

**CS61B, Fall 2014      Final Examination (W)      P. N. Hilfinger and Josh Hug**

READ THIS PAGE FIRST. Your exam should contain 14 problems on 16 pages. Officially, it is worth 46 points.

This is an open-book test. You have three hours 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: our 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 building once or twice.

Your name: \_\_\_\_\_

Login: \_\_\_\_\_

Login of person to your Left: \_\_\_\_\_

Right: \_\_\_\_\_

Discussion TA: \_\_\_\_\_

1. \_\_\_\_\_/2    2. \_\_\_\_\_/2    3. \_\_\_\_\_/4    4. \_\_\_\_\_/    5. \_\_\_\_\_/6

6. \_\_\_\_\_/4    7. \_\_\_\_\_/3    8. \_\_\_\_\_/3    9. \_\_\_\_\_/5    10. \_\_\_\_\_/4

11. \_\_\_\_\_/4    12. \_\_\_\_\_/2    13. \_\_\_\_\_/4    14. \_\_\_\_\_/3

## 1. [2 points]

- a. What does the following method return (as a function of
- $x$
- )?

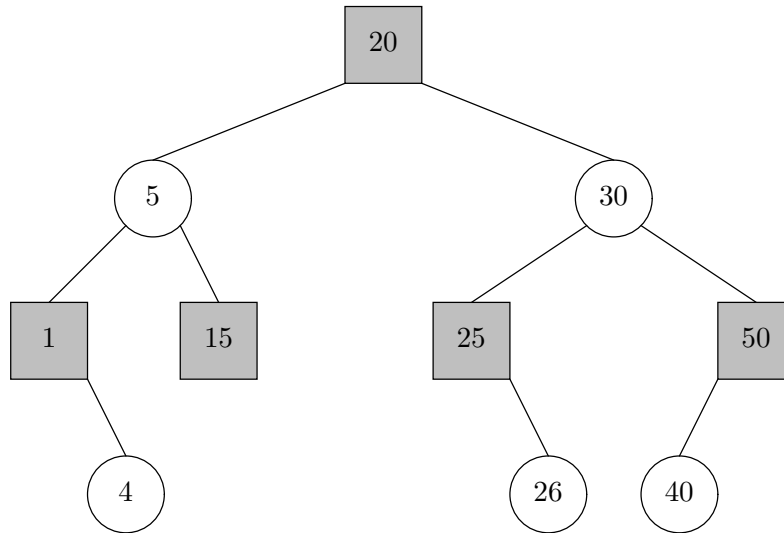
```
static int foo(int x) {
    x ^= -1;
    int c = x ^ x;
    while (x != 0) {
        x &= x - 1;
        c += 1;
    }
    return c;
}
```

- b. Suppose that
- $x$
- is of type
- `int`
- . Fill in the blank in the following method with a single expression to comply with the comment.

```
/** Return the value resulting from setting bits #LOW through #HIGH of X
 * (inclusive) to 1, leaving all other bits unchanged. Bits are
 * numbered 0..31, with 0 being the least significant (units) bit.
 * For example, if X is (in binary) 11001001, LOW is 2 and HIGH is
 * 4, the result would be 11011101. */
static int setBits(int x, int low, int high) {

    return _____;
}
```

2. [2 points] Consider the following red-black tree (black nodes are square, red nodes round).



Show the 2-3-4 tree (aka 2-4 tree) that corresponds to this red-black tree.



4. [1 point] Who has twice won the Gödel Prize in theoretical computer science and, as an undergraduate at CMU, took the equivalent of CS61B from one of the current instructors of that course?

5. [6 points] Consider the snippets of code below. Give the worst-case runtime for each complete snippet in  $\Theta(\cdot)$  notation (as a function of the value of  $N$  in each case). Give your answer in as simple a form as you can (e.g.  $\Theta(N^9)$ , not  $\Theta(12N^9)$ ). Just use 'lg' for any logarithm, since logarithms of all bases are related by constant factors. Each answer applies to executing its entire snippet.

a. `int sum = 0;`  
`for (int i = 0; i < N; i++)`  
`for (int j = 0; j*j < N; j++)`  
`sum++;`

Answer: \_\_\_\_\_

b. `int sum = 0;`  
`for (int i = 5*N*N*N; i > 1; i = i/2)`  
`sum++;`

Answer: \_\_\_\_\_

c. Assume here and in part (d) that `getRandomIntegers` generates  $N$  random unique integers in  $\Theta(N)$  time, and `mergeSort`, `insertionSort`, and `selectionSort` perform as described in class.

