VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Parth Rawat

1BM22CS190

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 **Parth Rawat(1BM22CS190)**, 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.

Dr.VIKRANTH B 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.

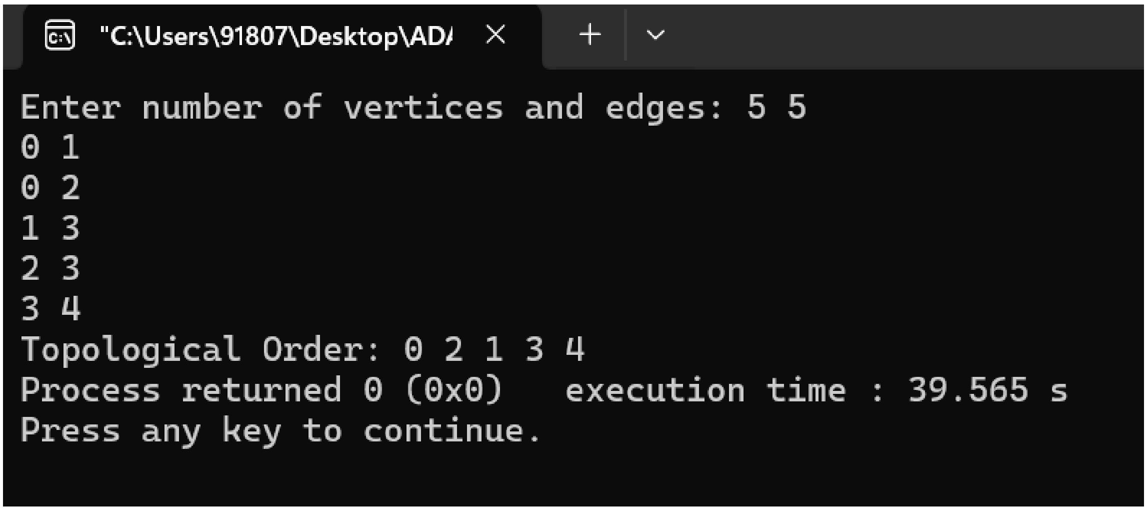
Lab program 1:

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

Code:

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

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

void dfs(int v) {
    visited[v] = 1;
    for (int i = 0; i < n; i++)
        if (adj[v][i] && !visited[i])
            dfs(i);
    stack[++top] = v;
}

void topologicalSort() {
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i);
    while (top >= 0)
        printf("%d ", stack[top--]);
}

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

    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
        adj[u][v] = 1;
    }

    printf("Topological Order: ");
    topologicalSort();
    return 0;
}
```

Output:



```
Enter number of vertices and edges: 5 5
0 1
0 2
1 3
2 3
3 4
Topological Order: 0 2 1 3 4
Process returned 0 (0x0)   execution time : 39.565 s
Press any key to continue.
```

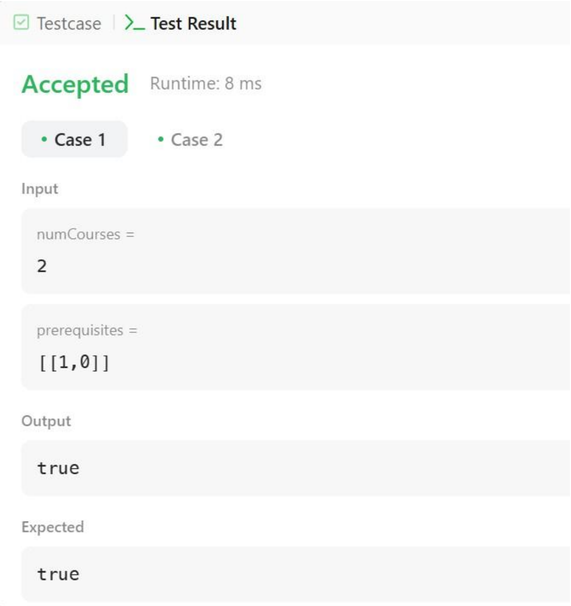
b] LeetCode Program related to Topological sorting

Code:

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

#define MAX_COURSES 2001  // Adjusted limit for safety

bool DFS(int course, int preMap[MAX_COURSES][MAX_COURSES], int numCourses, int
*visitedMap) {
    if (visitedMap[course] == 1) return false; // Cycle detected
    if (visitedMap[course] == 2) return true;  // Already checked

    visitedMap[course] = 1; // Mark as visiting

    for (int next = 0; next < numCourses; next++) {
        if (preMap[course][next]) {  // If there's a prerequisite edge
            if (!DFS(next, preMap, numCourses, visitedMap)) {
                return false;
            }
        }
    }

    visitedMap[course] = 2; // Mark as visited
    return true;
}
```

```
bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize) {
    int preMap[MAX_COURSES][MAX_COURSES] = {0};

    int visitedMap[MAX_COURSES] = {0}; // 0 = unvisited, 1 = visiting, 2 = visited


    memset(preMap, 0, sizeof(preMap));

    for (int i = 0; i < prerequisitesSize; i++) {
        int course = prerequisites[i][0];
        int prereq = prerequisites[i][1];
        preMap[prereq][course] = 1; // prereq → course
    }
    for (int i = 0; i < numCourses; i++) {
        if (visitedMap[i] == 0) {
            if (!DFS(i, preMap, numCourses, visitedMap)) {
                return false; // Cycle detected
            }
        }
    }

    return true;
}
```

Output:



Accepted Runtime: 8 ms

Input

numCourses =
2

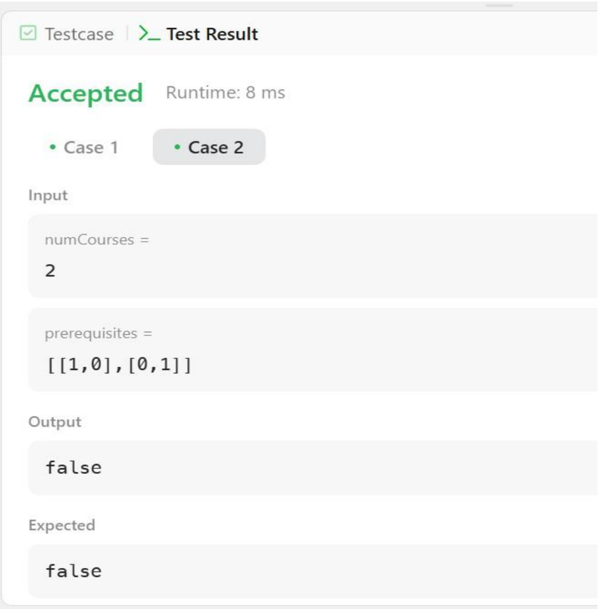
prerequisites =
[[1,0]]

Output

true

Expected

true



Accepted Runtime: 8 ms

Input

numCourses =
2

prerequisites =
[[1,0],[0,1]]

Output

false

Expected

false

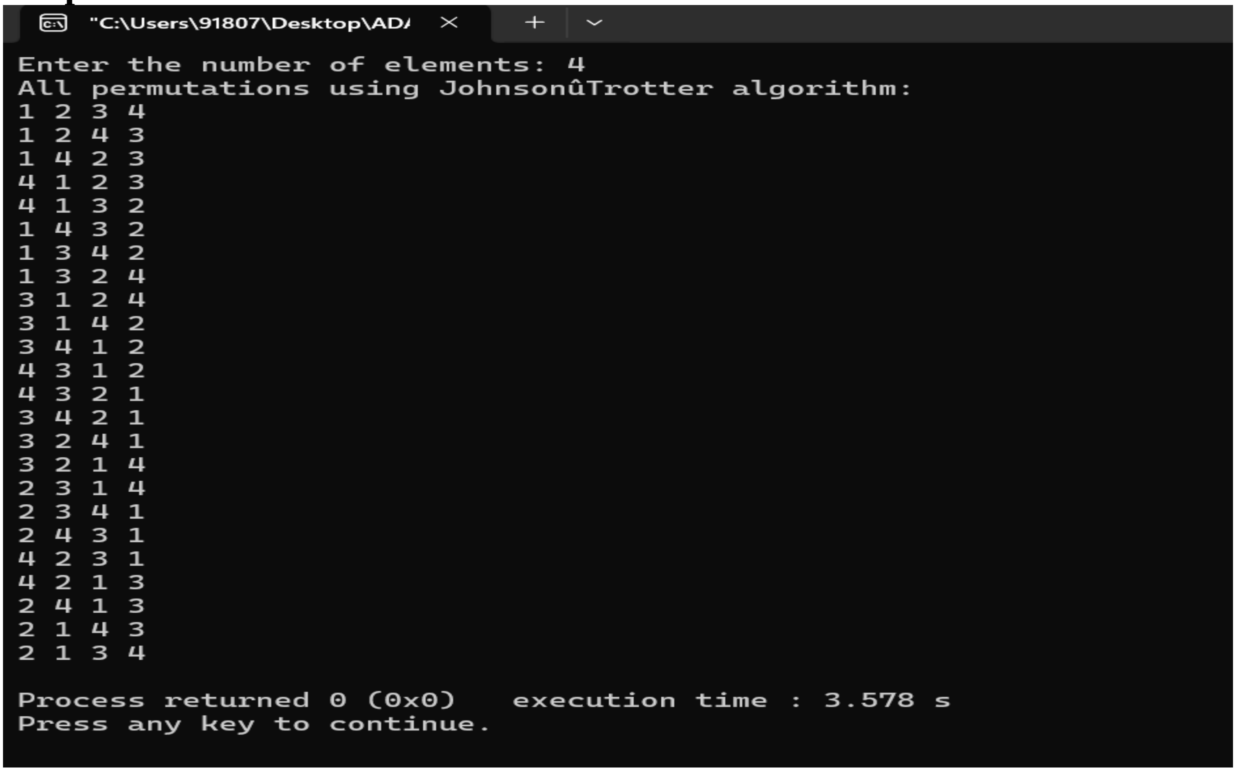
Lab program 2:

Implement Johnson Trotter algorithm to generate permutations.

Code:

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

#define LEFT_TO_RIGHT true
#define RIGHT_TO_LEFT false

int searchPosition(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (a[i] == mobile)
            return i;
    }
    return -1;
}

int getMobile(int a[], bool dir[], int n) {
    int mobile = 0;

    for (int i = 0; i < n; i++) {
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0 && a[i] > a[i - 1]) {
            if (a[i] > mobile)
                mobile = a[i];
        }

        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1 && a[i] > a[i + 1]) {
            if (a[i] > mobile)
                mobile = a[i];
        }
    }
    return mobile;
}

void printOnePerm(int a[], bool dir[], int n) {
    int mobile = getMobile(a, dir, n);
    if (mobile == 0)
        return;

    int pos = searchPosition(a, n, mobile);

    // Swap according to direction
    if (dir[mobile - 1] == RIGHT_TO_LEFT) {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    } else if (dir[mobile - 1] == LEFT_TO_RIGHT) {
        int temp = a[pos];
        a[pos] = a[pos + 1];
        a[pos + 1] = temp;
    }

    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            dir[a[i] - 1] = !dir[a[i] - 1];
        }
    }

    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int fact(int n) {
    int res = 1;
    for (int i = 1; i <= n; i++)
        res *= i;
    return res;
}

void printPermutations(int n) {
    int a[n];
    bool dir[n];

    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        dir[i] = RIGHT_TO_LEFT;
    }

    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

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

    printf("All permutations using Johnson–Trotter algorithm:\n");
    printPermutations(n);

    return 0;
}
```

Output:



```
Enter the number of elements: 4
All permutations using JohnsonûTrotter algorithm:
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4

Process returned 0 (0x0)   execution time : 3.578 s
Press any key to continue.
```

Lab program 3:

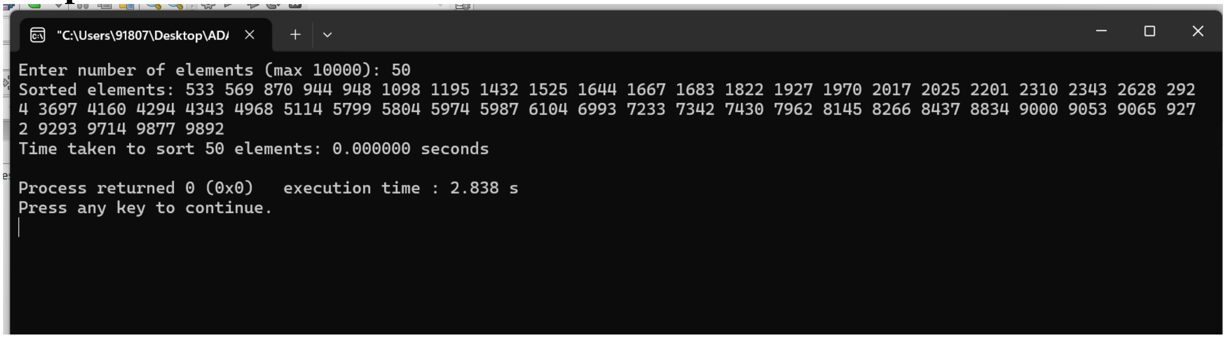
a] Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define max 10000
void mergeSort(int arr[],int low,int high)
{
   if(low<high)
   {
      int mid=low+(high-low)/2;
      mergeSort(arr,low,mid);
      mergeSort(arr,mid+1,high);
      merge(arr,low,mid,high);
   }
}
void merge(int arr[],int low,int mid,int high)
{
   int i,j,k;
   int n1=mid-low+1;
   int n2=high-mid;
```

```
    int L[max],R[max];
    for(i=0;i<n1;i++)
        L[i]=arr[low+i];
    for(j=0;j<n2;j++)
        R[j]=arr[mid+1+j];
    i=0,j=0,k=low;
    while(i<n1 && j<n2)
    {
        if(L[i]<=R[j])
            arr[k++]=L[i++];
        else
            arr[k++]=R[j++];
    }
    while(i<n1)
    {
        arr[k++]=L[i++];
    }
    while(j<n2)
    {
        arr[k++]=R[j++];
    }
}

int main()
{
    int N;
    int arr[max];

    printf("Enter number of elements (max %d): ", max);
    scanf("%d", &N);

    if (N >max) {
        printf("Error: Maximum size exceeded.\n");
        return 1;
    }
    srand(time(0));
    for (int i=0;i<N;i++)
        arr[i]=rand()%10000;

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

    printf("Sorted elements: ");
    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 to sort %d elements: %f seconds\n", N, time_taken);
    return 0;}
```

Output:



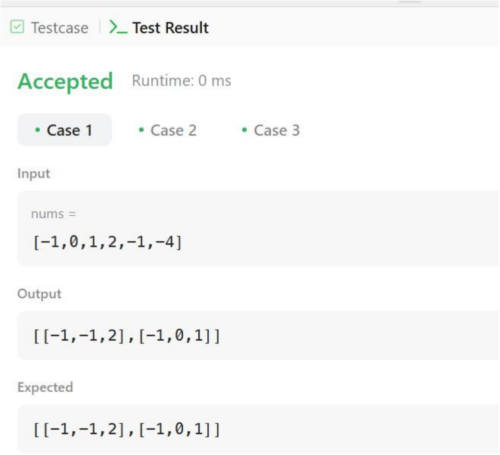
```
Enter number of elements (max 10000): 50
Sorted elements: 533 569 870 944 948 1098 1195 1432 1525 1644 1667 1683 1822 1927 1970 2017 2025 2201 2310 2343 2628 292
4 3697 4160 4294 4343 4968 5114 5799 5804 5974 5987 6104 6993 7233 7342 7430 7962 8145 8266 8437 8834 9000 9053 9065 927
2 9293 9714 9877 9892
Time taken to sort 50 elements: 0.000000 seconds

Process returned 0 (0x0)   execution time : 2.838 s
Press any key to continue.
```

b] LeetCode Program related to sorting.

