Divide and Conquer / Sorting Advanced - Written Problems

1. You are a hacker, trying to gain information on a secret array of size n. This array contains $n-1$ ones and exactly 1 two; you want to determine the index of the two in the array.

   Unfortunately, you don't have access to the array directly; instead, you have access to a function $f(l1, l2)$ that compares the sum of the elements of the secret array whose indices are in $l1$ to those in $l2$. This function returns -1 if the $l1$ sum is smaller, 0 if they are equal, and 1 if the sum corresponding to $l2$ is smaller.

   For example, if the array is $a = [1, 1, 1, 2, 1, 1]$ and you call $f([1, 3, 5], [2, 4, 6])$ then the return value is 1 because $a[1] + a[3] + a[5] = 3 < 4 = a[2] + a[4] + a[6]$. Design an algorithm to find the index of the 2 in the array using the least number of calls to $f()$. Suppose you discover that $f()$ runs in $\Theta(max(|l1|, |l2|))$, what is the overall runtime of your algorithm?

2. You are interested in finding the median salary in *Polarized County*. The county has two towns, *Happyville* and *Sadtown*. Each town mantains a database of all of the salaries for that particular town, but there is no central database.

   Each town has given you the ability to access their particular data by executing *queries*. For each query, you provide a particular database with a value k such that $1 \leq k \leq n$, and the database returns to you the k^{th} smallest salary in that town.
   You may assume the following:

   1. Each town has exactly n residents (so $2n$ total residents across both towns)
   2. Every salary is unique (i.e., no two residents, regardless of town, have the same salary)
   3. We define the *median* as the n^{th} highest salary across both towns

   Design an algorithm that finds the median salary across both towns in $\Theta(log(n))$ total queries.

3. State the complete recurrence for your algorithm. You may put your $f(n)$ in big-theta notation. Use the master theorem to show the solution for your recurrence is $\Theta(log(n))$. If your algorithm is slower, give us the actual running time.

4. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*

5. Suppose that *Netflix*, for some reason, is worried about account sharing and comes to you with n total login instances. Suppose also that *Netflix* provides you with the means only to compare the login info of two of the items in the list. By this, we mean that you can select any two logins from the list and pass them into an *equivalence tester* which tells you, in constant time, if the logins were produced from the same account.

   You are asked to find out if there exists a set of at least $\frac{n}{2}$ logins that were from the same account. Design an algorithm that solves this problem in $\Theta(nlog(n))$ total invocations of the *equivalence tester*.