CJS 320 Problem Set 2

Due: Wednesday 10/06/21

Welcome to Problem Set 2, which was written in collaboration with the ghost of J.R.R. Tolkien. A few notes before you begin.

- You may collaborate with up to 4 other people. All work must be written up individually. You may collaborate, but you cannot copy. E.g. you can talk to your friends about ideas for a problem together, but you cannot copy their solution and just change some words around. **List your collaborators at the top of your submission**.
- Googling or looking up solutions in any way is not allowed.
- Cheating is not worth it! Your grade in this course does not define your worth as a person, and in 10 years you will not care about your bad grade on a homework assignment. But you will care if you are caught plagiarizing: plagiarizing has serious consequences, including the potential of expulsion.
- These problems are designed to challenge you. Start them early; if we've done our job well writing them, you will have to chew on them for a while before finding the solutions, and that means you will do best if you can sleep on your solutions rather than starting them the night before the due date.
- You do not need to implement anything in code. In fact, please do not. Pseudocode or a clear English explanation of your algorithm are both acceptable: in some cases pseudocode may be clearer than plain English, and in others the plain English might be better.
- If a question asks you to come up with an algorithm with a certain target runtime (say $O(n \log n)$), and you don't know how to achieve this runtime but know how to solve the problem less efficiently (in, say, $\Theta(n^2)$ time), write that down! Depending on how inefficient your solution is compared to the benchmark, it will receive a varying amount of credit.
- For each of our algorithm design questions, you should come up with a *deterministic* algorithm unless the question specifies that a randomized algorithm is wanted.
- All analysis must be mathematically rigorous. Any answer you provide should be proven.
- Remember, we can't evaluate your work if we can't understand it. Communicating mathematical and/or complex ideas is an important skill in computer science; treat your problem sets as practice.
- You should use LaTeX to typeset your solutions.
- You do not need any knowledge of Lord of the Rings in order to complete this problem set. Note that the majority of the TAs writing the problems did not have any prior experience with Lord of the Rings. Ira has enough Lord of the Rings knowledge for everyone combined.
- Have fun!!

Problem 1 (Bean Bonanza)**.** It's bean-picking season! Samwise picked and shelled all the beans in the garden, and sorted them by size. Unfortunately, Sam is farsighted, so made some mistakes in ordering them: the first bean is the smallest and the nth is the largest, but currently some are out of order. Sam is pretty sure that all of the beans are within 5 indices of where they were supposed to be. Provide an $O(n)$ algorithm for sorting everything the rest of the way, assuming Sam's assumption is correct.

Problem 2 (Chili Challenges)**.** Now that Sam and Merry have all of the beans sorted, they want to make a hearty bean soup and serve it to their n friends. But their friends are fickle, and all have a preferred temperature for eating their soup, which they each wrote on a piece of parchment. Sam and Merry need to order the temperatures so that you know the serving order. They've come up with a divide-and-conquer algorithm to sort the piles, where at every step you divide the piles into 3 subpiles and recurse on those. Unfortunately, they also get some number of paper cuts at every step of the dividing, which leads to an additive factor of n^2 at every division. So, the algorithm ends up with the following recurrence relation:

$$T(n) = 3T(n/3) + n^2,$$
$$T(1) = 1.$$

Draw the computation tree associated with the recursion relation, and propose and prove the run-time.[1]

Problem 3 (So-so Sorting)**.** Dís, the lady of Erebor, has claims she has found a comparison-based sorting algorithm that performs only c comparisons, which she plans to use to sort the different quality-levels of gems mined from the Lonely Mountain. However, she admit that it's not perfect—the algorithm will only output a correctly sorted list with a certain probability p, where p is over the internal randomness of the algorithm. Find the highest value p can possibly be, in terms of n and c.

Problem 4 (Scrivened Scrolls)**.** Gandalf is sorting scrolls in the libraries of Minas Tirith, searching for the Scroll of Isildur. If he takes too long, he won't be able to get back to Frodo before the ringwraiths find him, and all of Middle Earth will be lost to Sauron's forces. To speed up his search, he has decided to first sort all n scrolls by title. Each scroll has a title between 2 and m characters long. It is written in the Tengwar script, which has 36 characters in it and has a standard lexicographic ordering. Devise an algorithm for sorting the names that runs in time $O(nm)$. Explain in a sentence or two why we are able to beat the $n \log n$ bound in this case.

