CS22203: Design and Analysis of Algorithms
Lab Assgn no. 6

The following is the recursive procedure for solving the O/1 Knapsack problem (as discussed in class).

A) Execute the procedure as Opt(n,M) for a problem instance of n objects and a knapsack capacity M. Try out for several problem instances including:
1) Solve the 0/1 Knapsack instance: n=3, $(p_1,p_2,p_3)=(1,2,5)$, $(w_1,w_2,w_3)=(2,3,4)$, M=6.
2) Solve the 0/1 Knapsack instance: n=4, $(p_1,p_2,p_3,p_4)=(10,5,20,30)$, $(w_1,w_2,w_3,w_4)=(3,2,3,4)$, M=9.

B) Modify the algorithm so that you also calculate number of times Opt( ) is called. This gives a measure of the size of the recursion tree, on which the order of magnitude of the algorithm depends.

C) Suppose, each call of Opt( ) denotes a node in the recursion tree. Can you also draw the tree using some Python module for visualization?

You can try out both the following algorithms, which give the same result.

DP Algorithm for solving O/1 Knapsack:

```
Opt1(i,X)
// y, p,w are arrays which store the solution, profits and weights respectively.
// y is initialized to 0. This procedure returns the value of the optimal solution for KNAP(1,i,X).

if X < 0 then //this condition must be checked first before the second condition
    return -α  //return a large negative value
else if i==0 then
    return 0
  else
    val1 = Opt(i-1, X)
    val2 = p[i]+Opt(i-1, X-w[i])
    if val2 > val1 then
        return val2
    else
        return val1
    endif
  endif
endif
```

OR

```
Opt2(i,X)
// y, p,w are arrays which store the solution, profits and weights respectively.
// y is initialized to 0. This procedure returns the value of the optimal solution for KNAP(1,i,X).
if i==0 then
```

```
    return 0
else
    if  X < w[i] then
        val1 = Opt(i-1, X);
        return val1
    else
        val1 = Opt(i-1, X)
        val2 = p[i]+Opt(i-1, X-w[i])
        if val2 > val1 then
            return val2
        else
            return val1
        endif
    endif
endif
-----------
```