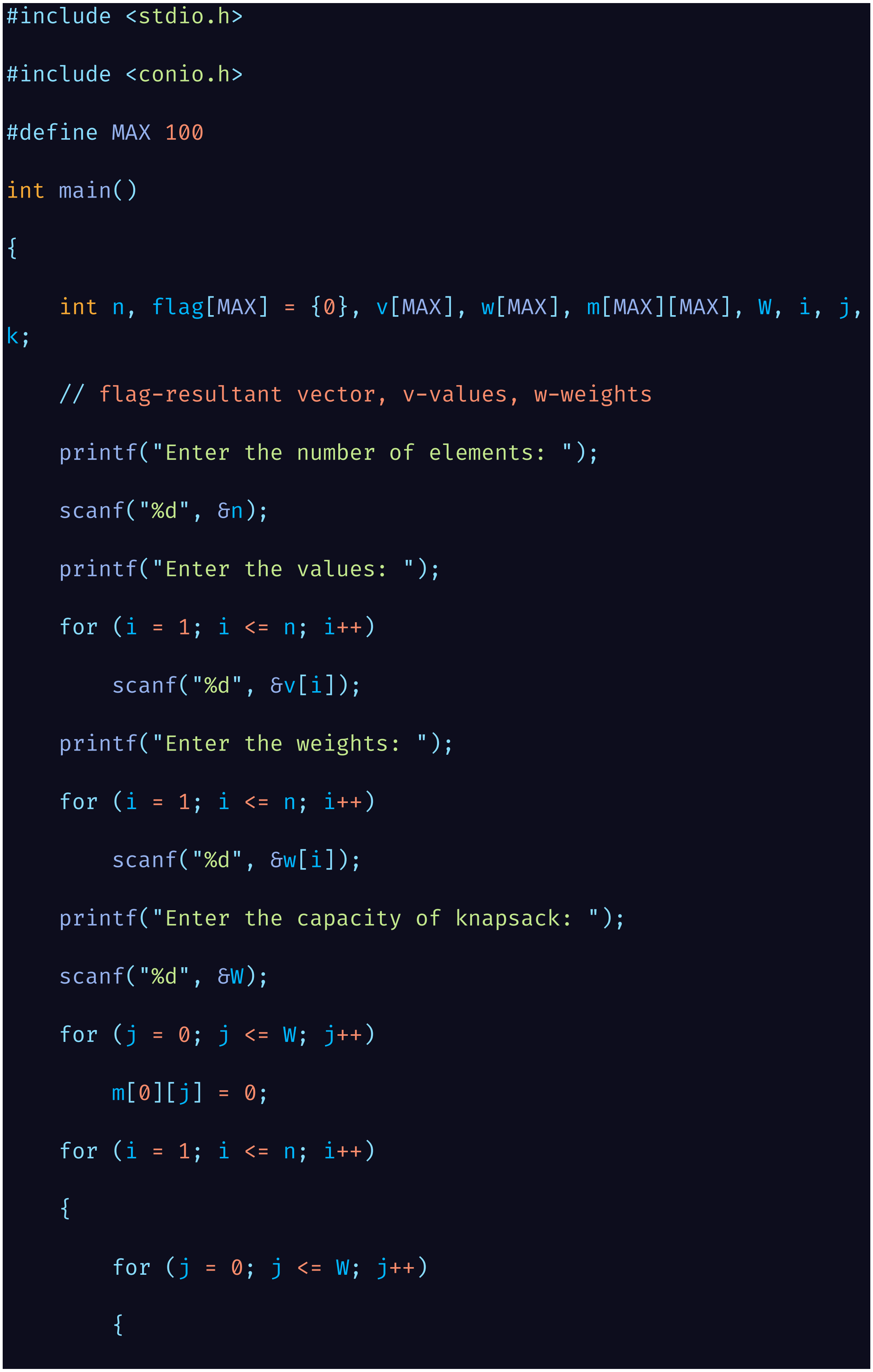
NAME:	Shubham Vishwakarma
UID No.	2021700071
BRANCH:	S.Y CSE-DS
BATCH:	D
SUBJECT	Design and Analysis of Algorithms
EXPERIMENT No.	8
Date of Performance	12/04/2023
Date of Submission	18/04/2023

AIM:	**Branch and bound (To implement 0/1 Knapsack problem using Branch and Bound.)**
Program 1	
PROBLEM STATEMENT :	Implement the 0/1 knapsack algorithm for the given scenario.

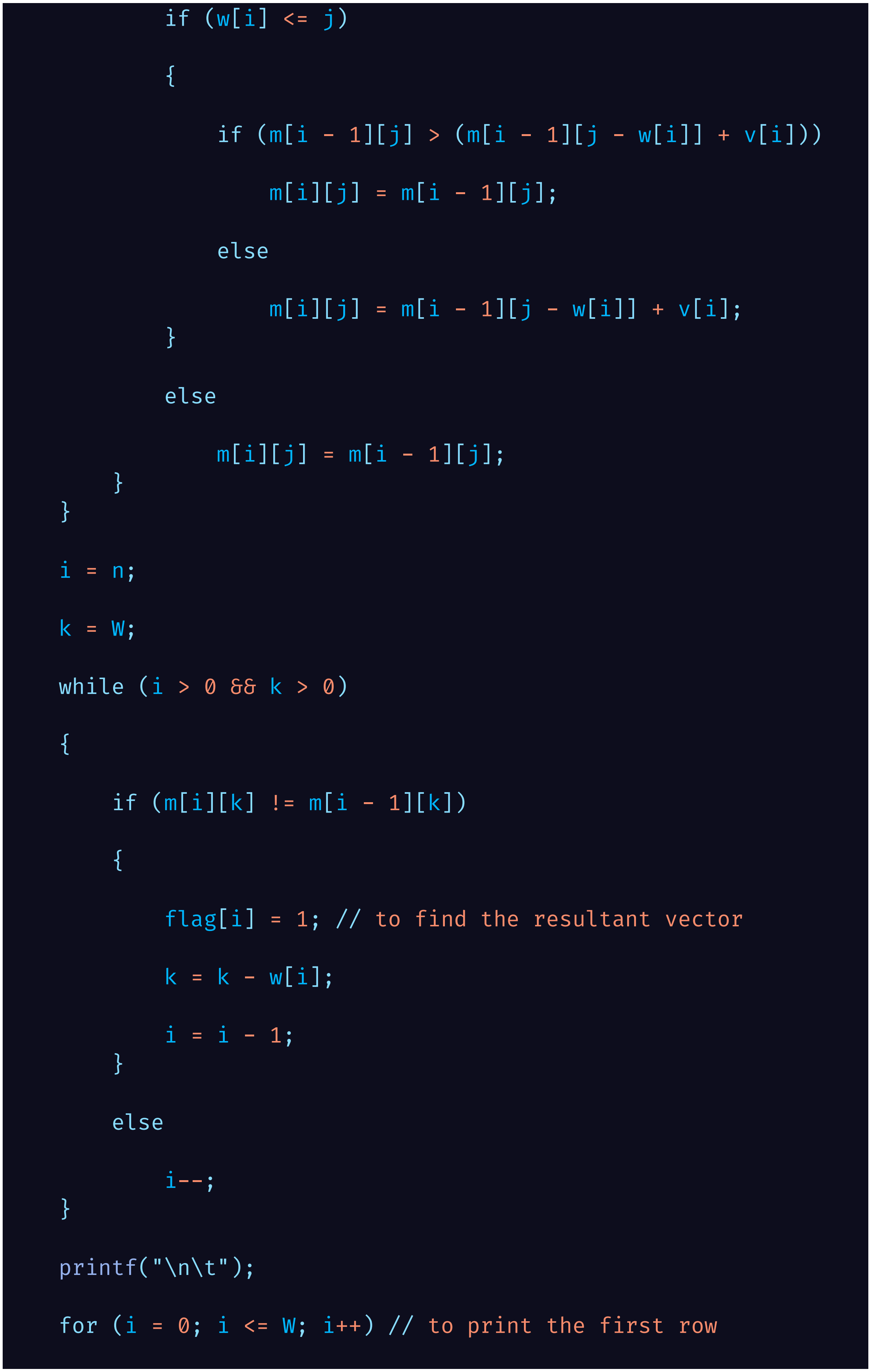
ALGORITHM/ THEORY:

```
Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
   c[0, w] = 0
for i = 1 to n do
   c[i, 0] = 0
   for w = 1 to W do
      if wi ≤ w then
         if vi + c[i-1, w-wi] then
            c[i, w] = vi + c[i-1, w-wi]
         else c[i, w] = c[i-1, w]
      else
         c[i, w] = c[i-1, w]
```

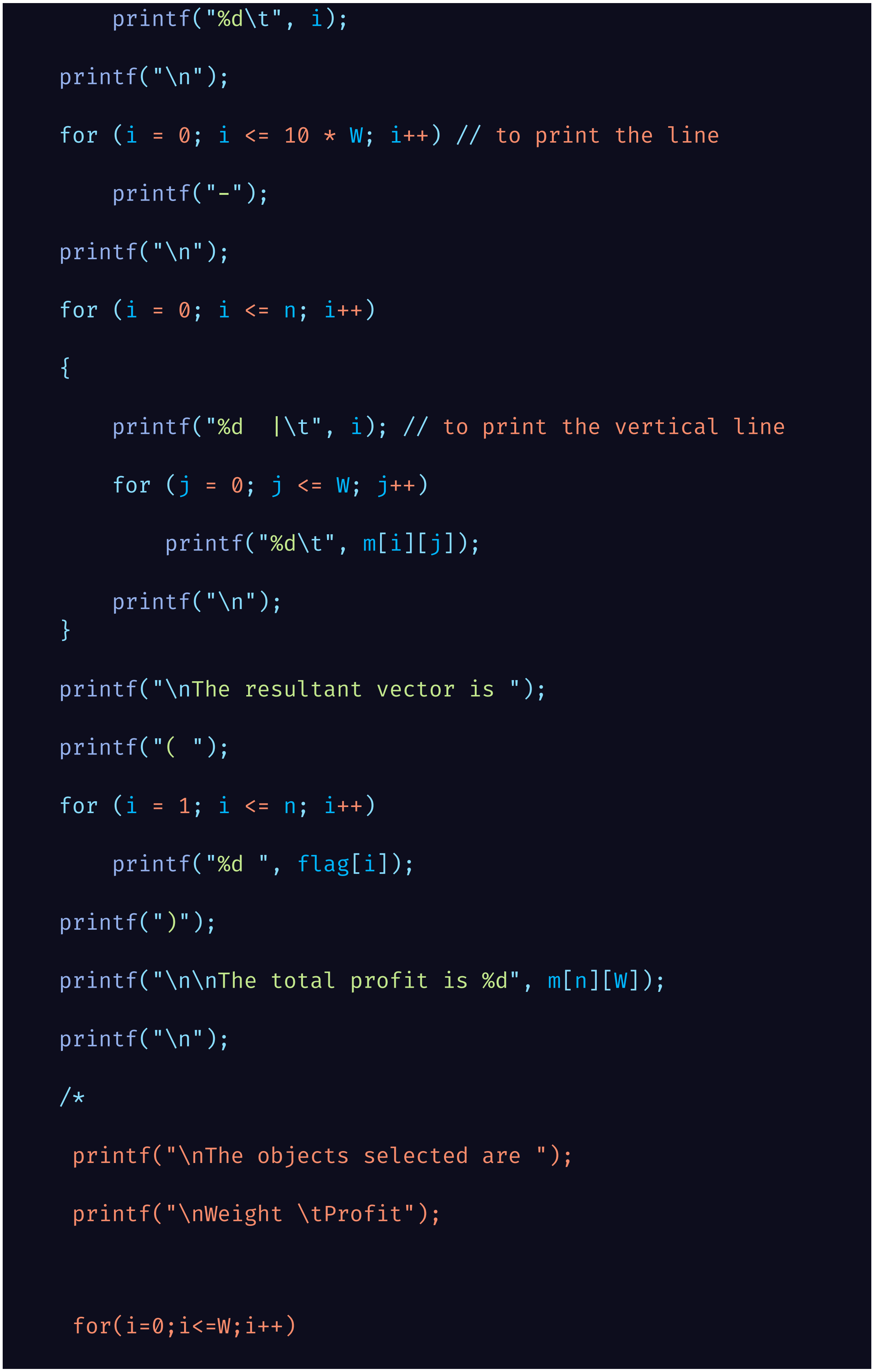
PROGRAM:



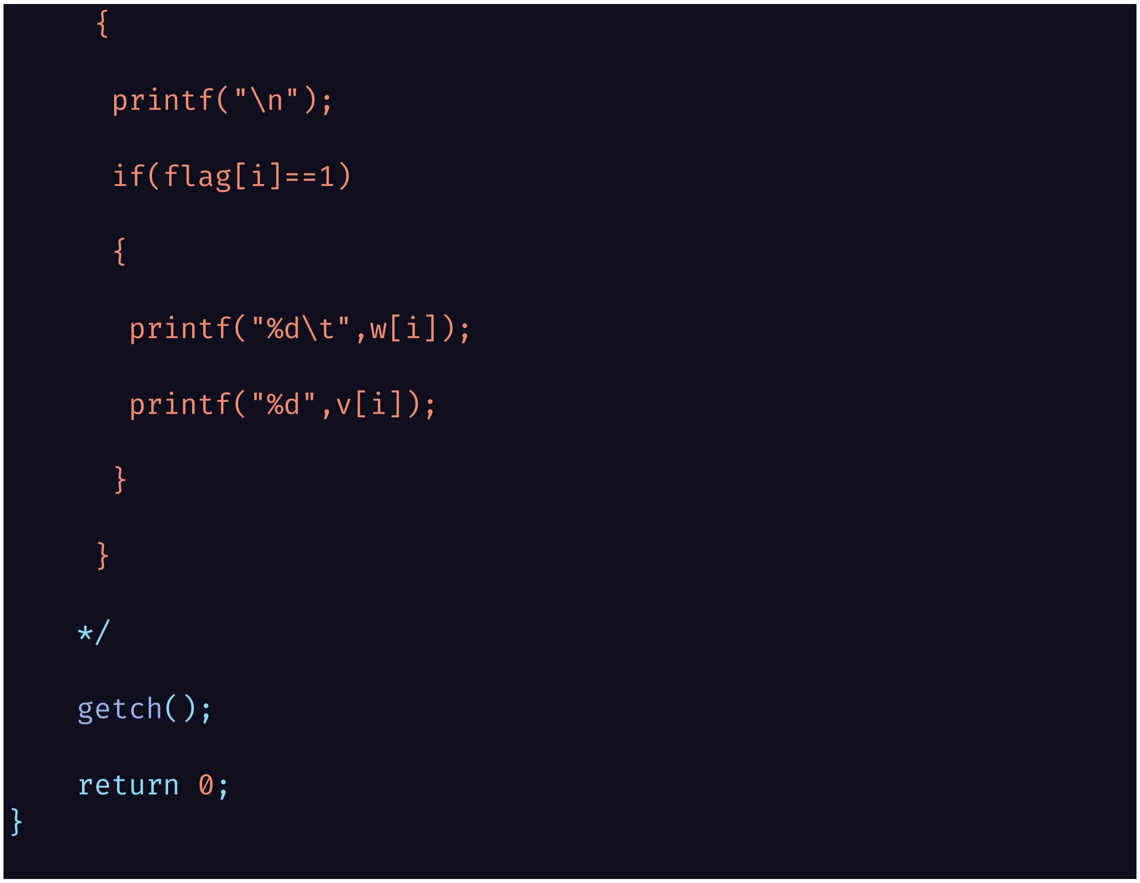
```c
#include <stdio.h>
#include <conio.h>
#define MAX 100
int main()
{
    int n, flag[MAX] = {0}, v[MAX], w[MAX], m[MAX][MAX], W, i, j, k;
    // flag-resultant vector, v-values, w-weights
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the values: ");
    for (i = 1; i <= n; i++)
        scanf("%d", &v[i]);
    printf("Enter the weights: ");
    for (i = 1; i <= n; i++)
        scanf("%d", &w[i]);
    printf("Enter the capacity of knapsack: ");
    scanf("%d", &W);
    for (j = 0; j <= W; j++)
        m[0][j] = 0;
    for (i = 1; i <= n; i++)
    {
        for (j = 0; j <= W; j++)
        {
```



```
            if (w[i] <= j)

            {

                if (m[i - 1][j] > (m[i - 1][j - w[i]] + v[i]))

                    m[i][j] = m[i - 1][j];

                else

                    m[i][j] = m[i - 1][j - w[i]] + v[i];
            }

            else

                m[i][j] = m[i - 1][j];
        }
    }

    i = n;

    k = W;

    while (i > 0 && k > 0)

    {

        if (m[i][k] != m[i - 1][k])

        {

            flag[i] = 1; // to find the resultant vector

            k = k - w[i];

            i = i - 1;
        }

        else

            i--;
    }

    printf("\n\t");

    for (i = 0; i <= W; i++) // to print the first row
```

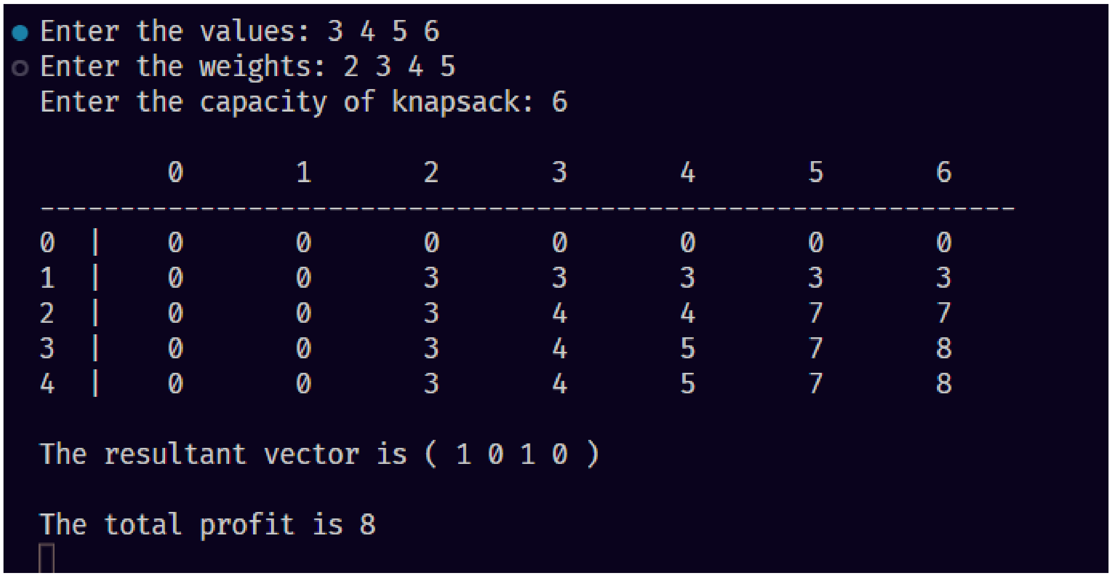


```
    printf("%d\t", i);

printf("\n");

for (i = 0; i <= 10 * W; i++) // to print the line

    printf("-");

printf("\n");

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

{

    printf("%d  |\t", i); // to print the vertical line

    for (j = 0; j <= W; j++)

        printf("%d\t", m[i][j]);

    printf("\n");
}

printf("\nThe resultant vector is ");

printf("( ");

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

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

printf(")");

printf("\n\nThe total profit is %d", m[n][W]);

printf("\n");

/*

 printf("\nThe objects selected are ");

 printf("\nWeight \tProfit");


 for(i=0;i<=W;i++)
```



```
     {

      printf("\n");

      if(flag[i]==1)

      {

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

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

      }

     }

    */

    getch();

    return 0;
}
```

RESULT:



```
● Enter the values: 3 4 5 6
○ Enter the weights: 2 3 4 5
  Enter the capacity of knapsack: 6

          0       1       2       3       4       5       6
  -------------------------------------------------------------
  0  |    0       0       0       0       0       0       0
  1  |    0       0       3       3       3       3       3
  2  |    0       0       3       4       4       7       7
  3  |    0       0       3       4       5       7       8
  4  |    0       0       3       4       5       7       8

  The resultant vector is ( 1 0 1 0 )

  The total profit is 8
```

CONCLUSION :	In the 0-1 Knapsack Problem we have to **pick the optimal set of items among all valid combinations of items**. The valid combination here means that the total weight of all the selected items is less than or equal to the maximum capacity of the knapsack.