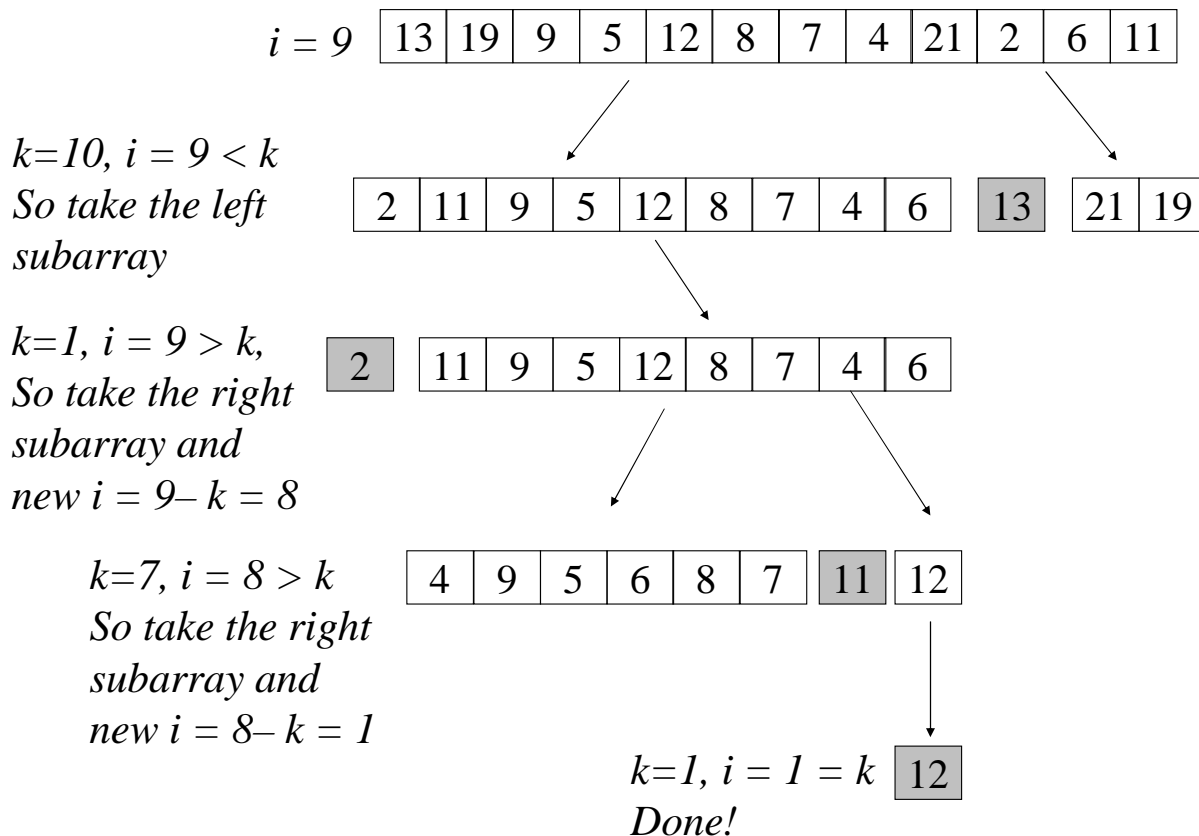


# CS 3343 (Spring 2008) Assignment 7

## Partial solution (Q1, Q2a-b).

1. (15 points) Order Statistics.

Study the pseudocode and example of Rand-Select on Slides #35-38 in Lecture 15. Use Slide #38 as a model, illustrate the operation of selecting the 9-th smallest element on array  $A = [13\ 19\ 9\ 5\ 12\ 8\ 7\ 4\ 21\ 2\ 6\ 11]$ . Similar to Slide #38, you can use ordinary Partition rather than Rand-Partition, i.e., you always select the first element as the pivot. Also you may re-use part of your results in Hw5 1(f) since the array is the same as the one we used there.



2. Longest common subsequence.

a. (15 points) Fill in the dynamic programming table below to compute the length of a longest common subsequence between ACGTACGA and TCGAACAG. Then add trace-back pointers to find out the actual longest common subsequence.

