VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Ghanshyam Sharma(1BM23CS100)

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 2025 - June 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 Ghanshyam Sharma **(1BM23CS100)**, 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 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.

Srushti C S

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

INDEX

Name Ghanshyam Sharma Class IV B

Roll No Subject ADA LAB School BMSCE

Sl No	Date	Title	Page No.	Teacher Sign/ Remarks
1	28/02	Stack using Array & Linked . List		
1)	21/03.	LAB-1 Merge Sort		4/4/ 10
2)	04/04	LAB-2 Quick Sort		4/4/23 10
3)	[illegible]/04	LAB-3 Prism's A.		10
4)	09/04	LAB-4 Krushkal A.		
5)	16/05	LAB-5		
		Topological sort		6
		0/1 knapsack		6 16/5
		floyd's algo.		6
6)	17/05	LAB-6		
		1) Dijkstra's Algo		10
		2. Rang sum(LT)		
		3. find kth Largest (LT)		
		4. Course Schedule (LT)		10 17/5
		5. Pizza slices (LT)		
		6) Max. Units on Truck (L.T)		

7) Num. of ways

7)	23/05/025		LAB-7	$\frac{10}{10}$
		10	1) Johnson trotter	
		10	2) NQueen	
		20	3) Sorting using heap	
		10	4) fractional knapsack	

$\frac{098}{110}$

12

112

Course outcomes:

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.

Lab Program 1

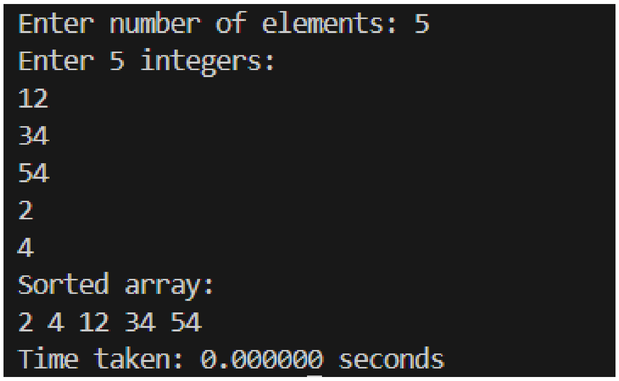
1.1.1 Question

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.

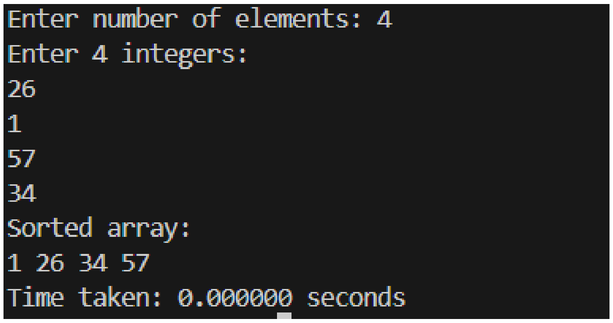
1.1.2 Code

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

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    i = 0; j = 0; k = left;
    while (i < n1 && j < n2) {
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
    free(L);
    free(R);
}

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

    int *arr = (int *)malloc(N * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

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

    clock_t start = clock();
    mergeSort(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("\n");

    printf("Time taken: %f seconds\n", time_taken);

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

1.1.3 Output

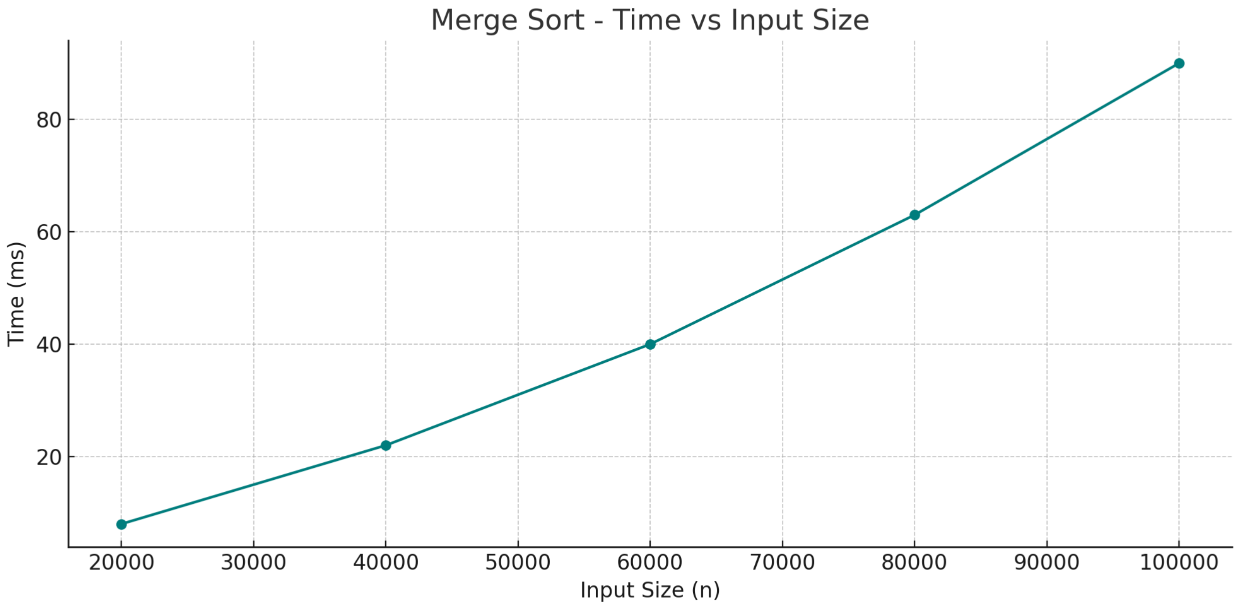


```
Enter number of elements: 5
Enter 5 integers:
12
34
54
2
4
Sorted array:
2 4 12 34 54
Time taken: 0.000000 seconds
```



```
Enter number of elements: 4
Enter 4 integers:
26
1
57
34
Sorted array:
1 26 34 57
Time taken: 0.000000 seconds
```

