Solution 1

```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Program {
  // Best: O(nlog(n)) time | O(1) space
  // Average: O(nlog(n)) time | O(1) space
  // Worst: O(nlog(n)) time | O(1) space
  public static int[] heapSort(int[] array) {
    buildMaxHeap(array);
    for (int endIdx = array.length - 1; endIdx > 0; endIdx--) {
      swap(0, endIdx, array);
      siftDown(0, endIdx - 1, array);
    }
    return array;
  }

  public static void buildMaxHeap(int[] array) {
    int firstParentIdx = (array.length - 2) / 2;
    for (int currentIdx = firstParentIdx; currentIdx >= 0; currentIdx--) {
      siftDown(currentIdx, array.length - 1, array);
    }
  }

  public static void siftDown(int currentIdx, int endIdx, int[] heap) {
    int childOneIdx = currentIdx * 2 + 1;
    while (childOneIdx <= endIdx) {
      int childTwoIdx = currentIdx * 2 + 2 <= endIdx ? currentIdx * 2 + 2 : -1;
      int idxToSwap;
      if (childTwoIdx != -1 && heap[childTwoIdx] > heap[childOneIdx]) {
        idxToSwap = childTwoIdx;
      } else {
        idxToSwap = childOneIdx;
      }
      if (heap[idxToSwap] > heap[currentIdx]) {
        swap(currentIdx, idxToSwap, heap);
        currentIdx = idxToSwap;
        childOneIdx = currentIdx * 2 + 1;
      } else {
        return;
      }
    }
  }

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

```