CS 161
Winter 2019

Problem Set 2
Due: Friday, January 25, 2019, at 3pm on Gradescope

Exercises

Please do the exercises on your own.

---

1. **(2 pt.)** In your pre-lecture Exercise for Lecture 3, you saw two different ways to solve the recurrence relation $T(n) = 2 \cdot T(n/2) + n$ with $T(1) = 1$. We saw that $T(n)$ is exactly $n(1 + \log(n))$, when n is a power of two. In this exercise, you'll do the same for a few variants.

   (a) What is the exact solution to $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$ with $T(1) = 3$, when n is a power of 2?

   (b) What is the exact solution to $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 3n$ with $T(1) = 1$, when n is a power of 2?

   [**We are expecting:** *Your answer—**no justification required.** Notice that we want the exact answer, so don't give a $O()$ statement.*]

2. **(4 pt.)** Use any of the methods we've seen in class so far to give big-Oh solutions to the following recurrence relations. You may treat fractions like $n/2$ as either $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$, whichever you prefer.

   (a) $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

   (b) $T(n) = T(n-2) + n$ for $n \geq 2$, and $T(n) = 1$ for $n < 2$. (You may assume n is even if it helps).

   (c) $T(n) = 7T\left(\frac{n}{3}\right) + n^2$ for $n \geq 3$, and $T(n) = 1$ for $n < 3$.

   (d) $T(n) = 7T\left(\frac{n}{2}\right) + n^2$ for $n \geq 2$, and $T(n) = 1$ for $n < 2$.

   [**We are expecting:** *For each item, the best answer you can give of the form $T(n) = O(\_\_\_)$ and a justification. (That is, all of these satisfy $T(n) = O(2^n)$, but you can do better). You do not need to give a formal proof, but your justification should be convincing to the grader. You may use the Master Theorem if it applies.*]

3. **(2 pt.)** Consider the following algorithm, which takes as input an array A:

```
def printStuff(A):
     n = len(A)
     if n <= 4:
         return
     for i in range(n):
         print(A[i])
     printStuff(A[:n/4])  # recurse on first n/4 elements of A
     printStuff(A[3*n/4:])  # recurse on last n/4 elements of A
     return
```

   What is the asymptotic running time of `printStuff`?

   [**We are expecting:** *The best answer you can give of the form "The running time of `printStuff` is $O(\_\_\_\_)$" and a short explanation.*]

Problems

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

- 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.

---

4. **(4 pt.)** [**Substitution Method.**] Consider the function $T(n)$ defined recursively by

$$T(n) = \begin{cases} T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lceil \frac{n}{4} \rceil\right) + n & n > 4 \\ 1 & n \leq 4 \end{cases}$$

Prove using the substitution method that $T(n) = O(n)$. If it helps **you may assume that n is a power of 2.** You may also take it as given than $T(n) \geq 0$ for all n.

[**We are expecting:** *A formal proof by induction. Remember to state your inductive hypothesis, base case, inductive step, and conclusion. Notice that it's fine to prove a big-Oh bound, you don't need to come up with exact formula for $T(n)$.*]

5. **(5 pt.)** [**Fishing.**] Plucky the Pedantic Penguin is fishing. Plucky catches n fish, but plans to keep only the k largest fish and to throw the rest back. Plucky has already named all of the fish, and has measured their length and entered the (name, length) pairs into an array F of length n. For example, F might look like this:

$$F = \begin{bmatrix} \text{(Frederick the Fish, 14.2in)} \\ \text{(Fabiola the Fish, 10in)} \\ \text{(Farid the Fish, 12.35in)} \\ \vdots \\ \text{(Felix the Fish, 6.234523in)} \\ \text{(Finlay the Fish, 6.234524in)} \end{bmatrix}$$

   (a) **(2 pt.)** Give an $O(n \log(n))$-time deterministic algorithm that takes F and k as input and outputs the names of the k largest fish. You may assume that the lengths of the fish are distinct.

   Note: Your algorithm should run in time $O(n \log(n))$ even if k is a function of n. For example, if Plucky wants to keep the largest $k = n/2$ fish, your algorithm should still run in time $O(n \log(n))$.

