VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS
(23CS4PCADA)

Submitted by

MAKADIA RISHIT DILIPBHAI (1BM23CS177)

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

February-May 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **“ANALYSIS AND DESIGN OF ALGORITHMS”** carried out by MAKADIA RISHIT DILIPBHAI **(1BM23CS177)**, 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 year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and Design of Algorithms Lab - **(23CS4PCADA)** work prescribed for the said degree.

Prof. Namratha M	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

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.

Program -1(a)

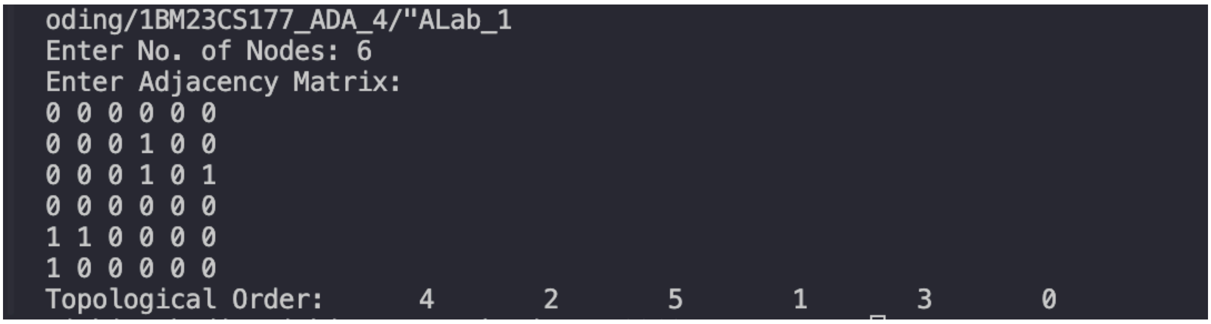
Question: Write a program to obtain the Topological ordering of vertices in a given digraph.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int adj[MAX][MAX], visited[MAX], result[MAX], top=-1;

void dfs_top(int n, int adj[][MAX]){

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

      visited[i]=0;

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

      if(visited[j]==0)

         dfs(j,n,adj);

   }

}

void dfs(int start, int n, int adj[][MAX]){

   visited[start]=1;

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

      if(adj[start][i]==1 && visited[i]==0)

         dfs(i,n,adj);

   }

   result[++top]=start;

}

int main(){

   int n;

   printf("Enter No. of Nodes: ");

   scanf("%d", &n);
```

```
    printf("Enter Adjacency Matrix: \n");
    for(int i=0; i<n;i++){
        for (int j=0; j<n; j++){
            scanf("%d", &adj[i][j]);
        }
    }
    dfs_top(n, adj);
    printf("Topological Order: ");
    for(int k=(n-1) ; k>=0; k--){
        printf("\t%d", result[k]);
    }
}
```

Output:



```
oding/1BM23CS177_ADA_4/"ALab_1
Enter No. of Nodes: 6
Enter Adjacency Matrix:
0 0 0 0 0 0
0 0 0 1 0 0
0 0 0 1 0 1
0 0 0 0 0 0
1 1 0 0 0 0
1 0 0 0 0 0
Topological Order:      4       2       5       1       3       0
```

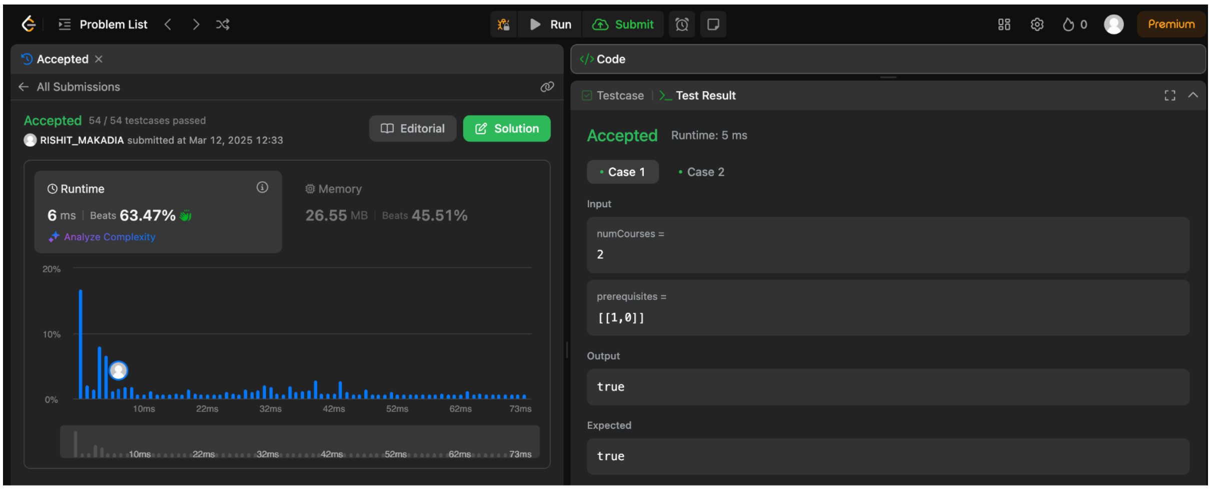
Program - 1(b)

Question: (Leetcode) Course Schedule

Code:

```
#define MAX 2000

bool dfs(int course, int adjList[][MAX], int* adjSize, int* visited){

   if (visited[course]==1) // cycle detected

      return true;

   if (visited[course]==2) // already visited

      return false;

   visited[course]=1;

    for (int i = 0; i < adjSize[course]; i++) {

      if (dfs(adjList[course][i], adjList, adjSize, visited)) {

         return true;

      }

   }

   visited[course] = 2;

   return false;

}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize) {

   int adjL[MAX][MAX];

   int adjSize[MAX]={0};

   int visited[MAX]={0};


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

      int course = prerequisites[i][0];

      int required = prerequisites[i][1];

