# CSCI 230 HW#5

Collaboration policy: **Individual Assignment**
Total Points: **100**

## Source Code

The Java classes and/or interfaces provided in the zip file attached to this Dropbox assignment are:

- `ArrayList.java`
- `DoublyLinkedList.java`
- `List.java`
- `Utils.java`

Under no circumstances are you allowed to modify or create new `List` interface. You must use these files **as is**.

You **may only** modify the `ArrayList`, `DoublyLinkedList`, and `Utils` classes. In particular, in this class you **may only** modify the methods listed in Part 1, and under no circumstances are you allowed to remove, add, or modify any other line of code in these classes (this include instance variables, class variables, constants, etc.).

Lastly, you **may not** change the package structure! Specifically, `edu.cofc.csci230` cannot be removed or modified. If a solution is submitted with a different package structure, it will not be graded, no exceptions.

## Part 1

In the `ArrayList` and `DoublyLinkedList` classes please fully implement the method listed below:

- `public void swap( int i, int j ) throws IndexOutOfBoundsException`

In the `Utils` class please fully implement the methods listed below:

- `public static <AnyType extends Comparable> void selectionSort( List<AnyType> list ) throws IndexOutOfBoundsException`
- `public static <AnyType extends Comparable> void bubbleSort( List<AnyType> list ) throws IndexOutOfBoundsException`
- `public static <AnyType extends Comparable> void insertionSort( List<AnyType> list ) throws IndexOutOfBoundsException`

In each method you will see a TODO comment - this is where you add your code. In the provided source code, numerous comments are given; please ensure you read them carefully. Note: the algorithms must sort the list in ascending or descending order.

## Part 2

The provided `ArrayList` and `DoublyLinkedList` classes each have a main method. In the main please add test cases that demonstrate you have fully evaluated the operational correctness of the methods you implemented in Part 1. To receive full credit, these test cases **must** be included.

## Submission

Create a zip file that **only** includes the completed `ArrayList.java` and `DoublyLinkedList.java` and `Utils.java` files. If you have any questions about the submission policy, you must resolve before the due date. Lastly, please plan appropriately, asking questions the day the assignment is due (within 12 hours) is too late. Please try to resolve any questions at least 2 days before the due date.

The name of the zip file must be your last name. For example, *ritchie.zip* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Only assignments submitted in the correct format will be accepted (no exceptions).

Please submit the zip file (via OAKS) to the Dropbox setup for this assignment by the due date. You may resubmit the zip file as many times as you like, Dropbox will only keep the newest submission. Per the syllabus, late assignments will not be accepted – no exceptions. Please do not email Hassam or I your assignment after the due date, we will not accept it.

## Grading Rubric

| | |
|---|---|
| ArrayList, DoublyLinkedList, and Utils Compiles | 10 points |
| Thoroughness of your test cases | 10 points |
| Instructor test cases. Several random test cases that sort random values (may include duplicates) in both the ArrayList and DoublyLinkedList data structures. In total these will be 80 points. | 80 points |
| | 100 points |

In particular, each data structure will be graded as follows. If the submitted solution

- Does not compile: 0 of 100 points
- Compiles but does not run: 10 of 100 points
- Thoroughness of your test cases: 20 of 100 points
- Passes test cases developed by instructor: 100 of 100 points.

---

**ALGORITHM** *SelectionSort*$(A[0..n-1])$

```
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]   min ← j
    swap A[i] and A[min]
```

**ALGORITHM** *BubbleSort*$(A[0..n-1])$

**// The algorithm sorts array** $A[0..n-1]$ **by bubble sort**

**// Input : An array** $A[0..n-1]$ **of orderable elements**

**// Output : Array** $A[0..n-1]$ **sorted in ascending order**

**for** $i \leftarrow 0$ **to** $n$-2 **do**

  **for** $j \leftarrow 0$ **to** $n$-2 $-$ $i$ **do**

    **if** $A[j+1] < A[j]$ **swap** $A[j]$ **and** $A[j+1]$

**ALGORITHM** *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```