Code:

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

int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes) {
    qsort(nums, numsSize, sizeof(int), compare);

    int** arr = calloc(numsSize * numsSize, sizeof(int*));
    *returnColumnSizes = (int*)calloc(numsSize * numsSize, sizeof(int));
    int index = 0;
    for(int i=0; i<numsSize-2; i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        int ptr1 = i+1, ptr2 = numsSize-1;
        while(ptr1 < ptr2){
            int sum = nums[i] + nums[ptr1] + nums[ptr2];
            if(sum == 0){
                arr[index] = (int*)calloc(3, sizeof(int));
                arr[index][0] = nums[i];
                arr[index][1] = nums[ptr1];
                arr[index][2] = nums[ptr2];
                (*returnColumnSizes)[index] = 3;
                index++;

                while(ptr1 < ptr2 && nums[ptr1] == nums[ptr1 + 1]) ptr1++;
                while(ptr1 < ptr2 && nums[ptr2] == nums[ptr2 - 1]) ptr2--;

                ptr1++;
                ptr2--;
            }
            else if(sum > 0) ptr2--;
            else ptr1++;
        }
    }
    *returnSize = index;
    return arr;}
```

Output:



☑ Testcase | ≻_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

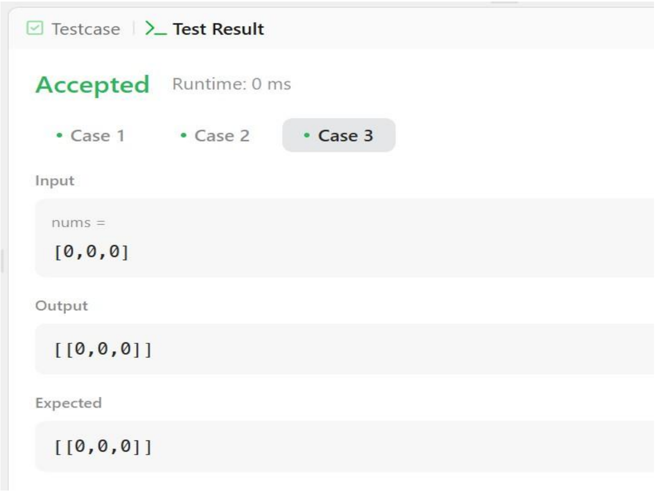
```
nums =
[-1,0,1,2,-1,-4]
```

Output

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

Expected

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



☑ Testcase | ≻_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
nums =
[0,0,0]
```

Output

```
[[0,0,0]]
```

Expected

```
[[0,0,0]]
```

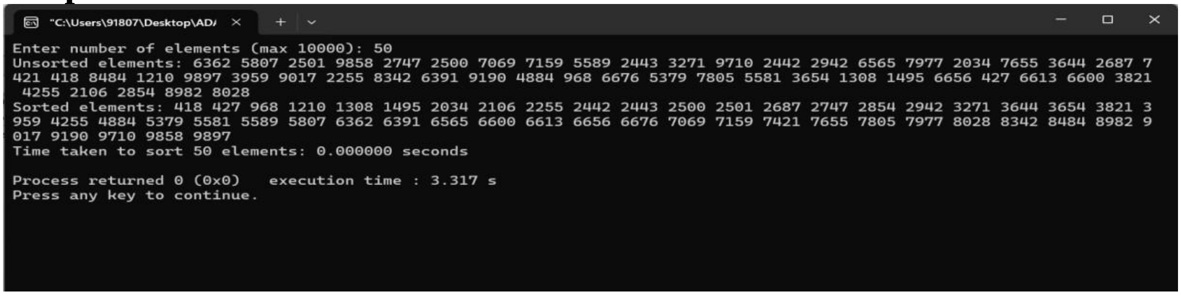
Lab program 4:

a] 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 10000
void quickSort(int arr[],int low,int high)
{
    if(low<high)
    {
        int p=Partition(arr,low,high);
        quickSort(arr,low,p-1);
        quickSort(arr,p+1,high);
    }
}

int Partition(int arr[],int low,int high)
{
    int pivot=arr[high];
    int i=low-1,j;
    for(int j=low;j<high;j++)
    {
        if(arr[j]<pivot)
        {
            i++;
            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }
    int temp=arr[i+1];
    arr[i+1]=arr[high];
    arr[high]=temp;
    return (i+1);
}

int main()
{
    int N;
    int arr[max];

    printf("Enter number of elements (max %d): ", max);
    scanf("%d", &N);

    if (N >max) {
        printf("Error: Maximum size exceeded.\n");
        return 1;
    }
    srand(time(0));
    for (int i=0;i<N;i++)
        arr[i]=rand()%10000;

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

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

    printf("Sorted elements: ");
    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 to sort %d elements: %f seconds\n", N, time_taken);

    return 0;
}
```

Output:



```
Enter number of elements (max 10000): 50
Unsorted elements: 6362 5807 2501 9858 2747 2500 7069 7159 5589 2443 3271 9710 2442 2942 6565 7977 2034 7655 3644 2687 7
421 418 8484 1210 9897 3959 9017 2255 8342 6391 9190 4884 968 6676 5379 7805 5581 3654 1308 1495 6656 427 6613 6600 3821
 4255 2106 2854 8982 8028
Sorted elements: 418 427 968 1210 1308 1495 2034 2106 2255 2442 2443 2500 2501 2687 2747 2854 2942 3271 3644 3654 3821 3
959 4255 4884 5379 5581 5589 5807 6362 6391 6565 6600 6613 6656 6676 7069 7159 7421 7655 7805 7977 8028 8342 8484 8982 9
017 9190 9710 9858 9897
Time taken to sort 50 elements: 0.000000 seconds

Process returned 0 (0x0)   execution time : 3.317 s
Press any key to continue.
```

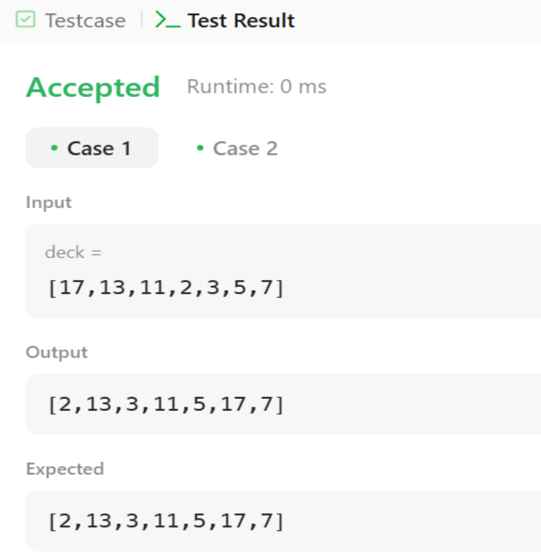
b] LeetCode Program related to sorting.

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:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

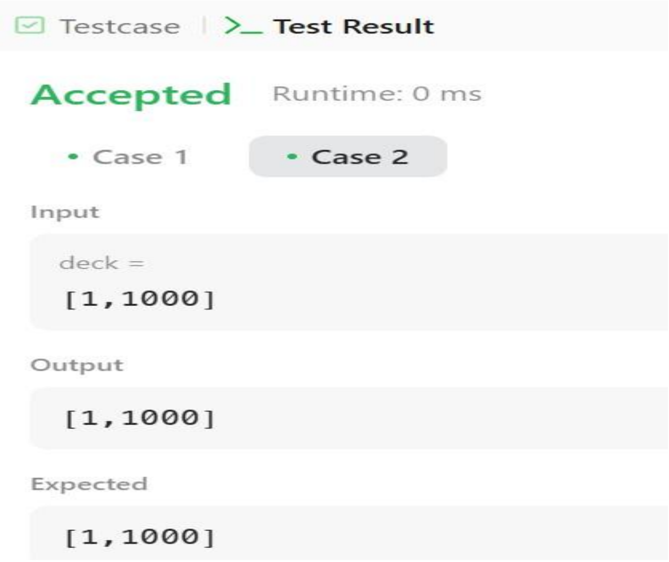
```
deck =
[17,13,11,2,3,5,7]
```

Output