   [**We are expecting:** *Pseudocode **AND** a short English description of your algorithm. You may (and, hint, may want to...) invoke algorithms we have seen in class. You do not need to justify why your algorithm is correct or its running time.*]

   (b) **(3 pt.)** Give an $O(n)$-time deterministic algorithm that takes F and k as input and outputs the names of the k largest fish. You may assume that the lengths of the fish are distinct. Your algorithm should also be fundamentally different than your algorithm for part (a). (That is, don't solve part (b) and then use that as your solution to part (a); part (a) should be easier).

   Note: Your algorithm should run in time $O(n)$ even if k is a function of n. For example, if Plucky wants to keep the largest $k = n/2$ fish, your algorithm should still run in time $O(n)$.

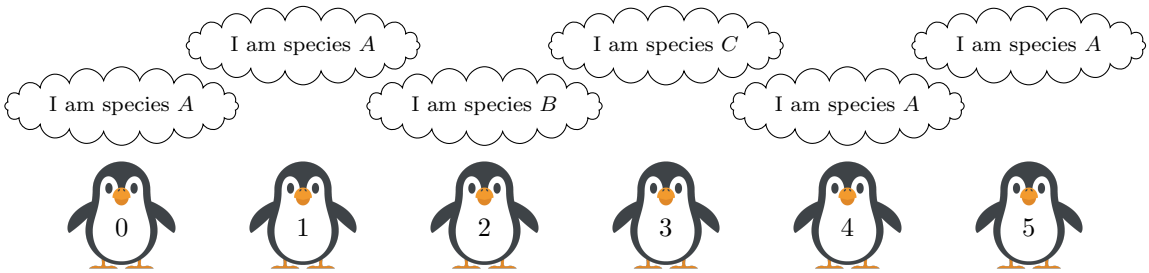
   [**We are expecting:** *Pseudocode **AND** a short English description of your algorithm. You may (and, hint, may want to...) invoke algorithms we have seen in class. You do not need to justify why your algorithm is correct or its running time.*]

6. **(8 pt.)** **[Penguinology.]** On an island, there are n penguins of many different species. The differences between the species are very subtle, so without help you can't tell the penguins apart at all. Fortunately, you have an expert with you, and she can tell you whether or not two penguins belong to the same species. More precisely, she can answer queries of the form:

$$\texttt{isTheSame(penguin1, penguin2)} = \begin{cases} \textbf{True} & \text{if } \texttt{penguin1} \text{ and } \texttt{penguin2} \text{ belong to the same species} \\ \textbf{False} & \text{if } \texttt{penguin1} \text{ and } \texttt{penguin2} \text{ belong to different species} \end{cases}$$

The only way you can get any information about the penguins is by running `isTheSame`. You cannot ask them what species they are, or compare them in any other way.

The expert assures you that one species of penguin is in the majority. That is, there are *strictly greater* than $n/2$ penguins of that species. Your goal is to return a single member of that majority species. For example, if the population looked like this:



then species A is in the majority, and your algorithm should return any one of Penguins 0, 1, 4, or 5.

If there is no species with a strict majority, your algorithm may return whatever it wants.

(a) **(4 pt.)** Design a deterministic divide-and-conquer algorithm which uses $O(n \log(n))$ calls to `isTheSame` and returns a penguin belonging to the majority species. You may assume that n is a power of 2 if it is helpful.

[**We are expecting:** *Pseudocode (which calls* `isTheSame`*)* ***AND*** *a clear English description of what your algorithm is doing.*]

(b) **(1 pt.)** Explain why your algorithm calls `isTheSame` $O(n \log(n))$ times.

[**We are expecting:** *A short explanation. You may use the Master Theorem if it applies.*]

(c) **(3 pt.)** Prove, using an argument by induction, that your algorithm is correct.

[**We are expecting:** *A rigorous proof by induction. Make sure to clearly state your inductive hypothesis, base case, inductive step, and conclusion.*]

(d) **(NOT REQUIRED. 1 BONUS pt.)** Is $O(n \log(n))$ the best guarantee you can come up with? Either give an asymptotically faster algorithm which finds a majority-species penguin, or else describe prove that no such algorithm exists. (Or, if you think that your algorithm from part (a) already does better that $\Theta(n \log(n))$, just write that :)).

[**We are expecting:** *Nothing. This part is not required.*]

Feedback

This part is not worth any points, but it is quick, painless, and anonymous, and we'd really appreciate it if you help us out by giving us feedback!

1. **(0 pt.)** Please fill out the following poll, which asks about the pace of lectures so far:

   `https://goo.gl/forms/mOUlY8r2eeZGP3fB2`