Practical No:- 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Name:- Kawale Sagar Dattatraya

Roll Number:- CO426

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and

bound strategy.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* A Naive recursive implementation of 0-1 Knapsack problem \*/ #include <bits/stdc++.h>

using namespace std;

int max(int a, int b) { return (a > b) ? a : b; } int knapSack(int W, int wt[], int val[], int n)

{

// Base Case

if (n == 0 || W == 0)

return 0;

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

else

}

return max(

val[n - 1]

+ knapSack(W - wt[n - 1], wt, val, n - 1),

knapSack(W, wt, val, n - 1));

int main()

{

int profit[] = { 60, 100, 120 };

int weight[] = { 10, 20, 30 }; int W = 50;

int n = sizeof(profit) / sizeof(profit[0]); cout << knapSack(W, weight, profit, n); return 0;

}

/\* OUTPUT:- 220

\*/