Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop Week 7

Learning outcomes

By the end of this session, you will have learnt about dealing with arrays of objects, and more specifically sorting them.

Import project from `workshop07template.zip`.

1. **Sorting trace**

   What is the status of the array `arr` at the end of each *iteration* while sorting it in **descending** order using SELECTION sort. The original array `arr` is $\{6,1,3,8,2,5,9\}$

2. **Debugging sorting algorithm**

   The following implementation to sort an array of integers in ascending order using selection sort has some bugs. Identify the bugs and fix the code. HINT: It's got to do with the way values are passed to functions, more specifically, the contrast between passing variables vs. passing an array.

```
1  public static void selectionSort(int[] arr) {
2          if(arr == null)
3                  return; //nothing to do
4
5          for(int i=0; i < arr.length - 1; i++) {
6                  int minIndex = smallestItemIndex(arr, i);
7                  swap(a[i], a[minIndex]);
8          }
9  }
10
11 /**
12 returns the index of the smallest item in the array,
13 starting at index startIndex
14 */
15 public static int smallestItemIndex(int[] a, int startIndex) {
16         if(a == null || startIndex < 0 || startIndex >= a.length)
17                 return -1;
18         int result = startIndex;
19         for(int i = startIndex + 1; i < a.length; i++)
20                 if(a[i] < a[result])
21                         result = i;
22         return result;
23 }
24
25 public static void swap(int a, int b) {
26         int temp = a;
27         a = b;
28         b = temp;
29 }
```

3. **Sorting trace**

   What is the status of the array `arr` at the end of each *iteration* while sorting it in **descending** order using INSERTION sort. The original array `arr` is $\{6,1,3,8,2,5,9\}$

4. **Fraction class**

   Go through the class `Fraction` to understand its purpose. Instantiate a `Fraction` object to represent the fraction $\frac{12}{18}$. Draw a memory diagram to illustrate what happens in the memory when this object is instantiated. Reduce the fraction to its simplest form ($\frac{2}{3}$).

5. **`compareTo(Fraction)` method**

   Complete the method `compareTo` in class `Fraction`. The requirements for this method have been provided as javadoc comments above the method header.

6. **`bubbleSort`**

   (a) Trace the execution of the method `bubbleSort(Fraction[])` in class `FractionArrayService` for the `myFractions` created as,

```
int[] nums = {5, 10, 2, 100, 6};
int[] dens = {2, 20, 5, 100, 5};
myFractions = new Fraction[5];
for(int i=0; i<myFractions.length; i++) {
        myFractions[i] = new Fraction(nums[i], dens[i]);
}
```

   (b) **(Assessed task)** What are the best case and worst case time complexities of `bubbleSort`?

7. **Correct the method `buggySort`**

   The method `buggySort(Fraction[])` in class `FractionArrayService` has some bugs. The corrected version is the method `bubbleSort(Fraction[])` in class `FractionArrayService`. Compare the two to identify the bug and highlight it using a memory diagram.

8. **(Assessed task)**

   Complete the method `sortNumDigits` that sorts an array of integers in the order of number of digits. You may, and probably should, add a helper method. For example, if the array before sorting is $\{54,1,45,834,91,540\}$, after sorting it should be $\{1,54,45,91,834,540\}$.

```
public static void sortNumDigits(int[] a) {
        if(a == null)
                return;
        for(int i=1; i < a.length; i++) {
                int backup = a[i];
                int k = i - 1;
                while(k >= 0 && ___________________) {
                        a[k+1] = a[k];
                        k--;
                }
                a[k+1] = backup;
        }
}
```

9. **(Voluntary Assessed task) Complete the method `secondarySort`**

The method `secondarySort` in class `FractionArrayService`, is a slight variation of insertion sort, and sorts a `Fraction` array by putting each item of array at the rightful place *so far* in a secondary array. The original array should then be manipulated so that each item of the old array refers to the object the corresponding item of the secondary array refers to. This step has been left incomplete in the template and your job is to complete this part. You are trying to go through each item of the passed array (the array that needs to be sorted) (using index *i*) and updating its reference to the item at the same index in the secondary array.