S.No: 1	Exp. Name: ***Write a Program to Search an element using Linear Search and Recursion***	**Date: 2025-08-06**

Aim:

Write a program to search the given element from a list of elements with linear search technique using **recursion**.

Note: Write the functions **read1()** and **linearSearch()** in Program911a.c

Source Code:

Program911.c

```
#include <stdio.h>
#include "Program911a.c"
void main() {
        int a[20], n, pos, key;
        printf("Enter n value : ");
        scanf("%d", &n);
        read1(a, n);
        printf("Enter a key element : ");
        scanf("%d", &key);
        pos = linearSearch(a, 0, n - 1, key);
        if (pos == -1) {
                printf("The key element %d is not found\n", key);
        } else {
                printf("The key element %d is found at position :
%d\n", key, pos);
        }
}
```

Program911a.c

```
#include<stdio.h>
void read1(int a[], int n) {
        printf("Enter %d elements : ",n);
        for (int i=0; i<n; i++) {
                scanf("%d", &a[i]);
        }
}

int linearSearch(int a[], int i, int n, int key){
        if(i > n)
                return -1;

        if(a[i] == key)
                return i;

        return linearSearch(a,i + 1, n, key);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter n value :
4
Enter 4 elements :
10 20 15 12
Enter a key element :
15
The key element 15 is found at position : 2

Test Case - 2
User Output
Enter n value :
6
Enter 6 elements :
2 6 4 1 3 7
Enter a key element :
5

```
The key element 5 is not found
```

S.No: 2	Exp. Name: ***Write a Program to Search an element using Binary Search and Recursion***	**Date: 2025-08-06**

Aim:

Write a program to `search` the given element from a list of elements with `binary search` technique using **recursion**.

Note: Write the functions **read()**, **bubbleSort()**, **display()** and **binarySearch()** in `Program912a.c`

Source Code:

Program912.c

```
#include <stdio.h>
#include "Program912a.c"
void main() {
        int a[20], n, key, flag;
        printf("Enter value of n : ");
        scanf("%d", &n);
        read1(a, n);
        bubbleSort(a, n);
        printf("After sorting the elements are : ");
        display(a, n);
        printf("Enter key element : ");
        scanf("%d", &key);
        flag = binarySearch(a, 0, n - 1, key);
        if (flag == -1) {
                printf("The given key element %d is not found\n", key);
        } else {
                printf("The given key element %d is found at position :
%d\n", key, flag);
        }
}
```

Program912a.c

```
# include <stdio.h>

void read1(int a[], int n){
        printf("Enter %d elements : ",n);
                for(int i =0; i<n ;i++){
                        scanf("%d",&a[i]);
                }
}


void bubbleSort(int a[], int n){
        int temp;
        for(int i=0; i<n-1; i++){
                for(int j=0; j<n-i-1;j++){
                        if(a[j]>a[j+1]){
                                temp = a[j];
                                a[j] = a[j + 1];
                                a[j+1] = temp;
                        }
                }
        }
}

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


int binarySearch(int a[], int low, int high, int key){
        if(low> high){
                return -1;
        }
        int mid = (low+ high)/2;

        if(a[mid] == key){
                return mid;
        }
        if(key < a[mid]){
                return binarySearch(a, low, mid - 1, key);
        }
        else{
                return binarySearch(a, mid + 1, high, key);
```

```
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n :
5
Enter 5 elements :
33 55 22 44 11
After sorting the elements are : 11 22 33 44 55
Enter key element :
11
The given key element 11 is found at position : 0

Test Case - 2
User Output
Enter value of n :
4
Enter 4 elements :
23 67 45 18
After sorting the elements are : 18 23 45 67
Enter key element :
24
The given key element 24 is not found

S.No: 3	Exp. Name: ***Quick sort***	**Date: 2025-08-06**

Aim:

Write a program to perform Quick sort. Display the partial pass-wise sorting done.

