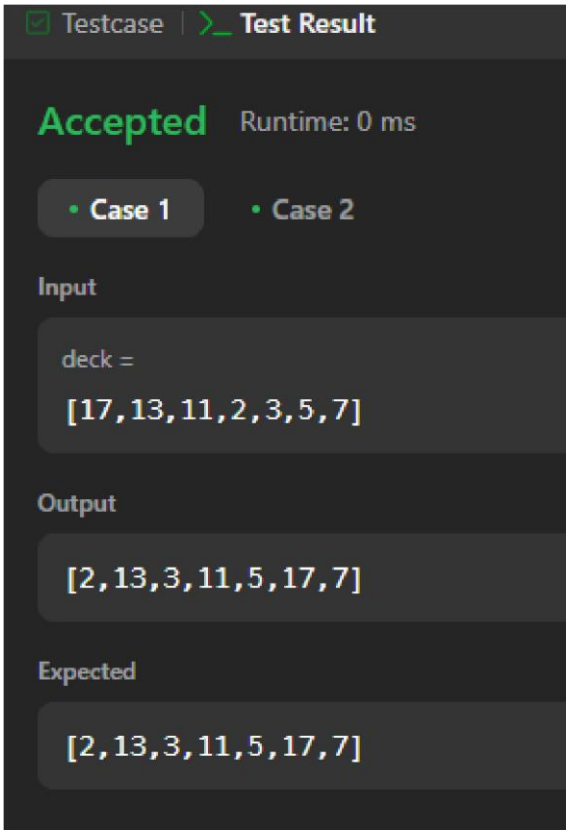
LeetCode Problem- Reveal Cards in Increasing Order

CODE:

```
int cmp(const void *a , const void *b )
{
    return *(int*)a - *(int*)b;
}



void PushBack(int deck[static 1] , int size , int currFull)
{
    //Swap      int end =
deck[size-1];     deck[size-
currFull-1] = end;

    //PushBack
    for(int i= size-1 ; i>=(size-currFull); i--)
    {
        deck[i] = deck[i-1];
    }
}



int*
deckRevealedIncreasing(
        int deck[static 1],          int
deckSize,          int* returnSize //
Reference
    )
{
    int *res = malloc(sizeof(int) * deckSize);
```

```
    *returnSize = deckSize;

    qsort(deck,deckSize , sizeof(int) , cmp);

    res[deckSize-1] = deck[deckSize-1];
    int currFull = 1;

    for(int i = deckSize - 2 ; i>= 0 ; --i )
    {
        PushBack(res,deckSize,currFull);
        currFull++;         res[deckSize-
currFull] = deck[i];
    }
    return res;

}
```

OUTPUT:



Question 3:

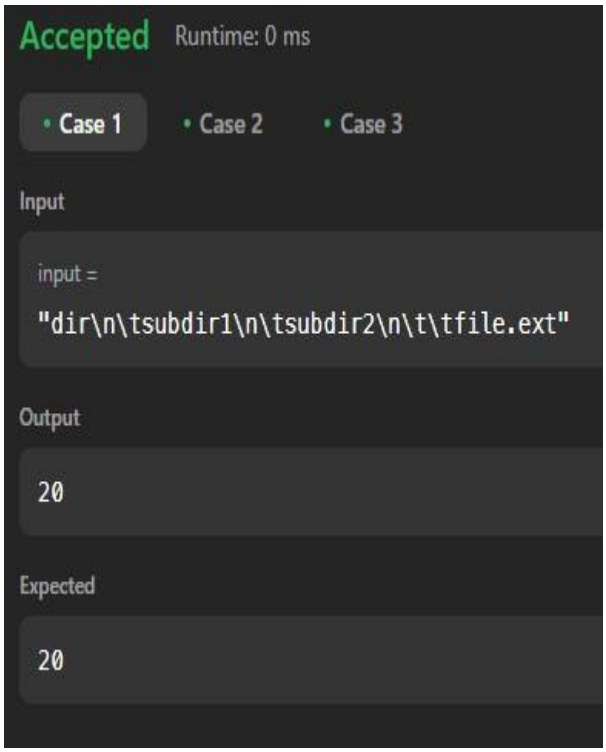
LeetCode Problem: Longest Absolute File Path

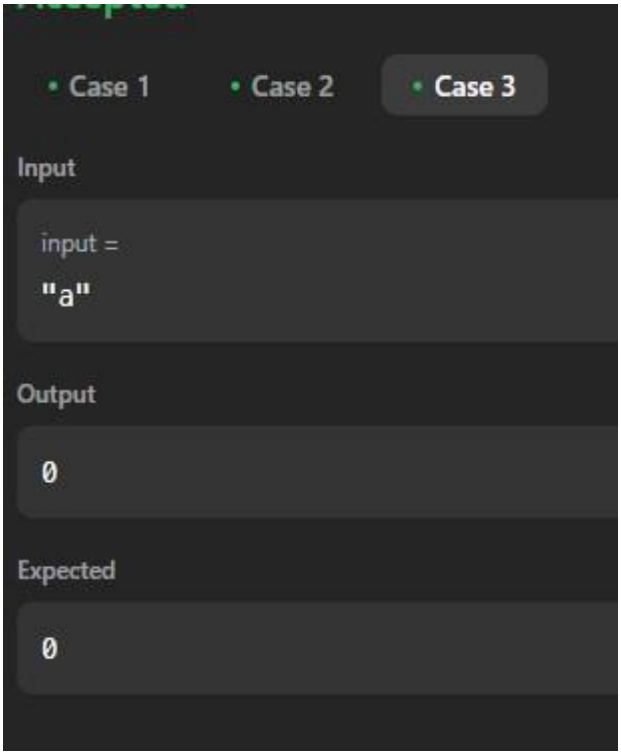
Code:

```
int lengthLongestPath(char *input) {
char *token;     int len[100] = {0};
int ans = 0;


   token = strtok(input, "\n");
while (token != NULL) {
```

```
        int l = 0;         while
(token[l] == '\t') {
            l++;
        }

        len[l] = strlen(token) - l;

        if (strchr(token, '.')) {
int totalLength = 0;             for
(int i = 0; i <= l; i++) {
totalLength += len[i];
            }
            ans = (ans > totalLength + l) ? ans : totalLength + l;
        }

        token = strtok(NULL, "\n");
    }

    return ans;
}
```

Output:





LAB 2

QUESTION 1:

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

CODE:

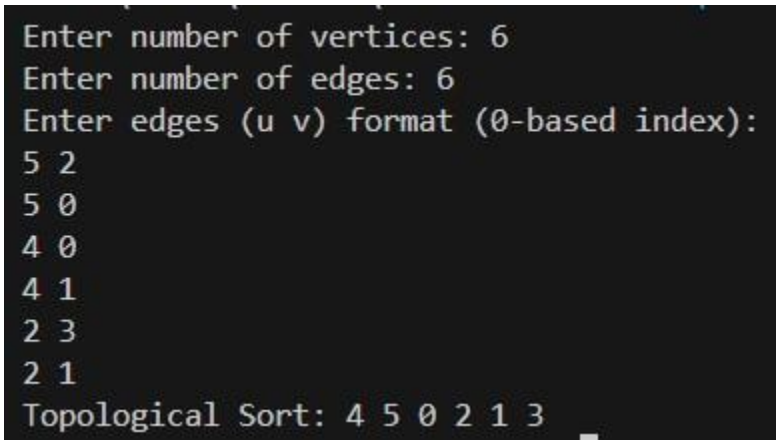
```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 100

typedef struct Graph {
    int vertices;
```

```
    int adj[MAX_VERTICES][MAX_VERTICES];
int in_degree[MAX_VERTICES];
} Graph;

void initGraph(Graph *g, int vertices) {
g->vertices = vertices;     for (int i = 0;
i < vertices; i++) {         g-
>in_degree[i] = 0;         for (int j = 0; j
< vertices; j++) {             g->adj[i][j] =
0;
        }
    }
}

void addEdge(Graph *g, int u, int v) {
g->adj[u][v] = 1;     g->in_degree[v]++;
}

void topologicalSort(Graph *g) {     int
queue[MAX_VERTICES], front = 0, rear = 0;     int
topOrder[MAX_VERTICES];
    int index = 0;

    for (int i = 0; i < g->vertices; i++) {
if (g->in_degree[i] == 0) {
queue[rear++] = i;
        }
    }
```

```
    while (front < rear) {
int u = queue[front++];
topOrder[index++] = u;

        for (int v = 0; v < g->vertices; v++) {
if (g->adj[u][v] == 1) {                g-
>in_degree[v]--;                if (g-
>in_degree[v] == 0) {
queue[rear++] = v;
                }
            }
        }
    }

    if (index != g->vertices) {         printf("Graph contains a cycle,
topological sorting is not possible.\n");
        return;
    }

    printf("Topological Sort: ");     for
(int i = 0; i < g->vertices; i++) {
printf("%d ", topOrder[i]);

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

```
int main() {     Graph g;
int vertices, edges, u, v;

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

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

    printf("Enter edges (u v) format (0-based index):\n");
    for (int i = 0; i < edges; i++) {
scanf("%d %d", &u, &v);
addEdge(&g, u, v);
    }

    topologicalSort(&g);

    return 0;
}
```

OUTPUT:



QUESTION 2:

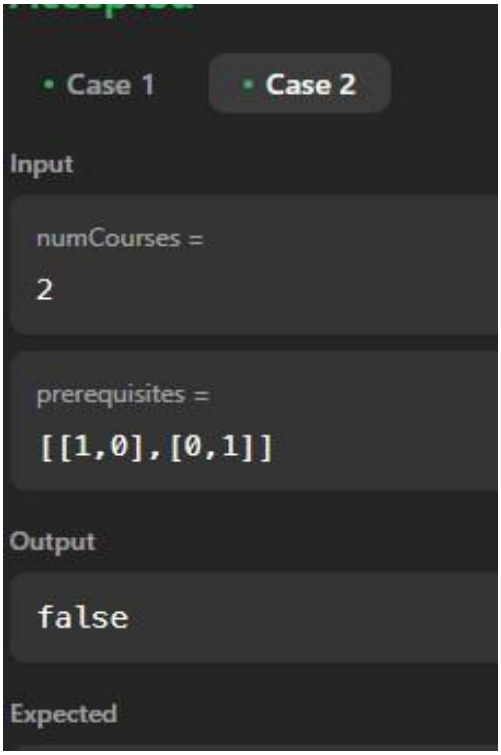
LeetCode Problem: Course Scheduling

CODE:

```
#define MAX (13)

bool Cycle( int id, int coursemap[][MAX], int cpr[], int *visited ) {

   if ( visited[id] > 0 )
return true;



   if ( visited[id] == 0 ) {         visited[id] = 1;         for ( int i=0;
i<cpr[id]; i++ ) {             if ( Cycle( coursemap[id][i],
coursemap, cpr, visited ) ) {
               return true;
          }
}
       visited[id] = -1;
   }
```

```
    return false;
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize){     int coursemap[2001][MAX];
    int cpr[2001]     = { 0 };
int visited[2001] = { 0 };



    for ( int i=0; i<prerequisitesSize; i++ ) {
int course = prerequisites[i][0];         int
prereq = prerequisites[i][1];

        coursemap[course][cpr[course]++] = prereq;
    }

    for ( int i=0; i<numCourses; i++ ) {

        if ( Cycle( i, coursemap, cpr, visited ) ) {
return false;
        }
    }

    return true;
}
```

OUTPUT:



LAB 3

QUESTION 1:

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.

CODE:

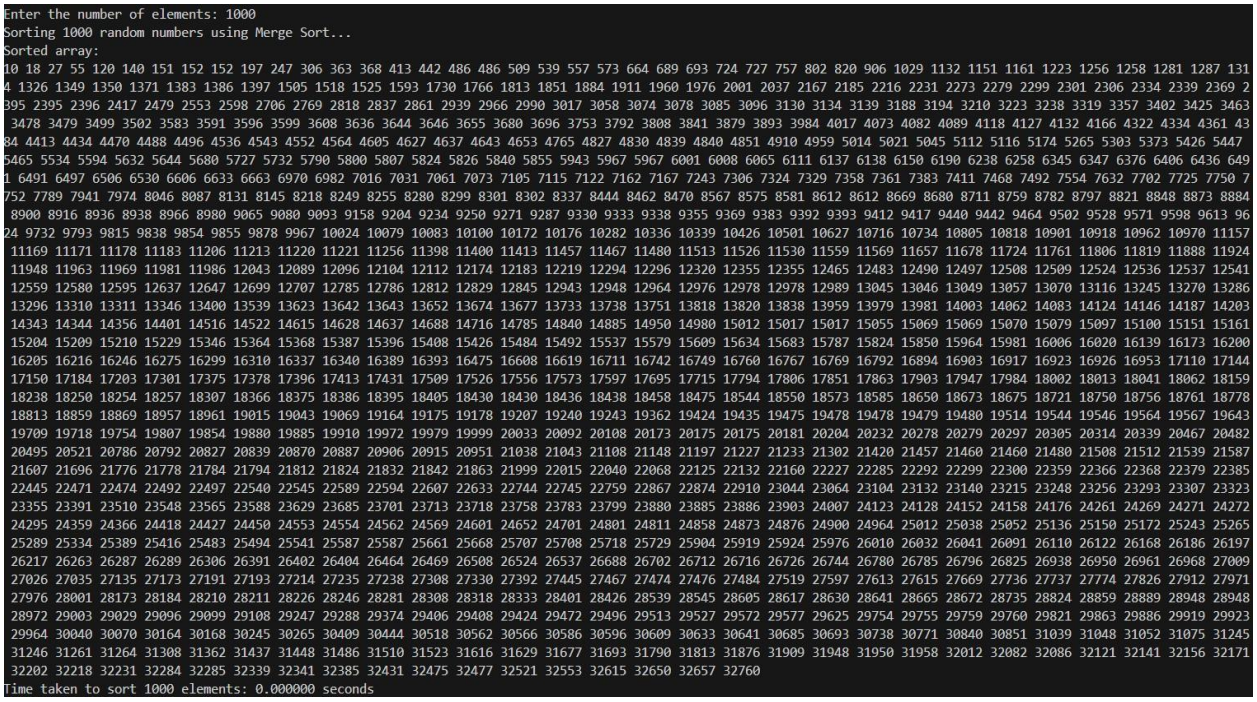
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;      int n2 = right - mid;

int L[n1], R[n2];
```

```
    for (int i = 0; i < n1; i++)
L[i] = arr[left + i];     for (int
j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    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 left, int right) {
```

```
    if (left < right) {         int mid =
left + (right - left) / 2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
    }
}

int main() {
    int n;

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

    int arr[n];
    srand(time(0));

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100000;  // Random integers between 0 and 99999
    }

    printf("Running Merge Sort for N = %d...\n", n);

    clock_t start = clock();
mergeSort(arr, 0, n - 1);     clock_t
end = clock();
```

```
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken to sort N = %d elements: %f seconds\n", n, time_taken);


    return 0;
}
```

OUTPUT:



QUESTION 2:

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

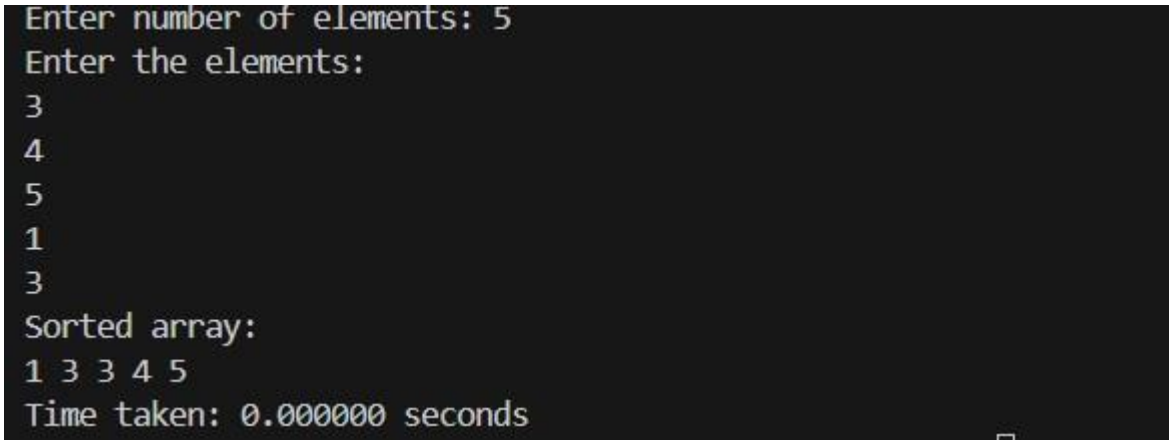
CODE:

```
#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; 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 n;
    printf("Enter number of elements: ");
scanf("%d", &n);     int *arr = (int
```

```
*)malloc(n * sizeof(int)); printf("Enter
the elements:\n");
    for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
    }

    clock_t start = clock();
quickSort(arr, 0, n - 1);     clock_t
end = clock();

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

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

    free(arr);
return 0;
}
```

OUTPUT:



QUESTION 3:

LeetCode Problem : 3Sum

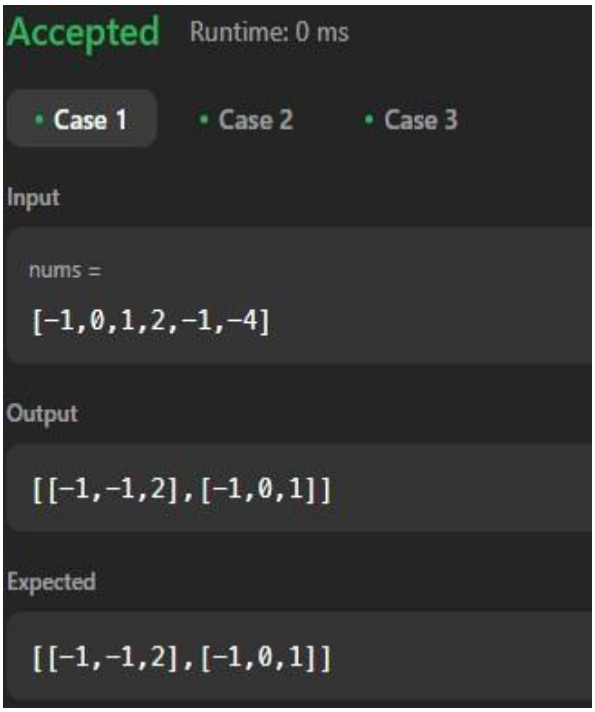
Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

CODE:

```
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}


int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes)
{
    if (numsSize < 3) {
*returnSize = 0;        return
NULL;
    }
    qsort(nums, numsSize, sizeof(int), compare);     int** result =
(int**)malloc(sizeof(int*) * numsSize * numsSize);
    *returnColumnSizes = (int*)malloc(sizeof(int) * numsSize * numsSize);
*returnSize = 0;
```

```
    for (int i = 0; i < numsSize - 2; i++) {         if (i
> 0 && nums[i] == nums[i - 1]) continue;         int
left = i + 1, right = numsSize - 1;         while (left
< right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {                 result[*returnSize] =
(int*)malloc(sizeof(int) * 3);
result[*returnSize][0] = nums[i];
result[*returnSize][1] = nums[left];
result[*returnSize][2] = nums[right];
                (*returnColumnSizes)[*returnSize] = 3;
                (*returnSize)++;                 while (left < right && nums[left]
== nums[left + 1]) left++;                 while (left < right && nums[right]
== nums[right - 1]) right--;
                left++; right--;
} else if (sum < 0) left++;
else right--;
        }
    }
    return result;
}
```