      adjL[required][adjSize[required]++] = course;
    }

    for(int j=0; j< numCourses; j++){
        if(visited[j] == 0 && dfs(j, adjL, adjSize, visited))
            return false;
    }
    return true;
}
```

Result:



Accepted

Accepted 54 / 54 testcases passed

RISHIT_MAKADIA submitted at Mar 12, 2025 12:33

Runtime

6 ms | Beats 63.47%

Analyze Complexity

Memory

26.55 MB | Beats 45.51%

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

numCourses =

2

prerequisites =

[[1,0]]

Output

true

Expected

true

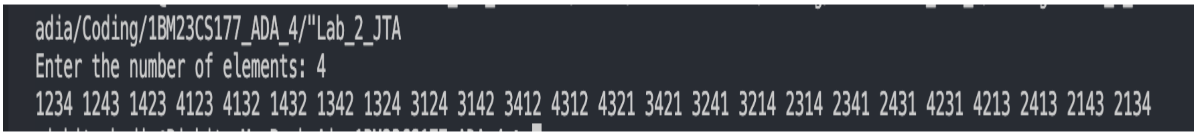
Program - 2

Question: Implement Johnson Trotter algorithm to generate permutations.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define LEFT_TO_RIGHT true

#define RIGHT_TO_LEFT false

// Function to get the position of the mobile integer

int searchArr(int a[], int n, int mobile) {

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

        if (a[i] == mobile)

            return i + 1;

    return -1;

}

// Function to get the largest mobile integer

int getMobile(int a[], bool dir[], int n) {

    int mobile_prev = 0, mobile = 0;

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

        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {

            if (a[i] > a[i - 1] && a[i] > mobile_prev) {

                mobile = a[i];

                mobile_prev = mobile;

            }

        }

        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {

            if (a[i] > a[i + 1] && a[i] > mobile_prev) {

                 mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }
    return mobile;
}
// Function to print one permutation and update the array
void printOnePerm(int a[], bool dir[], int n) {
    int mobile = getMobile(a, dir, n);
    if (mobile == 0) return;
    int pos = searchArr(a, n, mobile) - 1;
    if (dir[a[pos] - 1] == RIGHT_TO_LEFT)
        // Swap with the left element
        {
            int temp = a[pos];
            a[pos] = a[pos - 1];
            a[pos - 1] = temp;
        }
    else if (dir[a[pos] - 1] == LEFT_TO_RIGHT)
        // Swap with the right element
        {
            int temp = a[pos];
            a[pos] = a[pos + 1];
            a[pos + 1] = temp;
        }
    // After swapping, change the directions of all elements greater than mobile
    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            dir[a[i] - 1] = !dir[a[i] - 1];
        }
    }
    // Print current permutation
    for (int i = 0; i < n; i++)
        printf("%d", a[i]);
    printf(" ");
}
// Factorial function
int fact(int n) {
    int res = 1;
    for (int i = 1; i <= n; i++)
        res *= i;
    return res;
}
// Function to print all permutations using Johnson-Trotter algorithm
void printPermutation(int n) {
    int a[n];
    bool dir[n];
    // Initialize the array and direction
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        dir[i] = RIGHT_TO_LEFT;
        printf("%d", a[i]);
    }
    printf(" ");
    // Generate and print all permutations
    for (int i = 1; i < fact(n); i++) {
        printOnePerm(a, dir, n);
    }
    printf("\n");
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printPermutation(n);
    return 0;
}
```

Result:



```
adia/Coding/1BM23CS177_ADA_4/"Lab_2_JTA
Enter the number of elements: 4
1234 1243 1423 4123 4132 1432 1342 1324 3124 3142 3412 4312 4321 3421 3241 3214 2314 2341 2431 4231 4213 2413 2143 2134
```

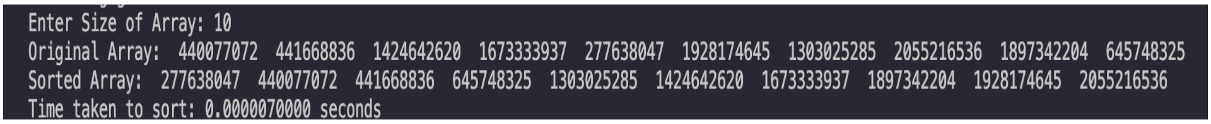
Program - 3 (a)

Question: 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>

#define MAX 100000

void merge(int a[], int low, int high, int mid);

void mergeSort(int arr[], int low, int high);

void printArray(int arr[], int no);

void merge(int a[], int low, int high, int mid)

{

  int i = low, j = mid + 1, k = 0;

  int temp[high - low + 1];

  while (i <= mid && j <= high)

  {

    if (a[i] <= a[j])

      temp[k++] = a[i++];

    else

      temp[k++] = a[j++];

  }

  while (i <= mid)

    temp[k++] = a[i++];

  while (j <= high)

    temp[k++] = a[j++];


  for (int i = low, k = 0; i <= high; i++, k++)

  {
        a[i] = temp[k];
    }
}
void mergeSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int mid = (high + low) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, high, mid);
    }
}

void printArray(int arr[], int no)
{
    for (int i = 0; i < no; i++)
    {
        printf(" %d ", arr[i]);
    }
    printf("\n");
}
void main()
{
    int N;
    printf("Enter Size of Array: ");
    scanf("%d", &N);
    int array[N];
    srand(time(NULL));
    for (int i = 0; i < N; i++) {
        array[i] = rand(); // Random integers between 0 and 99
    }
    printf("Original Array: ");
    printArray(array, N);
    clock_t start_time = clock();
    mergeSort(array, 0, N - 1);
    clock_t end_time = clock();
    double time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Sorted Array: ");
    printArray(array, N);
    printf("Time taken to sort: %.100f seconds\n", time_taken);
}
```

Result:



```
Enter Size of Array: 10
Original Array:  440077072  441668836  1424642620  1673333937  277638047  1928174645  1303025285  2055216536  1897342204  645748325
Sorted Array:  277638047  440077072  441668836  645748325  1303025285  1424642620  1673333937  1897342204  1928174645  2055216536
Time taken to sort: 0.0000070000 seconds
```

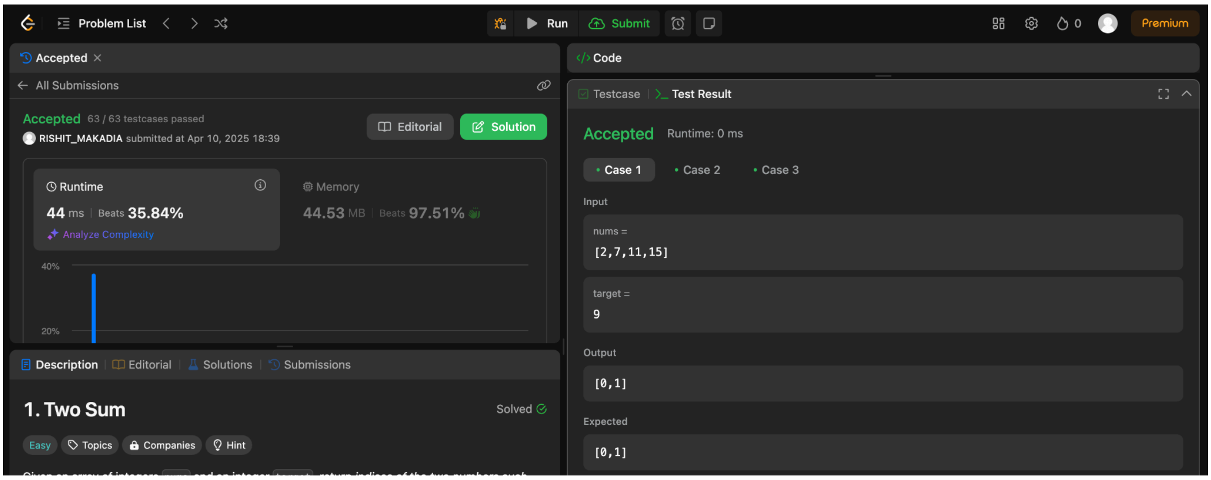
Program - 3 (b)

Question: LeetCode Program related to sorting.

Code:

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] ans = new int[2];
        for(int i=0;i<nums.length;i++){
            for(int j=(i+1) ; j< nums.length; j++){
                if ((nums[i]+nums[j]) == target){
                    ans[0]=i;
                    ans[1]=j;
                    return ans;
                }
            }
        }
        return ans;
    }
}
```

