CMPS 256 — ADVANCED ALGORITHMS AND DATA STRUCTURES
Fall 2012 – 13
Quiz 2, Section 2
Monday December 17, 10:00 – 10:50 a.m.

Instructions.
This quiz is scored out of 100.
This quiz is open book and open notes: you can use the textbook, notes you have taken in class, and your homework solutions.
Show your work, as partial credit will be given.
You may use any algorithm that was covered in class. Just give the name of the algorithm and the chapter or page number in the book where the algorithm is given. If taken from your lecture notes, just say "lecture notes."

Please draw a horizontal line to separate each answer from the next.
Best of luck!

Problem 1 (30 points, 15 points each).

You are given a red black tree T that is also a complete binary tree of height h, i.e., the distance from the root of **every** leaf is h.

Part a, (15 points). What is the largest number of black edges that T could possibly contain?

Part b, (15 points). In this case (maximum number of black edges), how many nodes does the corresponding 2-3 tree contain?

Problem 2 (30 points).

Consider following argument.

From problem set 5, question 5, We know how to merge two BST's in time $O(n)$.

We will now iterate this algorithm to create a BST from n items in total running time $O(n)$. This will let us sort an array of size n in time $O(n)$, therefore beating the lower bound of $\Omega(n \lg n)$.

1. Let A be an array of size n.
2. We start by considering each element of A to be an single-node BST.
3. We repeatedly merge pairs of BST's, until there is a single BST. Since most of the time that we do this, the BTS's are "small", the total running time will therefore be $O(n)$.

Prove that this argument is incorrect by showing thast the total running time is asymptotically greater than cn for any constant c.

Problem 3 (40 points).

You are given an unsorted array A of size n, with all elements distinct.

If x, y are elements of A, let $|x^2 - y^2|$ be the distance between x and y.

Part a, (30 points). Write an algorithm with worst case running time in $O(n \lg n)$, which finds the smallest distance in A, taken over all pairs of elements, i.e., it returns d where

$$d = (\text{MIN } i, j : 0 \leq i, j < n \wedge i \neq j : |A[i]^2 - A[j]^2|)$$

Grading is as follows:

- 10 points for the code itself, provided that it is actually correct and runs in time $O(n \lg n)$
- 10 points for an informal argument which proves that your code is correct, i.e., actually computes d
- 10 points for a running time analysis which proves that your code runs in time $O(n \lg n)$

Part b, (10 points). Now write an algorithm that will update d correctly when a single new element is added to the collection. This should run in time $O(\lg n)$.

Sample solutions to the quiz.

Problem 1 (30 points, 15 points each).

You are given a red black tree T that is also a complete binary tree of height h, i.e., the distance from the root of **every** leaf is h.

Part a, (15 points). What is the largest number of black edges that T could possibly contain?

Sample solution. It is possible to have no red edges in an LLRB. In this case the corresponding 2-3 tree consists entirely of 2-nodes, and has the same height h, and is therefore basically ther same tree.

The number of nodes in a complete binary tree of height h is $2^{h+1} - 1$. (We use the books definition of height to be the maximum number of edges from the root to some leaf). Hence number of black edges is $2^{h+1} - 2$, since it is one less than the number of nodes.

Part b, (15 points). In this case (maximum number of black edges), how many nodes does the corresponding 2-3 tree contain?

Sample solution. $2^{h+1} - 1$, see soln to part (a) above.

Problem 2 (30 points).

Consider following argument.

From problem set 5, question 5, We know how to merge two BST's in time $O(n)$.

We will now iterate this algorithm to create a BST from n items in total running time $O(n)$. This will let us sort an array of size n in time $O(n)$, therefore beating the lower bound of $\Omega(n \lg n)$.

1. Let A be an array of size n.
2. We start by considering each element of A to be an single-node BST.
3. We repeatedly merge pairs of BST's, until there is a single BST. Since most of the time that we do this, the BTS's are "small", the total running time will therefore be $O(n)$.

Prove that this argument is incorrect by showing thast the total running time is asymptotically greater than cn for any constant c.

Sample solution. We have to merge n BSTS, each consisting of a single node. Each merge operation merges two BST's.

Arrange the merge operations in a binary tree. The bottom level contains n leaves, for the n initial single-node BST's. The smallest height that such a binary tree can have is when it is roughly balanced: $\Omega(\lg n)$, which means every element is merged $\Omega(\lg n)$ times, giving a running time in $\Omega(n \lg n)$. If the tree is unbalanced, the running time will be larger, i.e., $\Omega(n^2)$ when the tree is a chain.

Problem 3 (40 points).

You are given an unsorted array A of size n, with all elements distinct.

If x, y are elements of A, let $|x^2 - y^2|$ be the distance between x and y.

Part a, (30 points). Write an algorithm with worst case running time in $O(n \lg n)$, which finds the smallest distance in A, taken over all pairs of elements, i.e., it returns d where

$$d = (\text{MIN } i, j : 0 \leq i, j < n \wedge i \neq j : |A[i]^2 - A[j]^2|)$$

Grading is as follows:

- 10 points for the code itself, provided that it is actually correct and runs in time $O(n \lg n)$
- 10 points for an informal argument which proves that your code is correct, i.e., actually computes d
- 10 points for a running time analysis which proves that your code runs in time $O(n \lg n)$

Part b, (10 points). Now write an algorithm that will update d correctly when a single new element is added to the collection. This should run in time $O(\lg n)$.

Sample solution.

Part a, (30 points).

Code:

```
1. Insert elements of A into an initially empty LLRB.

2. Perform an inorder traversal, computing distance between each node and its successor,
   and accumulating the minimum.
```

Correctness:
Inorder traversal yields elements in sorted order, and so successive pairs are the elements with least distance between them. Other pairs do not need to be checked. Hence minimum of distances between successive elements is the required d.

Running time:
Size of LLRB is always $\leq n$, so each insert operation runs in time $O(\lg n)$. Hence Step 1 takes time $O(n \lg n)$.

Step 2 takes time $O(n)$ since inorder traversal is $O(n)$ overall , and computing distance for each pair of elements is constant time, and so $O(n)$ overall.

Part b, (10 points). Now write an algorithm that will update d correctly when a single new element is added to the collection. This should run in time $O(\lg n)$.

Code:

```
1. Insert the new element x into the existing LLRB

2. Find predecessor and successor of x

3. Compare to current d and return smallest of the three values.
```

Step 1 runs in $O(\lg n)$ time. Step 3 is clearly contant time. Step 2 can be carried out using a constant number of rank and select operations, and so is also $O(\lg n)$. Hence total is $O(\lg n)$.