Name: Phadtare Shivendra Uday **Class:** BE Div: B

Roll No: COB434 **Batch:** B2

Assignment No:4

Write a program to solve the 0-1 Knapsack Problem using dynamic programming or branch and bound strategy.

CODE:

```
#include <bits/stdc++.h> using
namespace std;

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

// Recursive knapsack function
int knapSack(int W, int wt[], int val[], int n) {
// Base case: no items or capacity 0 if (n
== 0 || W == 0)
        return 0;

// If weight of current item is more than capacity, skip it if
(wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

else
        // Return maximum of two cases:
        // 1. Including current item
        // 2. Excluding current item
        return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1), knapSack(W, wt,
                val, n - 1));
}

int main() {
int profit[] = {70, 90, 10};
int weight[] = {100, 30, 30};
int W = 40;  // Capacity of knapsack  int n =
sizeof(profit) / sizeof(profit[0]);
cout << "Maximum profit for the given capacity is: " << knapSack(W, weight, profit, n) << endl;

return 0;
}
```

OUTPUT:

```
(base) sspm@sspm:~$ g++ daa1.cpp
(base) sspm@sspm:~$ ./a.out
Maximum profit for the given capacity is: 90
```