VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Shankar shivappa pujar (1BM23CS309)

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 Shankar Shivappa Pujar**(1BM23CS309)**, 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. Sarala D V
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
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:

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

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

#define MAX 100

int graph[MAX][MAX];
int visited[MAX];
int stack[MAX];
int top = -1;

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

void topologicalSort(int vertices) {
    for (int i = 0; i < vertices; i++)
        visited[i] = 0;

    for (int i = 0; i < vertices; i++) {
        if (!visited[i])
            dfs(i, vertices);
    }
```

```
    printf("Topological Order: ");
    while (top >= 0)
        printf("%d ", stack[top--]);
}

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

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

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

    printf("Enter edges (source destination):\n");
    for (int i = 0; i < edges; i++) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        graph[src][dest] = 1;
    }

    topologicalSort(vertices);

    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 6
Enter number of edges: 6
Enter edges (source destination):
5 2
5 0
4 0
4 1
2 3
3 1
Topological Order: 5 4 2 3 1 0
Process returned 0 (0x0)   execution time : 32.122 s
Press any key to continue.
```

1.1 LeetCode Program related to Topological sorting

Code:

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

#define MAX 100

struct Node {
    int vertex;
    struct Node* next;
};

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

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

```
}

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

struct Graph {
    int vertices;
    struct Node** adjList;
    int* inDegree;
};

struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->vertices = vertices;
    graph->adjList = malloc(vertices * sizeof(struct Node*));
    graph->inDegree = calloc(vertices, sizeof(int));
    for (int i = 0; i < vertices; i++)
        graph->adjList[i] = NULL;
    return graph;
}
```

```
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;
    graph->inDegree[dest]++;
}

void topologicalSort(struct Graph* graph) {
    for (int i = 0; i < graph->vertices; i++) {
        if (graph->inDegree[i] == 0)
            enqueue(i);
    }

    int count = 0;
    int topOrder[MAX];

    while (front <= rear) {
        int u = dequeue();
        topOrder[count++] = u;

        struct Node* temp = graph->adjList[u];
        while (temp) {
            int v = temp->vertex;
            graph->inDegree[v]--;
            if (graph->inDegree[v] == 0)
                enqueue(v);
            temp = temp->next;
        }
```

```
    }

    if (count != graph->vertices) {
        printf("Cycle detected! Topological sort not possible.\n");
        return;
    }

    printf("Topological Sort Order: ");
    for (int i = 0; i < count; i++)
        printf("%d ", topOrder[i]);
    printf("\n");
}

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

    struct Graph* graph = createGraph(vertices);

    printf("Enter edges (source destination):\n");
    for (int i = 0; i < edges; i++) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    topologicalSort(graph);
```

```
    return 0;
}
```

OUTPUT

```
Enter number of vertices and edges: 6 6
Enter edges (source destination):
5 2
5 0
4 0
4 1
2 3
3 1
Topological Sort Order: 4 5 0 2 3 1