OUTPUT:



LAB4:

QUESTION 1:

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

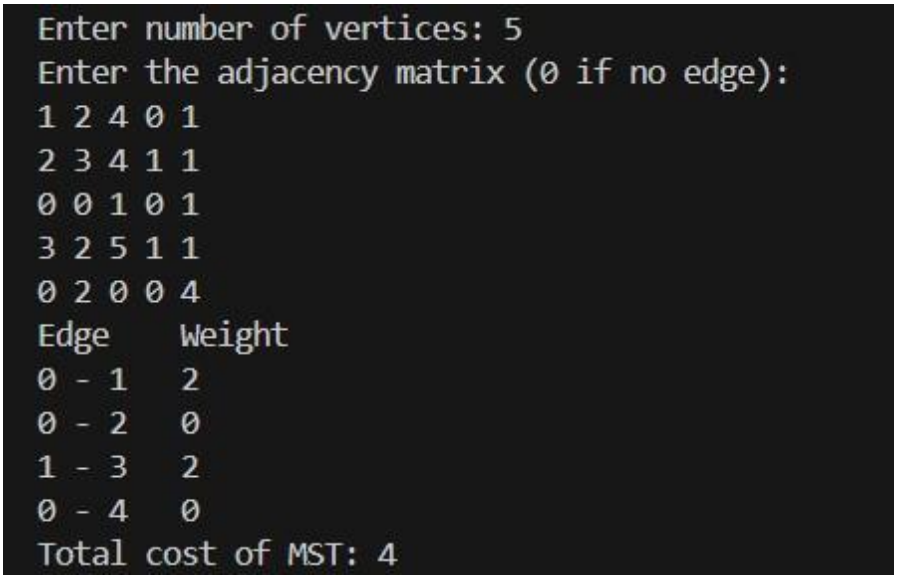
CODE:

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

#define MAX 100

int minKey(int key[], int mstSet[], int n) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < n; v++) {        if
(mstSet[v] == 0 && key[v] < min) {
min = key[v];            min_index = v;
```

```
        }
    }
    return min_index;
}

void primMST(int graph[MAX][MAX], int n) {
int parent[MAX];     int key[MAX];     int
mstSet[MAX];

    for (int i = 0; i < n; i++) {
        key[i] = INT_MAX;
mstSet[i] = 0;
    }

    key[0] = 0;
parent[0] = -1;

    for (int count = 0; count < n - 1; count++) {
int u = minKey(key, mstSet, n);         mstSet[u]
= 1;

        for (int v = 0; v < n; v++) {             if (graph[u][v] &&
mstSet[v] == 0 && graph[u][v] < key[v]) {
                parent[v] = u;
key[v] = graph[u][v];
            }
        }
    }
```

```
    int totalCost = 0;
printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++) {          printf("%d - %d \t%d\n",
parent[i], i, graph[i][parent[i]]);          totalCost +=
graph[i][parent[i]];
    }
    printf("Total cost of MST: %d\n", totalCost);
}

int main() {
    int n;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (0 if no edge):\n");
    for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
scanf("%d", &graph[i][j]);

        }
    }

    primMST(graph, n);
    return 0;
}
```

OUTPUT:



QUESTION 2:

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

CODE:

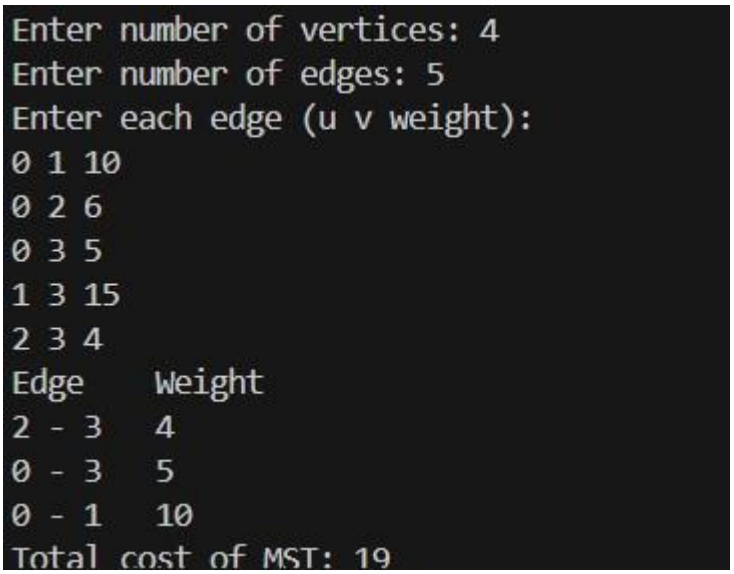
```
#include <stdio.h>

#include <stdlib.h>


#define MAX 100


typedef struct {

int u, v, weight;

} Edge;


int parent[MAX];
```

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

void unionSet(int i, int j) {
int a = find(i);     int b =
find(j);     parent[a] = b;
}

void kruskal(Edge edges[], int n, int e) {
Edge result[MAX];
   int i = 0, j = 0, totalCost = 0;

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


   while (j < n - 1 && i < e) {
int u = edges[i].u;         int v
= edges[i].v;         int set_u
= find(u);         int set_v =
find(v);

       if (set_u != set_v) {
result[j++] = edges[i];
totalCost += edges[i].weight;
unionSet(set_u, set_v);
```

```
        }
i++;
    }

    printf("Edge \tWeight\n");     for (int k = 0; k < j; k++)
printf("%d - %d \t%d\n", result[k].u, result[k].v, result[k].weight);
printf("Total cost of MST: %d\n", totalCost);
}

int compare(const void *a, const void *b) {
    Edge *e1 = (Edge *)a;     Edge
*e2 = (Edge *)b;     return e1-
>weight - e2->weight;
}

int main() {
    int n, e;
    Edge edges[MAX];

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

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

    printf("Enter each edge (u v weight):\n");
    for (int i = 0; i < e; i++)
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);
```

```
    qsort(edges, e, sizeof(Edge), compare);
kruskal(edges, n, e);

    return 0;
}
```

OUTPUT:



LAB 5:

QUESTION 1:

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

CODE:

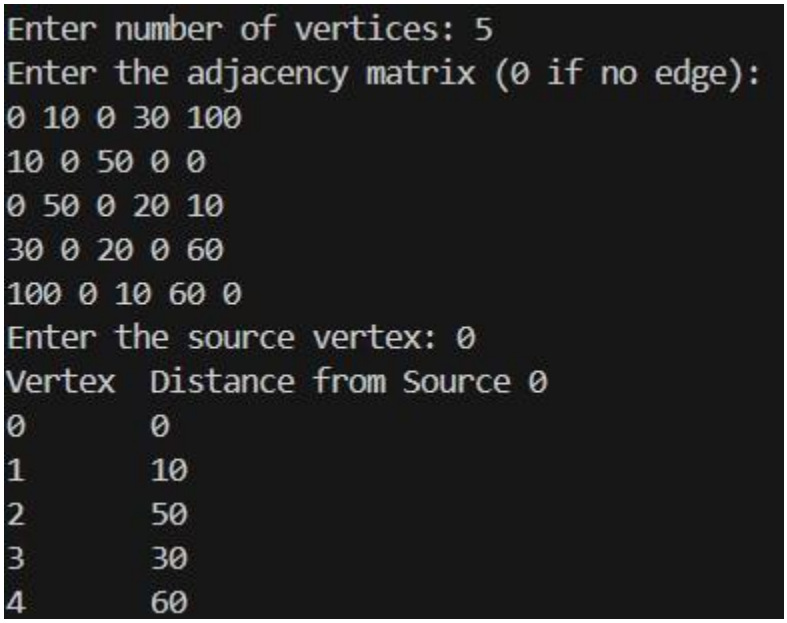
```
#include <stdio.h>
#include <limits.h>

#define MAX 100
#define INF 99999
```

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

void dijkstra(int graph[MAX][MAX], int n, int src) {
int dist[MAX], visited[MAX];

   for (int i = 0; i < n; i++) {
dist[i] = INF;         visited[i]
= 0;
   }

   dist[src] = 0;

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

       for (int v = 0; v < n; v++) {             if (!visited[v] &&
graph[u][v] && dist[u] != INF &&                dist[u] +
```

```
graph[u][v] < dist[v]) {                dist[v] = dist[u] +
graph[u][v];
            }
        }
    }

    printf("Vertex\tDistance from Source %d\n", src);
    for (int i = 0; i < n; i++)
printf("%d\t%d\n", i, dist[i]);
}

int main() {
    int n, src;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (0 if no edge):\n");
    for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
scanf("%d", &graph[i][j]);

    printf("Enter the source vertex: ");
scanf("%d", &src);

    dijkstra(graph, n, src);
```

```
    return 0;
}
```

OUTPUT:



QUESTION 2:

Implement Johnson Trotter algorithm to generate permutations.

CODE:

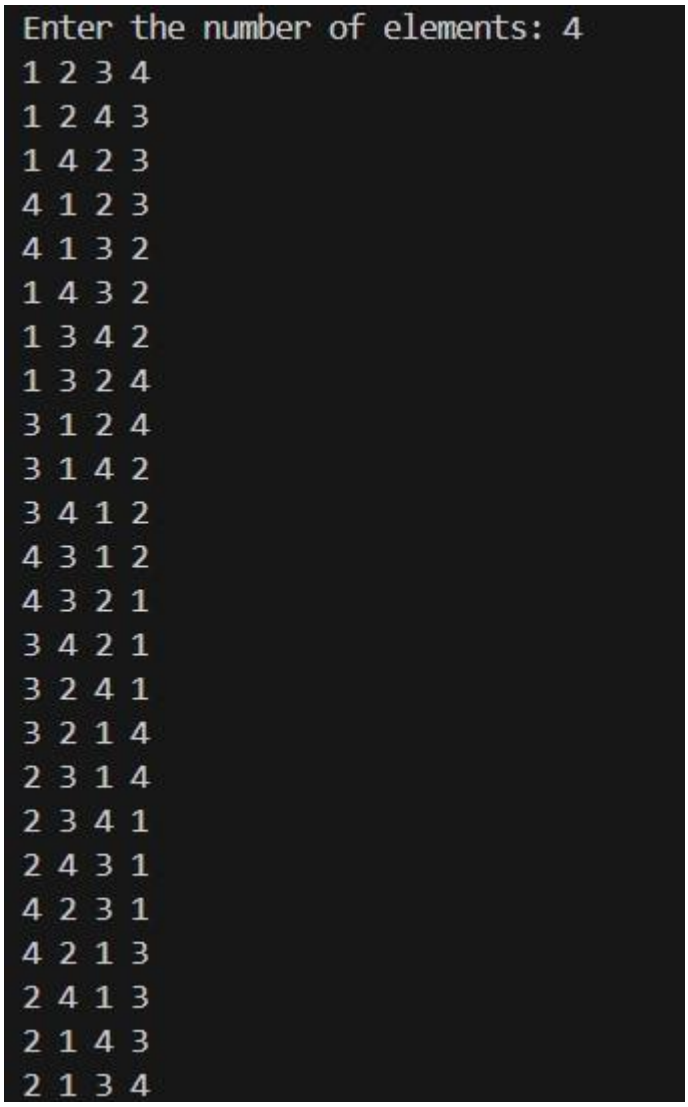
```
#include <stdio.h>

#define LEFT -1
#define RIGHT 1

typedef struct {
    int value;
    int dir;
```

```
} Element;

int getMobile(Element perm[], int n) {
int mobile = 0, mobileIndex = -1;
   for (int i = 0; i < n; i++) {
int next = i + perm[i].dir;
       if (next >= 0 && next < n && perm[i].value > perm[next].value) {
           if (perm[i].value > mobile) {
mobile = perm[i].value;
mobileIndex = i;
           }
       }
   }
   return mobileIndex;
}

void printPermutation(Element perm[], int n) {
   for (int i = 0; i < n; i++)
printf("%d ", perm[i].value);
printf("\n");
}

void generatePermutations(int n) {
Element perm[n];
   for (int i = 0; i < n; i++) {
perm[i].value = i + 1;         perm[i].dir
= LEFT;
   }
```

```
    printPermutation(perm, n);

    while (1) {
        int mobileIndex = getMobile(perm, n);
if (mobileIndex == -1)
            break;

        int swapIndex = mobileIndex + perm[mobileIndex].dir;
Element temp = perm[mobileIndex];         perm[mobileIndex]
= perm[swapIndex];         perm[swapIndex] = temp;

        mobileIndex = swapIndex;

        for (int i = 0; i < n; i++) {
            if (perm[i].value > perm[mobileIndex].value)
                perm[i].dir *= -1;
        }

        printPermutation(perm, n);
    }
}

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

    return 0;
```

}

OUTPUT:



LAB 6:

QUESTION 1:

Implement fractional knapsack problem using Greedy technique.

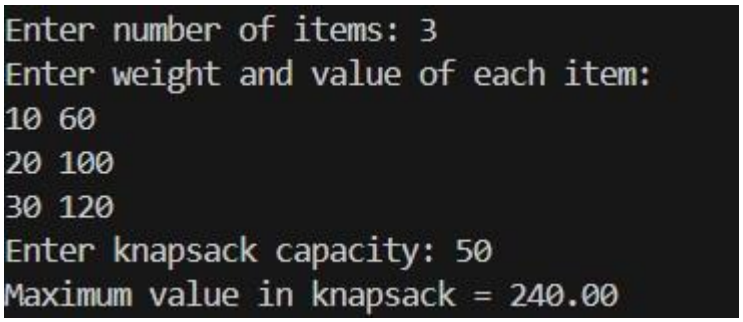
CODE:

```
#include <stdio.h>

typedef struct {
int weight;     int
value;     float
ratio;
} Item;

void sortItems(Item items[], int n) {
   for (int i = 0; i < n - 1; i++) {         for (int j
= 0; j < n - i - 1; j++) {             if
(items[j].ratio < items[j + 1].ratio) {
Item temp = items[j];                 items[j] =
items[j + 1];                items[j + 1] = temp;
           }
       }
   }
}

void fractionalKnapsack(Item items[], int n, int capacity) {     sortItems(items, n);

   float totalValue = 0.0;
int currWeight = 0;

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

```
        if (currWeight + items[i].weight <= capacity) {
currWeight += items[i].weight;              totalValue
+= items[i].value;
        } else {
            int remain = capacity - currWeight;
totalValue += items[i].ratio * remain;
            break;
        }
    }

    printf("Maximum value in knapsack = %.2f\n", totalValue);
}

int main() {     int
n, capacity;

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

    Item items[n];

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

```
    printf("Enter knapsack capacity: ");
scanf("%d", &capacity);

    fractionalKnapsack(items, n, capacity);

    return 0;
}
```

OUTPUT:



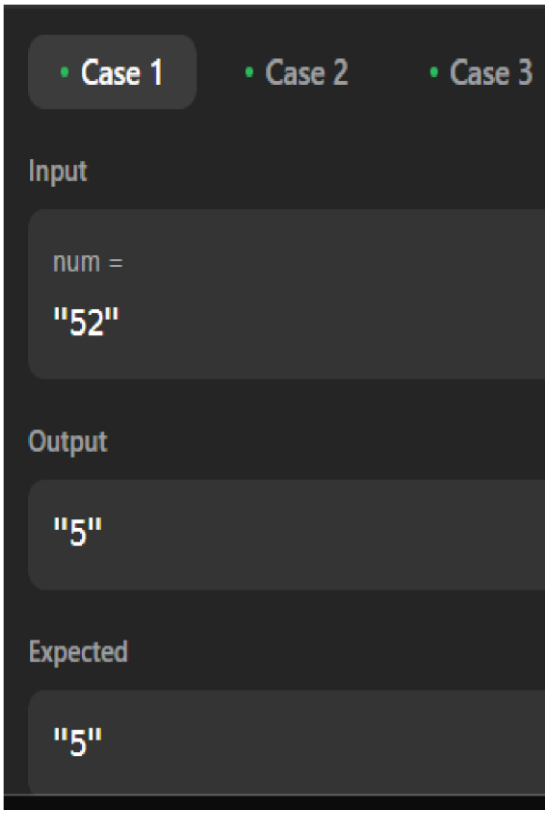
QUESTION 2:

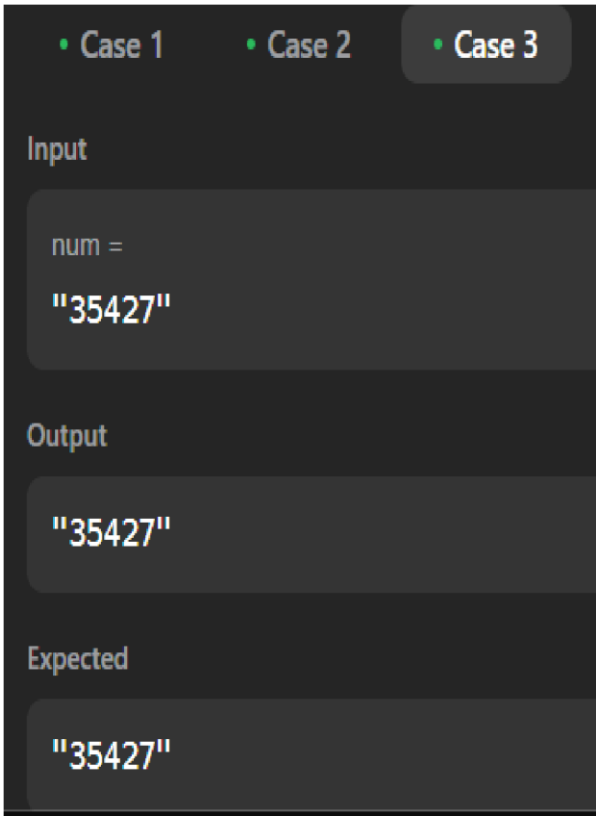
LeetCode Problem- Largest Odd Number in a String You are given a string num, representing a large integer. Return *the **largest-valued odd** integer (as a string) that is a **non-empty substring** of* num*, or an empty string* "" *if no odd integer exists*.

Code:

```
class Solution {
    public:
        string largestOddNumber(string num) {            return
num.substr(0, num.find_last_not_of("0|2|4|6|8") + 1);
        }
    };
```

OUTPUT:





LAB 7:

QUESTION 1:

Implement 0/1 Knapsack problem using dynamic programming.

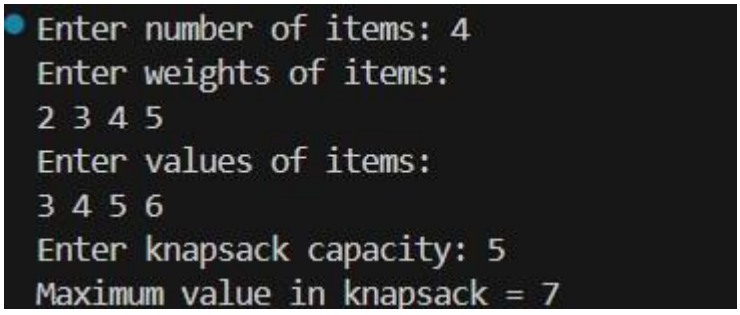
CODE:

```
#include <stdio.h>

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

int knapsack(int weight[], int value[], int n, int capacity) {
int dp[n + 1][capacity + 1];
```

```
    for (int i = 0; i <= n; i++) {         for (int
w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)                 dp[i][w] = 0;             else if (weight[i -
1] <= w)                dp[i][w] = max(value[i - 1] + dp[i - 1][w - weight[i - 1]],
dp[i - 1][w]);
            else                dp[i][w]
= dp[i - 1][w];
        }
    }

    return dp[n][capacity];
}

int main() {     int
n, capacity;

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

    int weight[n], value[n];

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

    printf("Enter values of items:\n");
```

```
    for (int i = 0; i < n; i++)
scanf("%d", &value[i]);

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

    int maxValue = knapsack(weight, value, n, capacity);
printf("Maximum value in knapsack = %d\n", maxValue);

    return 0;
}
```

OUPUT:



QUESTION 2:

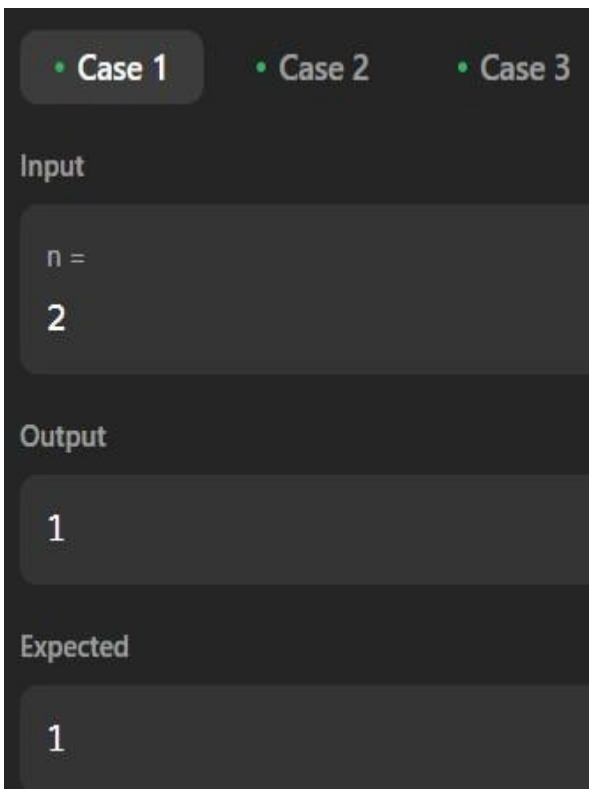
LeetCode Problemm : Fibonacci Number

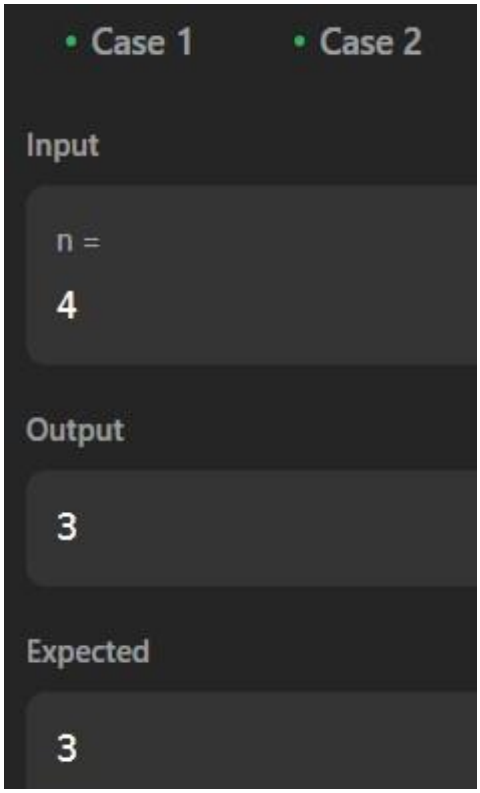
CODE:

```
int fib(int n){
if (n <= 1)
return n;

    // Array to store Fibonacci numbers      int
f[n + 1]; // 1 extra to handle case n = 0
```

```
    // First two Fibonacci numbers
    f[0] = 1;
f[1] = 1;

    // Build the Fibonacci sequence
    for (int i = 2; i <= n; i++) {
        f[i] = f[i - 1] + f[i - 2];
    }

    return f[n - 1];
}
```

OUTPUT:





LAB 8:

QUESTION 1:

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

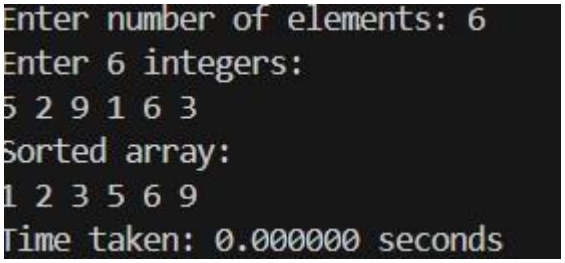
CODE:

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

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

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

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

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

    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
scanf("%d", &arr[i]);

    clock_t start, end;
start = clock();
heapSort(arr, n);     end
= clock();

    printf("Sorted array:\n");
for (int i = 0; i < n; i++)
```

```
printf("%d ", arr[i]);
printf("\n");

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken: %.6f seconds\n", time_taken);


    return 0;
}
```

OUTPUT:



QUESTION 2:

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

CODE:

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

#define INF 99999
#define MAX 100

void floydWarshall(int graph[MAX][MAX], int n) {
int dist[MAX][MAX];
```

```
   for (int i = 0; i < n; i++) {         for
(int j = 0; j < n; j++) {             if
(graph[i][j] == 0 && i != j) {
dist[i][j] = INF; // No edge between
i and j

          } else {                dist[i][j] = graph[i][j]; // Set
initial distances             }
       }
   }



   for (int k = 0; k < n; k++) {         for (int i = 0; i < n; i++) {             for (int j = 0; j
< n; j++) {                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j]
< dist[i][j]) {
                  dist[i][j] = dist[i][k] + dist[k][j]; // Update the distance if shorter path is
found
              }
          }
       }
   }


   printf("Shortest distances between every pair of vertices:\n");
   for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (dist[i][j] == INF) {
printf("INF ");
```

```
            } else {
printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}

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

    int graph[MAX][MAX];

    printf("Enter the adjacency matrix (0 if no edge, for diagonal enter 0):\n");
    for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
scanf("%d", &graph[i][j]);
        }
    }

    floydWarshall(graph, n);

