Homework 5,part 2

15.1-3)

15.1-3) As given in the book, there are two ways for solving rod-cutting problem:

1. Bottom-up-cut
2. Top-down cut.

Lets use Bottom-up cut for this case where we need to add an extra fixed cost for cutting the rod. Which should be subtracted from the prices ~~that we~~ of each ~~made to u~~ small rods that are cut.

Lets consider fixed cost to be c.

Given: to subtract c from prices of the pieces

(i.e) $p_{i_1}+p_{i_2}+\dots p_{i_k}$

$k \rightarrow$ no of pieces, $i \rightarrow$ inches of the rod.

this is nothing but r_n (revenue) of rod size n.

$r_n = p_{i_1}+p_{i_2}+\dots p_{i_k}$.

So subtracting the fixed cos c from price would be.

$$r_n = (p_{i_1} - c) + (p_{i_2} - c) + \cdots (p_{i_k} - c)$$
$$= (p_{i_1} + p_{i_2} + \cancel{p_{ik}} + \cdots p_k) - k(c)$$

k times the fixed cost as k is no. of pieces

this can be ~~the~~ called by

$\boxed{r_n - k(c)}$ when $p_{i_1} + p_{i_2} + \cdots p_k$ is r_n.

Lets use Bottom up approach.

MODIFIED-CUT ROD (p, n, c) → fixed cost

```
1. let r[0...n] be a new array
2. r[0] = 0
3. for j = 1 to n
4.     q = p[j]
5.     for i = 1 to j-1
6.         q = max(q, p[i] + r[j-i] - c)
7.     r[j] = q
8. return r[n].
```

In line number 6, we subtract the fixed cost from revenue, as discussed earlier it is similar to subtracting it from the pieces price

In line no. 4 & 5, we add q = p[j] ~~in case of~~ just to make sure that we dont subtract fixed costs for no cutting & run the for loop from i to j-1 instead of i [when i = j]

when we dont add that, we tend to subtract the fixed cost from no cutting

(i.e. q = max(q, p[i] - c)) → which is not valid

hence above is the modified algorithm with fixed cost

15.2-1)

15.2-1) Given: {5, 10, 3, 12, 5, 50, 6}

Let make a sequence with given dimension

$\overset{x_0}{\{5,10\}}$ $\overset{x_1}{\{10,3\}}$ $\overset{x_2}{\{3,12\}}$, $\overset{x_3}{\{12,5\}}$ $\overset{x_4}{\{5,50\}}$ $\overset{x_5}{\{50,6\}}$

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	0	150	27000	27000	20,25000	40,50,00
x_1		0	360	27000	1080,000	40,500,00
x_2			0	180	1,35,000	2,79000
x_3				0	3000	54,000
x_4					0	1500
x_5						0

1 | 5

$l = 2$

$x_0 x_1 = 150$

$x_1 x_2 = 360$

$x_2 x_3 = 180$

$x_3 x_4 = 3000$

$x_4 x_5 = 1500$

$l = 3$

$x_0 x_1 x_2 = 150 \times 5 \times 3 \times 12 = 27{,}000$ ✓

↳ optio

$x_0 x_1 x_2 = 360 \times 5 \times 10 \times 12 = 2{,}16{,}000$

$x_1 x_2 x_3 = 360 \times 12 \times 5 \times 10 = 2{,}16{,}000$

$x_1 x_2 x_3 = 180 \times 10 \times 3 \times 5 = 27{,}000$ ✓

$x_2 x_3 x_4 = 180 \times 5 \times 50 \times 3 = 1{,}35{,}000$ ✓

$x_2 x_3 x_4 = 3000 \times 3 \times 12 \times 50 = 5{,}400{,}00$

$x_3 x_4 x_5 = 3000 \times 50 \times 6 \times 12 = 10{,}800{,}000$

$x_3 x_4 x_5 = 1500 \times 12 \times 5 \times 6 = 5{,}40{,}000$ ✓

$l=4$

$\underbrace{x_0x_1}_{1}\underbrace{x_2x_3}_{1} = 150\times180 = 27{,}000$ ✓

$\underbrace{x_0x_1}_{1}\underbrace{x_2x_3}_{2} = 27000\times5\times12\times5$

$\underbrace{x_0}_{2}\underbrace{x_1x_2x_3}_{1} = 27000\times5\times10\times5$

} clearly these are high

$x_1x_2x_3x_4 = 360\times3000 = 1{,}080{,}000$ ✓

$x_1x_2x_3x_4 = 27000\times5\times450\times10$

$x_1x_2x_3x_4 = 135000\times10\times3\times50$

} clearly these are high.

