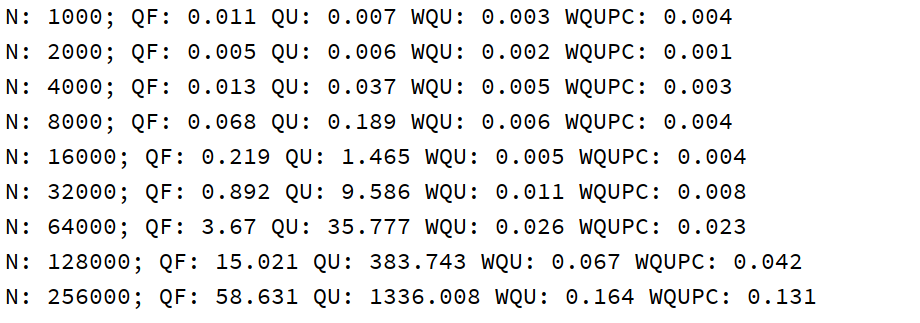
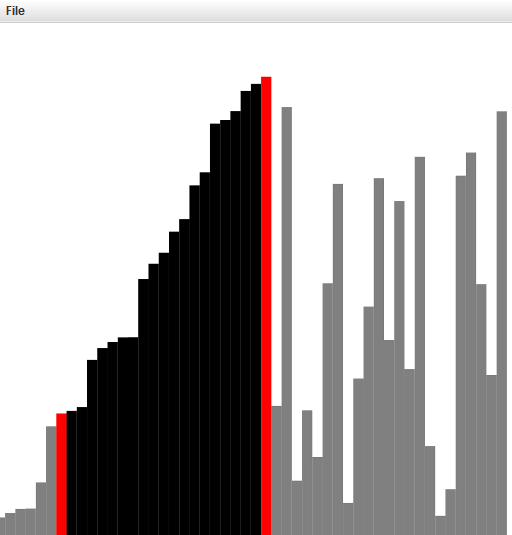
# 作业一 1\_5\_24

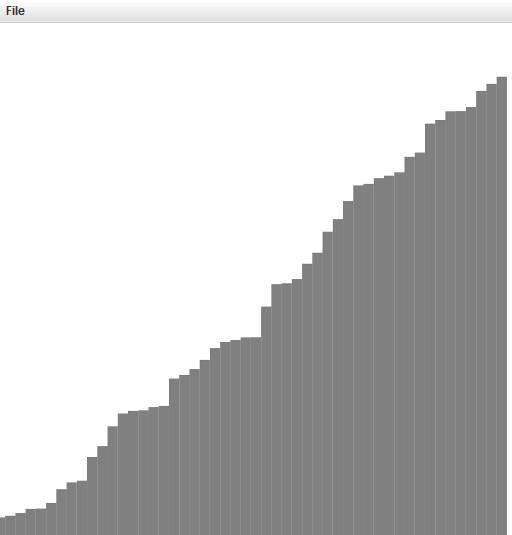
**package** algs;  
  
**import** edu.princeton.cs.algs4.\*;  
**import** edu.princeton.cs.algs4.WeightedQuickUnionUF;  
  
*/\*\*  
 \** ***@author*** *suyifan  
 \** ***@version*** *1.0  
 \** ***@date*** *2020/4/18 8:53  
 \*/***public class** ErdosRenyi {  
 *// Union-find算法* **public static int** count(**int** N) {  
 **int** edge = 0;  
 UF uf = **new** UF(N);  
 **while** (uf.count() != 1) {  
 **int** i = StdRandom.*uniform*(N);  
 **int** j = StdRandom.*uniform*(N);  
 **if** (uf.find(i) == uf.find(j)) {  
 edge++;*//按照题意应该是生成的整数对的数量* **continue**;*//connected方法已弃用！* }  
 uf.union(i, j);  
 edge++;  
 }  
 **return** edge;  
 }  
  
 *// QuickFind算法* **public static int** count1(**int** N) {  
 **int** edge = 0;  
 QuickFindUF QF = **new** QuickFindUF(N);  
 **while** (QF.count() != 1) {  
 **int** i = StdRandom.*uniform*(N);  
 **int** j = StdRandom.*uniform*(N);  
 **if** (QF.find(i) == QF.find(j)) {  
 edge++;*//按照题意应该是生成的整数对的数量* **continue**;*//connected方法已弃用！* }  
 QF.union(i, j);  
 edge++;  
 }  
 **return** edge;  
 }  
  