1.1.4 Graph

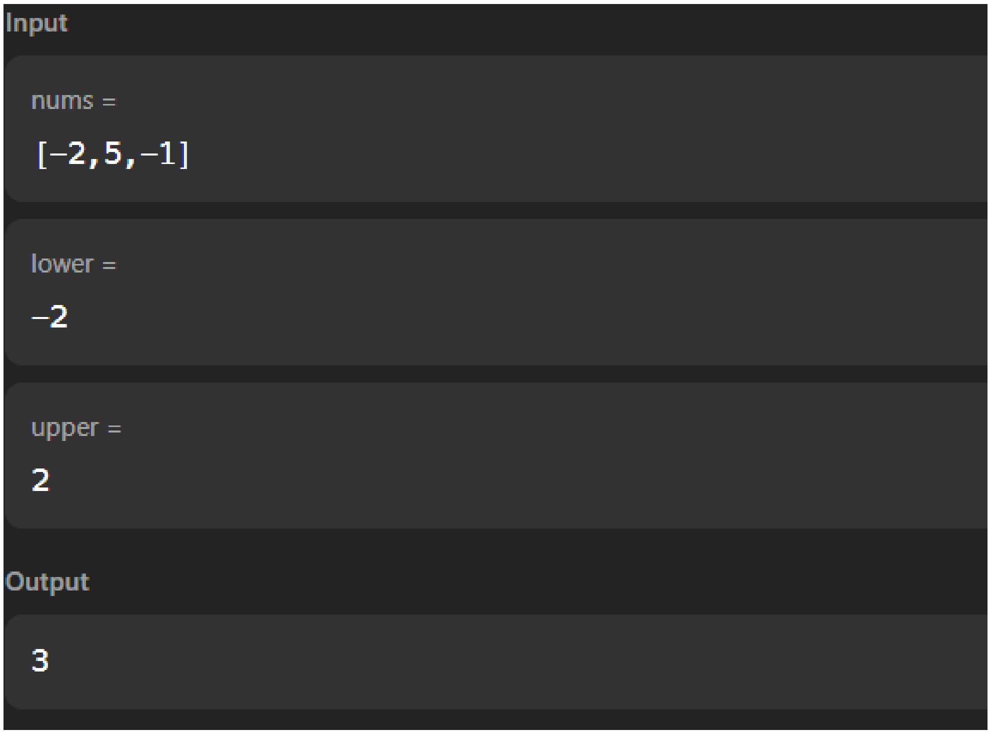


1.2.1 Leetcode Question
Count of Range Sum

1.2.2 Code

```
int countRangeSum(int* nums, int numsSize, int lower, int upper) {
    int count = 0;
    for (int i = 0; i < numsSize; i++) {
        long long sum = 0;
        for (int j = i; j < numsSize; j++) {
            sum += nums[j];
            if (sum >= lower && sum <= upper)
                count++;
        }
    }
    return count;
}
```

1.2.3 Output



```
Input
nums =
[-2,5,-1]
lower =
-2
upper =
2
Output
3
```

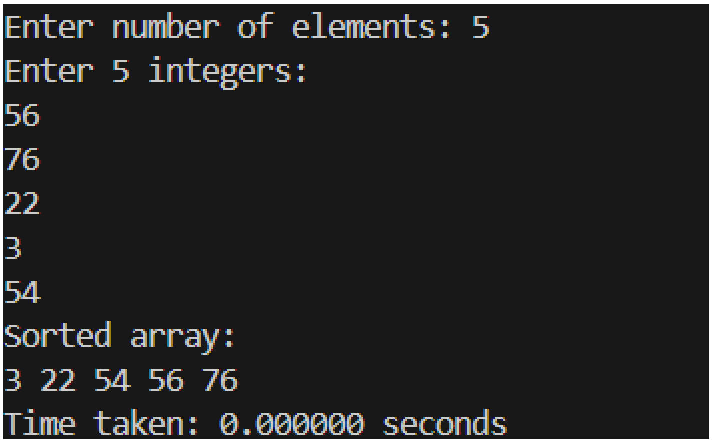
Lab Program 2

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

2.1.2 Code

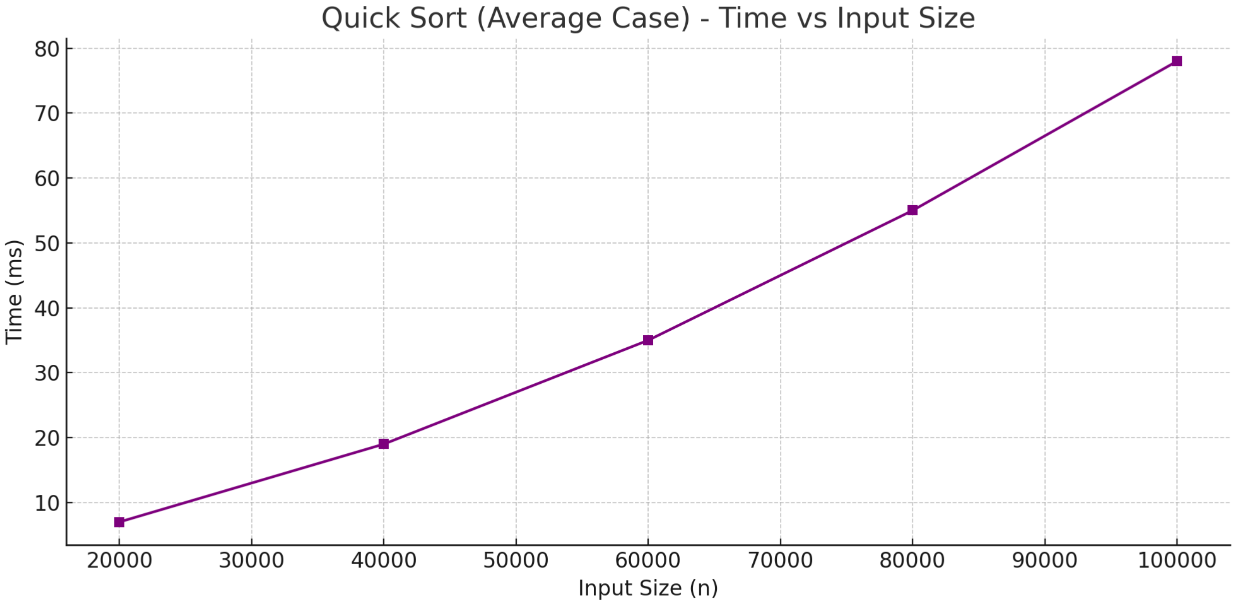
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1, temp;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
        }
    }
    temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
    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));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    printf("Enter %d integers:\n", 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("\n");

    printf("Time taken: %f seconds\n", time_taken);

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

2.1.3 Output



```
Enter number of elements: 5
Enter 5 integers:
56
76
22
3
54
Sorted array:
3 22 54 56 76
Time taken: 0.000000 seconds
```

2.1.4 Graph

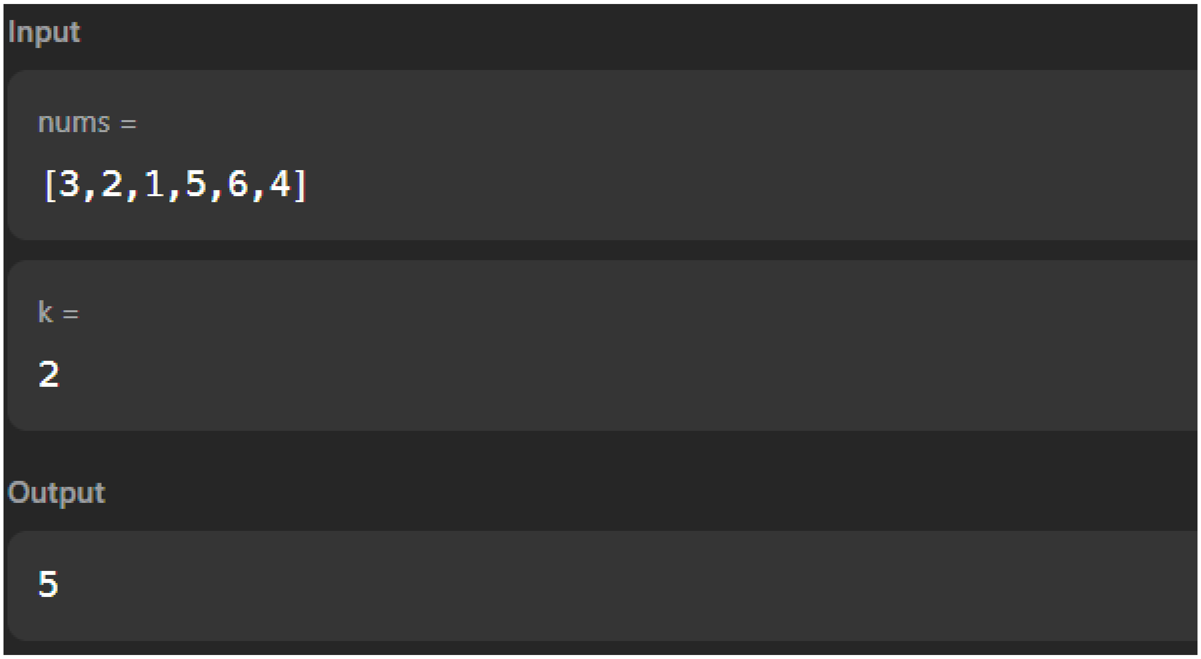


2.2.1 Leetcode Question
Kth Largest element in an array

2.2.2 Code

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

int partition(int arr[], int left, int right) {
    int pivot = arr[right];
    int i = left;
    for (int j = left; j < right; j++) {
        if (arr[j] <= pivot) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[right]);
    return i;
}