$x_2x_3x_4x_5 = 180\times1500 = 270{,}000$ ✓

$x_2x_3x_4x_5 =$ } Will be high clearly from above.

$x_2x_3x_4x_5 =$ }

$l=5$

$\underbrace{\underbrace{x_0x_1}_{1}\underbrace{x_2x_3}_{1}}_{2}x_4 = 150\times180\times5\times5\times50 = 33{,}750{,}000$

$\underbrace{x_0\underbrace{x_1x_2}_{1}}_{2}\underbrace{x_3x_4}_{1} = 360\times3000\times5\times10\times50 = \text{high}$

$\underbrace{x_0x_1}_{1}\underbrace{x_2x_3x_4}_{2} = 27000\times3000 = 81{,}000{,}000$

$\underbrace{x_0 x_1 \underbrace{x_2 x_3}_{1}}_{2} x_4 = 150 \times 135000 = 20,250,000$

$x_1 x_2 x_3 x_4 x_5 = 360 \times 3000 \times 50 \times 6 \times 10 =$ high

$x_1 x_2 x_3 x_4 x_5 = 180 \times 3000 \times 10 \times 3 \times 6 = 97,200,$

$x_1 x_2 x_3 x_4 x_5 = 27000 \times 1500 = 40,500,000$ ✓

$x_1 x_2 x_3 x_4 x_5 = \cancel{135000 \times 150}\ 5,40,000 \times 360 = 194,400,000$

$x_1 x_2 x_3 x_4 x_5 =$

$l = 6$

$\underbrace{x_0 x_1}_{1} \underbrace{x_2 x_3}_{1} \underbrace{x_4 x_5}_{1} = 150 \times 180 \times 1500 = 40,500,000$

$\underbrace{\underbrace{x_0 x_1 x_2}_{1} x_3 x_4 x_5}_{1} = 27000 \times 5,40,000$
$= \text{high}$

$\underbrace{\underbrace{x_0 x_1 x_2 x_3}_{1} x_4 x_5}_{1} = 27000 \times 1500$ ✓
$= 40,500,000$

$\underbrace{x_0 x_1}_{1} \underbrace{x_2 x_3 x_4 x_5}_{1} = 150 \times 2,70,000$ ✓
$= 40,500,000$

$\underbrace{\underbrace{x_0 x_1 x_2 x_3}_{1} x_4 x_5}_{2} = 20,250,000 \times 50 \times 6 \times 3$

high

$x_0 x_1 x_2 x_3 x_4 x_5 = 5 \times 10 \times 6 \times 40,500,000$
$= \text{high}$

From above
we have 3 sequence which gives minimal value for matrix multiplication.

~~$x_0 x_1$~~

(ie) $(A_1 A_2)(A_3 A_4)(A_5 A_6)$

$(A_1 A_2 A_3 A_4)(A_5 A_6)$

$(A_1 A_2)(A_3 A_4 A_5 A_6)$

15.4-1)

5-4.1)

