CS61B, Fall 2014 Final Examination Solution (T) P. N. Hilfinger and Josh Hug

1. [2 points]

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

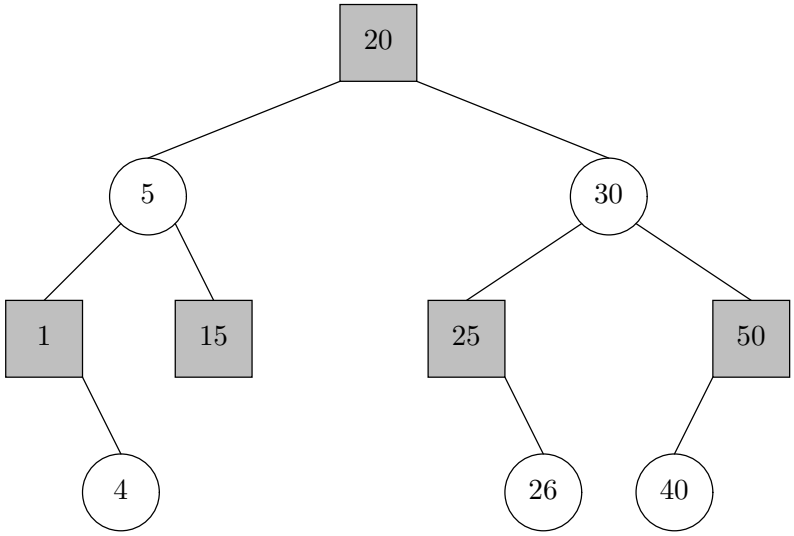
```
static int foo(int x) {
    x ^= -1;
    int c = x ^ x;
    while (x != 0) {
        x &= x - 1;
        c += 1;
    }
    return c;
}
```

Answer: The first line complements the bits of `x`. The second sets `c` to 0. The value `x-1` turns all trailing 0's of `x` to 1 and sets the least-significant 1 bit of of `x` to 0, leaving more significant bits unchanged. Hence the mask operation deletes the least significant 1-bit of `x`. Since `x` was initially complemented, therefore, the effect is to compute and return the number of 0-bits in `x`.

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.

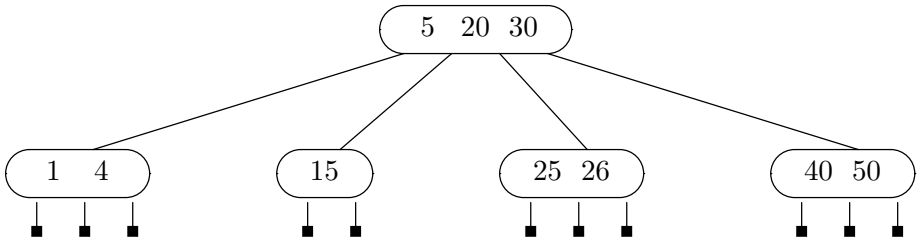
```
/** Return the value resulting from setting bits #LOW through #HIGH of X
 *  (inclusive) to 1, leaving all other bits unchanged.  Bits are
 *  numbered 0..31, with 0 being the least significant (units) bit.
 *  For example, if X is (in binary) 11001001, LOW is 2 and HIGH is
 *  4, the result would be 11011101. */
static int setBits(int x, int low, int high) RE{
    return    x | (-1) >>> (32 - (high - low + 1)) << low    ;
}
```

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.

Answer:



3. [4 points] In class, we looked at one way of implementing a priority queue: the binary heap. There is a natural generalization of this idea called a d-ary min-heap. Just as a binary heap is a complete binary tree such that any node is smaller than all of its descendants, a d-ary heap is a complete tree where every node is smaller than all of its descendants. But instead of every node having two children, every node has d children for some value d. Insertion and removal in a d-ary heap work similarly to those operations in a binary heap. Answer the following, use $\Theta(\cdot)$ notation as needed to express bounds. In this problem, d is not a constant.

a. What is the worst-case running time of inserting into a d-ary heap with N nodes in terms of d and N? Do not treat d as a constant for this problem.

Answer: $O(\log_d N) = O(\lg N / \lg d)$

b. What is the worst-case running time of finding the minimum element in a d-ary heap with N nodes in terms of d and N?

Answer: $O(1)$

c. What is the running time of removing the minimum element from a d-ary heap with N nodes in terms of d and N?

Answer: $O(d \log_d N) = O(d \lg N / \lg d)$

d. Suppose we have a list of N integers to sort. Consider this variation of heap sort: Let $d = N/8$ and insert all the integers into an initially empty d-ary heap. Then repeatedly remove the minimum element from the heap and add it to an initially empty list until the heap is empty. What is the worst-case running time of this sorting algorithm? Is it more efficient than a regular heap sort (with a binary heap)? Why or why not?

Answer: Since $d = N/8$, the heap will have height bounded by a constant. The amount of work done at each node meanwhile will scale as N, so that each reheapification will take $O(N)$ time, and there will be N of them. Running time is:

$$O(N \cdot d \cdot \log_d N) = O((N^2/8) \cdot \log_{N/8} N) = O(N^2).$$

4. [1 point] What are two notable things that D. Engelbart, M. Blum, S. Goldwasser, W. Kahan, R. Karp, D. Scott, and K. Thompson have in common?

