Homework Four, for Thu 10/22

CSE 101

Prepare a PDF file in which your solution to each of the following problems (1–6) begins on a fresh page. Upload the file to Gradescope, using your campus email address as login. The deadline is 11pm on Thursday.

These problems cover the following skills and concepts:

- Writing down a recurrence relation for the running time of a divide-and-conquer algorithm
- Solving a recurrence relation by expanding out the terms
- Applying the recurrence methodology to analyzing the expected and worst-case running times of randomized algorithms
- Familiarity with canonical divide-and-conquer procedures such as binary search and mergesort
- Formulating a divide-and-conquer solution for a suitable new problem, given hints

---

1. *Formulating and solving recurrences.* Suppose you are choosing between the following four algorithms:

   - Algorithm A solves problems by dividing them into three subproblems of one-third the size, recursively solving each subproblem, and then combining the solutions in linear time.
   - Algorithm B solves problems of size n by recursively solving a subproblem of size $n-1$, and then performing additional computation in linear time.
   - Algorithm C solves problems of size n by recursively solving two subproblems of one-third the size, and then combining the solutions in $O(n^2)$ time.
   - Algorithm D solves problems of size n by recursively solving five subproblems of size $n/4$ and combining the results in linear time.

   What are the running times of each of these algorithms? Which would you choose?

2. *Analyzing the number of calls of a simple recursive procedure.* How many times does the following program print "hello", as a function of n? Assume n is a power of two, and leave your answer in big-O form.

   ```
   function f(n)
      if n > 1:
         print(''hello'')
         f(n/2)
         f(n/2)
         f(n/2)
   ```

3. *The worst-case running time of quicksort.* Textbook problem 2.24(a,b).

4. *Randomized binary search.* Consider the following problem.

   > You are given a *sorted* array of numbers $S[1 \ldots n]$ as well as a number x. You want to determine whether S contains x.

   The usual algorithm for this problem is *binary search*, which, you will recall, works roughly as follows:

   - Query the midpoint of S. If it is x, halt.
   - Otherwise, eliminate either the top or bottom half of S, as appropriate, and repeat.

Since the array is effectively halved in each iteration, the total running time is $O(\log n)$.

We'll now look at a *randomized* binary search, which differs in only one small detail: instead of querying the midpoint of the current array, you query a randomly chosen position in the array.

(a) Write down the resulting algorithm.

(b) What is the expected running time of this algorithm? Show how you obtain this bound.

5. *Checking for a majority element using only equality queries.* Textbook problem 2.23.

6. *Closest pair.* In this problem we will develop a divide-and-conquer algorithm for a geometric task.

   CLOSEST PAIR
   Input: A set of points in the plane, $\{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$
   Output: The closest pair of points: that is, the pair $p_i \neq p_j$ for which the distance between p_i and p_j, that is,

   $$\|p_i - p_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

   is minimized.

   For simplicity, assume that n is a power of two, and that all the x-coordinates x_i are distinct, as are the y-coordinates.

   Here's a high-level overview of the algorithm:

   i. Find the median value v of the x-coordinates $x_1, \ldots, x_n$. Split the points into two groups: L, with x-coordinate $\leq v$, and R, with x-coordinate $> v$.

   ii. Recursively find the closest pair in L and in R. Say these pairs are $p_L, q_L \in L$ and $p_R, q_R \in R$, with distances d_L and d_R respectively. Let d be the smaller of these two distances.

   iii. It remains to be seen whether there is a point in L and a point in R that are less than distance d apart from each other.
   - Discard all points with $x_i < v - d$ or $x_i > v + d$.
   - Sort the remaining points by y-coordinate.
   - Go through this sorted list, and for each point, compute its distance to the *seven* subsequent points in the list.
   - Let p_M, q_M be the closest pair found in this way.

   iv. The answer is one of the three pairs $\{p_L, q_L\}$, $\{p_R, q_R\}$, $\{p_M, q_M\}$, whichever is closest.

   (a) In order to prove the correctness of this algorithm, start by showing the following property: any square of size $d \times d$ in the plane contains at most four points of L. *Hint:* Divide this square into four smaller squares. Do you see why each smaller square contains at most one point?

   (b) Now show that the algorithm is correct. The only case which needs careful consideration is when the closest pair is split between L and R.

   (c) Show that the running time of the algorithm is given by the recurrence:

   $$T(n) = 2T(n/2) + O(n \log n).$$

   The solution to this recurrence is $O(n \log^2 n)$ (you don't have to show this; in fact, with a little more care the running time can be reduced to $O(n \log n)$).