Problem Set 3

MIT students: This problem set is due in lecture on *Day 11*.

Reading: Chapters 8 and 9

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation instructor and time, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 3-1. Do exercise 8.1-2 on page 167 of CLRS.

Exercise 3-2. Do exercise 8.1-3 on page 168 of CLRS.

Exercise 3-3. Do exercise 8.2-3 on page 170 of CLRS.

Exercise 3-4. Do exercise 8.4-2 on page 177 of CLRS.

Exercise 3-5. Do exercise 9.3-1 on page 192 of CLRS.

Exercise 3-6. Show that the second smallest of n elements can be found with $n + \Theta(\lg n)$ comparisons in the worst case. (*Hint:* Also find the smallest element.)

Problem 3-1. Largest i numbers in sorted order

Given a set of n numbers, we wish to find the i largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the algorithms in terms of n and i.

(a) Sort the numbers, and list the i largest.

(b) Build a max-priority queue from the numbers, and call EXTRACT-MAX i times.

(c) Use an order-statistic algorithm to find the ith largest number, partition around that number, and sort the i largest numbers.

Problem 3-2. At the wading pool

You work at a summer camp which holds regular outings for the n children which attend. One of these outings is to a nearby wading pool which always turns out to be something of a nightmare at the end because there are n wet, cranky children and a pile of $2n$ shoes (n left shoes and n right shoes) and it is not at all clear which kids go with which shoes. Not being particularly picky, all you care about is getting kids into shoes that fit. The only way to determine if a shoe is a match for a child is to try the shoe on the child's foot. After trying on the shoe, you will know that it either fits, is too big, or is too small. It is important to note that you cannot accurately compare children's feet directly with each other, nor can you compare the shoes. You know that for every kid, there are at least two shoes (one left shoe and one right shoe) that will fit, and your task is to shoe all of the children efficiently so that you can go home. There are enough shoes that each child will find a pair which fits her. Assume that each comparison (trying a shoe on a foot) takes one time unit.

(a) Describe a deterministic algorithm that uses $\Theta(n^2)$ comparisons to pair children with shoes.

(b) Prove a lower bound of $\Omega(n \lg n)$ for the number of comparisons that must be made by an algorithm solving this problem. (*Hint:* How many leaves does the decision tree have?)

(c) How might you partition the children into those with a smaller shoe size than a "pivot" child, and those with a larger shoe size than the pivot child?

(d) Give a randomized algorithm whose expected number of comparisons is $O(n \lg n)$, and prove that this bound is correct. What is the worst-case number of comparisons for your algorithm?