

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Hemanshu Dhiman(1BM23CS109)

**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 Hemanshu Dhiman (**1BM23CS109**), 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.

Sowmya T
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Write program to obtain the Topological ordering of vertices in a given digraph. LeetCode Program related to Topological sorting	5
2	Implement Johnson Trotter algorithm to generate permutations.	11
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. LeetCode Program related to sorting.	13
4	Sort a given set of N integer elements using Quick Sort technique and compute its time taken. LeetCode Program related to sorting.	17
5	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	22
6	Implement 0/1 Knapsack problem using dynamic programming. LeetCode Program related to Knapsack problem or Dynamic Programming.	26
7	Implement All Pair Shortest paths problem using Floyd's algorithm. LeetCode Program related to shortest distance calculation.	31
8	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	36
9	Implement Fractional Knapsack using Greedy technique. LeetCode Program related to Greedy Technique algorithms.	42
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	46
11	Implement "N-Queens Problem" using Backtracking.	49

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.

i)using dfs

CODE:

```
#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;
}

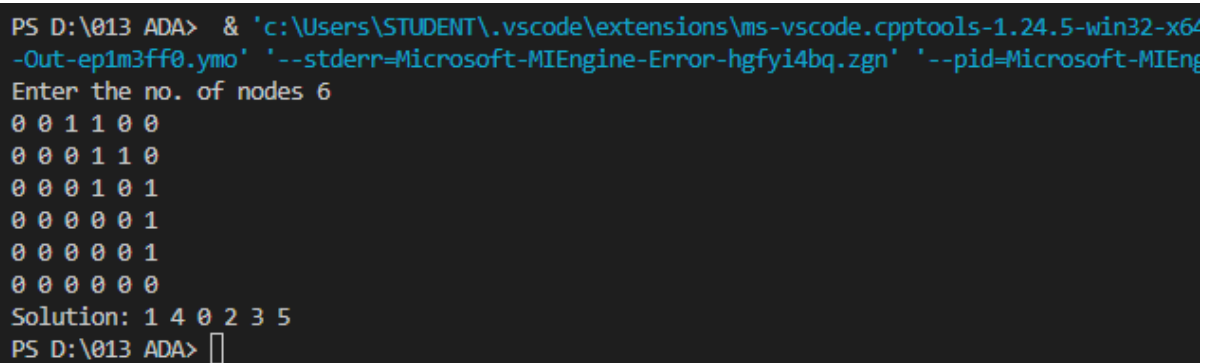
void dfs_top(int n, int a[][10]) {
    int i;
    for (i = 0; i < n; i++) {
        s[i] = 0;
    }
    for (i = 0; i < n; i++) {
        if (s[i] == 0) {
```

```

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:



```

PS D:\013 ADA> & 'c:\Users\STUDENT\.vscode\extensions\ms-vscode.cpptools-1.24.5-win32-x64-Out-ep1m3ff0.ymo' --stderr=Microsoft-MIEngine-Error-hgfyi4bq.zgn' --pid=Microsoft-MIEng
Enter the no. of nodes 6
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
Solution: 1 4 0 2 3 5
PS D:\013 ADA>

```

ii) using source removal method

CODE:

```

#include<stdio.h>
#define max 100
int adj[max][max];
int indeg[max];
int topo[max];
int vertices;
void init(int v){

```

```

vertices = v;
for(int i=0;i<v;i++){
    indeg[i] = 0;
    for(int j=0;j<v;j++){
        adj[i][j] = 0;
    }
}
}

void topologicalsort(){
    int q[max];
    int front = 0,rear = 0;
    int index = 0;

    for(int i=0;i<vertices;i++){
        if(indeg[i] == 0){
            q[rear++] = i;
        }
    }

    while(front != rear){
        int u = q[front++];
        topo[index++] = u;
        for(int i=0;i<vertices;i++){
            if(adj[u][i] == 1){
                indeg[i]--;
                if(indeg[i]==0) q[rear++] = i;
            }
        }
    }

    if(index != vertices){
        printf("Topological sort not possible\n");
        return;
    }
}

```


