Logbook 1 (Weeks 1-7)

https://github.com/ZMRamsey/APDCoursework.git
The full package, including a copy of this logbook, is stored in this git repository.

Week 1/2 - Randomiser

The randomiser package creates an array of non-repeating integers, which are used by the searcher methods to find the kth element within them.

I created three versions of the CleverSearcher class that have slightly different features.

CleverSearcher was the first to be made, and uses a sub array of size k, which stores the k largest elements seen at that point in the array. For the first k ints, this simply copies each value it comes across. From this point, it runs across the rest of the main array testing each int. If an element is found that is larger than the smallest element in the sub array, the new int overwrites it. Then, the sub array is sorted using the standard Arrays.sort() method.

This class isn't the most efficient way to find the kth element, since it relies on fully sorting the sub array. In extreme cases with vast arrays, this can severely delay the program compared to alternatives. Despite this, it works well, passing every test.

FullCleverSearcher improves on the idea of CleverSearcher by more efficiently sorting the sub array. Instead of sorting the sub array every time a new large element is found, the element is instead compared to the next largest element in the sub array. If it is larger, they are swapped. This means that the sub array stays perfectly ordered, and any sorting can be more focused on just the elements that need to be changed.

FullCleverSearcher also introduces another improvement. If k is greater than half the size of the full array, it will instead find the length - k smallest. For example, it is quicker to find the 10th smallest of 100 rather than the 90th largest, since it requires a smaller sub array and so less sorting.

GenericSearcher, the final searcher, uses generic typing to allow it to find the kth element in any comparable array of objects. The algorithm is the same as FullCleverSearcher, using a k size sub array and swapping until the elements are in the right positions. This searcher had to be tested with Integers instead of ints due to the latter's incompatibility with generics.

testMinusNumberError and testOutOfBounds

```
@Test
void testMinusNumberError() throws IndexingError {
    try{
        testSearcher(50, -1);
        fail("Expected indexing error didn't occur");
    } catch (IndexingError indexingError)
    {
        //Successful test
    }
}

@Test
void testOutOfBounds() throws IndexingError {
    try{
        testSearcher(100, 103);
        fail("Expected indexing error didn't occur");
    } catch (IndexingError indexingError)
    {
        //Successful test
    }
}
```

Each searcher has a relevant jUnit test which extends the standard SearcherTest. I added additional tests to the SearcherTest class to make sure the searchers were being checked for everything. This included two methods, testMinusNumberError and testOutOfBounds, to make sure that error handling was correctly implemented.

Week 3/4 - Generic Swap

The swap function is relatively simple, utilising the standard technique of having a temporary variable to store one of the values.

Generic Swap Function

```
public static <T> T[] swap(T[] array, int var1, int var2)
{
    T handler;
    if (var1 < array.length && var2 < array.length) {
        handler = array[var1];
        array[var1] = array[var2];
        array[var2] = handler;
    }
    return array;
}
```