CSC 371 Project 1: BioSort (for Genetically Engineering Dragons) *Fall 2025*

Collaboration Rules

If any person (tutor, classmate, etc) or other resource (website, book) contributed to something that you turn in, you should always CITE that source (e.g. in a code "comment"), to avoid plagiarism/academic dishonesty. You are encouraged to do online research for this project as long as you cite any resources that you use. I should always be able to clearly tell what work originated from you, and what contributions (if any) came from external sources.

Using ChatGPT or other LLM-based A.I. for this project is **discouraged**, but it is allowed (since I am not confident I could reliably/fairly enforce a ban). It is my hope that this task is still too challenging for the A.I. to do very well at it. However, I am curious about how clever the AI has gotten. **You MUST CITE (with a code comment) any code that was written by AI, and which A.I. (e.g. ChatGPT, Gemini, GitHub Copilot, etc) that you used.**

This is an individual project. You can talk to other students and offer/receive help, and assist others with debugging, but you should not share full code, copy other people's code, or let others copy your code. Students turning in plagiarized code will receive a zero and are subject to the consequences of an Honor Code violation.

Objective: Design/implement/analyze the MOST efficient sorting algorithm you can, using as basic building blocks only the specified operations.

Fictitious motivation: You have been hired by **Dracarys Bio Labs** to genetically engineer a dragon, using DNA from a variety of lizards, dinosaurs, and hydrothermal vent fauna. As part of this process, you need an algorithm to sort biological molecules. Your lab has designed a bio-robot, but due to its unique construction, it is capable of processing multiple elements. (Instead of the typical built-in operations for comparing or swapping **two** elements, the robot hardware has 4 nano-arms with sensors and one cut&flip arm, so it supports *O(1)* operations for:

- **flip** - reverses any contiguous subsequence of the array from index i to j.
  $(a_i, a_{i+1}, \ldots, a_{j-1}, a_j) \longrightarrow (a_j, a_{j-1}, \ldots, a_{i+1}, a_i)$

  Example: If the bio-array `arr` is `[7, 3, 8, 2, 9, 4]`
  if we call `arr.flip(0,3)`, the array will be modified to `[2, 8, 3, 7, 9, 4]`
  if instead we call `arr.flip(3,5)`, that would result in `[7, 3, 8, 4, 9, 2]`

- **compare4** - given the indexes of 4 array elements, this method returns a 4-element array containing those 4 indexes, but provided in non-decreasing order of the values at those indices.

  Example: if the bio-array `arr` is `[7, 3, 8, 2, 9, 4]`, and we call `arr.compare4(0,1,2,3)`, we are asking among these indices, which is the lowest, second lowest, third lowest, and the highest, and it would return the int[] array **`[3, 1, 0, 2]`** because the lowest value occurred at index 3, the 2nd lowest at index 1, the third lowest at index 0, and the highest at index 2.

- **isNonDecreasing4** – this is a convenience method (that also costs one "compare4") you can use to check if the values at four indices are currently in sorted (non-decreasing) order. It's equivalent to checking if `compare4(w,x,y,z)` returns `[w,x,y,z]` in that order.

  Example: if the bio-array `arr` is `[7, 6, 5, 9, 8]`, then
  `arr.isNonDecreasing4(0,1,2,3) → false` because `7,6,5,9` **isn't** non-decreasing, but
  `arr.isNonDecreasing4(2,0,0,4) → true` because `5,7,7,8` **is** non-decreasing.

Your goal: design the fastest possible algorithm for sorting of random BioArrays to bio-engineer a dragon.

Important!!! For the physical robot, rearranging molecules is much slower than scanning them. Technicians estimate that a **flip** op will take **1 second** (regardless of the length of the subarray being flipped), whereas a **compare4** op is **125 times faster**, and will only take **8 milliseconds**.

estimatedRobotTime (N) = *(# of flip ops for array size N)* + 0.008 x *(# of compare4 ops for array size N)*

Dracarys Bio Labs needs you to minimize the average-case for the performance metric above specifically for **N = 2000**, since that is the most common length of molecular array they expect the robot to process. Also, we will assume that each bio-array starts in a uniformly random order (not already partially-sorted, or reversed, or any other pattern), and the array is unlikely to contain any duplicates/equal value items (dragon genes tend to be unique!).

Tip: Your code's efficiency will ONLY be judged based on the **estimatedRobotTime**, so any code you write that doesn't call the **compare4** or **flip** operations essentially runs for "free", even if it isn't very CPU-efficient!

Grading: 120 points total. 100 for your code/algorithm, 20 points for the "write-up"/report.

Correctness: Always the 1st **priority** in new algorithm design. Algorithms that do not correctly sort all inputs will earn **at most 50 out of 100 pts.** Dracarys Bio Labs doesn't want to accidentally make the WRONG kind of dragon!