`int[] x = getRandomIntegers(N);`  
`x = mergeSort(x);`  
`x = insertionSort(x);`

Answer: \_\_\_\_\_

d. Under the same assumptions as (c):

`int[] x = getRandomIntegers(N);`  
`x = mergeSort(x);`  
`x = selectionSort(x);`

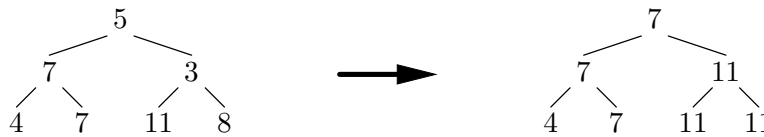
Answer: \_\_\_\_\_

e. Suppose that  $A$  and  $B$  are two strings of length  $L$ . Give bounds for the time required to compare  $A$  and  $B$  for equality (the actual time, not just worst-case time). Use  $O(\cdot)$  and  $\Omega(\cdot)$  notation to give the tightest upper and lower bounds you can on this running time.

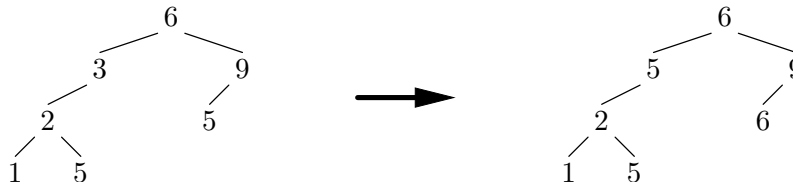
Answer:  $\Omega(\underline{\hspace{2cm}})$        $O(\underline{\hspace{2cm}})$ ;

6. [4 points] Consider the following problem: given an arbitrary binary tree with integer node labels, increase some of its node labels so that the resulting tree is a BST. We want to do this so that the total amount added to the nodes is minimized. For this problem, assume that BSTs are allowed to have duplicates.

**Example 1:**



**Example 2:**



It turns out to be possible to solve this problem using one of the three binary tree traversals we looked at in class: preorder, inorder, and postorder. Fill in the code on the next page so that the method `minBST` modifies its `TreeNode` argument to produce the BST described above.

The following types are provided for you:

```

public class TreeNode {
    public int val;
    public TreeNode left;
    public TreeNode right;
}

public interface BinaryTreeVisitor {
    void visit (TreeNode node);
}
  
```

Fill in the blanks in the minBST method below.

```
public class GraphUtils {
    public static void preOrder(TreeNode node, BinaryTreeVisitor visitor) {
        if (node != null) {
            visitor.visit(node);
            preOrder(node.left, visitor); preOrder(node.right, visitor);
        }
    }

    public static void inOrder(TreeNode node, BinaryTreeVisitor visitor) {
        if (node != null) {
            inOrder(node.left, visitor); visitor.visit(node);
            inOrder(node.right, visitor);
        }
    }

    public static void postOrder(TreeNode node, BinaryTreeVisitor visitor) {
        if (node != null) {
            postOrder(node.left, visitor); postOrder(node.right, visitor);
            visitor.visit(node);
        }
    }

    public static void minBST(TreeNode node) {
        _____Order(node, new MinBSTVisitor());
    }

    private static class MinBSTVisitor implements BinaryTreeVisitor {
        _____;

        public MinBSTVisitor() {
            _____;
        }

        public void visit(TreeNode node) {
            _____;
            _____;
        }
    }
}
```

7. [3 points] The game of FizzBuzzBoom is a simple one: Given a number (any integer greater than 0), the player's goal is to reduce that number to 1 by stringing together a sequence of fizz, buzz, and boom moves.

**Fizz** reduces the number by 1. A fizz move can be performed on any number.

**Buzz** divides the number by 2. A buzz move can only be performed on a number that is divisible by 2.

**Boom** divides the number by 3. A boom move can only be performed on a number that is divisible by 3.

Your task is to fill in the function `fizzBuzzBoom`, which given an integer  $n$ , returns the smallest number of moves required to win the game of FizzBuzzBoom.

For example,

| N  | fizzbuzzBoom(N) | Moves                                       |
|----|-----------------|---|
| 1  | 0               |   |
| 2  | 1               | Either a single Fizz move or a single Buzz. |
| 10 | 3               | Fizz, Boom, Boom.                           |

Fill in this program. Do not add any semicolons.

```

0 import static java.Math.min;
    ...
1     private static int fizzBuzzBoom(int n) {
2         int [] numSteps;

3         numSteps = _____;
4         numSteps[1] = _____;
5         for (int i = _____; _____; i += 1) {
6             _____;
7             _____;
8             _____;
9         }

10        return _____;
11    }
12    // NOTE: Feel free to format if statements without braces, as in
13    //         if (...) x = 3;

```



8. [3 points] Suppose that we add the ability to mark vertices to the `Graph` class from Project #3:

```
/** Set mark on vertex V. */
void mark(int v) { ... }

/** The current mark on vertex V (initially false). */
boolean marked(v) { ... }

/** Clear all marks in the graph to false. */
void clearMarks() { ... }
```

The following program is proposed to find if a directed graph has a cycle reachable from some vertex, `v`.

```
/** Returns true iff G contains a circularity that is reachable from
 * vertex V (that is, there is a path from V that eventually gets into a
 * cycle). */
public static boolean isCircular(Graph g, int v) {
    g.clearMarks();
    return reachesAlreadyVisited(g, v);
}

private static boolean reachesAlreadyVisited(Graph g, int v) {
    if (g.marked(v)) {
        return true;
    }
    g.mark(v);
    for (int w : g.successors(v)) {
        if (reachesAlreadyVisited(g, w)) {
            return true;
        }
    }
    return false;
}
```

Give a counterexample that shows that `isCircular` doesn't work.

9. [5 points] The class `InorderIterator` defines iterators that perform inorder traversals of binary trees, returning the values of the nodes traversed.

- a. Fill in the blanks to complete the class definition. This class `ucb.test.SimpleStack` is like `java.util.Stack`, but has *only* push and pop operations.

```
import java.util.Iterator;
import ucb.test.SimpleStack; // See above.

public class TreeNode {
    public int value;
    public TreeNode left; public TreeNode right;
}

public class InorderIterator implements Iterator<Integer> {
    private SimpleStack<TreeNode> fringe;
    /** An inorder iterator over the values in the tree ROOT. */
    public InorderIterator(TreeNode root) {
        fringe = new SimpleStack<TreeNode>();

        _____;
    }

    /** Returns true iff there are more values to deliver. */
    public boolean hasNext() {
        return !fringe.isEmpty();
    }

    /** Returns the value of the next node in inorder, advancing the iterator. */
    public Integer next() {
        _____;
        _____;
        _____;
    }

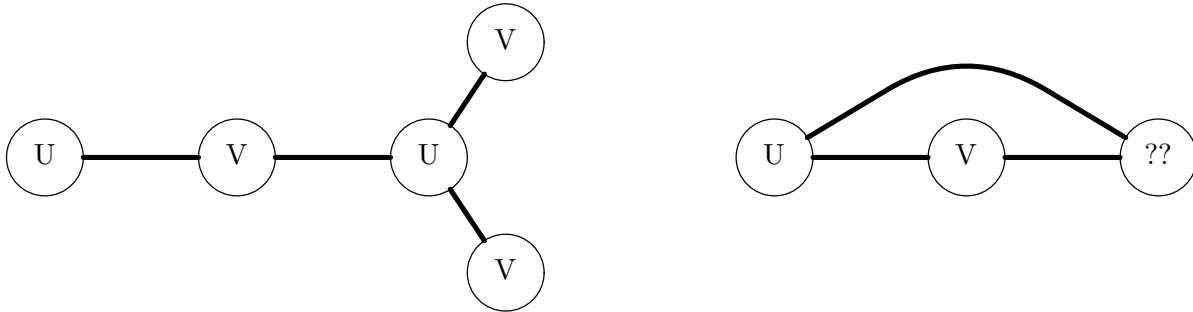
    public void remove() { throw new UnsupportedOperationException(); }

    private void addAllLeft(TreeNode n) {
        while (n != null) {
            fringe.push(n);
            n = n.left;
        }
    }
}

Parts b and c on the next page.
```



10. [4 points] An undirected graph is said to be *bipartite* if all of its vertices can be divided into two disjoint sets  $U$  and  $V$  such that every edge connects an item in  $U$  to an item in  $V$ . For example, the graph on the left is bipartite, whereas on the graph on the right is not.



In this problem, you are to design an algorithm that determines whether a graph is bipartite. For simplicity, assume the graph is connected (i.e. every vertex touches at least one edge, so that every node in  $U$  has a neighbor in  $V$  and vice-versa).

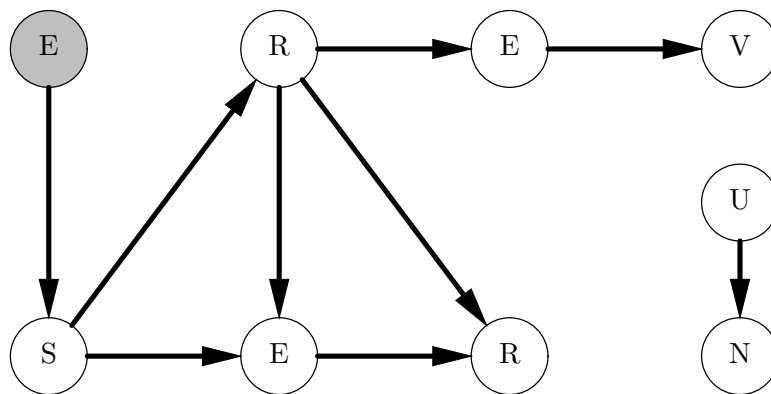
- Other than the graph itself, what other data structures do you need? Include all data structures, even arrays, ints, booleans, etc. List your data structures as **commented** Java variable declarations, much as you list member variables in your homeworks (if they pass the style check). Do not provide initializations for your data structures. If you're unsure on syntax, provide the closest approximation you can.
- Describe your algorithm in English or in code as concisely as possible, referring to your data structures as appropriate. Unclear algorithms will not be given credit. (This does not mean you need to explain how your data structures work; e.g. don't describe how BST insertion works). Algorithms that use data structures other than those listed in part (a) will not be given credit (so make sure to go back and list them in part (a) if you haven't already).