Process returned 0 (0x0)   execution time : 27.454 s
Press any key to continue.
```

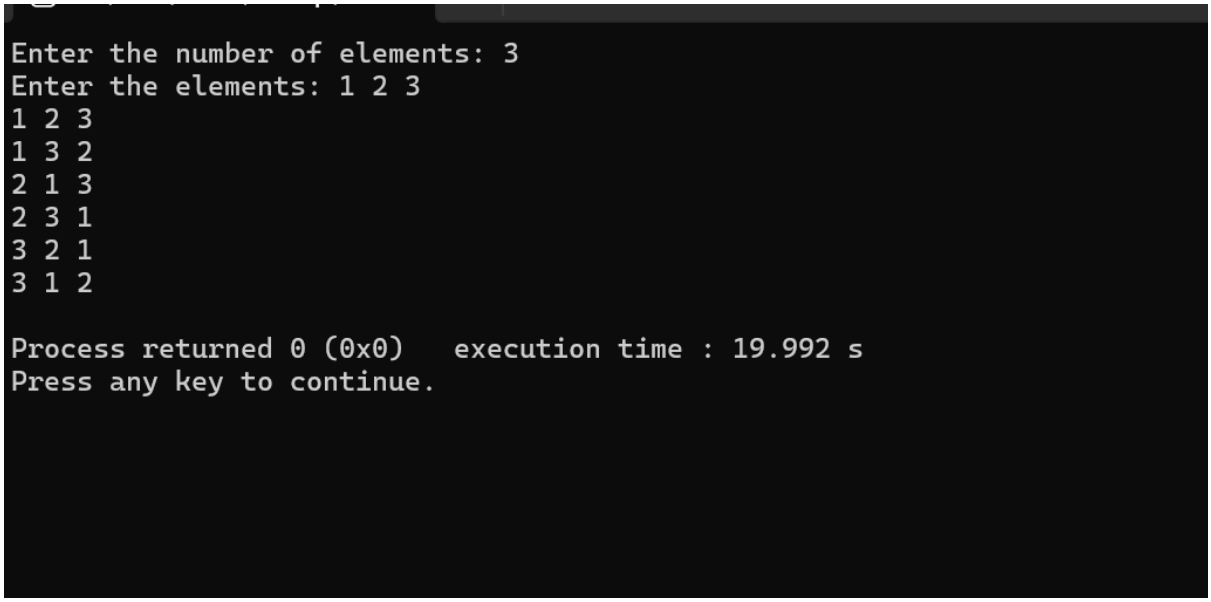
LAB PROGRAM 2:

Implement Johnson Trotter algorithm to generate permutations.

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

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

void generatePermutations(int arr[], int start, int end) {
    if (start == end) {
        for (int i = 0; i <= end; i++) {
```

```
            printf("%d ", arr[i]);
        }
        printf("\n");
    } else {
        for (int i = start; i <= end; i++) {
            swap(&arr[start], &arr[i]);
            generatePermutations(arr, start + 1, end);
            swap(&arr[start], &arr[i]); // backtrack
        }
    }
}

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

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

    generatePermutations(arr, 0, n - 1);
    free(arr);
    return 0;
}
```

OUTPUT



LAB PROGRAM 3:

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

```
#include <stdio.h>

#include <time.h>


int a[20], n;


void merge_sort(int [], int, int);

void merge(int [], int, int, int);


int main() {

   int i;

   clock_t start, end;

   double time_taken;



   printf("Enter the number of elements: ");
```

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


printf("Enter the array elements: ");
for (i = 0; i < n; i++) {
   scanf("%d", &a[i]);
}


start = clock();


merge_sort(a, 0, n - 1);


end = clock();


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


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


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

```
    return 0;
}

void merge_sort(int a[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;



        merge_sort(a, low, mid);
        merge_sort(a, mid + 1, high);



        merge(a, low, mid, high);
    }
}

void merge(int a[], int low, int mid, int high) {
    int i = low, j = mid + 1, k = low;
    int c[20];

    while (i <= mid && j <= high) {
        if (a[i] < a[j]) {
            c[k++] = a[i++];
        } else {
            c[k++] = a[j++];
        }
    }

    while (i <= mid) {
```

```
        c[k++] = a[i++];
    }



    while (j <= high) {
        c[k++] = a[j++];
    }


    for (i = low; i <= high; i++) {
        a[i] = c[i];
    }
}
```

OUTPUT

```
Enter the number of elements: 4
Enter the array elements: 23
32
45
56
Sorted array: 23 32 45 56
Time taken to sort: 0.000000 seconds

Process returned 0 (0x0)   execution time : 11.756 s
Press any key to continue.
```

3.1 LeetCode Program related to sorting.

Code

```
#include <stdio.h>


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

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

    int L[n1], R[n2];

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

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];

    while (j < n2)
        arr[k++] = R[j++];
}

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

    mergeSort(arr, 0, n - 1);

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

    return 0;
}
```

OUTUT

```
Enter number of elements: 5
Enter 5 elements:
3 1 4 1 5
Sorted Array:
1 1 3 4 5

Process returned 0 (0x0)   execution time : 5.161 s
Press any key to continue.
```

LAB PROGRAM 4:

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

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX 5000


void quicksort(int[], int, int);

int partition(int[], int, int);


int main() {

   int i, n, a[MAX], ch = 1;

   clock_t start, end;


   srand(time(0));


   while (ch) {

      printf("\nEnter the number of elements: ");
```

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

    if (n <= 0 || n > MAX) {
        printf("Invalid input! Please enter a number between 1 and %d.\n", MAX);
        continue;
    }

    for (i = 0; i < n; i++)
        a[i] = rand() % 200;

    printf("The randomly generated array is:\n");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    start = clock();
    quicksort(a, 0, n - 1);
    end = clock();

    printf("\n\nThe sorted array elements are:\n");
    for (i = 0; i < n; i++)
        printf("%d\n", a[i]);

    printf("\nTime taken = %f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    printf("\nDo you wish to continue? (0 for No, 1 for Yes): ");
    scanf("%d", &ch);
}

return 0;
```

```
}

void quicksort(int a[], int low, int high) {
    if (low < high) {
        int mid = partition(a, low, high);
        quicksort(a, low, mid - 1);
        quicksort(a, mid + 1, high);
    }
}

int partition(int a[], int low, int high) {
    int key = a[low], i = low + 1, j = high, temp;

    while (i <= j) {
        while (i <= high && a[i] <= key) i++;
        while (a[j] > key) j--;

        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    temp = a[j];
    a[j] = a[low];
    a[low] = temp;

    return j;
```

}

OUTPUT

```
Enter the number of elements: 4
The randomly generated array is:
165 41 11 61

The sorted array elements are:
11
41
61
165

Time taken = 0.000000 seconds

Do you wish to continue? (0 for No, 1 for Yes):
```

4.1 LeetCode Program related to sorting.

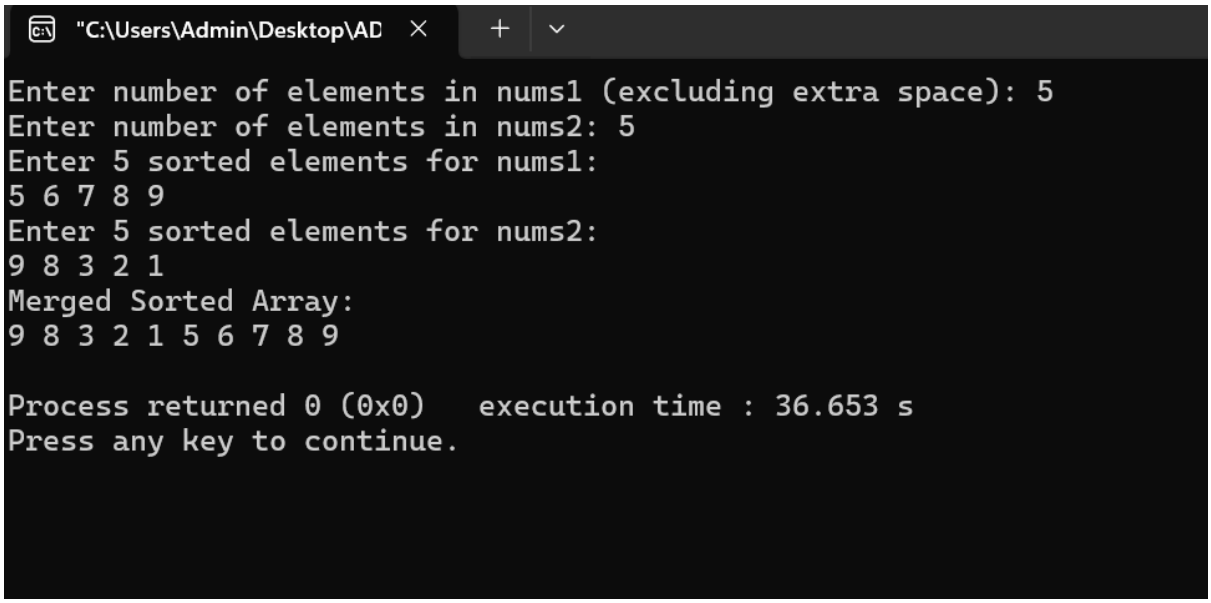
Code

```
#include <stdio.h>


void merge(int* nums1, int m, int* nums2, int n) {
    int i = m - 1;
    int j = n - 1;
    int k = m + n - 1;


    while (i >= 0 && j >= 0) {
        if (nums1[i] > nums2[j])
            nums1[k--] = nums1[i--];
        else
            nums1[k--] = nums2[j--];
    }
```

```
    while (j >= 0)
        nums1[k--] = nums2[j--];
}

int main() {
    int m, n;
    printf("Enter number of elements in nums1 (excluding extra space): ");
    scanf("%d", &m);

    printf("Enter number of elements in nums2: ");
    scanf("%d", &n);

    int nums1[m + n], nums2[n];

    printf("Enter %d sorted elements for nums1:\n", m);
    for (int i = 0; i < m; i++)
        scanf("%d", &nums1[i]);

    for (int i = m; i < m + n; i++)
        nums1[i] = 0;

    printf("Enter %d sorted elements for nums2:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &nums2[i]);

    merge(nums1, m, nums2, n);

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

