Problem Set 2

2020-2021, 1st Semester **Due:** 10-28, 23:59

Please answer each of the following problems.

Note: For all problems, if you include pseudocode in your solution, please also include a brief description of what the pseudocode does.

1. **Different-sized sub-problems.** (6 points) Solve the following recurrence relation.

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n,$$

where $T(1) = 1$. **[We are expecting a formal proof. For example, what is the value range of the parameters in your assumption? You may state your final running time with $O(\cdot)$ notation, but do not use it in your proof. Pay attention to $T(1) = 1$, which should be satisfied in your results.]**

2. **What's wrong with this proof?** (8 points) Consider the following recurrence relation:

$$T(n) = T(n-5) + 10 \cdot n$$

for $n \geq 5$, where $T(0) = T(1) = T(2) = T(3) = T(4) = 1$. Consider the following three arguments.

(1) **Claim: $T(n) = O(n)$.** To see this, we will use strong induction. The inductive hypothesis is that $T(k) = O(k)$ for all $5 \leq k < n$. For the base case, we see $T(5) = T(0) + 10 \cdot 5 = 51 = O(1)$. For the inductive step, assume that the inductive hypothesis holds for all $k < n$. Then

$$T(n) = T(n-5) + 10n,$$

and by induction $T(n-5) = O(n-5)$, so

$$T(n) = O(n-5) + 10n = O(n).$$

This establishes the inductive hypothesis for n. Finally, we conclude that $T(n) = O(n)$ for all n.

(2) **Claim: $T(n) = O(n)$.** To see this, we will use the Master Method. We have $T(n) = a \cdot T(n/b) + O(n^d)$, for $a = d = 1$ and

$$b = \frac{1}{1 - 5/n}.$$

Then we have that $a < b^d$ (since $1 < 1/(1-5/n)$ for all $n > 0$), and the master theorem says that this takes time $O(n^d) = O(n)$.

(3) **Claim: $T(n) = O(n^2)$.** Imagine the recursion tree for this problem. (Notice that it's not really a "tree," since the degree is 1). At the top level we have a single problem of size n. At the second level we have a single problem of size $n-5$. At the t'th level we have a single problem of size $n - 5t$, and this continues for at most $t = \lfloor n/5 \rfloor + 1$ levels. At the t'th level for $t \leq \lfloor n/5 \rfloor$,

the amount of work done is $10(n-5t)$. At the last level the amount of work is at most 1. Thus the total amount of work done is at most

$$1 + \sum_{t=0}^{\lfloor n/5 \rfloor} 10(n-5t) = O(n^2).$$

(a) (3 points) Which, if any, of these arguments are correct? **[We are expecting a single sentence stating which are correct or all are wrong.]**

(b) (5 points) For each argument that you said was incorrect, explain why it is incorrect. If you said that all three were incorrect, then give a correct argument. **[We are expecting a few sentences of detailed reasoning for each incorrect algorithm; and if you give your own proof we are expecting something with the level of detail of the proofs above—except it should be correct!]**

3. Suppose that p is an unknown value, $0 < p < 1$. Suppose that you can call a function `randP` which returns `true` with probability p and returns `false` with probability $1-p$. Every call to `randP` is independent. You have no way to generate random numbers except through `randP`.

(a) (2 pts) Describe an algorithm—using `randP`—that returns `true` with probability 1/2 and `false` with probability 1/2. Your algorithm should in expectation, use $\frac{1}{p(1-p)}$ calls to `randP` (in other words, run $\frac{1}{p(1-p)}$ times of function `randP` in expectation). Your algorithm **does not** have access to the value of p, and **does not** have access to any source of randomness other than calls to `randP`. **[We are expecting pseudocode, and a short description of what the algorithm does.]**

Hints: (i) Your algorithm does not have to compute p, or an approximation to it. (ii) Notice that in the worst case, your algorithm may use more calls to `randP`, possibly even infinitely many.

(b) (1pts) Formally prove that your algorithm runs using expected $\frac{1}{p(1-p)}$ calls to `randP`. **[We are expecting a mathematical calculation of the expected value of the total number of calls to `randP`.]**

(c) (1 pt) Informally argue that your algorithm returns `true` with probability 1/2 and `false` with probability 1/2. **[We are expecting an informal justification of why the algorithm returns `true` with probability 1/2 and `false` with probability 1/2. Your argument should convince the reader that you are correct, but does not have to be a formal proof.]**

4. Suppose that A and B are sorted arrays of length n in ascending order, and that all numbers in the arrays are distinct.

(a) (3pts) Design an algorithm to find the median of all $2n$ numbers in $O(\log n)$ time. For our purposes, we define the median of the $2n$ numbers as the nth smallest number in the $2n$ values. **[We are expecting: pseudocode, and a description of the algorithm.]**

(b) (3pts) Informally argue that your algorithm correctly finds the median of all $2n$ numbers. **[We are expecting a short (paragraph or two) argument that will convince the reader why your algorithm works correctly.]**

(c) (2pts) Prove that your algorithm runs in time $O(\log(n))$ time. **[We are expecting a formal proof.]**

5. Suppose you want to sort an array A of n numbers (not necessarily distinct), and you are guaranteed that all the numbers in the array are in the set $\{1, \ldots, k\}$. A **"20-question sorting algorithm"** is any deterministic algorithm that asks a series of YES/NO questions (not necessarily 20 of them, that's just a name) about A, and then *writes down* the elements of A in sorted order. (Specifically, the algorithm does not need to rearrange the elements of A, it can just write down the sorted numbers in a separate location).

