VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

CHAITHANYA SUDHAN (1BM23CS073)

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 **CHAITHANYA SUDHAN(1BM23CS073)**, 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.

Anusha S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

11	Implement “N-Queens Problem” using Backtracking.	29-30

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.

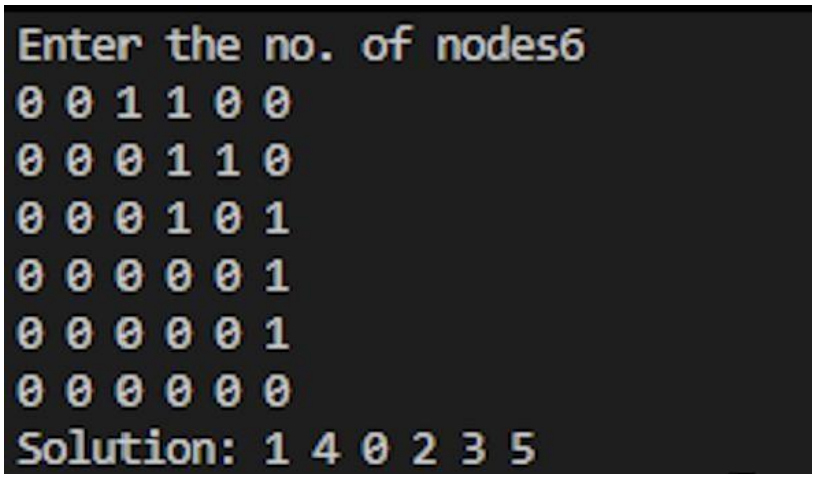
CODE-

DFS:

```
#include <stdio.h> int n, a[10][10],

res[10], s[10], top = 0;

void dfs(int, int, int[][10]);
void dfs_top(int, int[][10]);

int main() { printf("Enter the no.
   of nodes");
   scanf("%d", &n);
   int i, j;
   for (i = 0; i < n; i++) { for
      (j = 0; j < n; j++) {
         scanf("%d", &a[i][j]);
      }  }  dfs_top(n,  a);
   printf("Solution: "); for (i
   = n - 1; i >= 0; i--) {
   printf("%d ", res[i]);
   } return
   0;
}

void dfs_top(int n, int a[][10]) { int
   i;
   for (i = 0; i < n; i++) {
   s[i] = 0; }
   for (i = 0; i < n; i++) {
      if (s[i] == 0) { dfs(i,
      n, a);
      }
   }
}

void dfs(int j, int n, int a[][10]) {
   s[j] = 1; int i;
   for (i = 0; i < n; i++) { if (a[j][i]
      == 1 && s[i] == 0) { dfs(i, n,
      a);
```

```
    } }
  res[top++] =
  j;
}
```

OUTPUT-



Source removal method: #include

<stdio.h>

```
int a[10][10], n, t[10], indegree[10]; int
stack[10], top = -1;

void computeIndegree(int, int [][10]); void
tps_SourceRemoval(int, int [][10]);

int main() {
  printf("Enter the no. of nodes: ");
  scanf("%d", &n); int i, j; for (i =
  0; i < n; i++) {
    for (j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    } } computeIndegree(n,
  a); tps_SourceRemoval(n,
  a);  printf("Solution:  ");
  for (i = 0; i < n; i++) {
    printf("%d ", t[i]);
  }  return
  0;
}
void computeIndegree(int n, int a[][10]) {
```

```
    int i, j, sum = 0;
    for (i = 0; i < n; i++) {
        sum = 0; for (j = 0; j <
        n; j++) {
            sum = sum + a[j][i];
        }   indegree[i]   =
        sum;
    }
}

void tps_SourceRemoval(int n, int a[][10]) {
    int i, j, v; for (i = 0; i < n; i++) { if
    (indegree[i] == 0) {
            stack[++top] = i;
        }
    }

    int k = 0; while
    (top != -1) {
        v    =    stack[top--];
        t[k++] = v; for (i = 0;
        i < n; i++) {
            if (a[v][i] != 0) {
                indegree[i] = indegree[i] - 1; if
                (indegree[i] == 0) {
                    stack[++top] = i;
                }
            }
        }
    }
}
```

```
Enter the no. of nodes: 5
0 0 1 0 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0
Solution: 1 3 0 2 4
```

LeetCode Program related to Topological sorting Code

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


N* adjList[numCourses];


int visited[numCourses];


int recStack[numCourses];


for(int i=0;i<numCourses;i++){
    visited[i]=0;
    recStack[i]=0;
}
```

Lab Program 2:

Implement Johnson Trotter algorithm to generate permutations.

CODE-

```
#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]);
```

```
        }
    }
}

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-

```
Enter the number of elements: 4
Enter the elements: 1 2 3 4
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 3 2
1 4 2 3
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 3 1
2 4 1 3
3 2 1 4
3 2 4 1
3 1 2 4
3 1 4 2
3 4 1 2
3 4 2 1
4 2 3 1
4 2 1 3
4 3 2 1
4 3 1 2
4 1 3 2
4 1 2 3
```

Lab program 3:

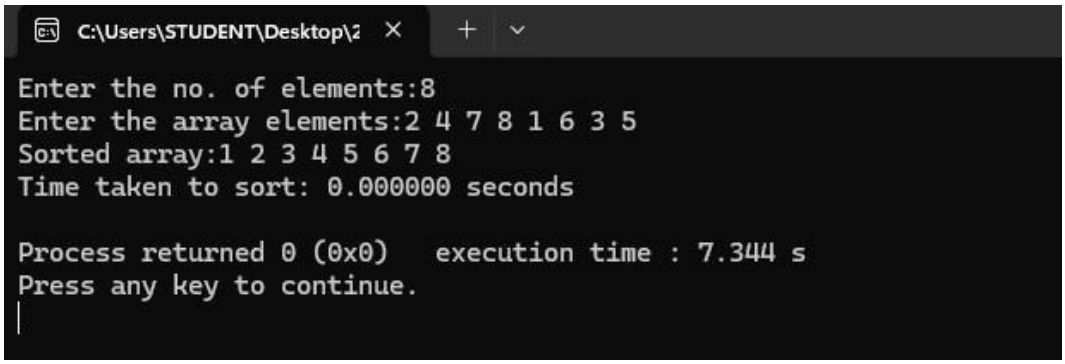
Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

CODE-

```
#include <stdio.h>
#include<time.h>
int a[20],n;
void simple_sort(int [],int,int,int);
void  merge_sort(int[],int,int);  int
main()
{  int  i;  clock_t  start,  end;  double
   time_taken; printf("Enter the no. 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);
        simple_sort(a,low,mid,high);
    } }
void simple_sort(int a[],int low, int mid, int high){
    int i=low,j=mid+1,k=low; int
    c[n];
    while(i<=mid          &&          j<=high){
        if(a[i]<a[j]){
            c[k++]=a[i];
            i++;
```

```
        }else{
            c[k++]=a[j];
            j++;
        }
    }
    while(i<=mid){
        c[k++]=a[i];
    i++; }
    while(j<=high)
    { c[k++]=a[j];
    j++; }
    for(i=low;i<=high;i++){
        a[i]=c[i];
    }
}
```

OUTPUT-



LeetCode Program related to sorting. Code

```
def countRangeSum(self, nums, lower, upper):
    first = [0]
    for num in nums:
        first.append(first[-1] + num)
    def sort(lo, hi):
        mid = (lo + hi) / 2
        if mid == lo:
            return 0
        count = sort(lo, mid) + sort(mid, hi)
i = j = mid
```

```
        for left in first[lo:mid]:
            while i < hi and first[i] - left <  lower: i += 1
              while j < hi and first[j] - left <= upper: j += 1
            count          +=          j          -          i
      first[lo:hi]             =              sorted(first[lo:hi])
    return                                                 count
return sort(0, len(first)
```

Lab program 4:

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

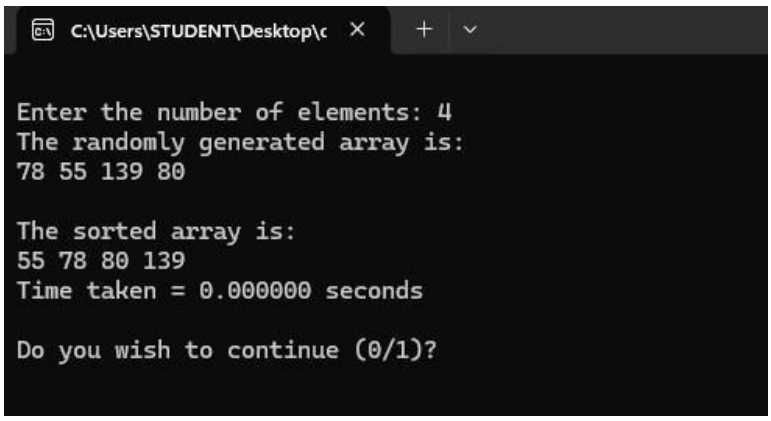
CODE-

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

    while (ch == 1) { 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;
       }

