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

CS61B **P. N. Hilfinger**
Fall 2017

Final Examination (revised) Solutions

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 11 problems on 16 pages. Officially, it is worth 46 points (out of a total of 200).

This is an open-book test. You have three hours to complete it. You may consult any books, notes, or other non-responsive 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 TA 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: _____

Your SID: _____ Discussion TA: _____

Login of person to your Left: _____ Right: _____

Please sign:

I pledge my honor that during this examination, I have neither given nor received assistance.

Signature: _____

1. [2 points] Fill in the blank in the following to fulfill the comment.

```
import java.util.regex.Pattern;

/** A Java pattern that matches an expression consisting of the sum
 *  of one or more non-negative rational numbers, each expressed either as a
 *  fraction or an integer numeral.  For example, the pattern matches any of
 *      12     1/2   2/3+6   3/4+1/7+15
 *  but not
 *      -3     2//3  5+      +5
 *
 */
static final Pattern SUM =

    Pattern.compile("\\d+(/\\d+)?(\\+\\d+(/\\d+)?)*                    ");
```

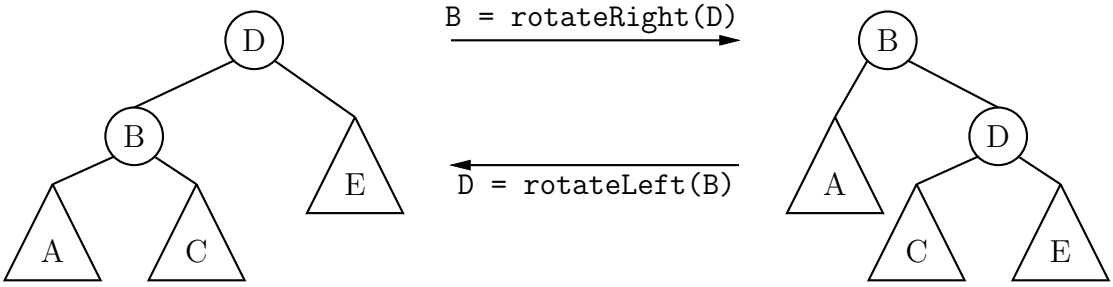
2. [2 points] Fill in the blanks in the following so that the code swaps the first (most significant) `k` bits of `x` with the last (least significant) `k` bits of `y`. If you desire, you may use the first two blank lines for additional declarations of intermediate results. Do not, however, introduce any loops, **if** statements, or function calls.

```
private static int x, y;

public static void mogrify(int k) {


    int x0 = y << (32 - k) | x << k >>> k;



    int y0 = y >> k << k | x >>> (32 - k);


    x = x0;
    y = y0;
 }
```

3. [6 points] For this question, we'll represent binary trees with the class

```
public class Tree {
    public int val;
    public Tree left;
    public Tree right;
    public Tree(int v, Tree L, Tree R) {
        val = v; left = L; right = R;
    }
}
```

a. [2 points] We've discussed *rotation* of binary trees:



Implement the `rotateRight` function.

```
/** Destructively rotate T right, returning the new
 *  tree root (whose right child's value will therefore be
 *  T's current value.)  Assumes that neither T nor its left child
 *  is null. */
public static Tree rotateRight(Tree T) {

     Tree result = T.left;

     T.left = result.right;

     result.right = T;

     return result;

}
```

b. [4 points] Implement `putOnTop`, which takes a binary search tree, `T`, and a value `V` contained in `T` and (destructively) modifies `T` into a new binary search tree containing the same values in which value `V` is at the root. It returns the new root. You may assume that `rotateLeft` and `rotateRight` are properly implemented. Your program must run in worst-case time proportional to the height of `T`.

