Answers to lab 6 questions

Problem 1

(a) *(0.5 points)* How many triples of integers in the input file `4Kints.txt` sum to zero?

Answer: 4039

(b) *(4.5 points)* What is the *exact* number of times the `if` statement on line 10 in Snippet 1 gets executed? Give an expression in terms of n.

Answer: This is the number of ways of picking 3 different elements from the input array of length n.

$$\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$$

Problem 2

In this problem we had to produce supporting evidence for the hypothesis that

$$T(n) = an^3, \quad \text{as } n \to \infty \quad (\text{ “as } n \text{ grows large” }), \tag{1}$$

where a is some constant, $T(n)$ is the running time of `ThreeSum.java`, and n is its input size (the number of integers in the input array). We also had to estimate a.

(a) *(2 points)* Run `DoublingTest.java`. As you watch the output, you'll notice that the first few lines are printed pretty quickly, but then your program slows down significantly, taking its sweet time to print the last few lines.

The output you see has two columns. Explain in one sentence what the numbers in the first column are. Explain in one sentence what the numbers in the second column are.

Answer: The first column has the number of elements in the input array, n. The second column shows $T(n)$, the time in seconds it takes `ThreeSum` to process an array of n elements.

(b) *(1 point)* Suppose the input to `ThreeSum.java` is an integer array of 8,000 6-digit integers. On your computer, how many seconds does it take `ThreeSum.java` to find the number of triples of these integers that sum to zero?

Answer: The answer varies, but an answer in the range of two-four minutes is reasonable. Table 1 shows that one particular run of `ThreeSum.java` took 137 seconds to process an array of 8000 integers.

(c) *(2 points)* Based on the output you obtained in part (a) as a result of running `DoublingTest.java`, determine the constant a of Equation 1.

Answer: Based on Table 1, a is estimated to be $2.63671875 \times 10^{-10}$. (A way to estimate a is discussed in the hint to this problem.)

`DoublingTest` Run Results		
Input Size n	Runtime $T(n)$ (seconds)	$T(2n)/T(n)$
250	0.00	–
500	0.00	–
1000	0.00	–
2000	2.00	–
4000	17.00	8.5
8000	137.00	8.06
16000	1092.00	7.97
32000	8768.00	8.03
...		

Table 1: The first two columns show an output of `DoublingTest.java`. The last column suggests that the runtime of `ThreeSum.java` is cubic. A cubic curve through this sample data is shown in Figure 1.

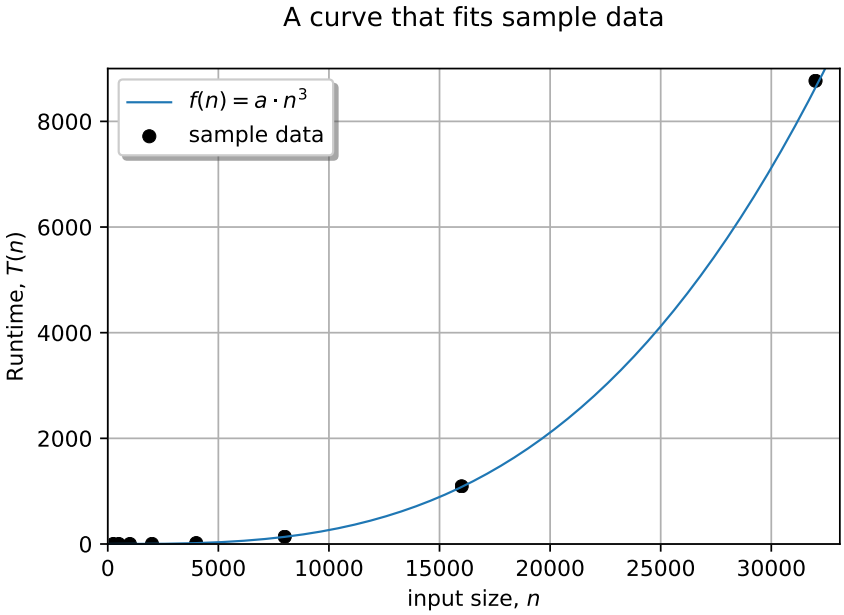

Figure 1: Sample data and the cubic curve $f(n) = a \cdot n^3 = 2.63671875 \times 10^{-10} \cdot n^3$.

Problem 3

(a) *(2 points)* What is the *exact* number of times the `if` statement on line 9 in Snippet 3 on page 6 gets executed? Give an expression in terms of n.

Answer: This is the number of ways of picking 2 different elements from the input array of length n.

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

(b) *(1 point)* How many pairs of integers in `16Kints.txt` sum to zero?

Answer: 66

(c) *(1 point)* What is the Big-O estimate of the method `TwoSum.count()` in Snippet 3 (lines 4 – 14)?

Answer: $O(n^2)$

Problem 4

(a) *(5.5 points)* What is the Big-O running time of the method `TwoSumFast.count()` in Snippet 4? Assume that the call to `Arrays.sort()` on line 5 takes $O(n \log n)$ time.

Answer: $O(n \log n)$

Line 5 (sorting) in Snippet 4 takes $O(n \log n)$ time. Lines 8 – 10 perform binary search n times. Since binary search runs in $O(\log n)$ time, the total time taken by lines 8 – 10 is $O(n \log n)$. The total time (about twice $n \log n$ time) is still $O(n \log n)$.

(b) *(0.5 points)* How many pairs of integers sum to zero in the following file of 1 million integers: `1Mints.txt`.

Answer: 249838

Problem 5

A solution is given in snippet 5. Again, the integers in the input array are assumed to be distinct. Our solution is based on the observation that

A pair `a[i]` and `a[j]` is part of a triple that sums to zero if and only if the value `-(a[i] + a[j])` is in the array (but not `a[i]` or `a[j]`).

The code in snippet 5 sorts the array, then does $n(n-1)/2$ binary searches, each taking $O(\log n)$ time, for a total time of $On^2(\log n)$.

Answer: ?

```java
import java.util.Arrays;

public class ThreeSum {
   public static int count(int[] a) {
      int n = a.length;
      int count = 0;
      for (int i = 0; i < n; i++) {
         for (int j = i+1; j < n; j++) {
            for (int k = j+1; k < n; k++) {
               if (a[i] + a[j] + a[k] == 0)
                  count++;
            }
         }
      }
      return count;
   }

   public static void main(String[] args) {
      In in = new In("8Kints.txt");
      int[] a = in.readAllInts();

      System.out.println("The original array of ints: " + Arrays.toString(a));

      long startTime = System.currentTimeMillis();
      System.out.println("Count is: " + ThreeSum.count(a));
      long endTime = System.currentTimeMillis();
      long timeElapsed = endTime - startTime;
      System.out.println("Execution time in milliseconds  : " + timeElapsed);
   }
}
```

Snippet 1: A naïve solution to the three-sum problem.

```java
import java.util.Random;

// DoublingTest class provides a client for measuring
// the running time of a method using a doubling test.
public class DoublingTest {
   private static final int MAXIMUM_INTEGER = 1000000;

   // This class should not be instantiated.
   private DoublingTest() { }

   // Returns the amount of time (in seconds) to call
   // ThreeSum.count() with n random 6-digit integers.
   public static double timeTrial(int n) {
      int[] a = new int[n];

      for (int i = 0; i < n; i++) {
         a[i] = -MAXIMUM_INTEGER + 1 + new Random().nextInt(2* MAXIMUM_INTEGER - 1) ;
      }
      long startTime = System.currentTimeMillis();
      ThreeSum.count(a);
      long endTime = System.currentTimeMillis();
      return (endTime - startTime)/1000;
   }

   // Prints table of running times to call ThreeSum.count()
   // for arrays of size 250, 500, 1000, 2000, and so forth.
   public static void main(String[] args) {
      for (int n = 250; n <= 64*250; n += n) {
         double time = timeTrial(n);
         System.out.printf("%7d %7.2f\n", n, time);
      }
   }
}
```

Snippet 2: Doubling Test for the Tree-Sum problem.

```java
import java.util.Arrays;

public class TwoSum {
   public static int count(int[] a) {
      int n = a.length;
      int count = 0;
      for (int i = 0; i < n; i++) {
         for (int j = i+1; j < n; j++) {
               if (a[i] + a[j] == 0)
                  count++;
         }
      }
      return count;
   }

   public static void main(String[] args) {
      In in = new In("16Kints.txt");
      int[] a = in.readAllInts();

      System.out.println("The original array of ints: " + Arrays.toString(a));
      System.out.println("Count is: " + TwoSum.count(a));
   }
}
```

Snippet 3: A naïve solution to the two-sum problem.

```java
import java.util.Arrays;

public class TwoSumFast {
   public static int count(int[] a) {
      Arrays.sort(a);
      int n = a.length;
      int count = 0;
      for (int i = 0; i < n; i++)
         if (BinarySearch.indexOf(a, -a[i]) > i)
            count ++;
      return count;
   }

   public static void main(String[] args) {
      In in = new In("8Kints.txt");
      int[] a = in.readAllInts();
      System.out.println("Count is: " + count(a));
   }
}
```

Snippet 4: A solution to the two-sum problem. This solution is faster than the solution in Snippet 3. Problem 4 asks to determine the Big-O running time of `TwoSumFast.count()`.

```java
import java.util.Arrays;

public class ThreeSumFast {
   public static int count(int[] a) {
      // How many triples sum to zero?
      Arrays.sort(a);
      int n = a.length;
      int count = 0;
      for (int i = 0; i < n; i++)
         for (int j = i + 1; j < n; j++)
            if (BinarySearch.indexOf(a, -a[i]-a[j]) > j)
               count++;
      return count;
   }

   public static void main(String[] args) {
      In in = new In("1Mints.txt");
      int[] a = in.readAllInts();

      //System.out.println("The original array of ints: " + Arrays.toString(a));

      long startTime = System.currentTimeMillis();
      System.out.println("Count is: " + ThreeSumFast.count(a));
      long endTime = System.currentTimeMillis();
      long timeElapsed = endTime - startTime;
      System.out.println("Execution time in milliseconds  : " + timeElapsed);
   }
}
```

Snippet 5: A solution to the three-sum problem. This solution is faster than the solution in Snippet 1.