```
[2,13,3,11,5,17,7]
```

Expected

```
[2,13,3,11,5,17,7]
```



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
deck =
[1,1000]
```

Output

```
[1,1000]
```

Expected

```
[1,1000]
```

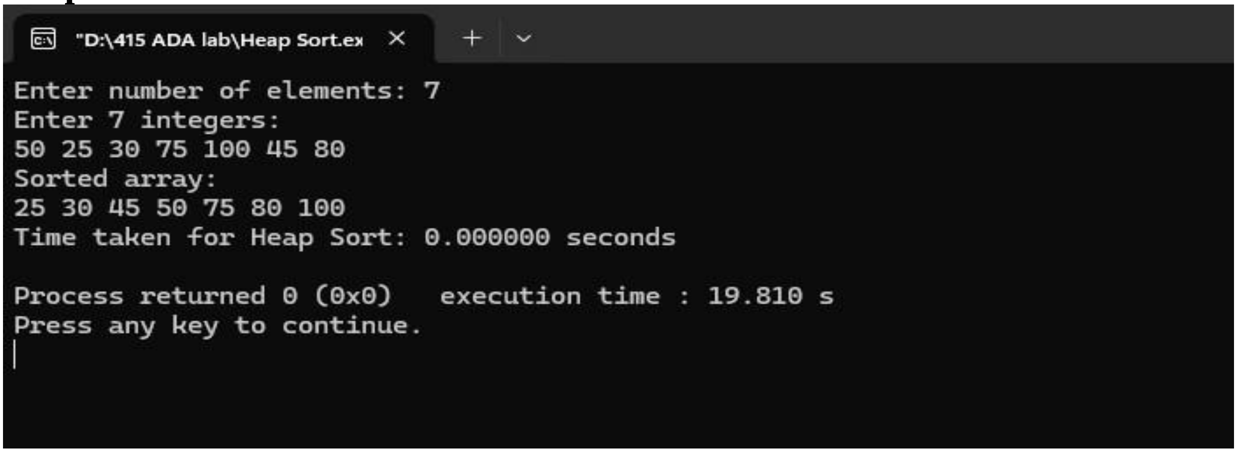
Lab program 5:

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>

// Function to swap two integers
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Heapify a subtree rooted at index i in array of size n
void heapify(int arr[], int n, int i) {
    int largest = i;      // Initialize largest as root
    int left = 2 * i + 1; // left = 2*i + 1
    int right = 2 * i + 2;// 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 largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

// Main function to perform heap sort
void heapSort(int arr[], int n) {
    // Build a maxheap
    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--) {
        swap(&arr[0], &arr[i]); // Move current root to end
        heapify(arr, i, 0);     // call max heapify on the reduced heap
    }
}

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}

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;
    double cpu_time_used;

    start = clock();             // Start timer
    heapSort(arr, n);            // Perform heap sort
    end = clock();               // End timer

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

    printf("Sorted array:\n");
    printArray(arr, n);

    printf("Time taken for Heap Sort: %f seconds\n", cpu_time_used);

    return 0;
}
```

Output:



```
Enter number of elements: 7
Enter 7 integers:
50 25 30 75 100 45 80
Sorted array:
25 30 45 50 75 80 100
Time taken for Heap Sort: 0.000000 seconds

Process returned 0 (0x0)   execution time : 19.810 s
Press any key to continue.
```

Lab program 6:

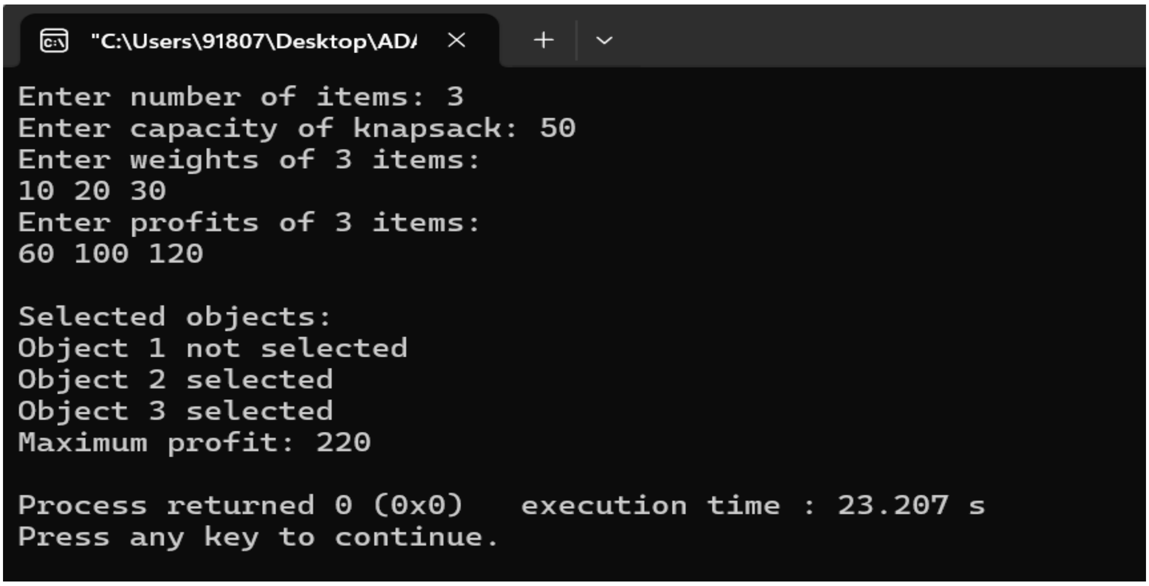
Implement 0/1 Knapsack problem using dynamic programming.

Code:

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

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

void knapsack(int n, int M, int w[], int p[]) {
    int v[n + 1][M + 1];
    int x[n + 1];
    int i, j;

    // Build table v[][] in bottom-up manner
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= M; j++) {
            if (i == 0 || j == 0)
```

```
                v[i][j] = 0;
            else if (w[i] > j)
                v[i][j] = v[i - 1][j];
            else
                v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i]] + p[i]);
        }
    }

    // Initialize selection array
    for (i = 1; i <= n; i++)
        x[i] = 0;

    // Traceback to find selected items
    i = n;
    j = M;
    while (i != 0 && j != 0) {
        if (v[i][j] != v[i - 1][j]) {
            x[i] = 1;
            j = j - w[i];
        }
        i--;
    }

    // Print selected objects and max profit
    printf("\nSelected objects:\n");
    for (i = 1; i <= n; i++) {
        if (x[i])
            printf("Object %d selected\n", i);
        else
            printf("Object %d not selected\n", i);
    }

    printf("Maximum profit: %d\n", v[n][M]);
}

int main() {
    int n, M, i;

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

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

    int w[n + 1], p[n + 1];

    printf("Enter weights of %d items:\n", n);
    for (i = 1; i <= n; i++) {
        scanf("%d", &w[i]);
    }
    printf("Enter profits of %d items:\n", n);
    for (i = 1; i <= n; i++) {
        scanf("%d", &p[i]);
    }

    knapsack(n, M, w, p);

    return 0;
}
```

Output:



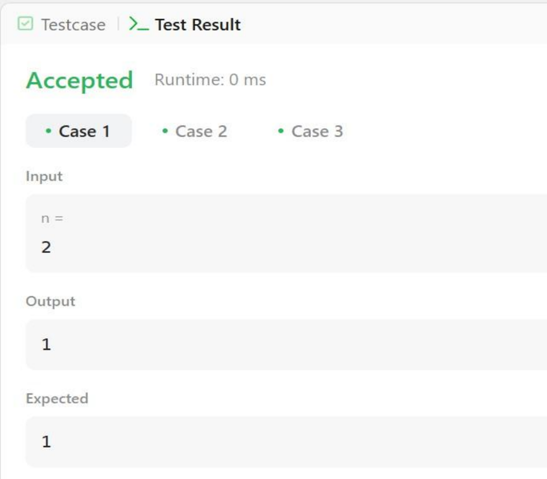
```
Enter number of items: 3
Enter capacity of knapsack: 50
Enter weights of 3 items:
10 20 30
Enter profits of 3 items:
60 100 120

Selected objects:
Object 1 not selected
Object 2 selected
Object 3 selected
Maximum profit: 220

Process returned 0 (0x0)   execution time : 23.207 s
Press any key to continue.
```

b] LeetCode Program related to Knapsack problem or Dynamic Programming.

Code:

```
int fib(int n) {
   if(n==0) return 0;
   if(n==1) return 1;
   int fibArr[n+1];
   int fibArr[0]=0;
   fibArr[1]=1;
   for(int i=2;i<=n;i++)
      fibArr[i]=fibArr[i-1]+fibArr[i-2];
   return fibArr[n];
}
```

Output:



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

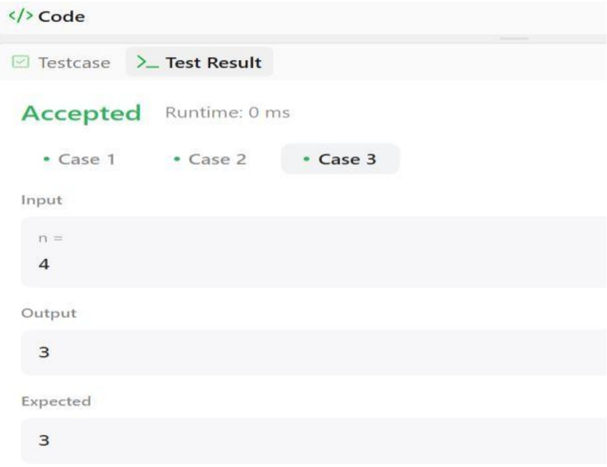
n =
2

Output

1

Expected

1



Code

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

n =
4

Output

3

Expected

3

Lab program 7:

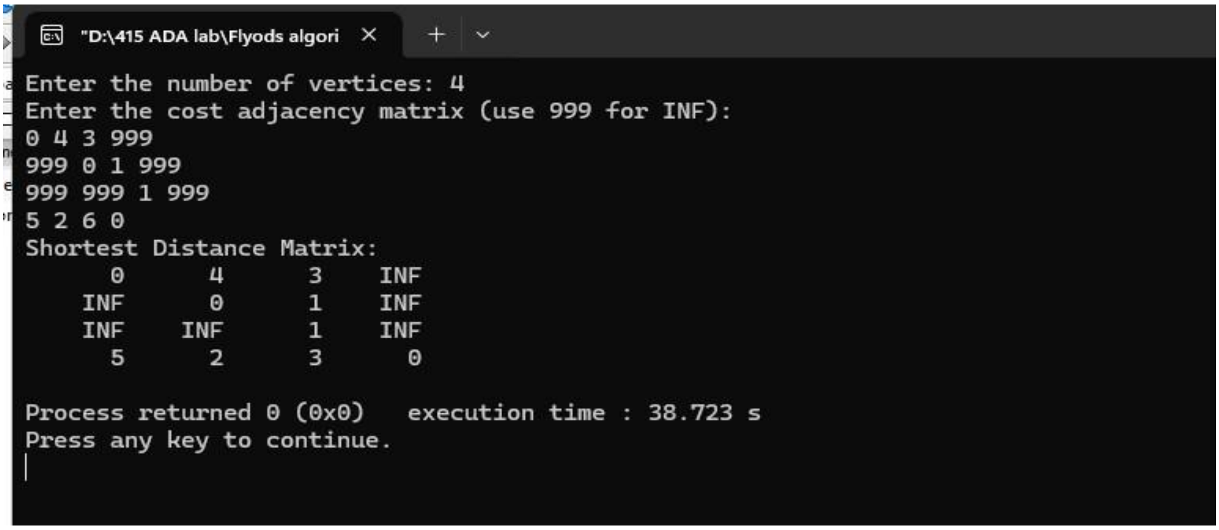
a] Implement All Pair Shortest paths problem using Floyd’s algorithm.

Code:

```
#include <stdio.h>
#define INF 999
#define MAX 10

void floyd(int a[MAX][MAX], int n) {
    int d[MAX][MAX];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            d[i][j] = a[i][j];
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (d[i][k] + d[k][j] < d[i][j]) {
                    d[i][j] = d[i][k] + d[k][j];
                }
            }
        }
    }

    printf("Shortest Distance Matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][j] == INF)
```

```
                printf("%7s", "INF");
            else
                printf("%7d", d[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n;

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

    int a[MAX][MAX];

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

    floyd(a, n);

    return 0;
}
```

Output:



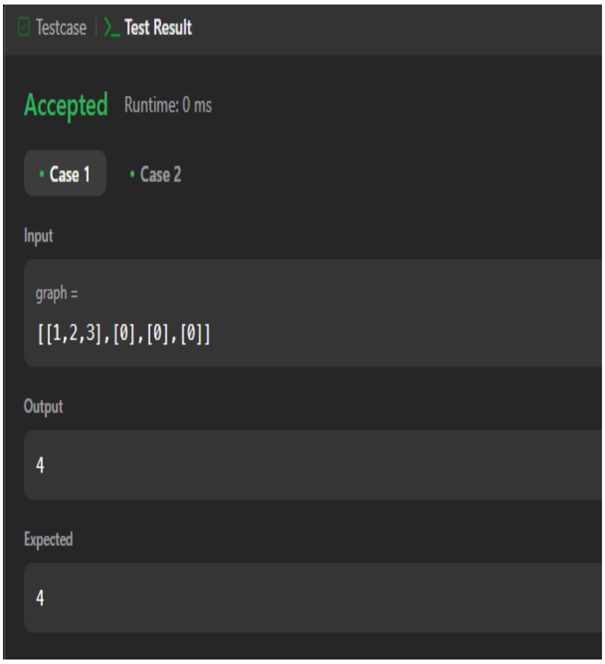
```
Enter the number of vertices: 4
Enter the cost adjacency matrix (use 999 for INF):
0 4 3 999
999 0 1 999
999 999 1 999
5 2 6 0
Shortest Distance Matrix:
      0      4      3    INF
    INF      0      1    INF
    INF    INF      1    INF
      5      2      3      0

Process returned 0 (0x0)   execution time : 38.723 s
Press any key to continue.
```

b] LeetCode Program related to shortest distance calculation.

