**CompletedMerging.java**  
  
import java.util.Random;  
  
*/\*\*  
 \* Implements various divide and conquer algorithms.  
 \*  
 \* Last updated 4/2/2022.  
 \*  
 \* Completion time: (your completion time)  
 \*  
 \* @author Eyad Mohamed AbdelMohsen Ghanem, Acuna, Sedgewick and Wayne  
 \* @verison 1.0  
 \*/*public class CompletedMerging implements MergingAlgorithms {  
  
 @Override  
 public <T extends Comparable> Queue<T> mergeQueues(Queue<T> q1, Queue<T> q2) {  
 Queue<T> mergedQueue = new ListQueue<>();  
  
 while (!q1.isEmpty() && !q2.isEmpty()) {  
 if (q1.peek().compareTo(q2.peek()) <= 0) {  
 mergedQueue.enqueue(q1.dequeue());  
 } else {  
 mergedQueue.enqueue(q2.dequeue());  
 }  
 }  
  
 while (!q1.isEmpty()) {  
 mergedQueue.enqueue(q1.dequeue());  
 }  
  
 while (!q2.isEmpty()) {  
 mergedQueue.enqueue(q2.dequeue());  
 }  
  
 return mergedQueue;  
 }  
  
 @Override  
 public void sort(Comparable[] a) {  
 mergesort(a, 0, a.length - 1);  
 }  
  
 private void mergesort(Comparable[] a, int low, int high) {  
 if (low < high) {  
 int mid = low + (high - low) / 2;  
 mergesort(a, low, mid);  
 mergesort(a, mid + 1, high);  
 merge(a, low, mid, high);  
 }  
 }  
  
 private void merge(Comparable[] a, int low, int mid, int high) {  
 int n1 = mid - low + 1;  
 int n2 = high - mid;  
  
 Comparable[] left = new Comparable[n1];  
 Comparable[] right = new Comparable[n2];  
  
 for (int i = 0; i < n1; i++) {  
 left[i] = a[low + i];  
 }  
  
 for (int j = 0; j < n2; j++) {  
 right[j] = a[mid + 1 + j];  
 }  
  
 int i = 0, j = 0, k = low;  
  
 while (i < n1 && j < n2) {  
 if (left[i].compareTo(right[j]) <= 0) {  
 a[k] = left[i];  
 i++;  
 } else {  
 a[k] = right[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 a[k] = left[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 a[k] = right[j];  
 j++;  
 k++;  
 }  
 }  
  
 @Override  
 public Comparable[] mergesort(Comparable[] a) {  
 mergesort(a, 0, a.length - 1);  
 return a;  
 }  
  
 @Override  
 public Comparable[] merge(Comparable[] a, Comparable[] b) {  
 int n1 = a.length;  
 int n2 = b.length;  
 Comparable[] merged = new Comparable[n1 + n2];  
  
 int i = 0, j = 0, k = 0;  
  
 while (i < n1 && j < n2) {  
 if (a[i].compareTo(b[j]) <= 0) {  
 merged[k] = a[i];  
 i++;  
 } else {  
 merged[k] = b[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 merged[k] = a[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 merged[k] = b[j];  
 j++;  
 k++;  
 }  
  
 return merged;  
 }  
  
 @Override  
 public void shuffle(Object[] a) {  
 Random random = new Random();  
 int n = a.length;  
  
 for (int i = 0; i < n; i++) {  
 int j = random.nextInt(n);  
 Object temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
 }  
 }  
  
 */\*\*  
 \* entry point for sample output.  
 \*  
 \* @param args the command line arguments  
 \*/* public static void main(String[] args) {  
 Queue<String> q1 = new ListQueue<>();  
 q1.enqueue("E");  
 q1.enqueue("L");  
 q1.enqueue("O");  
 q1.enqueue("R");  
 q1.enqueue("T");  
 Queue<String> q2 = new ListQueue<>();  
 q2.enqueue("A");  
 q2.enqueue("E");  
 q2.enqueue("M");  
 q2.enqueue("P");  
 q2.enqueue("S");  
 q2.enqueue("X");  
 Queue<Integer> q3 = new ListQueue<>();  
 q3.enqueue(5);  
 q3.enqueue(12);  
 q3.enqueue(15);  
 q3.enqueue(17);  
 q3.enqueue(20);  
 Queue<Integer> q4 = new ListQueue<>();  
 q4.enqueue(1);  
 q4.enqueue(4);  
 q4.enqueue(12);  
 q4.enqueue(13);  
 q4.enqueue(16);  
 q4.enqueue(18);  
  
 MergingAlgorithms ma = new CompletedMerging();  
  
 //Q1 - sample test cases  
 Queue merged1 = ma.mergeQueues(q1, q2);  
 System.*out*.println(merged1.toString());  
 Queue merged2 = ma.mergeQueues(q3, q4);  
 System.*out*.println(merged2.toString());  
  
 //Q2 - sample test cases  
 String[] a = {"S", "O", "R", "T", "E", "X", "A", "M", "P", "L", "E"};  
 ma.sort(a);  
 assert *isSorted*(a);  
 *show*(a);  
  
 //Q3 - sample test cases  
 String[] b = {"S", "O", "R", "T", "E", "X", "A", "M", "P", "L", "E"};  
 ma.shuffle(b);  
 *show*(b);  
  
 ma.shuffle(b);  
 *show*(b);  
 }  
  
 //below are utilities functions, please do not change them.  
  
 //sorting helper from text  
 private static boolean less(Comparable v, Comparable w) {  
 return v.compareTo(w) < 0;  
 }  
  
 //sorting helper from text  
 private static void show(Comparable[] a) {  
 for (Comparable a1 : a)  
 System.*out*.print(a1 + " ");  
  
 System.*out*.println();  
 }  
  
 //sorting helper from text  
 public static boolean isSorted(Comparable[] a) {  
 for (int i = 1; i < a.length; i++)  
 if (*less*(a[i], a[i - 1]))  
 return false;  
  
 return true;  
 }  
}