K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: C2 **Roll No.:16010122257**

Experiment No.____5____

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Knapsack Problem using Greedy strategy

Objective: To learn the Greedy strategy of solving the problems for different types of problems

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Saratai Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://lcm.csa.iisc.ernet.in/dsa/node184.htm**
4. **http://students.ceid.upatras.gr/~papagel/project/kruskal.htm**
5. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.html**
6. **http://lcm.csa.iisc.ernet.in/dsa/node183.html**
7. **http://students.ceid.upatras.gr/~papagel/project/prim.htm**
8. **http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:

The knapsack problem represents constraint satisfaction optimization problems' family. Based on nature of constraints, the knapsack problem can be solved with various problem saolving strategies. Typically, these problems represent resource optimization solution.

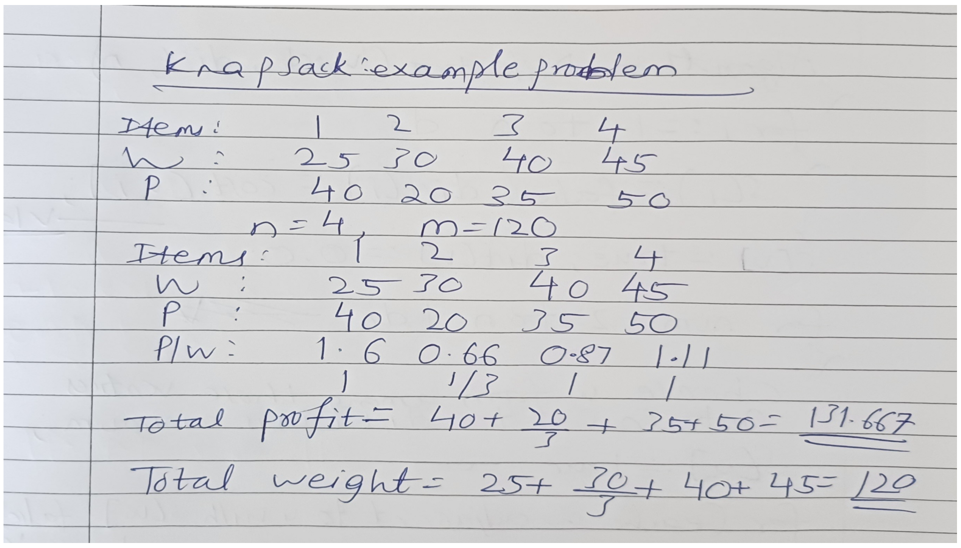
Given a set of n inputs. · Find a subset, called feasible solution, of the n inputs subject to some constraints, and satisfying a given objective function. · If the objective function is maximized or minimized, the feasible solution is optimal. · It is a locally optimal method.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution, knapsack problem and their applications

Knapsack Problem Algorithm

```
Algorithm GreedyKnapsack (m, n)
// P[1 : n] and w[1 : n] contain the profits and weights respectively of
// Objects ordered so that p[i] / w[i]> p[i + 1] / w[i + 1].
// m is the knapsack size and x[1: n] is the solution vector.
{
          for i := 1 to n do x[i]  := 0.0                    // initialize x
          U := m;
          for i := 1 to n do
          {
                    if (w(i) > U) then break;
                    x [i] := 1.0; U := U – w[i];
          }
          if (i ≤ n) then x[i] := U / w[i];
}
```

Example: Knapsack Problem



Knapsack : example problem

Item :	1	2	3	4
w :	25	30	40	45
P :	40	20	35	50

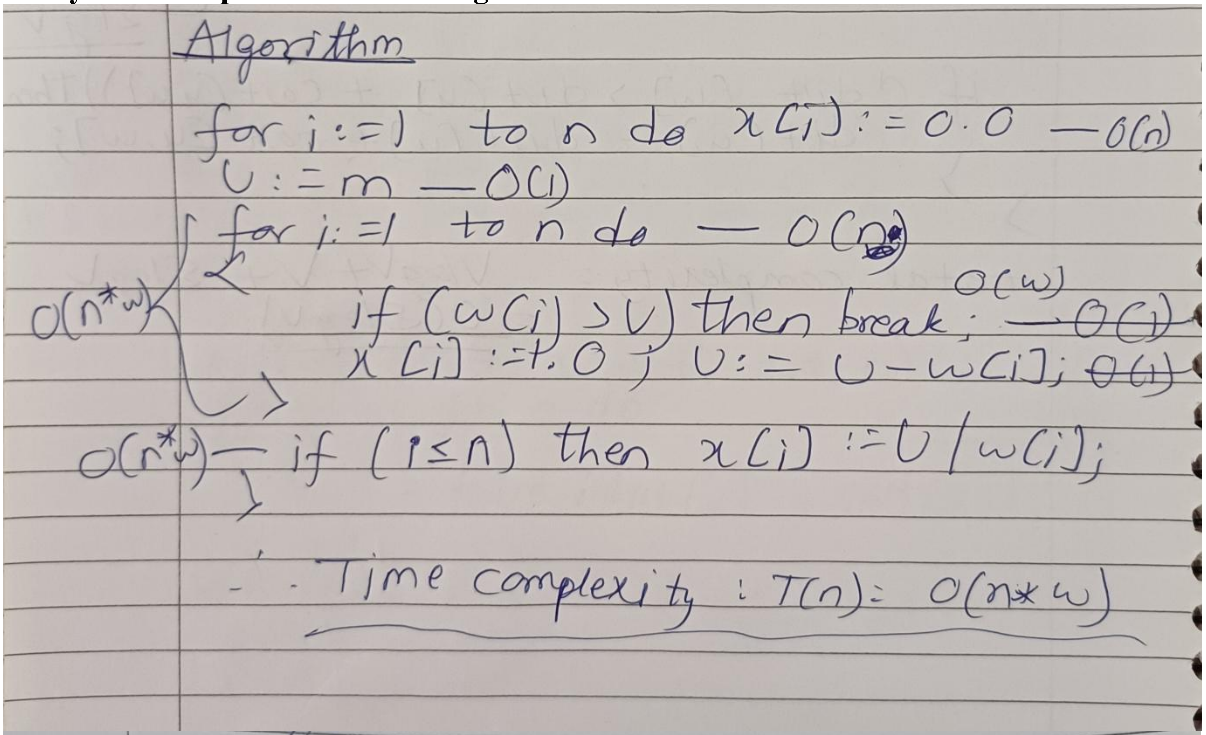
$n = 4, \quad m = 120$

Items :	1	2	3	4
w :	25	30	40	45
P :	40	20	35	50
P/w :	1.6	0.66	0.87	1.11
	1	1/3	1	1

Total profit $= 40 + \frac{20}{3} + 35 + 50 = \underline{\underline{131.667}}$

Total weight $= 25 + \frac{30}{3} + 40 + 45 = \underline{\underline{120}}$

Analysis of Knapsack Problem algorithm:

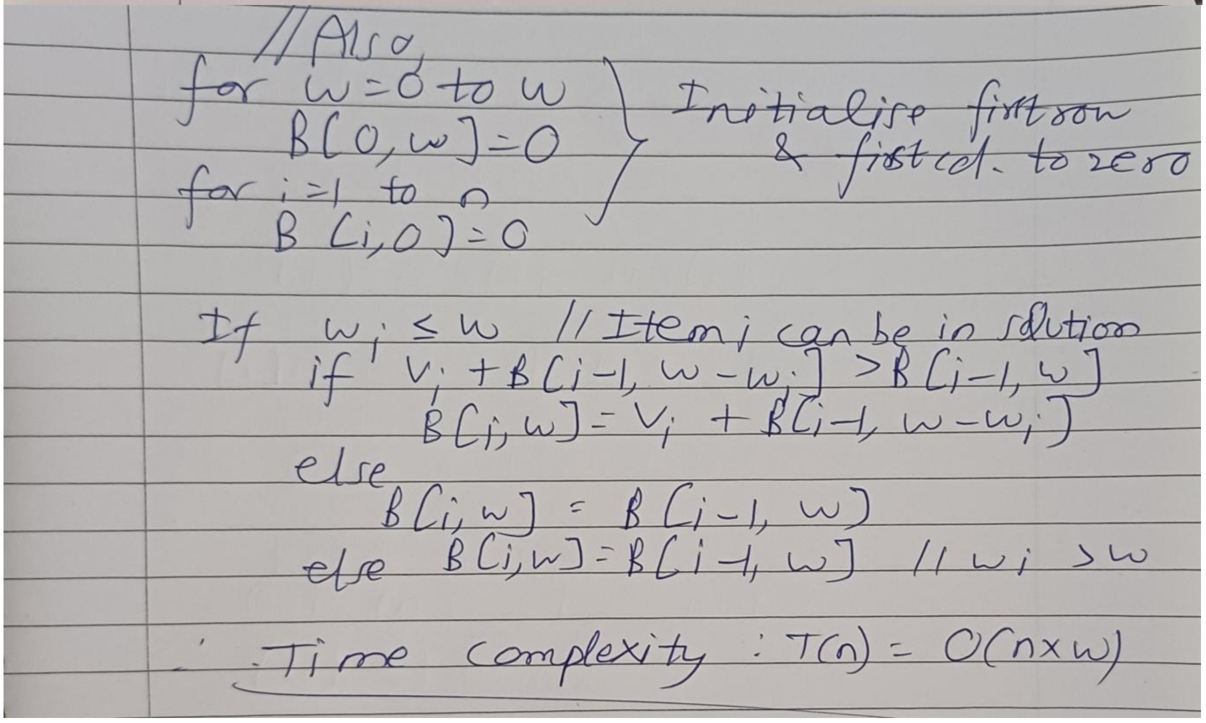


Algorithm

```
for i:=1 to n do x[i]:=0.0            — O(n)
 U:=m — O(1)
 for i:=1 to n do                     — O(n)
 {                                      O(w)
        if (w[i] > U) then break;     — ~~O(1)~~
        x[i] :=1.0 ; U:= U−w[i];      ~~O(1)~~
 }
O(n*w) — if (i≤n) then x[i] := U/w[i];
      }
```

$O(n*w)$

∴ Time complexity : $T(n) = O(n*w)$



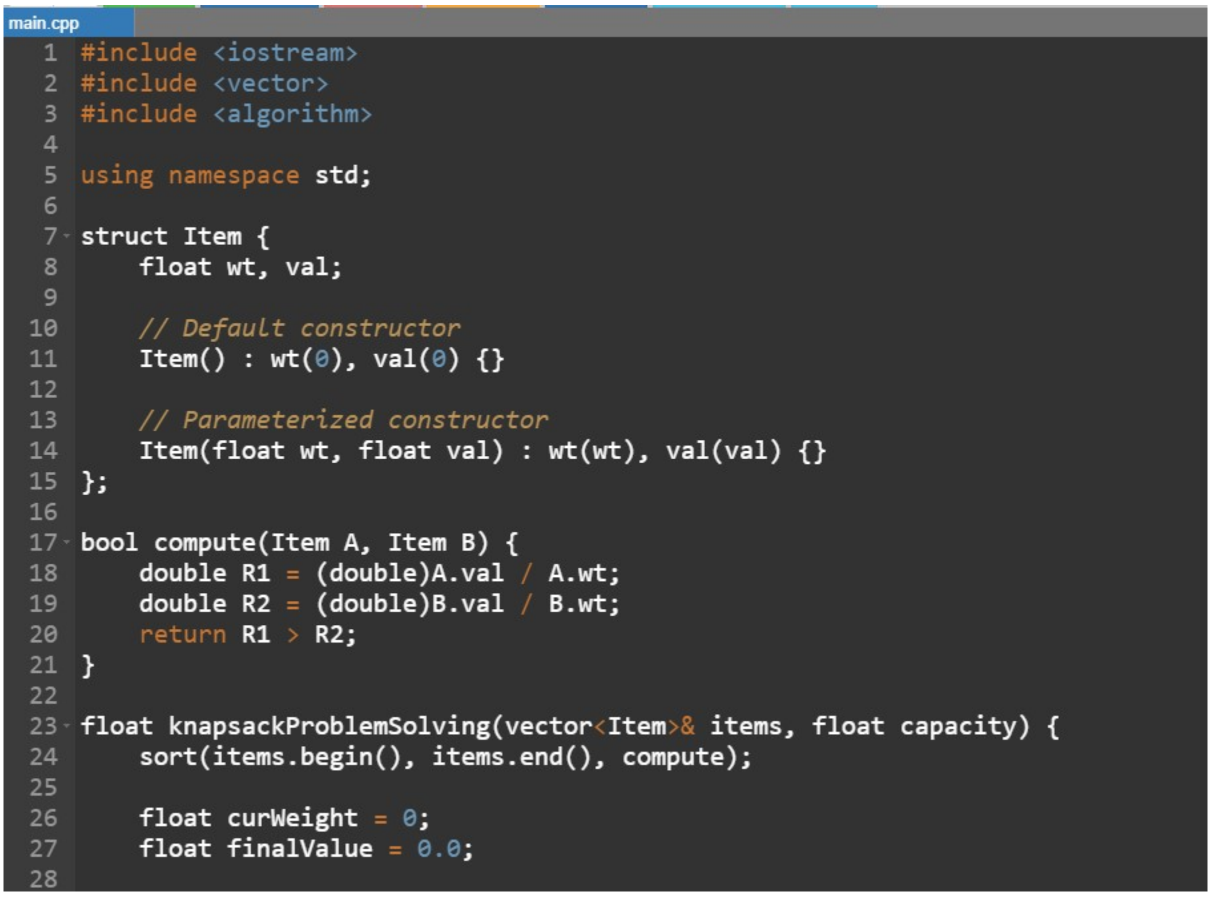
```
      //Also,
for w=0 to W
       B[0,w]=0
for i=1 to n
       B[i,0]=0
```

} Initialise first row & first col. to zero

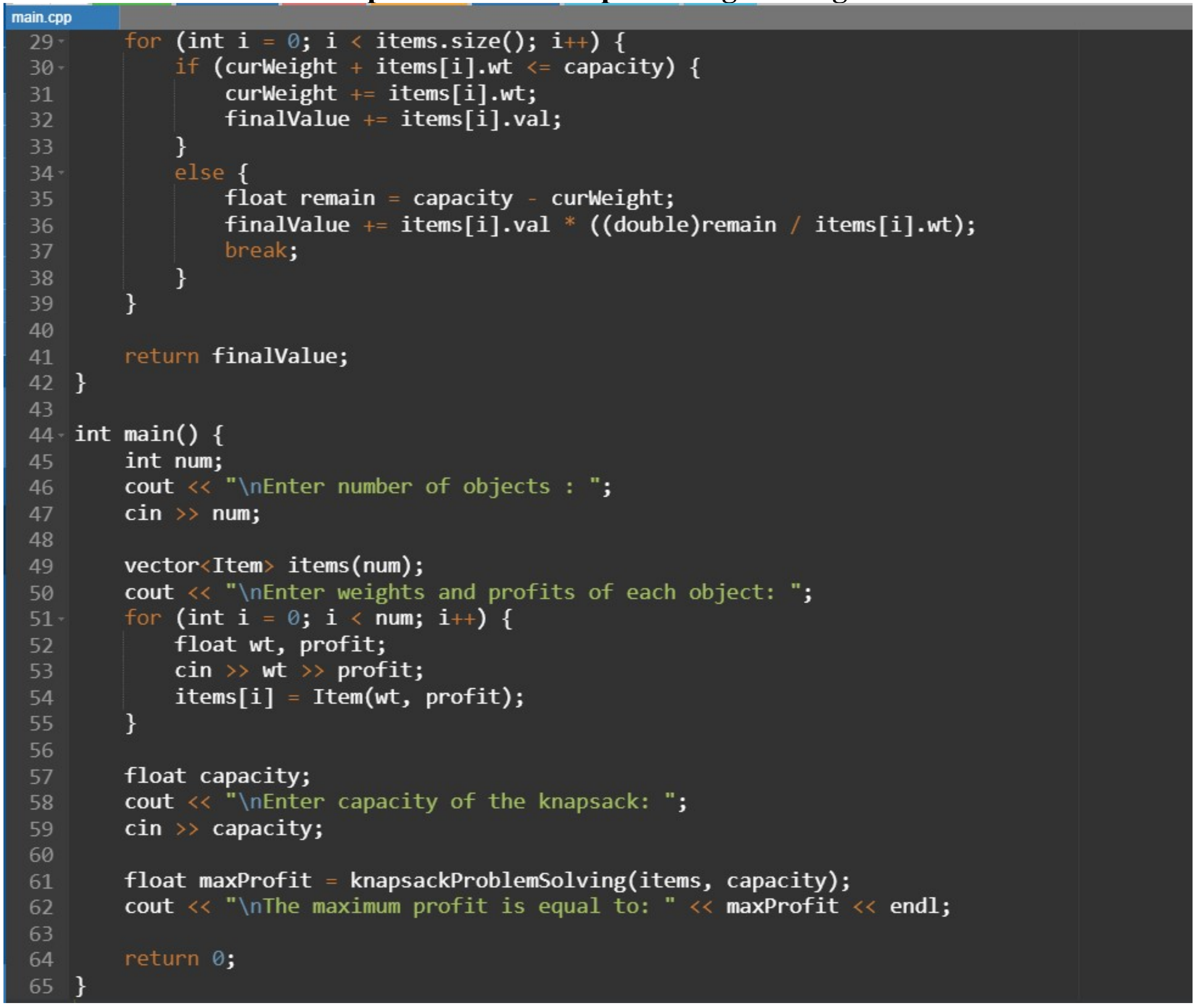
```
If  $w_i \le w$   //Item i can be in solution
    if  $v_i + B[i-1, w-w_i] > B[i-1, w]$
            $B[i,w] = v_i + B[i-1, w-w_i]$
    else
            $B[i,w] = B[i-1, w]$
    else  $B[i,w] = B[i-1, w]$   // $w_i > w$
```

∴ Time complexity : $T(n) = O(n \times w)$

CODE:


main.cpp

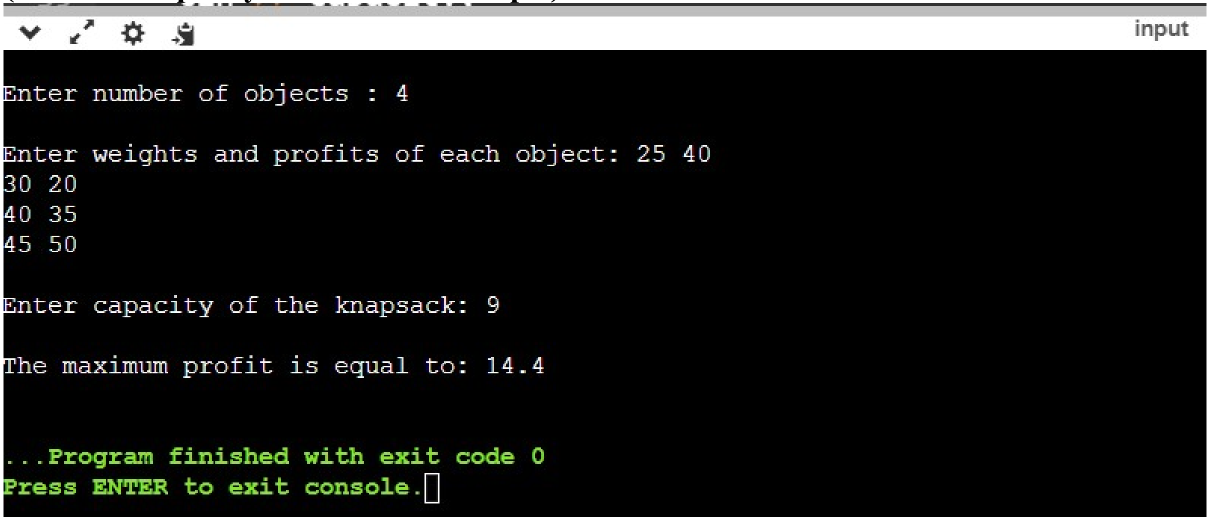
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Item {
    float wt, val;

    // Default constructor
    Item() : wt(0), val(0) {}

    // Parameterized constructor
    Item(float wt, float val) : wt(wt), val(val) {}
};

bool compute(Item A, Item B) {
    double R1 = (double)A.val / A.wt;
    double R2 = (double)B.val / B.wt;
    return R1 > R2;
}

float knapsackProblemSolving(vector<Item>& items, float capacity) {
    sort(items.begin(), items.end(), compute);

    float curWeight = 0;
    float finalValue = 0.0;

```



main.cpp

```cpp
    for (int i = 0; i < items.size(); i++) {
        if (curWeight + items[i].wt <= capacity) {
            curWeight += items[i].wt;
            finalValue += items[i].val;
        }
        else {
            float remain = capacity - curWeight;
            finalValue += items[i].val * ((double)remain / items[i].wt);
            break;
        }
    }

    return finalValue;
}

int main() {
    int num;
    cout << "\nEnter number of objects : ";
    cin >> num;

    vector<Item> items(num);
    cout << "\nEnter weights and profits of each object: ";
    for (int i = 0; i < num; i++) {
        float wt, profit;
        cin >> wt >> profit;
        items[i] = Item(wt, profit);
    }

    float capacity;
    cout << "\nEnter capacity of the knapsack: ";
    cin >> capacity;

    float maxProfit = knapsackProblemSolving(items, capacity);
    cout << "\nThe maximum profit is equal to: " << maxProfit << endl;

    return 0;
}
```

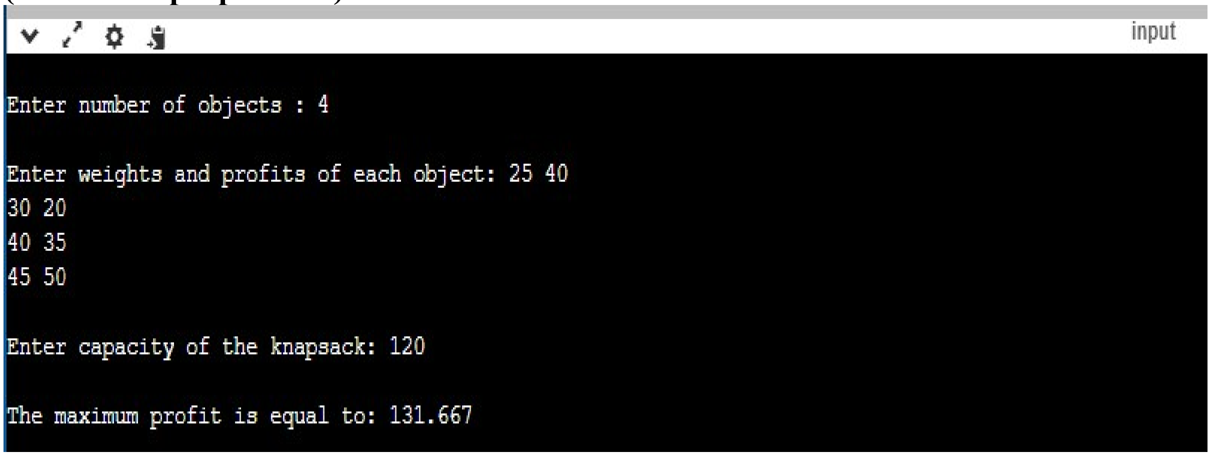
OUTPUT:
(Different capacity from solved example):



input

```
Enter number of objects : 4

Enter weights and profits of each object: 25 40
30 20
40 35
45 50

Enter capacity of the knapsack: 9

The maximum profit is equal to: 14.4


...Program finished with exit code 0
Press ENTER to exit console.
```

(Same example problem)



```
Enter number of objects : 4

Enter weights and profits of each object: 25 40
30 20
40 35
45 50

Enter capacity of the knapsack: 120

The maximum profit is equal to: 131.667
```

Conclusion:
Thus,in this experiment we learnt about another greedy algorithm,the knapsack problem(used 0/1 knapsack this time),solved an example problem of it in lab,derived its time complexity using the algorithm and finally implemented the algorithm using C++.