Name: [YOUR NAME HERE]

Problems

You may talk with your fellow CS161-ers about the problems. However:

- Read our Collaboration Policy on the course website before beginning.
- Try the problems on your own before collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

1. (2 pt.) Consider the recurrence relation $T(n) = T(n-10) + n$ for $n > 10$, with $T(n) = 1$ for all $n \leq 10$. Your friend claims that $T(n) = O(n)$, and offers the following justification:

Let’s use the Master Theorem with $a = 1$, $b = \frac{n}{n-10}$, and $d = 1$. This applies since, for any $n > 10$, we have

$$\frac{n}{b} = n \cdot \left(\frac{n-10}{n}\right) = n - 10.$$  

Then for any $n > 10$, we have $a < b^d$, so the Master Theorem says that $T(n) = O(n^d) = O(n)$.

What’s wrong with your friend’s argument, and what is the correct answer (i.e. the correct $\Theta(\cdot)$ bound, or the tightest $O(\cdot)$ bound you can come up with)? If it helps, you may assume that $n$ is a multiple of 10.

[We are expecting: A clear identification of the faulty logic above (i.e. a specific flaw in your friend’s reasoning, not just that their ultimate answer is incorrect); your solution to this recurrence (using asymptotic notation\(^1\)), and a short but convincing justification (no formal proof needed).]

---

\(^1\)Unless specified otherwise, in every concept check/problem set/quiz, when we ask for an answer in asymptotic notation, we are asking either for a $\Theta(\cdot)$ result, or else the tightest $O(\cdot)$ result you can come up with.
2. (12 pt.) [Batch Statistics] In this problem, we want you to design an algorithm which can retrieve “batch statistics” about some input data. In particular, imagine that you are given two things:

1) An array $A$ consisting of $n$ distinct integers, and
2) An array $R$ that contains $k$ integers $r_1, \ldots, r_k$, which are within the range $\{1, \ldots, n\}$. (You may assume that $k < n$.)

For each $j$ in $\{1, \ldots, k\}$, you should compute the $r_j$-th smallest of the $n$ elements in $A$, and output these $j$ numbers as an array. In other words, your output array should contain exactly the $r$-th smallest integer of $A$ for every $r$ in $R$. Here’s an example:

- Input: $A = [13, 19, 11, 14, 16, 18, 17, 12, 15]$; $R = [3, 7]$
- Output: $[13, 17]$
- Explanation: 13 is the 3-rd smallest of $A$ and 17 is the 7-th smallest element of $A$. $[17, 13]$ is also an acceptable output.

In this problem, if we ask for an $O(f(n))$-time algorithm, we want $O(f(n))$ to be the best (i.e. tightest) Big-O bound for the worst-case runtime of your algorithm. For example, if we ask for an $O(n \log n)$-time algorithm, presenting an algorithm that can also be described as $O(n)$ would not be what we’re looking for.

If it is helpful, you may assume that the size of each part of the input is a perfect power of 2.

(a) (3 pt.) Design an $O(n \log n)$ algorithm that takes as input $A$ and $R$ and outputs the corresponding batch statistics as described above.

[We are expecting: A clear English description of what your algorithm is doing (you may add pseudocode if it makes your algorithm more clear), and a justification of why the runtime is $O(n \log n)$.]

(b) (3 pt.) Design an $O(nk)$ algorithm that takes as input $A$ and $R$ and outputs the corresponding batch statistics as described above.

[We are expecting: A clear English description of what your algorithm is doing (you may add pseudocode if it makes your algorithm more clear), and a justification of why the runtime is $O(nk)$.]

(c) (6 pt.) Design a $O(n \log k)$ algorithm that takes as input $A$ and $R$ and outputs the corresponding batch statistics as described above.

[HINT: Use divide and conquer!]

[We are expecting: Pseudocode AND a clear English description of what your algorithm is doing, as well as a justification of why the runtime is $O(n \log k)$.]
3. (12 pt.) [Majority Genre] Suppose you’ve stumbled upon your friend’s massive book collection, which contains \( n \) books of many different genres. You’re not familiar with these books, so without help, you’re unable to figure out what the genre of each book is. Your friend assures you that one genre is in the majority. That is, there are strictly greater than \( n/2 \) books of that genre.

You’d like to borrow a book which belongs to this majority genre, and you decide to design a deterministic algorithm to find a book to pick. Your friend is a bit busy at the moment, but she is still willing to tell you whether two books are of the same genre or not. More specifically, she can answer queries of the form:

\[
\text{isTheSame( book1, book2 )} = \begin{cases} 
\text{True} & \text{if book1 and book2 share the same genre} \\
\text{False} & \text{if book1 and book2 have different genres}
\end{cases}
\]

The only way you can get any information about the books is by running \text{isTheSame}, which you may assume is a deterministic subroutine that is provided to you. The books are in no way labeled by genre, and you cannot compare them in any other way.

Remember, your friend assured you that one genre is in the majority. That is, there are strictly greater than \( n/2 \) books of that genre. Your goal is to return a single book with that majority genre. For example, if the collection looked like this\(^2\):

1) “The Abandoned House in the Woods” (Horror)
2) “The Invisible Stone” (Fantasy)
3) “Really Really Scary Stuff” (Horror)
4) “The Puppet Master” (Horror)
5) “Will I Ever Love Again” (Romance)

then the Horror genre is in the majority, and your algorithm should return any one of books 1, 3, or 4.

(a) (6 pt.) Design a deterministic divide-and-conquer algorithm which uses \( O(n \log n) \) calls to \text{isTheSame} and returns a book belonging to the majority genre. You may assume that \( n \), the number of books in the collection, is a power of 2 if it is helpful.

[We are expecting: Pseudocode (which calls \text{isTheSame}) AND a clear English description of what your algorithm is doing. For full credit, your algorithm must utilize divide-and-conquer.]

(b) (2 pt.) Explain why your algorithm calls \text{isTheSame} \( O(n \log n) \) times.

[We are expecting: A short justification of the number of calls to \text{isTheSame}. You may invoke the Master Theorem if it applies.]

(c) (4 pt.) Formally prove, using an argument by induction, that your algorithm is correct.

[We are expecting: A rigorous proof by induction.]

(d) (0 pt.) (This part’s just for fun!) Is \( O(n \log n) \) the best guarantee you can come up with? Either try coming up with an asymptotically faster algorithm which finds a majority-genre book, or else prove that no such algorithm exists.

[We are expecting: Nothing, this part is not required and will not be graded.]

\(^2\)We’ve only listed the genres for sake of clarity in this example, but the genres aren’t actually going to be labeled for each book in the problem setting.
4. (12 pt.) [Majority Genre, again!] Note: This problem relies on Lecture 5.

You’re still admiring your friend’s book collection, but this time, you’d like to employ randomized algorithms to accomplish the same goal as before: to find a book that has the majority genre. We have the same setting as before: there are \( n \) books of many different genres, and without help you can’t tell what the genre of each book is.

This time around, you’re provided with a deterministic subroutine \texttt{isMajority}, which takes as input a book \( b \), and returns true (if the book is of the majority genre) or false (if the book is not of the majority genre). Assume that running \texttt{isMajority} takes \( \Theta(n) \) time (imagine that this is equivalent to showing a book to your friend, and she needs to spend \( \Theta(n) \) time to scan her collection in order to tell you if the book you’ve picked up is of the majority genre or not). Consider the three randomized algorithms seen in Figure 1, all of which call the subroutine \texttt{isMajority}, and all of which attempt to find a majority book (i.e. a book whose genre is in the majority).

Fill in the table below.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Monte Carlo or Las Vegas?</th>
<th>Expected running time</th>
<th>Worst-case running time</th>
<th>Probability of returning a majority book</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[NOTE: Keep in mind that when we refer to expected running time, we’re taking the expectation over the randomness in the algorithm’s choices, NOT over any distribution of possible inputs.]