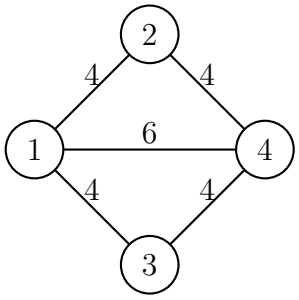
```
/** Assuming that T is a binary search tree that contains the value
 *  V, destructively modifies T so that it contains the same values
 *  as before, but has the value V at its root.  Returns this new
 *  root. */
public static Tree putOnTop(Tree T, int v) {

     if (T.value == v) {

         return T;

     } else if (T.value < v) {

         T.right = putOnTop(T.right, v);

         return rotateLeft(T);

     } else {

         T.left = putOnTop(T.left, v);

         return rotateRight(T);

     }
}
```

4. [1 point] What is the smallest possible size for the domain of a model that satisfies the first-order theory of the real numbers? **Answer:** $\aleph_0$

5. [6 points] Give short answers to the following questions.

a. Draw a weighted, undirected graph with four vertices labeled 1–4 that has multiple minimal spanning trees, but a unique shortest path tree from vertex 1.

Answer:



b. For an optimal algorithm, what is the worst-case number of comparisons required to find the second smallest value in a max heap with $2^n - 1$ values, all distinct (without modifying the heap)? Assume that the heap is represented by an array and that the algorithm can use array operations on it. We're asking for a precise formula involving n, not an asymptotic bound.

Answer: $2^{n-1} + n - 3$

c. A certain data center receives time-stamped messages from 100 remote stations, which it adds to a list as it receives them. Each station generates precisely one message every second in such a way that no two stations send messages with the same timestamp. However, the time delay in receiving the message varies from 100-200 milliseconds. After receiving N messages, the data center sorts the messages by timestamp using insertion sort. Asymptotically how long will this take, as a function of N?

Answer: $\Theta(N)$.

6. [6 points]

a. Given the following function definitions, what is the worst-case running time for the call `p(N)`? Assume that in this case, `h` is a boolean function requiring constant time.

Answer: $\Theta($*N*^{2}_____$)$.

```
int p(int M) {
    return r(0, M);
}

int r(int i, int M) {
    if (i >= M) return 0;
    if (s(i) > 0) return i;
    return r(i + 1, M);
}

int s(int k) {
    if (k <= 0) return 0;
    if (h(k))   return k;
    return s(k - 1);
}
```

b. What is the worst-case running time for the call `p(N)`? Assume that calls to `h` require constant time. **Answer:** $\Theta($*N*^{2}_____$)$.

```
void p(int M) {
    int L, U;
    for (L = U = 0; U < M; L += 1, U += 2) {
        for (int i = L; i < U; i += 1) {
            h(i);
        }
    }
}
```

c. The version of Kruskal's algorithm discussed in lecture takes as input an unordered list of edges and uses a union-find structure that performs union and find operations in very close to constant amortized time, processing the edges in order by edge weight to build the MST.

Consider a modification of the problem in which the input consists of a red-black tree whose node labels are the edges, ordered by weight. Will this algorithm provide an asymptotic improvement to the previous algorithm? Explain why or why not in 2-3 sentences.

Answer: Yes. A red-black tree is a binary search tree. We may perform an in-order traversal of the tree to get each edge in sorted order in linear time. This is an improvement on the previous algorithm's runtime, which first has to take worst-case $\Theta(N \lg N)$ time to sort the list of edges.

d. Assume that f and g return positive values. Is $f(x) + g(x) \in \Theta(\min(f(x), g(x)))$? If so, show why. Otherwise, provide a counter-example. To show that a formula $E \in \Theta(F)$ is true, use the definition of Θ, giving appropriate values K to make $K \cdot F$ larger or smaller than E.

e. Suppose that $h \in \Theta(1)$ and $k > 0$ is constant. Let

$$f(n) = \sum_{1 \le j \le k} h(n^j) n^j$$

. Fill in the blank with the simplest and shortest expression you can find so that

$$f(n) \in \Theta(\underline{n^k \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}).$$

f. Consider $f(n)$ as defined in (e) above, but suppose that $h(n) > 0$ everywhere and $h(n) \in O(1)$ rather than $h(n) \in \Theta(1)$. Give an example of such an h such that $f(n) \in \Theta(1)$.
Let $h(x) = 1/x$

7. [4 points] The following structure represents a single node in a linked list:

```
public class Link {
    public Link(int h, Link t) { head = h; tail = t; }

    public final int head;
    public Link tail;
}
```

Warning: Because `head` is declared **final**, it **may not be changed** once set by the constructor, unlike the `IntList` class we've been using.