int quickSelect(int arr[], int left, int right, int k) {
    if (left == right)
        return arr[left];

    int pivotIndex = partition(arr, left, right);

    if (k == pivotIndex)
        return arr[k];
    else if (k < pivotIndex)
        return quickSelect(arr, left, pivotIndex - 1, k);
    else
        return quickSelect(arr, pivotIndex + 1, right, k);
}

int findKthLargest(int arr[], int n, int k) {
    return quickSelect(arr, 0, n - 1, n - k);
}
```

2.2.3 Output



Input

nums =

[3,2,1,5,6,4]

k =

2

Output

5

Lab Program 3

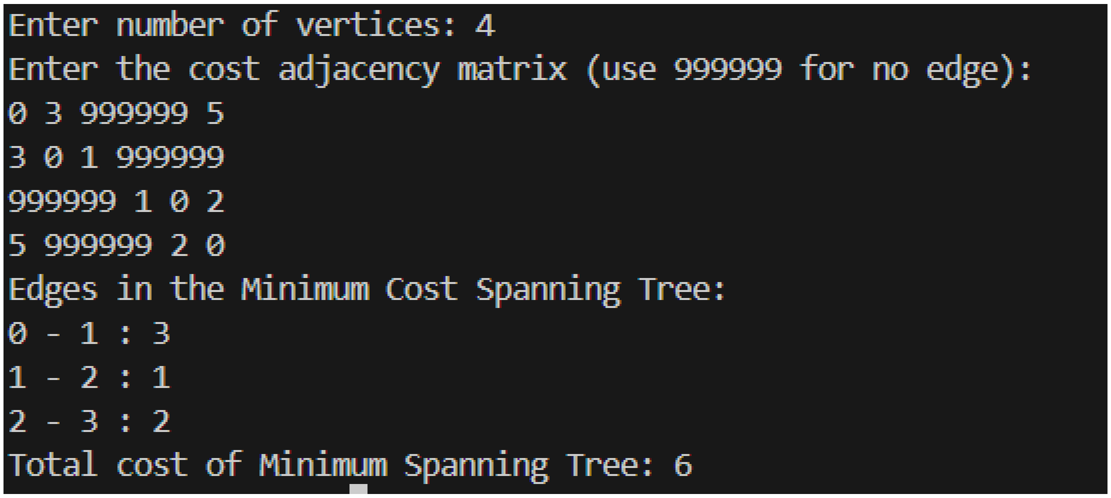
3.1.1 Question
Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

3.1.2 Code

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

#define MAX 100
#define INF 999999

int main() {
  int cost[MAX][MAX], visited[MAX];
  int n, i, j, min, u, v, total_cost = 0;

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

  printf("Enter the cost adjacency matrix (use %d for no edge):\n", INF);
```

```
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0)
                cost[i][j] = INF;
        }
    }

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

    visited[0] = 1;

    printf("Edges in the Minimum Cost Spanning Tree:\n");

    for (i = 0; i < n - 1; i++) {
        min = INF;
        for (j = 0; j < n; j++) {
            if (visited[j]) {
                for (int k = 0; k < n; k++) {
                    if (!visited[k] && cost[j][k] < min) {
                        min = cost[j][k];
                        u = j;
                        v = k;
                    }
                }
            }
        }
        visited[v] = 1;
        printf("%d - %d : %d\n", u, v, min);
        total_cost += min;
    }

    printf("Total cost of Minimum Spanning Tree: %d\n", total_cost);

    return 0;
}
```

3.1.3 Output



```
Enter number of vertices: 4
Enter the cost adjacency matrix (use 999999 for no edge):
0 3 999999 5
3 0 1 999999
999999 1 0 2
5 999999 2 0
Edges in the Minimum Cost Spanning Tree:
0 - 1 : 3
1 - 2 : 1
2 - 3 : 2
Total cost of Minimum Spanning Tree: 6
```

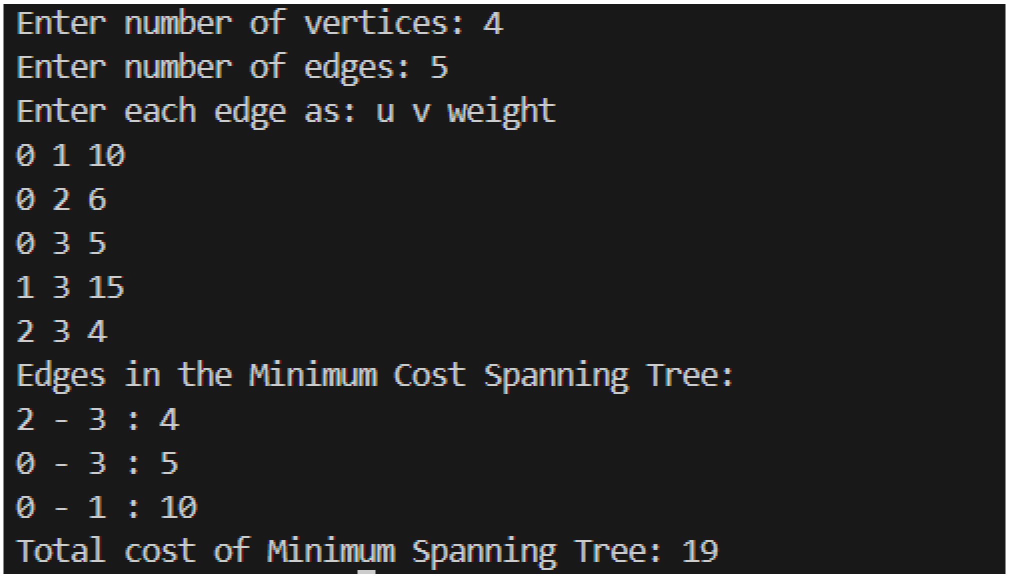
3.2.1 Question

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

3.2.2 Code

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

#define MAX 100
#define INF 999999

typedef struct {
   int u, v, w;
} Edge;

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

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

void kruskal(Edge edges[], int n, int e) {
    int i, j;
    Edge temp;
    for (i = 0; i < e - 1; i++) {
        for (j = 0; j < e - i - 1; j++) {
            if (edges[j].w > edges[j + 1].w) {
                temp = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = temp;
            }
        }
    }

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

    int total_cost = 0;
    printf("Edges in the Minimum Cost Spanning Tree:\n");

    int count = 0;
    for (i = 0; i < e && count < n - 1; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        int w = edges[i].w;

        if (find(u) != find(v)) {
            union_sets(u, v);
            printf("%d - %d : %d\n", u, v, w);
            total_cost += w;
            count++;
        }
    }
    printf("Total cost of Minimum Spanning Tree: %d\n", total_cost);
}

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 as: u v weight\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);
    }

    kruskal(edges, n, e);

    return 0;
}
```

3.2.3 Output



```
Enter number of vertices: 4
Enter number of edges: 5
Enter each edge as: u v weight
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Cost Spanning Tree:
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10
Total cost of Minimum Spanning Tree: 19
```

Lab Program 4

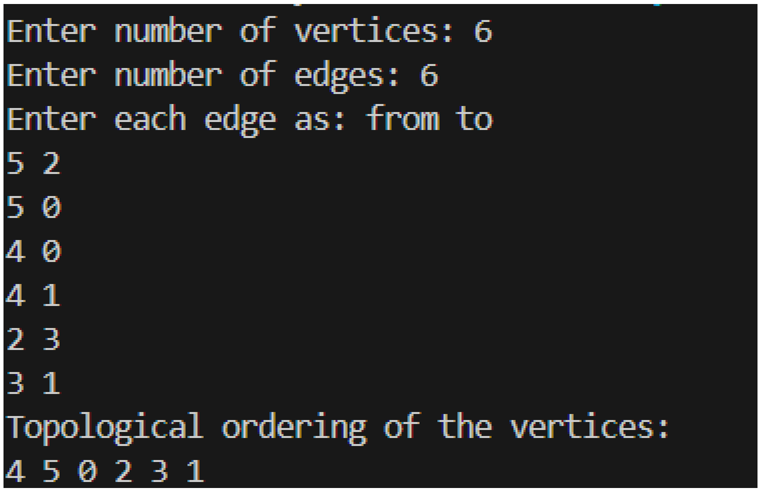
4.1.1 Question

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

4.1.2 Code

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

#define MAX 100

int queue[MAX], front = -1, rear = -1;

void enqueue(int v) {
    if (rear == MAX - 1) return;
    if (front == -1) front = 0;
    queue[++rear] = v;
}

int dequeue() {
    if (front == -1 || front > rear) return -1;
    return queue[front++];
}

int isEmpty() {
    return (front == -1 || front > rear);
}

int main() {
    int n, e, i, j;
    int graph[MAX][MAX] = {0};
    int indegree[MAX] = {0};

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

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

    printf("Enter each edge as: from to\n");
    for (i = 0; i < e; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
        indegree[v]++;
    }
    for (i = 0; i < n; i++) {
        if (indegree[i] == 0)
            enqueue(i);
    }

    int count = 0;
    int topo_order[MAX];

    while (!isEmpty()) {
        int u = dequeue();
        topo_order[count++] = u;

        for (j = 0; j < n; j++) {
            if (graph[u][j]) {
                indegree[j]--;
                if (indegree[j] == 0)
                    enqueue(j);
            }
        }
    }

    if (count != n) {
        printf("Cycle detected. Topological ordering not possible.\n");
    } else {
        printf("Topological ordering of the vertices:\n");
        for (i = 0; i < count; i++) {
            printf("%d ", topo_order[i]);
        }
        printf("\n");
    }

    return 0;
}
```

4.1.3 Output



```
Enter number of vertices: 6
Enter number of edges: 6
Enter each edge as: from to
5 2
5 0
4 0
4 1
2 3
3 1
Topological ordering of the vertices:
4 5 0 2 3 1
```

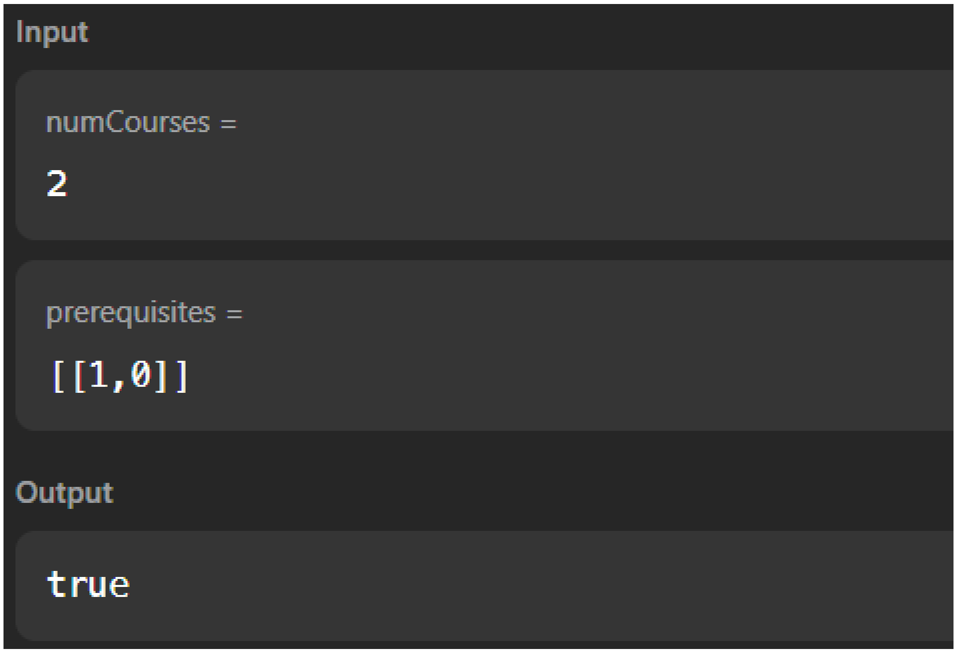
4.2.1 Leetcode Question
Course Schedule

4.2.2 Code

```
bool dfs(int node, int** graph, int* graphColSize, int* visited, int* inStack) {
   visited[node] = 1;
   inStack[node] = 1;

   for (int i = 0; i < graphColSize[node]; i++) {
      int neighbor = graph[node][i];
      if (!visited[neighbor]) {
         if (dfs(neighbor, graph, graphColSize, visited, inStack)) {
            return true;
         }
      } else if (inStack[neighbor]) {
         return true;
      }
   }

   inStack[node] = 0;
   return false;
}

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

   int** graph = (int**)malloc(numCourses * sizeof(int*));
   int* graphColSize = (int*)calloc(numCourses, sizeof(int));
   int* tempSizes = (int*)calloc(numCourses, sizeof(int));

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

   for (int i = 0; i < numCourses; i++) {
      graph[i] = (int*)malloc(tempSizes[i] * sizeof(int));
   }

   for (int i = 0; i < prerequisitesSize; i++) {
      int course = prerequisites[i][0];
      int pre = prerequisites[i][1];
      graph[course][graphColSize[course]++] = pre;
   }
   int* visited = (int*)calloc(numCourses, sizeof(int));
   int* inStack = (int*)calloc(numCourses, sizeof(int));
```

```
    for (int i = 0; i < numCourses; i++) {
        if (!visited[i]) {
            if (dfs(i, graph, graphColSize, visited, inStack)) {

                for (int j = 0; j < numCourses; j++) free(graph[j]);
                free(graph); free(graphColSize); free(visited); free(inStack); free(tempSizes);
                return false;
            }
        }
    }

    for (int i = 0; i < numCourses; i++) free(graph[i]);
    free(graph); free(graphColSize); free(visited); free(inStack); free(tempSizes);
    return true;
}
```

4.2.3 Output



```
Input

numCourses =
2

prerequisites =
[[1,0]]

Output

true
```

Lab Program 5

5.1.1 Question

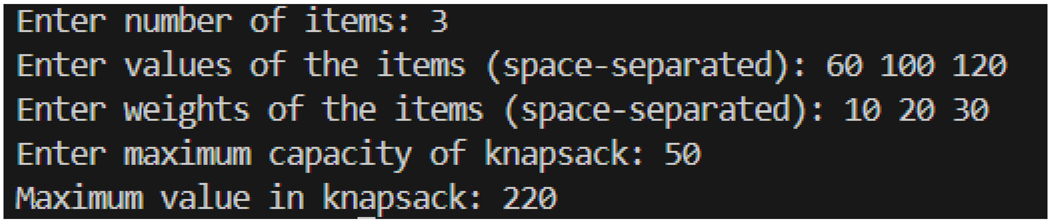
Implement 0/1 Knapsack problem using dynamic programming.

