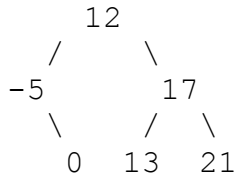
CS307 Spring 2011 Final Solution and Grading Criteria.

Grading acronyms

ABA - Answer by Accident
AIOBE - Array Index out of Bounds Exception may occur
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
ECF - Error carried forward.
Gacky or Gack - Code very hard to understand even though it works or solution is not elegant. (Generally no points off for this.)
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
NAP - No answer provided. No answer given on test
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.

1. As written or -1.5. No partial credit unless stated. On Big O, missing O() is okay.

A.



B. 26
C. 37 20 12 19 25 23 21
D. 12 20 19 37 25 21 23
E. 12 19 20 21 23 25 37
F. No
G. $O(N^2)$
H. 1.5 seconds
I. 17.6 seconds (16.4 - 18 okay)
J. 1 2 4
K. 0
L. O(NlogN)
M. O(NlogN)
N. 8 bits ($\log_2$ of 200 NOT okay)
O. any answer between 28 and 60 inclusive
P. 5000
Q. A String may not be added to an ArrayList declared to hold Integers. (or words to that effect)
R. The get methods declared return type is Object in this case and Object does not have a substring method. OR The value returned by get must be cast to a String. (or words to that effect)
S. The Elem class must implement the compareTo method or be declared abstract. (or words to that effect)
T. AIMNOP (lower case letters okay)

2. Suggested solution:

```
public void clear() {
    DoubleListNode<E> lead = first;
    DoubleListNode<E> trail = first;
    while(trail != null) {
        assert trail == lead;
        lead = lead.getNext();
        trail.setData(null);
        trail.setNext(null);
        trail.setPrev(null);
        trail = lead;
    }
    first = null;
    last = null;
}
```

Criteria. 15 points

move through list: (temp references or use first and last)
- attempt: 2 points
- correct: 5 points

null out prev, data, and next for each node:
- attempt: 1 point
- correct: 2 points

use trailing point correctly: 2 points
stop correctly: 2 points
set first and last to null: 1 point

not O(1) -> -7
use of methods not allowed -> -5

3A. Suggested Solution:

```
private int numBlackNodesFromRootToLeftMost() {
    RBNode<E> temp = root;
    int num = 0;
    while(temp != null) {
        if(temp.isBlack())
            num++;
        temp = temp.getLeft();
    }
    return num;
}
```

Criteria. 5 points

temp node assigned same reference as root: 1 point
loop or recurse until null: 1 point
only count black nodes: 1 point
move temp through tree: 1 point

3B. Suggested Solution:

```
private boolean allPathsCorrectHelper(RBNode<E> n, int magicNum,
         int blackNodesInCurrentPath) {
    if(n == null)
        return magicNum == blackNodesInCurrentPath;
    if(n.isBlack())
        blackNodesInCurrentPath++;

   return allPathsCorrectHelper(n.getLeft(), magicNum,
                                          blackNodesInCurrentPath)
           && allPathsCorrectHelper(n.getRight(), magicNum,
                                       blackNodesInCurrentPath);
}
```

Criteria. 8 points

base case for null including check: attempt 1 point, correct 2 points
check if current node black and if so update blackNodesInCurrentPath variable: 2 points
recursive case: attempt 1 point, correct 2 points

3C. Suggested Solution.

```
private boolean redHelper(RBNode<E> n) {
     if(n == null)
          return true; // base cae
     if(!n.isBlack()) {
          // red node, check children not red if they exist
          if(n.getLeft() != null && !n.getLeft().isBlack())
               return false; //red - red
          if(n.getRight() != null && !n.getRight().isBlack())
               return false; // red - red
     }
     // node is Black or red node with black children
     return redHelper(n.getLeft())
          && redHelper(n.getRight());
}
```

Criteria. 12 points
base case for null: 2 points
recursive case for black node: 3 points

for nodes that are red:
- check left and right children exist and if either red return false: 3 points
- if left and / or right child exist and are black, make recursive call(s): 3 points
- if red leaf return true: 1 point

early return -3
not using return value -3
not check children != null for red nodes -2

4. Suggested Solution

```
public static boolean properlyNested(Scanner fileScan) {
    Stack<String> tags = new Stack<String>();
    boolean good = true;

    // check the first line
    String current = fileScan.nextLine();
    good = isTag(current) && !isClosingTag(current)
    tags.push(currentString);

    while(good && fileScan.hasNextLine()) {
        current = fileScan.nextLine();
        if(isTag(current))  {
          if(!isClosingTag(current))
              tags.push(current);
          else
              good = !tags.isEmpty()
               && current.substring(2).equals(tags.pop().substring(1));
        }
    }
    return good && tags.isEmpty(); // no left over tags
}

private boolean isTag(String st) {
     return st != null && st.length() > 1
          &&   st.charAt(0) == '<'
          && st.charAt(st.length() - 1) == '>';
}

// pre: isTag
public boolean isClosingTag(String tag) {
     return tag.charAt(1) == '/';
}
```

Criteria: 15 points

check first line is opening tag: 1 point
loop through all lines of file: 2 points
push opening tags onto stack: 2 points
ignore things that are not tags: 1 point
for closing tags, if stack empty return false: 2 points
for closing tags, if top tag does not match return false: 2 points
if any left over opening tags return false: 1 point
return true if nested properly: 1 point

5A. Suggested Solution:

```
public static <T> int getSimilarityScore(Set<T> set1, Set<T> set2) {
    int sizeOfIntersection = 0;
    for(T set1Item : set1)
        for(T set2Item : set2)
            if(set1Item.equals(set2Item))
                sizeOfIntersection++;
    int sizeOfUnion = set1.size() + set2.size() - sizeOfIntersection;
    return 2 * sizeOfIntersection - sizeOfUnion;
}
```

Criteria. 5 points

use nested while loops with iterators or nested foreach loops: 2 points
check items equal: 1 point
count size of intersection: 1 point
calculate size of union correctly and calculate and return proper value: 1 point

5B Suggested Solution:

```
public static HashSet<String> findMostSimilarSenators (
                                Map<String, Set<Integer>> votingRecords) {

    HashSet<String> result = new HashSet<String>();
    int max = Integer.MIN_VALUE;
    Set<String> names = votingRecords.ketSet();
    for(String key1 : names
        Set<Integer> bills1 = votingRecords.get(key1);
        for(String key2 : names {
            if(!key1.equals(key2)) {
              int score = getSimilarityScore(bills1),
                                votingRecords.get(key2));
                if(score > max) {
                    result.clear();
                    result.add(key1);
                    result.add(key2);
                    max = score;
                }
                else if(score == max) {
                    result.add(key1);
                    result.add(key2);
                }
            }
        }
    return result;
}
```

Criteria. 10 points

use nested while loops with iterators or nested foreach loops: 2 points
don't check key against itself: 1 point
get similarity score: 1 point
track best score so far: 1 point
check if current score best so far: 1 point
and if so clear set, add results correctly and update best, 2 points
check current score equal best so far and add keys to result: 1 point
create HashSet and return as result: 1 point