CS314 Fall 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 2-2. No partial credit unless stated. On Big O, missing O() is okay.

A. 16

B. 6

C. 3 15 17 6 4

D. Do inorder traversal on tree, values in ascending order, no repeats.

E. $O(N^2)$

F. O(NlogN) (base 2 okay)

G. 1 point each
- not a binary search tree, 35 < 37
- path rule not met, 3 black nodes in path 37 - 50 42 - 45 all other paths have 2 black nodes

H. mergesort

I. 40 seconds

J. Chaining. Array elements are other data structures such as lists. When a collision occurs the elements are simply added to the same smaller data structure. (or words to that effect. Partial credit possible)

K. 1 point each

Average case od add: O(1)
Order of add method if load factor near 1.0: O(N) due to having to search most of the array for an open spot. (Or words to that effect.)

L. Any object may be added because the Object class has a hashcode method and all classes are descendants of object.

M. ABCZZZ

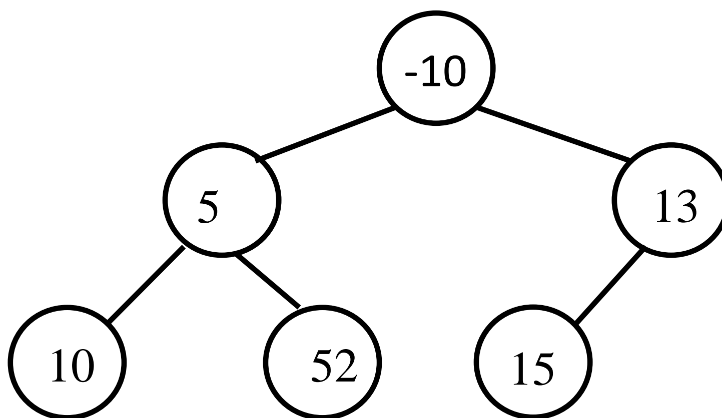
N. ACDCDAD

O. 1 point each
1. with small files the header information may prevent compression
2. if the frequency of chunks is uniform

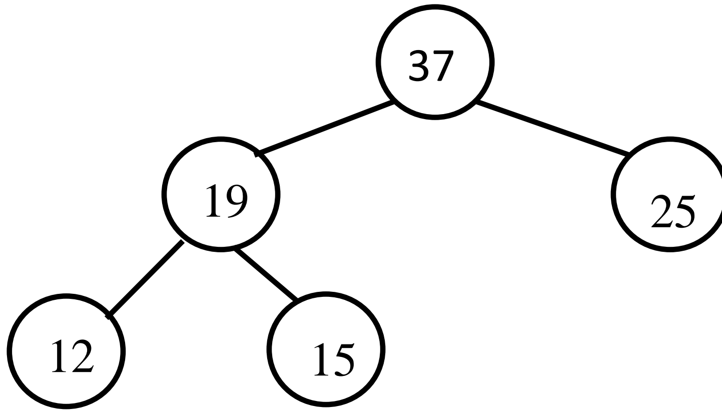
P. 30

Q. $O(N^2)$

R.



S.



T. implementation A internal storage container: Array based list

implementation B internal storage container: Hash table

EXTRA CREDIT : Password for the user id STUDENT is OVERWORKED (+1 point)

2.B

```
public HuffmanTree(Map<Integer, String) chunksAndCodes) {

     root = new HuffNode();
     root.freq = -1;

     for(int chunk : chunksAndCodes.keySet())
          addChunk(chunk, chunksAndCodes.get(chunk));
```

1 point: create root node

1 point: iterate through elements of map via keyset (other ways possible)

1 point: call addChunk correctly with chunk and code

2.C

```
private boolean completeHelper(HuffNode n) {

     if(n == null)
          return true; // not necessary for answer

     else if(n.leftChild == null && n.getRight == null)
          return true;
     else if(n.getLeft != null && n.getRight != null)
          return completeHelper(n.getLeft)
               && completeHelper(n.getRight);
     else
          return false;
}
```

1 point: base case for leaf, return true

1 point: base case for one child, return false

1 point: case with 2 children checked correctly

2 points: recursive call with 2 children correctly made

HASH

```
private class HashIterator implements Iterator<E> {

    private boolean removeOK;
    private int index;
    private int numReturned;
    private int maxReturn;

    private HashIterator() {
        maxReturn = size;
        index = -1;
    }

    public boolean hasNext() {
        return numReturned < maxReturn;
    }

    public E next() {
        if(!hasNext())
            throw new NoSuchElementException();

        // move to next spot
        index++;

        // find next;
        while(con[index] == null || con[index] == EMPTY)
            index++;

        removeOK = true;
        numReturned++;
        return con[index];
    }

    public void remove() {
        if(!removeOK)
            throw new IllegalStateException();

        removeOK = false;
        size--;
        con[index] = (E) EMPTY;

    }
}
```

GRAPH.

```
public boolean partOfCycle(String start) {
   if(!containsVertex(start))
        throw new IllegalArgumentException("No vertex with this name
                                            in graph: " + start);

    clearAll();
    Queue<Vertex> toVisit = new LinkedList<Vertex>();
    Vertex st = vertices.get(start);
    for(Edge e : st.adjacent)
        toVisit.add(e.dest);

    boolean backAtStart = false;
    while(!backAtStart && !toVisit.isEmpty()) {
        Vertex v = toVisit.remove();
        if(v.name.equals(start))
            backAtStart = true;
        else if(v.scratch == 0) {
            v.scratch = 1;
            for(Edge e : v.adjacent) {
                toVisit.add(e.dest);
            }
        }
    }
    return backAtStart;
}
```

BST

```
private void medianHelper(BSTNode<E> n, int[] count, E[] result) {
    if(n != null) {
        medianHelper(n.getLeft(), count, result);
        // now I am here
        count[0]++;
        //check if this node is median
        if(size % 2 == 1 && count[0] == size / 2 + 1)
            result[0] = n.getData();
        else if(size % 2 == 0 && count[0] == size / 2) {
            result[0] = n.getData();
            medianHelper(n.getRight(), count, result);
        }
        else if(size % 2 == 0 && count[0] == size / 2 + 1)
            result[1] = n.getData();
        else if(count[0] < size / 2)
            medianHelper(n.getRight(), count, result);
    }

}
```