Result:



Accepted 63 / 63 testcases passed

RISHIT_MAKADIA submitted at Apr 10, 2025 18:39

Runtime
44 ms | Beats 35.84%

Memory
44.53 MB | Beats 97.51%

1. Two Sum

Accepted Runtime: 0 ms

Input

```
nums =
[2,7,11,15]
```

```
target =
9
```

Output

```
[0,1]
```

Expected

```
[0,1]
```

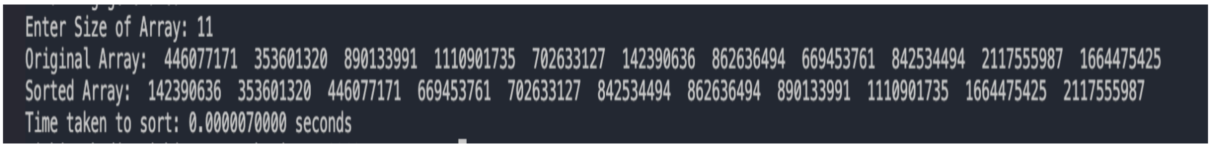
Program - 4 (a)

Question: 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>
#define MAX 100000000
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void quickSort(int arr[], int low, int high){
    if (low < high)
    {
        int par = partition(arr, low, high);
        quickSort(arr, low, par-1);
        quickSort(arr, par + 1, high);
    }
}
int partition(int a[], int low, int high){
    int pivot = a[high];
    int i=low-1;
    for (int j=low; j< high; j++){
        if (a[j]<pivot){
            i++;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i+1], &a[high]);
    return i+1;
}
void printArray(int arr[], int no){
    for (int i=0; i<no; i++)
        printf(" %d ", arr[i]);
    printf("\n");
}

void main(){
    int N;
    printf("Enter Size of Array: ");
    scanf("%d", &N);
    int array[N];
    srand(time(NULL));
    for (int i = 0; i < N; i++) {
        array[i] = rand();
    }
    printf("Original Array: ");
    printArray(array, N);
    clock_t start_time = clock();
    quickSort(array, 0, N - 1);
    clock_t end_time = clock();
    double time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Sorted Array: ");
    printArray(array, N);
    printf("Time taken to sort: %.10f seconds\n", time_taken);
}
```

Result:



```
Enter Size of Array: 11
Original Array:  446077171  353601320  890133991  1110901735  702633127  142390636  862636494  669453761  842534494  2117555987  1664475425
Sorted Array:  142390636  353601320  446077171  669453761  702633127  842534494  862636494  890133991  1110901735  1664475425  2117555987
Time taken to sort: 0.0000070000 seconds
```

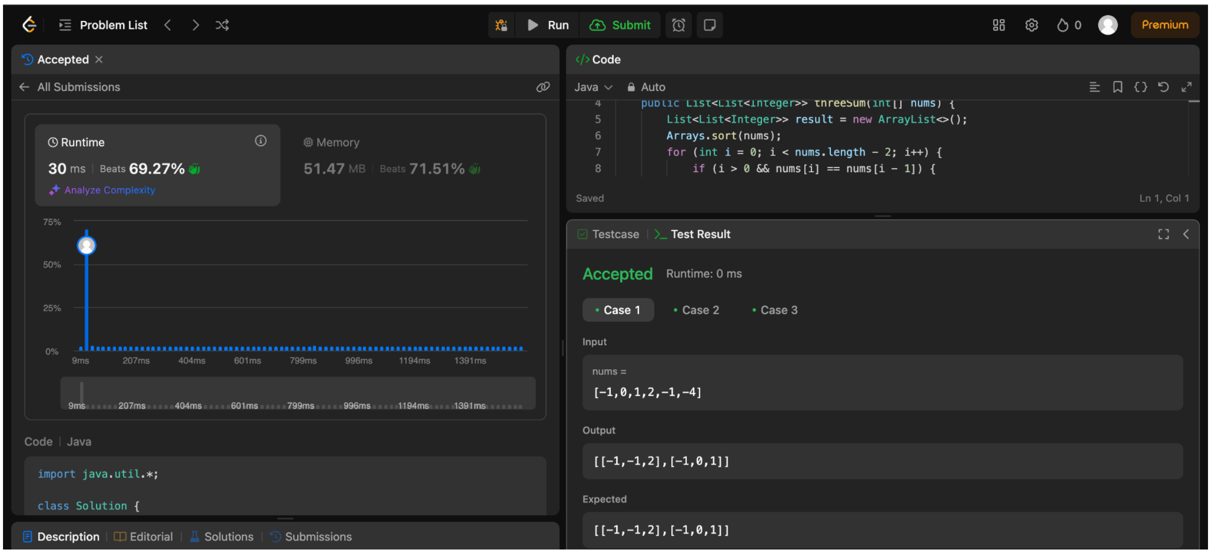
Program - 4 (b)

Question: LeetCode Program related to sorting.

Code:

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);
        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }
            int left = i + 1, right = nums.length - 1;

            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];

                if (sum == 0) {
                    result.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    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;
    }
}
```

Result:



Runtime

30 ms | Beats 69.27%

Memory

51.47 MB | Beats 71.51%

Code | Java

```
import java.util.*;

class Solution {
```

```
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);
        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) {
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =

```
[-1,0,1,2,-1,-4]
```

Output

```
[[-1,-1,2],[-1,0,1]]
```

Expected

```
[[-1,-1,2],[-1,0,1]]
```

Program - 5

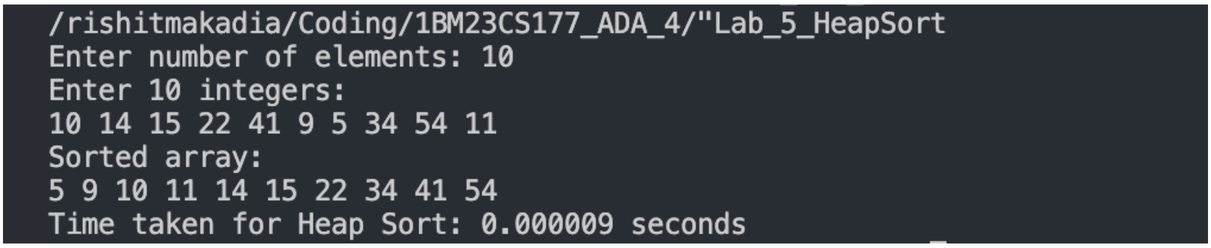
Question: Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void heapify(int arr[], int n, int i) {
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   // If left child is larger than root
   if (left < n && arr[left] > arr[largest])
      largest = left;
   // If right child is larger than current largest
   if (right < n && arr[right] > arr[largest])
      largest = right;
   // If largest is not root
   if (largest != i) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;
      heapify(arr, n, largest);
   }
}
void heapSort(int arr[], int n) {
   // Build heap (rearrange array)
   for (int i = n / 2 - 1; i >= 0; i--)
      heapify(arr, n, i);
```

```
    // One by one extract elements from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        // Call max heapify on the reduced heap
        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 = clock();
    heapSort(arr, n);
    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("\n");
    printf("Time taken for Heap Sort: %.6f seconds\n", time_taken);
}
```