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

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

       printf("\n\nDo you wish to continue? (1 for Yes / 0 for No): ");
       if (scanf("%d", &ch) != 1) { printf("Invalid input! Exiting
       program.\n"); break;
       }

       if (ch != 1) break;
    }

    return 0;
}

void quicksort(int a[], int low, int high) {
   int mid; if (low < high) { 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, i, j, temp, k;
    key = a[low]; i =
    low + 1; j = high;

    while (i <= j) { while (i <= high &&
       key >= a[i])
          i = i + 1;
       while (key < a[j])
          j = j - 1;

       if (i < j) {
          temp =
          a[i]; a[i] =
          a[j]; a[j] =
          temp;
       } else { k = a[j];
          a[j] = a[low];
          a[low] = k;
       } }
    return
    j;
}
```

OUTPUT-



LeetCode Program related to sorting. Code

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        maxHeap = nums
        for i in range(len(maxHeap)):
            maxHeap[i] = -maxHeap[i]
        heapify(maxHeap)
```

```
        for i in range(k-1):
            heappop(maxHeap)
    return -maxHeap[0]
```

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
<conio.h>
#include <time.h>

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)
     { if(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);
   } }

void main()
{ int i, n, a[20], ch = 1;
   clock_t   start,   end;
   while(ch)
   { printf("enter the number of elements to sort\n");
      scanf("%d",  &n);  printf("enter  the
      elements to sort\n"); for(i = 1; i <= n;
      i++)
         scanf("%d", &a[i]);
      start = clock();
      heapsort(a,  n);
      end = clock();
      printf("the sorted list of elements is\n"); for(i
      = 1; i <= n; i++)
         printf("%d\n", a[i]);
      printf("Time taken is %lf CPU cycles\n", (end - start) / CLK_TCK);
      printf("do u wish to run again (0/1)\n");
      scanf("%d", &ch);
   }
}
```

OUTPUT-

```
enter the number of elements to sort
6
enter the elements to sort
5 9 3 7 4 6
the sorted list of elements is
3
4
5
6
7
9
Time taken is 0.000000 CPU cycles
do u wish to run again (0/1)
0
```

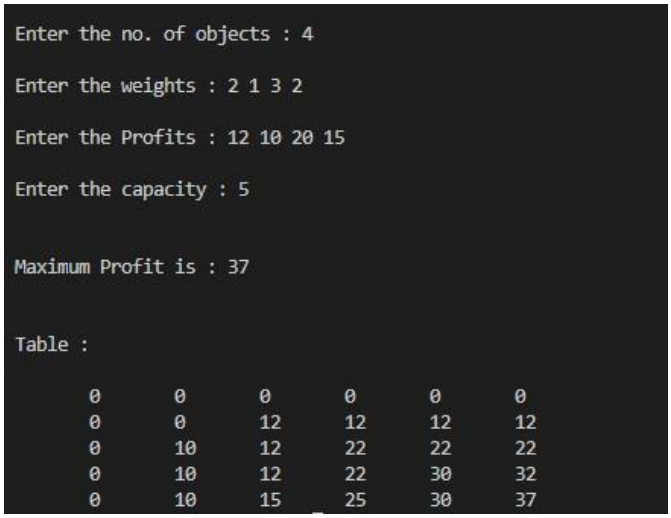
Lab program 6:

Implement 0/1 Knapsack problem using dynamic programming.

CODE-

```
#include <stdio.h>

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

void knapsack(int n, int w[10], int p[10], int c) {
   int max(int, int); 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\n Maximum Profit is : %d ", v[n][c]);
   printf("\n\n\n Table : \n\n"); for (i = 0; i <= n;
   i++) { for (j = 0; j <= c; j++) {
         printf("\t%d", v[i][j]);
      }
      printf("\n");
   }
}

