Problem Set 2

This problem set is due at 11:59pm on Monday, March 4, 2013.

Both exercises and problems should be solved, but only the problems should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with the following: (1) your name, (2) the name of your recitation instructor, and the time your recitation section meets, (3) the question number, (4) the names of any people you worked with on the problem, or “Collaborators: none” if you solved the problem completely alone.

Each problem must be turned as a separate PDF file to Stellar.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph or sentence should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. A proof of the algorithm’s correctness.
3. An analysis of the algorithm’s running time.
4. If appropriate, a worked example or diagram to show more precisely how your algorithm works.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Amortized Analysis

Exercise 2-1. Do Exercise 17.3-6 in CLRS on page 463.

Exercise 2-2. Do Exercise 17.3-7 in CLRS on page 463.

Randomized Algorithms

Exercise 2-3. Do Exercise 7.3-2 in CLRS on page 180.

Exercise 2-4. Do Exercise 7.4-4 in CLRS on page 185.

Problem 2-1. Points on a Circle

In this problem you will design a data structure for operating on a set of points on the unit circle. For simplicity, you may assume that the circle is centered at \((0, 0)\), has radius 1, and that all points are specified by the angle (in radians) that a line connecting them to the origin makes with the x-axis. For example, the point \((0, 1)\) would be represented by \(\pi/2\), and \((-1, 0)\) would be represented by \(\pi\). Also assume that all basic arithmetic operations can be performed in \(O(1)\) time.

The operations your data structure should support are the following:

- **INSERT**\((X)\): Inserts a point at angle \(X\) radians to the x-axis.
- **ROTATE**\((R)\): Rotates all points \(R\) radians counter-clockwise.
- **CLEAR-CLOCKWISE**\((Y)\): If there are currently \(n\) points in the data structure, removes the first \(\lceil n/2 \rceil\) points found by starting at \(Y\) radians to the x-axis and moving clockwise.

(a) Describe a data structure that implements **INSERT** and **ROTATE** that runs in worst-case time \(O(1)\).

(b) Describe an implementation of **CLEAR-CLOCKWISE** that runs in worst-case time \(O(n)\).

(c) Show that, as long as your implementation of **INSERT** and **ROTATE** have worst-case runtime \(O(1)\) and your implementation of **CLEAR-CLOCKWISE** has worst-case runtime \(O(n)\), that any sequence of \(m\) operations performed by your data structure has worst-case runtime \(O(m)\) (and therefore the amortized cost of each operation is \(O(1)\)).

Problem 2-2. Recursive Electoral Colleges

MIT’s new Committee on Committee Recursion is running a two-party election: EE versus CS. Assume that the population size \(n = 5^k\) is a power of 5. Create a quinary tree of height \(k\), where each internal node has exactly five children and each leaf corresponds to a unique voter. Define the value of each leaf to be the respective voter’s choice of candidate, and the value of each internal node to be the majority of the its five children’s values. The overall winner is the value of the root.

Consider the following randomized algorithm for determining the winner. At the top level, we call **COUNT-VOTES**\((\text{root})\). At each recursive level, the procedure **COUNT-VOTES**\((\text{node})\) randomly selects three out of the five children of \(\text{node}\). If those children’s values all agree (which is determined recursively), then the algorithm returns this common preference, because it must be the majority of all five children. Otherwise, the algorithm recursively determines the preference of a fourth random child from the two remaining. If three out of the four children agree, then the algorithm returns this majority, because again it must be the majority of all five children. Otherwise, the algorithm recursively examines the remaining fifth child, and declares the majority preference to be the winner of this subtree. More precisely, the recursion works as follows:
COUNT-VOTES(node)