Result:



```
/rishitmakadia/Coding/1BM23CS177_ADA_4/"Lab_5_HeapSort
Enter number of elements: 10
Enter 10 integers:
10 14 15 22 41 9 5 34 54 11
Sorted array:
5 9 10 11 14 15 22 34 41 54
Time taken for Heap Sort: 0.000009 seconds
```

Program - 6 (a)

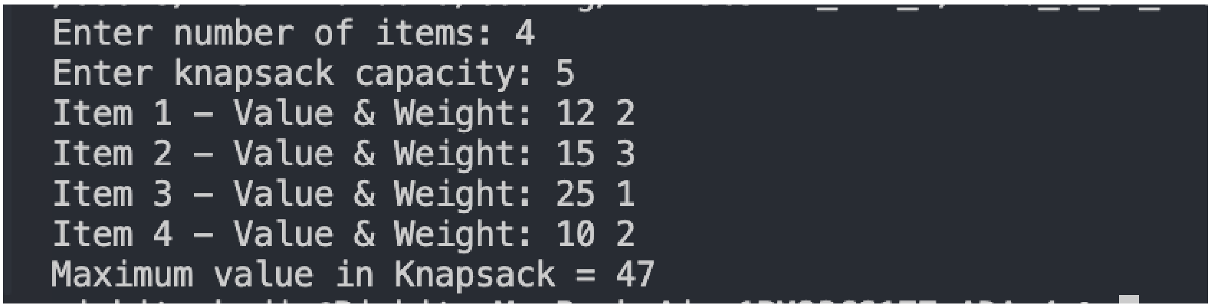
Question: Implement 0/1 Knapsack problem using dynamic programming.

Code:

```
#include <stdio.h>
int max(int a, int b) {
   return (a > b) ? a : b;
}
int knapsack(int capacity, int weights[], int values[], int n) {
   int i, w;
   int dp[n+1][capacity+1];
   // Build table dp[][] in bottom-up manner
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= capacity; w++) {
         if (i == 0 || w == 0)
            dp[i][w] = 0;
         else if (weights[i-1] <= w)
            dp[i][w] = max(values[i-1] + dp[i-1][w - weights[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);
   printf("Enter knapsack capacity: ");
   scanf("%d", &capacity);
```

```
    int values[n], weights[n];
    for (int i = 0; i < n; i++) {
        printf("Item %d - Value & Weight: ", i + 1);
        scanf("%d %d", &values[i], &weights[i]);
    }
    int maxValue = knapsack(capacity, weights, values, n);
    printf("Maximum value in Knapsack = %d\n", maxValue);
    return 0;
}
```

Result:



```
Enter number of items: 4
Enter knapsack capacity: 5
Item 1 - Value & Weight: 12 2
Item 2 - Value & Weight: 15 3
Item 3 - Value & Weight: 25 1
Item 4 - Value & Weight: 10 2
Maximum value in Knapsack = 47
```

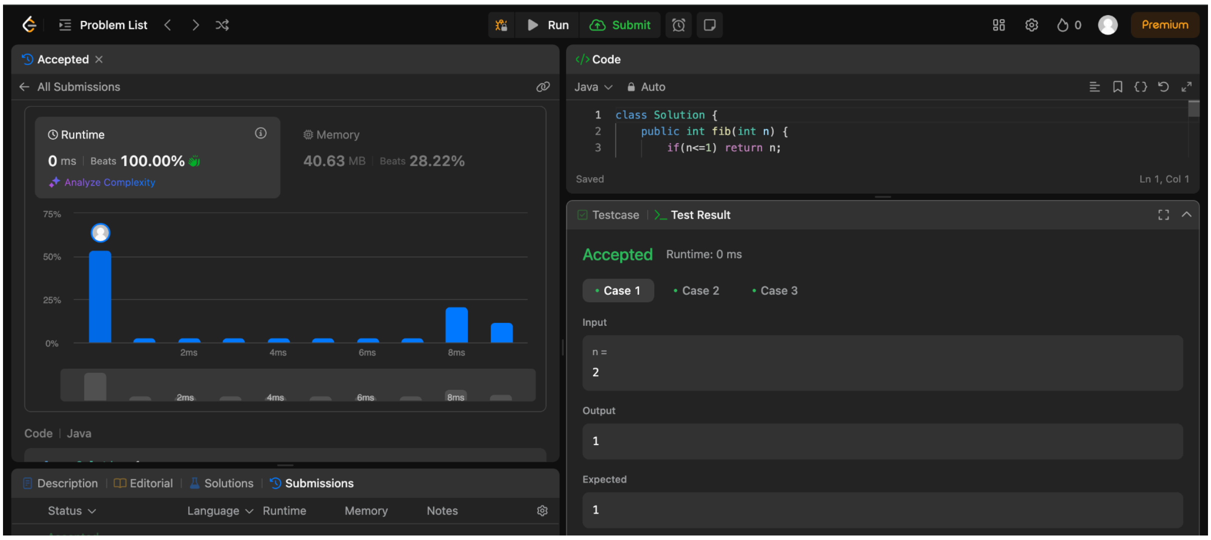
Program - 6 (b)

Question: LeetCode Program related to Knapsack problem or Dynamic Programming.

Code:

```
class Solution {
    public int fib(int n) {
        if(n<=1) return n;
        int[] dp = new int[n+1];
        dp[0]=0;
        dp[1]=1;
        for(int i =2 ; i<=n; i++){
            dp[i]=dp[i-1]+dp[i-2];
        }
        return dp[n];
    }
}
```

Result:



Accepted

All Submissions

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

40.63 MB | Beats 28.22%

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

Saved

Ln 1, Col 1

Accepted Runtime: 0 ms

Case 1 • Case 2 • Case 3

Input

n =
2

Output

1

Expected

1

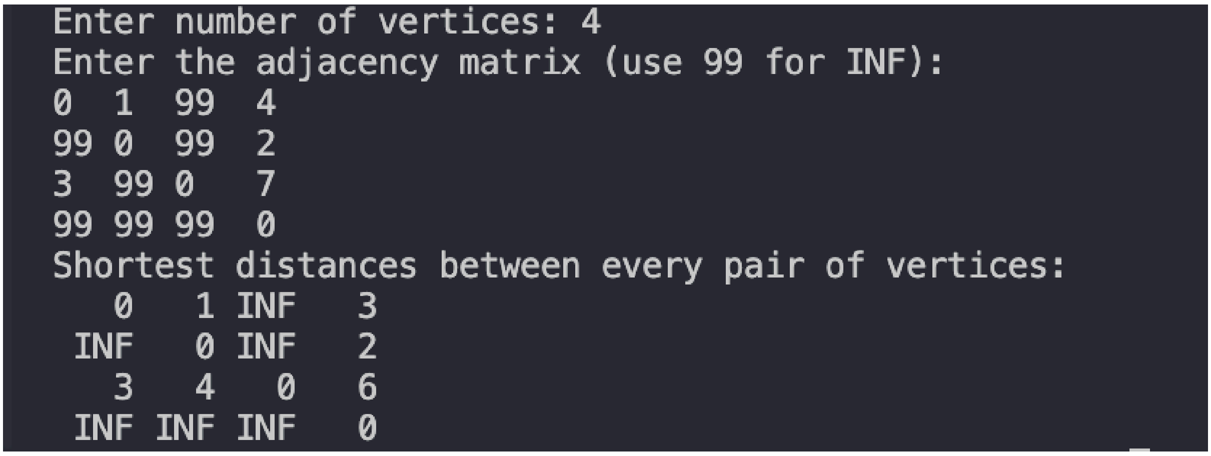
Program - 7 (a)

Question: Implement All Pair Shortest paths problem using Floyd's algorithm.

