Module 10 Assignment: Iterative Sorts

Marcar esta página

Coding Assignment fecha límite Oct 11, 2023 02:43 -05

Iterative Sorts

For this assignment you will be coding 3 different iterative sorts: **bubble sort**, **insertion sort**, and **selection sort**. In addition to the requirements for each sort, the autograder will be looking at the number of comparisons made between elements to test for efficiency.

For each sorting algorithm, you may assume that the array you are sorting will not contain null elements. You should also assume that the array may contain any number of duplicate elements.

Your implementations should match what was taught in this course.

IMPORTANT:

- **You will be given 5 attempts on this assignment, with a 30 minute cooldown between submissions.**
- **Please run your code before each submission to ensure that there are no formatting errors! If there are formatting errors in your code, your code will not be graded and a submission attempt will be logged. For more information, please review the Vocareum overview below.**

Comparator

Each method will take in a Comparator, which is used to compare two elements. You **must** use this Comparator, as the number of comparisons performed with it will be used when testing your assignment. The comparator is used as follows:

`comparator.compare(A, B)` will return:

- < 0 if A is less than B
- > 0 if A is greater than B
- 0 if A is equal to B

Generic Methods

Most of the assignments is this course so far have utilized generics by incorporating them into the class declaration. However, the rest of the assignments will have you implement various algorithms as static methods in a utility class. Thus, the generics from here on will use generic methods instead of generic classes (hence the **<T>** in each of the method headers and javadocs). This also means any helper methods you create will also need to be static with the same **<T>** in the method header.

In-Place Sorts

The three sorting algorithms that you will be implementing are in-place sorts. This means that the items in the

array passed in **should not** get copied over to another data structure. Note that you can still create variables that hold only one item. However, you should not create another data structure, such as an array or list, in the method.

Stable Sorts

Some of the sorts below are stable sorts. This means that duplicates **must** remain in the same relative positions after sorting as they were before sorting.

Adaptive Sorts

Some of the sorts below are adaptive sorts. This means that the algorithm takes advantage of existing order in the input array. The algorithm can detect existing order in the input array and optimize its performance based on that order.

Bubble Sort

Bubble sort should be in-place, stable, and adaptive. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n)$. **Note: Implement bubble sort with the optimization where it utilizes the last swapped index.** Remembering where you last swapped will enable some optimization for bubble sort. For example, traversing the array from smaller indices to larger indices, if you remember the index of your last swap, you know after that index, there are only the largest elements in order. Therefore, on the next iteration, you start at the front but stop at the last swapped index. Make sure you only look at the indices that you do not know are sorted. Do not make extra comparisons.

Exam of two iterations of bubble sort with last swapped optimization:
Start of bubble sort:

Start iteration 1:
Compare 1 (at index 0) with 2 (at index 1) and don't swap

Compare 2 (at index 1) with 6 (at index 2) and don't swap

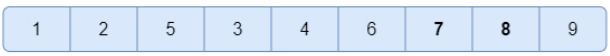
Compare 6 (at index 2) with 5 (at index 3) and swap

Compare 6 (at index 3) with 3 (at index 4) and swap

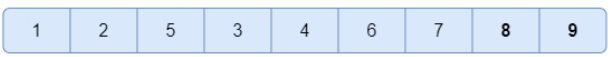
Compare 6 (at index 4) with 4 (at index 5) and swap

Compare 6 (at index 5) with 7 (at index 6) and don't swap

Compare 7 (at index 6) with 8 (at index 7) and don't swap



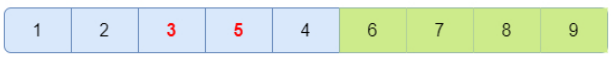
Compare 8 (at index 7) with 9 (at index 8) and don't swap



Start iteration 2:
Note: Skip over indices 5-8 since no swaps occurred there.
Compare 1 (at index 0) with 2 (at index 1) and don't swap

Compare 2 (at index 1) with 5 (at index 2) and don't swap

Compare 5 (at index 2) with 3 (at index 3) and swap



Compare 5 (at index 3) with 4 (at index 4) and swap

Finished iteration 2 of bubble sort.
Note: Next iteration, skip over indices 4 - 8 since no swaps occurred there.

Insertion Sort

Insertion sort should be in-place, stable, and adaptive. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n)$.

Note that, for this implementation, you should sort from the beginning of the array. This means that after the first pass, indices 0 and 1 should be relatively sorted. After the second pass, indices 0-2 should be relatively sorted. After the third pass, indices 0-3 should be relatively sorted, and so on.

Selection Sort

Selection sort should be in-place, unstable, and adaptive. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n^2)$. You can implement either the minimum version or the maximum version; both are acceptable since they will both yield the same number of comparisons.

General Tips

- If your comparison count is slightly over the required amount, double check your for-loop bounds to ensure that you are not unnecessarily comparing elements.
- We highly recommend copying the starter code and working in your preferred IDE in order to have access to features such as code completion, auto-formatting, and much more!

Here are general assignment guidelines that should be followed.

- Do not include any package declarations in your classes.
- Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add throws to the method headers since they are not necessary. Instead, exceptions should be thrown as follows: `throw new InsertExceptionHere("Error: some exception was thrown");`
- All helper methods you choose to write should be made private. Recall the idea of Encapsulation in Object-Oriented Programming!
- Do not use anything that would trivialize the assignment. (e.g. Don't import/use java.util.ArrayList for an ArrayList assignment.)
- Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
- If applicable, use the generic type of the class; do not use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`.

Use of the following statements should be avoided at all times.

package	System.arraycopy()	clone()
assert()	Arrays class	Array class
Thread class	Collections class	Collection.toArray()
Reflection APIs	Inner or nested classes	Lambda Expressions

The Vocareum (code editor) interface has six main components:

- The **Drop-Down** in the top left. This lets you choose from multiple available files. Note that this drop-down will only be visible in assignments that require multiple files.
- The **Run** button. This will compile your code and run a file scan. Running your code will not count towards your total allowed submission attempts, therefore you are free to run as many times as needed.
- The **Submit** button. This will compile your code, run a file scan, grade your assignment, and output results to console. Note that for most assignments in this class, you will only be allowed a limited number of submissions. A submission is counted when the submit button is clicked, regardless of whether or not your code can compile or if there are any file issues. Therefore, we **highly recommend** that you run your code before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. For large coding assignments, we highly recommend

copying the starter code and working in your preferred IDE to have access to features such as code completion, auto-formatting, and much more!

- The **Output Window**. This window will appear whenever you run or submit your code and will display the output for you to view.

For additional help, please visit the Vocareum information page located in the course information module!

< Anterior | Siguiente >