**Solution strategy:** First initialize the first row and first column to zeros. Then fill in the table with a row-by-row order using this recursive formula:

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j]; \\ \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise.} \end{cases}$$

This basically means that when we are trying to compute the table entry  $c[i, j]$ , we first check if  $x[i] = y[j]$ . If yes, we take the value in  $c[i-1, j-1]$  plus one and save it in  $c[i, j]$ . If not, we take the larger one from these two cells:  $c[i, j-1]$  and  $c[i-1, j]$ . To enable an easy traceback, you may choose

to remember for each cell where the value comes from (i.e., up-diagonal, left, or above). Alternatively, you can figure out this relationship later (as shown below). For example, when we need to figure out the edge that leads into the cell  $c[8, 8]$ , we check whether  $x[8] = y[8]$ . Since  $A \neq G$ , the value in  $c[8, 8]$  must be from either the cell to the left or the one above. In this case we have a tie (both  $c[7, 8]$  and  $c[8, 7]$  are equal to  $c[8, 8]$ ), which can be broken arbitrarily. The figure below shows both edges. For cell  $c[7, 6]$ , the only path that leads into it is from above, since  $G \neq C$ , and  $c[7, 5] \neq c[7, 6]$ . Once you have identified the complete path from  $c[0, 0]$  to  $c[m, n]$  ( $m = n = 8$  here), you can read off the LCS from all the diagonal edges on the path: an diagonal edge from  $c[i, j]$  to  $c[i - 1, j - 1]$  means that  $x[i]$  is matched to  $y[j]$  in the LCS. In the case above, if we take the red path entirely, the LCS is **CGACA**. If we take the blue edges as part of the path, the LCS is **CGACG**. Both have length 5.

		$j$	0	1	2	3	4	5	6	7	8
		$y[j]$	T	C	G	A	A	C	A	G	
$i$	$x[i]$										
0		0	0	0	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1	1	1	1
2	C	0	0	1	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	2	2	3
4	T	0	1	1	2	2	2	2	2	2	3
5	A	0	1	1	2	3	3	3	3	3	3
6	C	0	1	2	2	3	3	4	4	4	4
7	G	0	1	2	3	3	3	4	4	4	5
8	A	0	1	2	3	4	4	4	5	5	5

- b. (5 points) In the table you computed above, you should have observed that the values are non-decreasing when you move from the upper left corner to the bottom right corner. Can you give an intuitive explanation why this is expected? (Do not answer that the recurrence shows that.)

**Answer:** The table entry  $c[i, j]$  represents the length of the LCS between the prefixes  $x[1..i]$  and  $y[1..j]$ . When we increase the index  $i$  or  $j$ , the prefixes become longer. Since the LCS of two longer prefixes should have at least the same length as that of the shorter prefixes, we expect the values of  $c[i, j]$  to be non-decreasing when  $i$  or  $j$  increases.

- c. (10 points) A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence

ACGTGTCAAAATCG

has many palindromic subsequences, including ACGCA and CAAC (on the other hand, the subsequence ACG is not palindromic.) Design an efficient algorithm that takes a sequence  $x[1 \dots n]$  and returns the longest palindromic subsequence. Its running time should be in  $\Theta(n^2)$ .

**Answer:** First let me say that this problem is harder than originally I thought. The hint I gave actually does not lead to a completely correct solution.

Let  $S$  be a longest palindromic subsequence (LPS) of  $X$ . Then, we note that (1)  $S$  is a subsequence of  $X$ , and (2)  $S$  is also a subsequence of the reverse of  $X$ . That is,  $S$  must be a common subsequence of  $X$  and  $\text{reverse}(X)$ . Now, you may jump to the conclusion that  $\text{LCS}(X, \text{reverse}(X))$  is the LPS of  $X$  (which is what my hint was based on). Well, not quite true. First, we cannot say immediately that  $S$  is the LONGEST common subsequence of  $X$  and  $\text{reverse}(X)$  (this turns out to be true, though, but to prove this is not trivial). Second,  $X$  and  $\text{reverse}(X)$  may have several LCSs. Are they all palindromic? The answer is no. For example, if  $X$  is TACTA, then  $\text{LCS}(\text{TACTA}, \text{ATCAT})$  can be any of the four strings: ACA, TCT, TCA, ACT, only two of which are palindromic. Therefore, it is fair to say that we can find the length of the LPS using  $\text{LCS}(X, \text{reverse}(X))$ , but not the actual LPS. The actual LPS can be found by some tricks during trace-back (using two pointers from both ends simultaneously).

