Sorting and Searching

Lab setup

1. Create a new Java project called JavaSortingSearching.
2. Download the provided Java files, extract them, and place them in the edu.odu.cs.cs251 package in the src directory.
   Each file should be in the directory src/edu/odu/cs/cs251
3. Download the provided data files, extract them, and place them inside of your project.
4. Add two classes: SortingUtils and SearchingUtils
5. Add two classes: SortingLab and SearchingLab
6. Add a main class

Data description

The sorting portion of this lab will use simulated image data. Images are described by their path/file name, a set of keywords, the date the image was created, and its file size.

The image data is given as serialized lists with numerous sizes: 10, 100, 1000, 10000, 100000, 1000000, 10000000.

Sort algorithms

There are many algorithms for sorting. We will focus on bubble sort, insertion sort, selection sort, and merge sort. We will consider lists of integers for our examples, and modify the sorting methods we create for

Bubble sort

The intuition behind the bubble sort algorithm is that we can sort a list by “bubbling” larger elements to the top. Each element E is compared to the other elements in the list. If E is greater than an element it is compared to, then the positions of the two elements’ in the list are swapped. This procedure is repeated for every element in the list. Try to translate this procedure into a static method inside SortingUtils.

Insertion sort

The intuition behind the insertion sort algorithm is sorting cards in a hand. As an example, imagine you have a hand of cards [5, 9, 6, 11, 8]. Also imagine that you’re very particular about cards overlapping.

1. Start with [5] considered to be sorted.

2. Insert 9
    a. Compare 9 and 5
    b. Because 9 is greater than 5, it is inserted after 5
    c. Result: [5, 9]
3. Insert 6
    a. Compare 6 and 5, 6 is greater so we move to 9
    b. Compare 6 and 9
    c. Because 9 is greater 6 should be inserted before it
    d. Shift 9 to the right, and insert 6 at its old position
    e. Result: [5, 6, 9]
4. Insert 11
    a. Quickly, 11 is greater than 5, 6, and 9
    b. 11 stays in its position
    c. Result: [5, 6, 9, 11]
5. Insert 8
    a. 8 is greater than 5 and 6
    b. 8 is less than 9, so it must be inserted at its position
    c. Shift 9 and 11 to the right
    d. Result: [5, 6, 8, 9, 11]

Try to translate the procedure described in this example into a static method inside of SortingUtils.

<u>Selection sort</u>

The intuition behind the selection sort algorithm is to split the array into two sub lists, sorted and unsorted. Every iteration, we select the next element in order and add it from the unsorted array to the sorted array. We will use the same list from the insertion sort example: [5, 9, 6, 11, 8].

1. Find the smallest element and swap it with the first element in the list
    a. Our first element happens to be the smallest element already.
    b. Now we have the sorted list: [5] and the unsorted list [9,6,11,8]
    c. Sorted Range: (0) and Unsorted Range: (1,4)
2. Find the smallest element in the unsorted list, and append it to the end of the sorted list
    a. Sorted: [5, 6], Unsorted: [9, 11, 8]
3. Find the smallest element in the unsorted list, and append it to the end of the sorted list
    a. Sorted: [5, 6, 8], Unsorted: [9, 11]
4. Find the smallest element in the unsorted list, and append it to the end of the sorted list
    a. Sorted: [5, 6, 8, 9], Unsorted: [11]

5. Unsorted list only has one element, append it to the sorted list.
    a. Sorted: [5, 6, 8, 9, 11]

Try to translate the procedure described in the example above into a static method inside of SortingUtils.

<u>Merge sort</u>

The intuition behind the merge sort algorithm is divide-and-conquer. We can recursively split the lists into smaller sub lists, sort them, and merge them back into a sorted list. At every step, we split the list in half and recursively split each half again until we have lists of size 1.

The merge procedure takes two sub lists and merges them together in sorted order. This is done by iterating through both lists at the same time and determining which element should be inserted first to have sorted list

```
Left : [3,6,12], Right : [8,11,52]

List: [_,_,_,_,_,_]

lIdx = 0, rIdx = 0

listIdx = 0
```

First iteration:

- Is 3 less than 8?
    - Yes, insert left into List[listIdx]
    - Increment lIdx and listIdx

Second iteration:

- Is 6 less than 8?
    - Yes, insert 6 into List[listIdx]
    - Increment lIdx and listIdx

Third iteration:

- Is 12 less than 8?
    - No, insert 8 into List[listIdx]
    - Increment rIdx and listIdx

Fourth iteration:

- Is 12 less than 11?
    - No, insert 11 into List[listIdx]
    - Increment rIdx and listIdx

Fifth iteration:

- Is 12 less than 52?
    - Yes, insert 12 into List[listIdx]
    - Increment lIdx and listIdx

lIdx is now equal to Left.size() so we stop iteration and insert the leftovers in Right to fill out List.

Try to implement at least the merge procedure as a method in SortingUtils.

Task 1

We will use the data and sort them using the different sorting algorithms. We will work inside of SortingLab and make small changes to the methods in SortingUtils.

1. Deserialize one of the given lists (The 10000000 list will probably take a while to sort using bubble sort)
2. For each of the sorting algorithms
    a. Make a copy of the original list (the copy will need to be modifiable)
    b. Print the first 5 elements in the list
    c. Invoke the sorting method on the copy of the list
    d. Print the first 5 elements in the list

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Search algorithms

Data description

For the searching portion of this lab we will use randomized student data. A student is described by its name, age, and gpa.

The data is given as serialized lists with numerous sizes: 10, 100, 1000, 10000, 100000, 1000000, 10000000. Each set is guaranteed to have a student named "qin".

Sequential Search

Sequential search is a method of searching that looks at each object in a list, in order, to find a target value. If you ever fingered through a phone book, you've done a sequential search. If you ever scanned a room for your friend, you've probably done a sequential search.

In Java, we can perform a sequential search by looping over each element in our student list to check if that student is "qin".

Binary Search

Binary search is a method of searching that takes advantage of a precondition to optimize searching. That precondition is that the list is sorted by the same criteria that we are searching. For example, if we want to use binary search to search for the student named "Qin" then the list must be sorted by name.

Binary search splits the sorted list down in the middle into two sub-lists (left and right), excluding the middle element. If the middle element is not what we are searching for, then we can compare what we are searching for to the middle element. If what we are searching for is less-than the middle element, then it would be in the left sub-list. If it is greater, then it would be in the right sub-list. We repeat this procedure on the chosen sub-list

Try to translate the procedure described above into a method in SearchingUtils. You will need to consider how to represent splitting a list into sublists.

Task 2

We will search for "qin" using the two search methods. We will work in the SearchingLab class.

1. Read the student data into a list
2. Perform a sequential search for a student with the name "qin"
3. Perform a binary search for a student with the name "qin"
   a. Make a copy of the original list and sort it
   b. Invoke the binary search method you wrote in SearchingUtils.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder "LabSortingSearching_firstname_lastname", where first name and last name are
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (points allotted) - Implementations of the sorting algorithms (5) - Correct output of sorting algorithms (5)	Task 2 (points allotted) - Implementations of the searching algorithms (5) - Correct output of the searching algorithms (5)