CS146: Spring 2023
Homework 4: Sorting

Due Friday, March 24, at 11:59PM

75 points

This homework assignment consists of both programming and written components. Make sure to submit both your jar file and a pdf of your writeup. **To get started, import the starter files, ComparisonSorter.java, ComparisonSorterTest.java, Queue.java, RadixSorter.java, RadixSorterTest.java into the sorting package you create in a new Java Project.** You should not modify the Heap and Queue classes. Please do not change any of the method signatures in the ComparisonSorter and RadixSorter classes, but you are free to add additional helper methods. ComparisonSorterTest.java and RadixSorterTest.java are JUnit 4 test files. You need to add JUnit 4 to your build path, see the JUnit Setup section for instructions. If you are unfamiliar with JUnit tests, please see Lab 5 from CS46B that I added to the Canvas Resources.

Part 1: Sorting Comparisons (45 points)

For the first part of the assignment, you will compare different sorting algorithms. I have provided an implementation of InsertionSort and MergeSort. You will implement parts of HeapSort and QuickSort on future quizzes and I will provide a solution for those on Wednesday March 15 (MaxHeap) and Wednesday March 22 (QuickSort). For HeapSort, please use my implementation of a MaxHeap in the Heap.java file (it represents a solution to Quiz 6). For QuickSort, you can use either partition algorithm. I will provide a solution to the partition algorithm after class on March 22. *Note that all the sorting algorithms **work on empty arrays** and **on arrays of one element** as well as arrays of more than one element.*

HeapSort (15 points)

```
public static void heapSort(int[] arr)
```

Implement the heapSort algorithm as described in class and the textbook using the MaxHeap data structure provided for you Heap.java.

Testing (5 of the 15 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the heapSort method that test typical and edge cases. You get ½ point for creating the test case and ½ point for passing the test case. You are welcome to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

QuickSort (15 points)

`public static void quickSort(int[] arr)`

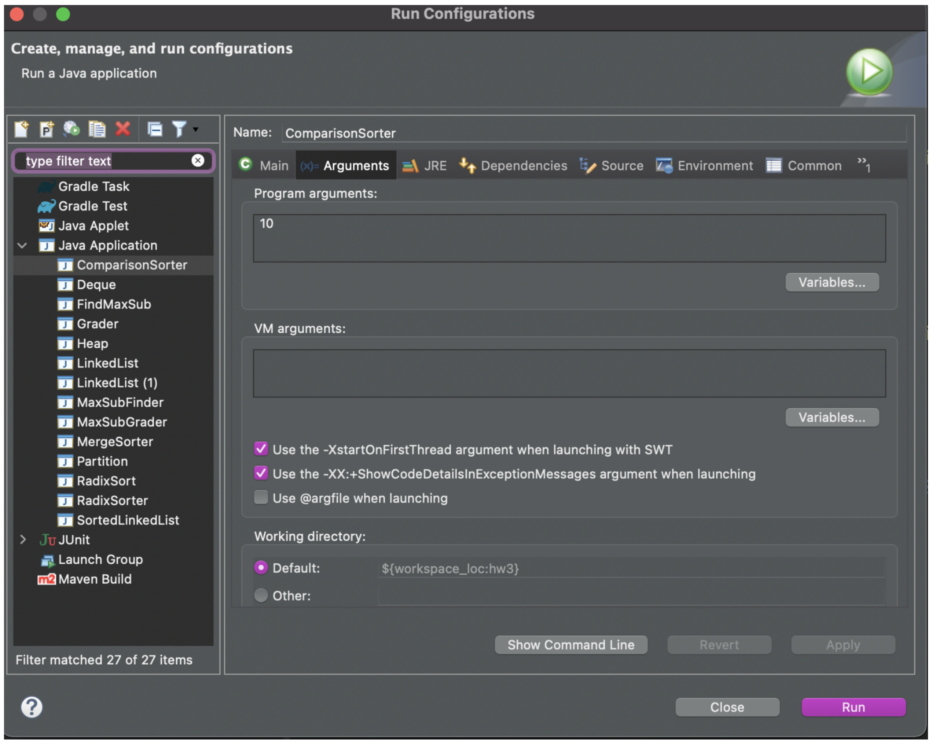
Implement the quickSort algorithm as described in class and the textbook. You are free to use whatever partition technique you choose.

Testing (5 of the 15 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the heapSort method that test typical and edge cases. You get ½ point for creating the test case and ½ point for passing the test case. You are welcome to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

Runtime Testing (15 points)

The method `public static void compare(int n)` creates arrays of integers of size n that contain random integers between -10000 and 10000. It runs each of the sorting algorithms on the arrays and prints the output time in nanoseconds. Your job is to run tests to determine how the different sorting algorithms compare in practice. The ComparisonSorter calls the compare method from its main method passing in the first command line argument for the value n in the compare function. To run classes with command line arguments in Eclipse, you will need to configure the run configuration for the class (see the image below)



Name: ComparisonSorter

Program arguments:

10

VM arguments:

☑ Use the -XstartOnFirstThread argument when launching with SWT

☑ Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching

☐ Use @argfile when launching

Working directory:

● Default: ${workspace_loc:hw3}

○ Other:

Written Prediction and Testing (5 points)

Predict the order in which the algorithms will perform in practice and explain your reasoning. You can predict that some algorithms will tie. Your reasoning should include the runtime of each algorithm.

Create a table that compares the runtimes of the different sorting algorithms on randomized inputs of sizes starting at 10 and increasing by a power of 10 as high as your computer can go. You don't need to run your code forever, but expect some of the larger sized inputs to take several minutes. For example it took my machines about 4 minutes to run the timing method on all four sorting algorithms with input size of 1,000,000.

Now implement similar methods to test special cases for sorting. You can use my compare method as a guide. Feel free to comment out the original compare method when you are running these methods. **Add these results to your table.**

```
public static void compareSorted(int n)
```

This method creates arrays of integers 1 to n in sorted order and runs the different sorting algorithms on the arrays.

```
public static void compareReverseSorted(int n)
```

This method creates an array of integers 1 to n in reverse sorted order and runs the different sorting algorithms on the arrays.

```
public static void compareDuplicates(int n)
```

This method creates an array of n 1s and runs the different sorting algorithms on the arrays.

Written Questions (10 points)

1) Now write down the relative order of the sorting algorithms.
    a) Did your predictions match reality? Why or why not? Think about how the hidden constant factors can influence the runtime.
    b) Did the size of the input affect the ordering?
2) How did each of the algorithms perform on the special cases? If the performance differed from the random case, explain why?
3) Explain how QuickSort behaved in each of the special cases. Why did you observe that behavior? If it performed poorly, how could your partitioning algorithm be improved to prevent the worst case runtime?

Part 2: Radix Sort using a Queue (30 points)

For the second part of the assignment you will program and analyze an alternative implementation of Radix Sort.

Programming Radix Sort (20 points)

You will implement Radix Sort on arrays of base 10 integers between 0 and 2,147,483,647 (Integer.MAX_VALUE). You will implement RadixSort in a new way using an array of Queues to sort the values by digit. Your algorithm must run in at most O(n) time. I have given you a simple implementation of a Queue (Queue.java) that inserts and removes elements in constant time.

Testing Radix Sort (5 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the mergeSort method that test typical and edge cases. You get ½ point for creating the test case and ½ point for passing the test case. You are welcome to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

Questions About Your Implementation (5 points)

Please write up your answers to the following questions.

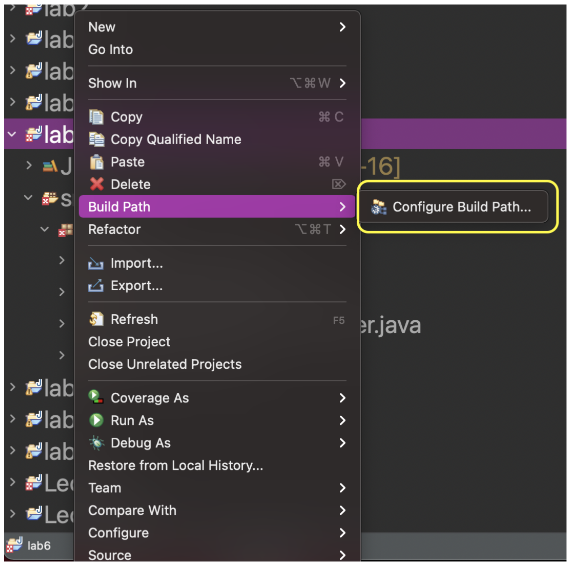
1. Why would I want to use arrays of Queues instead of an array of Stacks?
2. Justify that your program runs in at most O(n) time.

