"CS 374" Fall 2015 — Homework 4

Due Tuesday, October 6, 2015 at 10am

••• Some important course policies •••

- **You may work in groups of up to three people.** However, each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the names and NetIDs of each person contributing.
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use. See the academic integrity policies on the course web site for more details.
- **Submit your pdf solutions in Moodle.** See instructions on the course website and submit a separate pdf for each problem. Ideally, your solutions should be typeset in LaTex. If you hand write your homework make sure that the pdf scan is easy to read. Illegible scans will receive no points.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem. Yes, we are completely serious.
  - Give complete solutions, not just examples.
  - Declare all your variables.
  - Never use weak induction.
- Unlike previous editions of this and other theory courses we are not using the "I don't know" policy.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

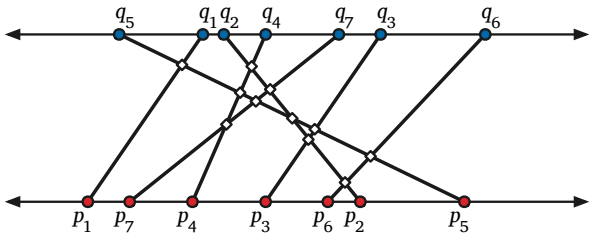
---

1. It is common these days to hear statistics about wealth inequality in the United States. A typical statement is that the the top 1% of earners together make more than ten times the total income of the bottom 70% of earners. You want to verify these statements on some data sets. Suppose you are given the income of people as an n element *unsorted* array A, where $A[i]$ gives the income of person i.

   - Describe an $O(n)$-time algorithm that given A checks whether the top 1% of earners together make more than ten times the bottom 70% together. Assume for simplicity that n is a multiple of 100 and that all numbers in A are distinct. Note that sorting A will easily solve the problem but will take $\Omega(n \log n)$ time.
   - More generally we may want to compute the total earnings of the top α% of earners for various values of α. Suppose we are given A and k numbers $\alpha_1, \alpha_2, \ldots, \alpha_k$ each of which is a number between 0 and 100 and we wish to compute the total earnings of the top α_i% of earners for each $1 \leq i \leq k$. Describe an algorithm for this problem that runs in $O(n \log k)$ time. Note that sorting will allow you to solve the problem in $O(n \log n)$ time but when $k \ll n$, $O(n \log k)$ is faster. Note that an $O(nk)$ time algorithm is relative easy. *Hint:* first sort the α_i's.

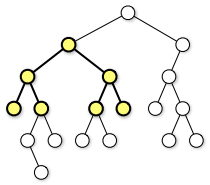
   You should prove the correctness of the second part of the problem. It helps to write a recursive algorithm that you can use induction to prove the correctness of.

2. Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and another set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connecting each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. See example below.



   No proof of correctness necessary but you should justify the running time.

3. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a **recursive** algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return both the root and the depth of this subtree. You do not need to describe or analyze a memoized version of the algorithm. Briefly justify the correctness of your algorithm by explaining the idea behind the recursion.



The largest complete subtree of this binary tree has depth 2.