1 if node is a leaf
2 then return node.vote
3 child₁, child₂, child₃ ← three randomly selected children of node (among the five)
4 choice₁, choice₂, choice₃ ← COUNT-VOTES(child₁), COUNT-VOTES(child₂), COUNT-VOTES(child₃)
5 if choice₁ = choice₂ = choice₃
6 then return choice₁
7 child₄ ← a randomly selected child from the two unpicked children of node
8 choice₄ ← COUNT-VOTES(child₄)
9 if three out of choice₁, choice₂, choice₃, choice₄ agree
10 then return majority of {choice₁, choice₂, choice₃, choice₄}
11 child₅ ← remaining unpicked child of node
12 choice₅ ← COUNT-VOTES(child₅)
13 return majority of {choice₁, choice₂, choice₃, choice₄, choice₅}

(a) Analyze the worst-case expected asymptotic running time of COUNT-VOTES(root) as a function of the population size n.

Now instead of studying the worst case, you will analyze the average case. Suppose that each voter decides independently of the others whether to vote for CS or EE, and that each voter chooses CS (independently) with probability p.

(b) What is the asymptotic probability (as n goes to infinity) that COUNT-VOTES(root) will return CS as the winner? How does the answer change depending on p?

Hint: Write a recurrence to determine the limit probability, and use the following fact about fixed points (no proof necessary). Consider the iteration $x_0, x_1, x_2, \ldots$ where $x_{i+1} = f(x_i)$. If $f : [0, x_0] \to [0, x_0]$ is continuous and has a fixed point at 0 (i.e. $f(0) = 0$), and $f(y) < y$ for all $y \in (0, x_0]$, then the iteration $x_i$ converges to 0.

(c) Analyze the average asymptotic running time of COUNT-VOTES(root) as a function of the population size n. You should analyze the case $p = \frac{1}{2}$ separately from the rest, and otherwise should treat p as an absolute constant (i.e., p need not appear in the running time).

Problem 2-3. Windowing Queries in 1D and 2D

In computer graphics, the problem of windowing queries asks to find all visible objects inside a view rectangle. For example, one might want to find all roads on a computerized map inside a rectangular window. In this problem, we will construct static data structures to support windowing queries in 1D and 2D, respectively.

In 1D, both the objects and the query window can be modeled as horizontal segments. Under this model, we are given $n$ horizontal segments $S_1 = [a_1, b_1], S_2 = [a_2, b_2], \ldots, S_n = [a_n, b_n]$, and we want to build a static data structure on them that can efficiently output all segments overlapping a given query segment $Q = [a, b]$. 
Problem Set 2

Inspired by the divide-and-conquer algorithms in this class, you decide to compute the median $x_{\text{mid}}$ of the multiset of endpoints $\{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$, and divide the segments into three sets:

- $L$: all segments $S_i$ lying completely left of $x_{\text{mid}}$ (i.e., for which $b_i < x_{\text{mid}}$).
- $R$: all segments $S_i$ lying completely right of $x_{\text{mid}}$ (i.e., for which $a_i > x_{\text{mid}}$).
- $M$: all segments $S_i$ that intersect $x_{\text{mid}}$ (i.e., for which $a_i \leq x_{\text{mid}} \leq b_i$).

Then you build a binary tree whose root stores $x_{\text{mid}}$ and a list of the segments in $M$, whose left subtree recursively represents $L$, and whose right subtree recursively represents $R$.

(a) Modify this data structure to implement 1D window queries in $O(\lg n + k)$ time, where $k$ is the number of overlapping (output) segments.

$Hint$: 1. Organize the segments in $M$ into two lists so that, if $m$ segments in $M$ intersect with $Q$, we can find and report them in $O(m)$ time. 2. Prove that the height of this binary tree is $O(\log n)$.

In 2D, for simplicity, we model the objects as either horizontal or vertical segments, and the query window as an axis-aligned rectangle $[x_1, x_2] \times [y_1, y_2]$.

(b) Given $n$ vertical and horizontal segments in the plane, and given a query rectangle, design a static data structure of size $O(n \log n)$ supporting the query of finding all $k$ segments intersecting the query rectangle in $O(\log^2 n + k)$ time.

$Hint$: Use part (a), range trees, and/or augmentation.