Lesson learned: never assume anything without a proof.

It is actually easier to develop a DP algorithm for this problem directly. Let  $\text{LPS}(i, j)$  be the length of the longest palindromic subsequence for the substring  $X[i..j]$ . The recursion to compute  $\text{LPS}(i, j)$  is as follows:

$$\text{LPS}(i, j) = \begin{cases} 1 & \text{if } i = j; \\ \text{LPS}(i + 1, j - 1) + 1 & \text{if } x[i] = x[j] \text{ and } j > i; \\ \max\{\text{LPS}(i + 1, j), \text{LPS}(i, j - 1)\} & \text{if } x[i] \neq x[j] \text{ and } j > i \end{cases}$$

When computing the dynamic programming table, first initialize all diagonal entries to 1. Then work on the upper triangle towards the upper-right corner. The length of the LPS is stored in the entry  $\text{LPS}(1, n)$ .

3. (25 points) Dynamic programming.

You are given a checkboard which has 3 rows and  $n$  columns, and has an integer written in each square. You are allowed to select a number from each column so as to maximize the sum of the selected numbers. There is one constraint: you cannot select two numbers from horizontally adjacent squares (diagonal adjacency is fine). For example, given the checkboard below, you can select 6, 7, 3, 8 or 5, 7, 9, 7, among other possibilities. 6, 7, 9, 10 is not a legal selection, since the last two numbers were selected from two adjacent squares. The optimal selection is 6, 9, 5, 10 (or 6, 7, 9, 8), which sums to 30.

6	5	3	7
4	7	5	8
5	9	9	10

- a. A natural greedy algorithm is to always select the square with the largest number, so long as it is not adjacent to the square selected in the preceding column.

```
sum = 0;
previousSelection = 0;
for i = 1 to n
    max = -infinity;
    currentSelection = 0;
```

```

for j = 1 to 3
  if (j != previousSelection && checkboard[j, i] > max) {
    max = checkboard[j, i];
    currentSelection = j;
  }
end
previousSelection = currentSelection;
sum = sum + max;
end

```

Show that this algorithm does not correctly solve this problem, by giving an example of a  $3 \times 4$  checkboard on which the algorithm does not return the correct answer.

**Answer:** In the example example, greedily taking 6 in the first column will force you to choose a suboptimal selection for all the other columns.

6	9	3	7
4	7	9	8
5	5	5	10

- b. Give an efficient dynamic programming algorithm for computing the optimal selection. Its running time should be  $\Theta(n)$ . (Hint: Let  $F(j, i)$  represents the sum of the optimal selection for the first  $i$  columns, with the condition that you select the  $j$ -th square in the  $i$ -th column. Which  $F(j, i)$  would give you the optimal solution for the complete checkboard? How would you compute  $F(j, i)$  recursively?).

**Solution:**

$$F(j, i) = \text{checkboard}[j, i] + \max_{k \neq j} (F(k, i - 1))$$

The final optimal solution is given by  $\max_k (F(k, n))$ .

- c. Use your algorithm to solve the following checkboard:

0	4	1	9	4	6	8	9	0	7
7	4	1	3	9	7	5	2	3	2
5	6	4	3	4	5	6	9	3	6

The dynamic programming table is as follows.

0	<b>11</b>	14	<b>24</b>	22	39	<b>46</b>	54	55	<b>65</b>
<b>7</b>	9	14	18	<b>33</b>	35	44	48	<b>58</b>	59
5	13	<b>15</b>	17	28	<b>38</b>	45	<b>55</b>	57	64

The optimal solution 65 can be obtained by selecting 7, 4, 4, 9, 9, 5, 8, 8, 3, and 7.