 *//QuickUnion算法* **public static int** count2(**int** N) {  
 **int** edge = 0;  
 QuickUnionUF QU = **new** QuickUnionUF(N);  
 **while** (QU.count() != 1) {  
 **int** i = StdRandom.*uniform*(N);  
 **int** j = StdRandom.*uniform*(N);  
 **if** (QU.find(i) == QU.find(j)) {  
 edge++;  
 **continue**;  
 }  
 QU.union(i, j);  
 edge++;  
 }  
 **return** edge;  
 }  
  
 *//加权Quick-Union算法* **public static int** count3(**int** N) {  
 **int** edge = 0;  
 WeightedQuickUnionUF WQU = **new** WeightedQuickUnionUF(N);  
 **while** (WQU.count() != 1) {  
 **int** i = StdRandom.*uniform*(N);  
 **int** j = StdRandom.*uniform*(N);  
 **if** (WQU.find(i) == WQU.find(j)) {  
 edge++;  
 **continue**;  
 }  
 WQU.union(i, j);  
 edge++;  
 }  
 **return** edge;  
 }  
  
 *//路径压缩的加权Quick-Union算法* **public static int** count4(**int** N) {  
 **int** edge = 0;  
 WeightedQuickUnionPathCompressionUF WQUPC = **new** WeightedQuickUnionPathCompressionUF(N);  
 **while** (WQUPC.count() != 1) {  
 **int** i = StdRandom.*uniform*(N);  
 **int** j = StdRandom.*uniform*(N);  
 **if** (WQUPC.find(i) == WQUPC.find(j)) {  
 edge++;  
 **continue**;  
 }  
 WQUPC.union(i, j);  
 edge++;  
 }  
 **return** edge;  
 }  
  
 **public static void** main(String[] args) {  
 **int** N = 1000;  
 **int** T = 20;  
 **for** (**int** i = 0; i < T; i++) {  
 *//运行* Stopwatch timer1 = **new** Stopwatch();  
 *count1*(N);  
 **double** time1 = timer1.elapsedTime();  
  
 Stopwatch timer2 = **new** Stopwatch();  
 *count2*(N);  
 **double** time2 = timer2.elapsedTime();  
  
 Stopwatch timer3 = **new** Stopwatch();  
 *count3*(N);  
 **double** time3 = timer3.elapsedTime();  
  
 Stopwatch timer4 = **new** Stopwatch();  
 *count4*(N);  
 **double** time4 = timer4.elapsedTime();  
*// StdOut.println("N = " + N + " 比值: " + time1 / time2);* StdOut.*println*(**"N: "** + N +**"; QF: "** + time1 + **" QU: "** + time2 + **" WQU: "** + time3 + **" WQUPC: "** + time4);  
 N += N;  
 }  
 }  
  
}

**实验结果**

# 作业二 2\_1\_17

**package** algs;  
  
**import** edu.princeton.cs.algs4.StdDraw;  
  
*/\*\*  
 \** ***@author*** *suyifan  
 \** ***@version*** *1.0  
 \** ***@date*** *2020/4/18 10:53  
 \*/***public class** Algs2\_1\_17 {  
  
 **public static void** sort(Comparable[] a) {  
 **int** N = a.**length**;  
 **for** (**int** i = 1; i < N; i++) {  
 **int** min = i;  
 **for** (**int** j = i; j > 0 && *less*(a[j], a[j - 1]); j--) {  
 *exch*(a, j - 1, j);  
 min = j;  
 }  
 *show*(a, i, min);  
 StdDraw.*pause*(100);*//显示所有图像并暂停500ms* StdDraw.*clear*();  
 }  
 *show*(a, N, N);  
 }  
  
  
 **private static boolean** less(Comparable v, Comparable w) {  
 **return** v.compareTo(w) < 0;  
 }  
  
 **private static void** exch(Comparable[] a, **int** i, **int** j) {  
 Comparable t = a[i];  
 a[i] = a[j];  
 a[j] = t;  
 }  
  
 **private static void** show(Comparable[] a, **int** now, **int** min) {  
 **int** N = a.**length**;  
 **for** (**int** i = 0; i < N; i++) {  
 **double** x = 1.0 \* i / N;  
 **double** y = (Double) a[i] / 2.0;  
 **double** rw = 0.5 / N;  
 **double** rh = (Double) a[i] / 2.0;  
 **if** (i == min || i == now) {  
 StdDraw.*setPenColor*(StdDraw.***RED***);  
 StdDraw.*filledRectangle*(x, y, rw, rh);  
 } **else if** (i < now && i > min) {  
 StdDraw.*setPenColor*(StdDraw.***BLACK***);  
 StdDraw.*filledRectangle*(x, y, rw, rh);  
 } **else** {  
 StdDraw.*setPenColor*(StdDraw.***GRAY***);  
 StdDraw.*filledRectangle*(x, y, rw, rh);  
 }  
 }  
 }  
  
 **private static boolean** isSorted(Comparable[] a) {  
 **for** (**int** i = 0; i < a.**length**; i++)  
 **if** (*less*(a[i], a[i - 1])) **return false**;  
 **return true**;  
 }  
  
 **public static void** main(String[] args) {  
 **int** N = 100;  
 Comparable[] a = **new** Comparable[N];  
 **for** (**int** i = 0; i < N; i++)  
 a[i] = Math.*random*();  
 *sort*(a);  
 **assert** *isSorted*(a);  
 }  
}

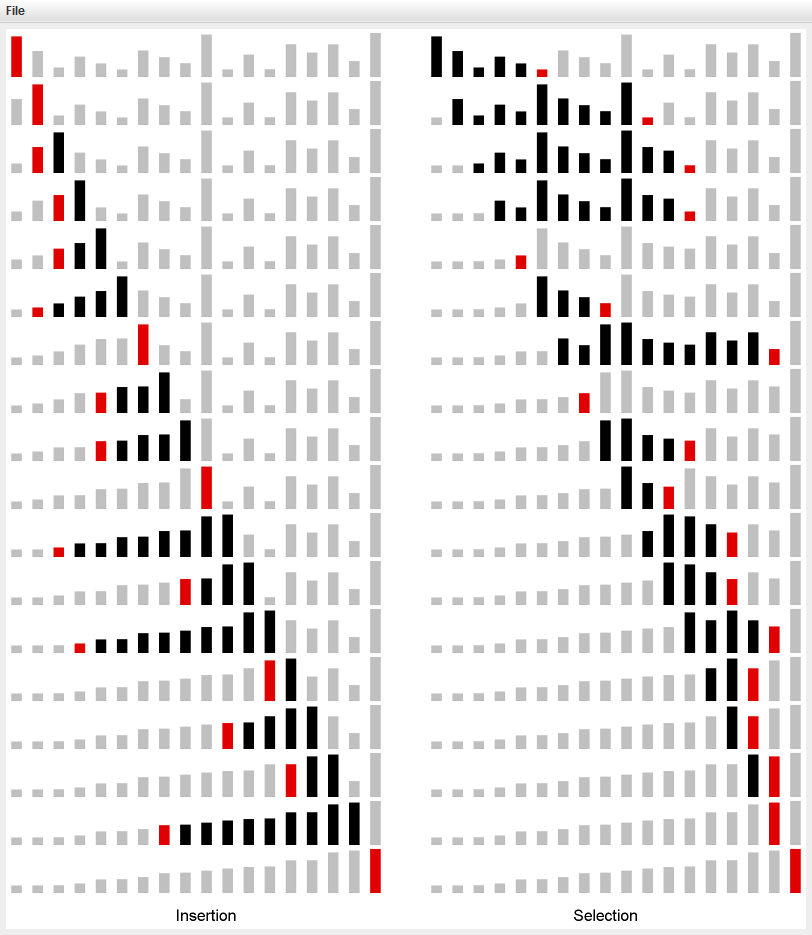
过程截图

结果截图

# 作业三 2\_1\_18（不会做，参考老师的答案做了部分修改）

**package** algs;  
*/\*  
 \* @author suyi  
 \* @version 1.0  
 \* @date 2020/4/26 13:10  
 \*/***import** edu.princeton.cs.algs4.StdDraw;  
**import** edu.princeton.cs.algs4.StdRandom;  
  
**public class** Algs2\_1\_18 {  
 **private static double** *padx* = 0.05; *// 插入图和选择图之间的填充* **private static double** *pady* = 0.04; *// 底部的文本标签填充* **public static class** Insertion {  
 **private static boolean** less(Comparable v, Comparable w) {  
 **return** v.compareTo(w) < 0;  
 }  
  
 **private static void** exch(Comparable[] a, **int** i, **int** j) {  
 Comparable t = a[i];  
 a[i] = a[j];  
 a[j] = t;  
 }  
  
 **public static void** sort(Comparable[] a) {  
 **int** N = a.**length**;  
 *show*(a, 0, 0, 0);  
 **for** (**int** i = 1; i < N; i++) {  
 **int** min = i;  
 **for** (**int** j = i; j > 0 && *less*(a[j], a[j - 1]); j--) {  
 *exch*(a, j, j - 1);  
 min = j;  
 }  
 *show*(a, i, 0, min);  
 }  
 }  
 }  
  
 **public static class** Selection {  
 **private static boolean** less(Comparable v, Comparable w) {  
 **return** v.compareTo(w) < 0;  
 }  
  
 **private static void** exch(Comparable[] a, **int** i, **int** j) {  
 Comparable t = a[i];  
 a[i] = a[j];  
 a[j] = t;  
 }  
  
 **public static void** sort(Comparable[] a) {  
 **int** N = a.**length**;  
 **for** (**int** i = 0; i < N; i++) {  
 **int** min = i;  
 **for** (**int** j = i + 1; j < N; j++) {  
 **if** (*less*(a[j], a[min])) min = j;  
 }  
 *show*(a, i, 1, min);  
 *exch*(a, i, min);  
 }  
 }  
 }  
  
 **public static void** show(Comparable[] a, **int** current, **int** xindex, **int** min) {  
 **double** yfloor = 1 - (1 - *pady*) / a.**length** \* (current + 1); *//从上到下* **double** dx = (1 - *padx*) / a.**length** / 2;  
 **double** x = ((1 - *padx*) / 2 + *padx*) \* xindex; *//从左到右* **for** (**int** i = 0; i < a.**length**; i++) {  
 **if** (i == min)  
 StdDraw.*setPenColor*(224, 0, 0);  
 **else if** (xindex==0 && ((i < current && i > min) || i == current))  
 StdDraw.*setPenColor*(0, 0, 0);  
 **else if** (xindex==1 && ((i > current && i < min) || i == current))  
 StdDraw.*setPenColor*(0, 0, 0);  
 **else** StdDraw.*setPenColor*(192, 192, 192);  
 **double** y = (Double) a[i] / a.**length**;  
 StdDraw.*filledRectangle*(x + dx / 2, yfloor + y / 2, dx / 4, y / 2);  
 x += dx;  
 }  
 StdDraw.*show*();  
 StdDraw.*pause*(30);  
 }  
  
 **public static void** main(String[] args) {  
 *// Setup draw* **int** n = 18;  
 **int** width = 800;  
 **int** height = n \* 50;  
 StdDraw.*setCanvasSize*(width, height);  
 StdDraw.*setScale*();  
 StdDraw.*enableDoubleBuffering*();  
 StdDraw.*text*(1.0 / 4, *pady* / 3, **"Insertion"**);  
 StdDraw.*text*(3.0 / 4, *pady* / 3, **"Selection"**);  
  
 Double[] a = **new** Double[n];  
 Double[] b = **new** Double[n];  
  
 **for** (**int** i = 0; i < n; i++) {  
 Double d = StdRandom.*uniform*(0.1, 0.9);  
 a[i] = d;  
 b[i] = d;  
 }  
 Insertion.*sort*(a);  
 Selection.*sort*(b);  
 }  
}

**结果见下图**



# 作业四 2\_4\_33

package sort;  
  
/\*\*  
 \* @author suyi  
 \* @version 1.0  
 \* @date 2020/5/1 22:14  
 \*/  
public class IndexMinPQ<Key extends Comparable<Key>> {  
 private int N;  
 private int[] pq; //这里保存了优先队列，使用索引来创建最大堆  
 private Key[] keys; //这里存储了优先级之分的元素，并赋给其索引  
 private int[] qp; //可以得知索引在pq中的位置, qp[pq[i]] = i  
  
 public IndexMinPQ(int maxN) {  
 keys = (Key[]) new Comparable[maxN + 1];  
 pq = new int[maxN + 1];  
 qp = new int[maxN + 1];  
 for (int i = 0; i < maxN + 1; i++) qp[i] = -1;//表示优先队列中无此元素  
 }  
  