Submitting

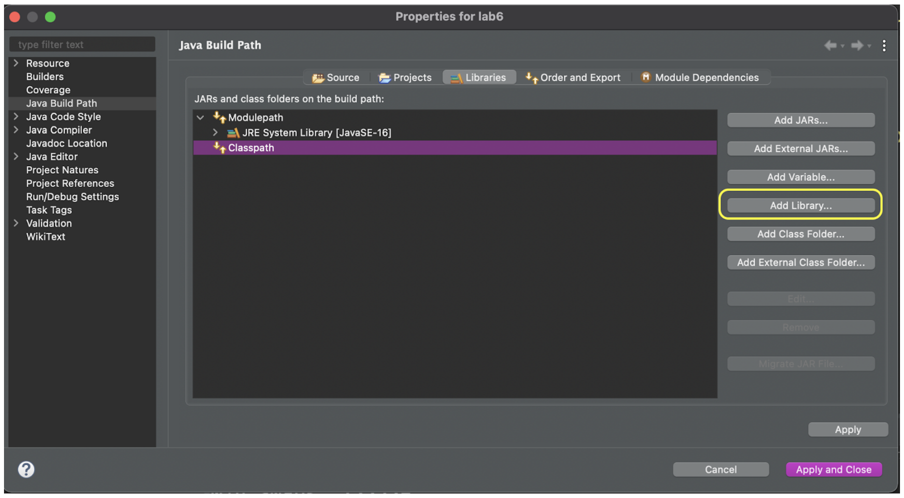
Export your project and upload it along with your solutions to the written questions. You can export your project as either a jar or a zip file. Make sure that the file you submit contains all your .java files in the package. If you upload a file that is missing any of the java source files you will get an automatic deduction of 10 points.

JUnit Setup

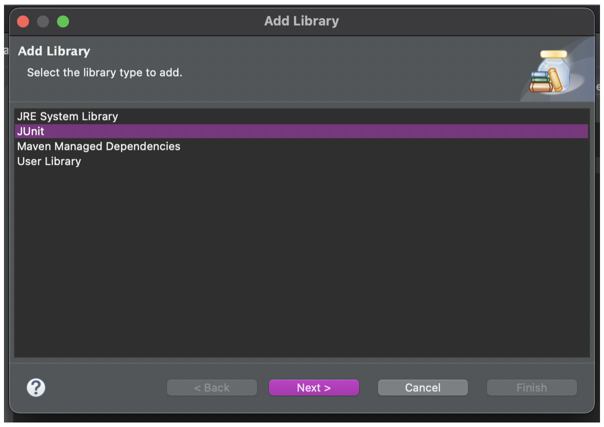
These instructions are for the Eclipse IDE. While you are welcome to use any IDE or programming environment, I will only provide instructions for Eclipse. To add the JUnit library to your build path, right-click on your homework project, look for Build Path and click on the Configure Build Path option.