The following method performs a step used in (for example) bubblesort. Fill it in to fulfill its specification. Do not create any new `Link`s.

```
/** Exchanges adjacent elements of the list staring at L in which the
 *  value in the first is larger than that of the next.  Performs this
 *  in a particular sequence: exchanges the first and second items (if
 *  needed), then the second and third of the resulting list, etc.
 *  For example if A initially contains the sequence [3, 8, 1, 6, 9, 2],
 *  then after A = bubble(A), it would contain [3, 1, 6, 8, 2, 9].
 *  Returns the resulting list.  Creates no new Links. */
static Link bubble(Link L) {

     if (L == null || L.tail == null) {
         return L;
     }

     if (L.head > L.tail.head) {
         Link second = L.tail;
         L.tail = second.tail;
         second.tail = bubble(L);
         return second;
      } else {
         L.tail = bubble(L.tail);
         return L;
      }
}
```

8. [6 points] The following classes define a type of directed-graph node (`Node`) and a class, `Traverser`, which performs a depth-first traversal of a graph. You are to fill in the definition of the method `GraphUtil.isAscending` so that it tells whether all paths from a given node have labels that are in strictly ascending order.

```
import java.util.*;

public class Node implements Iterable<Node> {
    Node(int label) { _next = new ArrayList<>(); this.label = label; }

    void addEdgeTo(Node successor) { _next.add(successor); }

    /** Because Node is Iterable, 'for (Node s : someNode) ...'  works. */
    public Iterator<Node> iterator() { return _next.iterator(); }

    public final int label;
    private ArrayList<Node> _next;
}

abstract class Traverser {
    abstract void visit(Node node);

    void traverse(Node start) {
        _visited.clear();
        traverse2(start);
    }

    private void traverse2(Node start) {
        if (!_visited.contains(start)) {
            _visited.add(start);
            visit(start);
            for (Node c : start) {
                traverse2(c);
            }
        }
    }

    private HashSet<Node> _visited = new HashSet<>();
}
```

The `Traverser` class has an abstract method, `visit`, which is called when the node is first reached in the traversal. By extending `Traverser` and overriding `visit`, one can get this class to do various things. Fill in the class below so that it fulfills its specifications. Your `visit` method must take time proportional to the number of successors of its argument in the worst case.

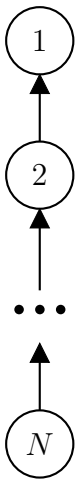
```
import java.util.*;

class GraphUtil {

      static class Checker extends Traverser {
          boolean ascending = true;
          void visit(Node node) {
              for (Node c : node) {
                  if (c.label <= node.label) {
                      ascending = false;
                  }
              }
          }
      }

      /** Return true iff all paths through the graph with vertices
       *  NODES have labels that are in strictly ascending order. */
      static boolean isAscending(Node[] nodes) {
          Checker chk = new Checker();
          for (Node n : nodes) {
              chk.traverse(n);
          }
          return chk.ascending;
      }

}
```

9. [4 points] Consider a union-find structure whose nodes are represented as follows:

```
public class UFNode {

    /** My parent, or null if I am the representative node of my set. */
    UFNode parent;
    int val;

    /** Return the representative node of the set I am currently in. */
    UFNode find() { implementation not shown }
    /** Join the set I am in to the set ITEM is in. */
    void union(UFNode item) { implementation not shown }

}
```

(For this problem, we write `x.find()` rather than the more familiar `find(x)` and `x.union(y)` rather than `union(x, y)`).

