Homework 1

Carl Eastlund

Due **Wed., Jan. 30** at **9:00pm**.

Collaboration Policy: Your work on this assignment must be your own. You *may not* copy files from other students in this class, from people outside of the class, from the internet, or from any other source. You *may not* share files with other students in this class.

You *may* discuss the problems, concepts, and general techniques used in this assignment with other students, so long as you do not share actual solutions.

If you are in doubt about what you *may* and *may not* do, ask the course instructor before proceeding. If you violate the collaboration policy, you will receive a zero as your grade for this entire assignment and you will be reported to OSCCR (`northeastern.edu/osccr`).

1. A *selection* algorithm chooses the kth-from-smallest element in a list. For instance, $\mathsf{select}(0,[20,10,50]) = 10$ because 10 is the smallest element; $\mathsf{select}(1,[20,10,50]) = 20$ because 20 is one from the smallest element; and $\mathsf{select}(2,[20,10,50]) = 50$ because 50 is two from the smallest element. The *quickselect* algorithm performs selection using a method similar to quicksort.

   (a) Implement quickselect in Racket. Given a list ℓ of length n and an index k such that $0 \le k < n$, quickselect performs the following steps:

      i. First, select a pivot element p; for our purposes, use the first element of the list.
      ii. Second, split the list into two partitions: ℓ_1, containing all elements less than p, and ℓ_2, containing all elements greater than p.
      iii. Third, compute the lengths n_1 and n_2 of ℓ_1 and ℓ_2, respectively.
      iv. Fourth, if $k < n_1$, then the kth-from-smallest element of ℓ is the kth-from-smallest element of ℓ_1. Recursively compute and return that value.
      v. Fifth, if $k \ge n - n_2$, then the kth-from-smallest element of ℓ is the $(k-[n-n_2])$th-from-smallest element of ℓ_2. Recursively compute and return that value.
      vi. Otherwise, the kth-from-smallest element of ℓ is p itself. Return p.

   (b) Like quicksort, quickselect has worst-case performance when the pivot is consistently chosen poorly: one partition has $n-1$ elements, and the desired index is in that partition.

      i. State the worst-case running time of quickselect as a recurrence.
      ii. Solve the recurrence using the master method, recursion trees, or summations.

   (c) Again like quicksort, quickselect has best-case performance when the pivot is consistently chosen well: both partitions have $\lfloor n/2 \rfloor$ elements.

      i. State the best-case running time of quickselect as a recurrence.
      ii. Solve the recurrence using the master method, recursion trees, or summations.

2. The *stooge sort* algorithm is named for the Three Stooges comedy bit where each member hits the other two. The algorithm first swaps the first and last elements if they are out of order—*i.e.*, if the first is greater than the last. Given a list of length three or more, the algorithm then splits the list into thirds. The algorithm proceeds by sorting the first two thirds, then the last two thirds, and finally the first two thirds again.

(a) Implement stooge sort in Racket.

(b) State the running time of stooge sort as a recurrence.

(c) Solve the recurrence using the master method, recursion trees, or summations.

3. For each pair of functions $f(n)$ and $g(n)$ below, state whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, or both. Recall that you can prove $f(n) \notin O(g(n))$ by proving $f(n) \in \omega(g(n))$. Similarly, recall that you can prove $f(n) \notin \Omega(g(n))$ by proving $f(n) \in o(g(n))$.

   (a) $f(n) = 5n^{1.25} + 3n\log n + 2n\sqrt{n}$

      i. $g(n) = n^2$

      ii. $g(n) = n^{3/2}$

      iii. $g(n) = n\log n$

      iv. $g(n) = n$

   (b) $f(n) = n(\log n/2)^2$

      i. $g(n) = n$

      ii. $g(n) = n\sqrt{n}$

      iii. $g(n) = n(\log n)^2$

      iv. $g(n) = n\log n$

   (c) $f(n) = 2^2 n$

      i. $g(n) = n^{65,536}$

      ii. $g(n) = 2^n$

      iii. $g(n) = 3^n$

      iv. $g(n) = 4^n$

      v. $g(n) = 5^n$

      vi. $g(n) = n!$

4. Solve each recurrence below using the master method, or show that the master method does not apply.

   (a) $T(n) = 5T(1/4\,n) + 5/4^n$

   (b) $T(n) = 9T(1/3\,n) + n^2\sqrt{\log n}$

   (c) $T(n) = 2T(1/4\,n) + \sqrt[3]{n}$

   (d) $T(n) = 2T(2/3\,n) + (\log n)^2$