LAB 09 :

Perform Knapsack problem using Dynamic programming technique using n=4 objects with associated weights and profits .
Display the table values and the objects selected in the knapsack to get maximum profit.

CODE:

```
#include <stdio.h>

#define MAX_OBJECTS 100

// Function to calculate maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve 0/1 Knapsack problem using Dynamic Programming
void knapsack(int n, int W, int weights[], int profits[]) {
    int i, w;
    int K[MAX_OBJECTS + 1][W + 1]; // DP table to store results

    // Build DP table K[][] in bottom-up manner
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (weights[i - 1] <= w)
                K[i][w] = max(profits[i - 1] + K[i - 1][w - weights[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // Print DP table
    printf("DP Table:\n");
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            printf("%d\t", K[i][w]);
        }
        printf("\n");
    }

    // Maximum profit will be at K[n][W]
    int maxProfit = K[n][W];
```

```
    printf("Maximum profit: %d\n", maxProfit);

    // To find the selected items
    printf("Objects selected in the knapsack:\n");
    int res = maxProfit;
    w = W;
    for (i = n; i > 0 && res > 0; i--) {
        if (res == K[i - 1][w])
            continue;
        else {
            printf("Object %d (weight = %d, profit = %d)\n", i, weights[i - 1], profits[i - 1]);
            // Move to the previous item considering its weight
            res -= profits[i - 1];
            w -= weights[i - 1];
        }
    }
}

int main() {
    int n, W;
    int weights[MAX_OBJECTS], profits[MAX_OBJECTS];
    int i;

    // Input number of objects
    printf("Enter number of objects (max %d): ", MAX_OBJECTS);
    scanf("%d", &n);

    if (n <= 0 || n > MAX_OBJECTS) {
        printf("Invalid number of objects\n");
        return 1;
    }

    // Input weights of objects
    printf("Enter the weights of the objects:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }

    // Input profits of objects
    printf("Enter the profits of the objects:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &profits[i]);
    }

    // Input knapsack capacity
    printf("Enter the capacity of the knapsack: ");
```

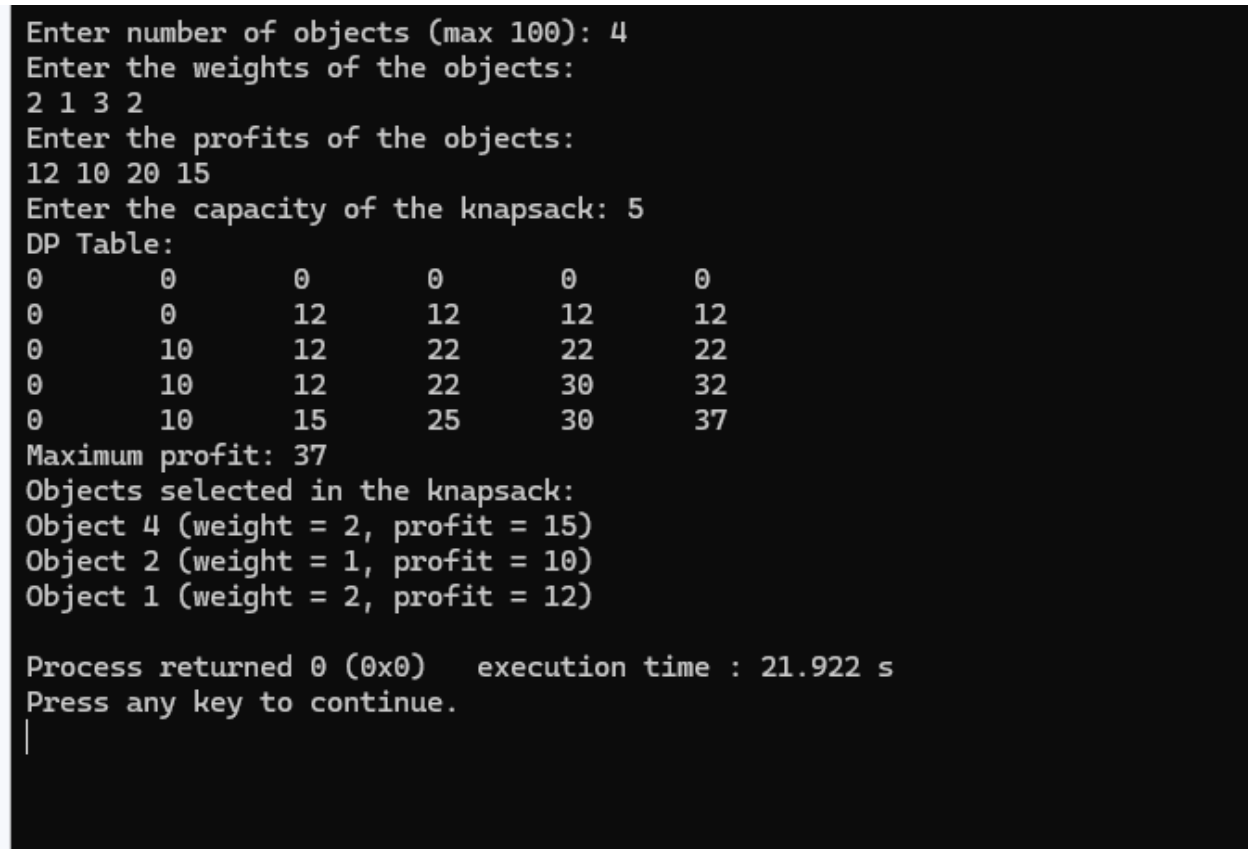
```
    scanf("%d", &W);

    if (W <= 0) {
        printf("Invalid knapsack capacity\n");
        return 1;
    }

    knapsack(n, W, weights, profits);

    return 0;
}
```