```
        printf("%d ", nums1[i]);
    printf("\n");


    return 0;
}
```

OUTPUT



LAB PROGRAM 5:

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

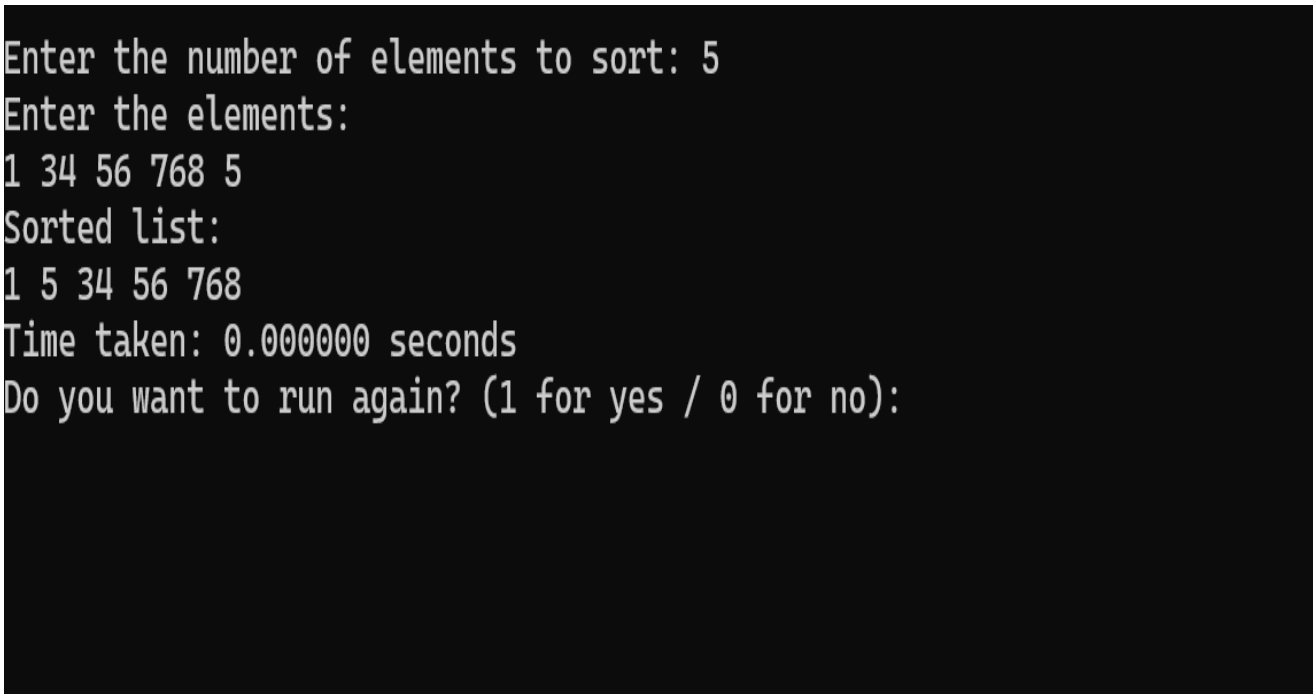
```
#include <stdio.h>
#include <time.h>


