CS 211, ALL SECTIONS

LAB EXERCISE-14

DUE MONDAY, APRIL 29TH AT MIDNIGHT

The objective of this lab is to practice working with sorting and searching algorthims.

OVERVIEW:

1. Write the class `SortSearch` which is a utility class that contains only static methods. The class will contain three static methods, each used for a specialized methodology related to sorting and searching.
2. Download and use a tester module to ensure that your program is correct.
3. Prepare the assignment for submission and submit it.

NOTES:

- This is a lab exercise on which collaboration (including discussing the solution on Piazza or searching online sources) is explicitly permitted. It is still in your interest to understand and prepare your final submission yourself.
- You are not forbidden to use imports, but you will most likely not need them.
- There is no restriction on which sort or search algorithm(s) you employ in your solution.
- If your method modifies the input array as a side effect rather than as an indended effect, it is good form to make a copy of the array instead (this will not be checked, but is a good idea to do anyways).
- You may include your own helper methods (both public or private).

BACKGROUND:

Most of us are familiar with what sorting and searching means: the former refers to locating a specific element in a list or array, while the latter refers to putting a list or collection into order. In class, we will have discussed various algorithms for sorting and for searching. Knowing these approaches, it is natural to adapt them to specific problems. For example, when it comes to searching, we may not want to sort in the natural order (i.e. increasing element values), but some variation instead (reverse order, or distance from a specific value, or something more complicated). When it comes to searching, maybe we do not want to find the exact match of a key, but instead find an element which matches a broader condition. In this lab, we will be implementing several variants on the basic idea of searching and sorting.

FINDFIRSTDIST TASK:

```
public static int findFirstDist(int[] array, int v, int d)
```

(*3pts*) This is a search method which finds the index of the first element in `array` which lies within a distance `d` of `v`. Thus, for example, if `d=3` and `v=2` then this method would return the index of the first array element which is either a -1, 0, 1, 2, 3, 4, or 5. If no such element is found, then -1 is returned instead.

SORTDIST TASK:

```
public static void sortDist(int[] array, int v)
```

(*3pts*) Here we implement a sorting task which will take the input `array` and place the elements in sorted order. However, sorted order will not mean smallest to largest but from the closest to value `v` to the farthest from value `v`. Ties are broken using the order from the original array.

FINDKTHFROMV TASK:

```
public static int findKthFromV(int[] array, int k, int v)
```

(*3pts*) The goal of this method is to find the `k`th closest element to a given value `v`. If `k=1` it would return the value of the array element which is closest in value to `v`. If `k=2` it would find the second closest, etc. This method is closesly related to sorting: if we knew what order the elements came in, then would we be able to use that to decude the `k`th element? As with the sort method above, ties are broken based on the order they show up in the array.

Tip 1: this method returns the value of the element found, not its index.
Tip 2: be aware that an array's index begins at zero, but the closest element is the 1st element.

EXAMPLE:

```
> int[] array = {10, 22, 13, 4, 8}
> SortSearch.findFirstDist(array,15,3) // index of the first element within a distance of 3 of 15
2
> SortSearch.sortDist(array,15)
> array
```

```
{ 13, 10, 22, 8, 4 }
> SortSearch.findKthFromV(array,2,19) // second-closest element to 19
13
```

TESTING:

Download the following:

- https://mason.gmu.edu/~iavramo2/classes/cs211/junit-cs211.jar (if necessary)
- https://mason.gmu.edu/~iavramo2/classes/cs211/s19/E14tester.java

This is a unit tester which is what we will use to test to see if your classes are working correctly.

When you think your classes are ready (you can try them even without the tester), do the following from the command prompt to run the tests.
On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar E14tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar E14tester
```

In Dr Java:

- open the files, including `E14tester.java`, and click the *Test* button.

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number and let *yyyyyyyy* be your GMU userid. Create the directory `xxx_yyyyyyyy_E14/`
2. Place the `.java` file that you've created into the directory.
3. Create the file `ID.txt` in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.

   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213

4. compress the folder and its contents into a .zip file, and upload the file to Blackboard.