Wiyninger, Caleb
CSCI 2114: Tashfeen's Data Structures
September 19, 2025

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?

I didnt, it recursively links to the same page

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?

properly parsing nested code. Eg: A loop in a loop in a function in a class being called from another class. Depth first search.

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 `counting(int[] array)` is the same as `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 the interface `GodricsHat.java` and run,

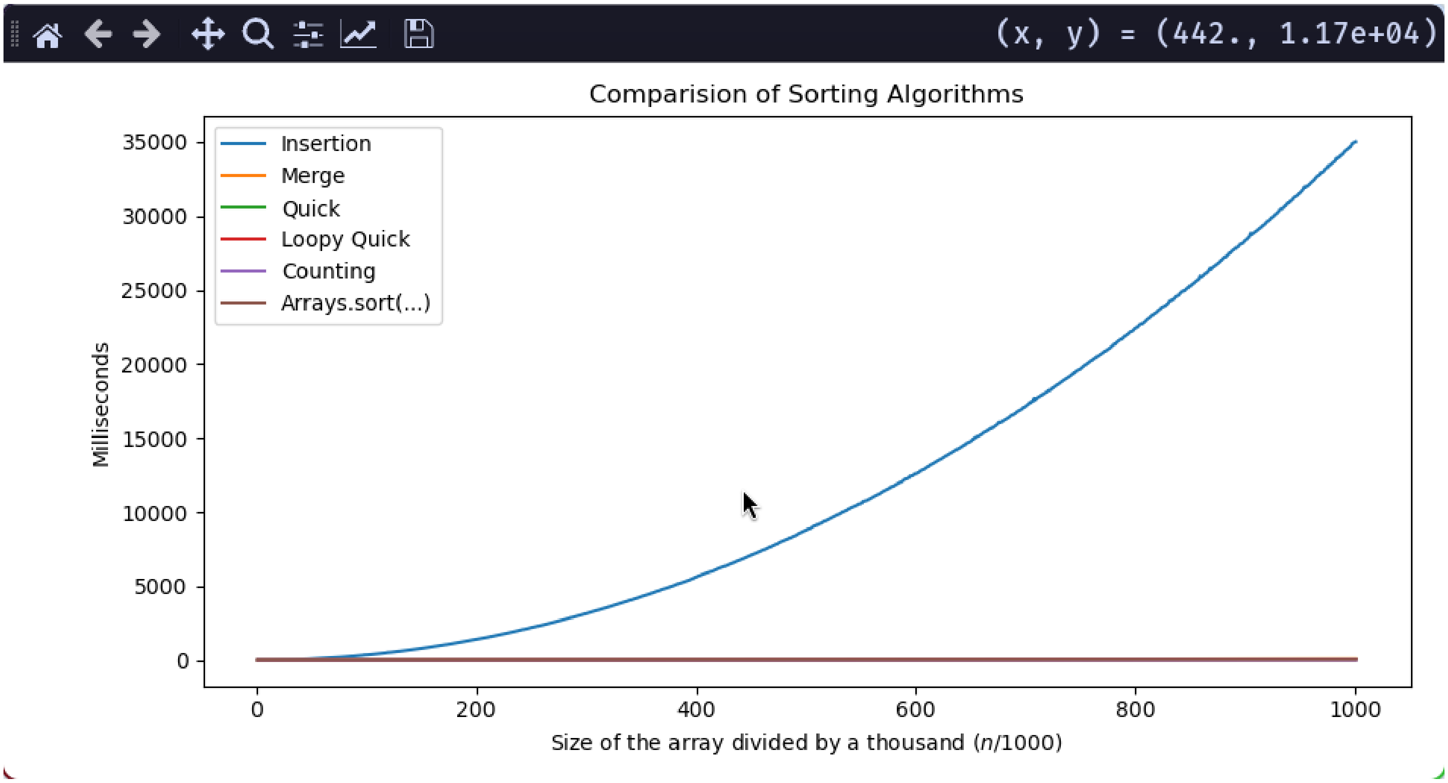
```
javac Experiment.java
java  Experiment
```

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?

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.



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?

1) Insertion sort took the longest on average
2) Yes
3) Counting and quick loopy

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

No

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?