#define MAX 20


void heapcom(int a[], int n) {
    int i, j, k, item;
    for (i = 1; i <= n; i++) {
        item = a[i];
        j = i;
```

```
        k = j / 2;
        while (k != 0 && item > a[k]) {
            a[j] = a[k];
            j = k;
            k = j / 2;
        }
        a[j] = item;
    }
}

void adjust(int a[], int n) {
    int item, i, j;
    j = 1;
    item = a[j];
    i = 2 * j;
    while (i < n) {
        if ((i + 1) < n && a[i] < a[i + 1]) {
            i++;
        }
        if (item < a[i]) {
            a[j] = a[i];
            j = i;
            i = 2 * j;
        } else {
            break;
        }
    }
    a[j] = item;
}
```

```
void heapsort(int a[], int n) {
    int i, temp;
    heapcom(a, n);
    for (i = n; i >= 1; i--) {
        temp = a[1];
        a[1] = a[i];
        a[i] = temp;
        adjust(a, i);
    }
}

int main() {
    int i, n, a[MAX], ch = 1;
    clock_t start, end;

    while (ch) {
        printf("\nEnter the number of elements to sort: ");
        scanf("%d", &n);

        printf("Enter the elements:\n");
        for (i = 1; i <= n; i++)
            scanf("%d", &a[i]);

        start = clock();
        heapsort(a, n);
        end = clock();

        printf("Sorted list:\n");
```

```
    for (i = 1; i <= n; i++)
        printf("%d ", a[i]);

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

    printf("Do you want to run again? (1 for yes / 0 for no): ");
    scanf("%d", &ch);
  }

  return 0;
}
```

OUTPUT



LAB PROGRAM 6:

Implement 0/1 Knapsack problem using dynamic programming

```
#include <stdio.h>

int i, j, n, c, w[10], p[10], v[10][10];

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

void knapsack(int n, int w[10], int p[10], int c) {
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= c; 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]);
      }
   }

   printf("\n\nMaximum Profit is: %d\n", v[n][c]);
   printf("\nDP Table:\n\n");
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= c; j++) {
         printf("%d\t", v[i][j]);
      }
      printf("\n");
```

```
    }
}

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

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

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

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

    knapsack(n, w, p, c);
    return 0;
}
```

OUTPUT

```
Enter the number of objects: 3
Enter the weights: 1 2 3
Enter the profits: 10 20 30
Enter the capacity: 5


Maximum Profit is: 50

DP Table:

0       0       0       0       0       0
0       10      10      10      10      10
0       10      20      30      30      30
0       10      20      30      40      50

Process returned 0 (0x0)   execution time : 82.264 s
Press any key to continue.
```

6.1 LeetCode Program related to Knapsack problem or Dynamic Programming.

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

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

int knapsack(int W, int wt[], int val[], int n) {
   int **dp = (int **)malloc((n + 1) * sizeof(int *));
   for (int i = 0; i <= n; i++) {
      dp[i] = (int *)malloc((W + 1) * sizeof(int));
   }

   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 (wt[i - 1] <= w)
            dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
        else
            dp[i][w] = dp[i - 1][w];
    }
  }

  int maxValue = dp[n][W];

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

  return maxValue;
}

int main() {
  int n, W;
  printf("Enter number of items and knapsack capacity: ");
  scanf("%d %d", &n, &W);

  int *wt = (int *)malloc(n * sizeof(int));
  int *val = (int *)malloc(n * sizeof(int));

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

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

    int maxValue = knapsack(W, wt, val, n);
    printf("Maximum value that can be obtained: %d\n", maxValue);

    free(wt);
    free(val);

    return 0;
}
```

OUTPUT

```
Enter number of items and knapsack capacity: 4 7
Enter weights of the items:
1 3 4 5
Enter values of the items:
1 4 5 7
Maximum value that can be obtained: 9

Process returned 0 (0x0)   execution time : 37.840 s
Press any key to continue.
```

LAB PROGRAM 7:

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

```
#include <stdio.h>

#define MAX 10
#define INF 99999
```

```
void floydWarshall(int dist[MAX][MAX], int n) {
    int i, j, k;
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
}

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

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

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

    floydWarshall(graph, n);

    printf("All-Pairs Shortest Path Matrix:\n");
    for (i = 0; i < n; i++) {
```

```
        for (j = 0; j < n; j++) {
            if (graph[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", graph[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4
Enter the adjacency matrix (use 99999 for INF):
0 3 99999 7
8 0 2 99999
5 99999 0 1
2 99999 99999 0
All-Pairs Shortest Path Matrix:
  0   3   5   6
  5   0   2   3
  3   6   0   1
  2   5   7   0

Process returned 0 (0x0)   execution time : 46.567 s
Press any key to continue.
```

7.1 LeetCode Program related to shortest distance calculation

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

#define MAX 100
#define INF INT_MAX
```

```
int minDistance(int dist[], int visited[], int n) {
    int min = INF, 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[MAX][MAX], int n, int src) {
    int dist[n];
    int visited[n];

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

    dist[src] = 0;

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

        visited[u] = 1;

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

```
            if (!visited[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v])
{
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    printf("Shortest distances from node %d:\n", src);
    for (int i = 0; i < n; i++) {
        if (dist[i] == INF)
            printf("Node %d: Unreachable\n", i);
        else
            printf("Node %d: %d\n", i, dist[i]);
    }
}

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

    int graph[MAX][MAX] = {0};

    printf("Enter edges (u v weight):\n");
    for (int i = 0; i < e; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        graph[u][v] = w;
        // For undirected graph, also set: graph[v][u] = w;
    }
```

```
    int src;
    printf("Enter source node: ");
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

```
Enter number of nodes and edges: 4 4
Enter edges (u v weight):
0 1 1
0 2 4
1 2 2
2 3 1
Enter source node: 0
Shortest distances from node 0:
Node 0: 0
Node 1: 1
Node 2: 3
Node 3: 4

Process returned 0 (0x0)   execution time : 46.809 s
Press any key to continue.
```

LAB PROGRAM 8:

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

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

#define MAX 100

int minKey(int key[], int mstSet[], int V) {
```

```
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

void printMST(int parent[], int graph[MAX][MAX], int V) {
    int totalCost = 0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
        totalCost += graph[i][parent[i]];
    }
    printf("Total cost of Minimum Spanning Tree: %d\n", totalCost);
}

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

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;

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

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

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

    printMST(parent, graph, V);
}

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

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

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

            if (i != j && graph[i][j] == 0)
                graph[i][j] = INT_MAX;
        }
    }
```

```
    primMST(graph, V);

    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 5
Enter the adjacency matrix (use 0 if no edge):
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total cost of Minimum Spanning Tree: 16

Process returned 0 (0x0)   execution time : 18.904 s
Press any key to continue.
```

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

Code

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

#define MAX 100

typedef struct {
    int u, v, weight;
} Edge;
```

```
int parent[MAX];

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

void unionSet(int u, int v) {
    int set_u = find(u);
    int set_v = find(v);
    parent[set_v] = set_u;
}

void kruskal(Edge edges[], int V, int E) {
    Edge result[MAX];
    int totalCost = 0, count = 0;

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

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

        if (find(u) != find(v)) {
            result[count++] = edges[i];
            totalCost += edges[i].weight;
            unionSet(u, v);
```

```
        }
    }

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

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

int main() {
    int V, E;
    Edge edges[MAX];

    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);

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

    qsort(edges, E, sizeof(Edge), compare);
    kruskal(edges, V, E);
```

```
    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
999 2   3  999
2  999  999 4
3  999  999 5
999 4   5  999
Edges of the minimal spanning tree:
(0, 1) (0, 2) (1, 3)
Sum of minimal spanning tree: 9