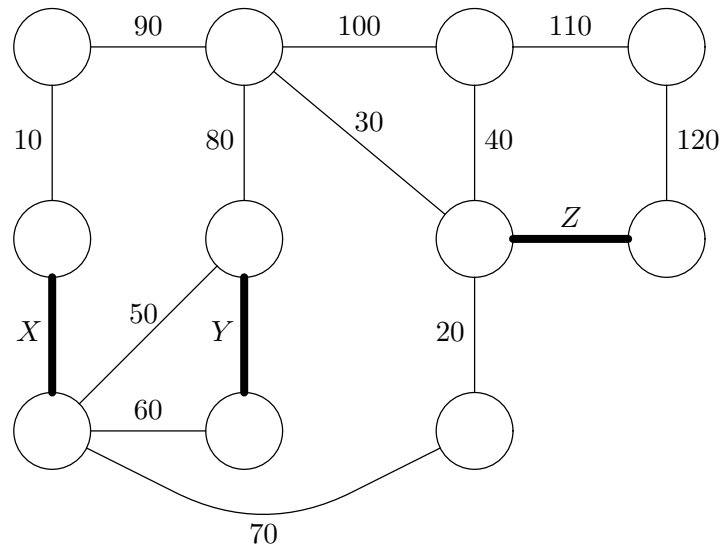
12. [2 points] Suppose we perform a depth first traversal of the labeled graph below starting from the shaded E in the top left corner. Suppose further that when given a choice between two edges, we always take the edge to the vertex whose label comes lexicographically earlier (e.g. from A, we'd take  $A \rightarrow B$  before  $A \rightarrow X$ ).

What are the letters of the nodes in the order they are **post**-visited?

Answer: \_\_\_\_\_



13. [4 points]



In the graph above, a certain minimal spanning tree comprises the edges with weights  $X$ ,  $Y$ ,  $Z$ , as well as seven other edges.

a. List the weights of the seven edges in the MST other than  $X$ ,  $Y$ , and  $Z$ .

10    \_\_\_\_\_

b. True or false: The value of edge weight  $X$  could be 120.

**Answer:** \_\_\_\_\_

c. True or false: The value of edge weight  $Y$  could be 55.

**Answer:** \_\_\_\_\_

d. What is the largest possible value of edge weight  $Z$ , assuming  $Z$  is some integer, and all edge weights are unique? **Answer:** \_\_\_\_\_

14. [3 points] For the following, fill in the blanks with T, F, or D, for “definitely true,” “definitely false,” or “depends on the data.” Assume that all compares take constant time. Also assume that there are no duplicates stored in these data structures.

- a. Consider a balanced binary search tree, such as a red-black tree.

\_\_\_ The median item is in the root node, if the number of items is odd.

\_\_\_ The largest item has no children.

\_\_\_ The time to find the smallest item is  $O(\log N)$ .

- b. Consider a heap-ordered array representing a max heap, with the largest item in position 1, its children in positions 2 and 3, their children in positions 4, 5 and 6, 7 respectively, and so forth.

\_\_\_ The second largest item is in either position 2 or 3.

\_\_\_ The third largest item is in either position 2 or 3.

\_\_\_ In the worst case, the time needed to iterate through the items of the heap in order is  $O(N)$ . [For this problem, answer either definitely true or definitely false]

- c. Consider using an external chaining hash table with  $M$  buckets and  $N$  items to implement a (Java-style) set. Suppose we multiplicatively resize the table as needed so that the load factor  $N/M$  never exceeds 5 or falls below  $1/5$ . Assume that each bucket is implemented as a linked list, that the hash function takes constant time to compute, and that comparing items also takes constant time to compute.

\_\_\_ The time to implement the `.contains()` operation is constant.

\_\_\_ For a given sequence of random insertions, the amortized time to insert a single item is constant.

\_\_\_ In the worst case, the time to find the smallest item is  $O(\log N)$ . (Answer either definitely true or definitely false.)





