CS 4349 Lecture—September 11th, 2017

Main topics for #lecture include #divide-and-conquer, #selection, and #homework_review.

Prelude

- Last Wednesday, we discussed Quicksort, QuickSelect, and I gave the code for a Selection algorithm. We'll finish discussing Selection and divide-and-conquer today, but first, any questions about Quicksort or QuickSelect?

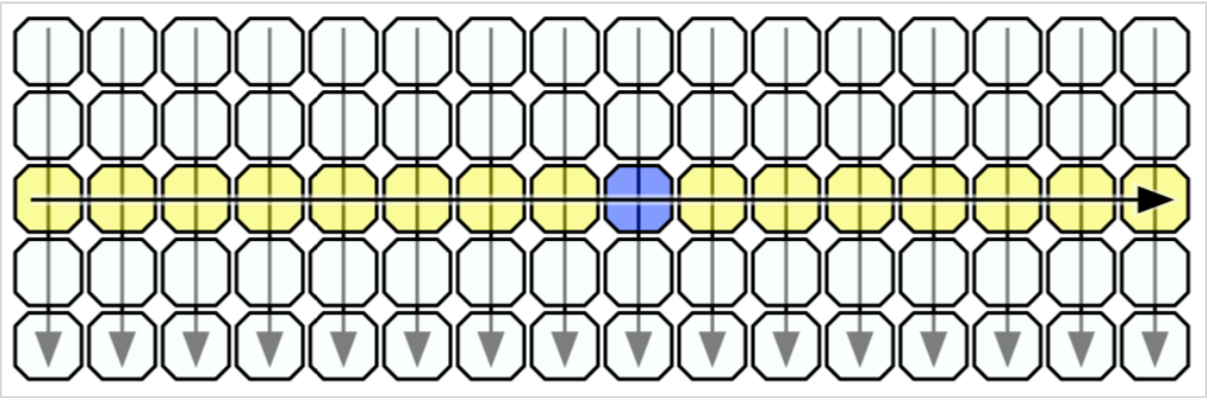
Homework Review

- Before we return to Selection, I want to discuss two of the homework problems.
- First is the tournament problem. **see official solution**
  - Some things I want to point out. Most importantly is that I fixed n as the input size *and then worked my way down*. The lemma is talking about arbitrary tournaments, so *start with an arbitrary tournament of size n*. Then when you remove the one vertex, you have tournaments of size *less than n*.
  - Even so much as writing n+1 temps you into *adding* an extra vertex and creating your own object, not the arbitrary one considered by the theorem. So don't do that.
  - Second, is that the inductive hypothesis only works for things of size *strictly* less than n. You cannot assume the theorem is true for objects of size n, that would be circulation reasoning. "The theorem is true because the theorem is true."
  - Finally, I think some of you have this pattern you're using from an earlier class where you make a hypothesis about k and then do k+1. I already explained why I don't like + 1. But also, using this pattern verbatim only gives you weak induction. And weak induction simply does not work for certain induction proofs like proofs of correctness for recursive algorithm. I'm afraid you'll *have to* get used to strong induction, i.e., making an assumption for all k strictly less than n.
- Next is the popular pet problem.
  - First off, you need to be careful to do what the question is asking for. Your algorithm should find some member of X, or some person at the party, that likes the most popular pet. Some people assumed I just wanted you to test against a given pet, but I never mentioned a given pet. Some others tried to return a count of how many people liked the most popular pet or some given pet, but again, I asked for an element of X.
  - Second, almost everybody offered an algorithm that did two nested loops that ran in quadratic time, but we can do much better using divide-and-conquer.

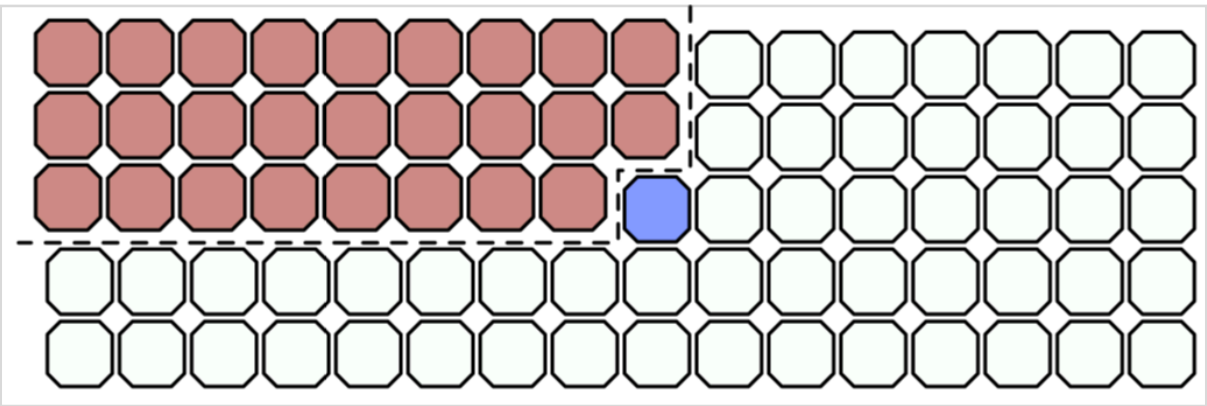
- I'll guarantee partial credit for correct solutions that are slower than the algorithm I am looking for, but you should strive to use the tools seen in class to get faster algorithms.
- **see official solution**
- Again, note that I **had** to use strong induction for this proof. With weak induction alone, I could not say anything about problems of size n/2 < n -1.