   Note that there are many YES/NO questions beyond just comparison-questions—for example, the following are also valid YES/NO questions: "If I ignored A[3] and A[17] would the array be sorted?" and "Did it rain today?"

   (a) (2 pts) Describe a 20-question sorting algorithm that will, for every input, ask only $O(k \log n)$ questions. Feel free to assume that the algorithm is also told n and k, although this isn't necessary. [Hint: If you are stuck, first think about how you would do this with $\log n$ questions if $k = 2$. What would you need to know about the array to write down the sorted list of elements?] **[We are expecting a description of the algorithm and an informal (1-paragraph) argument that it achieves the desired runtime.]**

   (b) (2 pts) Prove that for *every* 20-question sorting algorithm, there is some array A consisting of n integers between 1 and k that will require $\Omega(k \log \frac{n}{k})$ questions, assuming $k \leq n$. [Hints: Why is it sufficient for this problem to lower-bound the number of ordered arrays, instead of counting exactly? Once you have understood this, use a counting argument: how can you lower-bound the number of ordered arrays are there that consist of n integers $\{1, \ldots, k\}$ (not necessarily distinct)? There are a number of ways to do this; we suggest you do NOT use Stirling's approximation: you don't need this in order to prove the result, and it will be complicated.] **[We are expecting a mathematically rigorous proof (which does NOT necessarily mean something long and tedious).]**

6. Suppose that on your computer you have stored n password-protected files, each with a unique password. You've written down all of these n passwords, but you do not know which password unlocks which file. You've put these files into an array F and their passwords into an array P in an arbitrary order (so $P[i]$ does not necessarily unlock $F[i]$). If you test password $P[i]$ on file $F[j]$, one of three things will happen:

   1) $P[i]$ unlocks $F[j]$

   2) The computer tells you that $P[i]$ is lexicographically smaller than $F[j]$'s true password

   3) The computer tells you that $P[i]$ is lexicographically greater than $F[j]$'s true password

You **cannot** test whether a password is lexicographically smaller or greater than another password, and you **cannot** test whether a file's password is lexicographically smaller or greater than another file's password.

(a) (3pts) Design an randomized algorithm to match each file to its password, which runs in expected runtime $O(n \log(n))$. [**We are expecting: pseudocode and a description of the algorithm.**]

(b) (2pts) Explain why your algorithm is correct. [**We are expecting: an informal argument (a paragraph or so) about why your algorithm is correct, which is enough to convince the reader/grader. You may also submit a formal proof if you prefer.**]

(c) (2pts) Analyze the running time of your algorithm, and show that it runs in expected runtime $O(n \log(n))$. [**We are expecting: a formal analysis of the runtime.**]

7. (6pts, 1 pt per part) In the `select` algorithm from class, in order to find a pivot, we break up our array into blocks of length 5. Why 5? In this question, we explore `select-3`, in which we break up our array into blocks of length 3. In addition, to simplify your logic, you should assume throughout this problem that your array is a power of 3.

(a) Consider the pseudocode for `select` and `choosePivot` in the slides of lecture 4. In this pseudocode, we break up our array into blocks of size 5. What change(s) would you need to make to this pseudocode in order to write `select-3` and `choosePivot-3`? [**We are expecting a one or two sentence description of changes made.**]

(b) Prove that the recursive call to `select-3` inside of `choosePivot-3` is on an array of size at most $n/3$. You should assume that your array is a power of 3. [**We are expecting a rigorous proof.**]

(c) Prove that the recursive call inside of `select-3` is on an array of size at most $2n/3 + 2$. You should again assume that your array is a power of 3. (Hint: it might be helpful to note that $\lceil x \rceil \leq x + 1$.) [**We are expecting a rigorous proof.**]

(d) Explain why the work done within a single call to `select-3` and `choosePivot-3` on an array of size n is $\Theta(n)$. [**We are expecting a few sentences of explanation. You do *not* need to do a formal proof with the definition of Θ.**]

(e) Using what you proved in (b), (c), and (d), write down a recurrence relation for the runtime of `select-3`. (You can do this problem even if you did not complete all of (b), (c), and (d).) [**We are expecting a one-line answer with your recurrence relation.**]

(f) Is `select-3` $O(n)$? Justify your answer. [**We are not expecting a formal proof, but you should describe clear reasoning, such as analyzing a tree, unraveling the recurrence relation to get a summation, or attempting the substitution method. (Note that succeeding at the substitution method would prove select-3 is O(n), but failing at the substitution method does not prove select-3 is not O(n). 代入法成功可以证明 select-3 的复杂度是 O(n) 的，但是代入法不成功不能说明 select-3 的复杂度不是 O(n)。)**]