Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop - Sorting

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 `workshop_sorting_template.zip`.

1. **Selection sort trace**

   (a) What is the status of the array `arr` at the end of each *iteration* while sorting it in **descending** order using selection sort (when sorting from left to right). The original array `arr` is $\{6, 1, 3, 8, 2, 5, 9\}$

   (b) What is the status of the array `arr` at the end of each *iteration* while sorting it in **descending** order using selection sort (when sorting from **right** to **left**). The original array `arr` is $\{6, 1, 3, 8, 2, 5, 9\}$

   (c) What is the status of the array `arr` at the end of each *iteration* while sorting it in **ascending** order of **number of digits** using selection sort (when sorting from **left** to **right**). The original array `arr` is $\{125, 10, 9, 20, 300, 2\}$

2. **Insertion sort trace**

   Write an answer to question 1, but for insertion sort instead of selection sort.

3. **Importance of deep learning - debugging sorting algorithm**

   The method `selectionSortBuggy` in class `Buggy.java` contains some bugs (duh!). A client is provided in `BuggySortClient.java` that calls the said method but the array remains unsorted. Identify the bug in `selectionSortBuggy` with the help of drawing a memory diagram and logic table, and fix it. **TIP: Note all memory transactions.**

4. 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 $\{1, 54, 45, 91, 834, 540\}$.

```java
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;
        }
}
```

5. Complete the method `sortNumberOfDivisors` that sorts an array of integers in the order of number of non-trivial divisors (divisors other than 1 and the number itself). For example, 15 has 2 non-trivial divisors - 3 and 5. You may, and probably should, add a helper method. For example, if the array before sorting is $\{24, 1, 65, 31, 25\}$, after sorting it should be $\{1, 31, 25, 65, 24\}$.

```java
public static void sortNumberOfDivisors(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;
        }
}
```

6. **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 ($\frac{2}{3}$).

7. **`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.

8. **Additional 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.

9. **Additional task (ADVANCED): Graph validity**

In graph theory, it is possible to draw a (simple) graph only if,

a. if the sum of edges is even.
b. none of the vertices has a degree greater than or equal to the number of vertices.
c. the graph remains valid after removing all (say *k*) edges of the *most connected* vertex (and one from other *k* vertices each). While repeating this procedure, an unconnected vertex can be ignored. Only if we reach an empty graph by repeatedly applying this step, is it a valid graph. If we reach a scenario where the most connected vertex has degree *k* and there aren't *k* other vertices with non-zero degree left, the graph is invalid.

For example, if the degree (number of edges with the vertex as an end-point) set for the vertices of a graph is {3, 3, 3, 3, 3}, we can see that the sum of degrees is 15 (not even), therefore it's not a valid graph.

If the degree (number of edges with the vertex as an end-point) set for the vertices of a graph is {3, 3, 10, 4, 2}, the third vertex's degree (10) is invalid (max degree for a graph with 5 vertices is 4), therefore it's not a valid graph.

If the degree (number of edges with the vertex as an end-point) set for the vertices of a graph is {3, 3, 2, 4, 2}, the first two aspects are ok (no violation). The sorted set is {4,3,3,2,2}, and after removing the 4 edge connected to the first vertex, the set becomes {2,2,1,1} (first vertex is disconnected and can be ignored. Repeating the process, the set becomes {1,1,0,0}, which is {1,1}, which becomes {0,0}, which is an empty graph. Hence the original graph is valid.

Similarly, if the set is {4,3,2,2,1}

$$\begin{aligned} &\{4,3,2,2,1\} \\ reduced &: \{0,2,1,1,0\} \\ sorted &: \{2,1,1,0,0\} \\ reduced &: \{0,0,0,0,0\} \\ &valid \quad (empty \quad graph \quad reached) \end{aligned}$$

Another example - if the set is {2,5,2,4,3,4},

$$\begin{aligned} &\{2,5,2,4,3,4\} \\ sorted &: \{5,4,4,3,2,2\} \\ reduced &: \{0,3,3,2,1,1\} \\ sorted &: \{3,3,2,1,1,0\} \\ reduced &: \{0,2,1,0,1,0\} \\ sorted &: \{2,1,1,0,0,0\} \\ reduced &: \{0,0,0,0,0,0\} \\ &valid \end{aligned}$$

Last example - if the set is {3,3,3,3},

Similarly, if the set is {2,5,2,5,3,4},

$$\begin{aligned} &\{5,5,4,3,2,2\} \\ reduced &: \{0,4,3,2,1,1\} \\ &invalid \quad (odd \quad sum \quad of \quad degrees) \end{aligned}$$

Write a method that when passed an array of degrees of the vertices of a graph, returns `true` if it's a valid graph, and `false` otherwise.