Source Code:

quickSort.c

ID: 24070521106
2024-28-CSE-B
Symbiosis Institute of Technology | SIT Nagpur

```c
#include <stdio.h>

// Function to print a part of the array
void printArray(int arr[], int start, int end) {
    for (int i = start; i <= end; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Partition function for Quick Sort
int partition(int arr[], int low, int high, int* pass) {
    int pivot = arr[high];
    int i = low - 1, temp;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            // Swap arr[i] and arr[j]
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Swap arr[i+1] and arr[high]
    temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    // Print current pass
    printf("Pass: ");
    printArray(arr, low, high);
    (*pass)++;

    return i + 1;
}
```

```
            // Quick Sort function
void quickSort(int arr[], int low, int high, int* pass) {
    if (low < high) {
        int pi = partition(arr, low, high, pass);
        quickSort(arr, low, pi - 1, pass);
        quickSort(arr, pi + 1, high, pass);
    }
}

int main() {
    int n, i, pass = 1;

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

    int arr[n]; // VLA: valid in C99+

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

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

    quickSort(arr, 0, n - 1, &pass);

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

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
number of elements:
4
elements:
5 8 9 4
Original array: 5 8 9 4
Pass: 4 8 9 5
Pass: 5 9 8
Pass: 8 9
Sorted array: 4 5 8 9

Test Case - 2
User Output
number of elements:
6
elements:
5 1 10 8 9 7
Original array: 5 1 10 8 9 7
Pass: 5 1 7 8 9 10
Pass: 1 5
Pass: 8 9 10
Pass: 8 9
Sorted array: 1 5 7 8 9 10

S.No: 4	Exp. Name: ***Merge Sort***	Date: 2025-09-24

Aim:

Write a C program to perform Merge sort. Display the partial pass-wise sorting done.

Source Code:

mergeSortAlgo.c

ID: 24070521106

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

void print_subarray(int a[], int l, int r) {
for (int i = l; i <= r; i++) {
printf("%d ", a[i]);
if (i < r)
printf("");
}
printf("\n");
}

void merge(int a[], int l, int m, int r, int temp[]) {
int i = l, j = m + 1, k = l;
while (i <= m && j <= r) {
if (a[i] <= a[j])
temp[k++] = a[i++];
else
temp[k++] = a[j++];
}
while (i <= m)
temp[k++] = a[i++];
while (j <= r)
temp[k++] = a[j++];
for (i = l; i <= r; i++)
a[i] = temp[i];
printf("Pass: ");
print_subarray(a, l, r);
}
void mergesort(int a[], int l, int r, int temp[]) {
if (l >= r)
return;
int m = (l + r) / 2;
mergesort(a, l, m, temp);
mergesort(a, m + 1, r, temp);
merge(a, l, m, r, temp);
}

int main() {
int n;
printf("no of elements: ");
scanf("%d", &n);

int *a = (int *)malloc(n * sizeof(int));
int *temp = (int *)malloc(n * sizeof(int));
```

```
printf("elements: ");
for (int i = 0; i < n; i++) {
scanf("%d", &a[i]);
}
printf("Given array:\n");
for (int i = 0; i < n; i++) {
printf("%d", a[i]);
if (i < n - 1)
printf(" ");
}
printf(" \n");

mergesort(a, 0, n - 1, temp);

printf("Sorted array:\n");
for (int i = 0; i < n; i++) {
printf("%d", a[i]);
if (i < n - 1)
printf(" ");
}
printf(" \n");

free(a);
free(temp);
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
no of elements:
5
elements:
5 3 7 1 9
Given array:
5 3 7 1 9
Pass: 3 5
Pass: 3 5 7

Pass: 1 9
Pass: 1 3 5 7 9
Sorted array:
1 3 5 7 9

Test Case - 2
User Output
no of elements:
8
elements:
8 4 2 7 1 5 3 6
Given array:
8 4 2 7 1 5 3 6
Pass: 4 8
Pass: 2 7
Pass: 2 4 7 8
Pass: 1 5
Pass: 3 6
Pass: 1 3 5 6
Pass: 1 2 3 4 5 6 7 8
Sorted array:
1 2 3 4 5 6 7 8

S.No: 5	Exp. Name: ***Prims Algorithm***	**Date: 2025-08-13**

Aim:

Write a C program to implement Prim's algorithm for finding the Minimum Cost Spanning Tree of a given undirected graph represented by an adjacency matrix.

Input Format:

- The first line contains an integer n, representing the number of vertices in the graph.
- The next n lines each contain n space-separated integers, representing the adjacency matrix of the undirected weighted graph.
- The value at row i and column j denotes the weight of the edge between vertex i and vertex j.
- A value of "0" indicates that there is no edge between the corresponding vertices.

Output Format:

- The program prints the Minimum Spanning Tree (MST) as edges along with their weights.

Note:

- The algorithm starts from **vertex 0**.
- Refer to the visible test cases for better understanding.

Source Code:

```
minCostFinding.c
```


ID: 24070521106

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#define V 100

int minKey(int key[], bool mstSet[], int vertices) {
int min = INT_MAX, min_index;

        for(int v = 0; v < vertices; v++){
                if(!mstSet[v] && key[v] < min) {
                        min = key[v];
                        min_index = v;
                }
        }
        return min_index;
}
void printTree(int parent[], int graph[V][V], int vertices) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < vertices; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

void prim(int graph[V][V], int vertices) {
        int parent[V];
        int key[V];
        bool mstSet[V];
        for (int i = 0; i < vertices; i++){
                key[i] = INT_MAX;
                mstSet[i] = false;
        }

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

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

                for(int v = 0; v < vertices; v++){
                        if (graph[u][v] && !mstSet[v] && graph[u][v] <
key[v]) {
                                parent[v] = u;
                                key[v] = graph[u][v];
                        }
                }
        }
```

```
        printTree(parent, graph, vertices);
}

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

    printf("No of vertices: ");
    scanf("%d", &vertices);

    printf("Adjacency matrix elements (row wise):\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    prim(graph, vertices);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
No of vertices:
5
Adjacency matrix elements (row wise):
0 0 4 0 0
0 0 5 3 0
4 5 0 0 0
0 3 0 0 2
0 0 0 2 0
Edge Weight
2 - 1 5
0 - 2 4
1 - 3 3
3 - 4 2

S.No: 6	Exp. Name: ***kruskals Algorithm***	**Date: 2025-09-24**

Aim:

Write a C program to implement Kruskal's algorithm for finding the Minimum Cost Spanning Tree (MCST) and the total minimum cost of travel for a given undirected graph. The graph will be represented by an adjacency matrix.

Input Format:

- The first input should be an integer, representing the number of vertices in the graph.
- The next input should be an adjacency matrix representing the weighted graph.
- If there is no edge between two vertices, the weight should be given as 9999 (representing infinity).

Output Format:

- The program should print the edges selected in the Minimum Spanning Tree (MST) along with their weights.

Note:

- Refer to the visible test cases to strictly match the input and output layout.

Source Code:

minCostFinding.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define INF 9999

typedef struct {
    int src, dest, weight;
} Edge;

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

void union1(int parent[], int rank[], int x, int y) {
    int xroot = find(parent, x);
    int yroot = find(parent, y);

    if (rank[xroot] < rank[yroot])
        parent[xroot] = yroot;
    else if (rank[xroot] > rank[yroot])
        parent[yroot] = xroot;
    else {
        parent[yroot] = xroot;
        rank[xroot]++;
    }
}

int compareEdges(const void *a, const void *b) {
    Edge *e1 = (Edge *)a;
    Edge *e2 = (Edge *)b;
    if (e1->weight != e2->weight)