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

void main() {
   printf("\n Enter the no. of objects : ");
   scanf("%d", &n); printf("\n Enter the
   weights : "); for (i = 1; i <= n; i++) {
      scanf("%d", &w[i]);
   } printf("\n Enter the Profits :
   "); for (i = 1; i <= n; i++) {
      scanf("%d", &p[i]);
   }
   printf("\n Enter the capacity : ");
   scanf("%d",  &c);  knapsack(n,
   w, p, c);
}
```

OUTPUT-



LeetCode Program related to Knapsack problem or Dynamic Programming. Code

```
class Solution:
    def maxSizeSlices(self, slices: List[int]) -> int:
        n = len(slices) // 3
        def linear(arr):
            eat = [[0] + [-math.inf]*n] * 2
            for x in arr:
                eat.append([i and max(eat[-1][i], eat[-2][i-1]+x) for i in range(n+1)])
            return max(l[n] for l in eat)
        return max(linear(slices[1:]), linear(slices[:-1]))
```

Lab Program 7:

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

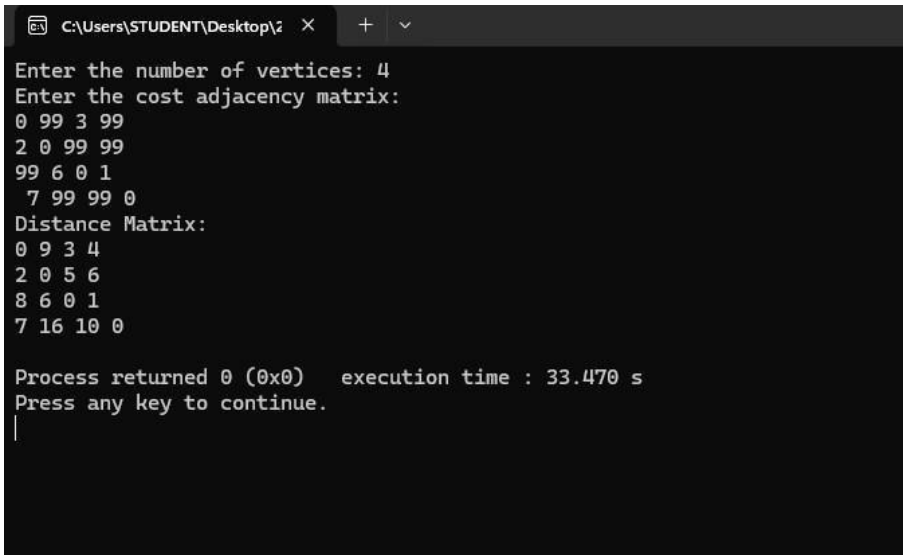
CODE-

```
#include <stdio.h> int
a[10][10], D[10][10], n;
void floyd(int [][10], int);
```

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

    printf("Distance      Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%d ", D[i][j]);
        }
        printf("\n");
    } return
0; }
void floyd(int a[][10], int n) {
    int i, j, k; for (i = 0; i < n;
    i++) { for (j = 0; j < n; j++)
    { D[i][j] = a[i][j];
        }
    }
```

```
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
        D[i][j] = min(D[i][j],
        D[i][k] + D[k][j]);
            }
        }
    } }
int min(int a, int b) {
    return (a < b) ? a : b;
}
```

OUTPUT-



LeetCode Program related to shortest distance calculation. Code

```
import heapq

from collections import defaultdict
import sys

def countPaths(n, roads):

    mod = 10**9 + 7

    adj = defaultdict(list)
```

```
    for u, v, t in roads:

      adj[u].append((v, t))

     adj[v].append((u, t))

shortesttime = [sys.maxsize] * n
    cnt = [0] * n
    pq = [(0, 0)]  # (time, node)

 shortesttime[0] = 0

  cnt[0] = 1

 while pq:

   time, node = heapq.heappop(pq)

    if time > shortesttime[node]:
       continue


    for nbr, rtime in adj[node]:
           if time + rtime < shortesttime[nbr]:

              shortesttime[nbr] = time + rtime

        cnt[nbr] = cnt[node]

            heapq.heappush(pq,(shortesttime[nbr], nbr))

          elif time + rtime == shortesttime[nbr]:

              cnt[nbr] = (cnt[nbr] + cnt[node]) % mod

return cnt[-1]
```

Lab program 8:

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

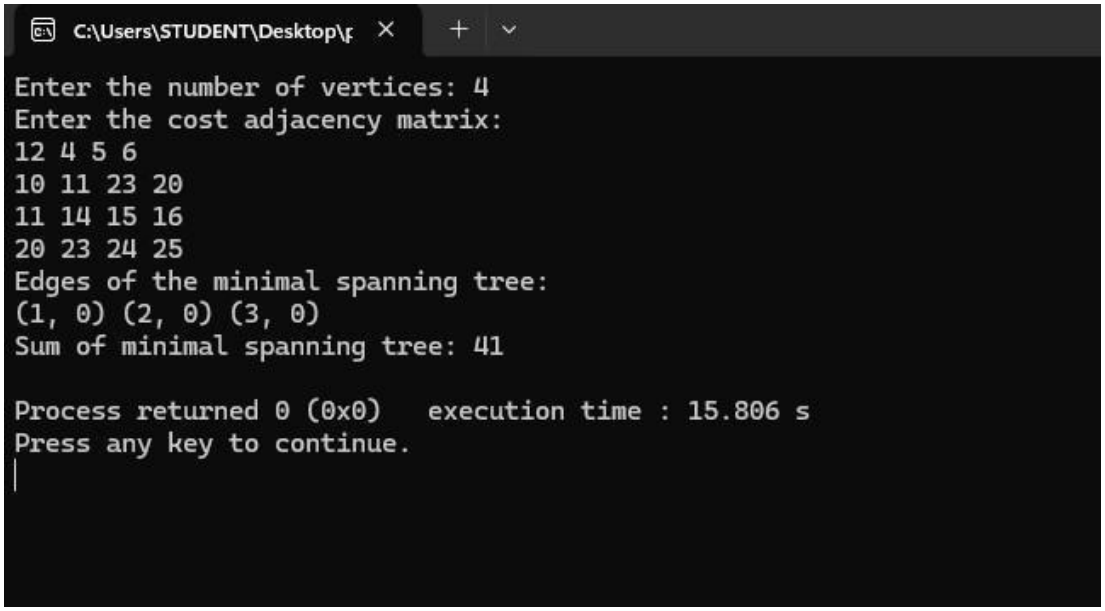
CODE-

```
#include <stdio.h>
int cost[10][10], n, t[10][2], sum;
void prims(int cost[10][10], int n);
int main() { int i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

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

    prims(cost, n);

    printf("Edges of the minimal spanning tree:\n"); for
    (i = 0; i < n - 1; i++) {
        printf("(%d, %d) ", t[i][0], t[i][1]);
    }

    printf("\nSum of minimal spanning tree: %d\n", sum);

   return 0; }
void prims(int cost[10][10], int n) {
   int i, j, u, v; int min, source;
```

```
int p[10], d[10], s[10];


min = 999;
source = 0;


for (i = 0; i < n; i++) {
   d[i]                    =
cost[source][i]; s[i] = 0;
p[i]    =    source;    }
s[source] = 1;
sum = 0;


int k = 0;


for (i = 0; i < n - 1; i++) {
   min = 999;
   u = -1;


   for (j = 0; j < n; j++) {
      if (s[j] == 0 && d[j] < min) {
         min = d[j]; u
         = j;
      }
   }


   if (u != -1) {
      t[k][0] = u; t[k][1] =
      p[u];  k++;  sum  +=
      cost[u][p[u]];  s[u]  =
      1;
```

```
        for (v = 0; v < n; v++) {
            if (s[v] == 0 && cost[u][v] < d[v]) {
                d[v] = cost[u][v]; p[v]
                = u;
            }
        }
    }
  }
}
```