Highlight Classpath and click Add Library

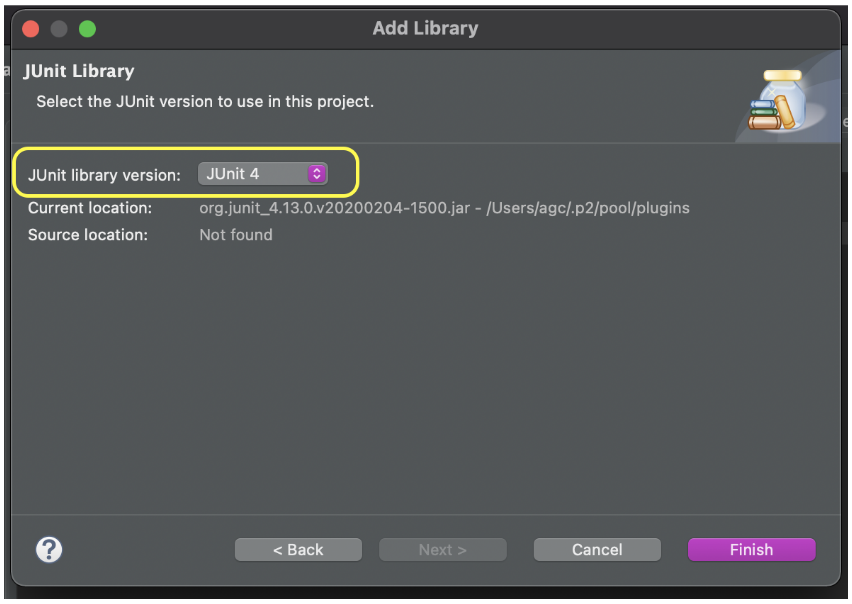


Select JUnit and click Next



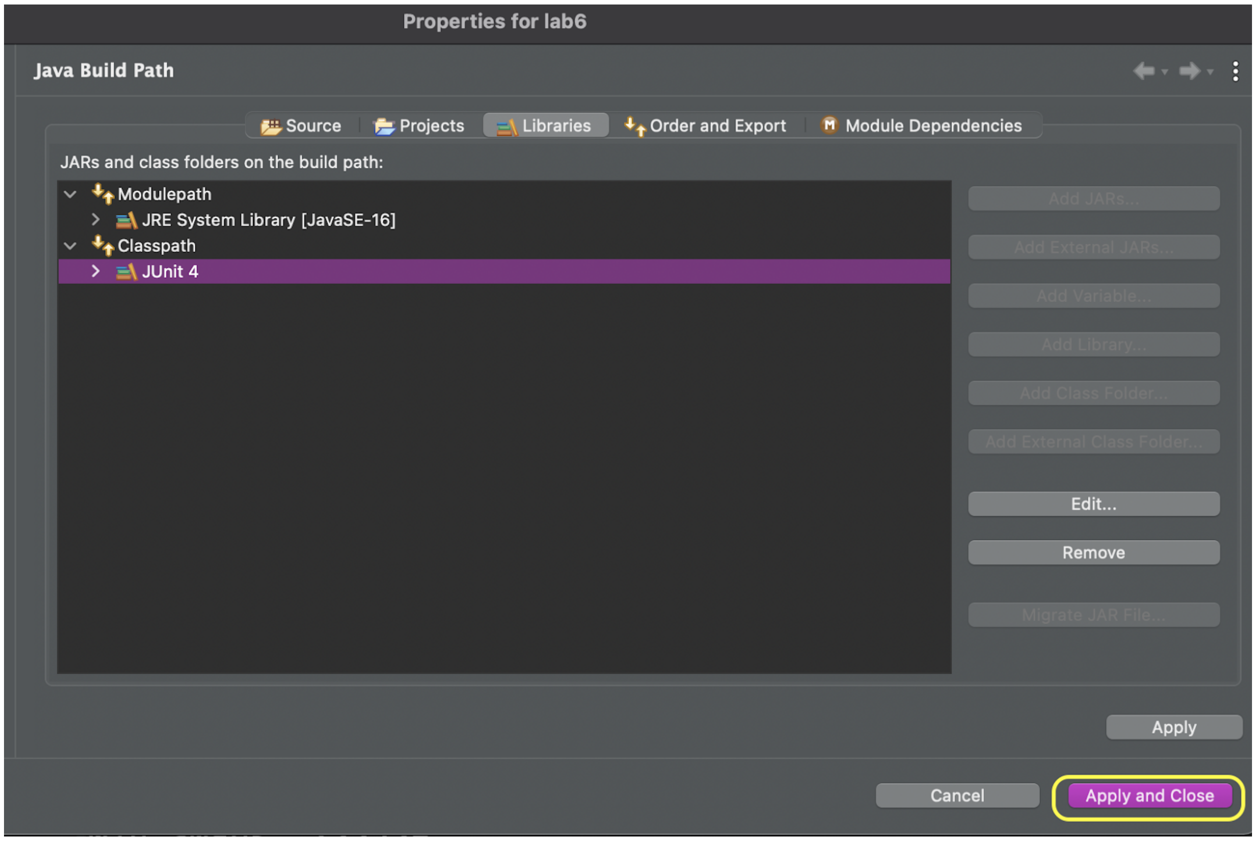
JRE System Library
JUnit
Maven Managed Dependencies
User Library

Make sure you select JUnit 4. Then click Finish.



JUnit library version: JUnit 4
Current location: org.junit_4.13.0.v20200204-1500.jar - /Users/agc/.p2/pool/plugins
Source location: Not found

Your build path should look something like this. Click Apply and Close.



JARs and class folders on the build path:

- Modulepath
  - JRE System Library [JavaSE-16]
- Classpath
  - JUnit 4

That should have gotten rid of any compilation errors you had when you originally copied over the files.