README.md

Assignment 2 (White Box Testing)

We have four functions

- insert
- member
- section
- containsArithTriple

We will go through the test cases required and bug fixes for each funciton one by one.

insert(int x)

Bugs

The simplest way to fix the bug would be to replace the `break` statments with `return` statements.
Adding a condition before running the last `a.add(x)` to check if the iteration is completed would be another way.

```diff
   public void insert(int x) {
      for (int i = 0; i < a.size(); i++) {
        if (a.get(i) > x) {
          a.add(i, x);
-         break;
+         return;
        } else {
          if (a.get(i) == x) {
-           break;
+           return;
          }
        }
      }
      a.add(x);
   }
```

Statement Coverage

We have three scenarios here.

- Add new element in the middle
- Add new element at the end
- Add existing element

So, we have three test cases. All of them start with the state of `[1, 3, 4]`.

```java
public class TestInsert {

    Set s = new Set();

    @Before
    public void setup() {
       s.insert(1);
```

```
        s.insert(4);
        s.insert(3);
     }

     @Test
     public void insertTest1() {
         s.insert(2);
         assertEquals("[1, 2, 3, 4]", Arrays.toString(s.toArray()));
     }

     @Test
     public void insertTest2() {
         s.insert(5);
         assertEquals("[1, 3, 4, 5]", Arrays.toString(s.toArray()));
     }

     @Test
     public void insertTest3() {
         s.insert(4);
         assertEquals("[1, 3, 4]", Arrays.toString(s.toArray()));
     }

 }
```

Branch Coverage

In this case, branch coverage is also satisfied with these three test cases.

member(int x)

Bugs

No bugs in this method.

Statment Coverage

We again have three scenarios here.

- element found
- element not found by middle
- element not found by end

So, we have three test cases. All of them start with initial state of `[1, 3, 4]` .

```
 public class TestMember {

     Set s = new Set();

     @Before
     public void setup() {
        s.insert(1);
        s.insert(3);
        s.insert(4);
     }

     @Test
     public void memberTest1() {
         assertEquals(true,s.member(4));
     }
```

```
    @Test
    public void memberTest2() {
        assertEquals(false,s.member(2));
    }

    @Test
    public void memberTest3() {
        assertEquals(false,s.member(5));
    }

}
```

Branch Coverage

In this case, branch coverage is also done with these three test cases.

section(Set s)

Bugs

When an element is removed from the array list, the same index now points to the next element to be compared with.
So, we should not increment the pointer.

```
   public void section(Set s) {
      for (int i = 0, j = 0; i < a.size() && j < s.a.size();) {
        if (a.get(i).equals(s.a.get(j))) {
          a.remove(i);
-         i++;
          j++;
       } else {
          if (a.get(i) < s.a.get(j)) {
            i++;
          } else {
            j++;
          }
        }
      }
    }
  }
```

Statement Coverage

Even though we have multiple scenarios, we can cover all statments with a single test case.
We initialise the two sets with `[1, 3, 4]` and `[1, 2, 3, 5]` .

```
public class TestSection {

    Set s1 = new Set();
    Set s2 = new Set();

    @Test
    public void sectionTest1() {
        s1.insert(1);
        s1.insert(3);
        s1.insert(4);

        s2.insert(1);
        s2.insert(2);
        s2.insert(3);
        s2.insert(5);

        s1.section(s2);
        assertEquals("[4]", Arrays.toString(s1.toArray()));
    }

}
```

Branch Coverage

For branch coverage we need one more test case here.
The reason being the condition

```
for (int i = 0, j = 0; i < a.size() && j < s.a.size();)
```

The code can exit if *i* or *j* crosses the size. This leads to two branches.
Our test cases only takes one branch where i crosses. So, we add a new test case.

```
    @Test
    public void sectionTest2() {
        s1.insert(1);
        s1.insert(3);
        s1.insert(4);

        s2.insert(1);
        s2.insert(2);
        s2.insert(3);

        s1.section(s2);
        assertEquals("[4]", Arrays.toString(s1.toArray()));
    }
```

containsArithTriple()

Bugs

If *i* is allowed to be equal to *j*, then `member(2 * a.get(i) - a.get(j))` will always be true.

```diff
    public boolean containsArithTriple() {
      for (int i = 0; i < a.size(); i++) {
-       for (int j = 0; j <= i; j++) {
+       for (int j = 0; j < i; j++) {
          if (member(2 * a.get(i) - a.get(j))) return true;
        }
      }
      return false;
    }
```

Statement Coverage

We have two scenarios. The set may or maynot contain triple.
So, we have two test cases.

```
public class TestTriple {

    Set s = new Set();

    @Test
    public void tripleTest1() {
        s.insert(1);
        s.insert(3);
        s.insert(5);
        assertEquals(true, s.containsArithTriple());
    }

    @Test
    public void tripleTest2() {
        s.insert(1);
        s.insert(3);
        s.insert(4);
        assertEquals(false, s.containsArithTriple());
    }

}
```

Branch Coverage

In this case, branch coverage is also satisfied with the same test cases.

Results

The test files with 100% statement coverage are here.
The test files with 100% branch coverage are here.
Code coverage results are here and you can see it online here.

Test results for original code with bugs is here and you can see it online here.
Test results for modified code is here and you can see it online here.

The original `Set.java` is here.
The new `Set.java` is here.
The patch file for modifications (diff file) is here.