[We are expecting: Your filled in table. You must use asymptotic notation for the running times, as usual ignoring constants and lower order terms. For the probability of outputting a majority book, give an exact answer if possible or the tightest lower bound that you can given the information provided (i.e., your bound should still hold with the worst case inputs that still meet the conditions outlined above). No justification is required.]

Algorithms described on next page...
Algorithm 1: findMajorityBook1

Input: A collection $B$ of $n$ books

while true do
    Choose a random $b \in B$;
    if isMajority($b$) then
        return $b$;

Algorithm 2: findMajorityBook2

Input: A collection $B$ of $n$ books

for 100 iterations do
    Choose a random $b \in B$;
    if isMajority($b$) then
        return $b$;

return $B[0]$;

Algorithm 3: findMajorityBook3

Input: A collection $B$ of $n$ books

Put the books in $B$ in a random order;
/* Assume it takes time $\Theta(n)$ to put the $n$ books in a random order */

for $b \in B$ do
    if isMajority($b$) then
        return $b$;

Algorithm 4: isMajority

Input: A collection $B$ of $n$ books and an book $b \in B$

Output: True if $b$ is a book of the majority genre
/* Assume some $\Theta(n)$ work is being done to examine the book collection */

inspectBooksInLinearTime($B$, $b$)
if book $b$ is of the majority genre then
    return True;
else
    return False;

Figure 1: Three randomized algorithms for finding a majority book
5. (11 pt.) [Duck sorbet party] Suppose we have \( n \) ducks standing in a line.

Each duck has a favorite flavor of sorbet: raspberry, lemon, or mango sorbet. You’d like to sort the ducks so that all the lemon-loving ducks are on the left, the raspberry-loving ducks are on the right, and the mango-sorbet-loving ducks are in the middle.\(^3\) You can only do two types of operations on the ducks:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{ask}(j)</td>
<td>Ask the duck in position ( j ) about its favorite sorbet flavor</td>
</tr>
<tr>
<td>\text{swap}(i,j)</td>
<td>Swap the duck in position ( j ) with the duck in position ( i )</td>
</tr>
</tbody>
</table>

The ducks are eager to get their sorbet, but each of the above operations takes a constant number of seconds to execute. Also, you didn’t bring a piece of paper or a pencil, so you can’t write anything down and have to rely on your memory. Like many humans, you can remember up to seven integers\(^4\) between 0 and \( n - 1 \) at a time.

(a) (8 pt.) Design an algorithm to sort the ducks which takes \( O(n) \) seconds, and requires you to remember no more than seven integers\(^5\) between 0 and \( n - 1 \) at a time.

[We are expecting: Pseudocode \text{ AND} a short English description of your algorithm.]

(b) (3 pt.) Justify why your algorithm is correct, why it takes only \( O(n) \) seconds, and why it requires you to remember no more than seven integers at a time.

[We are expecting: Informal justifications of the correctness, runtime, and memory usage of your algorithm that are both clear and convincing to the grader. If it’s easier for you to be clear, you can give a formal proof of correctness, but you do not have to. It is okay to appeal to the correctness of an algorithm that we have seen in class, as long as you clearly explain the relationship between the two algorithms.]

---

\(^3\)Since the ducks will be eating the sorbet right out of the carton, it’s easier if they are sorted by their preferred sorbet flavor.

\(^4\)see, e.g., https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two

\(^5\)You don’t need to use all seven storage spots, but you can if you want to. Can you do it with only two?
Feedback

There’s no “correct” answer here, and it is completely anonymous.

6. (1 pt.) Please fill out the following poll, which asks about how you’re finding the office hours and sections so far:

https://forms.gle/VFoufAchPQPC27p88

Did you fill out the poll?

[We are expecting: The answer “yes.”]