Process returned 0 (0x0)   execution time : 61.687 s
Press any key to continue.
```

LAB PROGRAM 9

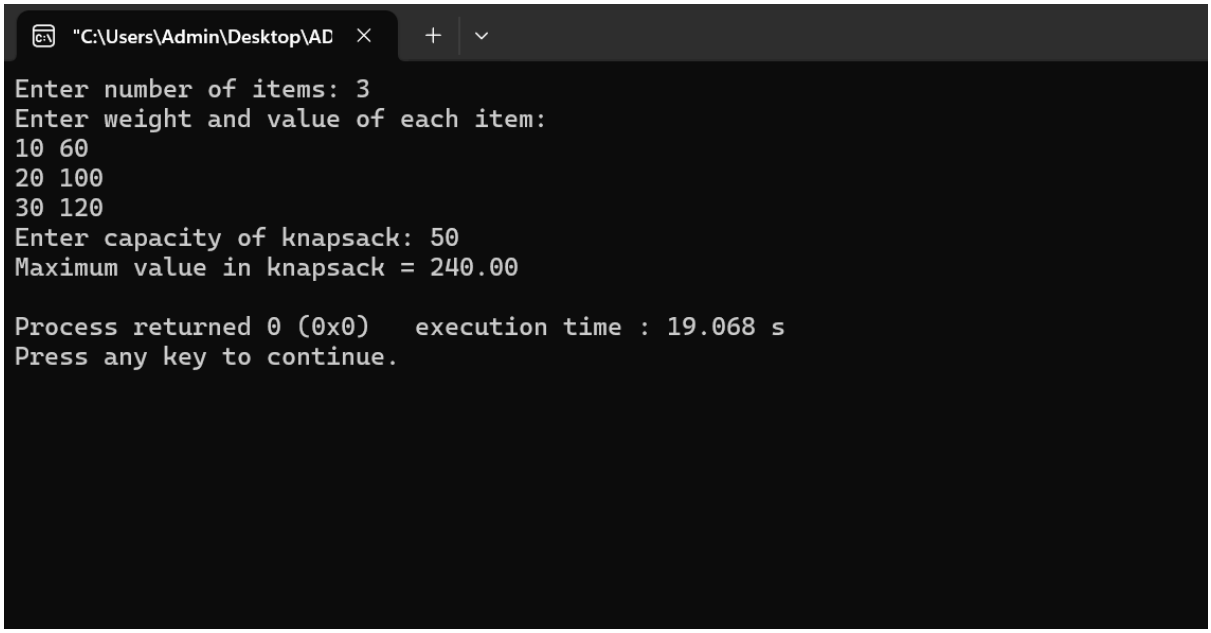
Implement Fractional Knapsack using Greedy technique.

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

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

int compare(const void *a, const void *b) {
    Item *item1 = (Item *)a;
    Item *item2 = (Item *)b;
    if (item1->ratio < item2->ratio)
        return 1;
    else if (item1->ratio > item2->ratio)
        return -1;
```

```
    else
        return 0;
}

float fractionalKnapsack(Item items[], int n, int capacity) {
    qsort(items, n, sizeof(Item), compare);

    float totalValue = 0.0;
    int currentWeight = 0;

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

    return totalValue;
}

int main() {
    int n, capacity;

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

```
    Item items[n];

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

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

    float maxValue = fractionalKnapsack(items, n, capacity);

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

    return 0;
}
```

OUTPUT



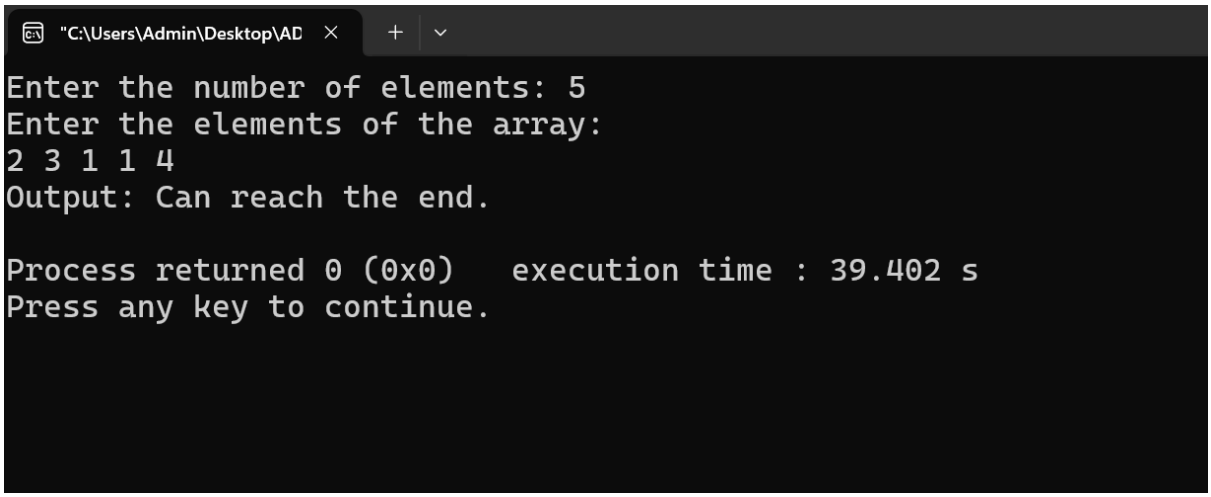
9.1 LeetCode Program related to Greedy Technique algorithms.

Code