Sorting things alphabetically. I create a stack for each first letter then sort those stacks by second letter until everything is in order

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

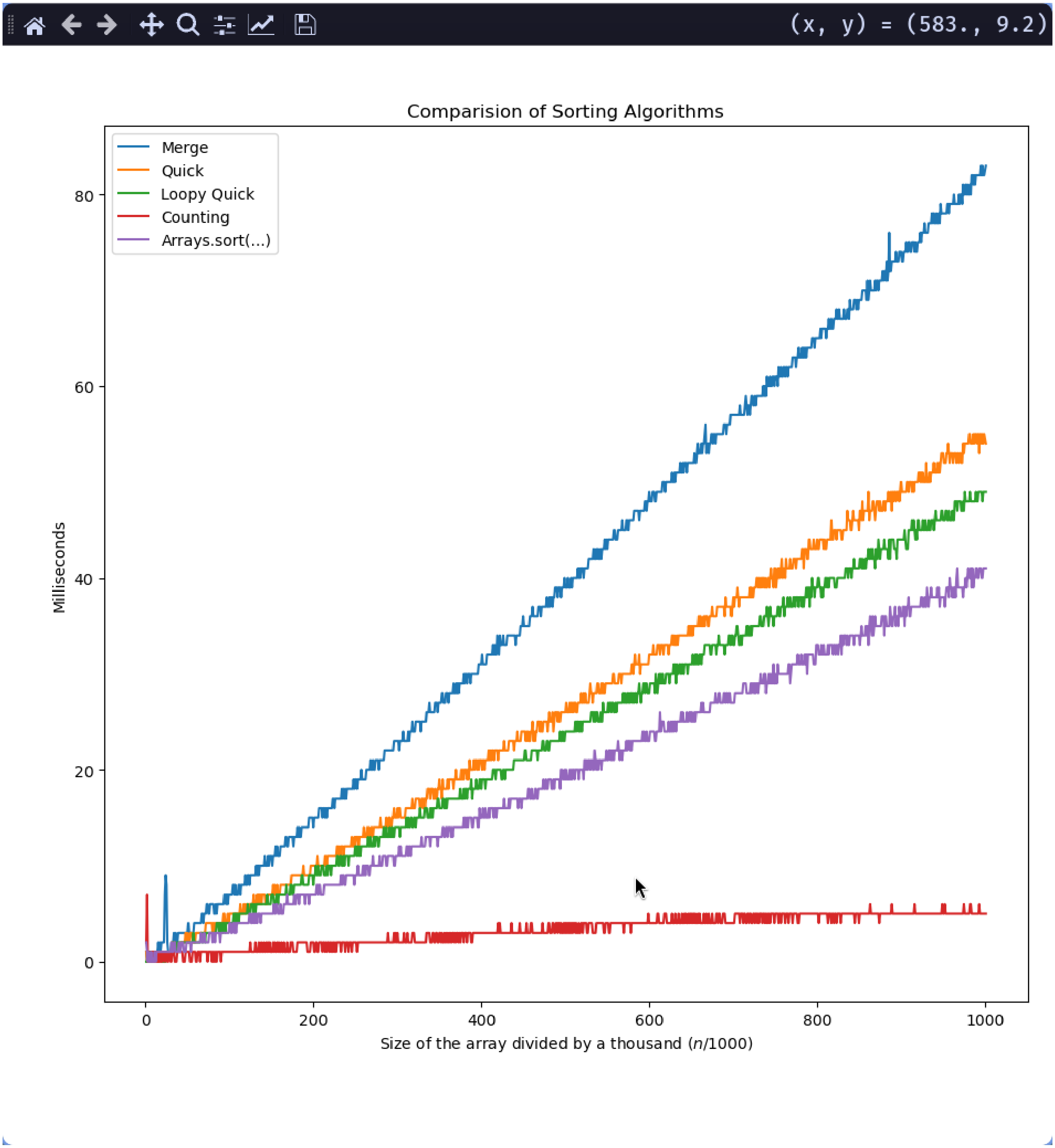
I still dont entirely understand quick/quickLoopy. I get it splits the array into partitions and compares between a cursor and pivot but i dont get it intuitively or understand why its better than others

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.

Oklahoma City University, Petree College of Arts & Sciences, Computer Science

[2] Do not draw any plots by hand.


(x, y) = (583., 9.2)
Comparision of Sorting Algorithms
Merge
Quick
Loopy Quick
Counting
Arrays.sort(...)
Milliseconds
0
20
40
60
80
0
200
400
600
800
1000
Size of the array divided by a thousand (n/1000)