    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter the adjacency matrix (0 if no edge, for diagonal enter 0):
0 3 0 0
3 0 1 0
0 1 0 7
0 0 7 0
Shortest distances between every pair of vertices:
0 3 4 11
3 0 1 8
4 1 0 7
11 8 7 0
```

QUESTION 3:

LeetCode Problem :Shortest Path Visiting All Nodes

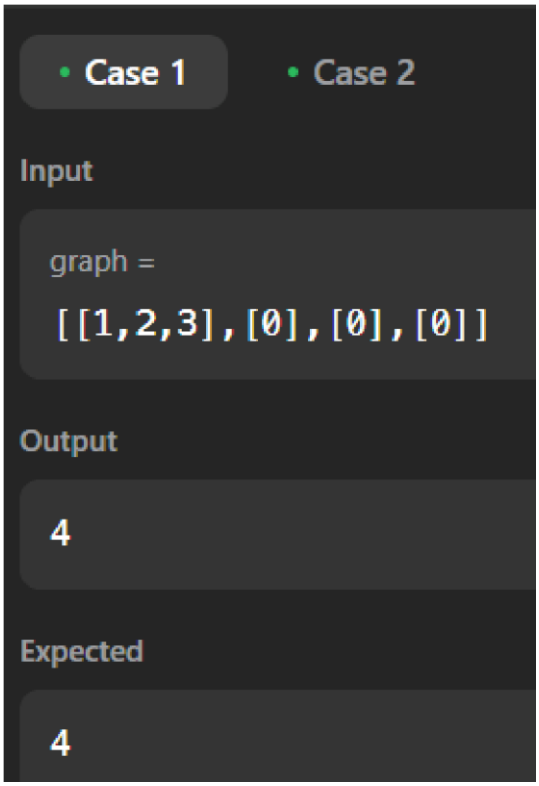
CODE:

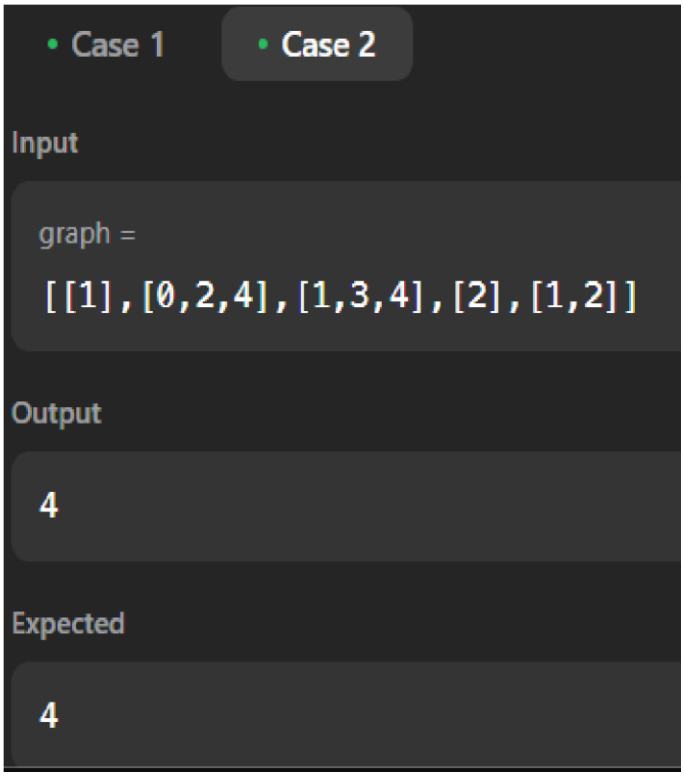
```
struct Node {
    int id;
    int mask;
    int level;
};

struct Queue {     struct
Node data[10000];
    int i;
int j;
    int size;
};
```

```
void qpush(struct Queue* q, struct Node n) {
q->size ++;     q->data[q->i++] = n;     q->i =
q->i % 10000;
}

struct Node qpop(struct Queue* q) {
struct Node r = q->data[q->j++];
    q->size --;
    q->j = q->j % 10000;
    return r;
}

int qempty(struct Queue* q) {
return q->size == 0;
}

int shortestPathLength(int** graph, int graphSize, int* graphColSize) {
struct Queue q;     char V[(1<<(13+4)) + 100] = {0}; // visited
memset(&q, 0, sizeof(q));
    for (int i = 0; i < graphSize; i++) {
struct Node n = {i, 1 << i, 1};
qpush(&q, n);
    }

    while (!qempty(&q)) {         struct
Node n = qpop(&q);         if (n.mask ==
(1<<graphSize)-1) {
```

```
            return n.level-1;
        }

        for (int i = 0; i < graphColSize[n.id]; i++) {
int mask = n.mask;
            int jd = graph[n.id][i];
mask |= 1<<jd;

            struct Node m = {jd, mask, n.level+1};

            if (!V[(jd << 13) | mask]) {
V[(jd << 13) | mask] = 1;
qpush(&q, m);
            }
        }
    }
    return -1;
}
```

OUTPUT:





LAB 9:

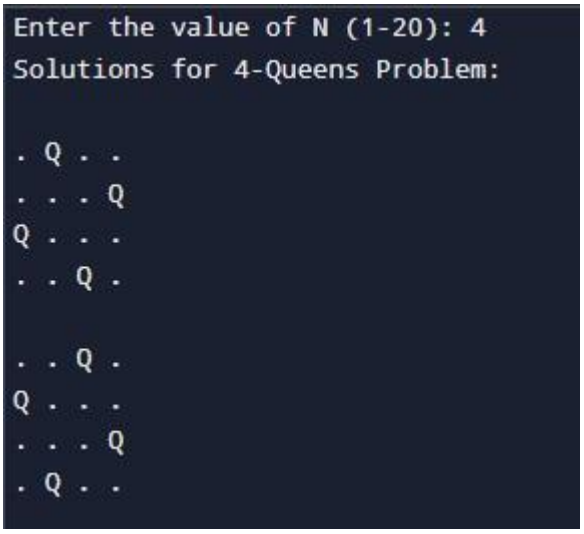
QUESTION 1:

Implement “N-Queens Problem” using Backtracking.

CODE:

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

#define MAX 20

int board[MAX];
int N;
```

```
bool isSafe(int row, int col) {
for (int i = 0; i < row; i++) {
if (board[i] == col ||
board[i] - i == col - row ||
board[i] + i == col + row)
return false;
   }
   return true;
}

void solveNQueens(int row) {
   if (row == N) {        for (int i =
0; i < N; i++) {           for (int j =
0; j < N; j++) {               if
(board[i] == j)
printf("Q ");              else
printf(". ");
           }
printf("\n");        }
       printf("\n");
return;
   }

   for (int col = 0; col < N; col++) {
       if (isSafe(row, col)) {
board[row] = col;
solveNQueens(row + 1);
       }
```

```
    }
}

int main() {     printf("Enter the value of N (1-
%d): ", MAX);     scanf("%d", &N);

    if (N < 1 || N > MAX) {         printf("Invalid input. N should be
between 1 and %d.\n", MAX);
        return 1;
    }

    printf("Solutions for %d-Queens Problem:\n\n", N);
solveNQueens(0);

    return 0;
}
```

OUTPUT: