Selection Sort

```java
//selectionSort
//sort the array in ascending order (in place)
//parameters:
//  data: an array of integers
public static void selectionSort(int[] data){
    //for each item in the array
    //  find the index of the minimum
    //  swap it into place
    for(int i = 0; i < data.length - 1; i++){
        int index = findMinimumIndex(data, i);
        //check if we need to swap
        if(index != i){
            swap(data, i, index);
        }
    }
}
```

Insertion Sort

```java
/* Insertion Sort
 * For each item stating at 1
 *   while the item is < the one to its left
 *     swap the item with the one to its left
 */
public static void insertionSort(int[] data) {
    for(int i = 1; i < data.length; i++) {
        int idx = i;
        while(idx >= 1 && data[idx] < data[idx-1]) {
            //swap
            swap(data, idx, idx-1);
            idx--;
        }
    }
}
```

Bubble Sort

```java
/* Bubble Sort
 * Repeat while items have been swapped
 *   for each item (up to the second to last)
 *     if the item is > the one to its right
 *        swap item with the one to its right
 */

public static void bubbleSort(int[] data) {

    boolean swaps = false;
    int steps = 1;
    //save a little time since we know each step puts the end item in order
    do {
        swaps = false;
        for(int j = 0; j < data.length-steps; j++) {
            if(data[j] > data[j+1]) {
                swap(data, j, j+1);
                swaps = true;
            }
        }
        steps++;
    } while(swaps);
}
```

Quick Sort

```java
public static void quickSort(int[] data) {
    quickSort(data, 0, data.length-1);
}

/*
 * Partition the data into two sections
 * (smaller part and larger part)
 * Quick sort the two sections
 */
private static void quickSort(int[] data, int start, int end) {
    if (start >= 0 && end >= 0 && start < end) {
        int p = partition(data, start, end);
        quickSort(data, start, p);
        quickSort(data, p+1, end);
    }
}
```

```java
/*
 * Partition:
 * Pick a pivot item
 * Move all of the data smaller than that item before it
 * Move all the data larger than that item after it.
 * Return the location of the pivot item
 */
private static int partition(int[] data, int start, int end) {

    int pivot = data[start];
    int i = start - 1;
    int j = end + 1;
    while(true) {

        do { i++; } while (data[i] < pivot);
        do { j--; } while (data[j] > pivot);

        if(i >= j) {
            return j;
        }
        swap(data, i, j);

    }
}
```