Selection

- Given an array A[1 .. n] and an integer k ≥ 1, the Selection problem asks for the element of rank k in A (or A[n] if n < k).
- MoMSelect will rearrange A and then return the index of the rank k element after A has been rearranged. The solution to the Selection problem is therefore A[MoMSelect(A[1 .. n], k)].
- For convenience, let A[n + c] = infinity
- MoMSelect(A[1 .. n], k):
  - if n ≤ 25
    - use brute force
  - else
    - m ← \ceil{n/5}
    - M ← array of length m
    - for i ← 1 to m
      - M[i] ← Median(A[5i - 4 .. 5i]) select median of five elements
    - mom ← MoMSelect(M, \floor{m / 2})
    - r ← Partition(A[1 .. n], mom)
    - if k < r
      - return MoMSelect(A[1 .. r-1], k)
    - else if k > r
      - return MoMSelect(A[r+1 .. n], k - r)
    - else
      - return mom
- So we're recursively calling MoMSelect on the medians of the five element subarrays. If T(n) is the running time of MoMSelect, that call takes T(n / 5) time.
- But what about the second call?



- Imagine we drew the array as a 5 x \ceil{n / 5} grid so each column was five consecutive elements. Then we sort every column **independently** from top down, and then sort the columns by their median elements from left to right.
- The algorithm does not do this!
- So here's the median of these medians, right in the middle.
- There are \ceil{\ceil{n / 5} / 2} - 1 lessor medians to its left. That's about n/10. And each of those lessor medians is at least as large as 3 elements from their column. So, the median of medians is larger than about 3 * n / 10 elements.



- If k > r, then those 3n / 10 elements do not appear in the second recursive call. That call takes T(7n / 10) time.
- Symmetrically, there are about 3n / 10 elements that are bigger than the median of medians. If k < r, then those elements don't appear in the recursive call which again takes T(7n / 10) time.
- So T(n) ≤ T(n / 5) + T(7n / 10) + cn for some constant c.
- **Can anybody suggest a way to solve this recurrence?**
- We'll use recursion trees.
- The root gets cn. The row at depth i sums to (9 / 10)^i * cn.
- The full rows are decreasing geometric series, and the less than full rows sum to values even smaller. The largest term dominates so T(n) = O(n).
- That said, this is not a practical algorithm unless n is several million. Otherwise, you may as well sort the array in O(n log n) time and then pick the kth element. Or just run QuickSelect using some reasonable heuristic for the pivot and hope for the best.
- Now, there is a way to pick pivots that is both practical and theoretically sound, though.

Pick an element of A uniformly at random. No matter what A is, the *expected* running time of Quicksort will be O(n log n), the expected running time of QuickSelect will be O(n), and the time to pick a pivot will be barely worse than running one of the common heuristics. We might come back to this later in the semester.

Fibonacci Numbers

- Let's look at a particular recurrence function that you've probably seen before. F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for all n ≥ 2.
- These are the Fibonacci numbers. And here's a problem: given n, compute F(n).
- There's a relatively simple recursive algorithm to do so
- RecFibo(n):
  - if n < 2
    - return n
  - else
    - return RecFibo(n-1) + RecFibo(n-2)
- But how long does this algorithm take? Let T(n) be the running time of the algorithm. We'll just count the number of times RecFibo is called.
- T(0) = 1, T(1) = 1, and T(n) = T(n-1) + T(n-2) + 1 for all n ≥ 2.
- That looks a lot like the Fibonacci recurrence! And yes, T(n) = 2F(n+1) -1.
  - True for n = 0 and n = 1.
  - Assuming it is true for T(k), k < n, we have T(n) = 2F(n) - 1 + 2F(n-1) - 1 + 1 = 2F(n+1) - 1.
- But using more complicated techniques, we can show T(n) = Theta(phi^n) where phi = (sqrt(5) + 1)/2 ~= 1.618. It takes exponential time to compute F(n)!
- The problem here is that we're repeating a ton of work. If we make a tree just listing the recursive calls, the calls for smaller values of n are done over and over again.
- Wednesday, we'll learn a better way.