MACQUARIE University

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 $\{6,1,3,8,2,5,9\}$

   Solution:

   ```
   6 1 3 8 2 5 9
   9 1 3 8 2 5 6
   9 8 3 1 2 5 6
   9 8 6 1 2 5 3
   9 8 6 5 2 1 3
   9 8 6 5 3 1 2
   9 8 6 5 3 2 1
   ```

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.

```
public static void selectionSort(int[] arr) {
        if(arr == null)
                return; //nothing to do

        for(int i=0; i < arr.length - 1; i++) {
                int minIndex = smallestItemIndex(arr, i);
                swap(a[i], a[minIndex]);
        }
}

/**
returns the index of the smallest item in the array,
starting at index startIndex
*/
public static int smallestItemIndex(int[] a, int startIndex) {
```

```java
        if(a == null || startIndex < 0 || startIndex >= a.length)
                return -1;
        int result = startIndex;
        for(int i = startIndex + 1; i < a.length; i++)
                if(a[i] < a[result])
                        result = i;
        return result;
}

public static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
}
```

Solution: The problem is with the swap method. In its current version, you are passing two variables, and the swap is done on a copy of those, not the actual parameters (that are passed). Thus a[i] and a[minIndex] are not actually swapped. We need to pass the array and the indices of the two items that must be swapped.

```java
public static void selectionSort(int[] arr) {
        if(arr == null)
                return; //nothing to do

        for(int i=0; i < arr.length - 1; i++) {
                int minIndex = smallestItemIndex(arr, i);
                swap(a, i, minIndex);
        }
}

/**
returns the index of the smallest item in the array,
starting at index startIndex
*/
public static int smallestItemIndex(int[] a, int startIndex) {
        if(a == null || startIndex < 0 || startIndex >= a.length)
                return -1;
        int result = startIndex;
        for(int i = startIndex + 1; i < a.length; i++)
                if(a[i] < a[result])
                        result = i;
        return result;
}

public static void swap(int[] a, int idx1, int idx2) {
        if(a == null)
                return;
        if(idx1 < 0 || idx1 >= a.length)
                return;
        if(idx2 < 0 || idx2 >= a.length)
                return;
        int temp = a[idx1];
        a[idx1] = a[idx2];
        a[idx2] = temp;
}
```

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 $\{6, 1, 3, 8, 2, 5, 9\}$

Solution:

```
6 1 3 8 2 5 9
6 1 3 8 2 5 9
6 3 1 8 2 5 9
8 6 3 1 2 5 9
8 6 3 2 1 5 9
8 6 5 3 2 1 9
9 8 6 5 3 2 1
```

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.

   Solution:

```java
public int compareTo(Fraction other) {
        if(equals(other))
                return 0;
        if(getRealValue() > other.getRealValue())
                return 1;
        return -1; //this object's real value < other object's real value
}
```

6. **`bubbleSort`**

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

```java
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]);
}
```

   Solution:

```
i=0 {5/2, 10/20, 2/5, 100/100, 6/5}
k=0 {10/20, 5/2, 2/5, 100/100, 6/5}
k=1 {10/20, 2/5, 5/2, 100/100, 6/5}
k=2 {10/20, 2/5, 100/100, 5/2, 6/5}
k=3 {10/20, 2/5, 100/100, 6/5, 5/2}

i=1 {10/20, 2/5, 100/100, 6/5, 5/2}
k=0 {2/5, 10/20, 100/100, 6/5, 5/2}
k=1 {2/5, 10/20, 100/100, 6/5, 5/2}
```

```
k=2 {2/5, 10/20, 100/100, 6/5, 5/2}

i=2 {2/5, 10/20, 100/100, 6/5, 5/2}
k=0 {2/5, 10/20, 100/100, 6/5, 5/2}
k=1 {2/5, 10/20, 100/100, 6/5, 5/2}

i=3 {2/5, 10/20, 100/100, 6/5, 5/2}
k=0 {2/5, 10/20, 100/100, 6/5, 5/2}

Final array: {2/5, 10/20, 100/100, 6/5, 5/2}
```

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

Solution: $B(n) = W(n) = O(n^2)$

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.

Solution: When you trace the method `buggySort` for the `Fraction` array `{5/2, 10/20, 2/5, 100/100, 6/5}`, you will see that its the references `first` and `second` that are swapped but the actual items of the arrays `fractions[k]` and `fractions[k+1]` are not swapped. Thus we should swap the items of the arrays and not their shallow copies.

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

```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;
        }
}
```

Solution:

```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 && nDigits(a[k]) > nDigits(backup)) {
                        a[k+1] = a[k];
                        k--;
                }
                a[k+1] = backup;
        }
}

public static int nDigits(int n) {
        int count = 0;
        while(n != 0) {
                count++;
                n/=10;
        }
        return count;
}
```

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.

Solution: Re-referencing the array as `fractions = temp;` doesn't work as any re-referencing applied to a parameter array is ignored upon exiting the method. Hence, we need to re-reference each item of the array individually.

```
for(int i=0; i<temp.length; i++)
        fractions[i] = temp[i];
```