CS302 - Problem Set 8

1. `Selection`(A, k) finds the k^{th} smallest element of an array A. For example, if $A = [3, 11, 20, 6, 7, 38]$ and $k = 3$, then `Selection`(A, k) returns 7. Here is pseudocode for a divide and conquer algorithm for `Selection`. Note we index A starting at 1, not 0.

Algorithm 1: `Selection`(A, k)

Input : An array A of n integers indexed starting at 1 and an integer $k \in \{1, \ldots, n\}$

Output: The kth smallest element of A.

```
1  ChoosePivot(A);
2  Run Partition on elements of A using chosen pivot;
3  Let p be the index of the pivot after Partition;
4  if p = k then
5  |  return A[p];
6  end
7  if p < k then
   |  // Everything larger than the pivot, including the kth
   |     element, is to the right of p
8  |  return Selection(A[p + 1 : n], k − p);
9  else
   |  // Everything smaller than the pivot, including the kth
   |     element, is to the left of p
10 |  return Selection(A[1 : p − 1], k);
11 end
```

   (a) Before analyzing the divide and conquer algorithm for Selection, think about the "brute force" approach. Describe in words or using pseudocode an algorithm that solves this problem using as simple a method as possible, and give a big-O bound on the runtime if the input array has size n.

   (b) What is a big-O bound on the runtime of Algorithm 1 (the divide and conquer approach) if `ChoosePivot` somehow always chooses the pivot to be the median value of the array, and the input array has size n.

   (c) What is a big-O bound on the runtime of Algorithm 1 (the divide and conquer approach) if `ChoosePivot` somehow always chooses the pivot to be the smallest element the array, and the input array has size n.

(d) Suppose `ChoosePivot` chooses a pivot uniformly at randomly from the elements of A. [For a challenge, without turning the page, come up with an expression (that might involve some unsimplified summations) for the number of comparisons that are made over the course of the algorithm. For an extra challenge, bound and simplify the summation terms. Otherwise, turn the page for more guidance.]

i. Please describe in words the sample space of this problem.

ii. Let B be the original array that was input to `Selection`. As in class, let z_i be the i^{th} smallest element of B. Let X_{ij}, for $i < j$, be a random variable that is the number of times z_i and z_j are compared over the course of the algorithm. Explain why X_{ij} is an indicator random variable.

iii. What is the probability of z_i and z_j being compared if we are trying to find the kth smallest element, and $k \leq i$?

iv. What is the probability of z_i and z_j being compared if we are trying to find the kth smallest element, and $j \leq k$?

v. What is the probability of z_i and z_j being compared if we are trying to find the kth smallest element, and $i < k < j$?

vi. Use linearity of expectation, and properties of the expectation value of indicator random variables to create an explicit expression involving sums that gives the average number of comparisons done over the course of the algorithm. (See hint on last page for a key trick.)

vii. (Challenge) Analyze the sums from the previous part to get a runtime of $O(n)$, which should be better than your brute force approach!

2. You run a plant that produces sheets of aluminum alloy, and then you cut them to size for customers. Your machine produces rectangular sheets of dimension $Q \times R$, and you can cut any sheet into two smaller sheets by making a vertical or horizontal cut. So for example, if you have a 2×4 sized piece, you have the following options that you could produce from a single cut:

   - Two 1×4 pieces.
   - Two 2×2 pieces.
   - A 1×2 and a 2×3 piece.

   Say you decide to make the cut that produces a 1×2 and a 2×3 piece. You could then cut the 1×2 piece into two 1×1 pieces, and the 2×3 piece into a 2×1 piece and a 2×2 pieces, and so on.

   You can also rotate pieces (so a 2×3 piece is the same as a 3×2 piece).

   Suppose you produce n different sized products, where product i has dimensions $x_i \times y_i$, and you can sell product i for v_i dollars. (For example, perhaps product 1 is a 2×2 piece that you can sell for \$2, and product 2 is a 3×5 piece that you can sell for \$3 dollars.) You can sell multiple copies of the ith product if you can produce multiple pieces of that size from your original sheet. Assume Q, R, x_i, y_i, and v_i for $i \in \{1, 2, \ldots, n\}$ are positive integers. You will design an algorithm that figures out your maximum profit among any possible sequence of horizontal/vertical cuts.

   (a) Please provide pseudocode for a dynamic programming algorithm that outputs your maximum profit. (Note: your code doesn't have to output how you should

actually divide the piece, just the profit.) The input to your algorithm should be (Q, R, x, y, z) where Q and R are the size of the piece, array x contains the values x_i, array y contains the values y_i, and array v contains the values v_i.

(b) Explain why your algorithm is correct. (You don't need to write a full proof, but you should describe the logic behind the recurrences that you developed to create your algorithm.)

(c) What is the runtime of your algorithm in terms of Q, R, and n?

(d) Explain briefly how you would modify your algorithm to tell you whether you should divide the current sheet, and if so, where you should make the cut.

3. Approximately how long did you spend on this assignment (round to the nearest hour)?

Hints!

1a. The first step of your algorithm should be to sort the array!

1b. Use the master method! Be careful about the number of recursive calls...

1c. Go through a couple levels of recursion and try to figure out the pattern. (Use the iterative method of solving recurrence relations.)

1diii. In class, we were concerned with the set $\{z_i, z_{i+1}, \ldots, z_j\}$. Now we are interested in a different Which is the relevant set? From z_1 to z_i? From z_k to z_j? From z_k to z_i? From z_k to z_n? Think about when the "decision" point occurs that determines whether z_i and z_j are compared or not, and what range of values are involved.

1dvi. After linearity of expectation, you should get a term like:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}] \tag{1}$$

to add all terms with $i < j$.

We can divide this sum into parts that correspond to the three cases of the earlier parts of the problem, where case iv is the first sum, case v is the second sum, and case iii is the last sum:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}] = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \mathbb{E}[X_{ij}] + \sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \mathbb{E}[X_{ij}] + \sum_{i=k}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]. \tag{2}$$

2a. Usually we only have a couple options to check, but in this case we have lots of options to check. So you'll need a for loop to go through all of the possible options.

2c. In this case, we don't know what is the biggest term, Q, R, or n. So if you have a term like $O(Q + R + n)$ we can't simplify any further except to rewrite this as $O(\max\{Q, R, n\})$.)