OUTPUT-



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

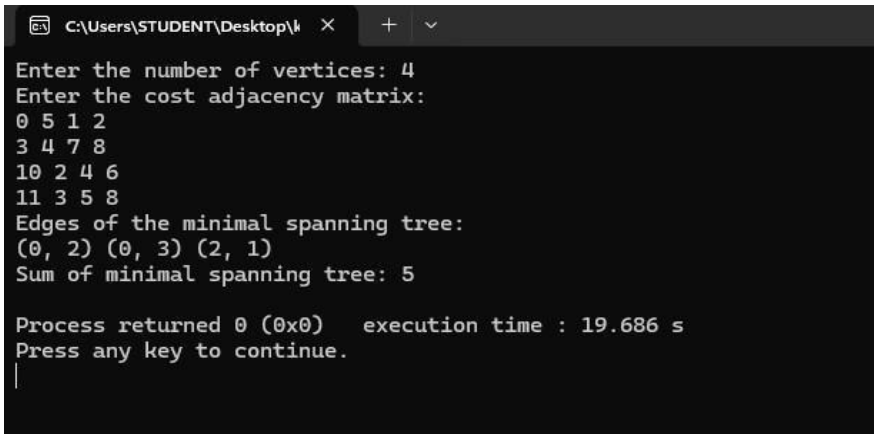
CODE-

```
#include <stdio.h>

int cost[10][10], n, t[10][2], sum;

void kruskal(int cost[10][10], int n);
int find(int parent[10], int i);
```

```
int main() { int
   i, j;

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

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

   kruskal(cost, n);

   printf("Edges of the minimal spanning tree:\n");
   for (i = 0; i < n - 1; i++) {
      printf("(%d, %d) ", t[i][0], t[i][1]);
   }

   printf("\nSum of minimal spanning tree: %d\n", sum);

   return 0;
}

void kruskal(int cost[10][10], int n) {
   int min, u, v, count, k;
   int parent[10];

   k = 0; sum
   = 0;

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

   count = 0;

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

      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (find(parent, i) != find(parent, j) && cost[i][j] < min) {
               min = cost[i][j];
```

```
                u = i; v
                = j;
              }
          } } int root_u =
      find(parent, u); int root_v =
      find(parent, v);

      if (root_u != root_v) {
         parent[root_u]          =
         root_v;   t[k][0]   =   u;
         t[k][1] = v; sum += min;
         k++;
         count++;
      }
   }
}

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

OUTPUT-



Lab program 9:

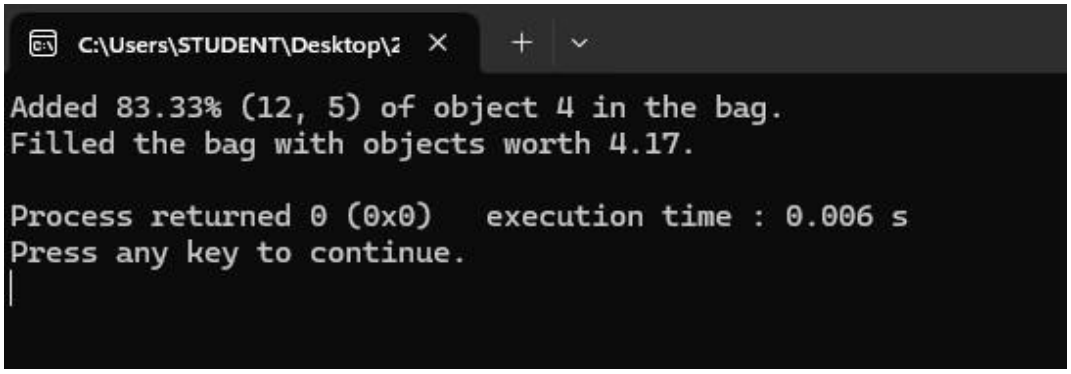
Implement Fractional Knapsack using Greedy technique.

CODE-

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

```
int n = 5;

int p[10] = {3, 3, 2, 5, 1}; int
w[10] = {10, 15, 10, 12, 8}; int
W = 10;

int main() { int
   cur_w; float tot_v
   = 0.0; int i, maxi;
   int used[10] = {0};
   cur_w = W;

   while (cur_w > 0) { maxi = -1; for (i = 0; i < n; ++i) { if (used[i] == 0 && (maxi == -
      1 || (float)p[i] / w[i] > (float)p[maxi] / w[maxi])) { maxi = i;
         }
      }

      used[maxi] = 1;
      if (w[maxi] <= cur_w) {

         cur_w -= w[maxi]; tot_v
         += p[maxi];
         printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi +
         1, w[maxi], p[maxi], cur_w);
      } else {

         tot_v += (float)p[maxi] * cur_w / w[maxi];
         printf("Added %.2f%% (%d, %d) of object %d in the bag.\n", (float)(cur_w * 100) /
         w[maxi], w[maxi], p[maxi], maxi + 1); cur_w = 0;
      }
   }

   printf("Filled the bag with objects worth %.2f.\n", tot_v); return
   0;
}
```

OUTPUT-



LeetCode Program related to Greedy Technique algorithms.

Code