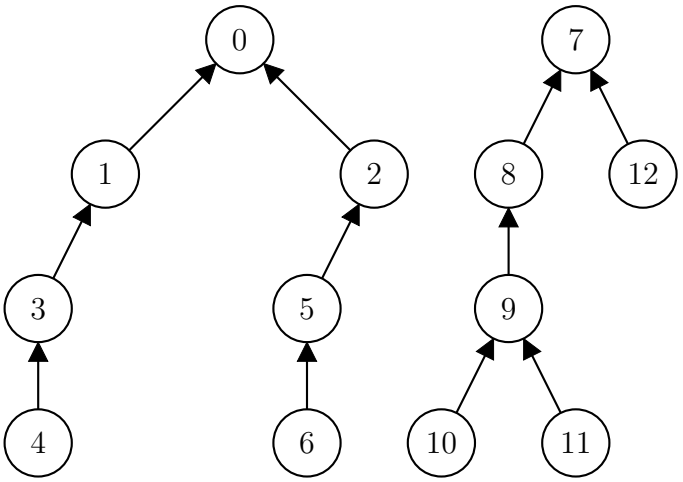
a. [1 point] If we aren't careful in implementing the `S1.union(S2)` operation and instead simply cause `S2`'s set to become `S1`'s set's parent in all cases, it is possible to create an N-node set in the union-find structure that looks like this:



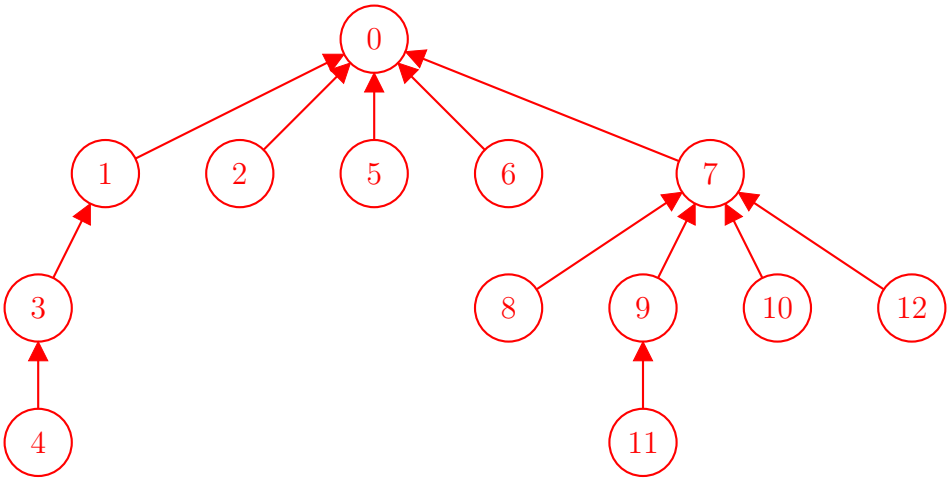
Assuming we use path compression, how long will it take in the worst case to execute N x`.find()` operations, where all of the x's are in this one set?

Answer: $\Theta($*N*_____$)$.

b. [3 points] To get optimal results, `union` should use path compression and always arrange that the representative of the resulting structure is the representative of the tree that had the larger height (after path compression). Show the union find structure that results if we perform the operation `I6.union(I10)` with the two-set structure below (where `I`N denotes the node labeled N in the diagrams).

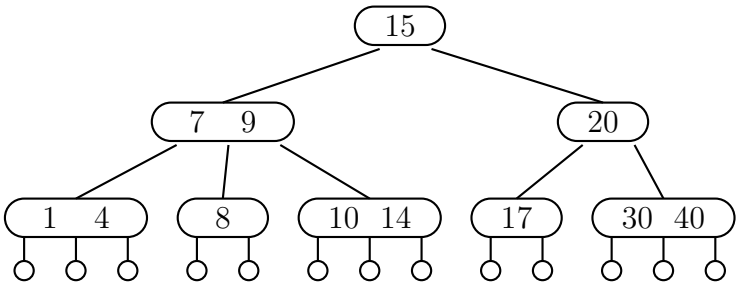


Answer:

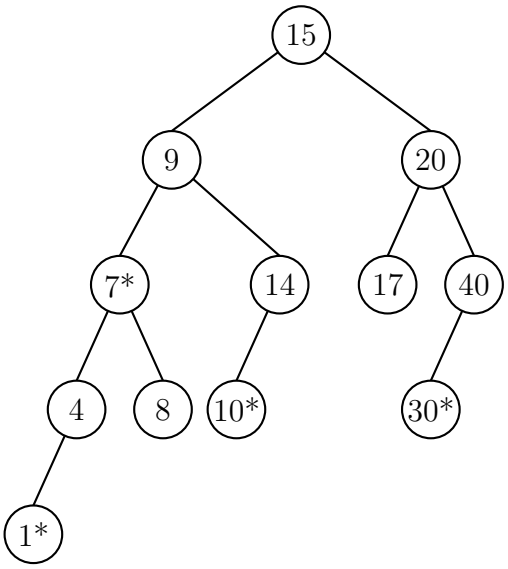


10. [5 points] The following questions involve red-black and B-trees.

a. [4 points] Show the unique left-leaning red-black tree corresponding to the (2, 3) tree below. Denote red nodes by putting an asterisk after the node label (as in `41*`). To avoid clutter, you need not show empty (null) nodes explicitly.



Answer:



b. [1 point] (2, 3) trees are perfectly balanced by construction. All children of a given node have the same height. Red-black trees can be unbalanced. If the left subtree of a given black node in a red-black tree has height h, what is the maximum height of its right subtree? For this purpose, count an empty branch as a (leaf) node. Do not give an asymptotic answer.

Answer: $2h$

11. [5 points] Below you will find some intermediate steps in performing various sorting algorithms on the same input list. The steps do not necessarily represent consecutive steps in the algorithm (that is, many steps are missing), but they are in the correct sequence. For each of them, write the name of the algorithm it illustrates on the blank line. Select the algorithm from among the following choices: insertion sort, selection sort, mergesort, quicksort (first element of sequence as pivot), heapsort, LSD radix and MSD radix sort.

In all these cases, the final step of the algorithm will be this:

```
560, 1127, 2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 7971, 9744
```

Input List:

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
```

a. Quicksort

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
 560, 1127, 2576, 5207, 3291, 5012, 9744, 5901, 6833, 3569, 4932, 7971
 560, 1127, 2576, 4932, 3291, 5012, 3569, 5207, 6833, 9744, 5901, 7971
 560, 1127, 2576, 4932, 3291, 5012, 3569, 5207, 5901, 6833, 9744, 7971
 560, 1127, 2576, 4932, 3291, 5012, 3569, 5207, 5901, 6833, 7971, 9744
 560, 1127, 2576, 3569, 3291, 4932, 5012, 5207, 5901, 6833, 7971, 9744
```

b. MSD radix sort

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
 560, 1127, 2576, 3569, 3291, 4932, 5207, 5012, 5901, 6833, 7971, 9744
 560, 1127, 2576, 3291, 3569, 4932, 5207, 5012, 5901, 6833, 7971, 9744
 560, 1127, 2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 7971, 9744
```

c. Heapsort

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
9744, 6833, 7971, 5901, 3291, 5207, 5012, 2576, 4932, 1127,  560, 3569
3569, 6833, 7971, 5901, 3291, 5207, 5012, 2576, 4932, 1127,  560, 9744
7971, 6833, 5207, 5901, 3291, 3569, 5012, 2576, 4932, 1127,  560, 9744
 560, 6833, 5207, 5901, 3291, 3569, 5012, 2576, 4932, 1127, 7971, 9744
6833, 5901, 5207, 4932, 3291, 3569, 5012, 2576,  560, 1127, 7971, 9744
```

Continued on next page.

d. Insertion sort

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5207, 5012, 9744, 5901, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 9744, 5901, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 9744, 1127,  560, 7971
1127, 2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 9744,  560, 7971
 560, 1127, 2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 9744, 7971
 560, 1127, 2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 7971, 9744
```

e. Mergesort

```
2576, 3569, 4932, 5207, 3291, 5012, 9744, 5901, 6833, 1127,  560, 7971
2576, 3569, 4932, 3291, 5207, 5012, 9744, 5901, 6833, 1127,  560, 7971
2576, 3569, 4932, 3291, 5012, 5207, 9744, 5901, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 9744, 5901, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 5901, 9744, 6833, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 9744, 1127,  560, 7971
2576, 3291, 3569, 4932, 5012, 5207, 5901, 6833, 9744,  560, 1127, 7971
```