CS 170 Homework 2

Due **Saturday 9/13/2025, at 10:00 pm (grace period until 11:59pm)**

1 Study Group (1 point)

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Squaring vs. multiplying matrices (11 points)

The square of a matrix A is its product with itself. Namely, it is the matrix product AA.

(a) (3 points) Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

(b) (4 points) What is wrong with the following algorithm for computing the square of an $n \times n$ matrix?

"Use a divide-and-conquer approach as in Strassen's algorithm, except that instead of getting 7 subproblems of size $n/2$, we now get 5 subproblems of size $n/2$ thanks to part (a). Using the same analysis as in Strassen's algorithm, we can conclude that the algorithm runs in $\mathcal{O}(n^{\log_2 5})$ time."

(c) (4 points) In fact, squaring matrices is no easier than multiplying them. Show that if $n \times n$ matrices can be squared in $\Theta(n^c)$ time, then any $n \times n$ matrices can be multiplied in $\Theta(n^c)$ time.

(Hint: Any $2n \times 2n$ matrix A consists of 4 blocks of $n \times n$ matrices. What would A^2 look like if two carefully-chosen blocks were set to be zero matrices?)

3 Satellite diagnostics (12 points)

A satellite has n onboard sensors, of which s are faulty. To diagnose them, mission control can run *batch tests*: in each round, you select a subset of sensors to activate simultaneously. A test passes if none of the selected sensors are faulty, and fails if *at least one* faulty sensor is included. After each test, you only learn whether it passed or failed. In particular, it does not specify which of the sensors was the faulty one.

(a) (2 points) If there is exactly one faulty sensor ($s = 1$), devise a strategy that identifies it in $O(\log n)$ tests. You do not need to prove that your strategy works.

(b) (2 points) In the general case with s faulty sensors, evenly split the n sensors into x disjoint groups (each of size $\approx n/x$) and test every group. In terms of x and s, at least how many of these x tests are guaranteed to successfully pass?

(c) (8 points) Design a strategy that identifies *all* faulty sensors using $O(s \log(n/s))$ tests.

Hint: Choose x in part (b) so that you eliminate at least half the sensors each round.

(i) Describe your strategy.

(ii) Prove that your strategy works.

(iii) Analyze the worst-case number of tests.

4 Two sorted arrays (10 points)

You are given two sorted arrays of integers, each of size k. Give an efficient (i.e., better than $O(k)$-time) algorithm to find the k-th smallest element in the union of the two arrays. You may assume that all the elements are distinct.

Your solution should contain a description of the algorithm, a proof of correctness, and a runtime analysis. (i.e a 3-part solution). **In addition, please provide corresponding pseudo-code.**

5 The magical keys and locks (12 points)

In the ancient kingdom of Keydom, there are n magical keys and n enchanted locks, each of a unique shape and size. Every key opens exactly one lock, and every lock fits exactly one key. You cannot directly compare two keys with each other, nor can you compare two locks. The only thing you can do is insert any key into any lock and see whether it is too small for the lock's keyhole, too big for it, or if it is a perfect fit.

After a lively lantern festival in the village square, the townsfolk accidentally jumbled all the keys and locks together into a single array of $2n$ items, in completely random order. No one remembers which key matches which lock. Your task is to match each key back to its corresponding lock.

(a) (3 points) Say a lock has r keys that are too small for it. Using $O(n)$ key-lock comparisons, show that the problem can be reduced to two sub-problems: one asking to match r keys with their respective r locks, and the other asking to match $n-r-1$ keys with $n-r-1$ locks. *(Hint: That lock has a matching key.)*

(b) (4 points) Using part (a), design a *randomized* algorithm that takes this array of $2n$ items as input and returns an array of n key-lock pairs, each of which is a perfect fit. Furthermore, if $A(n)$ denotes the expected runtime of this algorithm, prove that it satisfies the recurrence relation

$$A(n) = \frac{1}{n}\sum_{r=0}^{n-1}\left(A(r) + A(n-r-1)\right) + (2n-1).$$

(Hint: How can you pick the lock in part (a) so that the two sets of keys are roughly the same size?)

(c) (5 points) Let us make the (unproven!) assumption that the more balanced the split is, the less time the algorithm we designed in part (b) will take. This can be formally stated as follows: for any positive integers $x \geq y \geq z$, we have

$$A(x+y) + A(x-y) \geq A(x+z) + A(x-z).$$

Using the recurrence relation from part (b) and this assumption, show that $A(n) = O(n \log n)$.

(d) (**Extra credit**, 5 points) How can you provably show that $A(n) = O(n \log n)$ without making any unjustified assumptions?

(Hint: Try to prove by strong induction that the expected running time is at most $4nH_n$, where $H_n = \sum_{i=1}^{n} 1/i$ is the n'th harmonic sum. In your solution, you can assert without proof that $H_n = O(\log n)$.)

6 [Coding] Quickselect (6 points)

For this week's homework, you'll implement the Quickselect algorithm in a Python Jupyter notebook called `quick_select.ipynb`. There are two ways that you can access the notebook and complete the problems:

1. **On Local Machine**: `git clone` (or if you already have it, `git pull`) from the coding homework repo,

   `https://github.com/Berkeley-CS170/cs170-fa25-coding`

   and navigate to the `hw02` folder. Refer to the `README.md` for local setup instructions.

2. **On Datahub**: Click here and navigate to the `hw02` folder if you prefer to complete this question on Berkeley DataHub.

