Name: Md. Habibur Rahman Rony
Student ID: 984582
Weekday: Week 3- Day 11(b)

Answer to the Q. No. C:

Algorithm 01Knapsack(S, w)

```
for i <- 0 to S.size() do
    for j <- 0 to w + 1 do
        B[i, j] <-  -1
01Knapsack(S, k, w)
```

Algorithm 01Knapsack(S, k, w)

```
if B[k, w] >= 0
    then return B[k, w]
if k = 0 then
    B[k, w] <- 0
else if w<= 0 then
    B[k, w] <- 0
else
    (b(k), w(k)) <- S.elementAtRank(k)
    if w(k) > w
        B[k, w] <-  01Knapsack(S, k - 1, w)
    else
        B[k, w] <-  max(01Knapsack(S, k - 1, w),
                        01Knapsack(S, k - 1, w - w(k)) + b(k))
return B[k, w]
```

This algorithm has lower performance than the one given in the lectures, which is $O(n^2)$. Because we are filling the same matrix structure, but in one point max(x, y) we need 2 recursive call.

Answer to the Q. No. D:

Algorithm 01Knapsack(S, L)

```
Input: set S of n items with size $s_i$; upper limit L
Output: a set of largest sum of sizes that is no greater than L
for w <- 0 to L do
    B[w] <- 0
for k<-1 to n do
    copy array B into array A
    for w <- w(k) to L do
        if A[w - w(k)] + b(k) > A[w] then
            B[w] <- A[w - w(k)] + b(k)
return B[L]
```

The running time, T(n)= O(nL)

Answer to the Q. No. C-5.9:

We can modify the subset which would be filled as the program runs to count the maximum benefit. This solution will compute the assignment that will gives us the maximum benefit along with the items.