Tester File Output (It is filled with the times from the majority method)

```
Starting 5.30 Test
Finished testing 5.30
Starting Prime Test
Method returned true on the following non-prime 121
Finished testing primes
Starting majority element test
459299
72100
81400
65600
12499
51699
52101
19999
78100
24200
18501
44700
43699
34600
13301
20800
49500
28701
42800
10899
Finished testing majority element
```

User Input file:

```
Please enter a large integer that you are just dying to know if it is prime or not
459673
That number is NOT prime

Please enter 5 integers
14
61
36
76
96
There is NOT a location where [i] == i
110900
The array does NOT have a majority element
PS C:\Users\easto\OneDrive\Documents\CS 2420\CS-2420>
```

This method just checks to see if there is an index that matches the value in an array. It basically just runs all the way through the element and if it runs into one it returns true.

```java
public static boolean doesTheArrayMeetCriteria_5_30(int [] a) {
    //TODO Add code here that determines whether an integer i e
    //TODO What is the runtime of your algorithm?
    for (int i = 0; i < a.length; i++) {
        if (a[i]==i) {
            return true;
        }
    }
    return false;
```

This code runs every number less than the sqrt of the given integer and checks to see if it divides into it with no remainder.

```java
public static boolean isAPrime(int num) {
    //TODO Add code here.  I am assuming you will be div
    for (int i = 2; i < Math.sqrt(num); i++) {
        if (num%i==0) {
            return false;
        }
    }
    return true;
```

This code is a little more involved when we check for a majority element in an array. The code itself just sets a temp int and then it runs through each index of the array and if it has the same value as the temp int, it adds 1 to the counter. If the counter ever exceeds half of the length of the array. It'll just return true.

```java
public static boolean majorityElement(int [] a) {
    //TODO Finish this method appropriately
    int temp;
    int count=0;
    Long startTime = System.nanoTime();
    boolean majority = false;
    for (int i = 0; i < a.length; i++) {
        temp = a[i];
        for (int j = 0; j < a.length; j++) {
            if (temp==a[j]) {
                count++;
            }
            if (count>a.length/2){
                majority=true;
            }
        }
        count=0;
    }
    Long endTime = System.nanoTime();
    Long totalTime = endTime - startTime;
    System.out.println(totalTime);
    return majority;
}
```

Big Oh Tester output

```
5.30: Median time for size         100 is: 2201
5.30: Median time for size       1,000 is: 18901
5.30: Median time for size      10,000 is: 164200
5.30: Median time for size     100,000 is: 117400
Prime: Median time for num bits      5 is: 300
Prime: Median time for num bits     10 is: 1301
Prime: Median time for num bits     15 is: 9001
Prime: Median time for num bits     20 is: 29400
Prime: Median time for num bits     25 is: 187400
Prime: Median time for num bits     30 is: 284401
Majority Element: Median time for size         100 is: 324901
Majority Element: Median time for size       1,000 is: 734900
Majority Element: Median time for size      10,000 is: 42023501
Majority Element: Median time for size     100,000 is: 2447907899
PS C:\Users\easto\OneDrive\Documents\CS 2420\CS-2420>
```

	A	B	C
1	5.3	Median time for size 100 is	2201
2	5.3	Median time for size 1,000 is	18901
3	5.3	Median time for size 10,000 is	164200
4	5.3	Median time for size 100,000 is	117400
5	Prime	Median time for num bits 5 is	300
6	Prime	Median time for num bits 10 is	1301
7	Prime	Median time for num bits 15 is	9001
8	Prime	Median time for num bits 20 is	29400
9	Prime	Median time for num bits 25 is	187400
10	Prime	Median time for num bits 30 is	284401
11	Majority Element	Median time for size 100 is	324901
12	Majority Element	Median time for size 1,000 is	734900
13	Majority Element	Median time for size 10,000 is	42023501
14	Majority Element	Median time for size 100,000 is	2447907899