MATH 416, PROBLEM SET 2

Comments about homework.

- Solutions to homework should be written clearly, with justification, in complete sentences. Your solution should resemble something you'd write to teach another student in the class how to solve the problem.
- You are encouraged to work with other 416 students on the homework, but solutions must be written independently. Include a list of your collaborators at the top of your homework.
- You should submit your homework on Gradescope, indicating to Gradescope where the various pieces of your solutions are. The easiest (and recommended) way to do this is to start a new page for each problem.
- Attempting and struggling with problems is **critical** to learning mathematics. Do not search for published solutions to problems. I don't have to tell you that doing so constitutes academic dishonesty; it's also a terrible way to get better at math.

  If you get stuck, ask someone else for a hint. Better yet, go for a walk.

Problem 1. In class we described a binary-search algorithm that searches a sorted array $L[1, \ldots, n]$ for a given element x in time $O(\log n)$. Show that this is optimal, in the following sense: any algorithm that access the array only via comparisons must take $\Omega(\log n)$ steps.

Problem 2. Consider the Bubblesort algorithm, given in pseudocode below.

Algorithm 1: Bubblesort

```
  Data: A = (A[1], ..., A[n]) a list of n numbers
1 for i = 1 to i = n do
2     for j = n downto i + 1 do
3         if A[j] < A[j − 1] then
4             swap A[j − 1] and A[j]
5 return A
```

(a) Let A' denote the output of Bubblesort on input A. To prove that Bubblesort is correct, we need to prove at least that A' is correctly sorted. What else must be proved?

(b) Identify and prove the validity of a loop invariant for the **for** loop in lines 1–4.

(c) Identify and prove the validity of a loop invariant for the **for** loop in lines 2–4.
(d) What is the worst-case running time of Bubblesort? How does it compare to the running time of Insertion Sort?

Problem 3. Prove (carefully!) by induction that a binary tree of height $\leq h$ has at most 2^h leaves.

Problem 4. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \log_2(n)$.

Problem 5. We can express insertion sort as a recursive procedure as follows. In order to sort $A[1 \ldots n]$, we recursively sort $A[1 \ldots n-1]$ and then insert $A[n]$ into the sorted array $A[1 \ldots n-1]$. Write an recurrence for the worst-case running time of this recursive version of insertion sort.

Problem 6. Let $A[1 \ldots n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A.

(a) List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.
(b) Without proving it. What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? (and how many are there?).
(c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.
(d) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \log(n))$ worst-case time. (Hint: Modify merge sort.) You do not need to prove that your algorithm runs in $\Theta(n \log(n))$, but you should give a brief explanation.

Problem 7. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a flip – insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over. Describe an algorithm to sort an arbitrary stack of pancakes using $O(n)$ flips.