Code:

```
#include <stdio.h>
#define INF 99    //99 for infinity
#define V 100
void printSolution(int dist[][V], int n) {
   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("%4s", "INF");
         else
            printf("%4d", dist[i][j]);
      }
      printf("\n");
   }
}
void floyd(int graph[][V], int n) {
   int dist[V][V];
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         dist[i][j] = graph[i][j];
   // Main loop
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         for (int k = 0; k < n; k++) {
            if (dist[j][i] != INF && dist[i][k] != INF && dist[j][i] + dist[i][k] < dist[j][k])
               dist[j][k] = dist[j][i] + dist[i][k];
            }
        }
    }
    printSolution(dist, n);
}

int main() {
    int n;
    int graph[V][V];
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (use 99 for INF):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);
    floyd(graph, n);
}
```

Result:



```
Enter number of vertices: 4
Enter the adjacency matrix (use 99 for INF):
0  1  99  4
99 0  99  2
3  99 0   7
99 99 99  0
Shortest distances between every pair of vertices:
   0   1 INF   3
 INF   0 INF   2
   3   4   0   6
 INF INF INF   0
```

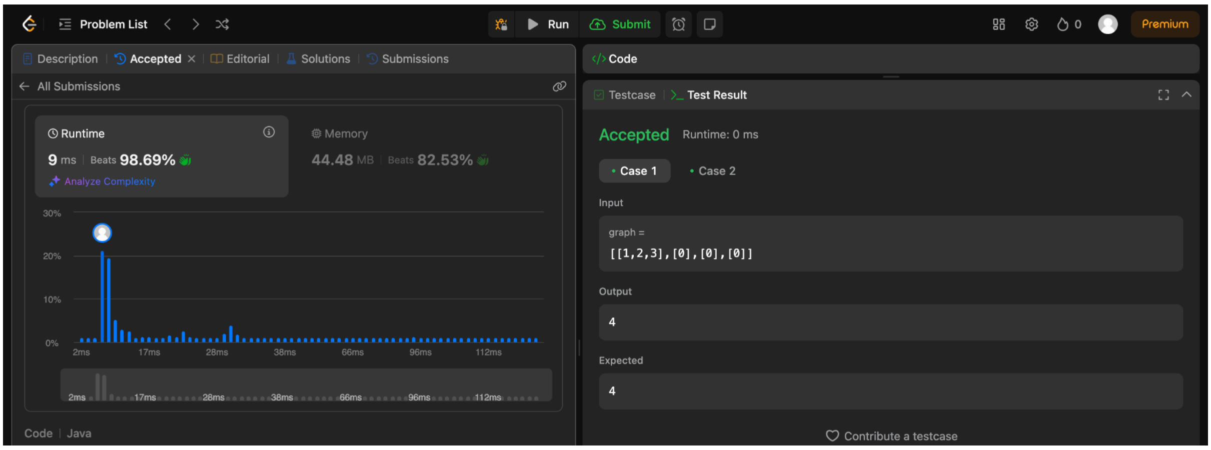
Program - 7 (b)

Question: Shortest Path visiting All Nodes

Code:

```
class Solution {
    public int shortestPathLength(int[][] graph) {
        int n = graph.length;
        if (n == 1) return 0;

        int target = (1 << n) - 1; // All nodes visited (all bits set)
        Queue<int[]> queue = new LinkedList<>();
        boolean[][] visited = new boolean[n][1 << n];

        // Initialize queue with all nodes as starting points
        for (int i = 0; i < n; i++) {  // Fixed: Removed illegal character
            queue.offer(new int[]{i, 1 << i});
            visited[i][1 << i] = true;
        }

        int steps = 0;
        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                int[] current = queue.poll();
                int node = current[0];
                int state = current[1];

                if (state == target) {
                    return steps;
                }
                // Visit all neighbors
                for (int neighbor : graph[node]) {
                    int newState = state | (1 << neighbor);
                    if (!visited[neighbor][newState]) {
                        visited[neighbor][newState] = true;
                        queue.offer(new int[]{neighbor, newState});
                    }
                }
            }
            steps++;
        }
        return -1; // Shouldn't reach here for connected graph
    }
}
```

Result:



Runtime

9 ms | Beats 98.69%

Memory

44.48 MB | Beats 82.53%

Accepted Runtime: 0 ms

Input

graph =

```
[[1,2,3],[0],[0],[0]]
```

Output

```
4
```

Expected

```
4
```

Program - 8 (a)

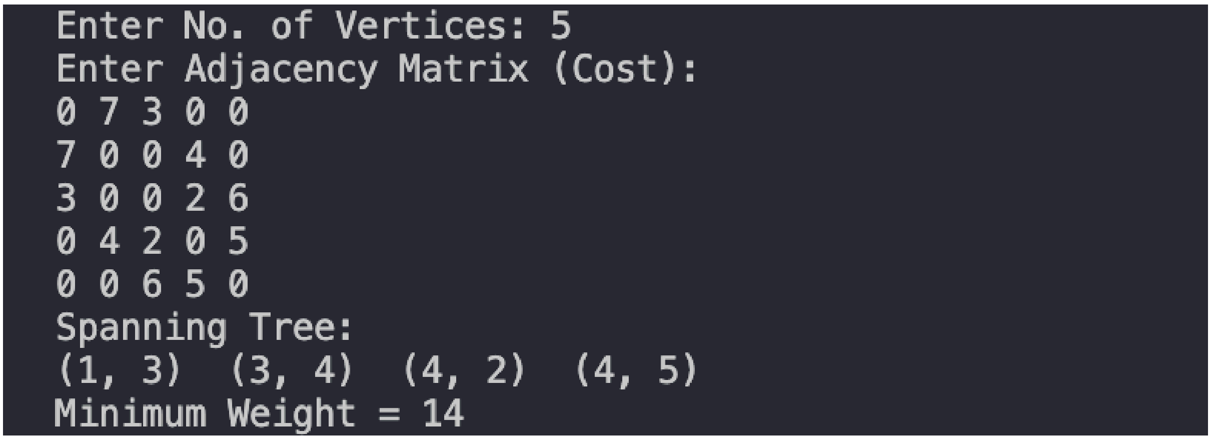
Question: Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Code:

```
#include <stdio.h>
int cost[10][10], vt[10], et[10][2], vis[10], n;
int sum = 0, x = 1, e = 0;
void prims() {
  int m, j, min, u, v, k;
  vt[x] = 1;
  vis[1] = 1;
  for (int s = 1; s < n; s++) {
    min = 999;
    j = x;
    while (j > 0) {
      k = vt[j];
      for (m = 1; m <= n; m++) {
        if (vis[m] == 0 && cost[k][m] != 0 && cost[k][m] < min) {
          min = cost[k][m];
          u = k;
          v = m;
        }
      }
      j--;
    }
    vt[++x] = v;
    e++;
    vis[v] = 1;
    sum += min;
```

```
        et[e][0] = u;
        et[e][1] = v;
    }
}

void main() {
    printf("Enter No. of Vertices: ");
    scanf("%d", &n);
    printf("Enter Adjacency Matrix (Cost):\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0 && i != j) {
                cost[i][j] = 999;
            }
        }
        vis[i] = 0;
    }
    prims();
    printf("Spanning Tree:\n");
    for (int i = 1; i <= e; i++) {
        printf("(%d, %d)\t", et[i][0], et[i][1]);
    }
    printf("\nMinimum Weight = %d\n", sum);
}
```

