0

0.1 Instance Variables (3 points)

private String myName;
private int myAge;
private int myHashCode;

0.2 Constructor (3 points)

public LongNameFirst(String name, int age)
{
    myName = name;
    myAge = age;
    computeHashCode();
}

0.3 compareTo (5 points)

public int compareTo(LongNameFirst other)
{
    if (myName.length() > other.myName.length()) {
        return -1;
    }
    if (myName.length() < other.myName.length()) {
        return 1;
    }
    if (myAge < other.myAge) {
        return -1;
    }
    if (myAge > other.myAge) {
        return 1;
    }
    return 0;
}
0.4  **hashCode (3 points)**

```java
private void computeHashCode()
{
    // One possibility.
    String temp = myName + myAge;
    myHashCode = temp.hashCode();
}

public int hashCode()
{
    return myHashCode;
}
```

0.5  **equals (2 points)**

```java
public boolean equals(Object otherObject)
{
    if(otherObject == null) {
        return false;
    }

    if(getClass() != otherObject.getClass())
        return false;

    LongNameFirst other = (LongNameFirst)otherObject;

    return compareTo(other) == 0;
}
```
int x = 42;

void doSomething() {
    System.out.println("Go!");
}

for (String i : v) {
    System.out.println(i);
}
2

2.1 3 points
6  8  6

2.2 3 points
10 10 10

One point each.
3

3.1 Running Times

A
$O(1)$, as it does constant work (returning) that’s independent of $n$.

B
$O(n)$, as generating result is $O(1)$, iterating is $O(n)$ as it does $n/2$ work, and returning is constant.

C
$O(n^2)$, as the inner loop is $O(n)$ (see B) and the outer loop is $O(n)$.

D
$O(n^2)$. The first pair of loops is $O(n^2)$, and the second loop is $O(n)$. Recall that $O(n^2) + O(n) \in O(n^2)$.

E
$O(\sqrt{n})$. Consider taking the square root of both sides of the while check. In that case, the left-hand-side is growing linearly towards $\sqrt{n}$.

F
$O(\log n)$. Growing by powers of 2 is the very definition of $\log n$.

3.2 Ordering

It’s A–F–E–B–(C, D). Note the tie.
I'd use `modeTwo`, as it has better big-O running time. Specifically, the two nested loops in `modeOne` mean that it runs in $O(n^2)$ time. The use of the HashMap in `modeTwo` means that it runs in $O(n)$ time, which is far better, particularly on large inputs.
5 7 points

Basically a copy-and-paste of the code from the previous question. Note that the details of the helper method are, in fact, irrelevant.
public ArrayList<String> computeUniqueWords(ArrayList<String> words) {
    ArrayList<String> result = new ArrayList<String>();
    result.add(words.get(0));
    for (int i = 1 ; i < words.length ; ++i) {
        if (!result.get(result.size() - 1).equals(words[i])) {
            result.add(words[i]);
        }
    }
    return result;
}
public static boolean areMatched(String s) {
    if (s.length() == 0) {
        return true;
    }
    if (s.length() == 1) {
        return false;
    }
    boolean paren =
        s.charAt(0) == '(' && s.charAt(s.length() - 1) == ')';

    boolean brace =
        s.charAt(0) == '{' && s.charAt(s.length() - 1) == '}';

    if (!(paren || brace))
        return false;

    return areMatched(s.substring(1, s.length() - 1));
}