NUMBER 1, SHORT ANSWER:

A. $N^2 + 2N + 5$ (+/- 0.5 on first two terms, +/- 1 on last term)

B. O(N) (dependent loop 1 + 2 + 4 + 8 + ... + N/4 + N/2 + N = 2N - 1)

C. $O(N^2)$

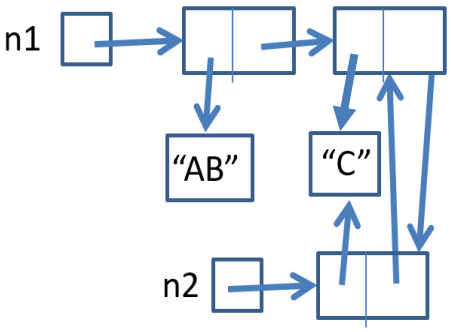
D. 8 seconds (algo is N^2)

E. 21 or not a valid postfix expression

F. 1. The root isn't black 2. 2 paths with 2 black nodes (2 -> 1, 2 -> 4 -> left) and 2 paths with 3 black nodes (2 -> 4 -> 6 -> 5 or 7)

G. 10

H. A stack to track nodes we need to back track to.



I. 3

J. 4 seconds (code is best case, O(N))

K. 21 seconds (code is $O(N\log_2 N)$)

L. ---------->

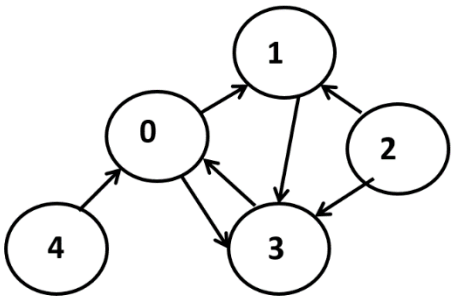
M. $O(N^2)$

N. {-3=Z, 2=B, 5=C} (differences in spacing, separators ok, any quotes wrong)

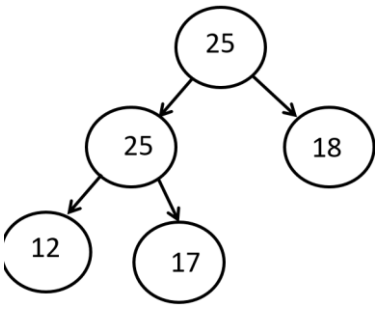
O. 12

P. 8 seconds (Method is $O(2^N)$)

Q. ---------->



R. Singly linked to save space. Only need to access first element (Or words to that effect.)



S. ---------->

T. average number of edges in shortest paths to other nodes (OWTTE)

U. 40 seconds (method is O(N))

V. 3 (Only. Return type of get for av is Object, no charAt method.)

W. 9

X. The header information that was included in the compressed file. (OWTTE)

2. Suggested Solution

```
// pre: at least 2 vertices
public boolean isConnected(String ignore) {
    clearAll();
    Iterator<String> it = vertices.keySet().iterator();
    Vertex start = vertices.get(it.next());
    if (ignore.equals(start.name)) {
        start = vertices.get(it.next());
    }
    Queue<Vertex> q = new LinkedList<>();
    q.add(start);
    int visited = 0;
    int required = vertices.size() - 1;
    while (visited < required && !q.isEmpty()) {
        Vertex current = q.remove();
        if (current.scratch == 0) {
            visited++;
            current.scratch = 1;
            for (Edge e : current.adjacent) {
                if (!ignore.equals(e.dest.name)) {
                    q.add(e.dest);
                }
            }
        }
    }
    return visited == required;
}
```

16 points , Criteria:

- correctly get first node, checking it isn't ignore. 2 points
- call clearAll and create Queue, 1 point
- loop while q not empty and still vertices to visit, 2 points total
  - -1 if don't stop when visited all vertices
- check current vertex has not yet been visited, 2 points
- when visiting vertex:
  - update scratch for vertex, 1 point
  - increment number vertices visited, 1 point
  - loop through edges and add destinations, 1 point (okay to check scratch)
- correctly ignores any edges going to the vertex to be ignored, 5 points
- return correct answer

Other penalties:

- altering graph, -6 (including remove ignore)
- using recursion, -5 (space)
- $O(N^2)$, -5
- Only checking edges (friends of friends only), MCE, -10
- OBOE, -2
- enqueue all vertices to start, -8
- don't enqueue at all , -5

