Merge sort

```
public class MergeSort {
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int mid = array.length / 2;
            int[] left = Arrays.copyOfRange(array, 0, mid);
            int[] right = Arrays.copyOfRange(array, mid, array.length);

            mergeSort(left);
            mergeSort(right);

            merge(array, left, right);
        }
    }

    private static void merge(int[] array, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) {
                array[k++] = left[i++];
            } else {
                array[k++] = right[j++];
            }
        }

        while (i < left.length) {
            array[k++] = left[i++];
        }

        while (j < right.length) {
            array[k++] = right[j++];
        }
    }
}
```

Quick sort

```
public class QuickSort {
    public static void quickSort(int[] array, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(array, low, high);
            quickSort(array, low, pivotIndex - 1);
            quickSort(array, pivotIndex + 1, high);
        }
```

```
    }

    private static int partition(int[] array, int low, int high) {
        int pivot = array[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (array[j] <= pivot) {
                i++;
                swap(array, i, j);
            }
        }

        swap(array, i + 1, high);
        return i + 1;
    }

    private static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

Heap sort

```
public class HeapSort {
    public static void heapSort(int[] array) {
        int n = array.length;

        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(array, n, i);
        }

        for (int i = n - 1; i > 0; i--) {
            int temp = array[0];
            array[0] = array[i];
            array[i] = temp;

            heapify(array, i, 0);
        }
    }

    private static void heapify(int[] array, int n, int i) {
        int largest = i;
```

```
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && array[left] > array[largest]) {
            largest = left;
        }

        if (right < n && array[right] > array[largest]) {
            largest = right;
        }

        if (largest != i) {
            int temp = array[i];
            array[i] = array[largest];
            array[largest] = temp;

            heapify(array, n, largest);
        }
    }
}
```