UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **P. N. Hilfinger**
Fall 2011

Test #2 (with corrections)

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 8 problems on 7 pages. Officially, it is worth 20 points (out of a total of 200).

This is an open-book test. You have fifty minutes to complete it. You may consult any books, notes, or other inanimate objects available to you. You may use any program text supplied in lectures, problem sets, or solutions. Please write your answers in the spaces provided in the test. Make sure to put your name, login, and lab section in the space provided below. Put your login and initials *clearly* on each page of this test and on any additional sheets of paper you use for your answers.

Be warned: my tests are known to cause panic. Fortunately, this reputation is entirely unjustified. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious. Should you feel an attack of anxiety coming on, feel free to jump up and run around the outside of the building once or twice.

Your name: _____ Login: _____

Login of person to your Left: _____ Right: _____ Discussion TA: _____

1. _____/3 6. _____/2

2. _____/3 7. _____/4

3. _____/2 8. _____/2

4. _____/3 9. _____/1

5. _____/ TOT _____/20

1. [3 points] Consider insertion sort, merge sort, and quicksort. For each algorithm, what is the worst-case asymptotic bound on the running time for a list of N elements if you know additionally that:

- the input is already sorted?
- the input is reversely sorted?
- the input is a list containing N copies of the same number?

Condition	**Insertion Sort**	**Merge Sort**	**Quicksort**
Sorted			
Reverse sorted			
N copies			

2. [3 points] We want a data structure, `T`, on which we can perform the following operations with the given complexity (N being the number of elements already inserted in the data structure):

insert(x, T) insert a new element in $O(\lg N)$ time.

contains(x, T) check if the structure contains element x in $O(\lg N)$ time.

delete(k, T) delete the k^{th} smallest element in $O(\lg N)$ time, where we take k to be a constant.

Out of all the data structures given below, which one(s) would you pick? For any you reject, explain briefly. Simply circle those that you accept; no explanation is necessary.

- a heap
- a red-black tree
- a binary search tree
- an unsorted linked list
- a sorted linked list
- an unsorted array
- a sorted array
- a hash table (nothing said about the hash function)

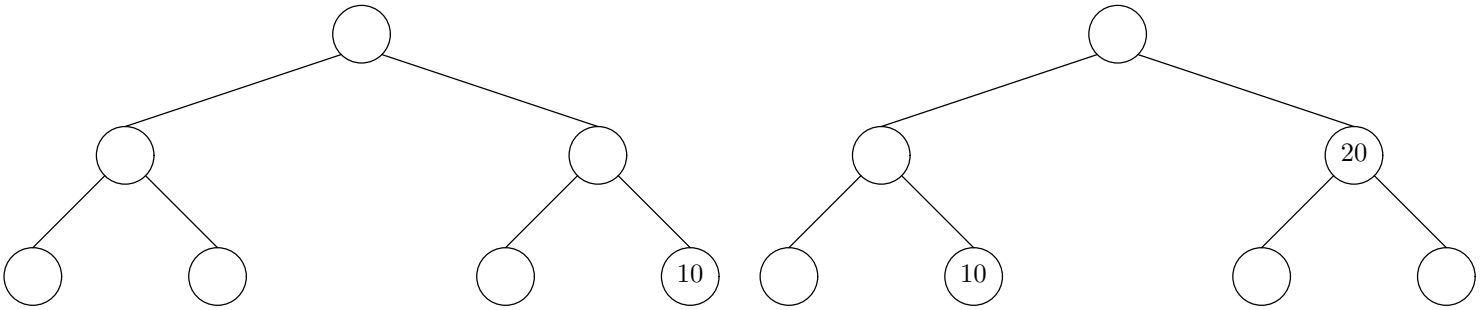
3. [2 points] In doing Project #2, Bill Bucket has written the following function:

```
/** Returns a copy of A that is unaffected by subsequent operations on
 *  A. */
public static int[][] copy(int[][] A) {
    int[][] result = new int[A.length][];
    for (int i = 0; i < A.length; i += 1) {
        result[i] = A[i];
    }

    return result;
}
```

Devise a unit test that shows that this method does not work (does not fulfill its documentation).

```
@Test
public void checkCopy() {









}
```

4. [3 points] Starting from the max-heap on the left, we remove the largest element and insert 20. The resulting heap is shown on the right. Assume that all values are unique and that 20 is not in the original heap.



Fill in the blank nodes in both heaps to show one possible way that this result could come about.

5. [1 point] What non-mathematical property do the numbers 31, 45, 66, 99, and 500 have in common?

6. [2 points] Show the steps taken by quicksort on the following unordered list, assuming that the pivot node is always the first item in the (sub)list being sorted, and the array is sorted in place.

```
36  22  15  56  48  90  72  06
```

To show a "step," show the contents of the array at the beginning of each call (initial or recursive) on the quicksort procedure. Assume that we sort all the way down to the trivial case of subarrays of size 1.

7. [4 points] A certain binary search tree representation uses the following representation

```
/** A binary tree node.  The empty tree is represented by null. */
class BinNode {
      /** Returns my left child. */
      BinNode left() { ... }

      /** Returns my right child. */
      BinNode right() { ... }

      /** Returns my parent, or null if I am the root of the entire
       *  tree. */
      BinNode parent() { ... }

      /** Returns my key value (label). */
      ValueType label() { ... }
      ...
}
```

Fill in the following functions to conform to their comments.

```
/** True iff B is the left child of another node. */
static boolean amLeftChild(BinNode b) {


    return ____________________________________________________;
}

/** Returns the node in B's tree that contains the value next larger
 *  than B's. Assumes that the tree I am part of is a binary search
 *  tree and that all its values are unique.  Returns null if G is
 *  the node containing the largest value.  Does not call label(). */
static BinNode next(BinNode b) {









}
```

8. [2 points] Consider a hash table that uses a very good hashing function—one that (miraculously) always distributes keys evenly across the bins in the table. Assuming the hash table expands at need to maintain a load factor of 3,

- What can you say about the worst-case time for adding an item to the table when it contains N items to begin with?
- What can you say about the worst-case time for adding N items to the table when it contains N items to begin with?

9. [1 point] I create a list of integers by calling `semiRandomList(B, 10)`, using the following method:

```
static void semiRandomList(int[] A, int jiggle) {
    Random R = new Random();
    for (int i = 0; i < A.length; i += 1) {
        A[i] = i + R.nextInt(jiggle);
    }
}
```

(The `nextInt(jiggle)` call produces a random integer ≥ 0 and <jiggle.) Having done this, I sort `B` using insertion sort. What can you say about the time required to do so? Be as precise as possible.