Homework 4: Divide and Conquer

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long our justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less.

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to Gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Applying the Master Theorem [10 points]

For each recurrence relation below, do the following:

- First, indicate whether the master theorem can be used to find a Θ bound on that recurrence relation.
- Next, If the master theorem does apply, then use it to find an asymptotic bound. If the master theorem does not apply, explain why.

(a) $T(n) = 5T(\frac{n}{5}) + 5$

(b) $T(n) = T(\frac{n}{8}) + n^2$

(c) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

(d) $T(n) = 5T(\frac{n}{5}) + \log n$

(e) $T(n) = 3T(\frac{2n}{3}) + n^3$

2. What a Bargain! More for Less! [25 points]

In Section 4, you developed a divide and conquer algorithm to answer the following problem: "Given a 1-dimensional array A of n real numbers find the total sum of the maximum-weight sub-array of A." Your job for this problem will be to do more than was asked in section, namely to solve even more problems: That is you need to find:

- the maximum sum of any sub-array of the input.
- the maximum sum of any prefix of the array, and
- the maximum sum of any suffix of the array.

For example, in the array

$$\{31, -49, 59, 26, -53, 58, 97, -93, -23, 54\}$$

- the sub-array with maximum sum is achieved by summing the 3rd through 7th elements, whose sum is 187. (When all numbers are positive the answer would be the whole array; when all numbers are negative the answer would be an empty sub-array which has a total of 0.)
- the prefix sum is maximized with the prefix ending in the 7th element, which has sum 169.
- the suffix sum is maximized with the suffix that begins with the 3rd element, which has sum 125.

Give an $O(n)$ time *divide and conquer* algorithm to find all three of these maximum sums in an array of n numbers. This problem shows how sometimes asking for more in our recursive calls can help us with designing more efficient recursive solutions.

Hint: It may help to think about how there is work repeated in the different recursive calls in the solution from section. You will likely need to add one more item to be computed than required here in order to make your solution as efficient as we are asking you to make it.

3. Flight Planning [25 points]

You have been hired to plan the flights for Professor Brunelle's brand new passenger air company, "Receding Airlines". You are going to provide service to n cities. All flights for this airline will fly exclusively northward (assume all cities are at different latitudes). Southbound travelers will need to book elsewhere.

You recognize that in order to enable all your passengers to travel from any city to any other city (to the North) with a single flight requires $\Omega(n^2)$ different routes. Prof. Brunelle says that the airline cannot be profitable when supporting so many routes. Another option would be to order the cities in a list (from South to North), and have flights that go from the city at index i, to the city at index $i+1$. This would mean that, in the worst case, passengers would need $n-1$ flights to get to their destination. Give an algorithm which produces a compromise set of flights which requires no passenger have more than a single connection (i.e. must take at most two flights), and requires $O(n \log n)$ routes.

Demonstrate your algorithm is correct by showing that the collection of flights satisfies both requirements (that no passengers would need more than 2 flights and that there are $O(n \log n)$ routes total).

4. Dog For Mayor [25 points]

A small town has hired you to design an algorithm to process the votes of their mayoral election. The mayoral election operates under the following rules. Let n be the number of votes cast in the election. Your task is to detect all candidates who received more than $n/3$ votes.

- If there are two such candidates, they will advance to a "runoff" election (a second election with just those two candidates).
- If there is exactly one such candidate, they are declared the mayor.
- If there are no such candidates, a dog is installed.

Note that there cannot be 3 or more winning candidates, as that would require the 3 of them combine for more than n votes.

The election allows for write-in candidates, which means that "Mickey Mouse", "Mickey T. Mouse", and "Micky Mouse" are all possible votes for the same candidate. As a result, you will not be able to use a hash table to accurately represent a candidate, nor will you be able to sort the array accurately.

Instead, you have written a detailed `equals()` method, which (by checking for alternative forms and spelling mistakes) will accurately decide if the two strings are really votes for the same candidate.

Design a divide and conquer algorithm that processes the election as follows

> `Election Processor`
> **Input:** A list of n strings
> **Output:** returns any string representation of all candidates that have more than $n/3$ votes (if there are any), or "Dog For Mayor!" if there is no such candidate.

Your algorithm may use the provided `equals()` method, but not hash tables, nor sorting. In analyzing your algorithm, assume that each call to the `equals()` method takes t time. Give your running time analysis in terms of t and n.

(a) Give pseudocode (and/or English) to describe a divide and conquer algorithm for this problem.

(b) Prove your algorithm is correct (you almost certainly want induction).

(c) Give the big-$\mathcal{O}$ running time for your algorithm. Briefly (3-4 sentences) justify your response. If you choose to use a recurrence, you may solve it without showing us the work (but you do not have to use one if you can explain the running time another way).

5. Binary Stars [Extra Credit]

You are not required to submit attempts at extra credit problems (they do not count toward the dropped/counted problems at the end of the quarter). They will have much *smaller effects on grades than the main problems, so we do not recommend attempting them until you've done all the other problems on the assignment. At the end of the quarter, after determining grade breaks, we will add in extra credit points.*

Binary stars form in space when two stars end up close together. The closer they are, the more likely they are to end up circling each other. Your job is the following: Given the positions of n (non-binary) stars in 3 dimensional space, determine which pair is the best candidate for forming a binary star in the future.

Use the same ideas as used to solve the problem for closest pair in the plane to create an $O(n \log^2 n)$ algorithm for finding closest pairs in 3 dimensions and prove its running time. (This is not optimal; an $O(n \log n)$ algorithm exists that uses ideas of this algorithm plus more flexibility in choosing how to split the points into sub-problems so that the difficult strip in the middle has fewer points.)