```
def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        boxTypes.sort(key=lambda   a:-a[1])
        max_units = 0 for box in
        boxTypes:
           if truckSize < 0 :
          break  max_units+= min(truckSize,
        return max_units
```

Lab program 10:

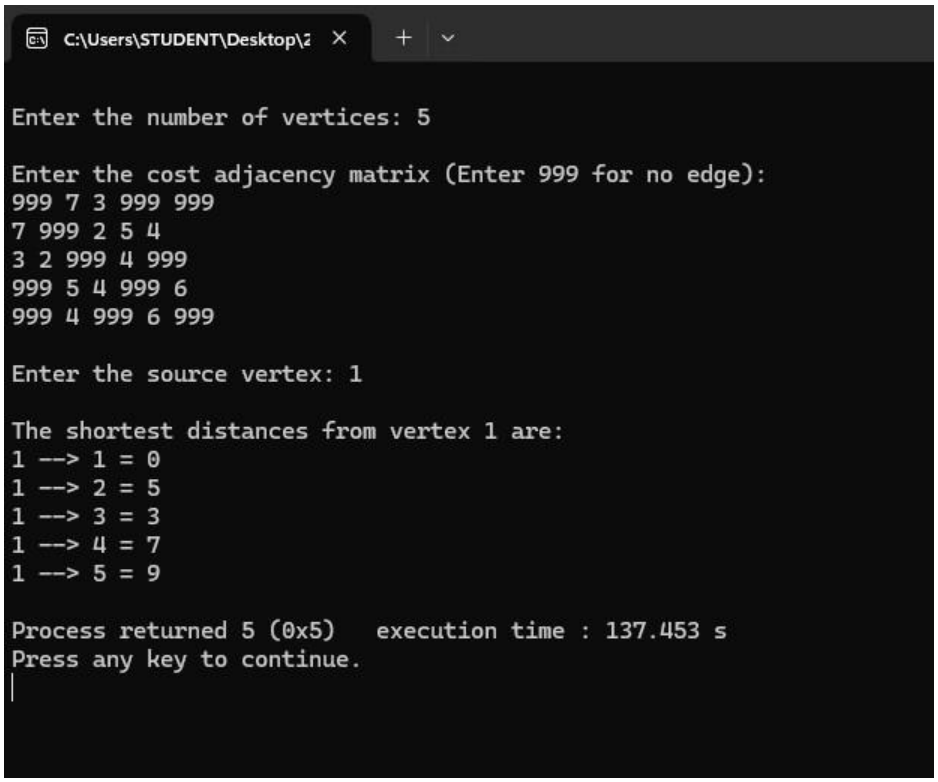
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

CODE-

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

INF 999

void main() { int i, j, n, v, k, min, u, c[20][20],
   s[20], d[20]; printf("\nEnter the number of
   vertices: "); scanf("%d", &n);

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

   printf("\nEnter 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) {
             if(d[i] > (d[u] + c[u][i])) {
                d[i] = d[u] + c[u][i];
             }
          }
       }
    }


    printf("\nThe shortest distances from vertex %d are:\n", v); for(i
    = 1; i <= n; i++) {
       if(d[i] == INF) {
          printf("%d --> %d = No Path\n", v, i);
       } else { printf("%d --> %d = %d\n", v, i,
       d[i]); }
    }
}
```

OUTPUT-



Lab program 11:

Implement “N-Queens Problem” using Backtracking.

CODE-

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

int x[20], count = 1;

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

void main()
{ int n, k = 1;
   printf("\n enter the number of queens to be placed\n");
    scanf("%d", &n); queens(k,
    n);
}
void queens(int k, int n)
{
    int i, j;
    for(j = 1; j <= n; j++)
    { if(place(k, j))
```

```
      { x[k] = j;
        if(k ==
        n)
        { printf("\n %d solution", count);
          count++; for(i = 1; i
          <= n; i++)
            printf("\n \t %d row <--- %d column", i, x[i]);
          getch();
        } 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 to be placed
4

 1 solution
         1 row <--- 2 column
         2 row <--- 4 column
         3 row <--- 1 column
         4 row <--- 3 column
 2 solution
         1 row <--- 3 column
         2 row <--- 1 column
         3 row <--- 4 column
         4 row <--- 2 column
```