CMPS 256 — ADVANCED ALGORITHMS AND DATA STRUCTURES
Fall 2012 – 13
Quiz 2, Section 1
Monday December 17, 9:00 – 9:50 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 (20 points).

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.

What is the largest number of red edges that T could possibly contain? Give your answer as $\Theta(f(h))$, where $f(h)$ is some function of h.

Problem 2 (40 points).

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

If x, y are elements of A, let $|x - y|$ 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

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

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 $O(\lg n)$.

Problem 3 (40 points).

Consider the following code.

```
private void doWhatever(int[] A, int lf, int rt)
{
   // rt - lf + 1 is a power of 2, i.e,  rt - lf + 1 = 2^k for k >= 0

   int[] B = new int[A.length];   //auxiliary array B of same size as A

   if (rt - lf + 1 == 1) return;

   m = (lf + rt)/2;
   copy(A, lf, m, B, m+1, rt);
   copy(A, m+1, rt, B, lf, m);
   copy(B, lf, rt, A, lf, rt);
   doWhatever(A, lf, m);
   doWhatever(A, m+1, rt);
}

private void copy(int[] C, int lf, int rt, int[] D, int ll, int rr)
{
   // rt - lf + 1 = rr - ll + 1, i.e., C[lf..rt] and  D[ll..rr] have the same length

   copies C[lf..rt] to D[ll..rr];
}
```

You may assume that the running time of `copy` is $\Theta(rt - lf)$.

Part a, (20 points). State in words what the call `doWhatever(A, 0, n-1)` does, where `n = A.length`.

Part b, (20 points). Write down the recurrence for the running time of `doWhatever(A, 0, n-1)`. You do not need to solve the recurrence.

Sample solutions to the quiz.

Problem 1 (20 points). 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. What is the largest number of red edges that T could possibly contain? Give your answer as $\Theta(f(h))$, where $f(h)$ is some function of h.

Sample solution. The rightmost branch from ther root to a leaf does not contain any red edges, since edges must lean left. Thus this branch contains h black edges. By the black balance condition, all other branches (paths from the root to a leaf) also contain h black edges. Since the tree is complete and of height h, no branch can contain a red edge, since it would then have fewer than h black edges. So, T the largest number of red edges is 0, or equivalently $\Theta(0)$.

Problem 2 (40 points). You are given a collection of n elements, as an unsorted array A of size n, with all elements distinct. If x, y are elements of A, let $|x - y|$ 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

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

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 $\le n$, 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 $O(\lg n)$.

Problem 3 (40 points).

Consider the following code.

```
private void doWhatever(int[] A, int lf, int rt)
{
   // rt - lf + 1 is a power of 2, i.e,  rt - lf + 1 = 2^k for k >= 0

   int[] B = new int[A.length];   //auxiliary array B of same size as A

   if (rt - lf + 1 == 1) return;

   m = (lf + rt)/2;
   copy(A, lf, m, B, m+1, rt);
   copy(A, m+1, rt, B, lf, m);
   copy(B, lf, rt, A, lf, rt);
   doWhatever(A, lf, m);
   doWhatever(A, m+1, rt);
}

private void copy(int[] C, int lf, int rt, int[] D, int ll, int rr)
{
   // rt - lf + 1 = rr - ll + 1, i.e., C[lf..rt] and  D[ll..rr] have the same length
   copies C[lf..rt] to D[ll..rr];
}
```

You may assume that the running time of `copy` is $\Theta(rt - lf)$.

Part a, (20 points). State in words what the call `doWhatever(A, 0, n-1)` does, where `n = A.length`.
Sample solution. Reverses the array A.

Part b, (20 points). Write down the recurrence for the running time of `doWhatever(A, 0, n-1)`. You do not need to solve the recurrence.
Sample solution. $T(n) = 2T(n/2) + cn$.