problem 1 *Master Theorem*

Use the master theorem to solve the following recurrence relations. State which case of the theorem you are using and why.

1. $T(n) = 2T(\frac{n}{4}) + 1$
2. $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$
3. $T(n) = 2T(\frac{n}{4}) + n$
4. $T(n) = 2T(\frac{n}{4}) + n^2$

Solution:

1. Case 1: $n^{log_4(2)} = n^{\frac{1}{2}}$
2. Case 2: $n^{\frac{1}{2}} \cdot log n$
3. Case 3: n
4. Case 3: n^2

problem 2 *Median Salaries*

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:

- Each town has exactly n residents (so $2n$ total residents across both towns)
- Every salary is unique (i.e., no two residents, regardless of town, have the same salary)
- 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.

Solution:

1. let s be the start index (initially 1) and e be the end index (initially n) of interest. These are given as parameters to our function (note that there are actually two of each (one for happyville and one for sadtown).
2. choose the $k = \frac{(s+e)}{2}$ highest item from each respective database (technically you want the $k+1$ salary if $e-s$ is even and the $\lceil k \rceil$ if $e-s$ is odd). Note that these are the median salaries for each respective town. Let's call these values A and B.
3. Compare A and B (and suppose that $A < B$. We are going to ignore the half of the first list with values below A and the values in the second list with values greater than B. Notice that if $e-s$ is even, we need to remove B as well (see proof below for reasoning).
4. recursively call find median with left indices set to $s = 1$ and $e = k$ and the right indices set to $s = k$ and $e = n$. Note that if $e-s$ was even, then the right call has $s = k+1$.
5. the base case is when there are $n = 1$, and we query the database twice and simply select the larger of the two items as our median.

Proof (students do not need to prove just including this for our reference):
We do this by induction on the size of the lists.

Base case is $n = 1$, and we know the median is the larger of the two items, and that is what the base case of the algorithm returns! Base case proven!

I.H. We then assume the algorithm works (finds the correct median) for all $i \leq k-1$.

I.S. We must then show that it works for a list of size k. To do this we show two things:

1. The list does not remove the median in the divide step
2. the median of the new half-sized list is the same median as the original list (thus our recursive call, by inductive hypothesis, will find it correctly).

For the first one, we make sure that what we remove is not the median. We do this by using the fact that if at least half the total list size (k in this case) is greater then (or less than) the items you remove, then they do not contain the median (notice that for the upper items you remove, at least $k+1$ items must be less than those items). You define A and B as the medians of the respective lists. Assume that $A < B$, and we count the items greater than or equal to A. We know that the $\frac{k}{2}$ items above and including A in the first list are in this category, and because $A < B$, the $\frac{k}{2}$ items above and including B, are also in this category. Thus, the items below A do not contain the median (and by similar argument, the items above B as well).

For the second part (from the list above), we must have removed the same number from each end of the list (this ensures that the median of the new list is the same as the median of the whole list). If k is odd, this is not an issue (we remove $\lfloor \frac{k}{2} \rfloor$ from each side). If k is even, then we throw out the $\frac{k}{2}$ items exactly below A, and the $\frac{k}{2} - 1$ items above B along with B. Thus, the median must be in the new list and by the inductive hypothesis, the algorithm works for size k.

problem 3 *Password Sharing*

Netflix is worried about account sharing and comes to you with n total login instances. *Netflix* also 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 a *login analyzer* which tells you, in constant time, if the logins were produced from the same account. However, you DO NOT have access to the account data itself (account numbers, etc.). Netflix does not want to share that private information with you directly.

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 *login analyzer*.

Solution: The intuition here involves the following observations:

1. To get $\Theta(n * log(n))$ uses of the tester, we probably want to split into 2 sub-problems each of size $\frac{n}{2}$ and use no more than $\Theta(n)$ tests to combine the solutions.
2. At least one of the two sub-problems MUST have returned *true* in order for the combined solution to return *true*.
3. In order to make this work, we will need to return a pointer to one of account instances associated with the solution when the answer returned is *true* along with the exact count of the instances of that account. Note that because the problem specified that ”at least half” of the instances needed to be true, we may need to return at most a set of two solutions, along with a count for each.

With that being said, here is the algorithm:

1. Base case is n=1, the answer is automatically *true*, with count of 1 and pointer to the one instance in the list.
2. Otherwise, split the list in half and make a recursive call to each side. The return values have three cases.
3. Case 1: If both subproblems return $false$, then the combined solution is also $false$
4. Case 2: If one solution is *true*, then we compare the account(s) (remember it's possible the subproblem returned up to two solutions that each occur in half of the list) returned by this solution to each instance in the other half of the list. We start with the returned *count* associated with the return value and increment everytime we find a match. This is $\frac{n}{2} \in \Theta(n)$ uses of the test. If, at the end, the count is at least $\frac{n}{2}$, then we return true with the count and the pointer we already have. It's possible that we return both solutions to the subproblem as the result here.
5. Case 3: If both subproblems return *true*, we do case 2 on each one after the other.