COP 6405: Analysis of Algorithms MIDTERM REVIEW; Fall 2019

---

1. **Short Questions**

   (a) Prove or disprove
   $$3n(\log n)^2 + 4n = O(2n^2 \log n + 1).$$

   (b) Prove or disprove
   $$3n(\log n)^2 + 4n = \Omega(2n^2 \log n + 1).$$

   (c) Solve using any of the 3 methods discussed in class:
   $$T(n) = \frac{10}{9}T(4n/5) + O(n)$$

   (d) Solve using any of the 3 methods discussed in class:
   $$T(n) = T(4n/7) + O(1)$$

   (e) Insert the following integer values into an initially empty red-black tree.
   $$7, 17, 23, 6, 5, 27, 31, 3, 4, 32$$

2. For my birthday, I got a cake 2 feet in length and of small width. The cake could only be cut perpendicular to its longest side. I had n people at the party excluding me. My nerdy friends devised a strange way to divide the cake. Each person at the party (excluding me) wrote down a real number between 0 and 2.0 and I had to make a cut of the cake at that distance from the left end of the cake (one of the two ends was designated as the "left end" of the cake). At the end of this process I had made n cuts leaving $n + 1$ pieces. Since it was my birthday, I got to eat the largest of the $n + 1$ pieces. For example, if $n = 4$, and the cuts were made at 0.6511123, 1.3, 0.454545, and 1.99, then the longest of the 5 pieces would be of length 0.69 feet. Given as input the real numbers written down by each of the n people at the party, design an algorithm that outputs the length of the piece that I consumed. Analyze its time complexity.

3. I have the GPA (real number) of all the n students in the class. The dean has asked me to identify the k students from this class (k is an integer in the range $[0..n-1]$) with the lowest GPA and to provide the list in the order of increasing GPA. Your task is to design an efficient algorithm that takes as input the n GPA values and the integer k, and outputs the required information. Analyze its time complexity (in terms of n and k). k is an arbitrary number from $[0..n-1]$. Consider 3 cases as shown:

   (a) $k = O(1)$

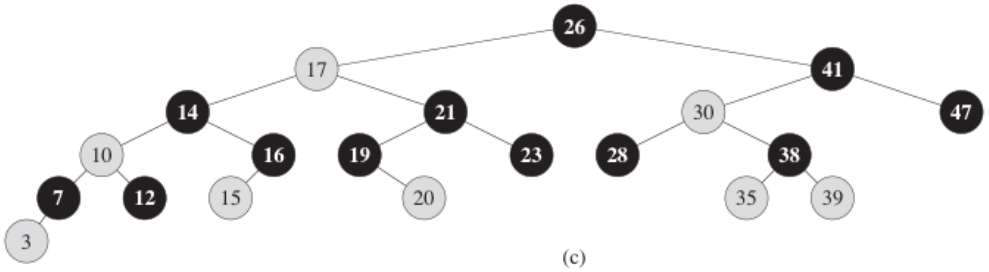
   (b) $k = O(\sqrt{n})$

   (c) $k = O(n)$

4. You have been asked by your insane boss to design an augmented RB-tree that (a) stores integer key values in its nodes and (b) efficiently answers a sequence of queries from the following set:

   - RB-INSERT(T, z) – inserts node z in tree T
   - RB-DELETE(T, z) – deletes mode z from tree T
   - OS-SOODD(x, y) – given node x, computes the number of odd values stored in the subtree rooted at y that are smaller than the key stored in node x.

   You may assume that the tree has no duplicate key values. If your current tree looks like the RB-tree shown below and node x is the node with key value 20, then OS-SOODD$(x, \text{ROOT}(T))$ should return 5 because the key values 3, 7, 15, 17 and 19 are the only 5 odd numbers smaller than 20 stored in the tree. If y points to the node with key value 21, then OS-SOODD(x, y) should return 1 since only key value 19 is counted.



(c)

   (a) State what **augmented** piece(s) of information you will store in each node (besides KEY, LEFT, RIGHT, COLOR) in order to facilitate the above queries.

   (b) Explain very briefly why this information can be maintained in the presence of inserts and deletes in the RB-tree.

   (c) Using the augmented pieces of information from above write down an efficient algorithm for OS-SOODD(x, y). The classic OS-RANK from class is provided for your benefit. Feel free to directly modify the pseudocode below.

OS-RANK(x, y)
▷ Finds rank of node x in subtree rooted at node y
▷ Recursive version discussed in class
▷ RB tree stores KEY, LEFT, RIGHT, COLOR in each node
▷ Every node is augmented with SIZE information

```
1: r ← SIZE[LEFT[y]] + 1
2: if (x == y) then return r
3: else if (KEY[x] < KEY[y]) then return OS-RANK(x, LEFT[y])
4: else return r + OS-RANK(x, RIGHT[y])
5: end if
```