5.1.2 Code

```
#include <stdio.h>

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

int knapsack(int W, int weights[], int values[], int n) {
   int dp[n + 1][W + 1];
   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= W; 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][W];
}

int main() {
   int n, W;

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

   int values[n], weights[n];

   printf("Enter values of the items (space-separated): ");
   for (int i = 0; i < n; i++) scanf("%d", &values[i]);

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

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

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

```
    return 0;
}
```

5.1.3 Output



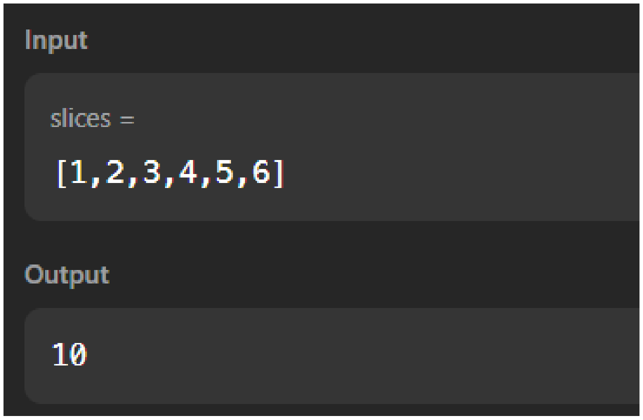
```
Enter number of items: 3
Enter values of the items (space-separated): 60 100 120
Enter weights of the items (space-separated): 10 20 30
Enter maximum capacity of knapsack: 50
Maximum value in knapsack: 220
```

5.2.1 Leetcode Question
Pizza With 3n slices

5.2.2 Code

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

int maxSizeSlicesLinear(int* slices, int start, int end, int n) {
    int len = end - start + 1;
    int dp[len + 1][n + 1];

    for (int i = 0; i <= len; i++)
        for (int j = 0; j <= n; j++)
            dp[i][j] = 0;

    for (int i = 1; i <= len; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == 1)
                dp[i][j] = slices[start + i - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 2][j - 1] + slices[start + i - 1]);
        }
    }
    return dp[len][n];
}

int maxSizeSlices(int* slices, int slicesSize) {
    int n = slicesSize / 3;
    int case1 = maxSizeSlicesLinear(slices, 0, slicesSize - 2, n);
    int case2 = maxSizeSlicesLinear(slices, 1, slicesSize - 1, n);
    return max(case1, case2);
}
```

5.2.3 Output



Input

```
slices =
[1,2,3,4,5,6]
```

Output

```
10
```

Lab program 6

6.1.1 Question

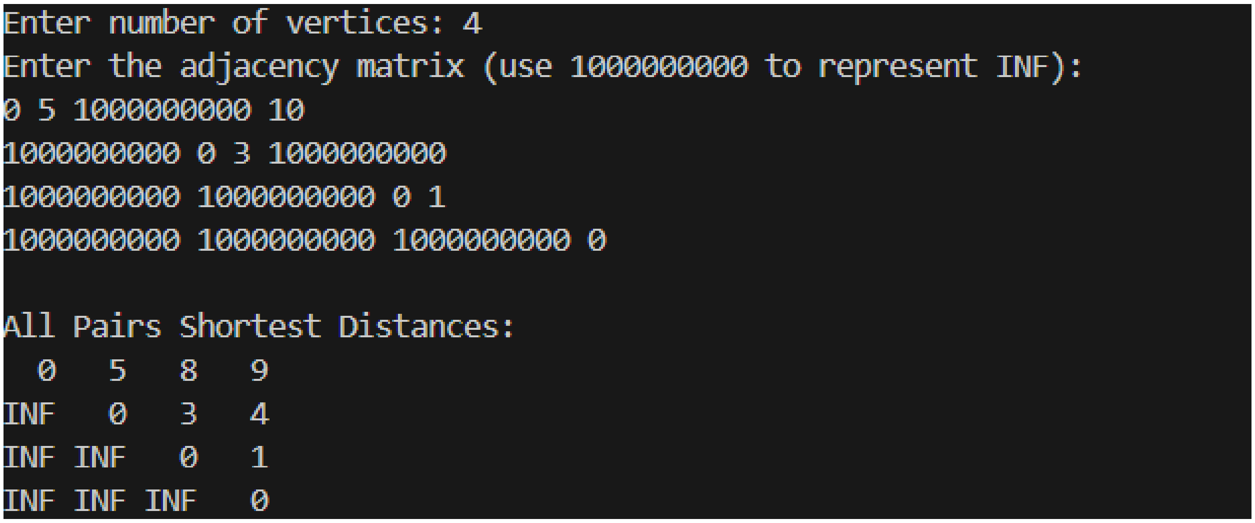
Implement All Pair Shortest paths problem using Floyd's algorithm.

6.1.2 Code

```
#include <stdio.h>

#define INF 1000000000

void floydWarshall(int n, int graph[100][100]) {
   int dist[100][100];

   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         dist[i][j] = graph[i][j];

   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] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];

   printf("\nAll Pairs Shortest Distances:\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         if (dist[i][j] == INF)
            printf("INF ");
         else
            printf("%3d ", dist[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int n;
   int graph[100][100];

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

   printf("Enter the adjacency matrix (use %d to represent INF):\n", INF);
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &graph[i][j]);
```

```
    floydWarshall(n, graph);

    return 0;
}
```

6.1.3 Output



```
Enter number of vertices: 4
Enter the adjacency matrix (use 1000000000 to represent INF):
0 5 1000000000 10
1000000000 0 3 1000000000
1000000000 1000000000 0 1
1000000000 1000000000 1000000000 0

All Pairs Shortest Distances:
  0   5   8   9
INF   0   3   4
INF INF   0   1
INF INF INF   0
```

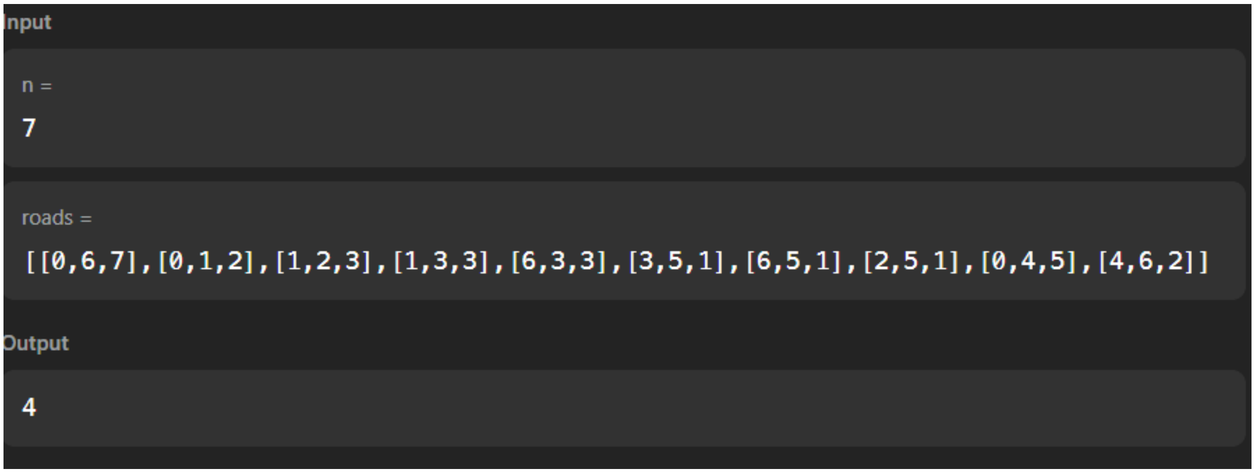
6.2.1 Leetcode question

Number of ways to arrive at a destination.

6.2.2 Code

```
#define MOD 1000000007
#define MAXN 100001
#define MAXEDGES 300000

int head[MAXN], to[MAXEDGES], cost[MAXEDGES], next[MAXEDGES], edgeCount;

int heap[MAXN], heapSize;
long long dist[MAXN];

int ways[MAXN];
int pos[MAXN];

void addEdge(int u, int v, int c) {
    to[edgeCount] = v;
    cost[edgeCount] = c;
    next[edgeCount] = head[u];
    head[u] = edgeCount++;
}

void swap(int i, int j) {
    int tmp = heap[i];
    heap[i] = heap[j];
    heap[j] = tmp;
    pos[heap[i]] = i;
    pos[heap[j]] = j;
}

void push(int node) {
    int i = heapSize++;
    heap[i] = node;
    pos[node] = i;
    while (i > 0 && dist[heap[i]] < dist[heap[(i - 1) / 2]]) {
        swap(i, (i - 1) / 2);
        i = (i - 1) / 2;
    }
}

int pop() {
    int top = heap[0];
    heap[0] = heap[--heapSize];
    pos[heap[0]] = 0;
    int i = 0;
    while (1) {
        int smallest = i, l = 2 * i + 1, r = 2 * i + 2;
        if (l < heapSize && dist[heap[l]] < dist[heap[smallest]]) smallest = l;
        if (r < heapSize && dist[heap[r]] < dist[heap[smallest]]) smallest = r;
        if (smallest == i) break;
        swap(i, smallest);
        i = smallest;
    }
    return top;
}

int countPaths(int n, int** roads, int roadsSize, int* roadsColSize) {
    for (int i = 0; i < n; i++) head[i] = -1;

    edgeCount = 0;
    for (int i = 0; i < roadsSize; i++) {
        int u = roads[i][0], v = roads[i][1], c = roads[i][2];
        addEdge(u, v, c);
        addEdge(v, u, c);
    }

    for (int i = 0; i < n; i++) {
        dist[i] = LLONG_MAX;
        ways[i] = 0;
        pos[i] = -1;
    }

    dist[0] = 0;
    ways[0] = 1;
    heapSize = 0;
    push(0);

    while (heapSize > 0) {
        int u = pop();
        for (int e = head[u]; e != -1; e = next[e]) {
            int v = to[e];
            long long d = dist[u] + cost[e];
            if (d < dist[v]) {
                dist[v] = d;
                ways[v] = ways[u];
                if (pos[v] == -1)
                    push(v);
                else {
                    int i = pos[v];
                    while (i > 0 && dist[heap[i]] < dist[heap[(i - 1) / 2]]) {
                        swap(i, (i - 1) / 2);
                        i = (i - 1) / 2;
                    }
                }
            } else if (d == dist[v]) {
                ways[v] = (ways[v] + ways[u]) % MOD;
            }
        }
    }

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

6.2.3 Output



```
Input
n =
7
roads =
[[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],[2,5,1],[0,4,5],[4,6,2]]
Output
4
```

Lab program 7

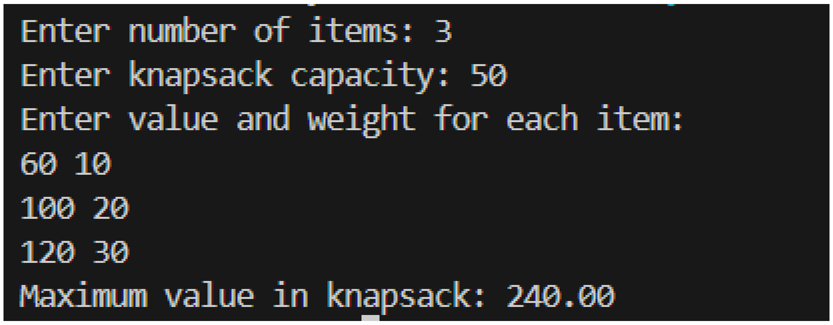
7.1.1 Question

Implement Fractional Knapsack using Greedy technique.

7.1.2 Code

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

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

int compare(const void *a, const void *b) {
    double r1 = ((Item *)b)->ratio;
    double r2 = ((Item *)a)->ratio;
    return (r1 > r2) - (r1 < r2);
}

int main() {
    int n;
    double capacity;

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

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

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

    qsort(items, n, sizeof(Item), compare);

    double totalValue = 0.0;

    for (int i = 0; i < n && capacity > 0; i++) {
        if (items[i].weight <= capacity) {
            totalValue += items[i].value;
            capacity -= items[i].weight;
        } else {
            totalValue += items[i].ratio * capacity;
            capacity = 0;
        }
    }

    printf("Maximum value in knapsack: %.2lf\n", totalValue);

    return 0;
}
```

7.1.3 Output



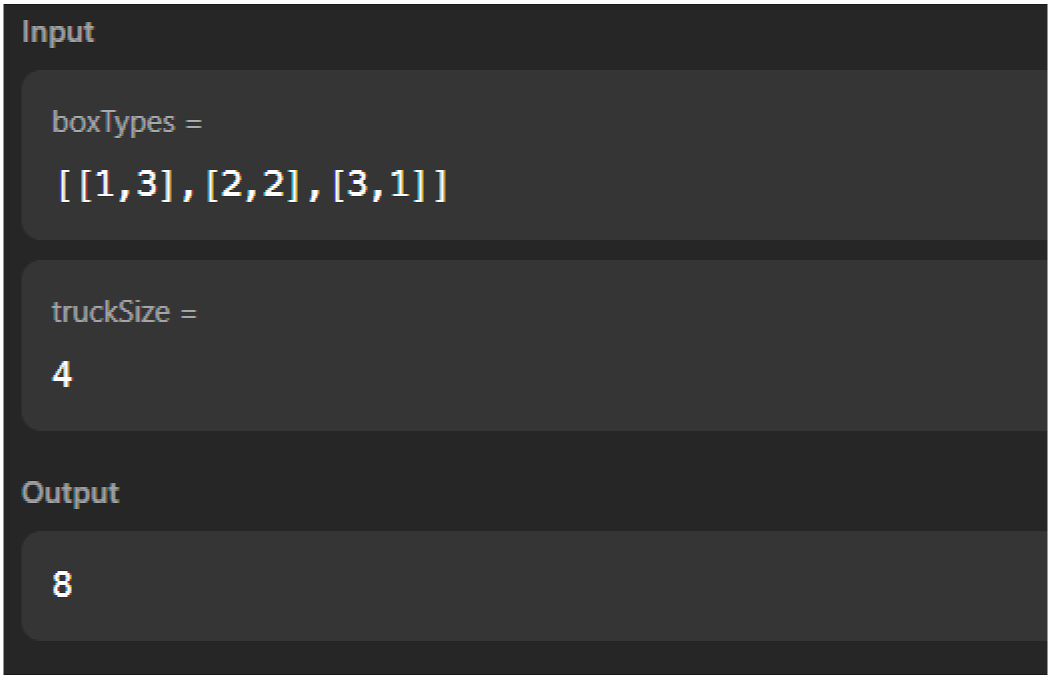
```
Enter number of items: 3
Enter knapsack capacity: 50
Enter value and weight for each item:
60 10
100 20
120 30
Maximum value in knapsack: 240.00
```

7.2.1 Leetcode Questions
Maximum units on a truck

7.2.2 Code

```
int compare(const void *a, const void *b) {
   int u1 = ((int **)b)[0][1];
   int u2 = ((int **)a)[0][1];
   return u1 - u2;
}

int maximumUnits(int** boxTypes, int boxTypesSize, int* boxTypesColSize, int truckSize) {
   qsort(boxTypes, boxTypesSize, sizeof(int *), compare);

   int totalUnits = 0;
   for (int i = 0; i < boxTypesSize && truckSize > 0; i++) {
      int boxesToTake = boxTypes[i][0] < truckSize ? boxTypes[i][0] : truckSize;
      totalUnits += boxesToTake * boxTypes[i][1];
      truckSize -= boxesToTake;
   }
   return totalUnits;
}
```

7.2.3 Output



```
Input

boxTypes =
[[1,3],[2,2],[3,1]]

truckSize =
4

Output

8
```

Lab Program 8

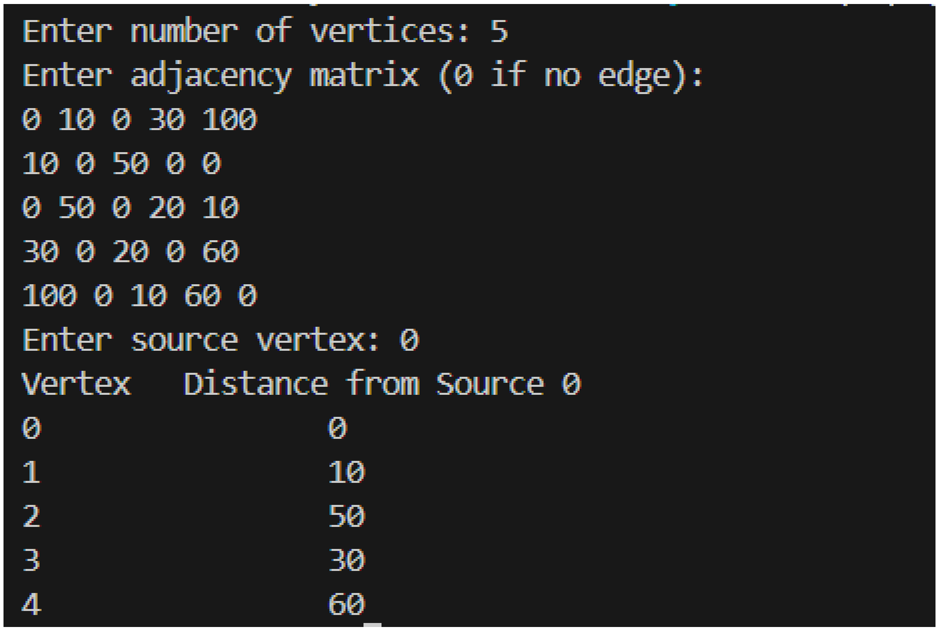
8.1.1 Question
Dijkstra's Algorithm

8.1.2 Code

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

#define V 100  // Max number of vertices

int minDistance(int dist[], int visited[], int n) {
   int min = INT_MAX, min_index = -1;
   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[V][V], int n, int src) {
   int dist[V];     // Shortest distance from src to i
   int visited[V];  // Visited vertices

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

   dist[src] = 0;

   for (int count = 0; count < n - 1; count++) {
      int u = minDistance(dist, visited, n);
      if (u == -1) break;  // All reachable nodes are processed

      visited[u] = 1;

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

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

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

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

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

    dijkstra(graph, n, src);

    return 0;
}
```

8.1.3 Output



```
Enter number of vertices: 5
Enter adjacency matrix (0 if no edge):
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter source vertex: 0
Vertex   Distance from Source 0
0                0
1                10
2                50
3                30
4                60
```

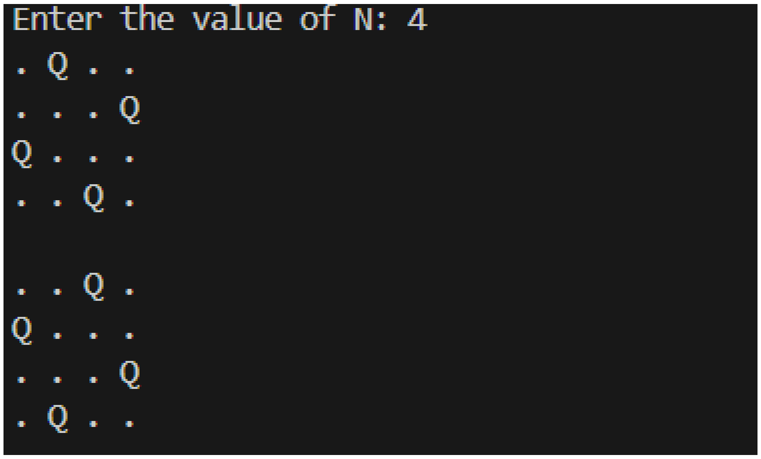
Lab Program 9

9.1.1 Question
N Queens

9.1.2 Code

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

int *board;
int N;

int isSafe(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i] == col ||
            abs(board[i] - col) == abs(i - row)) {
            return 0;
        }
    }
    return 1;
}

void printSolution() {
    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");
}

void solveNQueens(int row) {
    if (row == N) {
        printSolution();
        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: ");
    scanf("%d", &N);

    board = (int *)malloc(N * sizeof(int));

    solveNQueens(0);

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

9.1.3 Output



```
Enter the value of N: 4
. Q . .
. . . Q
Q . . .
. . Q .

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

Lab Program 10

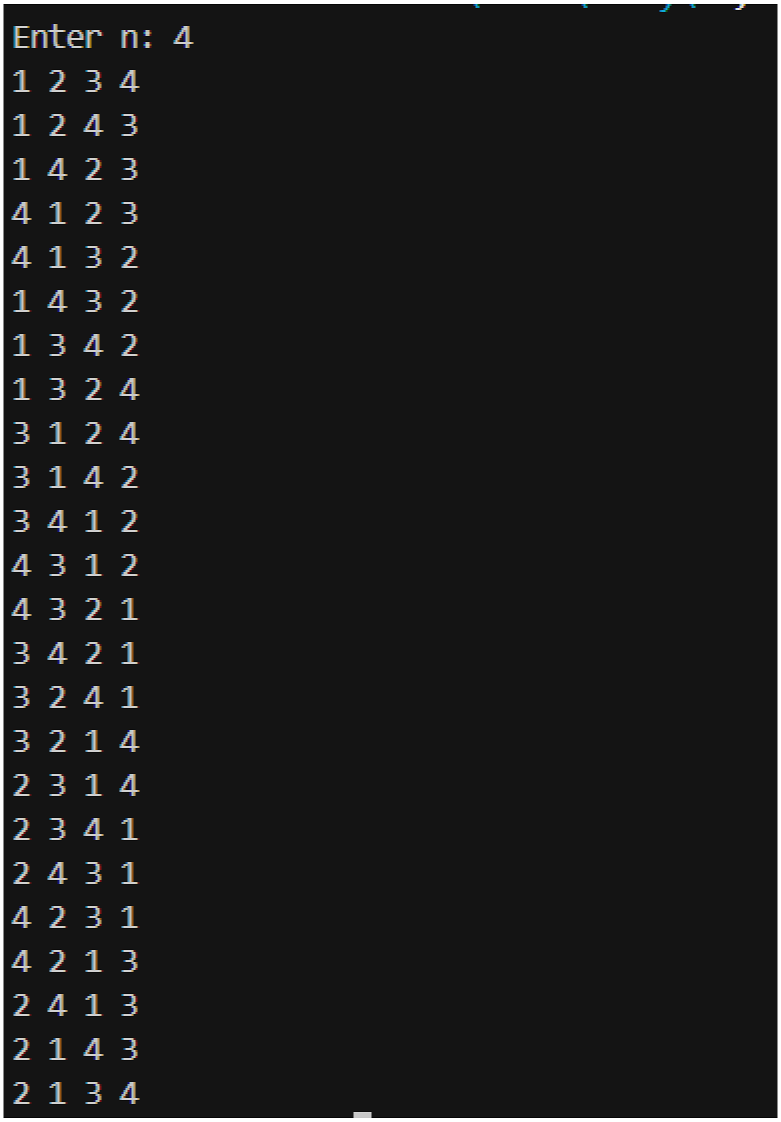
10.1.1 Question
Implement Johnson Trotter algorithm to generate permutations.

