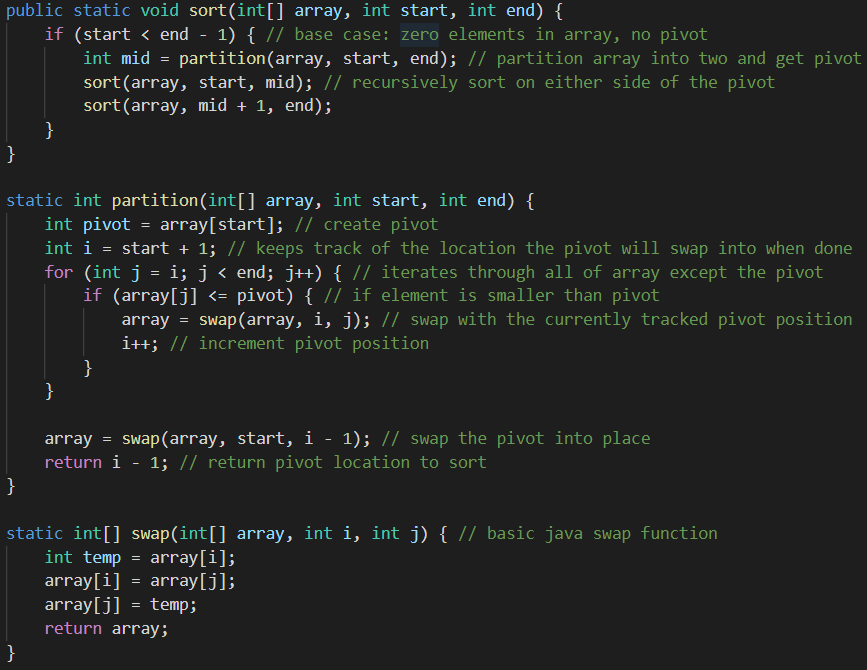
## QuickSort Analysis:



The partition function runs in O(n) because it has to iterate through the entire length of the array.

In order to get the runtime complexity of sort we have to solve the following recurrence relation:

Since , according to Master’s Theorem.

The worst-case scenario for sort is that partition always returns the first element of the array as the pivot because that forces sort to be called recursively times. Therefore, the worst-case time complexity of sort is O(n2)

Line by line breakdown below:

public static void sort(int[] array, int start, int end) { // O(n2)

    if (start < end - 1) { // O(n2)

        int mid = partition(array, start, end); // O(n)

        sort(array, start, mid); // O(n2)

        sort(array, mid + 1, end); // O(n2)

    }

}

static int partition(int[] array, int start, int end) {

    int pivot = array[start]; // O(1)

    int i = start + 1; // O(1)

    for (int j = i; j < end; j++) { // O(1)

        if (array[j] <= pivot) { // O(1)

            array = swap(array, i, j); // O(1)

            i++; // O(1)

        }

    }

     array = swap(array, start, i - 1); // O(1)

    return i - 1; // O(1)

}

static int[] swap(int[] array, int i, int j) { // O(1)

    int temp = array[i]; // O(1)

    array[i] = array[j]; // O(1)

    array[j] = temp; // O(1)

    return array; // O(1)

}