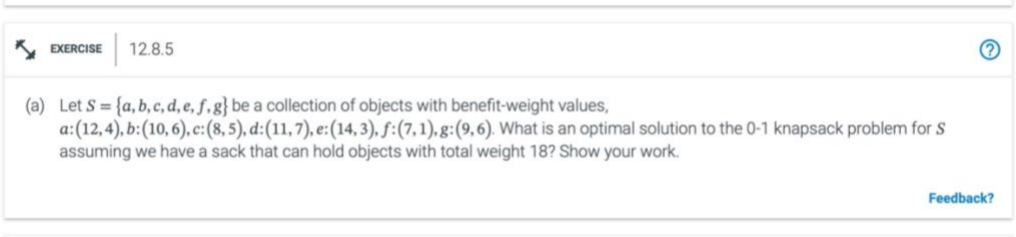
Jay Kalyanbhai Savani
CWID – 20009207
Assignment – 5.1


EXERCISE | 12.8.5

(a) Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit-weight values, $a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1), g:(9,6)$. What is an optimal solution to the 0-1 knapsack problem for S assuming we have a sack that can hold objects with total weight 18? Show your work.

Feedback?

ANS: Total Weight object can hold is 18.

Here,

a: (12, 4)
b: (10, 6)
c: (8, 5)
d: (11, 7)
e: (14, 3)
f: (7, 1)
g: (9, 6)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B, W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12, 4	1	0	0	0	0	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
10, 6	2	0	0	0	0	12	12	12	12	12	12	22	22	22	22	22	22	22	22	22
8, 5	3	0	0	0	0	12	12	12	12	12	20	22	22	22	22	22	30	30	30	30
11, 7	4	0	0	0	0	12	12	12	12	12	20	22	23	23	23	23	30	31	33	33
14, 3	5	0	0	0	14	14	14	14	26	26	26	26	26	34	36	37	37	37	37	44
7,1	6	0	0	0	14	14	14	14	26	26	26	26	26	34	36	37	37	37	37	44
9,6	7	0	0	0	14	14	14	14	26	26	26	26	26	34	36	37	37	37	37	44

As we fill up the values as per the formula:

$$B[k,w] = \begin{cases} B[k-1,w] & \text{if } w_k > w \\ \max\{B[k-1,w], B[k-1,w-w_k] + b_k\} & \text{else}. \end{cases}$$

We see that the maximum benefit is 44.
We will get the optimal solution from {(12,4), (10,6), (8,5), (14,3)}

EXERCISE | 12.8.14

(a) Show that we can solve the telescope scheduling problem in $O(n)$ time even if the list of n observation requests is not given to us in sorted order, provided that start and finish times are given as integer indices in the range from 1 to n^2.

Feedback?

ANS: Algorithm for telescope scheduling:

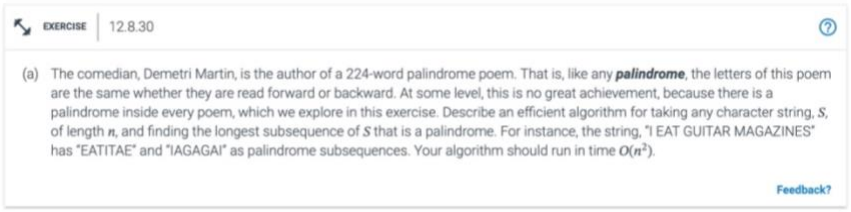
Ordering of requests by finish times and an array, P, so that P[i] = pred(i), then we can fill in the array, B

$B[0] \leftarrow 0$
for i = 1 to n do
$B[i] \leftarrow \max\{B[i-1], B[P[i]] + b_i\}$ A

It will take O(n log n) time to sort the list of n observations since it is not sorted, and O(n) time for the for loop in the method. As a result, this method will execute in O(n log n + n) total time.

We investigate the value O(n) omitting log n for high values of n. It is simple to sort n observations in O(n) time. Hence, the complete array B can be computed in O(n) time.

We use dynamic programming to identify the best solution for inputs 1 to n^2. Nevertheless, the brute force technique will perform better if the size changes by 2^n.


EXERCISE | 12.8.30

(a) The comedian, Demetri Martin, is the author of a 224-word palindrome poem. That is, like any ***palindrome***, the letters of this poem are the same whether they are read forward or backward. At some level, this is no great achievement, because there is a palindrome inside every poem, which we explore in this exercise. Describe an efficient algorithm for taking any character string, S, of length n, and finding the longest subsequence of S that is a palindrome. For instance, the string, "I EAT GUITAR MAGAZINES" has "EATITAE" and "IAGAGAI" as palindrome subsequences. Your algorithm should run in time $O(n^2)$.

Feedback?

ANS: Algorithm to find the longest subsequence of S that is a palindrome:

Input: Palindrome of string S of length [0 … n-1].
Output: The length L[i, j] of longest common sequence of a palindrome S[0 … n-1]

```
for i ← 1 to n do
        L[i, i] ← 1
for i ← 2 to n do
        L[i, i-1] ← 0
for i ← n-1 to 2 do
        for j ← i + 1 to n do
                if  L[i] = L[j] then
                        L[i, j] = L [i+1, j-1] + 2
                else
                        L[i, j] = max {L[i+1, j], L [i, j-1]}
return array L
```

Now, we verify the sequence's first and final characters first. As a result, we will know if two characters are the same or not.

If both characters match, we subtract both and add 2 to the result before adding the result to the array L.

If the two characters vary, we calculate the remaining subsequence while taking into account the first character to solve the issue. After that, we fix it by going back to the first step and selecting the last character.

The highest of the two outcomes will be recorded.

The method employs two for loops, one inner and one outer, both iterating n times. Within the loop, there is an assignment and an if statement that take up O(1) time.
Hence, the algorithm's overall running time is $O(n^2)$.