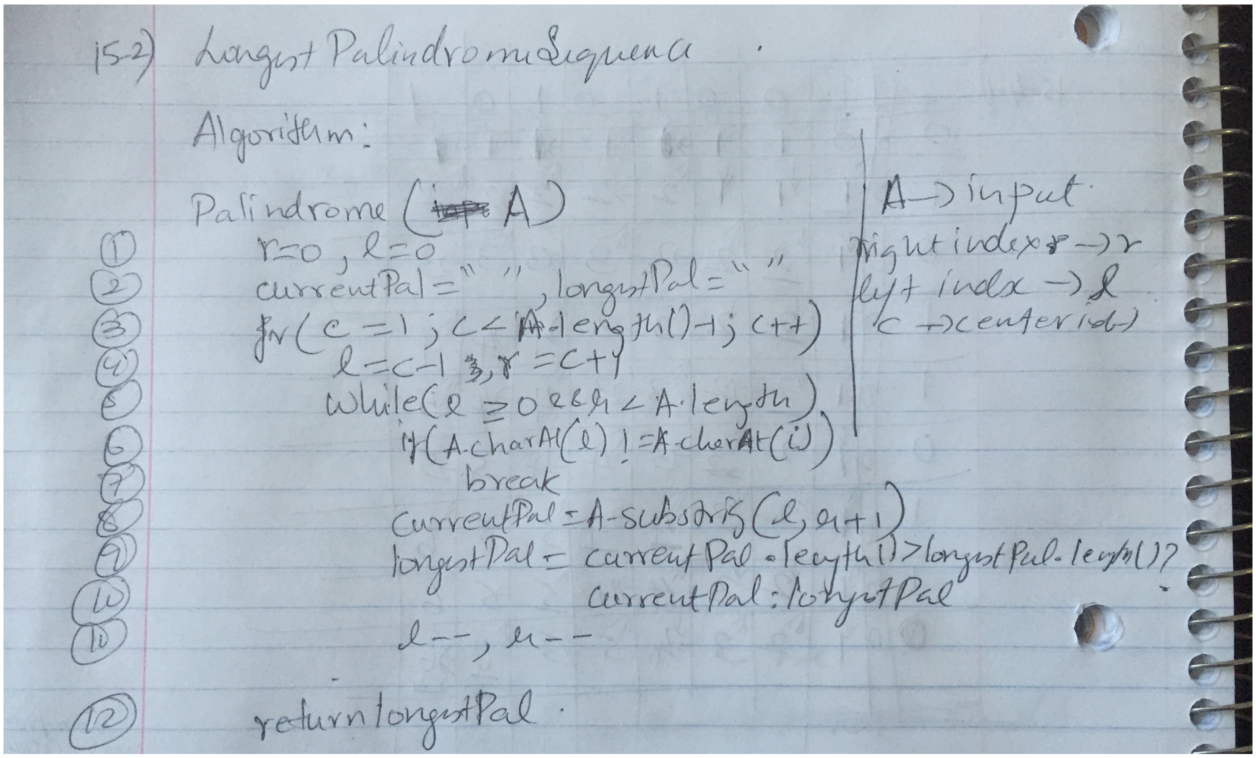
		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	0	↑0	↖1	↖1	←1	↖1	←1	↖1	←1
1	0	↖1	↑1	↑1	↖2	←2	↖2	←2	↖2
0	0	↑1	↖2	↖2	←2	↖3	←3	↑3	←3
1	0	↖1	↑2	↑2	↖3	←3	↖4	←4	↖4
1	0	↖1	↑2	←2	↖3	←3	↖4	←4	↖5
0	0	↑1	↖2	↖3	←3	↖4	←4	↖5	←5
1	0	↖1	↑2	↑3	↖4	←4	↖5	←5	↖6
1	0	↖1	↑2	↑3	↖4	←4	↖5	←5	↖6
0	0	↑1	↖2	↖3	↑4	↖5	←5	↖6	←6

size ~~of~~ longest common subsequence is 6.

& the LCS is.

{1 0 0 1 1 0}

15-2)



15-2) Longest Palindromic Sequence

Algorithm:

Palindrome (A)

1. r=0, l=0
2. currentPal=" ", longestPal=" "
3. for(c=1; c<A.length()-1; c++)
4. l=c-1 ; r=c+1
5. while(l ≥ 0 && r < A.length)
6. if(A.charAt(l) != A.charAt(r))
7. break
8. currentPal = A.substring(l, r+1)
9. longestPal = currentPal.length() > longestPal.length() ? currentPal : longestPal
10.
11. l--, r--
12. return longestPal.

A → input
right index → r
left index → l
c → center index

Code in Java:

```
public class Palindrome {
        public static String getfinalLongPalindrome(final String input) {
    int r = 0, l = 0;
    String currentPalindrome = "", finalLongPalindrome = "";
    for (int c = 1; c < input.length() - 1; c++) {
        l = c - 1;  r = c + 1;
        while (l >= 0 && r < input.length()) {
            if (input.charAt(l) != input.charAt(r)) {
                break;
            }
            currentPalindrome = input.substring(l, r + 1);
            finalLongPalindrome = currentPalindrome.length() > finalLongPalindrome.length() ?
currentPalindrome : finalLongPalindrome;
            l--;  r++;
```

```
            }
        }
        return finalLongPalindrome;
    }

    public static void main(String args[]) {
        String str = "cabacar";
        String finalLongPali = getfinalLongPalindrome(str);
        System.out.println("String: " + str);
        System.out.println("Longest Palindrome: " + finalLongPali);
    }
}
```

Output:

```
String: cabacar
Longest Palindrome: cabac
```

Running time would be O(n) since there is only one for loop.