**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**on**

Analysis and Design of Algorithms

***Submitted by***

**AKSHAY RAJ ARYAN (1BM22CS032)**

***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**

**April-2024 to August-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated to Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **AKSHAY RAJ ARYAN (1BM22CS032),** 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 academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

Vikranth B.M Dr. Jyothi S Nayak

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **Program Details** | **Page No.** |
| 1 | Leetcode exercises. | 1-3 |
| 2 | * Write program to obtain the Topological ordering of vertices in a given digraph. * Leet Code. | 4-9 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 10-13 |
| 4 | 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. | 14-16 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 17-19 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 20-22 |
| 7 | * Implement 0/1 Knapsack problem using dynamic programming. * Leet Code. | 23-28 |
| 8 | * Implement All Pair Shortest paths problem using Floyd’s algorithm. * Leet Code. | 29-33 |
| 9 | * 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. | 34-39 |

|  |  |  |
| --- | --- | --- |
| 10 | Implement Fractional Knapsack using Greedy technique. | 40-42 |
| 11 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 43-45 |

**Course Outcome**

|  |  |
| --- | --- |
| 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. |

Q1) Leetcode exercises.

int maximumScore(int\* nums, int numsSize, int k) { int n=numsSize;

int left=k; int right=k;

int ans=nums[k];

int currMin=nums[k]; while(left>0 || right<n-1){

if ((left > 0 ? nums[left - 1]: 0) < (right < n - 1 ? nums[right + 1] : 0)){ right++;

currMin=fmin(currMin, nums[right]);

}

else{

left--;

currMin=fmin(currMin, nums[left]);

}

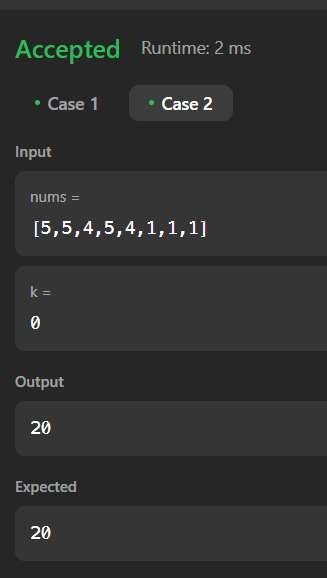
ans = fmax(ans, currMin \* (right - left + 1));

}

return ans;

}

OP:



Q1b) Leetcode exercises.

bool canMeasureWater(int jug1, int jug2, int t) { if (jug1 + jug2 < t)

return false; while (jug2 != 0) {

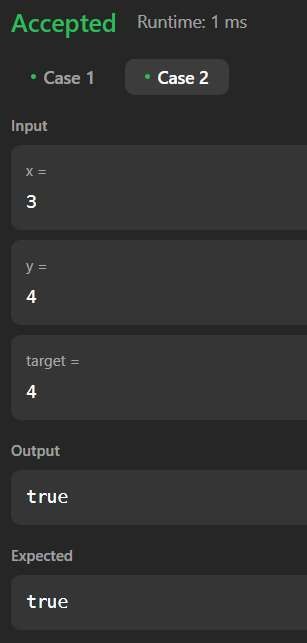
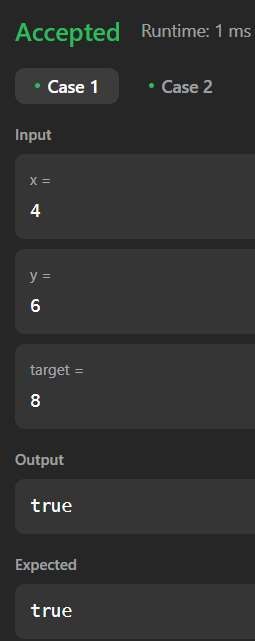
int temp = jug2; jug2 = jug1 % jug2; jug1 = temp;

}

return t % jug1 == 0;

}

OP:



Q2) Write program to obtain the Topological ordering of vertices in a given digraph. #include <stdio.h>

#include <stdlib.h> #define MAX 100

void topologicalSort(int n, int adj[][MAX]) { int inDegree[MAX] = {0};

int stack[MAX], top = -1;

int topOrder[MAX], orderIndex = 0; for (int i = 0; i < n; i++) {

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

inDegree[j]++;

}

}

}

for (int i = 0; i < n; i++) { if (inDegree[i] == 0) {

stack[++top] = i;

}

}

while (top != -1) {

int u = stack[top--]; topOrder[orderIndex++] = u;

for (int i = 0; i < n; i++) { if (adj[u][i] == 1) {

inDegree[i]--;

if (inDegree[i] == 0) { stack[++top] = i;

}

}

}

}

if (orderIndex != n) {

printf("The graph has a cycle, topological sort not possible.\n");

} else {

printf("Topological order: "); for (int i = 0; i < n; i++) {

printf("%d ", topOrder[i]);

}

printf("\n");

}

}

int main() { int n, e;

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

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

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

printf("Enter the edges (u v) format:\n"); for (int i = 0; i < e; i++) {

int u, v;

scanf("%d %d", &u, &v);