10.1.2 Code

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

#define LEFT 0
#define RIGHT 1

int n;

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

int getLargestMobile(int *arr, int *dir) {
    int largestMobileIndex = -1;
    int largestMobile = 0;

    for (int i = 0; i < n; i++) {
        int nextIndex = (dir[i] == LEFT) ? i - 1 : i + 1;

        if (nextIndex >= 0 && nextIndex < n) {
            if (arr[i] > arr[nextIndex] && arr[i] > largestMobile) {
                largestMobile = arr[i];
                largestMobileIndex = i;
            }
        }
    }
    return largestMobileIndex;
}

void johnsonTrotter() {
    int *arr = malloc(n * sizeof(int));
    int *dir = malloc(n * sizeof(int));

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

    printPermutation(arr);
```

```
  while (1) {
    int largestMobileIndex = getLargestMobile(arr, dir);
    if (largestMobileIndex == -1)
      break; // No more mobile integer, done

    int swapIndex = (dir[largestMobileIndex] == LEFT) ? largestMobileIndex - 1 :
largestMobileIndex + 1;

    int temp = arr[largestMobileIndex];
    arr[largestMobileIndex] = arr[swapIndex];
    arr[swapIndex] = temp;

    temp = dir[largestMobileIndex];
    dir[largestMobileIndex] = dir[swapIndex];
    dir[swapIndex] = temp;

    largestMobileIndex = swapIndex;

    for (int i = 0; i < n; i++) {
      if (arr[i] > arr[largestMobileIndex]) {
        dir[i] = (dir[i] == LEFT) ? RIGHT : LEFT;
      }
    }

    printPermutation(arr);
  }

  free(arr);
  free(dir);
}

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

  johnsonTrotter();

  return 0;
}
```

10.1.3 Output



```
Enter n: 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
```

Lab Program 11

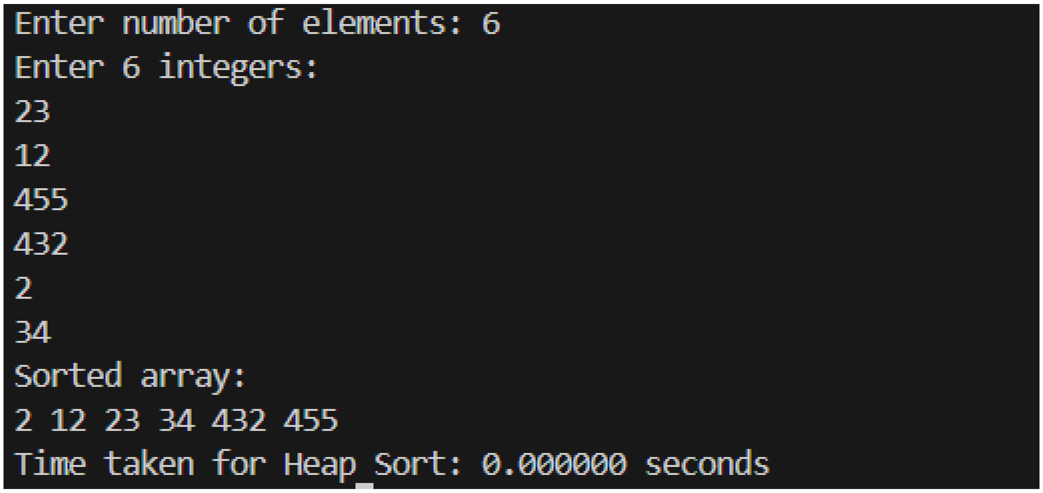
11.1.1 Question

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

11.1.2 Code

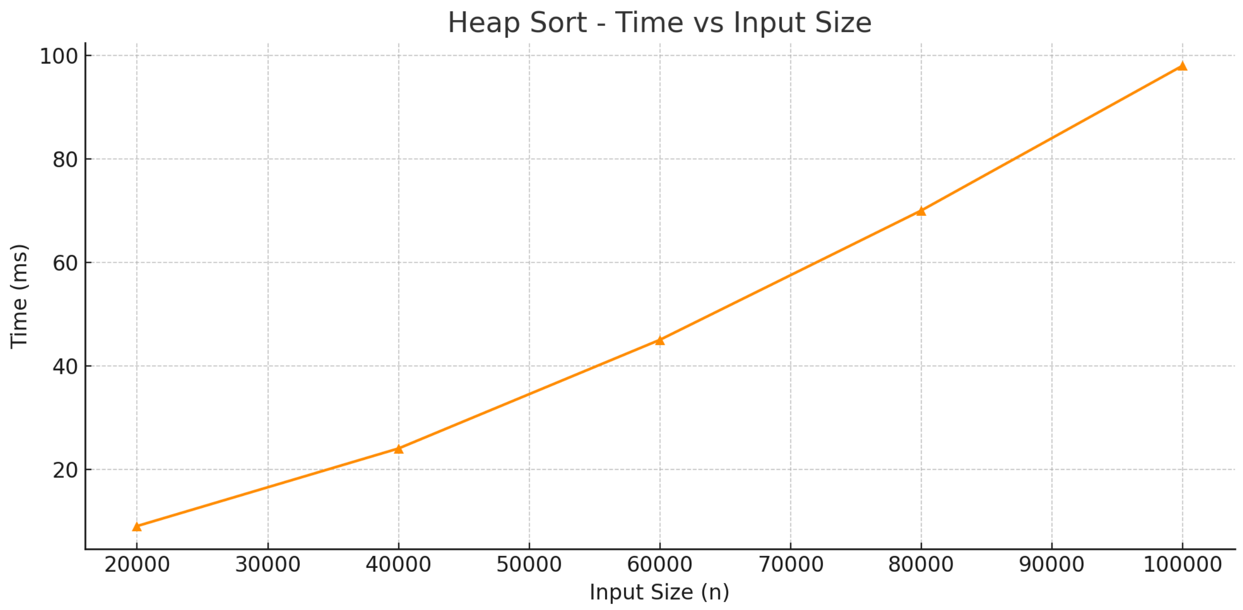
```
#include <stdio.h>
#include <time.h>

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

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

    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 = 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: %f seconds\n", time_taken);

    return 0;
}
```

11.1.3 Output



```
Enter number of elements: 6
Enter 6 integers:
23
12
455
432
2
34
Sorted array:
2 12 23 34 432 455
Time taken for Heap Sort: 0.000000 seconds
```

11.1.4 Graph


Heap Sort - Time vs Input Size
Time (ms)
100
80
60
40
20
20000
30000
40000
50000
60000
70000
80000
90000
100000
Input Size (n)