A. 6

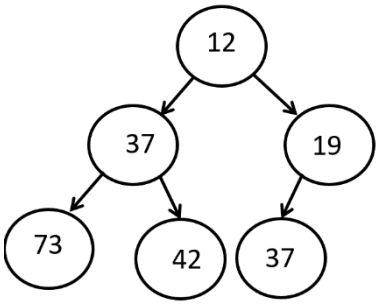
B. Binary search trees are not always complete trees and so too much space would be wasted. (or words to that effect = OWTTE)

C. The data would already have to be sorted to find the median (OWTTE)

D. 47

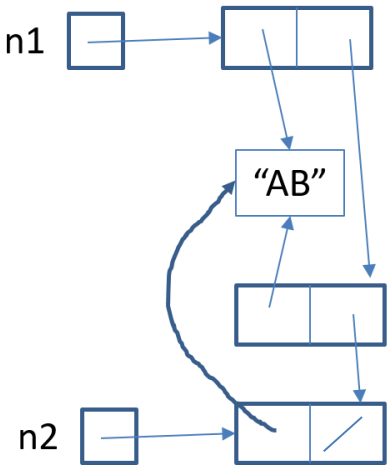
E. 15 127

F. ---------->

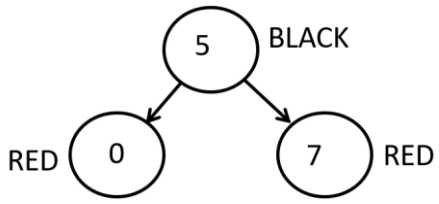


G. $2N^2 + 6N + 5$ (+/- 1 on each coefficient)

H. ---------->



I.



J. 1640 minutes

K. 3.0 seconds

L. 10, 12, 19, 17, 21, -6

M. linked list. With the array based list one of queue or enqueue will be O(N) due to shifting. (or words to that effect)

N. Larger because each code will likely be 8 bits (small differences in frequency) plus the header information. (or words to that effect)

O. {AB=5, DO=12, IS=7, IT=12} (differences in spacing, separators ok)

P. 84 seconds

Q. LISPSIP

R. 17

S. Most significantly, recursive algorithms start with a large version of the problem and works towards smaller, simpler versions. Dynamic programming algorithms starts with the simple versions of the problem and work up towards the large version. (or words to that effect)

T. No. Not every problem can be solved from simpler versions and not every problem has an ordered set of solutions. (or words to that effect)

2. Suggested Solution

```
public LinkedList314<E> getSubList(int startIndex, int stopIndex) {
      LinkedList314<E> result = new LinkedList<>();
      // special case when empty result, efficiency
      if (startIndex == stopIndex) {
            return result;
      }

      // move to the startIndex;
      Node<E> tempThis = first;
      int pos = 0;
      while (pos < startIndex && tempThis!= null) {
            tempThis = tempThis.next;
            pos++;
      }

      if (tempThis != null) {
            // add first element
            result.first = new Node<>(temp.data);
            Node<E> tempResult = result.first;
            tempThis = tempThis.next;
            pos++;
            // add elements until run off list or done
            while (tempThis != null && pos < stopIndex) {
                  tempResult.next = new Node<>(tempThis.data);
                  tempResult = tempResult.next;
                  tempThis = tempThis.next;
                  pos++;
            }
      }
      return result;
}
```

16 points , Criteria:

- special case, startIndex == stopIndex, O(1), 2 points
- move to startIndex
    - stop if out of bounds, 2 points
    - if inbounds, moves successfully to startIndex, 3 points
- add first element to result by updating result.first, 2 points
- while loop to add elements, check temp != null (possibly temp.next) and position < stopIndex, 1 point
- for each node in loop:
    - add to result correctly, 1 point
    - move temp node references, 4 points
- return result, 1 point

Other penalties:

- other data structures, -4 add method without implementing, -6
- Not O(N), -4
- Alter this list now or possibly in future due to sharing references, - 6

