# Divide and Conquer / Sorting Basic: Recurrence Relations

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. In class, we looked at the *Quicksort algorithm*. Consider the **worst-case scenario** for quicksort in which the worst possible pivot is chosen (the smallest or largest value in the array). Answer the following questions:

   - What is the probability of choosing one of the two worst pivots out of $n$ items in the list?
   - Extend your formula. What is the probability of choosing the one of the worst possible pivots *for EVERY recursive call* until reaching the base case. In other words, what is the probability quicksort fully sorts the list while choosing the worst pivot choice every time it attempts to do so?
   - What is the limit of your formula above as the size of the list grows. Is the chance of getting Quicksort's worst-case improving, staying constant, or converging on some other value.
   - Present one sentence on what this means. What are the chances that we actually get Quicksort's worst-case behavior?

Directly solve, by unrolling the recurrence, the following relation to find its exact solution.

3. $T(n) = T(n-1) + n$

Use induction to show bounds on the following recurrence relations.

4. Show that $T(n) = 2T(\sqrt{n}) + log(n) \in O(log(n) * log(log(n)))$. *Hint: Try creating a new variable m and substituting the equation for m to make it look like a common recurrence we've seen before. Then solve the easier recurrence and substitute n back in for m at the end.*

5. Show that $T(n) = 4T(\frac{n}{3}) + n \in \Theta(n^{log_3(4)})$. You'll need to subtract off a lower-order term to make the induction work here. *Note: we are using big-theta here, so you'll need to prove the upper AND lower bound.*

Use the master theorem (or main recurrence theorem if applicable) to solve the following recurrence relations. State which case of the theorem you are using and why.

6. $T(n) = 2T(\frac{n}{4}) + 1$

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

8. $T(n) = 2T(\frac{n}{4}) + n$

9. $T(n) = 2T(\frac{n}{4}) + n^2$