Problem Set 3

This third problem set is all about divide-and-conquer algorithms and recurrence relations. Some of these problems are of practical importance, while others explore theoretical aspects of recurrences and recurrence relations. By the time you're done, you'll have a much stronger understanding of this algorithmic technique.

Please be sure to write your answers different problems on separate pieces of paper to make it easier for us to grade. Also, **please put your name on each page of your assignment**.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 36 possible points. It is weighted at 12% of your total grade. The earlier questions serve as a warm-up for the later problems, so the difficulty of the problems increases over the course of this problem set.

Good luck, and have fun!

Due Monday, July 22 at 2:15 PM

Problem One: Majority Voting* (9 Points)

In safety-critical environments like nuclear reactors, it's important to ensure a sporadic hardware or software failure can't bring down a computer system. One way to do this is to introduce redundancy by having multiple computers independently perform the same computation. The computers can then use a majority vote to decide which value to use. The system will then work as long as strictly more than half the computers get the right answer.

Let's suppose that you have a computer system with n total computers, each of which have run identical computations but may have encountered errors. You want to determine whether strictly more than half of the computers performing the computation arrived at the same result. However, you are only allowed to determine whether the computers arrived at the same result by querying a pair of computers and receiving one of two pieces of information:

- The computers produced the same result, or
- The computers produced different results.

Design an algorithm that uses queries of the above form to determine whether strictly more than $n / 2$ computers in the system arrived at the same result. Your algorithm should make at total of at most $O(n \log n)$ queries. Then:

- Describe your algorithm.
- Prove that your algorithm is correct.
- Prove that your algorithm makes at most $O(n \log n)$ total queries.

Problem Two: Why Five? (4 Points)

Why does the linear-time selection algorithm split the input into blocks of size five? Why doesn't it split it into blocks of some other size, like three? This question explores why.

If the selection algorithm were to split the input apart into blocks of size three, it would differ from the current algorithm in the following ways:

1. The recursive call to determine the median-of-medians would be made on an input of size $\lceil n / 3 \rceil$ rather than roughly $\lceil n / 5 \rceil$.
2. The median-of-medians would guarantee roughly a 67% / 33% split during the partitioning step instead of a 70% / 30% split, though as with the original analysis some of the blocks will have to be discounted.

Suppose that we want to provide a lower bound on the worst-case runtime for this function. We could do so by using this recurrence relation, which assumes that the partition step always causes the recursion to continue into the larger subarray:

$$T(n) \geq c \qquad \text{if } n \leq 100$$

$$T(n) \geq T(\lceil n / 3 \rceil) + T(\lceil 2n / 3 \rceil + 4) + cn \qquad \text{otherwise}$$

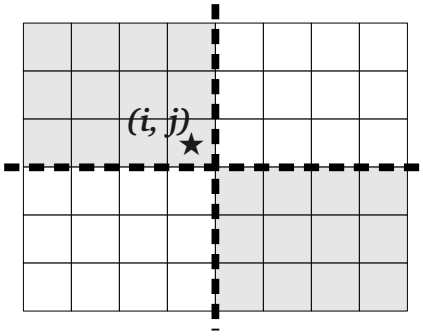
Find an asymptotic lower bound for this recurrence and prove that your bound is correct by using the substitution method. To receive full credit, your lower bound should be as tight as possible, though you don't need to prove that you have the tightest bound. *(Hint: Try thinking about the shape of the recursion tree.)*

* Adapted from Exercise 5.3 of *Algorithm Design* by Kleinberg and Tardos

Problem Three: Searching a Grid, Part II (4 Points)

In the first problem set, you devised an $O(m + n)$-time algorithm for searching a sorted grid for a particular value. In lecture, we saw a second algorithm for solving this problem that worked by considering the middle element of the grid, then choosing whether to recurse in the upper-left quadrant of the grid or into the other three quadrants. This gave an $O((mn)^{\log_4 3})$-time algorithm.