Result:



```
Enter No. of Vertices: 5
Enter Adjacency Matrix (Cost):
0 7 3 0 0
7 0 0 4 0
3 0 0 2 6
0 4 2 0 5
0 0 6 5 0
Spanning Tree:
(1, 3)  (3, 4)  (4, 2)  (4, 5)
Minimum Weight = 14
```

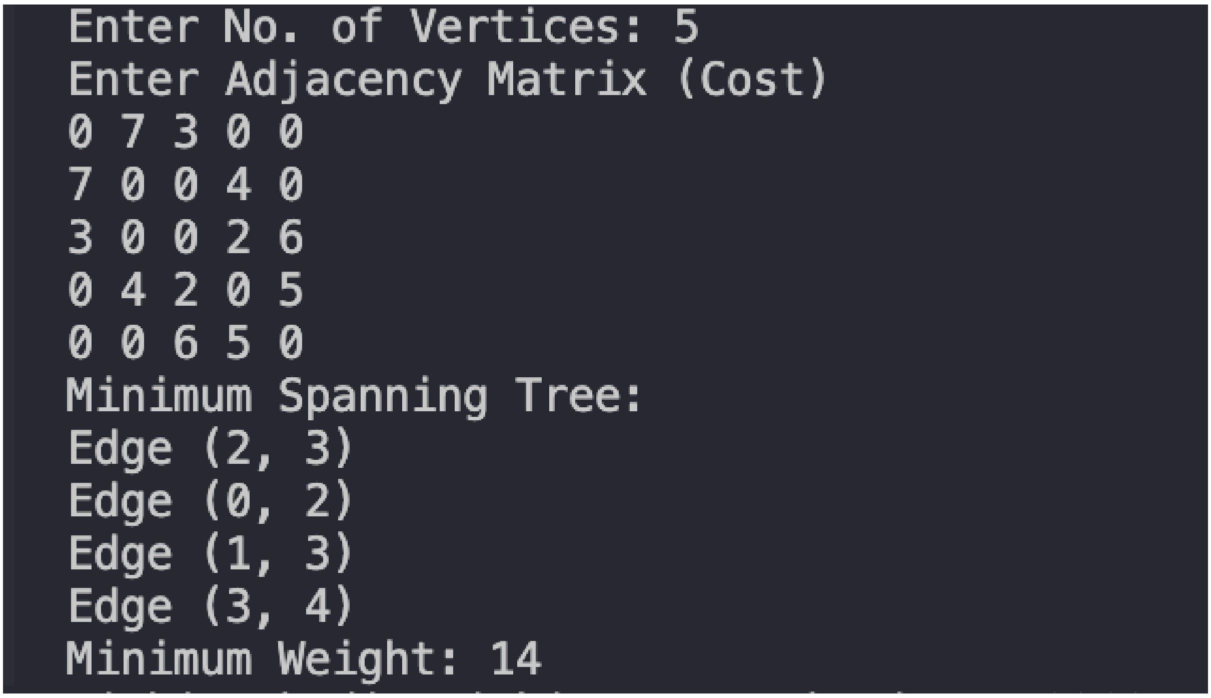
Program - 8 (b)

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

Code:

```
#include <stdio.h>
void unionn(int i, int j, int parent[]) {
    if (i < j)
        parent[j] = i;
    else
        parent[i] = j;
}
int find(int v, int parent[]) {
    while (parent[v] != v)
        v = parent[v];
    return v;
}
void kruskal(int n, int a[10][10]) {
    int count = 0, k = 0, sum = 0;
    int min, i, j, u, v, parent[10], temp[10][2];
    for (int l = 0; l < n; l++)
        parent[l] = l;
    while (count < n - 1) {
        min = 999;
        for (int i = 0; i < n; i++) {
            for (int 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) {
            unionn(i, j, parent);
            temp[k][0] = u;
            temp[k][1] = v;
            sum += a[u][v];
            count++;
            k++;
        }
        a[u][v] = a[v][u] = 999;
    }
    if (count == n - 1) {
        printf("Minimum Spanning Tree:\n");
        for (int m = 0; m < k; m++)
            printf("Edge (%d, %d)\n", temp[m][0], temp[m][1]);
        printf("Minimum Weight: %d\n", sum);
    } else {
        printf("Not a Spanning Tree\n");
    }
}
void main() {
    int n, i, j, a[10][10];
    printf("Enter No. of Vertices: ");
    scanf("%d", &n);
    printf("Enter Adjacency Matrix (Cost)\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    }
    kruskal(n, a);
}
```

Result:



```
Enter No. of Vertices: 5
Enter Adjacency Matrix (Cost)
0 7 3 0 0
7 0 0 4 0
3 0 0 2 6
0 4 2 0 5
0 0 6 5 0
Minimum Spanning Tree:
Edge (2, 3)
Edge (0, 2)
Edge (1, 3)
Edge (3, 4)
Minimum Weight: 14
```

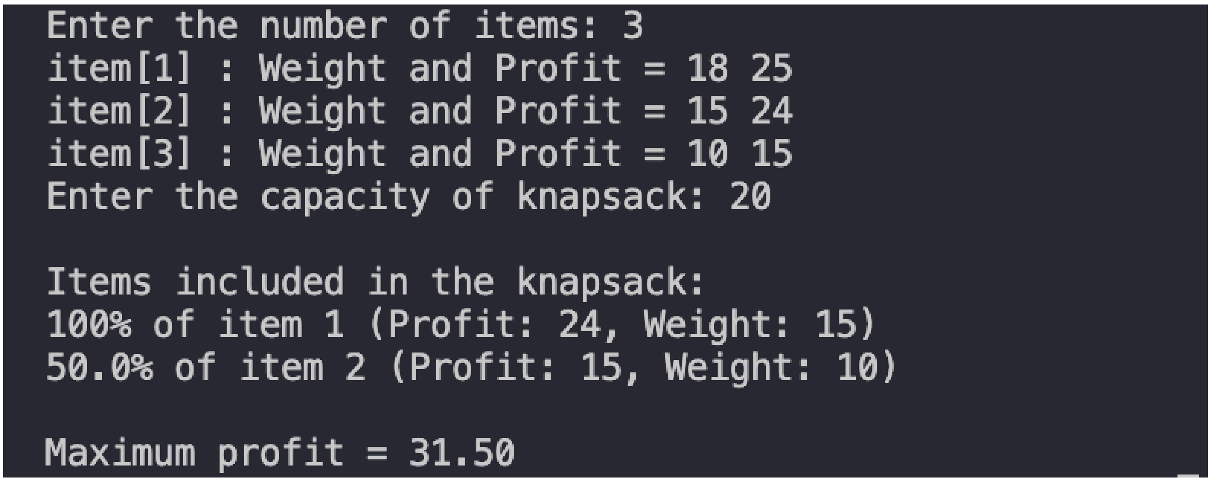
Program - 9 (a)

Question: Implement Fractional Knapsack using Greedy technique.

