Tashfeen, Ahmad
CSCI 2114: Data Structures

Homework 3

Question 1. Point your favourite web-browser to `https:www.google.com`. Perform a search for the term, "recursion". Why did you misspell it? What happened?

Some developer at Google added a silly little joke to the search engine where "recursion" is always corrected to "recursion"— a joke about the nature of recursion.

Question 2. Watch the YouTube video Programming Loops vs Recursion - Computerphile. Did you learn something new? Can you think of something where you might opt for recursion over loops?

I didn't know that older languages lacked recursion for users, nor did I know that recursion is used in compilers. I think finding prime numbers is better done with recursion.

Question 3. Write a `public class Sorter` in Java that implements the Java interface[1] given in listing **??**. The sorter class may not have any methods other than those which exist in the Godric's Hat interface.

The method `public void counting(int[] array)` is the same as `public static void sortIntegers(int[] toSort)` from the last homework. If you wish, you may use the implementation from there.

While the following two methods,

```
void merge(int[] array);                // use recursion only
void quick(int[] array, int p, int r); // use recursion only
```

must be implemented using recursion. The method,

```
void quickLoopy(int[] array);
```

must be implemented iteratively, i. e., using only loops. The only import statement you may have in the sorter class should import the stack,

```
import java.util.Stack;
```

The four algorithms you are implementing by implementing the interface are,

1) Insertion Sort
2) Merge Sort
3) Quick Sort (recursively)
4) Quick Sort (loopy)
5) Counting Sort (sometimes called a different name)

Use the class `Test.java` to test your logic once your `Sorter.java` compiles. You'll need to put it under the same directory as the sorter class and then compile and run. Change the `int n = 7;` within `Test.java` to a bigger number once it works for smaller ones.

Question 4. Download the experiment class. Place it in the same directory as the `Sorter.java` and `GodricsHat.java` and run,

```
javac Experiment.java
java  Experiment
```

The output will be six rows of ordered pairs (n elements, t milliseconds). The first row corresponds to the insertion sort, the second to the merge sort, the third to the recursive quick sort, the fourth to the loopy quick sort, the fifth to the counting sort and the finally the last one corresponds to the Java's built-in `Arrays.sort(int[] a)`.

Do you remember which algorithm `Arrays.sort(int[] a)` used from the previous assignment?

I believe it was Timsort.

Plot each row where n is on the x-axis and t is on the y-axis. Label your plots appropriately. You'll need to create *two plots*. The first plot should have the data from all the six rows and the second plot should have the data from all but the first row, i. e., skipping the times for the insertion sort.

[1] Java interfaces are just abstract classes where all the interface methods are left to be implemented. Furthermore, a Java class may implement more than one interface but inherit from only one parent class.

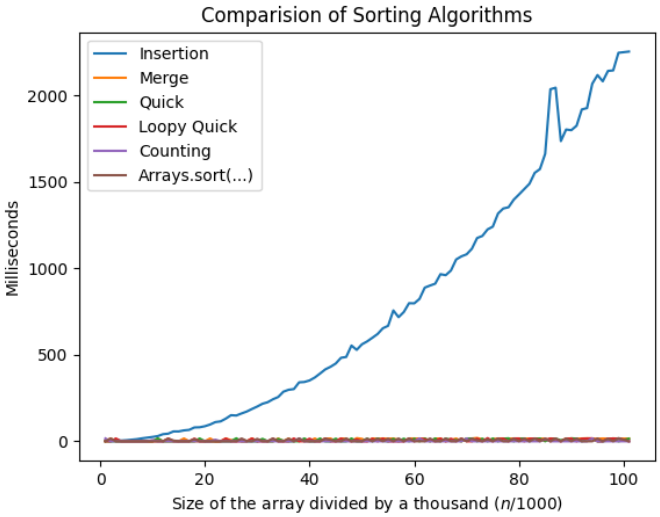

FIGURE 1. Enter Caption

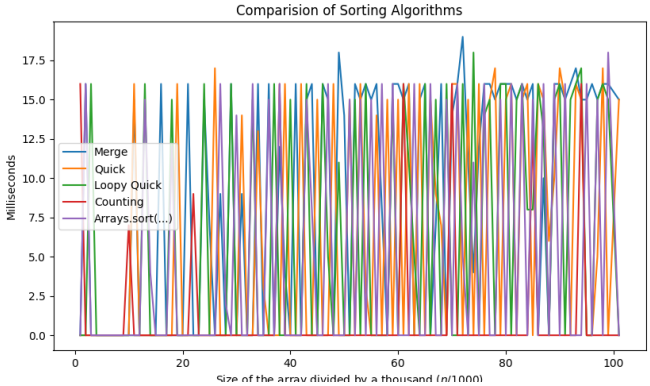

FIGURE 2. Enter Caption

If you know your way around some Python, Numpy and Matplotlib then run `java Experiment > data.txt` to redirect the output to a text file and use this python program to build your plots. Comment out the line number 15 for the second plot.

Other than that, the simplest way is to use `https://www.desmos.com` or however[2] you do plots.

Question 5. Analyse the plots from question 4. Which algorithm was the worst? Does it match its asymptotic runtime upper-bound? Did any of our implementations beat Java's built-in sorting algorithm?

Insertion was the worst by far. Counting is better than java's built in.

Question 6. Can you implement the quick sort algorithm without recursion or an explicit stack?

I don't believe there is any way to.

Question 7. Around nine minutes in the video from question 2, Brady asks professor Brailsford about a more real life use-case of recursion where it is not just nice for implementation's sake but also makes the runtime better. Do you now have an answer for such a use-case?

Since quicksort basically can't exist without recursion(or a stack), sorting can be made better with recursion.

[2]Do not draw any plots by hand.

Question 8. Among the ones you implemented, which algorithm do you still think you don't quite understand? Why?

I think I get them all pretty well, but quicksort is still the hardest to understand.

Submission Instructions

1) Turn in a PDF containing any plots and answers from the homework.
2) Also turn in your `Sorter.java`. Note that since you may not change anything in either `Experiment.java` or `GodricsHat.java`, there is no need to turn those in.