CS314 Fall 2012 Final Exam Solution and Grading Criteria.

Grading acronyms:
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.
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
LE - Logic error in code.
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. Answer as shown or -2 unless question allows partial credit.
No points off for differences in spacing, capitalization, commas, and braces

A. 54

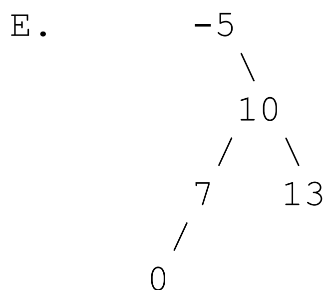
B. O(N)

C. 2,001,000

D. Uses the iterator method specified by the Collection interface and the equals method that all objects have. (or words to that effect)

("Polymorphism" only -1)

E.



F. max heap (max at root, every node value greater than all its descendants)

G. O(N)   (due to resize)

H. ATAOXZJK

I. OATZKJXA

J. TreeSet: O(NlogN)   (base 2 okay)
   HashSet: O(N)

K. 6

L. Root is Red. Must be black - 1 point
   Not a BST (45 is out of place) 1 point

M. [5, 7, 12, 12] (missing braces and commas okay)

N. 42 seconds

O. Lists with same elements in different order will have the same hashcode. (e.g. [A, B, C] same hashcode as [C, A, B] (or words to that effect)

P.
```
          6
        /   \
       11    9
      /  \
     12  15
```

Q. 11 000 01 10 001   (spaces not necessary)

R. 1. sparse
   2. uses (or wastes) too much space (or words to that effect)

S. 13

T. Due to shifting elements either the enqueue or dequeue operations would fe O(N). Adding to front and back of LinkedList always O(1)

extra credit: Dijkstra and Emerson (Phonetic spelling okay)

2.A Comments.

Common problems:

Suggested Solution:

```
public boolean validPath(String code) {
     HuffNode temp = root;
     int index = 0;
     while(index < code.length() && temp != null) {
          char bit = code.charAt(index);
          if(bit == '0')
               temp = temp.left;
          else
               temp = temp.right;
          index++;
     }
     // valid path if used all bits and temp refers to leaf
     return index == code.length() && temp.left == null
               && temp.right == null;
}
```

General Grading Criteria: 5 points

- loop through code (or recursion): 1 point
- move left or right as appropriate for '0' or '1': 1 point
- stop if temp references stores null: 1 point
- stop when used up all of code 1 point
- return true if all of code used and temp refers to leaf: 1 point

Other:

- Not O(N): where N = code.length(), -1
- Uses classes or methods other than HuffNode: -2
- Destroy tree: -2

2.B Comments.

Common problems:

Suggested Solution:

```
public int numDivisible(int num) {
    return helper(root, num);
}

private int helper(BSTNode n, int num) {
    if(n == null)
        return 0;
    count = n.val % num == 0 ? 1 : 0;
    return count + helper(n.left, num) + helper(n.right, num);
}
```

General Grading Criteria: 5 points

- helper method with correct parameters: 1 point
- base case, return 0: 1 point
- check current node divisible by num: 1 point
- recursive call to left and right: 1 point
- return answer in recursive case: 1 point

Other:

- Not O(N): where N = number of nodes, -1
- Uses classes or methods other than BSTNode: -2
- Destroy tree: -2

2.C Comments.

Common problems:

Suggested Solution:

```
private boolean redRuleMet() {
    return helper(root);
}

private boolean helper(RBNode<E> n) {
    // base case, empty, everything is good
    if(n == null)
        return true;

    // black node, just check descendants
    else if(!n.isRedNode)
        return helper(n.left) && helper(n.right);

    // red node, work to do!
    else {
        // coloring is correct if children exist and they
        // are not red
        boolean okay = (n.left != null && !n.left.isRedNode)
                    && (n.right != null && !n.right.isRedNode);

        // check children.
        // (short circuits if okay == false, base case)
        return okay && helper(n.left) && helper(n.right);
    }
}
```