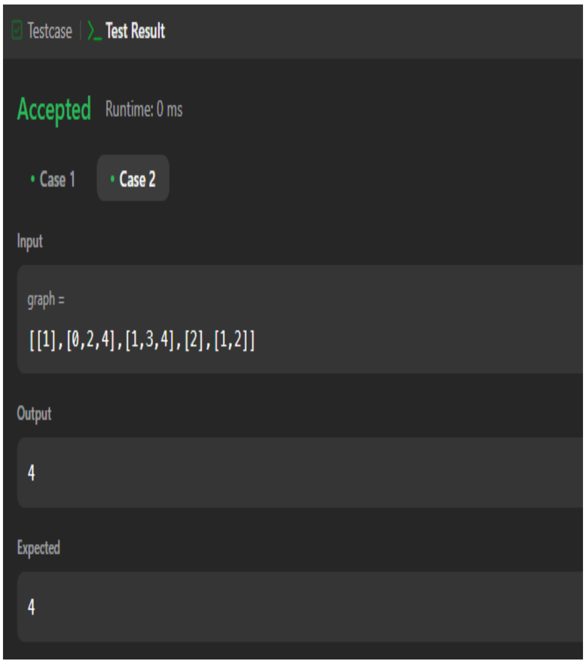
Code:

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

#define MAXN 12
#define MAXQ (1 << MAXN) * MAXN

typedef struct {
    int node;
    int mask;
    int dist;
} State;

int shortestPathLength(int** graph, int graphSize, int* graphColSize) {
    int allVisited = (1 << graphSize) - 1;
    bool visited[MAXN][1 << MAXN] = { false };

    State queue[MAXQ];
    int front = 0, rear = 0;

    // Initialize queue with each node as starting point
    for (int i = 0; i < graphSize; i++) {
        int mask = 1 << i;
        queue[rear++] = (State){i, mask, 0};
        visited[i][mask] = true;
    }

    while (front < rear) {
        State curr = queue[front++];

        if (curr.mask == allVisited) {
            return curr.dist;
        }

        for (int i = 0; i < graphColSize[curr.node]; i++) {
            int neighbor = graph[curr.node][i];
            int nextMask = curr.mask | (1 << neighbor);

            if (!visited[neighbor][nextMask]) {
                visited[neighbor][nextMask] = true;
                queue[rear++] = (State){neighbor, nextMask, curr.dist + 1};
            }
        }
    }

    return -1;  // Should never reach here}
```

Output:



```
Accepted Runtime: 0 ms

Input
graph =
[[1,2,3],[0],[0],[0]]

Output
4

Expected
4
```



```
Accepted Runtime: 0 ms

Input
graph =
[[1],[0,2,4],[1,3,4],[2],[1,2]]

Output
4

Expected
4
```

Lab program 8:

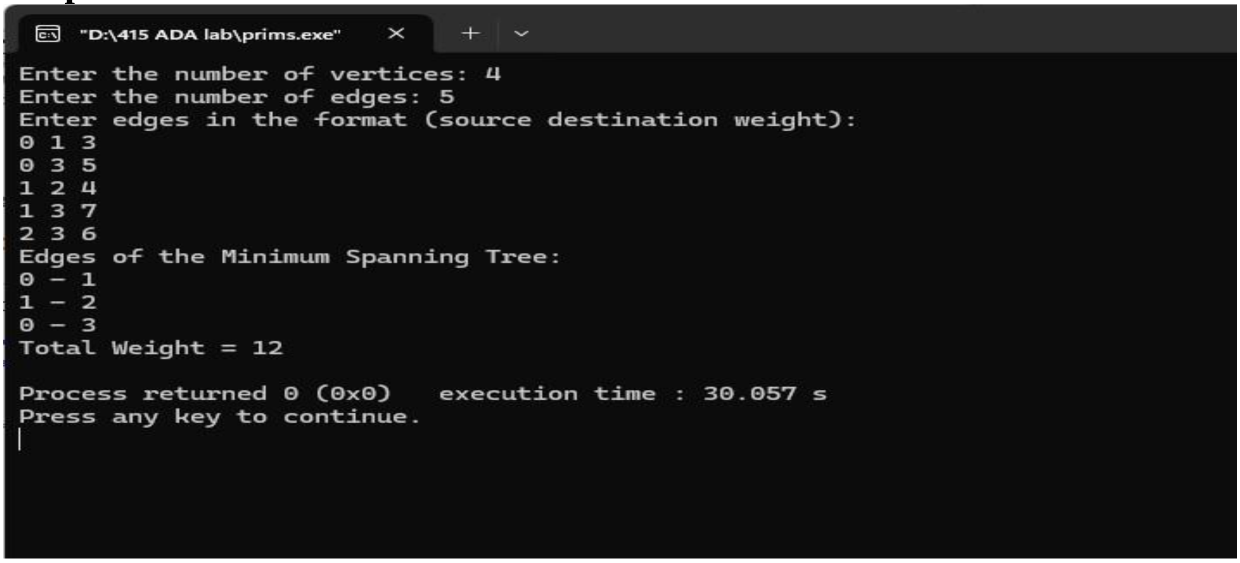
a] Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

Code:

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

#define MAX 10
#define INF 999

int cost[MAX][MAX], et[MAX][2], vis[MAX], n, e = 0, sum = 0;

void prims();

int main() {
    int i, j, edges, u, v, w;

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


    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            cost[i][j] = INF;
        }
        vis[i] = 0;
    }

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

    printf("Enter edges in the format (source destination weight):\n");
    for (i = 0; i < edges; i++) {
        scanf("%d %d %d", &u, &v, &w);
        cost[u][v] = w;
        cost[v][u] = w;
    }

    prims();

    printf("Edges of the Minimum Spanning Tree:\n");
    for (i = 0; i < e; i++) {
        printf("%d - %d\n", et[i][0], et[i][1]);
    }
    printf("Total Weight = %d\n", sum);

    return 0;
}

void prims() {
    int i, j, min, u, v;


    vis[0] = 1;
    e = 0;

    for (i = 1; i < n; i++) {
        min = INT_MAX;
        u = v = -1;


        for (j = 0; j < n; j++) {
            if (vis[j]) {
                for (int k = 0; k < n; k++) {
```

```
                if (!vis[k] && cost[j][k] < min) {
                    min = cost[j][k];
                    u = j;
                    v = k;
                }
            }
        }
    }

    if (u != -1 && v != -1) {
        et[e][0] = u;
        et[e][1] = v;
        vis[v] = 1;
        sum += min;
        e++;
    }
  }
}
```

Output:



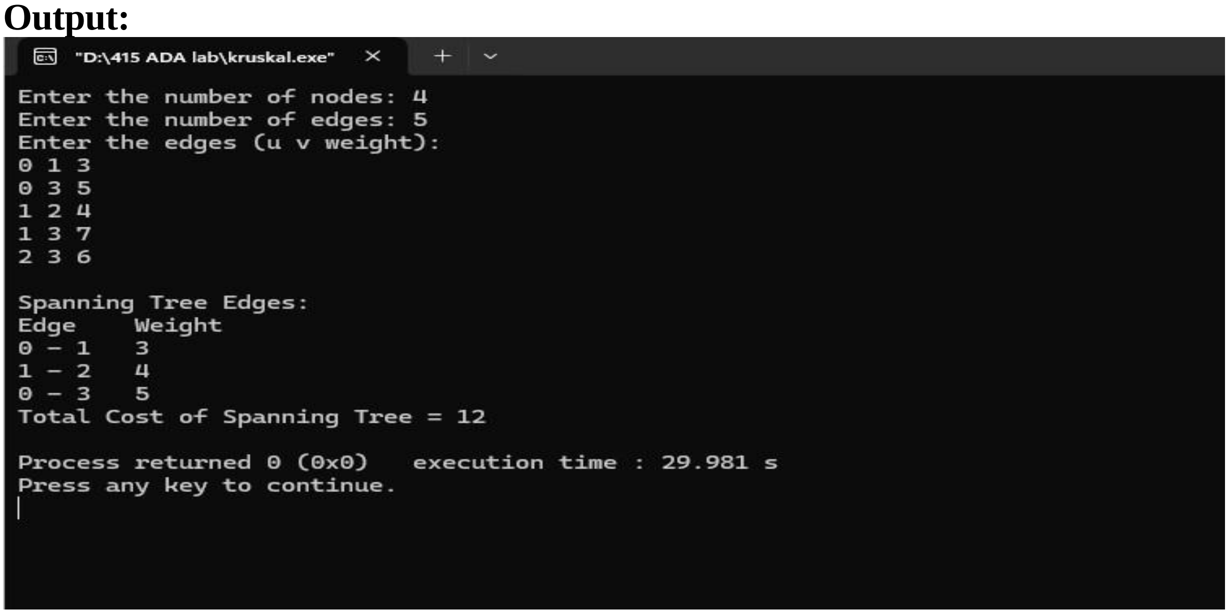
```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter edges in the format (source destination weight):
0 1 3
0 3 5
1 2 4
1 3 7
2 3 6
Edges of the Minimum Spanning Tree:
0 - 1
1 - 2
0 - 3
Total Weight = 12

Process returned 0 (0x0)   execution time : 30.057 s
Press any key to continue.
```

b] Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.