```

printf("Topological order:");
for(int i=0;i<vertices;i++){
    printf("%d ",topo[i]);
}
printf("\n");
}

int main(){
    int v,u,w,e;
    printf("enter no of vertices:");
    scanf("%d",&v);
    init(v);
    printf("enter adjacency matrix:\n");
    for(int i=0;i<v;i++){
        for(int j=0;j<v;j++){
            scanf("%d",&adj[i][j]);
            if(adj[i][j]){
                indeg[j]++;
            }
        }
    }
    topologicalsort();
    return 0;
}

```

OUTPUT:



```

} ; if ($?) { .\tp }
enter no of vertices:5
enter adjacency matrix:
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
Topological order:1 0 3 2 4

```


LeetCode Program related to Topological sorting

CODE:

```
bool dfs(int course, int** prerequisites, int prerequisitesSize, int* prerequisitesColSize, int*
visited, int
numCourses) {
    if (visited[course] == 1) {
        return false;
    }
    if (visited[course] == 2) {
        return true;
    }
    visited[course] = 1;
    for (int i = 0; i < prerequisitesSize; i++) {
        if (prerequisites[i][0] == course) {
            int nextCourse = prerequisites[i][1];
            if (!dfs(nextCourse, prerequisites, prerequisitesSize, prerequisitesColSize, visited,
numCourses)) {
                return false;
            }
        }
    }
    visited[course] = 2;
    return true;
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize) {
    int visited[numCourses];
    for (int i = 0; i < numCourses; i++) {
        visited[i] = 0;
    }
    for (int i = 0; i < prerequisitesSize; i++) {
```

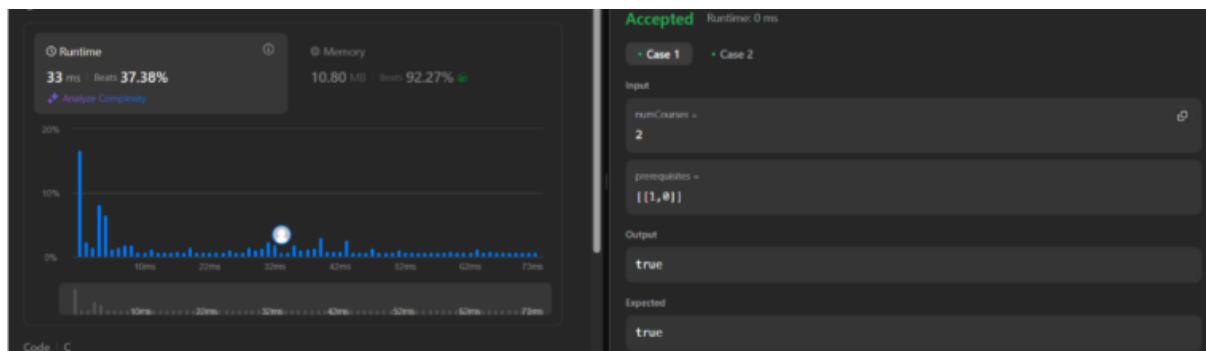
```

if (visited[i] == 0) {
    if (!dfs(i, prerequisites, prerequisitesSize, prerequisitesColSize, visited, numCourses)) {
        return false;
    }
}
}

return true;
}

```

OUTPUT:



Lab program 2:

Implement Johnson Trotter algorithm to generate permutations.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

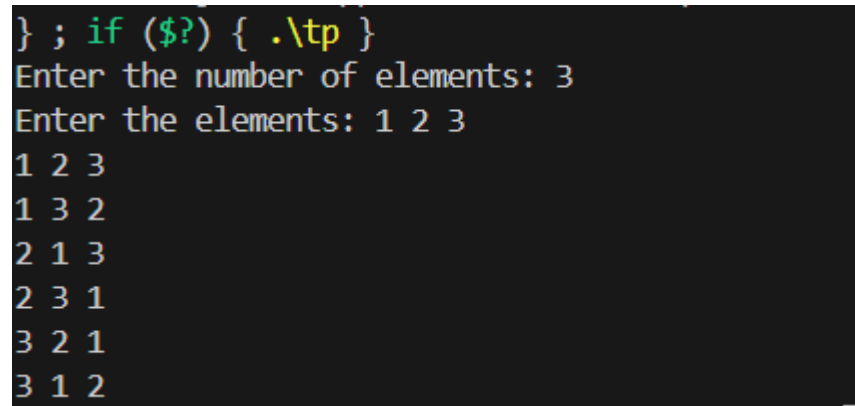
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

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]); // 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:



```
} ; if ($?) { .\tp }  
Enter the number of elements: 3  
Enter the elements: 1 2 3  
1 2 3  
1 3 2  
2 1 3  
2 3 1  
3 2 1  
3 1 2
```

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 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];
}
}
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;
}

```

OUTPUT:

```

Enter the no. of elements:12
Enter the array elements:12
67
33
2
1
88
4
16
30
29
5
9
Sorted array:1 2 4 5 9 12 16 29 30 33 67 88
Time taken to sort: 0.000000 seconds

```

LeetCode problem related to Merge Sort

CODE:

```

class Solution {
public:
    int countRangeSum(vector<int>& nums, int lower, int upper) {
        multiset <long long> mpp;
        mpp.insert(0);
        long long sum=0,ans=0;
        for(long long i=0;i<nums.size();i++)
        {
            long long low,high;
            sum+=nums[i];
            low=sum-upper;
            high=sum-lower;
            auto lb=mpp.lower_bound(low);

```

```

        auto ub=mpp.upper_bound(high);

        ans+= distance(lb,ub);

        mpp.insert(sum);

    }

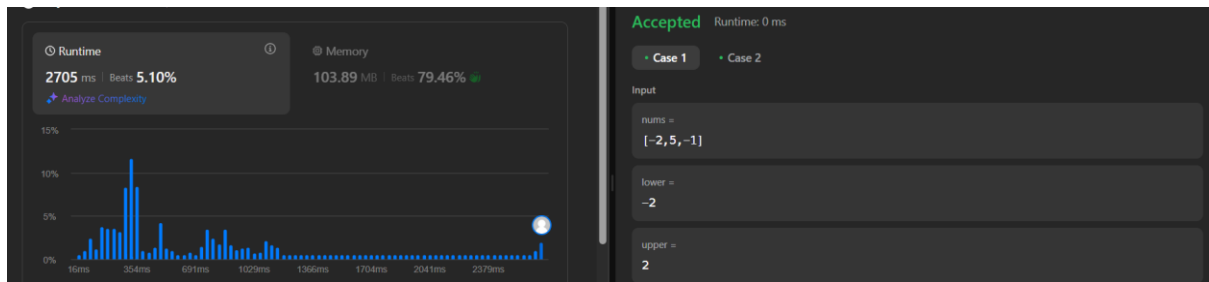
    return ans;

}

};

```

Output:



Lab program 4:

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

CODE:

```
#include <stdio.h>

#include <stdlib.h> // for rand()
#include <time.h>   // for clock()


#define MAX 5000


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


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


    while (1) {
        printf("\nEnter the number of elements: ");
        scanf("%d", &n);


        // Generate random array elements
        for (i = 0; i < n; i++) {
            a[i] = rand() % 200; // Random number between 0 and 199
        }


        // Display the random array
        printf("The random generated array is:\n");
        for (i = 0; i < n; i++) {
            printf("%d ", a[i]);
        }
        printf("\n");
    }
}
```

```

// Measure the time taken for sorting
start = clock();
quicksort(a, 0, n - 1);
end = clock();

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

// Calculate and print the time taken for sorting
printf("Time taken = %f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

// Ask user if they want to continue
printf("\nDo you wish to continue? (0/1): ");
scanf("%d", &ch);
if (ch == 0) {
    break;
}
}

return 0;
}

// QuickSort function
void quicksort(int a[], int low, int high) {
    if (low < high) {
        int mid = partition(a, low, high);
        quicksort(a, low, mid - 1); // Recursively sort the left part
        quicksort(a, mid + 1, high); // Recursively sort the right part
    }
}

```

```

    }
}

// Partition function: Returns the partition index
int partition(int a[], int low, int high) {
    int pivot = a[low]; // Pivot is the first element in the array
    int i = low + 1;
    int j = high;
    int temp;

    while (i <= j) {
        // Find an element greater than the pivot
        while (i <= high && a[i] <= pivot) {
            i++;
        }

        // Find an element less than the pivot
        while (a[j] > pivot) {
            j--;
        }

        // If there are elements to swap, swap them
        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    // Swap the pivot element with a[j]
    temp = a[low];
    a[low] = a[j];

```

```
a[j] = temp;
```

```
return j; // Return the partition index
```

```
}
```

OUTPUT:

```
Enter the number of elements: 6
The random generated array is:
41 67 134 100 169 124

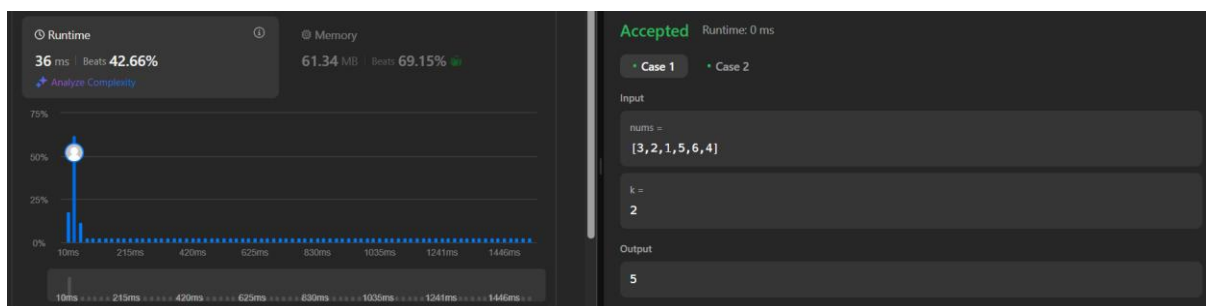
The sorted array elements are:
41 67 100 124 134 169
Time taken = 0.000000 seconds
Do you wish to continue? (0/1): 0
```

LeetCode question related to Quick Sort

CODE:

```
class Solution {  
public:  
    int findKthLargest(vector<int>& nums, int k) {  
        priority_queue<int, vector<int>, greater<int>> pq;  
        for(int num:nums){  
            pq.push(num);  
            if(pq.size() > k){  
                pq.pop();  
            }  
        }  
        return pq.top();  
    }  
};
```

Output:



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("\n enter the number of elements to sort\n");
        scanf("%d",&n);
        printf("\n enter the elements to sort\n");
        for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
        start=clock();
        heapsort(a,n);
        end=clock();
        printf("\n the sorted list of elemnts is\n");
        for(i=1;i<=n;i++)
            printf("%d\n",a[i]);
        printf("\n 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
5

enter the elements to sort
8 5 6 3 1

the sorted list of elemnts is
1
3
5
6
8

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 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:

```
Enter the no. of objects : 4
Enter the weights : 2
1
3
2
Enter the Profits : 12
10
20
15
Enter the capacity : 5
Maximum Profit is : 37
Table :
```

0	0	0	0	0	0
0	0	12	12	12	12
0	10	12	22	22	22
0	10	12	22	30	32
0	10	15	25	30	37

LeetCode Program related to Knapsack problem or Dynamic Programming.

CODE:

```
class Solution {
private:
    int solveMem(int sindex,int eindex,vector<int>& slices,
    int n,vector<vector<int>> &dp)
    {
        if(n==0 || sindex>eindex) return 0;

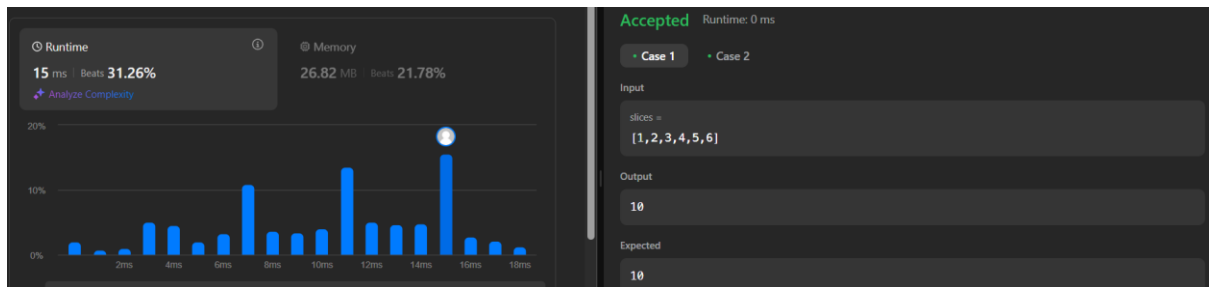
        if(dp[sindex][n]!=-1) return dp[sindex][n];

        int take=slices[sindex]+solveMem(sindex+2,eindex,slices,n-1,dp);
        int nottake=0+solveMem(sindex+1,eindex,slices,n,dp);

        return dp[sindex][n]=max(take,nottake);
    }

public:
    int maxSizeSlices(vector<int>& slices) {
        int k=slices.size();
        vector<vector<int>> dp1(k,vector<int> (k,-1));
        int case1=solveMem(0,k-2,slices,k/3,dp1);
        vector<vector<int>> dp2(k,vector<int> (k,-1));
        int case2=solveMem(1,k-1,slices,k/3,dp2);
        return max(case1,case2);
    }
};
```

OUTPUT:



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 no. 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){
    if(a<b){
        return a;
    }else{
        return b;
    }
}

```


OUTPUT:

```
Enter the no. of vertices:4
Enter the cost adjacency matrix:
0
99
3
99
2
0
99
99
99
6
0
1
7
99
99
0
Distance Matrix:
0 9 3 4
2 0 5 6
8 6 0 1
7 16 10 0
```

LeetCode Program related to shortest distance calculation.

CODE:

```
class Solution {
public:
    int countPaths(int n, vector<vector<int>>& roads) {
        vector<vector<pair<int, int>>> graph(n);
        for (const auto& road : roads) {
            int u = road[0], v = road[1], time = road[2];
            graph[u].emplace_back(v, time);
            graph[v].emplace_back(u, time); //create adj
        }
        vector<long long> dist(n, LLONG_MAX); //dist
        vector<int> ways(n, 0); //ways to reach

        dist[0] = 0;
        ways[0] = 1;

        priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<>> pq; //pq
        for dijkstra
        pq.emplace(0, 0); //source

        const int MOD = 1e9 + 7;
        while(!pq.empty()){
            auto [d,node] = pq.top(); //topnode
            pq.pop();
            if(d > dist[node]) continue; //dist[node] already less

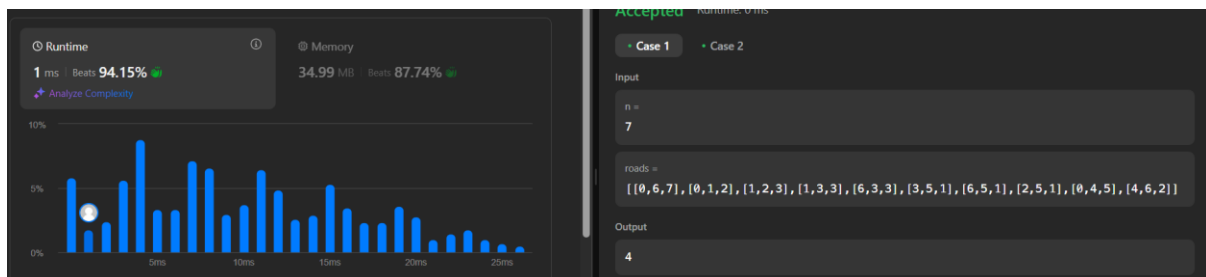
            for(auto& [neighbour,time] : graph[node]){
                if(dist[node] + time < dist[neighbour]){ //dijkstra condition
                    dist[neighbour] = dist[node]+time;
                    ways[neighbour] = ways[node]; //ways same as neighbour
                }
            }
        }
    }
};
```

```

        pq.emplace(dist[neighbour],neighbour);
    }
    else if(dist[node]+time == dist[neighbour]){
        ways[neighbour] = (ways[neighbour]+ways[node]) % MOD; //same dist, so one
        more or few more ways to reach
    }
}
}
}
return ways[n-1];
}
};

```

OUTPUT:



Lab program 8:

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:

```
Enter the number of vertices: 5
Enter the cost adjacency matrix:
```



```
Edges of the minimal spanning tree:
(1, 0) (2, 0) (3, 0) (4, 0)
Sum of minimal spanning tree: 10
```

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:

```

Enter the number of vertices: 5
Enter the cost adjacency matrix:
0
1
2
3
4
1
0
3
5
7
2
3
0
3
9
3
5
3
0
7
4
7
9
7
0
Edges of the minimal spanning tree:
(0, 1) (0, 2) (0, 3) (0, 4)
Sum of minimal spanning tree: 10

```

Lab program 9:

Implement Fractional Knapsack using Greedy technique.

CODE:

```
#include <stdio.h>

#define MAX 100

void fractionalKnapsack(int n, float weight[], float profit[], float capacity) {

    float ratio[MAX],

    temp; int i, j;

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

        ratio[i] = profit[i] / weight[i];

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

    float totalProfit = 0;
    for (i = 0; i < n; i++) {

        if (capacity >= weight[i]) {
            capacity -= weight[i];
            totalProfit += profit[i];
        } else {
```

```

totalProfit += ratio[i] * capacity;
break;
}
}

printf("Total Profit = %.2f\n", totalProfit);

}

int main() {
    int n;

    float weight[MAX], profit[MAX], capacity;

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

    printf("Enter the weights of the items: ");
    for (int i = 0; i < n; i++) {
        scanf("%f", &weight[i]);
    }

    printf("Enter the profits of the items: ");
    for (int i = 0; i < n; i++) {
        scanf("%f", &profit[i]);
    }

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

    fractionalKnapsack(n, weight, profit, capacity);
}

```

```
    return 0;  
}
```

OTUPUT:

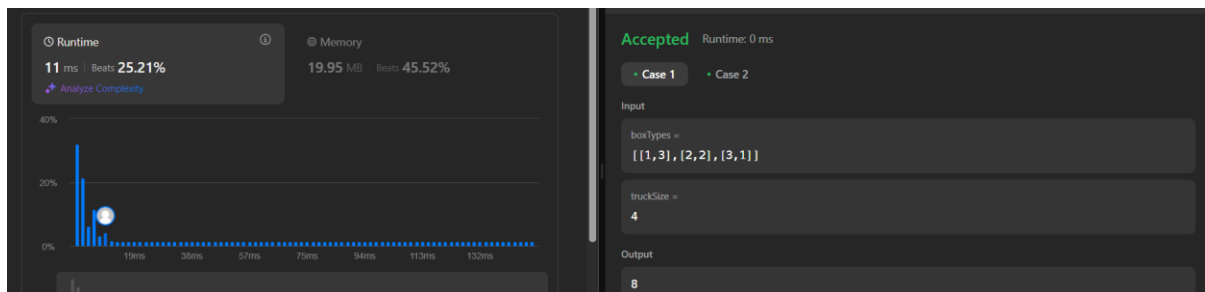
```
Enter the number of items: 7  
Enter the weights of the items: 2 1 3 4 7 3 1  
Enter the profits of the items: 3 4 6 8 3 7 2  
Enter the capacity of the knapsack: 17  
Total Profit = 31.29
```

LeetCode Program related to Greedy Technique algorithms.

CODE:

```
class Solution {
public:
    int maximumUnits(vector<vector<int>>& B, int T) {
        sort(B.begin(), B.end(), [](auto& a, auto& b) { return b[1] < a[1];});
        int ans = 0;
        for (auto& b : B) {
            int count = min(b[0], T);
            ans += count * b[1], T -= count;
            if (!T) return ans;
        }
        return ans;
    }
};
```

OUTPUT:



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>

void main()
{
    int i,j,n,v,k,min,u,c[20][20],s[20],d[20];
    printf("\n Enter the no. of vertices : ");
    scanf("%d",&n);
    printf("\n Enter the cost adjacency matrix : ");
    printf("\n Enter 999 for no edge ");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    printf("\n 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=999;
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("\n The shortest distance from %d is ",v);
for(i=1;i<=n;i++)
{
printf("\n %d --> %d = %d ",v,i,d[i]);
}
}

```

OUTPUT:

```
Enter the no. of vertices : 5  
Enter the cost adjacency matrix :  
Enter 999 for no edge 999
```

```
7  
3  
999  
999  
7  
999  
2  
5  
4  
3  
2  
999  
4  
999  
999  
5  
4  
999  
6  
999  
4  
999  
6  
999
```

```
Enter the source vertex : 1
```

```
The shortest distance from 1 is  
1 --> 1 = 0  
1 --> 2 = 5  
1 --> 3 = 3  
1 --> 4 = 7  
1 --> 5 = 9
```


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

```