CSC236 – Problem Set 5

There are two components of this problem set. The preliminary question is not marked or submitted: it is there as a suggested exercise that you should do early to make sure that you're on track. The problem set itself is what you will submit for marks.

Get in the habit of starting work early – the less time you give yourself, the more stressed you'll find yourself each week!

To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

The PDF file you submit must be typed, scanned handwritten submissions will not be marked.

Preliminary: Not Marked

This question is an opportunity for you to check your understanding of the topics and practice writing formal solutions. This is a valuable *learning opportunity* – if you see that you're at a loss, get help quickly!

In class, we studied the MergeSort sorting algorithm. It divides the list in two, sorts each half, and merges the sorted results. A similar idea is to split the list in three, sort each third, then merge the sorted results.

Write the pseudocode (or Python code) of this three-way MergeSort algorithm and analyze the runtime of this algorithm. Begin by developing a recurrence, and explain why your recurrence is correct. Then, use the substitution method then the big-Oh proof to give the asymptotic runtime of the algorithm.

Problem Set: due October 28, 2016 22:00, required filename: ps5sol.pdf

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity.

Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

1. **[12]** This problem set has only one question, in which you will experience the whole work flow of designing and analyzing a divide-and-conquer algorithm to solve a real-life problem efficiently.

   Here is the problem: Suppose we are running a presidential election in some country, and the constitution of the country states that the winning candidate must obtain **strictly more than half** of the votes to be elected, i.e., let n be the total number of votes, if $n = 20$, you must obtain at least 11 votes to win; if $n = 21$, you also need 11 votes to win. Your job is to, given all the votes, figure out who is the winner of the election, or decide that nobody is winning.

   The input of your algorithm is a list "`votes`" of all casted votes, where each element is a `Candidate` object. The list may look like the following

   ```
   votes = [larry, dan, hillary, larry, donald, dan, larry, larry, larry, arnold, larry]
   ```

   You algorithm should return a `Candidate` object who is the winner of the election (e.g., in the above example you should return candidate "`larry`"), or return `None` if nobody is a majority winner.

   Note that you CANNOT sort the list of votes because the output of the comparison operation between `Candidate` objects, like "`larry < dan`" is NOT defined. However, you CAN test the equality between `Candidate` objects, because the output of the equality test "`larry == larry`" is well defined.

   Use the divide-and-conquer technique to develop an efficient solution to this problem. Present your solution by including the following components in your submission.

   (a) Write the pseudocode of your algorithm, written in the syntax that is similar to the Python-alike pseudocode that we used in the lectures.

   (b) Describe in English the ideas behind your solution, and justify why your algorithm provides the correct answer.

   (c) Find the recursively defined function that represents the worst-case runtime of your algorithm, and justify why the recurrence is correct.

(d) Find the closed form of the recursively defined function, using repeated substitution. You may skip Step 5, the induction proof for the potential closed form.

(e) Find an asymptotic upper-bound (big-Oh) of your algorithm's runtime, and prove it.

Notes and hints:

(a) In order to get the full mark, you should aim at developing an $\mathcal{O}(n \log n)$ algorithm. A clever $\mathcal{O}(n)$ algorithm is also possible, but it is not required for this problem set.

(b) You must use divide-and-conquer and develop a recursive algorithm. No mark will be given to an algorithm that is not using divide-and-conquer. Also, you're NOT allowed to use any data structures like dictionaries or hash tables.

(c) Start by thinking about this: If we divide the list into two halves and get the solutions for the two sub-problems on the left and right halves of the list. How do we use these sub-solutions to form the solution to the original problem? Think different cases.

(d) If there exists a candidate that has more than half of the votes, what are necessary conditions that this candidate must satisfy? In other words, under what condition can you say things like "There is no way this candidate can be a winner", or "There is no way there exists a winner"?