3. Suggested Solution:

```
public int removeAll(E tgt) {
     Node<E> temp = header.next;
     int numRemoved = 0;
     while (temp != header) {
          boolean match = (tgt == null) ? temp.data == null
               : tgt.equals(temp.data);
          if (match) {
               numRemoved++;
               temp.prev.next = temp.next;
               temp.next.prev = temp.prev;
          }
          temp = temp.next;
     }
     return numRemoved;
}
```

16 points, Criteria:

- var to track num removed, 1 point
- temp node var, initialized correctly, 1 point
- loop until back to header node, 3 points
- handle cases when tgt is null, 3 points
- handle cases when tgt isn't null, 1 point (lose if ==)
- if node contains data that matches tgt, correctly remove it, 3 points
- advance temp correctly, 2 points
- return correct result: 1 point

Other deductions:

- worse than O(N), -5
- infinite loop for any reason, -4 (== null for example)
- attempt to alter what node header refers to, -3
- use any methods besides equals, -2 to -6 depending on severity
- OBOE, -2
- assume header is first node with data, -4
- destroy all or most of list, -6
-

4. Suggested solution:

```
public int numPaths(int tgt, int req) {
    return pathHelp(tgt, req, 0, false, root);
}

private int pathHelp(int tgt, int req, int cur,
        boolean reqMet, BNode n) {
    int result = 0;
    if (n != null) {
        cur += n.data;
        reqMet = reqMet || n.data == req;
        if (reqMet && cur == tgt) {
            result++;
        }
        result += pathHelp(tgt, req, cur, reqMet, n.left);
        result += pathHelp(tgt, req, cur, reqMet, n.right);
    }
    return result;
}
```

13 points, Criteria:

- create helper, 1 point
- base case, return 0 if null, 2 points (or look ahead)
- recursive case, add data in node to running total, 1 point
- check if data in node meets requirement (if necessary), 3 points
- check if current path meets target total, 2 points
- correct recursive calls, 4 points

Other:

- use any methods besides equals, -2 to -6 depending on severity
- resetting boolean / flag for required value each time (-3)
- early return, -5
- using array, -3 (even length 1)

5. Suggested Solution:

```
public ArrayList<Integer> writeFile(BitInputStream in,
          BitOutputStream out)  throws IOException {

        ArrayList<Integer> result = new ArrayList<>();
        int chunkSize = in.readBits(6) + 1;
        int numChunks = in.readBits(32);
        for (int i = 0; i < numChunks; i++) {
            int num1s = 0;
            for (int j = 0; j < chunkSize; j++) {
                int bit = in.readBits(1);
                out.writeBits(1, bit);
                num1s += bit; // if 0, doesn't change           }
            // Read the parity bit, but don't write it out to the file.
            num1s += in.readBits(1);
            // do we have the correct parity?
            if (num1s % 2 != 0) {
                result.add(i); // chunk i has an error based on parity
            }
        }
        return result;
    }
```

15 points, Criteria:

- read in chunk size and +1, 1 point
- read in and store number of chunks, 1 point
- loop for number of chunks, 1 point
- read 1 bit at a time, adding up number of 1s, 4 points
- write out data bits to output file, 2 point
- read parity bit and check parity correctly, 4 points
- add number of chunk with error to result, 1 point
- add number of errors to end of result and return, 1 point

Other:

- write parity bit to output file, -3
- try to read file multiple times, -5
- no return, -1
- scope wrong on variables, -3
- infinite loop, -5
- -7 assume a parity bit of 1 means error, 07
- write out parity bit, -3
- calculate powers of 2, eff, -3
- using methods not available, -5

6. Comments:

```
public List<String> getWords(String pre) {
      List<String> result = new ArrayList<>();
      TNode start = getNodeForPrefix(pre);
      if (start != null) {
            getWords(result, start, pre);
      }
      return result;
}

    private void getWords(TNode n, List<String> result, String word) {
        if (n.word) {
            result.add(word);
        }
        if (n.children != null) {
            for (TNode child : n.children) {
                getWords(child, result, word + child.ch);
            }
        }
    }
```

16 points, Criteria:

- check if TNode is null or not and make recursive call, 3 points
- add helper method with correct parameters, 2 points
- correctly check if current node is a word and add to result, 3 points
- check children list isn't null, 2 points
- loop through children, 1 point
- change word by adding current char, 3 points
- recursive call correct, 2 points

Other:

- altering TRIE in any way, -6
-