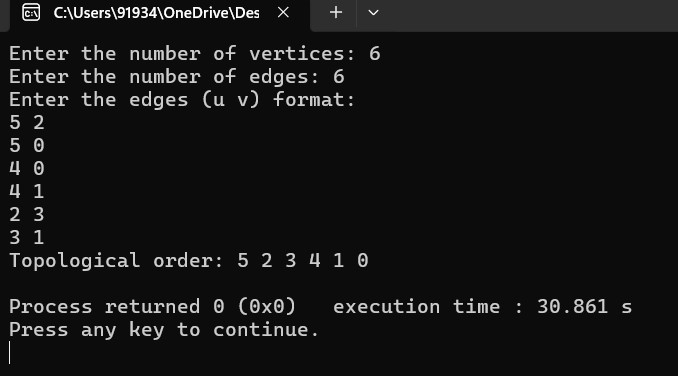
adj[u][v] = 1;

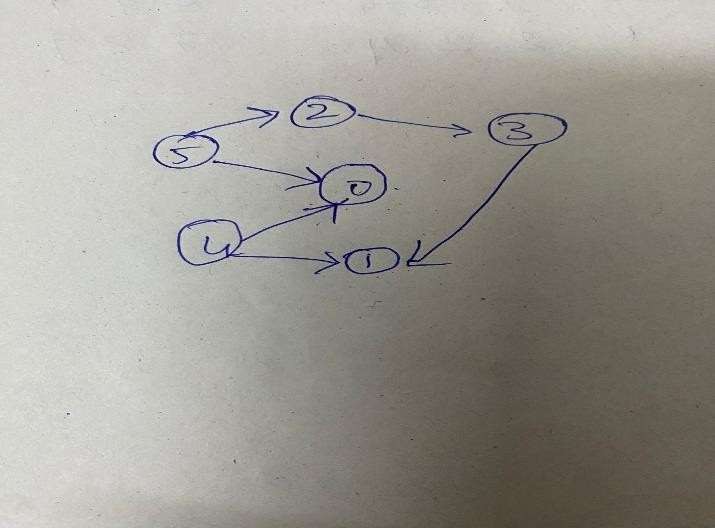
}

topologicalSort(n, adj); return 0;

}

op:





Q2 b) Leet code. Loud and Rich

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int\* loudAndRich(int\*\* richer, int richerSize, int\* richerColSize, int\* quiet, int quietSize, int\* returnSize) {

int\* indegree = (int\*)calloc(quietSize, sizeof(int)); int\* result = (int\*)malloc(quietSize \* sizeof(int)); int\* queue = (int\*)malloc(quietSize \* sizeof(int)); int front = 0, rear = 0;

int\*\* adj = (int\*\*)malloc(quietSize \* sizeof(int\*));

for (int i = 0; i < quietSize; i++) {

adj[i] = (int\*)malloc(quietSize \* sizeof(int)); result[i] = i;

}

int\* adjSize = (int\*)calloc(quietSize, sizeof(int)); for (int i = 0; i < richerSize; i++) {

int a = richer[i][0]; int b = richer[i][1];

adj[a][adjSize[a]++] = b; indegree[b]++;

}

for (int i = 0; i < quietSize; i++) { if (indegree[i] == 0) {

queue[rear++] = i;

}

}

while (front < rear) {

int u = queue[front++];

for (int i = 0; i < adjSize[u]; i++) { int v = adj[u][i];

if (quiet[result[v]] > quiet[result[u]]) { result[v] = result[u];

}

if (--indegree[v] == 0) {

queue[rear++] = v;

}

}

}

\*returnSize = quietSize;

for (int i = 0; i < quietSize; i++) { free(adj[i]);

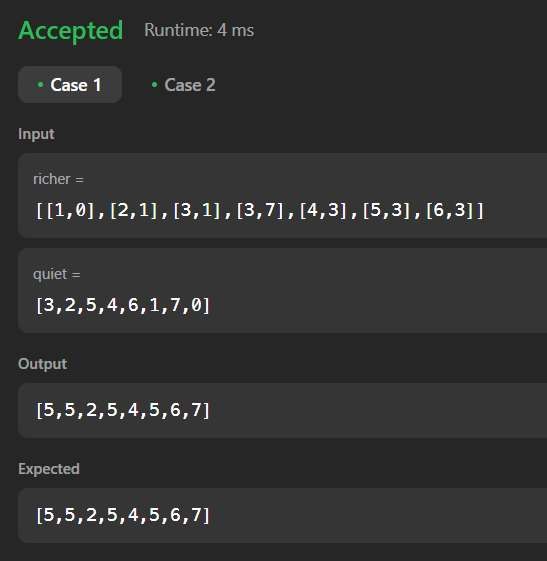
}

free(adj); free(indegree); free(queue); free(adjSize);

return result;

}

op:



Q3) Implement Johnson Trotter algorithm to generate permutations. #include <stdio.h>

#include <stdlib.h> #include <stdbool.h>

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) { printf("%d ", arr[i]);

}

printf("\n");

}

int searchArr(int arr[], int size, int x) { for (int i = 0; i < size; i++) {

if (arr[i] == x) { return i;

}

}

return -1;

}

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

\*a = \*b;

\*b = temp;

}

void generatePermutations(int n) { int perm[n];

int dir[n]; int num;

for (int i = 0; i < n; i++) { printf("\nEnter number:");

scanf("%d",&num); perm[i] = num; dir[i] = 0;

}

while (1) { printArray(perm, n);

int mobile = 0, mobileIndex = -1; for (int i = 0; i < n; i++) {

if ((dir[i] == 0 && i != 0 && perm[i] > perm[i - 1]) || (dir[i] == 1 && i != n - 1 && perm[i] > perm[i + 1])) {

if (perm[i] > mobile) { mobile = perm[i]; mobileIndex = i;

}

}

}

if (mobileIndex == -1) { break;

}

if (dir[mobileIndex] == 0) {

swap(&perm[mobileIndex], &perm[mobileIndex - 1]); swap(&dir[mobileIndex], &dir[mobileIndex - 1]); mobileIndex--;

} else {

swap(&perm[mobileIndex], &perm[mobileIndex + 1]); swap(&dir[mobileIndex], &dir[mobileIndex + 1]); mobileIndex++;

}

for (int i = 0; i < n; i++) { if (perm[i] > mobile) {

dir[i] = !dir[i];

}

}

}

}

int main() { int n;

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

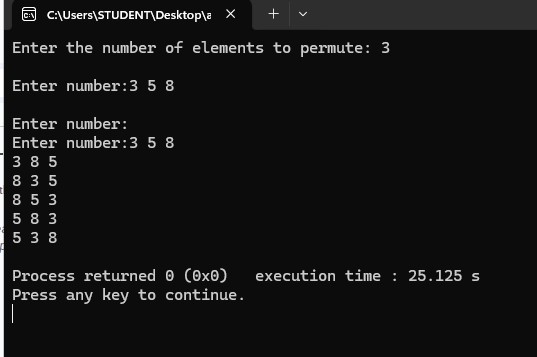
if (n <= 0) {

printf("Number of elements should be greater than 0.\n"); return 1;

}

generatePermutations(n); return 0;

}



Q4.) 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 <stdlib.h> #include <time.h>

void merge(int arr[], int l, int m, int r) { int i, j, k;

int n1 = m - l + 1; int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++) L[i] = arr[l + i];

for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) { if (L[i] <= R[j]) {

arr[k] = L[i]; i++;

} else {

arr[k] = R[j]; j++;

} k++;

}

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

k++;

}

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

k++;

}

}

void mergeSort(int arr[], int l, int r) { if (l < r) {

int m = l + (r - l) / 2; mergeSort(arr, l, m); mergeSort(arr, m + 1, r); merge(arr, l, m, r);

}

}

int main() { int N;

printf("Enter the 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]);

printf("Unsorted array: "); for (int i = 0; i < N; i++)

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

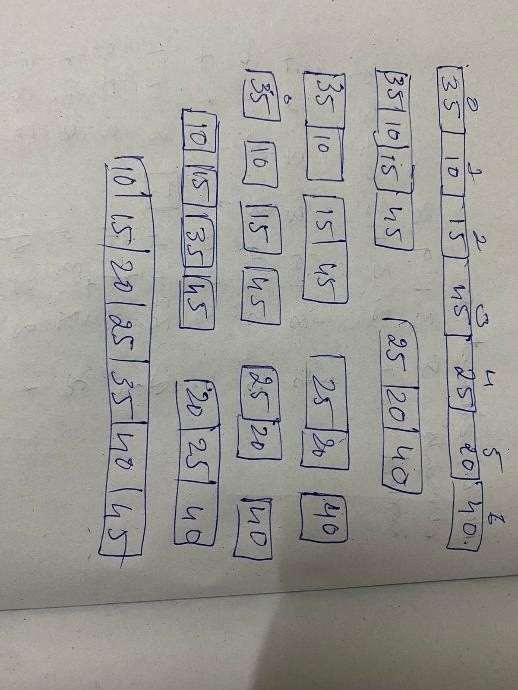
struct timespec start, end; clock\_gettime(CLOCK\_MONOTONIC, &start); mergeSort(arr, 0, N - 1); clock\_gettime(CLOCK\_MONOTONIC, &end);

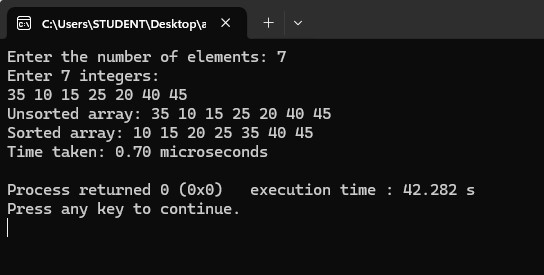
double time\_taken = (end.tv\_sec - start.tv\_sec) \* 1e6 + (end.tv\_nsec - start.tv\_nsec) / 1e3; printf("Sorted array: ");

for (int i = 0; i < N; i++) printf("%d ", arr[i]);

printf("\n");

printf("Time taken: %.2lf microseconds\n", time\_taken); return 0;

}



Q5) 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>

int partition(int arr[], int low, int high) { int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) { if (arr[j] < pivot) {

i++;

int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;

}

}

int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp; return (i + 1);

}

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[n];

printf("Enter the elements: "); for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

clock\_t start, end; double cpu\_time\_used; start = clock(); quickSort(arr, 0, n - 1); end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC; printf("Sorted array: ");

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

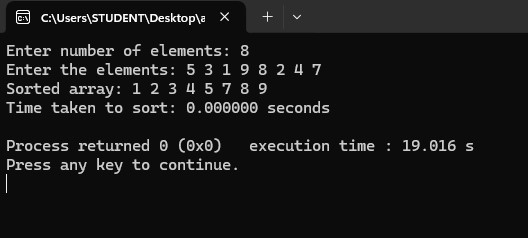
}

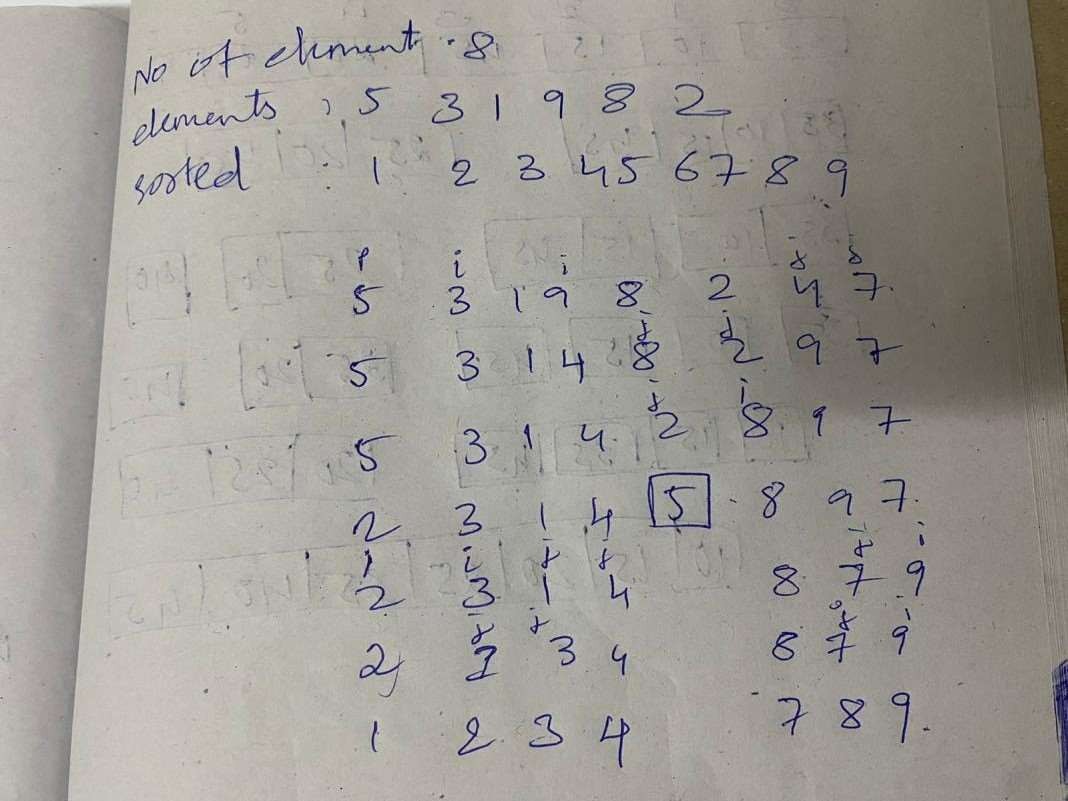
printf("\n");

printf("Time taken to sort: %f seconds\n", cpu\_time\_used); return 0;

}

op:





Q6) Sort a given set of N integer elements using Heap Sort technique and compute its time taken. #include <stdio.h>

#include <stdlib.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 buildHeap(int arr[], int n) { int startIdx = (n / 2) - 1;

for (int i = startIdx; i >= 0; i--) { heapify(arr, n, i);

}

}

void heapSort(int arr[], int n) {

buildHeap(arr, n); printf("Heap before sorting: "); for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

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 the elements: "); for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

clock\_t start, end; double cpu\_time\_used; start = clock();

heapSort(arr, n); end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC; printf("Sorted array: ");

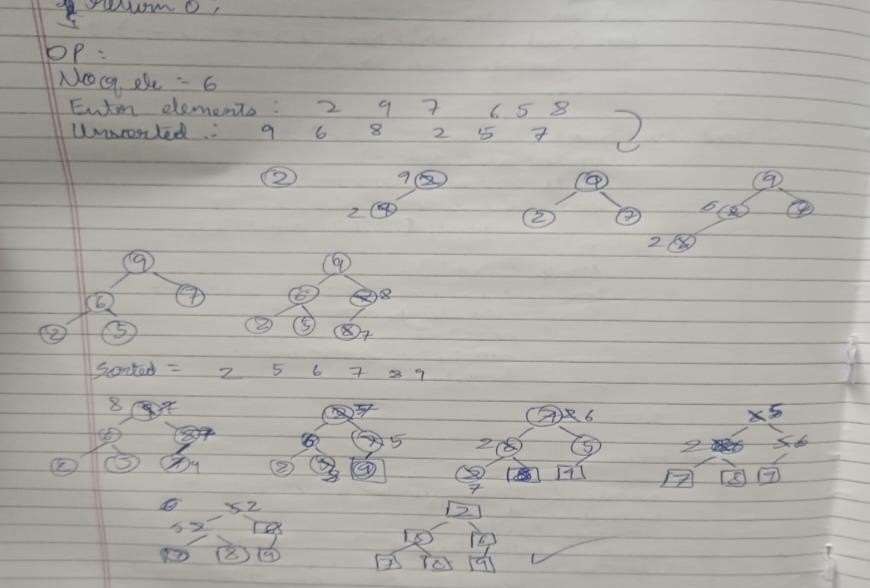
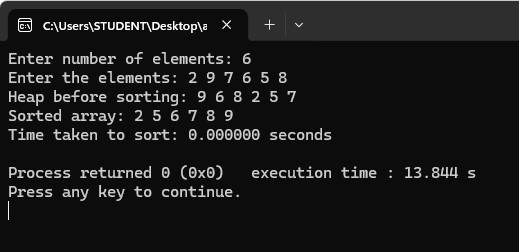
for (int i = 0; i < n; i++) { printf("%d ", arr[i]);

}

printf("\n");

printf("Time taken to sort: %f seconds\n", cpu\_time\_used); return 0;

}



Q7) Implement 0/1 Knapsack problem using dynamic programming. #include <stdio.h>

#include <stdlib.h> struct Item {

int value; int weight;

double ratio;

};

void swap(struct Item\* a, struct Item\* b) { struct Item temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(struct Item arr[], int n, int i) { int largest = i;

int left = 2 \* i + 1; int right = 2 \* i + 2;

if (left < n && arr[left].ratio > arr[largest].ratio) { largest = left;

}

if (right < n && arr[right].ratio > arr[largest].ratio) { largest = right;

}

if (largest != i) {

swap(&arr[i], &arr[largest]); heapify(arr, n, largest);

}

}

void buildHeap(struct Item arr[], int n) { int startIdx = (n / 2) - 1;

for (int i = startIdx; i >= 0; i--) { heapify(arr, n, i);

}

}

int knapsack(int W, struct Item items[], int n) { int \*K = (int \*)calloc((W + 1), sizeof(int));

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

for (int w = W; w >= items[i].weight; w--) {

K[w] = max(K[w], items[i].value + K[w - items[i].weight]);

}

}

int result = K[W];

free(K); return result;

}

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

}

int main() { int n, W;

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

struct Item\* items = (struct Item\*)malloc(n \* sizeof(struct Item));

printf("Enter the values and weights of the items:\n"); for (int i = 0; i < n; i++) {

printf("Item %d:\n", i + 1); printf("Value: ");

scanf("%d", &items[i].value); printf("Weight: "); scanf("%d", &items[i].weight);

items[i].ratio = (double)items[i].value / items[i].weight;

}

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

buildHeap(items, n);

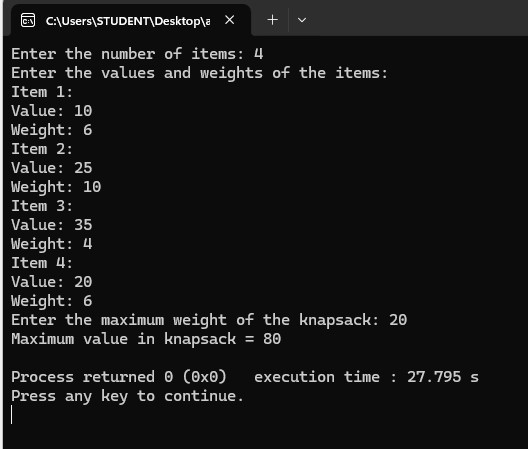
int max\_value = knapsack(W, items, n);

printf("Maximum value in knapsack = %d\n", max\_value);

free(items);

return 0;

}



Q7b) Leet Code: Egg Drop With 2 Eggs and N Floors int twoEggDrop(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

if (n == 2) return 2; int bests[n + 1]; bests[0] = 0;

bests[1] = 1;

bests[2] = 2;

for (int col = 3; col <= n; ++col) { int bestThisCol = n;

for (int row = 1; row <= col; ++row) { int breaks = 1 + row - 1;

int survive = 1 + bests[col - row];

if (bestThisCol > (breaks > survive ? breaks : survive)) { bestThisCol = (breaks > survive ? breaks : survive);

}

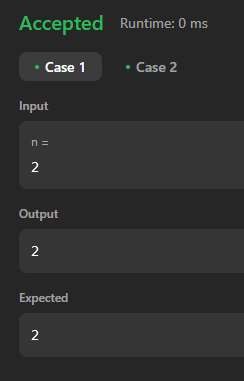
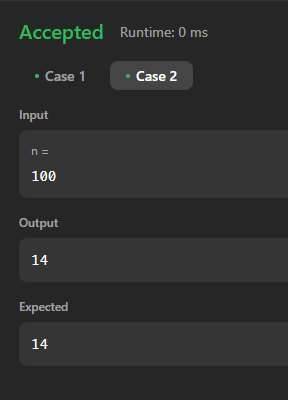
}

bests[col] = bestThisCol;

}

return bests[n];

}

Q8) Implement All Pair Shortest paths problem using Floyd’s algorithm. #include <stdio.h>

#include <stdlib.h> #include <limits.h> #define INF 99999

void floydWarshall(int\*\* graph, int V) { int\*\* dist = (int\*\*)malloc(V \* sizeof(int\*)); for (int i = 0; i < V; i++) {

dist[i] = (int\*)malloc(V \* sizeof(int)); for (int j = 0; j < V; j++) {

if (graph[i][j] == 0 && i != j) {

dist[i][j] = INF; // Use INF to indicate no direct path

} else {

dist[i][j] = graph[i][j];

}

}

}

for (int k = 0; k < V; k++) { for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) { dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

printf("Shortest distance matrix:\n"); for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) { if (dist[i][j] == INF) {

printf("INF ");

} else {

printf("%d ", dist[i][j]);

}

}

printf("\n");

}

for (int i = 0; i < V; i++) { free(dist[i]);

}

free(dist);

}

int main() { int V;

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

int\*\* graph = (int\*\*)malloc(V \* sizeof(int\*)); for (int i = 0; i < V; i++) {

graph[i] = (int\*)malloc(V \* sizeof(int));

}

printf("Enter the weight matrix:\n"); for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) { scanf("%d", &graph[i][j]);

}

}

floydWarshall(graph, V); for (int i = 0; i < V; i++) {

free(graph[i]);

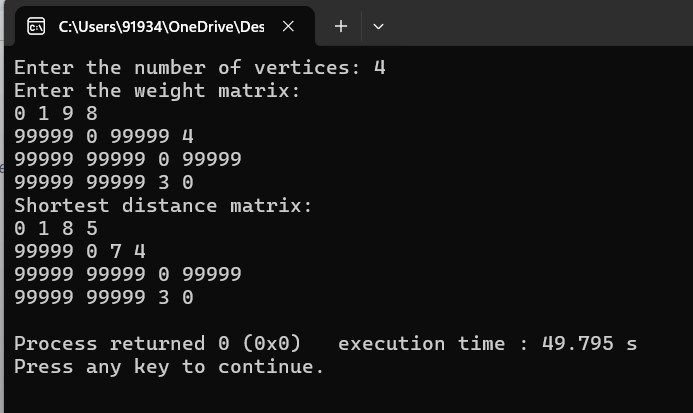
}

free(graph);

return 0;

}

OP:



Q 8b) Leet code.

int findCheapestPrice(int n, int\*\* flights, int flightsSize, int\* flightsColSize, int src, int dst, int k) { int\* dp = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i < n; i++) { dp[i] = INT\_MAX;

}

dp[src] = 0;

for (int i = 0; i <= k; i++) {

int\* temp = (int\*)malloc(n \* sizeof(int)); for (int j = 0; j < n; j++) {

temp[j] = dp[j];

}

for (int f = 0; f < flightsSize; f++) { int\* flight = flights[f];

if (dp[flight[0]] != INT\_MAX) {

temp[flight[1]] = temp[flight[1]] < dp[flight[0]] + flight[2] ? temp[flight[1]] : dp[flight[0]]

+ flight[2];

}

}

free(dp); dp = temp;

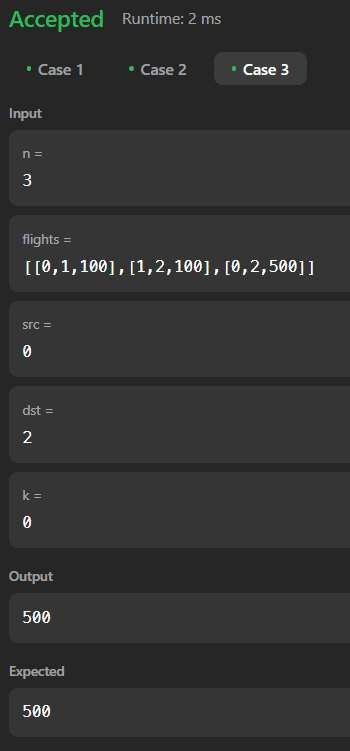
}

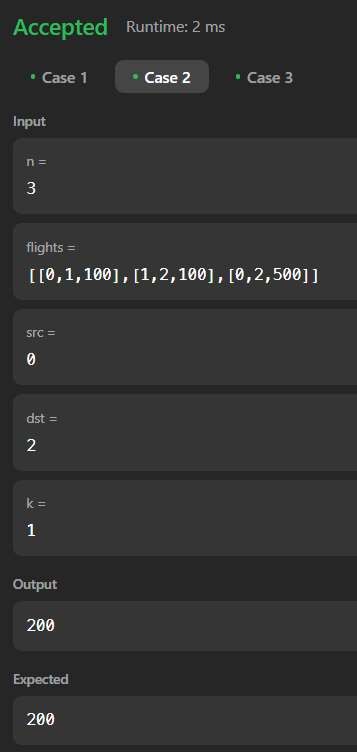
int result = dp[dst] == INT\_MAX ? -1 : dp[dst];

free(dp); return result;

}

op:





Q9a) Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. #include <stdio.h>

#include <float.h> #include <stdbool.h> #define MAX 100

double minKey(double key[], bool mstSet[], int V) { double min = DBL\_MAX;

int min\_index;

for (int v = 0; v < V; v++) {

if (mstSet[v] == false && key[v] < min) { min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], double graph[MAX][MAX], int V) { double totalCost = 0.0;

printf("Edge \tWeight\n"); for (int i = 1; i < V; i++) {

printf("%d - %d \t%.2lf \n", parent[i], i, graph[i][parent[i]]); totalCost += graph[i][parent[i]];

}

printf("Total Cost of MST: %.2lf\n", totalCost);

}

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

double key[MAX]; bool mstSet[MAX];

for (int i = 0; i < V; i++) { key[i] = DBL\_MAX; mstSet[i] = false;

}

key[0] = 0.0;

parent[0] = -1;

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

for (int v = 0; v < V; v++) {

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) { parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph, V);

}

int main() { int V;

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

double graph[MAX][MAX]; printf("Enter the adjacency matrix:\n"); for (int i = 0; i < V; i++) {

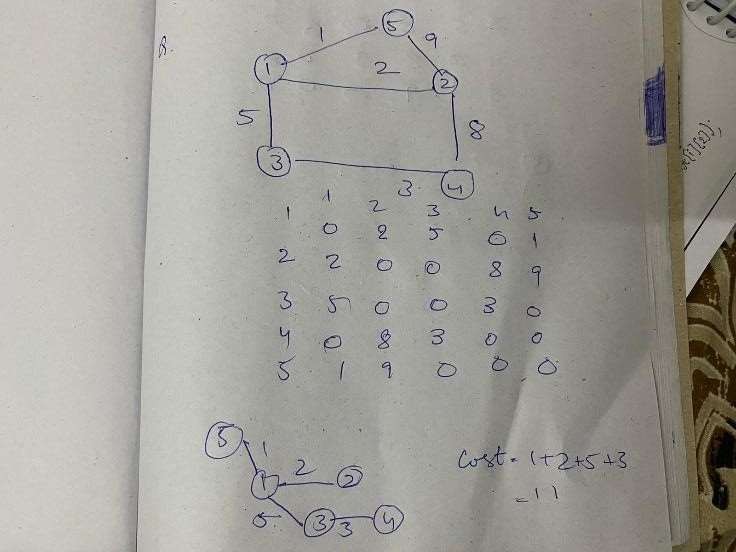
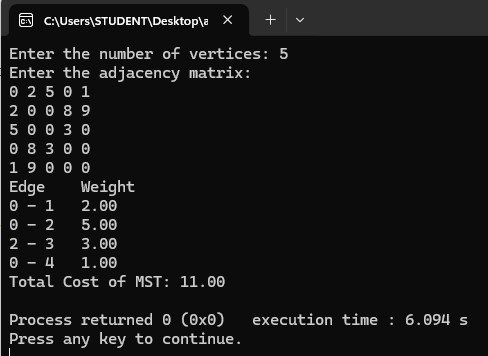
for (int j = 0; j < V; j++) { scanf("%lf", &graph[i][j]);

}

}

primMST(graph, V); return 0;

}



Q9b) Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm. #include <stdio.h>

#include <stdlib.h>

int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[9][9], parent[9]; int find(int i) {

while (parent[i]) i = parent[i];

return i;

}

int uni(int i, int j) { if (i != j) {

parent[j] = i; return 1;

}

return 0;

}

int main() {

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

printf("\nEnter the cost adjacency matrix:\n");

for (i = 1; i <= n; i++) { for (j = 1; j <= n; j++) {

scanf("%d", &cost[i][j]); if (cost[i][j] == 0)

cost[i][j] = 999;

}

}

printf("The edges of Minimum Cost Spanning Tree are:\n"); while (ne < n) {

for (i = 1, min = 999; i <= n; i++) { for (j = 1; j <= n; j++) {

if (cost[i][j] < min) {

min = cost[i][j]; a = u = i;

b = v = j;

}

}

}

u = find(u); v = find(v);

if (uni(u, v)) {

printf("%d edge (%d,%d) = %d\n", ne++, a, b, min); mincost += min;

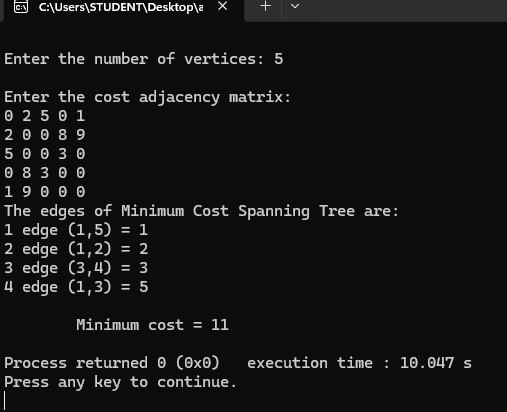
}

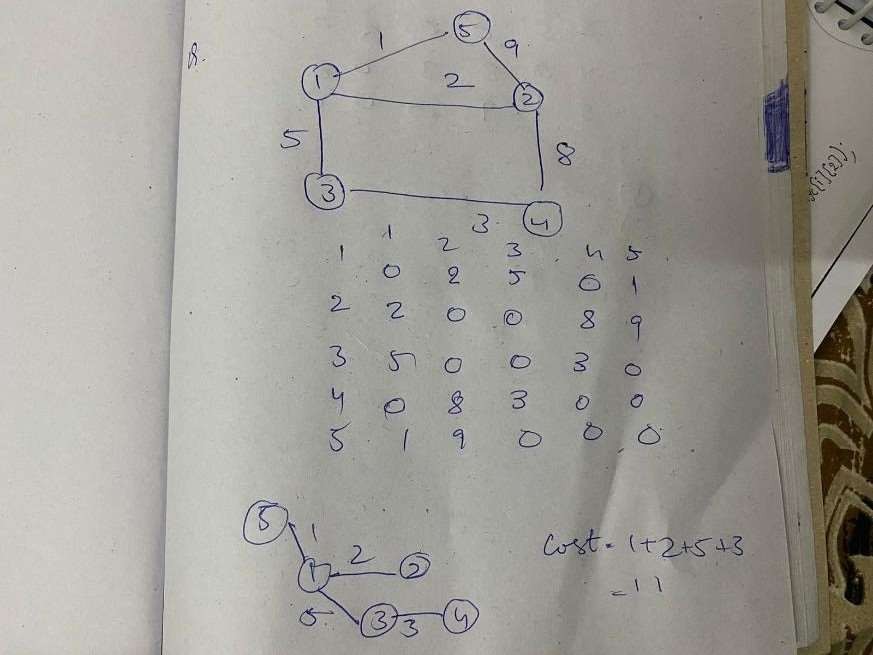
cost[a][b] = cost[b][a] = 999;

}

printf("\n\tMinimum cost = %d\n", mincost); return 0;

}





Q 10) Implement Fractional Knapsack using Greedy technique. #include <stdio.h>

#include <stdlib.h>

double fractionalKnapsack(int n, double W, double weight[], double value[]) { double ratio[n];

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

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

}

int index[n];

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

}

for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) {

if (ratio[index[i]] < ratio[index[j]]) { int temp = index[i];

index[i] = index[j]; index[j] = temp;

}

}

}

double maxVal = 0.0;

for (int i = 0; i < n && W > 0; i++) { int idx = index[i];

if (weight[idx] <= W) { W -= weight[idx]; maxVal += value[idx];

} else {

maxVal += value[idx] \* (W / weight[idx]); break;

}

}

return maxVal;

}

int main() { int n; double W;

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

double weight[n], value[n];

printf("Enter the weights and values of the items:\n"); for (int i = 0; i < n; i++) {

printf("Item %d weight: ", i + 1); scanf("%lf", &weight[i]);

printf("Item %d value: ", i + 1); scanf("%lf", &value[i]);

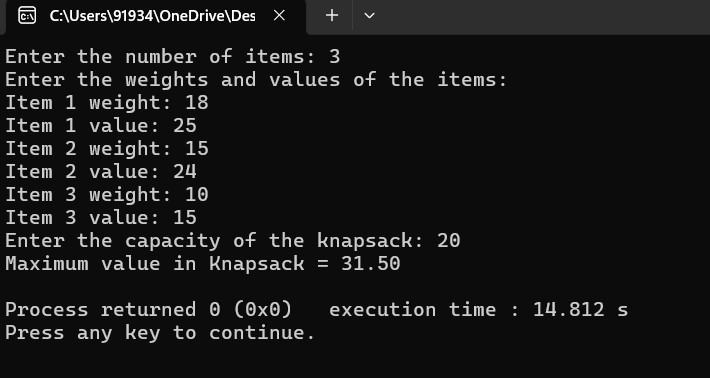
}

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

printf("Maximum value in Knapsack = %.2f\n", fractionalKnapsack(n, W, weight, value)); return 0;

}

O/P



Q11) From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

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

#define MAX\_VERTICES 100

int minDistance(int dist[], bool sptSet[], int V) { int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (sptSet[v] == false && dist[v] <= min) { min = dist[v];

min\_index = v;

}

}

return min\_index;

}

void printSolution(int dist[], int V, int src) { printf("Vertex\tDistance from Source %d\n", src); for (int i = 0; i < V; i++) {

printf("%d\t%d\n", i, dist[i]);

}

}

void dijkstra(int graph[MAX\_VERTICES][MAX\_VERTICES], int src, int V) {

int dist[V]; bool sptSet[V];

for (int i = 0; i < V; i++) { dist[i] = INT\_MAX; sptSet[i] = false;

}

dist[src] = 0;

for (int count = 0; count < V - 1; count++) { int u = minDistance(dist, sptSet, V); sptSet[u] = true;

for (int v = 0; v < V; v++) {

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) { dist[v] = dist[u] + graph[u][v];

}

}

}

printSolution(dist, V, src);

}

int main() { int V;

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

int graph[MAX\_VERTICES][MAX\_VERTICES];

printf("Enter the adjacency matrix (use 0 for no edge and positive weights for edges):\n");

for (int i = 0; i < V; i++) { for (int j = 0; j < V; j++) {

scanf("%d", &graph[i][j]);

}

} int src;

printf("Enter the source vertex (0 to %d): ", V - 1); scanf("%d", &src);

dijkstra(graph, src, V); return 0;

}