        return e1->weight - e2->weight;
    if (e1->src != e2->src)
        return e1->src - e2->src;
    return e1->dest - e2->dest;
}


void kruskalMST(int **cost, int V) {
    Edge *edges = (Edge *)malloc(V * V * sizeof(Edge));
    int edgeCount = 0;

    for (int i = 0; i < V; i++) {
        for (int j = i + 1; j < V; j++) {
```

```
            if (cost[i][j] != INF) {
                edges[edgeCount].src = i;
                edges[edgeCount].dest = j;
                edges[edgeCount].weight = cost[i][j];
                edgeCount++;
            }
        }
        }
    qsort(edges, edgeCount, sizeof(Edge), compareEdges);

    int *parent = (int *)malloc(V * sizeof(int));
    int *rank = (int *)malloc(V * sizeof(int));
    for (int i = 0; i < V; i++) {
        parent[i] = i;
        rank[i] = 0;
    }

    int e = 0, i = 0, minCost = 0;

    while (e < V - 1 && i < edgeCount) {
        Edge next_edge = edges[i++];
        int x = find(parent, next_edge.src);
        int y = find(parent, next_edge.dest);

        if (x != y) {
            printf("Edge %d:(%d, %d) cost:%d\n", e, next_edge.src,
next_edge.dest, next_edge.weight);
            minCost += next_edge.weight;
            union1(parent, rank, x, y);
            e++;
                }
        }

    printf("Minimum cost= %d\n", minCost);
    free(edges);
    free(parent);
    free(rank);
}
int main() {
    int V;
    printf("No of vertices: ");
    scanf("%d", &V);

    int **cost = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++)
        cost[i] = (int *)malloc(V * sizeof(int));
```

```
    printf("Adjacency matrix:\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &cost[i][j]);

    kruskalMST(cost, V);

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

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
No of vertices:
5
Adjacency matrix:
9999 2 9999 9999 5
2 9999 3 9999 9999
9999 3 9999 4 9999
9999 9999 4 9999 9999
5 9999 9999 9999 9999
Edge 0:(0, 1) cost:2
Edge 1:(1, 2) cost:3
Edge 2:(2, 3) cost:4
Edge 3:(0, 4) cost:5
Minimum cost= 14

Test Case - 2
User Output
No of vertices:
4
Adjacency matrix:

9999 3 6 3
3 9999 5 2
6 5 9999 4
3 2 4 9999
Edge 0:(1, 3) cost:2
Edge 1:(0, 1) cost:3
Edge 2:(2, 3) cost:4
Minimum cost= 9

S.No: 7	Exp. Name: ***Dijkstra's Shortest Path Algorithm***	**Date: 2025-09-24**

Aim:

Given a graph G and source vertex S, Dijkstra's shortest path algorithm is used to find the shortest paths from source S to all vertices in the given graph.

The Dijkstra algorithm is also known as the single-source shortest path algorithm. It is based on the greedy technique. A little variation in the algorithm can find the shortest path from the source nodes to all the other nodes in the graph.

The function **void dijkstra(int** $G[MAX][MAX]$**, int** n**, int** $startnode$**)** computes and prints the shortest path distances and corresponding paths from the given source node to all other nodes in a weighted directed graph using Dijkstra's algorithm. It outputs the distance or "INF" if unreachable, along with the path or "NO PATH" for each node.

Note:

- Vertices are numbered from 1 through V.
- All input values are separated by spaces and/or newlines.

Sample Input and Output:

```
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2
Enter the source :1
Node    Distance    Path
2   4   2<-1
3   6   3<-1
4   8   4<-3<-1
```

Source Code:

Dijkstras.c

```
#include <limits.h>
#include <stdio.h>
#define MAX 20
int V, E;
int graph[MAX][MAX];
#define INFINITY 99999
void printPath(int parent[], int j) {
    int path[MAX];
    int pathIndex = 0;

    // Collect path from destination to source
    while (j != -1) {
        path[pathIndex++] = j;
        j = parent[j];
    }

    // Now print in reverse: destination <- ... <- source
    for (int i = 0; i < pathIndex; i++) {
        printf("%d", path[i]);
        if (i < pathIndex - 1) {
            printf("<-");
        }
    }
}
void dijkstra(int G[MAX][MAX], int n, int startnode) {
    int distance[MAX];
    int visited[MAX] = {0};
    int parent[MAX];

    for (int i = 1; i <= n; i++) {
        distance[i] = INFINITY;
        parent[i] = -1;
    }
    distance[startnode] = 0;

    for (int count = 1; count <= n - 1; count++) {
        int min = INFINITY, u = -1;
        for (int i = 1; i <= n; i++) {
            if (!visited[i] && distance[i] <= min) {
                min = distance[i];
                u = i;
            }
        }

        if (u == -1) break;
```

```
        visited[u] = 1;

        for (int v = 1; v <= n; v++) {
            if (!visited[v] && G[u][v] != 0 && distance[u] != INFINITY
&&
                distance[u] + G[u][v] < distance[v]) {
                distance[v] = distance[u] + G[u][v];
                parent[v] = u;
            }
        }
        }
        // write your code here
        printf("Node\tDistance\tPath\n");
        for (int i = 1; i <= n; i++) {
            if (i == startnode) continue;

            // Print node with 3 leading spaces and arrow
            printf("   %d\t", i);

            if (distance[i] == INFINITY) {
                // 7 spaces after arrow
                printf("     INF\tNO PATH\n");
            } else {
                // Print distance followed by 7 spaces, then arrow and
path
                printf("       %d\t", distance[i]);
                printPath(parent, i);
                printf("\n");
            }
        }
}
int main() {
        int s, d, w, i, j;
        printf("Enter the number of vertices : ");
        scanf("%d", &V);
        printf("Enter the number of edges : ");
        scanf("%d", &E);
        for(i = 1 ; i <= V; i++) {
                for(j = 1; j <= V; j++) {
                        graph[i][i] = 0;
                }
        }
        for(i = 1; i <= E; i++) {
                printf("Enter source : ");
                scanf("%d", &s);
                printf("Enter destination : ");
```

```
                scanf("%d", &d);
                printf("Enter weight : ");
                scanf("%d", &w);
                if(s > V || d > V || s <= 0 || d <= 0) {
                        printf("Invalid index. Try again.\n");
                        i--;
                        continue;
                } else {
                        graph[s][d] = w;
                }
        }
        printf("Enter the source :");
        scanf("%d", &s);
        dijkstra(graph, V, s);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
1
Enter destination :
4
Enter weight :
10
Enter source :
1

```
Enter destination :
3
Enter weight :
6
Enter source :
2
Enter destination :
4
Enter weight :
5
Enter source :
3
Enter destination :
4
Enter weight :
2
Enter the source :
1
Node    Distance        Path
   2          4        2<-1
   3          6        3<-1
   4          8        4<-3<-1
```

Test Case - 2

User Output

```
Enter the number of vertices :
5
Enter the number of edges :
6
Enter source :
1
Enter destination :
2
Enter weight :
2
Enter source :
1
Enter destination :
5
```

```
Enter weight :
3
Enter source :
2
Enter destination :
4
Enter weight :
4
Enter source :
2
Enter destination :
3
Enter weight :
7
Enter source :
4
Enter destination :
3
Enter weight :
2
Enter source :
5
Enter destination :
4
Enter weight :
1
Enter the source :
2
Node    Distance        Path
   1         INF        NO PATH
   3           6        3<-4<-2
   4           4        4<-2
   5         INF        NO PATH
```

Test Case - 3

User Output

```
Enter the number of vertices :
4
Enter the number of edges :
```

```
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
3
Enter destination :
2
Enter weight :
5
Enter source :
4
Enter destination :
1
Enter weight :
1
Enter source :
4
Enter destination :
2
Enter weight :
3
Enter source :
4
Enter destination :
3
Enter weight :
8
Enter the source :
1
Node    Distance        Path
   2          4        2<-1
   3        INF        NO PATH
   4        INF        NO PATH
```

S.No: 8	Exp. Name: ***Knapsack Problem***	**Date: 2025-09-24**

Aim:

Problem Description:

Given the weights and values of N objects, place them in a bag with a capacity of W to calculate the bag's maximum possible total value.To put it another way, given are two integer arrays, val[0..N-1] and wt[0..N-1], which, respectively, represent values and weights connected to N items.

Additionally, given an integer W that represents the capacity of a knapsack, determine the largest value subset of val[] such that the total of its weights is less than or equal to W. An item cannot be broken; you must either pick it in its entirety or not at all (0-1 property).

Note: Please take a note that we only have one quantity of each item.