Performance-Based Grading

`estimatedRobotTime(2000) <= ?`	`expected code grade` *(out of 100)*
`1,030,000 seconds` (≈ 12 robot days for one sorting task)	`60`
`500,000 seconds` (≈ 6 robot days)	`65`
`420,000 seconds` (≈ 5 robot days)	`70`
`24,000 seconds` (≈ 6.7 robot hours)	`75`
`8,000 seconds` (≈ 2.2 robots hours)	`80`
`7,000 seconds` (≈ 2 robot hours)	`84`
`6,000 seconds` (≈ 1.7 robot hours)	`88`
`5,400 seconds` (≈ 1.5 robot hours)	`92`
`4,000 seconds` (≈ 1.1 robot hours)	`96`
`3,820 seconds` (≈ 1 robot hour)	`100`
`3,000 seconds` (≈ 50 robot minutes)	`100 + 15 bonus points`
`2,070 seconds` (≈ 34.5 mins), *better than Stonedahl's best?!*	`100 + 50 bonus points`

Speed Contest: Also, the 4 fastest student algorithms will earn bonus points, regardless of their absolute speed!

Prizes: **1st place earns 20 bonus points** *(and of course bragging rights...)*
2nd place → 15 bonus pts, 3rd place → 10 bonus pts, 4th place → 5 bonus pts

WARNING: If your code is primarily A.I. generated, I reserve the right to withhold BONUS points, because only HUMAN ingenuity deserves extra credit. (However, if your AI buddy designs a sorting algorithm faster than 3000 sec, you CAN turn that code in (with a comment/note that it was AI generated) and earn 100% on the code, just not be eligible for bonus points). Recall: if you use A.I. code WITHOUT citing/acknowledging it, you risk earning a 0 on the assignment and a possible honor code violation.

Getting Started Coding

Since these bio-robots are VERY expensive (and also don't exist), you will implement this task in simulation. :-)

Unzip the provided sample .java files, and copy them into a project in your favorite Java IDE.

1. Read through the code in **Main.java**. You are welcome to change this code,but you shouldn't need to modify it very much. Change it so that it's using your sorting algorithm, maybe try different manual tests for debugging, maybe change some numbers in order to collect specific data for your project "tech report".

You are NOT allowed to modify **BioArray.java**, which contains the simulated robot operations and performance counting code, but feel free to read it.

Definitely look at **BubbleSortBad.java** to see one example of a 4-op sorting algorithm.

Rename **YourFullNameSort.java** to actually include your First and Last Name in order.

After coding up a sorting algorithm, use the Main.java to test it for correctness, collect efficiency data, and measure its performance.

It's a good idea to create MORE different sorting algorithm classes and compare them with each other.

In the end, just turn in your best one, with **YourFullName.java** as the file / class name.

Coding Style Guidelines

Your code should be readable. (Ideally, aim for *more* readable than the code our textbook authors' provide, but it's fine if you want to use subtle tricks like **`a[i++]`** just like Sedgewick & Wayne did...)!)

These kind of algorithms can be tricky, so **comments are expected**, BUT comments should explain algorithm at a **higher level than each statement.** DO NOT merely repeat in English what each single line of Java already tells the reader. That's BAD commenting. Instead, explain WHY the code is the way it is, and what whole chunks of code are accomplishing.

Make sure you REMOVE any debugging **println** statements from your search algorithm codes before submitting it on Moodle.

***Warning:** Up to 5 points may be deducted for poor quality code (e.g. confusing variable names, no explanatory comments, comments that are FALSE/wrong)*

Write-up Task - Tech. Report Deliverable (20 pts)

Make sure your tech report includes these points A-F.

A) Which classic sorting algorithm (s) is your code most closely based on, and how did you adapt it to the parameters of this new BioSort problem?

B) Your algorithm's (well-commented) code, **formatted nicely for printing** – NOT a black background!

C) The code's theoretical Θ(…) average-case order of growth for both **flip** and **compare4** ops, and a discussion of how you arrived at those conclusions.

D) Two graphs, showing the (empirical) # of **flip** and **compare4** ops for increasing N (there's already some code in Main to help you collect this data).

   Using the most appropriate math function you can find (based on your theoretical analysis), add a "trend line" to your graph (Excel and Google Sheets both support this), and it should show what type of regression model/function your trend line is using to try to "fit" it with.

   Clearly label your x and y axes, and make sure your graphs easy to understand.

E) A brief discussion of whether your empirical data and your theoretical Θ(…) analysis agree.

F) Answer the question: Would your algorithm's performance scale well to allow the bio-robot to process very large N (millions? billions?), or is it only going to be efficient for fairly small N (like N=2000)?

Finishing Up

1. On Moodle, submit:
   - The **.java** file for your best sorting algorithm
   - A PDF file of your Tech Report.

Due date: 11:59 PM on Mon, Oct 27,

but you must also submit a PAPER COPY of the tech report in class on Wed, Oct 29.