Answer: They are all Turing Award winners who are or have been associated with Berkeley—Douglas Engelbart, Shafi Goldwasser, and Ken Thompson as students; Manuel Blum, William Kahan, and Richard Karp, as professors (Manuel Blum is now at CMU); Dana Scott as both student and professor.

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: $\Theta(N^{3/2})$

b.
```
int sum = 0;
for (int i = 5*N*N*N; i > 1; i = i/2)
    sum++;
```
Answer: $\Theta(\lg N)$

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: $\Theta(N \lg N)$

d. Under the same assumptions as (c):

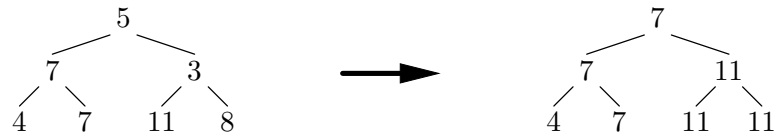
```
int[] x = getRandomIntegers(N);
x = mergeSort(x);
x = selectionSort(x);
```
Answer: $\Theta(N^2)$

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.

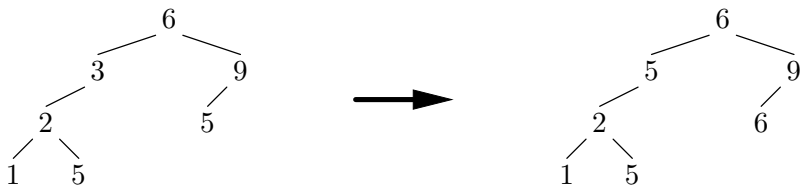
Answer: $\Omega(1)$ $O(L)$;

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) {
        inOrder(node, new MinBSTVisitor());
    }

    private static class MinBSTVisitor implements BinaryTreeVisitor {
            private int largest;          ;

        public MinBSTVisitor() {
              largest = Integer.MIN_VALUE      ;
        }

        public void visit(TreeNode node) {
            node.val = Math.max(largest, node.val);
            largest = node.val;
        }
    }
}
```

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 = new int[n+1];
4          numSteps[1] = 0;
5          for (int i = 2; i<=n; i += 1) {
6              numSteps[i] = 1 + numSteps[i-1];
7              if (i % 2 == 0) numSteps[i] = min(numSteps[i], numSteps[i/2]);
8              if (i % 3 == 0) numSteps[i] = min(numSteps[i], numSteps[i/3]);

9          }
10         return numSteps[n];
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() { ... }
```

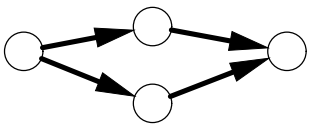
The following program is proposed to find if a directed graph has a cycle reachable from some vertex, `v`.

```
/** 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.

Answer:



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. Ths 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>();
        addAllLeft(root);
    }

    /** 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() {
        TreeNode n = fringe.pop();
        addAllLeft(n.right);
        return n.value;
    }

    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.

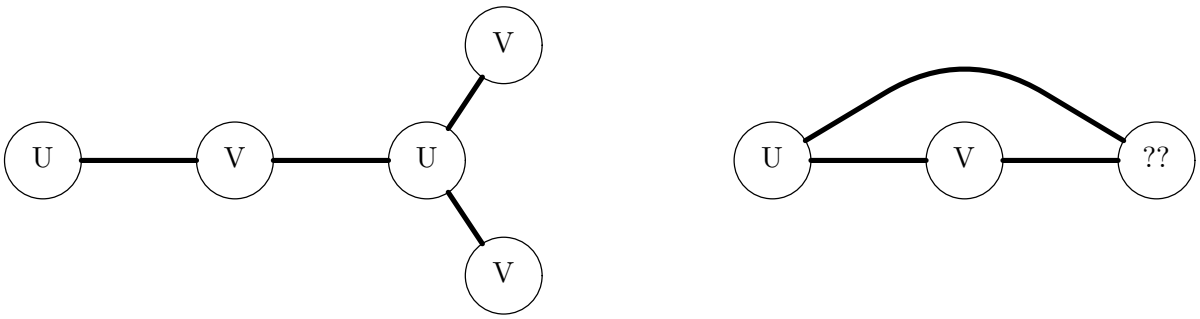
b. Given that N is the number of nodes in the tree starting at the given root, what is the worst-case runtime complexity needed to traverse all nodes using this inorder iterator (in $\Theta(\cdot)$ notation.) Assume that the tree is balanced.

Answer: $\Theta(N)$

c. What is the space complexity of traversing all nodes with this inorder iterator? In other words, at any given time, what is the maximum size of the fringe in the worst case as a function of N, the number of nodes? Again, use $\Theta(\cdot)$ notation. Assume that the tree is balanced.

Answer: $\Theta(\lg N)$

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).

a. 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.

Answer:

```
/* Assume import java.util.Map */

/** Mapping from vertices to true/false depending on whether they
 *  are in set U. */
Map<int, Boolean> isInU;
/** Number of vertices processed so far. */
int count;
```

b. 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).

Answer: Perform a depth-first traversal of the graph, and then return **true** if the algorithm does not terminate early. To visit a vertex, v:

- If v is not in `isInU`, set its value there to true.
- Otherwise, let x be the value of `isInU` on v. If any of v's neighbors have value x as well, the graph is not bipartite, so return **false**, terminating early. Otherwise set the value of each neighbor to not x in `isInU`.

11. [4 points] Answer the following questions true or false, assuming that our edge weights are unique, and may be negative, zero, or positive. If the answer is true, give a brief explanation why (do not provide a formal proof). If it is false, give a counterexample. Assume we are always referring to connected graphs with no self edges in the following questions.

a. Given a graph G, if we add some constant k to every edge weight, G's minimal spanning tree(s) remain unchanged.

Answer: True. If $|V|$ is the number of vertices, the MST must have $|V| - 1$ edges. Therefore, since the total weight of any tree with $|V|-1$ edges would increase by $k|V|-k$, any minimal tree will remain minimal relative to all other trees.

b. Assuming every vertex is reachable from a given source, Dijkstra's algorithm always finds a shortest path from that source to every vertex.

Answer: False. In this problem, edge weights can be negative, and Dijkstra's algorithm does not work in that case.

c. The shortest edge in any cycle can always be a part of a minimal spanning tree.

Answer: False. A minimal spanning tree need not contain *any* of the edges in a cycle. Consider a tree, T, to which we then add k new edges to form a cycle of k vertices, and assume that the weight of each new edge is larger than the sum of the edge weights in the original T. Then clearly T itself will be the only MST of the resulting graph.

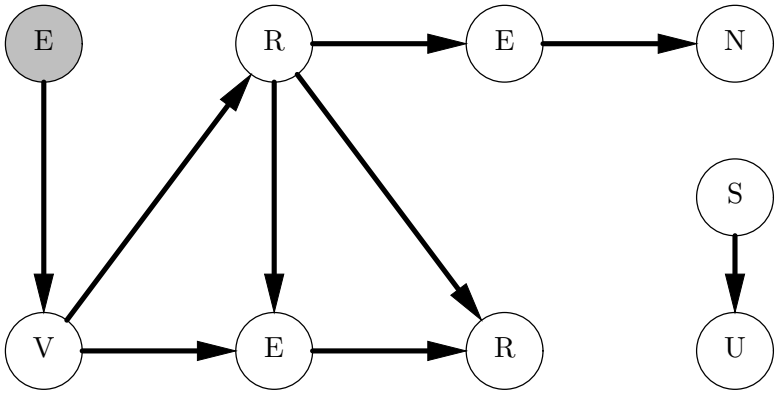
d. The total weight of a MST of an undirected graph is always less than or equal to the total weight of any shortest-path tree for that graph.

Answer: True by definition. A shortest-path tree is a spanning tree (it contains all vertices of a connected graph), so its total weight cannot be less than that of a MST.

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⟶B before A⟶X).

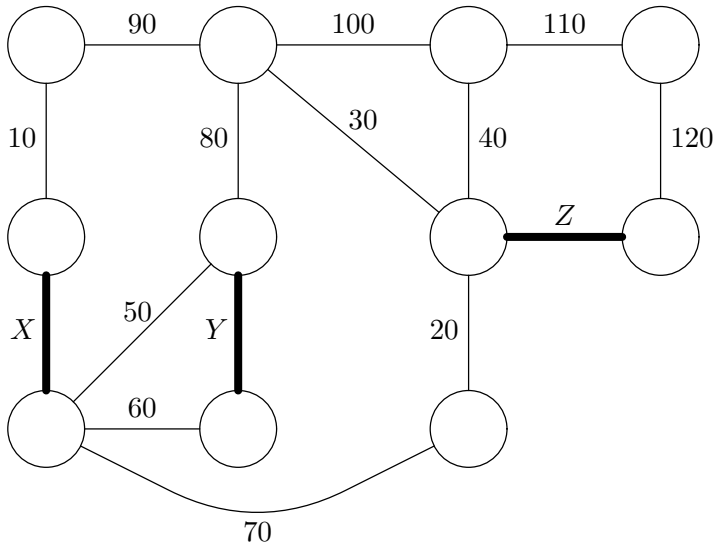
What are the letters of the nodes in the order they are **post**-visited?

Answer: RENERVE



NOTE: We also accepted answers that appended the labels of the two remaining vertices (which are not connected to the one in the upper left) in either order.

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 20 30 40 50 70 110

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

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

d. What is the largest possible value of edge weight Z, assuming Z is some integer, and all edge weights are unique? **Answer:** 119

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.

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

_D_ The largest item has no children.

_T_ 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.

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

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

_F_ 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.

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

_D/T_ For a given sequence of random insertions, the amortized time to insert a single item is constant. [*NOTE: if we assume that the implementation requires that duplicates never be added (so that it does not have to check), the answer is T, since the lists never need to be scanned. Otherwise, the answer depends on the data.*]

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