OUTPUT:



2. Pfa of the Prims algorithm pseudo code please try to convert this into C program and find the MST of a Given graph with cost adjacency matrix as input.

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

#define MAX_VERTICES 100
#define INF INT_MAX // Infinity

// Function to find vertex with minimum distance value from the set of vertices not yet
included in MST
int minKey(int n, int d[], int s[]) {
    int min = INF, min_index;

    for (int v = 0; v < n; v++) {
        if (s[v] == 0 && d[v] < min) {
            min = d[v];
            min_index = v;
        }
    }

    return min_index;
}

// Function to print MST using parent array and calculate total cost
int printMST(int n, int p[], int cost[MAX_VERTICES][MAX_VERTICES]) {
    int total_cost = 0;
    printf("Edge   Weight\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d    %d \n", p[i], i, cost[i][p[i]]);
        total_cost += cost[i][p[i]];
    }
    return total_cost;
}

// Function to implement Prim's MST algorithm
void primMST(int n, int cost[MAX_VERTICES][MAX_VERTICES]) {
    int p[MAX_VERTICES];  // Array to store constructed MST
    int d[MAX_VERTICES];  // Key values used to pick minimum weight edge in cut
    int s[MAX_VERTICES];  // To represent set of vertices included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < n; i++) {
        d[i] = INF;
        s[i] = 0;
    }
```

```
    // Start with the first vertex as source
    d[0] = 0;     // Make key 0 so that this vertex is picked as first vertex
    p[0] = -1;    // First node is always root of MST

    // The MST will have n vertices
    for (int count = 0; count < n - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet included in MST
        int u = minKey(n, d, s);

        // Add the picked vertex to the MST set
        s[u] = 1;

        // Update key value and parent index of the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not yet included in MST
        for (int v = 0; v < n; v++) {
            // Update the key only if cost[u][v] is smaller than d[v]
            if (cost[u][v] && s[v] == 0 && cost[u][v] < d[v]) {
                p[v] = u;
                d[v] = cost[u][v];
            }
        }
    }

    // Print the constructed MST and calculate total cost
    int total_cost = printMST(n, p, cost);
    printf("Total cost of Minimum Spanning Tree (MST): %d\n", total_cost);
}

int main() {
    int n;
    int cost[MAX_VERTICES][MAX_VERTICES];

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

    // Input the cost adjacency matrix
    printf("Enter the cost adjacency matrix (use %d for infinity):\n", INF);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0 && i != j) { // Diagonal elements remain zero
                cost[i][j] = INF; // Replace zero with infinity
            }
        }
    }
```

```
    // Apply Prim's algorithm to find Minimum Spanning Tree
    printf("Minimum Spanning Tree (MST) using Prim's algorithm:\n");
    primMST(n, cost);

    return 0;
}
```

OUTPUT: