Advanced Java – Tyler Hunt
Week 3 – Chapter 7 (Single-Dimensional Arrays) HOMEWORK

Note that you have *two* programs to write this week. Finish them both, and submit them together in the Moodle drop box as usual.

SORT AND MERGE

Write a program that performs the following tasks:

- ~~Display a friendly greeting to the user~~
- ~~Prompt the user for a file name (a list of numbers to be sorted)~~
- ~~Accept that file name~~
- ~~Prompt the user for a second file name (another list of numbers to be sorted)~~
- ~~Accept that file~~ name
- ~~Open and read the data in the first file~~
- ~~Sort that data using an algorithm that you wrote yourself (*not* Array.sort)~~
    - ~~One of the lists will be sufficiently large that a "simple" algorithm won't suffice~~
- ~~Open and read the data in the second file~~
- ~~Sort that data, again using your own algorithm~~
- ~~Merge the two sorted lists into a single sorted file (see Problem 7-31) using a linear algorithm.~~
- ~~Prompt the user for a file name~~
- ~~Create that file and dump the merged list to the file, using the format specified~~

The file format is as follows: a single integer, followed by *that number* of integers. That way you can open the file, read the first value, create an array of the appropriate size, and then read the rest of the file to populate the array. For example

```
10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

is data in the specified format. The first 10 is the count, *not* a data item; the second one is. Note that there can be more data items than indicated by the count; ignore the extras.

Every discrete task should be implemented as an independent method. At a minimum, you will have one sorting method and one merging method.

Submit your Java source code in the Moodle drop box as usual.

SIEVE OF ERATOSTHENES

The Greek mathematician Eratosthenes (276 BCE – 195 BCE) devised a method of finding prime numbers, as follows:

- Create a list of consecutive integers from 2 up to the desired limit.
- Beginning with 2, mark off every second integer on the list.
- Move to the next *unmarked* number (which will be 3) and mark off every third integer.
- Move to the next *unmarked* number (which will be 5) and mark off every fifth integer.
- Move to the next *unmarked* number (which will be 7) and mark off every seventh integer.
- Move to the next *unmarked* number (which will be 11) and mark off every eleventh int…
- Continue until the end of the list has been reached.

When completed, every unmarked entry in the list (beginning with 2) represents a prime number. Implement the Sieve of Eratosthenes with a program which performs the following tasks:

- ~~Display a friendly greeting to the user.~~
- ~~Prompt the user for a start value > 1. We will use this later.~~
- ~~Prompt the user for a stop value. This will be the size of the array – 1.~~
- ~~Use the stop value to create an array of type **boolean** and set every value to true.~~
- ~~Execute the Sieve of Eratosthenes algorithm (using false to mark off entries)~~
- ~~Display the total number of primes found in the interval [start, stop].~~

What is the largest array size you can run using Java? If your program can handle arrays of larger than 2 GB, make sure you input the user's integer as a long, not an int. If not, explain why.

- *With the way, the program was initially built using an array of Booleans, the Boolean array does not accept a long as an input, so therefore the max number that can be used in Integer.MAXVAULE or* 2^{31}

Use your program to answer the following questions:

What is the largest sieve that your program can (in principle) complete?
What is the significance of that value? How could you overcome that limitation?
How long does (or would) it take your program (in seconds) to compute a sieve…

… up to 100,000?
- *There are 9592 prime(s) that are > 2 and <= 100000*
  Elapsed time is 0.005287 seconds..

… up to 1,000,000?
- *There are 78498 prime(s) that are > 2 and <= 1000000*
  Elapsed time is 0.023659 seconds..

… up to 10,000,000?
- *There are 664579 prime(s) that are > 2 and <= 10000000*
  Elapsed time was 52 milliseconds.

… up to 100,000,000?

- *There are 5761455 prime(s) that are > 2 and <= 100000000*
  Elapsed time was 579 milliseconds.

… up to 1,000,000,000?

- *There are 50847534 prime(s) that are > 2 and <= 1000000000*
  Elapsed time was 11 seconds.

… up to 1,000,000,000,000?

- *Error*

What is the asymptotic performance of the Sieve of Eratosthenes?

- $O(\frac{n\sqrt{n}}{logn})$

How many prime numbers exist in each of the ranges given above (see above)? Compute as many as you can, and explain why some can't be done, at least not in time for class next week.

- *Due to the available memory and the available size restrictions of the int type in java is was unable to compute the last value of 1,000,000,000,000.*
- *If I had more available time, I planned on using an input of long and then parseing it into smaller int sized chunks, then recursively calling the function to calculate the result (obviously, I would only be able to go to the Long.MAXVALUE). From their I guess you could do the same using BigInt class.*

Submit your Java source code using the Moodle drop box. Also write a brief report giving these answers and include it in your submission.