Problem 5 (Optimizing for Orodruin)**.** Frodo needs to cast the one ring of power into the fires of Mount Doom, but he does not know the way. He has made it into Mordor, and he has a map divided into an $n \times n$ grid of n^2 quadrants, each marked with their average elevation. Other than Mount Doom, Mordor consists of the plateau of Gorgoroth, which, while not completely flat, does not contain any local high points (e.g. the only quadrant of the map that is higher than any of its neighbors is the peak of Mount Doom).[2] Assume that each altitude is distinct, i.e. no two quadrants have the same elevation. Frodo needs to find Mount Doom on his map, since there is too much smoke to see it directly. But he is really tired (the ring of power takes a lot of energy), so looking at a single quadrant of his map to figure out if it's the right one takes a lot of work. Assume that inspecting the ij^{th} quadrant of the map takes constant time, and come up with an $O(n)$ algorithm for Frodo to find Mount Doom on his map.

Problem 6 (Cocky Cocklebur)**.** In the year 1427 of the Shire Reckoning, a wonderful event happened: Sam planted Galadriel's gift, a silver Mallorn nut, in the Party Field, and very quickly a beautiful Mallorn tree grew out of it! When it bloomed next summer, the residents of the Shire became so infatuated with its golden blossom that they insisted that Sam should cultivate an entire Mallorn tree arboretum. Happy to oblige, Sam spent the next year obtaining $n \cdot m$ Mallorn saplings, and eventually planted them in a regular grid consisting of $n \times m$ equal-sized quadrants in the Party Field in the spring of 1429. Unfortunately, it quickly turned out that the Party Field had become infested with cocklebur, a fairly nasty weed propogated by dark forces, that poses a grave threat to the existence of the young and frail Mallorn saplings.

[1] To draw the tree, we recommend mathcha.io, which can generate either Tikz or a png for an image.

[2] This is not strictly true, but since Mordor is already fictional, we make liberties with Tolkien's work for the sake of a tractable problem.

Thankfully, Sam has a solution: he can apply the light of Eärendil to any quadrant of the arboretum, and its power will repel the darkness of the cocklebur weed. To each of the $n \cdot m$ saplings' quadrants, he will apply the light entirely or not at all. There is a tradeoff—Sam should not use too much or too little of it. To describe this tradeoff, let us refer to any 4 neighboring quadrants forming a 2×2 subsection of the $n \times m$ grid as a 2×2-*square*):

- *(Overconcentration)* If there is a 2×2-square all of whose 4 quadrants receive a dose of the light, the soil will dry too much and the Mallorn saplings will die.
- *(Underconcentration)* if there is a 2×2-square none of whose 4 quadrants receive a dose of the light, cocklebur will persist on that 2×2 square and thus become impossible to fully weed out.

Subject to these constraints—that no 2×2 square should suffer from overconcentration or underconcentration — design an algorithm that helps Sam count up the total number of ways for him to distribute the light of Eärendil across the $n \times m$ quadrants. The target runtime is $O(nm \cdot 2^{\min\{n,m\}})$.

Problem 7 (Hippity Hop)**.** Frodo and Bilbo like to play a collaborative version of chess, which reminds them that life is about cooperation as much as it is about competition. The game takes place on a gigantic $n \times m$ board ($n, m \geq 100$). Frodo controls Bill the Pony and Bilbo controls Shadowfax, Lord of All Horses, which are assigned initial positions (x_f, y_f) and (x_b, y_b). Bill the Pony's move is described as follows: from position (x, y) it can jump to one of the following 8 locations (provided the respective location is within the board's bounds): $(x \pm 2, y \pm 1), (x \pm 2, y \mp 1), (x \pm 1, y \pm 2), (x \pm 1, y \mp 2)$.[3] Meanwhile, Shadowfax jumps farther than Bill, namely, from position (x, y) it can jump to one of the following 8 locations (provided the respective location is not off the board): $(x \pm 3, y \pm 2), (x \pm 3, y \mp 2), (x \pm 2, y \pm 3), (x \pm 2, y \mp 3)$. Frodo moves first and the players alternate moves (skipping a move is not allowed). The objective is for Frodo and Bilbo's horsies to end up at the same position as quickly as possible.[4]

Given m, n and the initial positions of Bill the Pony and Shadowfax, Lord of All Horses, Frodo and Bilbo ask you to design an $O(mn)$ time algorithm that either outputs the minimum number of half-moves[5] until Bill the Pony and Shadowfax can meet at some location on the board, or attests that such an encounter is impossible.

[3] In other words, Bill the Pony hops twice in some direction and then once in an orthogonal direction.

[4] Unlike real chess, Bill the Pony and the Shadowfax are allowed to jump to the same position at any point of the game.

[5] A *half-move* is a single move by either Bill the Pony or Shadowfax; e.g. Frodo moving first and Bilbo moving next counts as two half-moves.