```
#include <stdio.h>

int canJump(int* nums, int numsSize) {
    int maxReach = 0;

    for (int i = 0; i < numsSize; i++) {
        if (i > maxReach)
            return 0;  // Cannot reach this index
        if (i + nums[i] > maxReach)
            maxReach = i + nums[i];
    }

    return 1;
}

int main() {
    int nums[100], n;

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

    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &nums[i]);

    if (canJump(nums, n))
        printf("Output: Can reach the end.\n");
```

```
    else
        printf("Output: Cannot reach the end.\n");

    return 0;
}
```

OUTPUT



LAB PROGRAM 1O:

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

```
#include <stdio.h>

#define MAX 20
#define INF 999

int main() {
    int i, j, n, v, k, min, u, c[MAX][MAX], s[MAX], d[MAX];

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

    printf("Enter the cost adjacency matrix (999 for no edge):\n");
```

```
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        scanf("%d", &c[i][j]);

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

for (i = 1; i <= n; i++) {
    s[i] = 0;
    d[i] = c[v][i];
}

d[v] = 0;
s[v] = 1;

for (k = 2; k <= n; k++) {
    min = INF;
    for (i = 1; i <= n; i++) {
        if (s[i] == 0 && d[i] < min) {
            min = d[i];
            u = i;
        }
    }
    s[u] = 1;
    for (i = 1; i <= n; i++) {
        if (s[i] == 0 && d[i] > d[u] + c[u][i]) {
            d[i] = d[u] + c[u][i];
        }
    }
```

```
    }

    printf("The shortest distances from vertex %d are:\n", v);
    for (i = 1; i <= n; i++) {
        printf("%d --> %d = %d\n", v, i, d[i]);
    }

    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4
Enter the cost adjacency matrix (999 for no edge):
0   4  999   6
4   0   1    999
999 1   0    2
6   999 2    0
Enter the source vertex: 1
The shortest distances from vertex 1 are:
1 --> 1 = 0
1 --> 2 = 4
1 --> 3 = 5
1 --> 4 = 6

Process returned 0 (0x0)   execution time : 8.163 s
Press any key to continue.
```

LAB PROGRAM 11:

Implement "N-Queens Problem" using Backtracking.

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <math.h>

int x[20], count = 1;

void queens(int, int);
int place(int, int);

int main() {
   int n;
   printf("Enter the number of queens: ");
   scanf("%d", &n);
   queens(1, n);
   return 0;
}

void queens(int k, int n) {
   int j, i;
   for (j = 1; j <= n; j++) {
      if (place(k, j)) {
         x[k] = j;
         if (k == n) {
            printf("\nSolution %d:\n", count++);
            for (i = 1; i <= n; i++) {
               printf("Row %d --> Column %d\n", i, x[i]);
            }
            printf("\n");
         } else {
```

```
            queens(k + 1, n);
        }
      }
    }
}

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

OUTPUT

```
Enter the number of queens: 4

Solution 1:
Row 1 --> Column 2
Row 2 --> Column 4
Row 3 --> Column 1
Row 4 --> Column 3


Solution 2:
Row 1 --> Column 3
Row 2 --> Column 1
Row 3 --> Column 4
Row 4 --> Column 2


Process returned 0 (0x0)   execution time : 14.515 s
Press any key to continue.
```

****