# School of Computer and Security Science

# CSP2348/CSP5243 Data Structures

# Programming Practice 1

# Java Arrays

1. Write an algorithm to test whether an array *a [left.* . *.right]* is sorted in ascending order.
   1. In terms of the number of comparisons required, determine the time efficiency of your algorithm: in the best case; in the worst case; and on average.
   2. Implement your algorithm as a Java method, assuming that the array elements are Comparable objects.
2. A *palindrome* is a sequence that is identical to the reverse of itself. For example, the sequence «'m', 'a', 'd', 'a', 'm'» is a palindrome.
   1. Write an algorithm to test whether a character array *a[left. ..right]* is a palindrome.
   2. What are the time efficiency and space efficiency of your algorithm?
   3. Implement your algorithm as a Java method.
3. Consider the implementations of selection sort (Program 3.30), insertion sort (Program 3.34), merge-sort (Program 3.37), and quick-sort (Program 3.44).
   1. Modify each of these Java methods to count the number of comparisons and the number of copies. Run the modified sorting methods with a range of array lengths (say 10, 50, 100,500, 1000, and 5000).
   2. Compare your experimental results with the theoretical results given in Table 3.45.
4. Consider the sorting algorithm given below.

To sort *a[left. ..right]:*

1. For *i* = 0, ..., *right-left-I,* repeat:

1.1. For j = *left+1*, ..., *right-i,* repeat:

1.1.1. If *a[j-1]* is greater than *a[j],* swap *a[j-1]* and *a[j].*

2. Terminate.

**Array bubble sort algorithm**

1. Implement this algorithm as a Java method.
2. Modify your code to count the number of comparisons and the number of copies.
3. Run your method with a range of array lengths. Compare your experimental results with those given in Table 3.45 for other sorting algorithms.