Constraints:

$1 \leq N, W \leq 1000$

$1 \leq val[i], wt[i] \leq 1000$

Input Format:

- The first line represents the size of both the arrays N.
- The second line represents the set of elements of val[ ].
- The third line represents the set of elements of wt[ ].
- The next line contains an integer representing the knapsack capacity W.

Output Format:

- An integer representing the maximum total value in the knapsack which is smaller than or equal to W.

Sample Test Case:

Input: N = 3, W = 4

values[N] = {1,2,3}

weight[N] = {4,5,1}

Output: 3

Source Code:

maxValueInKnapsack.c

```
#include <stdio.h>

#define MAX 100

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

int knapsack(int val[], int wt[], int N, int W) {
        int dp[MAX + 1][MAX + 1];

        for (int i = 0; i <= N; i++) {
                for (int w = 0; w <= W; w++) {
                        if (i == 0 || w == 0)
                                dp[i][w] = 0;
                        else if (wt[i - 1] <= w)
                                dp[i][w] = max(dp[i - 1][w], val[i - 1]
+ dp[i - 1][w - wt[i - 1]]);
                        else
                                dp[i][w] = dp[i - 1][w];
                }
        }

        return dp[N][W];
}

int main() {
        int N, W;
        int val[MAX], wt[MAX];

        scanf("%d", &N);
        for (int i = 0; i < N; i++)
                scanf("%d", &val[i]);
        for (int i = 0; i < N; i++)
                scanf("%d", &wt[i]);
        scanf("%d", &W);

        int result = knapsack(val, wt, N, W);
        printf("%d\n", result);

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
1 2 3
4 5 1
4
3

Test Case - 2
User Output
3
1 2 3
4 5 6
3
0

S.No: 9	Exp. Name: ***Fractional Knapsack Problem***	**Date: 2025-09-24**

Aim:

The below program has a method void knapsack(). Which takes four parameters **number of objects**, the **weight of each object**, the **profit** corresponding to each one and the **capacity of the knapsack**. Write a program using a fractional knapsack algorithm to get the maximum profit.

Print the output as follows:

```
Sample Input and Output:
Enter the no. of objects: 6
Enter the weights and profits of each object:
1 2
4 5
8 9
4 6
5 2
3 5
Enter the capacity of knapsack:10
Maximum profit is:- 15.500000
```

Source Code:

knapsack.c

ID: 24070521106

```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
  // write your code here
        float total_profit = 0.0;
        int i;

        for (i = 0; i < n; i++) {
                if (capacity == 0) {
                        break;
                }
                if (weight[i] <= capacity) {
                        capacity -= weight[i];
                        total_profit += profit[i];
                } else {
                        total_profit += profit[i] * (capacity /
weight[i]);
                        capacity = 0;
                }
        }

        printf("Maximum profit is:- %.6f\n", total_profit);









}

int main() {
        float weight[20], profit[20], capacity;
        int num, i, j;
        float ratio[20], temp;
        printf("Enter the no. of objects: ");
        scanf("%d", &num);
        printf("Enter the weights and profits of each object:\n");
        for (i = 0; i < num; i++) {
                scanf("%f %f", &weight[i], &profit[i]);
        }
        printf("Enter the capacity of knapsack:");
        scanf("%f", &capacity);
        for (i = 0; i < num; i++) {
                ratio[i] = profit[i] / weight[i];
        }
```

```
        for (i = 0; i < num; i++) {
                for (j = i + 1; j < num; j++) {
                        if (ratio[i] < ratio[j]) {
                                temp = ratio[j];
                                ratio[j] = ratio[i];
                                ratio[i] = temp;
                                temp = weight[j];
                                weight[j] = weight[i];
                                weight[i] = temp;
                                temp = profit[j];
                                profit[j] = profit[i];
                                profit[i] = temp;
                        }
                }
        }
        knapsack(num, weight, profit, capacity);
        return(0);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the no. of objects:
6
Enter the weights and profits of each object:
1 2
4 5
8 9
4 6
5 2
3 5
Enter the capacity of knapsack:
10
Maximum profit is:- 15.500000

Test Case - 2
User Output

Enter the no. of objects:
5
Enter the weights and profits of each object:
4 6
1 3
7 5
5 3
3 4
Enter the capacity of knapsack:
10
Maximum profit is:- 14.428572

S.No: 10	Exp. Name: ***Floyd - Warshall's Algorithm***	**Date: 2025-09-24**

Aim:

Implement the Floyd-Warshall algorithm in C for finding the shortest distances between all pairs of vertices in a weighted directed graph. Prompt the user to input the number of vertices (N) and edges (E), and then accept edge information (source, destination, and weight) to build the adjacency matrix.

Source Code:

Warshall.c

ID: 24070521106

```c
#include <stdio.h>
#define INF 99999
#define MAX_N 20 // Maximum value for N

#include <stdio.h>

#define MAX 100         // Max number of vertices

int main() {
    int n, e;
    int dist[MAX][MAX];

    // Use the INF constant already defined by Codetantra (INF = 99999)

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

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

    // Initialize the distance matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j)
                dist[i][j] = 0;
            else
                dist[i][j] = INF;
        }
    }

    // Read edges
    for (int i = 0; i < e; i++) {
        int u, v, w;
        printf("Enter source : ");
        scanf("%d", &u);
        printf("Enter destination : ");
        scanf("%d", &v);
        printf("Enter weight : ");
        scanf("%d", &w);
        dist[u - 1][v - 1] = w;  // 1-based to 0-based index
    }

    // Floyd-Warshall Algorithm
    for (int k = 0; k < n; k++) {
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF &&
                    dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    // Output the distance matrix
    printf("The following matrix shows the shortest distances between
all pairs of the vertices.\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("%5s", "INF");
            else
                printf("%5d", dist[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter weight :
4

```
Enter source :
1
Enter destination :
4
Enter weight :
10
Enter source :
1
Enter destination :
3
Enter weight :
6
Enter source :
2
Enter destination :
4
Enter weight :
5
Enter source :
3
Enter destination :
4
Enter weight :
2
The following matrix shows the shortest distances between all pairs
of the vertices.
    0    4    6    8
  INF    0  INF    5
  INF  INF    0    2
  INF  INF  INF    0
```

Test Case - 2

User Output

```
Enter the number of vertices :
5
Enter the number of edges :
6
Enter source :
1
```

```
Enter destination :
2
Enter weight :
2
Enter source :
1
Enter destination :
5
Enter weight :
3
Enter source :
2
Enter destination :
4
Enter weight :
4
Enter source :
2
Enter destination :
3
Enter weight :
7
Enter source :
4
Enter destination :
3
Enter weight :
2
Enter source :
5
Enter destination :
4
Enter weight :
1
The following matrix shows the shortest distances between all pairs
of the vertices.
    0    2    6    4    3
  INF    0    6    4  INF
  INF  INF    0  INF  INF
  INF  INF    2    0  INF
  INF  INF    3    1    0
```

Test Case - 3
User Output
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
3
Enter destination :
2
Enter weight :
5
Enter source :
4
Enter destination :
1
Enter weight :
1
Enter source :
4
Enter destination :
2
Enter weight :
3
Enter source :
4
Enter destination :
3
Enter weight :
8
The following matrix shows the shortest distances between all pairs of the vertices.
0 4 INF INF

INF 0 INF INF
INF 5 0 INF
1 3 8 0

Test Case - 4
User Output
Enter the number of vertices :
4
Enter the number of edges :
6
Enter source :
1
Enter destination :
2
Enter weight :
1
Enter source :
1
Enter destination :
4
Enter weight :
3
Enter source :
2
Enter destination :
3
Enter weight :
6
Enter source :
3
Enter destination :
1
Enter weight :
-2
Enter source :
4
Enter destination :
2
Enter weight :

```
5
Enter source :
4
Enter destination :
3
Enter weight :
10
The following matrix shows the shortest distances between all pairs
of the vertices.
    0    1    7    3
    4    0    6    7
   -2   -1    0    1
    8    5   10    0
```