EXERCISE 1: A Comparable Point2D

Download `precept3.zip` from the precepts page, unzip the project and open it using IntelliJ.

(a) The goal of this part is to define a default comparison behavior for the Point2D class and use it in a simple test client. Proceed according to the following steps:

- Modify the class declaration of `Point2D` to make it implement the `Comparable` interface.
- Implement the `compareTo` method. This method allows the point to be compared to another given point (passed as an argument to the method).
  - Use the `y-coordinate` for comparison and break ties using the `x-coordinate`.
  - Return 1 if the point is greater than the method argument, -1 if it is less and 0 otherwise.
- Complete the given test program to perform the following:
  - Sort the array according to the default order defined in the `compareTo` method.
  - Check (in linear time in the worst case) if all the points in the array are distinct.

(b) The goal of this part is to define an alternate comparison behavior for 2D points and use it in a simple test client. Proceed according to the following steps:

- Uncomment the code marked as `/* *** PART (B) *** */`.
- Complete the implementation of class `DistanceToOrder` such that it allows comparing between two given points based on their distance to a given reference point.
  - Make the class implement the `Comparator` interface.
  - Implement the constructor to receive and store the reference point.
  - Implement the `compareTo` method. This method compares the two given argument points: Returns 1 if the first argument is farther from the reference point than the second argument, -1 if it is closer and 0 otherwise.
- Complete the given test program to perform the following:
  - Sort the array according to the distance of the points from the origin (0, 0).
  - Check (in linear time in the worst case) that no two points have the same distance from the origin (0, 0).
EXERCISE 2: Algorithm Design (Midterm Spring 2015)

Let \( a = a_0, a_1, ..., a_{n-1} \) be an array of length \( n \). An array \( b \) is a circular shift of \( a \) if it consists of the subarray \( a_k, a_{k+1}, ..., a_{n-1} \) followed by the subarray \( a_0, a_1, ..., a_{k-1} \) for some integer \( k \). In the example below, \( b \) is a circular shift of \( a \) (with \( k = 7 \) and \( n = 10 \)).

<table>
<thead>
<tr>
<th>sorted array ( a[] )</th>
<th>circular shift ( b[] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 5 6 8 9</td>
<td>34 55 89 1 2 3 5 6 8 9</td>
</tr>
</tbody>
</table>

Suppose that you are given an array \( b \) that is a circular shift of some sorted array (but you have access to neither \( k \) nor the sorted array). Assume that the array \( b \) consists of \( n \) comparable keys, no two of which are equal. Design an efficient algorithm to determine whether a given key appears in the array \( b \). The order of growth of the running time of your algorithm should be \( \lg n \) (or better) in the worst case, where \( n \) is the length of the array.
ASSIGNMENT TIPS: Autocomplete

(1) Given an array of elements with duplicates, can we use the book implementation of Binary Search to find the first occurrence of an element?

- The standard implementation of Binary Search finds an occurrence, which is not necessarily the first occurrence.
- Finding the element and then scanning left to find the first occurrence yields a linear running time (in the worst case), which is not good!
- In this assignment, you will have to modify Binary Search to find the first (and last) occurrence of an element in a sorted array in logarithmic time (in the worst case).
- For full credit, your algorithm has to make at most \( 1 + \lceil \log_2 n \rceil \) compares. However, if your algorithm has a logarithmic order of growth but makes more than \( 1 + \lceil \log_2 n \rceil \) compares, you will lose only 1 point.

(2) What is the difference between a Comparable and a Comparator?

- A Comparable\(<T>\) is an object of a class that has the method \( \text{compareTo}(T \ other) \). This method allows the object to compare itself to other objects.
- A Comparator\(<T>\) is an object that can be used to compare two given objects. It has the method \( \text{compare}(T \ obj1, \ T \ obj2) \).
- Making an object Comparable makes it comparable with other objects using the logic provided in the \( \text{compareTo} \) method. However, if we want to implement multiple ways of comparison (for e.g. compare files by name, date created, date modified, etc.), then we need to have multiple Comparators.
- A good example of the use of Comparable and Comparator is Point2D. java, which is available at: https://algs4.cs.princeton.edu/code/. You can use this as a guide when working on the assignment.
- Note that a Comparator class can have a constructor that takes arguments. This may be needed in the assignment!

(3) What is the order of growth of the substring method?

- Creating a substring of length \( r \) takes time proportional to \( r \).
- Note that the string comparison functions in the assignment should take time proportional to the number of characters needed to resolve the comparison.
- Example: The comparison between \( X=“AAAAAAA” \) and \( Y=“AABBB” \) can be resolved when the first “B” in Y is reached. The comparison function should not take time proportional to the size of X or the size of Y. It should take time proportional to the number of characters needed to resolve the comparison!
- Most uses of the substring method in the compare functions do not meet the above time constraint. So, be careful!

(4) A video that provides some tips for the assignment is available on the assignment Checklist page. The video was made in 2014, so a few things are outdated, but most of it still useful!