3. Suggested Solution:

```
public int numOddInternal() {
     return helper(root);
}

private int helper(BNode n) {
     if (n == null) // base case
          return 0;

     int result = 0;
     // is this an internal node with an odd value
     if ( (n.left != null || n.right != null) && n.data % 2 != 0)
          result++;

     // check children
     result += helper(n.left) + helper(n.right);
     return result;
}
```

14 points, Criteria:

- create helper, 2 points
- base case, 3 points (-2 if misses empty tree. Okay to have two base cases, empty tree and leaf node)
- check if internal code correctly, 2 points (lose if add leaf node to results)
- check if not evenly divisible by 2, 1 point (-1 for -3 % 2 == 1 error)
- add 1 if meets criteria of internal node and odd, 1 point
- recursive calls correct, 3 points
- return correct result, 2 points

Other deductions:

- early return / not checking all nodes, -6
- use of array or other value to count, -5
- counting leaf nodes in results, -4

4.

```
public boolena putIfAbsent(K key, V val) {
      if ( (double) size / con.length >= LOAD_LIMIT) {
            resize();
      }
      int index = ky.hashCode() % con.length;
      index = Math.abs(index);
      boolean search = true;
      int addIndex = -1;
      // search until we find key or a null
      while (search) {
            if(con[index] == null) {
                  search = false;
                  addIndex = (addIndex == -1) ? index : addIndex;
            } else if (con[index] == EMPTY && addIndex == -1) {
                  // add at this spot if we do add
                  addIndex = index;
            } else if (key.equals(con[index].key)) {
                  // key already present
                  search = false;
                  addIndex = -1; // so we don't add
            }
            index = (index + 1) % con.length;
      }
      if (addIndex != -1) {
            con[addIndex] = new Pair(key, value);
            size++;
      }
      return addIndex != -1;
}
```

20 points, Criteria:

- calculate array index from hash code correctly, 2 points
- resize if necessary, 3 points
- loop until first null to ensure key not present, 4 points (must go to first null. First EMPTY is not adequate.)
- return false if find key, 3 points
- wrap index correctly when searching, 2 points
- if not present, place in first available spot found (could be EMPTY, even though must search until null)
- put correctly, 1 point
- update size if put, 2 points
- return correct answer, 1 point

Other:

- O(N), -6
- NPE -4
- infinite loop, -4

5.

```
public int decodeTriplets(BitInputStream in, BitOutputStream out) {
      // read three bits at a time and map to majority element
      int[] bitmap = {0, 0, 0, 1, 0, 1, 1, 1};
      int errors = 0;
      int inBits = in.readBits(3);
      while (inBits! = -1) {
            int outBit = bitmap[inBits];
            out.writeBits(1, outBit);
            if (inBits != 0 && inBits != 7) {
                  errors++;
            }
            inBits = in.readBits(3);
      }
      return errors;
}
```

16 points, Criteria:

- read bits correctly 2 points
- while loop correct, 3 points
- determine majority bit, 4 points
- track errors, correctly, 4 points
- write out bits, 2 points
- return, 1 point

Other:

- skips bits, -4
- call to hasNext on BitInputStream, -4
- treating ints as values such as 101 instead of 5, -8 (major conceptual error)
- using Strings, -4
- recreate array each time in loop, -3 (space efficiency)
- write out too many bits, -4
- x != 0 || x != 3 logic error (that is always true. Likewse (num1 != 3 || num2 != 3) , -3

6. Comments:

```
public void add(E val) {
      size++;
      if (size == con.length) {
            resize();
      }
      int posChild = size;
      int posParent = (size + 1) / 3;
      while (posChild > 1 && val.compareTo(con[(posParent]) > 0) {
            con[posChild] = con[posParent];
            posChild = posParent;
            posParent = (posChild + 1) / 3;
      }
      con[posChild] = val;
      }
```

14 points, Criteria:

- resize if necessary, 4 points
- calculate index of parent correctly, 2 points
- increment size, 2 points
- stop when val is new root, 2 points
- stop when val is <= parent, 2 points
- bubble parent down correctly, 2 points

Other:

- O(N) -5
-