General Grading Criteria: 10 points

- create helper method with correct parameter: 1 point
- base case when null: 2 points
- if black return result of children, proper logic: 2 points
- if red and any children red, return false (base case): 2 points
- if red and all children black or null, recursive calls on children and proper logic: 3 points

Other:

- Not O(N): where N = number of nodes, - 3
- Uses classes or methods other than RBNode: -4
- Destroy tree: -3

3. Comments:

Common problems:

Suggested Solution:

```
public boolean pathExists(int start, int dest) {
     HashSet<Integer> visited = new HashSet<Integer>();
     return helper(visited, start, dest);
}

private boolean helper(HashSet<Integer> visited,
                              int current, int dest) {
     // Have I been to this vertex before?
     // If so, I don't want to go around in circles.
     if(visited.contains(current))
          return false;

     // Does direct link exist??
     else if(adjMat[current][dest])
          return true;

     // Darn it. Try the other choices.
     else {
          // I have now visited current.
          visited.add(current);

          for(int vertex = 0; vertex < adjMat.length; vertex++) {
               // If edge exists, try it!
               if(adjMat[current][vertex]) {
                    if(helper(visited, vertex, dest))
                         return true;
          }

          // never found a path
          visited.remove(current);
          return false;
     }
}
```

General Grading Criteria: 20 points

- helper method: 1 point
- base case when a vertex already visited: 2 points
- base case when direct link exists: 2 points
- recursive case, iterator through choices: 3 points
- mark vertex as visited: 2 points
- if link exists, make recursive call: 3 points (-4 if early)
- if recursive call found solution, return true: 1 point
- if no solution after checking links, return false: 1 points
- infinite loop due to not marking vertex -4

4.A Comments:

Common problems:

Suggested Solution:

```
private boolean repairPreviousReferences() {
     Node<E> scout = header.next;
     Node<E> trailer = header;
     boolean fixedAny = false;
     while(scout != header) {
          if(scount.prev != trailer) {
               scount.prev = trailer;
               fixedAny = true;
          }
          trailer = scout;
          scout = scout.next;
     }
     return fixedAny;
}
```

General Grading Criteria: 10 points

- loop through entire list: 2 points
- check if prev reference incorrect: 2 points
- track if repairs made: 1 point
- fix prev reference if incorrect: 1 point
- actually move points through list correctly: 3 points
- return true if fixed any references, false otherwise: 1 point

Other:

- Not O(N): -3
- Uses classes or methods other than Node: -4
- Destroy list: -4

4.B Comments:

Common problems:

Suggested Solution:

```
public void addSumEnd() {
     // empty list case
     if(first == null) {
          first = last = new Node(0, null);
     else {
          int sum = 0;
          Node temp = first;
          while(temp != null) {
               sum += temp.data;
               temp = temp.next;
          }
          last.next = new Node(sum, null);
          last = last.next;
     }
}
```

General Grading Criteria: 10 points

- handle empty case correctly: 2 points (-1 if don't assign first)
- loop through entire list: 1 point
- find cumulative sum: 2 points
- move pointer correctly: 2 points
- create last node and link correctly: 1 point
- move last to refer to new last node: 2 points

Other:

- Not O(N): -3
- Uses classes or methods other than Node: -4
- Destroy list: -4

5.
precondition question: size of queue must be less than capacity
If precondition not met throw an exception.

```
public void addFront(E val) {
     if(size == 0) {
          front = last = 0;
          con[front] = val;
     else {
          first = first == 0 ? con.length - 1 ? first - 1;
          con[first] = val;
     }
     size++;
}

public void addBack(E val) {
     if(size == 0)
          addFront(val);
     else {
          last = (last + 1) % con.length;
          con[last] = val;
          size++;
     }
}
```

General Grading criteria: 8 points

- shifting elements in addFront -3 (O(N) when O(1) possible)
- shifting elements in addBack -2 (O(N) when O(1) possible)
- not updating size -1
- not update front -1
- not update last -1
- creating new array -3

Unbuffered? If con full then must create new array and copy elements over and update first and last appropriately.

Data Structure? LinkedList. (-2 if ArrayList)