Code:

```
#include <stdio.h>
int main() {
   int weight[50], profit[50], capacity, n, i, j;
   float pTOw[50], totalValue = 0.0, temp;
   printf("Enter the number of items: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("item[%d] : Weight and Profit = ", i + 1);
      scanf("%d %d", &weight[i], &profit[i]);
   }
   printf("Enter the capacity of knapsack: ");
   scanf("%d", &capacity);
   for (i = 0; i < n; i++) {
      pTOw[i] = (float)profit[i] / weight[i];
   }
   for (i = 0; i < n; i++) {
      for (j = i + 1; j < n; j++) {
         if (pTOw[i] < pTOw[j]) {
            temp = pTOw[j];      pTOw[j] = pTOw[i];           pTOw[i] = temp;
            temp = weight[j];    weight[j] = weight[i];       weight[i] = (int)temp;
            temp = profit[j];    profit[j] = profit[i];       profit[i] = (int)temp;
         }
      }
   }
   printf("\nItems included in the knapsack:\n");
   for (i = 0; i < n; i++) {
        if (weight[i] <= capacity) {
            printf("100%% of item %d (Profit: %d, Weight: %d)\n", i + 1, profit[i], weight[i]);
            totalValue += profit[i];
            capacity -= weight[i];
        } else {
            printf("%.1f%% of item %d (Profit: %d, Weight: %d)\n",
                   ((float)capacity / weight[i]) * 100, i + 1, profit[i], weight[i]);
            totalValue += pTOw[i] * capacity;
            break;
        }
    }
    printf("\nMaximum profit = %.2f\n", totalValue);
    return 0;
}
```

Result:



```
Enter the number of items: 3
item[1] : Weight and Profit = 18 25
item[2] : Weight and Profit = 15 24
item[3] : Weight and Profit = 10 15
Enter the capacity of knapsack: 20

Items included in the knapsack:
100% of item 1 (Profit: 24, Weight: 15)
50.0% of item 2 (Profit: 15, Weight: 10)

Maximum profit = 31.50
```

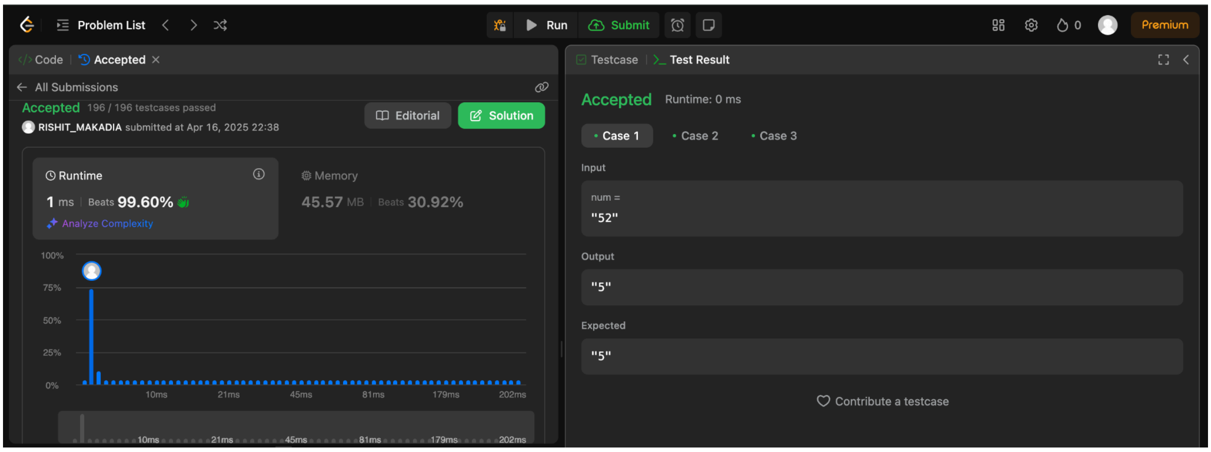
Program - 9 (b)

Question: Largest Odd Number in String

Code:

```
class Solution {
    public String largestOddNumber(String num) {
        int index = num.length() - 1;
        for (int i=index ; i>=0; i--){
            if((num.charAt(i)-'0')%2==1){
                // return num; // Wrong
                return num.substring(0, i+1);
            }
            // num=num.substring(0, i-1); // Wrong
        }
        return "";
    }
}
```

Result:



Accepted 196 / 196 testcases passed

RISHIT_MAKADIA submitted at Apr 16, 2025 22:38

Runtime
1 ms | Beats 99.60%

Memory
45.57 MB | Beats 30.92%

Accepted Runtime: 0 ms

Input

num =
"52"

Output

"5"

Expected

"5"