However, this can be improved by using the following algorithm. Begin by using binary search on the main diagonal of the grid to find a position (i, j) where $A[i, j] \leq k$ and $A[i + 1, j + 1] > k$.* If $A[i, j] = k$, then we're done. Otherwise, split the grid into four smaller rectangles as shown below, then recursively search the two white rectangles for the value k:



The runtime of this algorithm, in terms of Z, the total number of elements in the grid, is given by the recurrence

$$T(Z) \leq c \qquad \text{if } Z \leq 16$$
$$T(Z) \leq 2T(\lceil Z / 4 \rceil) + c \log_2 Z \qquad \text{otherwise}$$

Intuitively, the $c \log_2 Z$ term accounts for performing a binary search along the diagonal of the matrix, and the $2T(\lceil Z / 4 \rceil)$ term accounts for the two recursive calls, both of which are made on subgrids that each contain at most a quarter of the total number of elements as the original grid.

Find an asymptotic upper bound on this recurrence and prove that your bound is correct. To receive full credit, your upper bound should be as tight as possible, though you don't need to prove that you have the tightest bound.

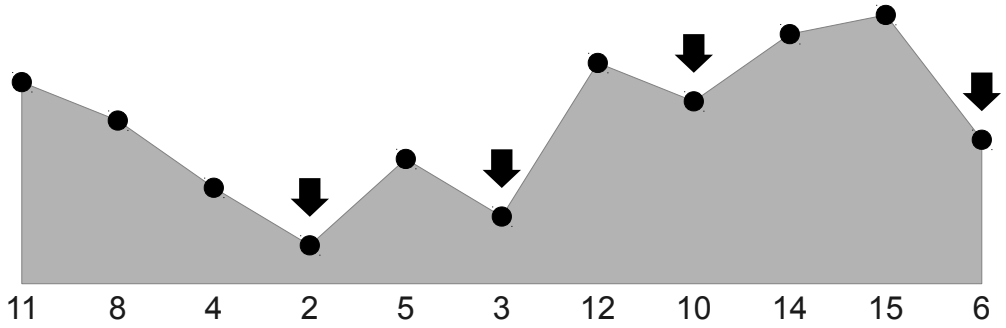
The next problem didn't fit on this page, so here's a picture of a quokka instead!

* If you haven't seen binary search used this way before, try working out how to do this on a normal 1D array. You would use it in the same way here, except that you'd go along the diagonal of the grid.

Problem Four: Collecting Rainwater (9 Points)

Suppose that you are interested in setting up a collection point to funnel rainwater into a town's water supply. The town is next to a ridge, which for simplicity we will assume is represented as a 1D array of the elevations of different points along the ridge.

When rain falls on the ridge, it will roll downhill along the ridge. We'll call a point where water naturally accumulates (that is, a point lower than all neighboring points) a "good collection point." For example, here is a possible ridge with good collection points identified:



You are given an array of n elements representing the altitudes of points along the ridge, all of which are distinct. Design an O(log n)-time algorithm that finds a good collection point. Then:

- Describe your algorithm.
- Prove your algorithm is correct.
- Prove your algorithm runs in time O(log n).

Problem Five: Order Statistics in Compressed Data Sets (9 Points)

Suppose that you have a collection of data points where there are a huge number of copies of each data point. For example, you might have the data set

A, B, A, C, A, B, A, D, A, B, A, C, A, B, A, E, A, B, A, C, A, B, A, D, A, B, A, C, A, B, A, F

Here, there are 32 data points, but there are only six distinct values. Rather than storing the data as above, suppose that you store the data as a list of tuples of the form (*value*, *frequency*). For example, the above data might be stored as

(A, 16), (B, 8), (C, 4), (D, 2), (E, 1), (F, 1)

That is, there are sixteen A's, eight B's, three C's, two D's, one E, and one F.

Suppose that you have a data set represented as m of these tuples, where the tuples are stored in no particular order. Design an O(m)-time algorithm for computing the kth order statistic of the data set. Note that m is the number of tuples rather than the total number of elements n in the uncompressed data set, so your algorithm's runtime should not depend asymptotically on n. Then

- Describe your algorithm.
- Prove that your algorithm is correct.
- Prove that your algorithm runs in time O(m).

(Hint: You may want to use the linear-time selection algorithm as a subroutine. If you do, we ***strongly recommend*** *using the algorithm as a black box rather than trying to modify it.)*

Problem Six: Course Feedback (1 Point)

We want this course to be as good as it can be, and we'd appreciate your feedback on how we're doing. For a free point, please answer the following questions. We'll give you full credit no matter what you write, as long as you write something.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit course?

ii. Did you attend office hours? If so, did you find them useful?

iii. Did you read through the textbook? If so, did you find it useful?

iv. How is the pace of this course so far? Too slow? Too fast? Just right?

v. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?