CPSC 320 Notes: Physics, Tug-o-War, and Divide-and-Conquer

February 10, 2015

In tug-o-war, two teams face each other and **carefully** pull on a **well-selected** rope (to avoid injury). The team that pulls the other past the centerline wins.

1 Algorithmic Tug-o-War

1. Given $2n$ people, you decide make these two tug-o-war teams {heaviest, 3rd heaviest, 5th heaviest, ...} and {2nd heaviest, 4th heaviest, 6th heaviest, ...} using this algorithm: "While people remain, scan the line of people for the heaviest and move that person into Team A then scan the line for the heaviest (remaining) and move that person onto Team B."

   Give a good asymptotic bound on the runtime of this algorithm.

2. Give a more efficient algorithm and analyse its asymptotic runtime.

3. Now imagine you are analysing weight measurements and want to find the median (the $\lceil \frac{n}{2} \rceil$-th largest).

   (a) Modify the first algorithm above to solve this problem and give a good asymptotic bound on its runtime.

   (b) Modify your algorithm above to solve this problem and give a good asymptotic bound on its runtime.

2 Analysing QuickSort

Remember the QuickSort algorithm:

```
// Note: for simplicity, we assume all elements of A are unique
QuickSort(list A):
  If length of A is greater than 1:
    Select a pivot element p from A
    Let Lesser  = all elements from A less than p
    Let Greater = all elements from A less than p
    Let LesserSorted  = QuickSort(Lesser)
    Let GreaterSorted = QuickSort(Greater)
    Return the concatenation of LesserSorted, [p], and GreaterSorted
  Else:
    Return A
```

1. Assuming that QuickSort somehow always selects the $\lceil \frac{n}{4} \rceil$-th largest element as its pivot, give a recurrence relation for the runtime of QuickSort.

2. Draw a recursion tree for QuickSort labeled by the amount of time taken by each recursive call and the total time for each "level" of calls, both in terms of the list's length n. (For simplicity, ignore ceilings, floors, and the effect of the removal of the pivot element on the list sizes in recursive calls.)

3. Find:

   (a) the number of levels in the tree up to the shallowest leaf (base case), and

   (b) the number of levels in the tree up to the deepest leaf.

4. Use these to asymptotically upper- and lower-bound the solution to your recurrence. (Note: if, on average, QuickSort takes two pivot selections to find a pivot at least this good, then your upper-bound also upper-bounds QuickSort's average-case performance.)

5. Draw the **specific** recursion tree generated by `QuickSort([6, 2, 3, 7, 9, 1, 8, 4, 5])`. Assume QuickSort: (1) selects the first element as pivot and (2) maintains elements' relative order when producing `Lesser` and `Greater`.

3 Tug-o-War Winner (for Median)

1. Circle only those branches in your specific recursion tree in which the median appears.

2. How could you determine **before** making `QuickSort`'s recursive calls whether the median is the pivot or appears in the left or right branch?

3. Modify `QuickSort` to make it a median-finding algorithm. (Oddly enough, you'll want to make it do **more** than only find medians.)

4. Give a good asymptotic bound on the average-case runtime of your algorithm by summing the runtime of only the $\frac{3n}{4}$ branches of your abstract recursion tree.

4 Challenge

Note: assume all elements are unique for the first two problems below.

1. Compare the average-case performance of your `QuickSelect` algorithm above against the performance of one that picks a random pivot (rather than using the first element).

2. Explain how this statement can possibly make sense for the random-pivot version of `QuickSelect`: "There is no difference between the best- and worst-case performance of this algorithm." Note: we instead use "expected performance" to describe this scenario.

3. Which one is better and why: good average-case performance or good expected performance?

4. Compare the actual (not asymptotic) number of comparisons made by the standard algorithm for finding the largest element of a list and `QuickSelect` used to do the same, assuming `QuickSelect` always "gets lucky" and picks the median of what remains as its pivot. P.S. Don't look here until after class, but more about the physics of tug-o-war and the reason for all the cautionary notes at the start are at what-if.xkcd.com/127.