Program - 10

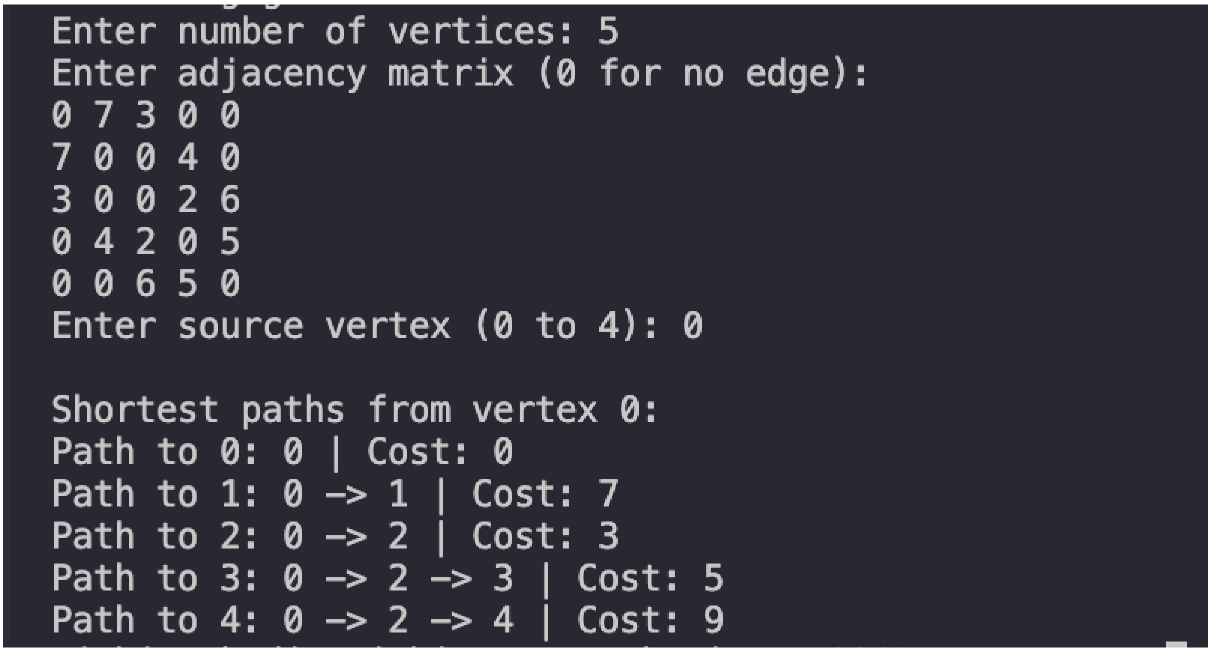
Question: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Code:

```
#include <stdio.h>
#define INF 9999
void printPath(int parent[], int j) {
   if (parent[j] == -1)
      return;
   printPath(parent, parent[j]);
   printf(" -> %d", j);
}
void dijkstra(int n, int cost[10][10], int src) {
   int dist[10], visited[10], parent[10];
   int i, j, u, min;
   // Initialize distances and visited array
   for (i = 0; i < n; i++) {
      dist[i] = cost[src][i];
      visited[i] = 0;
      if (cost[src][i] != INF && i != src)
         parent[i] = src;
      else
         parent[i] = -1;
   }
   dist[src] = 0;           visited[src] = 1;          parent[src] = -1;
   for (i = 1; i < n; i++) {
      min = INF;
      u = -1;
      // Find the nearest unvisited vertex
```

```
        for (j = 0; j < n; j++) {
            if (!visited[j] && dist[j] < min) {
                min = dist[j];
                u = j;
            }
        }
        if (u == -1) break;
        visited[u] = 1;
        // Update distances of adjacent vertices
        for (j = 0; j < n; j++) {
            if (!visited[j] && cost[u][j] != INF && dist[u] + cost[u][j] < dist[j]) {
                dist[j] = dist[u] + cost[u][j];
                parent[j] = u;
            }
        }
    }
    // Output the shortest path and cost
    printf("\nShortest paths from vertex %d:\n", src);
    for (i = 0; i < n; i++) {
        if (dist[i] != INF) {
            printf("Path to %d: %d", i, src);
            printPath(parent, i);
            printf(" | Cost: %d\n", dist[i]);
        } else {
            printf("No path to vertex %d\n", i);
        }
    }
}
void main() {
    int n, i, j, src;
    int cost[10][10];
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix (0 for no edge):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (i != j && cost[i][j] == 0)
                cost[i][j] = INF;
        }
    }
    printf("Enter source vertex (0 to %d): ", n - 1);
    scanf("%d", &src);
    dijkstra(n, cost, src);
}
```

Result:



```
Enter number of vertices: 5
Enter adjacency matrix (0 for no edge):
0 7 3 0 0
7 0 0 4 0
3 0 0 2 6
0 4 2 0 5
0 0 6 5 0
Enter source vertex (0 to 4): 0

Shortest paths from vertex 0:
Path to 0: 0 | Cost: 0
Path to 1: 0 -> 1 | Cost: 7
Path to 2: 0 -> 2 | Cost: 3
Path to 3: 0 -> 2 -> 3 | Cost: 5
Path to 4: 0 -> 2 -> 4 | Cost: 9
```

Program - 11

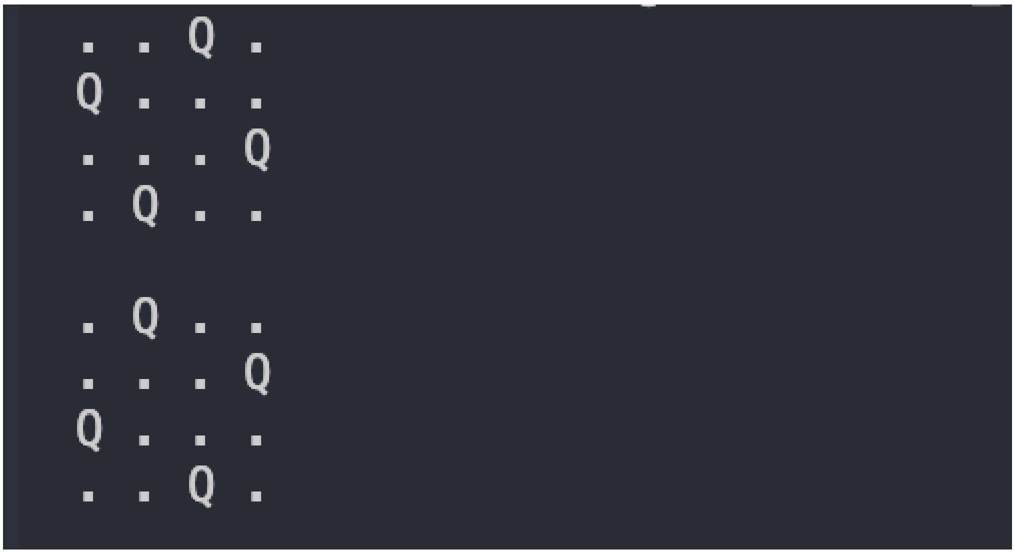
Question: Implement “N-Queens Problem” using Backtracking.

Code:

```
#include<stdio.h>
#include<stdbool.h>
#define N 4
void printSolution(int board[N][N]){
  for(int i=0; i<N;i++){
    for(int j=0; j<N;j++)
      printf("%s", board[i][j]? "Q ":". ");
    printf("\n");
  }
  printf("\n");
}
bool isSafe(int board[N][N], int row,int col){
  for (int i=0; i<col;i++)
    if(board[row][i])
      return false;
  for(int i=row, j=col; i>=0 && j>=0; i--, j--)
    if(board[i][j])
      return false;
  for(int i=row, j=col; i<=N && j>=0; i++, j--)
    if(board[i][j])
      return false;
  return true;
}
void solveNQueen(int board[N][N], int col){
  if(col == N) {
    printSolution(board);
```

```
        return;
    }
    for (int i=0; i<N; i++){
        if(isSafe(board, i, col)){
            board[i][col] = 1;
            solveNQueen(board, col+1);
            board[i][col] = 0; // Backtrack
        }
    }
}
void NQueens(){
    int board[N][N] = {0}; // Initialize board with 0 (no queens)
    solveNQueen(board, 0); // Start solving the N-Queens problem from the first column
}
int main(){
    NQueens();
}
```

Result:



```
. . Q .
Q . . .
. . . Q
. Q . .

. Q . .
. . . Q
Q . . .
. . Q .
```