Code:

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

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

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

void kruskal(int n, int a[10][10]) {
    int count = 0, k = 0, sum = 0;
    int t[10][3], parent[10];

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

    while (count < n - 1) {
        int min = 999, u = -1, v = -1;

        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;
                }
            }
        }

        if (u == -1 || v == -1) {
            printf("Spanning tree does not exist\n");
            return;
        }

        int i = find(u, parent);
        int j = find(v, parent);

        if (i != j) {
            union1(i, j, parent);
            t[k][0] = u;
            t[k][1] = v;
            t[k][2] = min;
            k++;
            count++;
            sum += min;
        }
        a[u][v] = a[v][u] = 999;
    }
    printf("\nSpanning Tree Edges:\n");
    printf("Edge \tWeight\n");
    for (int i = 0; i < n - 1; i++) {
        printf("%d - %d \t%d\n", t[i][0], t[i][1], t[i][2]);
    }
    printf("Total Cost of Spanning Tree = %d\n", sum);
}

int main() {
    int n, m, u, v, weight;
    int a[10][10];

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


    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = (i == j) ? 0 : 999;

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

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

    kruskal(n, a);
    return 0;
}
```

Output:



```
Enter the number of nodes: 4
Enter the number of edges: 5
Enter the edges (u v weight):
0 1 3
0 3 5
1 2 4
1 3 7
2 3 6

Spanning Tree Edges:
Edge    Weight
0 - 1   3
1 - 2   4
0 - 3   5
Total Cost of Spanning Tree = 12

Process returned 0 (0x0)   execution time : 29.981 s
Press any key to continue.
```

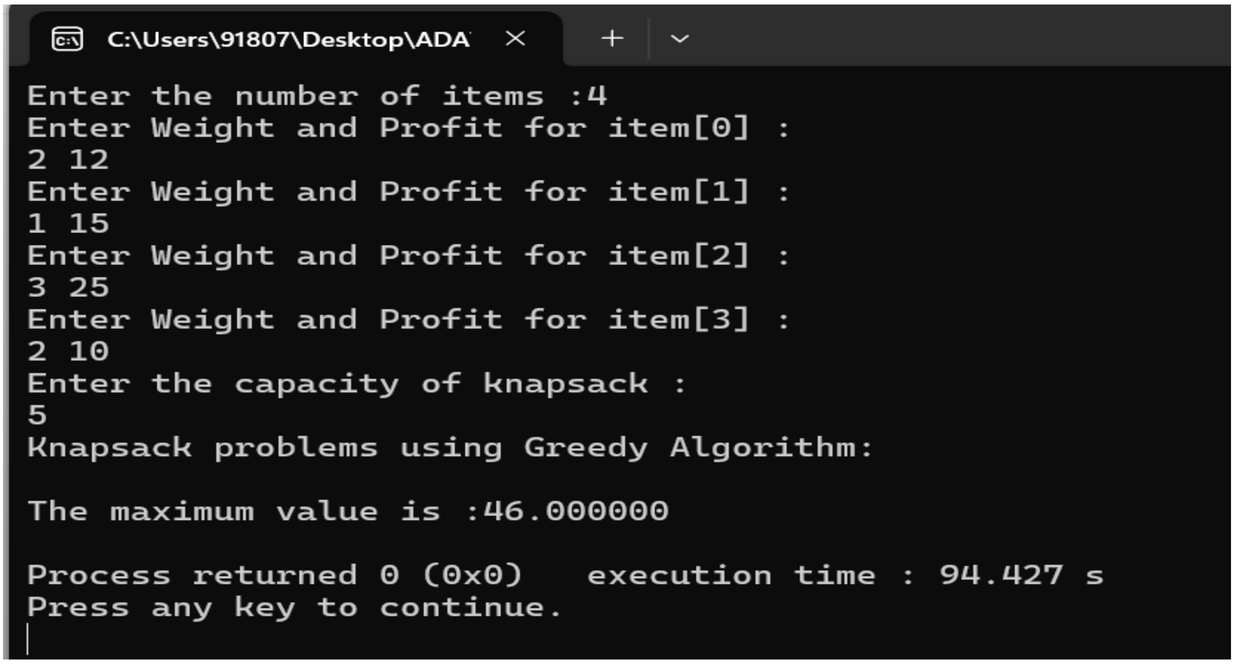
Lab program 9:

a] Implement Fractional Knapsack using Greedy technique.

Code:

```
#include<stdio.h>
int main()
{
     float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
     int n,i,j;
     printf("Enter the number of items :");
     scanf("%d",&n);
    for (i = 0; i < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n");
    scanf("%f",&capacity);

     for(i=0;i<n;i++)
         ratio[i]=profit[i]/weight[i];

    for (i = 0; i < n; i++)
      for (j = i + 1; j < n; j++)
         if (ratio[i] < ratio[j])
        {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;

            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;

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

     printf("Knapsack problems using Greedy Algorithm:\n");
     for (i = 0; i < n; i++)
     {
      if (weight[i] > capacity)
         break;
       else
      {
         Totalvalue = Totalvalue + profit[i];
         capacity = capacity - weight[i];
       }
     }
       if (i < n)
       Totalvalue = Totalvalue + (ratio[i]*capacity);
     printf("\nThe maximum value is :%f\n",Totalvalue);
     return 0;
}
```

Output:



```
Enter the number of items :4
Enter Weight and Profit for item[0] :
2 12
Enter Weight and Profit for item[1] :
1 15
Enter Weight and Profit for item[2] :
3 25
Enter Weight and Profit for item[3] :
2 10
Enter the capacity of knapsack :
5
Knapsack problems using Greedy Algorithm:

The maximum value is :46.000000

Process returned 0 (0x0)   execution time : 94.427 s
Press any key to continue.
```

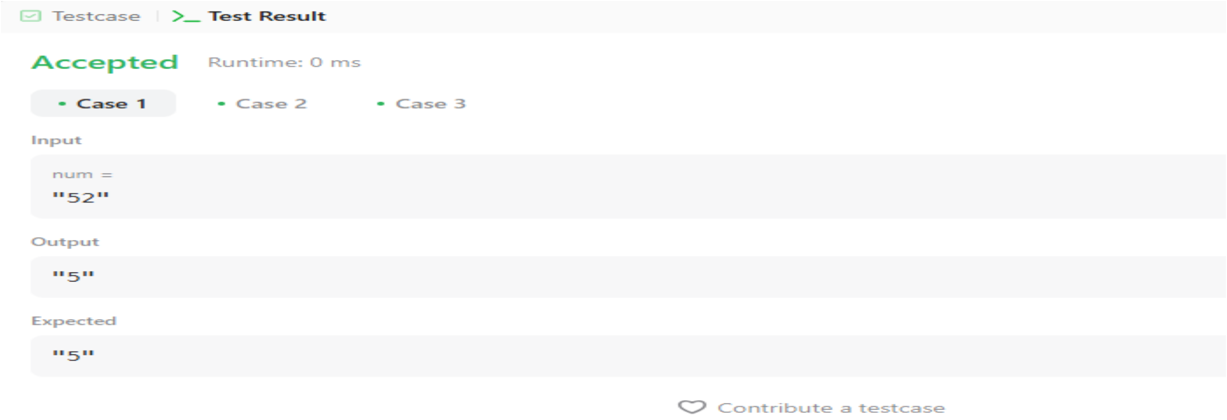
b] LeetCode Program related to Greedy Technique algorithms.

Code:

```
char* largestOddNumber(char* num) {
   int len = strlen(num);
   for (int i = len - 1; i >= 0; i--) {
      if (num[i] % 2 != 0) {
          return num;
      } else {
          num[i] = '\0';
```

```
        }
    }
    return num;
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
num =
"52"
```

Output

```
"5"
```

Expected

```
"5"
```

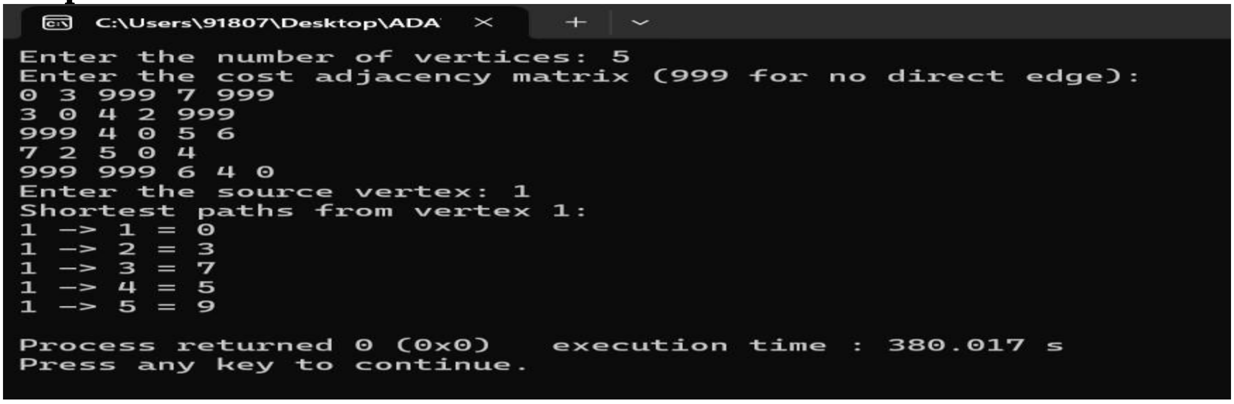
Lab program 10:

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 999

void dijkstra(int n, int cost[10][10], int src) {
    int dis[10], vis[10] = {0};
    int i, j, u = src, min;

    // Initialize distance array
    for (i = 1; i <= n; i++) {
        dis[i] = cost[src][i];
        vis[i] = 0;
    }
    dis[src] = 0;
    vis[src] = 1;

    for (i = 1; i < n; i++) {
        min = INF;
        // Find the vertex with minimum distance
        for (j = 1; j <= n; j++) {
            if (!vis[j] && dis[j] < min) {
                min = dis[j];
                u = j;
            }
        }

        vis[u] = 1;

        // Update distances
        for (j = 1; j <= n; j++) {
            if (!vis[j] && dis[u] + cost[u][j] < dis[j]) {
                dis[j] = dis[u] + cost[u][j];
            }
        }
    }

    printf("Shortest paths from vertex %d:\n", src);
    for (i = 1; i <= n; i++) {
        if (dis[i] == INF)
            printf("%d -> %d = Unreachable\n", src, i);
        else
            printf("%d -> %d = %d\n", src, i, dis[i]);
    }
}

int main() {
    int n, src, cost[10][10];
    int i, j;

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

    printf("Enter the cost adjacency matrix (999 for no direct edge):\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

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

    dijkstra(n, cost, src);

    return 0;
}
```

Output:



```
Enter the number of vertices: 5
Enter the cost adjacency matrix (999 for no direct edge):
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 6 4 0
Enter the source vertex: 1
Shortest paths from vertex 1:
1 -> 1 = 0
1 -> 2 = 3
1 -> 3 = 7
1 -> 4 = 5
1 -> 5 = 9

Process returned 0 (0x0)   execution time : 380.017 s
Press any key to continue.
```

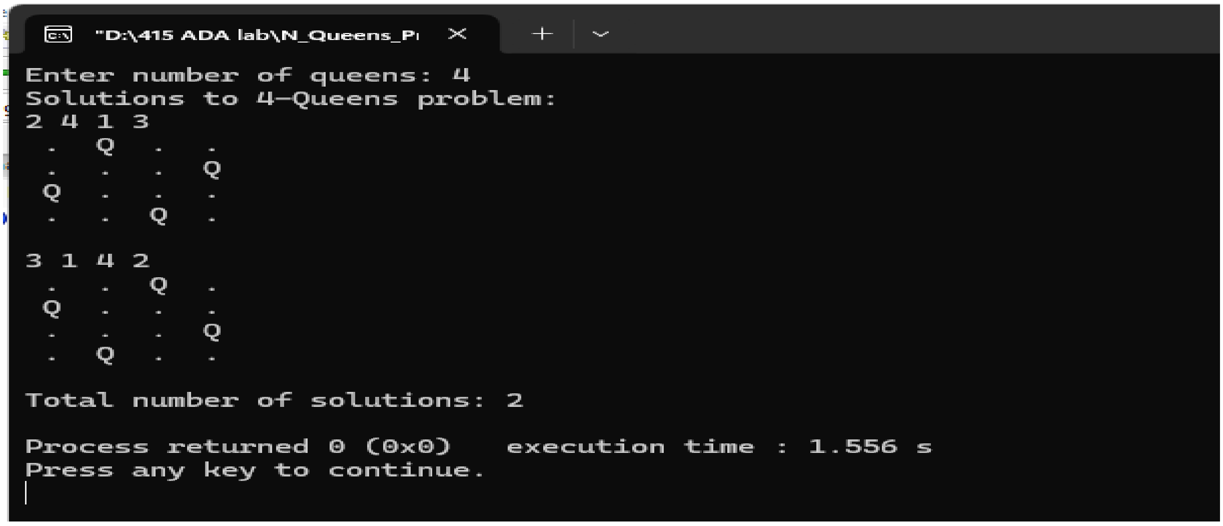
Lab program 11:

Implement “N-Queens Problem” using Backtracking.

Code:

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

#define MAX 20

int x[MAX], n, count = 0;

int place(int k) {
    for (int i = 1; i < k; i++)
        if (x[i] == x[k] || abs(i - k) == abs(x[i] - x[k]))
            return 0;
    return 1;
}

void printSolution() {
    for (int i = 1; i <= n; i++)
        printf("%d ", x[i]);
    printf("\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            printf(x[i] == j ? " Q " : " . ");
        printf("\n");
    }
    printf("\n");
}

void nQueen() {
    int k = 1;
    x[k] = 0;
    printf("Solutions to %d-Queens problem:\n", n);
    while (k > 0) {
        x[k]++;
        while (x[k] <= n && !place(k)) x[k]++;
        if (x[k] <= n) {
            if (k == n) {
                count++;
                printSolution();
            } else {
                x[++k] = 0;
            }
        } else {
            k--;
        }
    }
}

int main() {
    printf("Enter number of queens: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) {
        printf("Invalid input (1 - %d allowed)\n", MAX);
        return 1;
    }
    nQueen();
    printf("Total number of solutions: %d\n", count);
    return 0;
}
```

Output:



```
Enter number of queens: 4
Solutions to 4-Queens problem:
2 4 1 3
 .  Q  .  .
 .  .  .  Q
 Q  .  .  .
 .  .  Q  .

3 1 4 2
 .  .  Q  .
 Q  .  .  .
 .  .  .  Q
 .  Q  .  .

Total number of solutions: 2

Process returned 0 (0x0)   execution time : 1.556 s
Press any key to continue.
```