 public boolean isEmpty() {  
 return N == 0;  
 }  
  
 public boolean contain(int k) {  
 return pq[k] == -1;  
 }  
  
 public void insert(int k, Key key) {  
 N++;  
 qp[k] = N;  
 pq[N] = k;  
 keys[k] = key;  
 swim(N); // 对pq[]中的索引值进行操作, 但比较的还是keys中的元素  
 }  
  
 public Key min() {  
 return keys[pq[1]];  
 }  
  
 public int delMin() {  
 int indexOfMin = pq[1];  
 exch(1, N--);  
 sink(1);  
 keys[pq[N + 1]] = null;  
 qp[pq[N + 1]] = -1;  
 return indexOfMin;  
 }  
  
 public void change(int k ,Key key){  
 keys[k] = key;  
 swim(qp[k]); //qp[k]表示pq中元素为k的索引位置  
 sink(qp[k]);  
 }  
  
 private void swim(int k) {  
 while (k > 1 && less(k / 2, k)) {  
 exch(k / 2, k);  
 k = k/2;  
 }  
 }  
  
 private void sink(int k) {  
 while (2 \* k <= N) {  
 int j = 2 \* k;  
 if (j < N && less(j, j + 1)) j++;  
 if (!less(k, j)) break;  
 exch(k, j);  
 k = j;  
 }  
 }  
  
 //exch方法要改变pq[]和qp[]中的元素  
 private void exch(int i, int j) {  
 int temp = pq[i];  
 pq[i] = pq[j];  
 pq[j] = temp;  
 //接下来对qp[]进行更新  
 qp[pq[i]] = i;  
 qp[pq[j]] = j;  
 }  
 private boolean less(int i, int j) {  
 Key key1 = keys[pq[i]];  
 Key key2 = keys[pq[j]];  
 return key1.compareTo(key2) > 0;  
 }  
}

### 用例测试代码

package sort;  
  
import edu.princeton.cs.algs4.In;  
import edu.princeton.cs.algs4.StdOut;  
  
/\*\*  
 \* @author suyi  
 \* @version 1.0  
 \* @date 2020/5/1 23:01  
 \*/  
public class Multiway {  
 public static void merge(In[] streams) {  
 int N = streams.length;  
 IndexMinPQ<String> pq = new IndexMinPQ<>(N);  
 for (int i = 0; i < N; i++) {  
 if (!streams[i].isEmpty())  
 pq.insert(i, streams[i].readString());  
 }  
 while (!pq.isEmpty()) {  
 StdOut.*println*(pq.min());  
 int i = pq.delMin();  
  
 if (!streams[i].isEmpty()) {  
 pq.insert(i, streams[i].readString());  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 int N = args.length;  
 In[] streams = new In[N];  
  
 for (int i = 0; i < N; i++) {  
 streams[i] = new In(args[i]);  
 }  
 *merge*(streams);  
 }  
}

### 用例测试截图



# 作业五 3\_1\_3

package search;  
  
/\*\*  
 \* @author suyi  
 \* @version 1.0  
 \* @date 2020/6/2 10:39  
 \* @purpose 开发一个符号表的实现OrderedSequentialSearchST, 使用有序链表来实现我们的有序符号表API  
 \* 请注意，Comparable接口中的CompareTo方法仅在自定义的数据类型下需要实现，如果Key是String类型的，则不需要  
 \*/  
public class OrderedSequentialSearchST<Key extends Comparable<Key>, Value> {  
 private Node first;  
  
 public static void main(String[] args) {  
 OrderedSequentialSearchST<String, Integer> o = new OrderedSequentialSearchST<>();  
 System.*out*.println(o.isEmpty());  
 o.put("a", 1);  
 o.put("b", 2);  
 o.put("c", 3);  
 o.put("z", 26);  
 o.put("y", 25);  
 o.put("x", 24);  
 o.put("zz", 52);  
 System.*out*.println(o.isEmpty());  
 System.*out*.println(o.first.key);  
 System.*out*.println(o.max());  
 System.*out*.println(o.min());  
 System.*out*.println(o.get("y"));  
 o.delete("x");  
 OrderedSequentialSearchST.Node node = o.first;  
 while (node != null) {  
 System.*out*.println(node.key + " " + node.value);  
 node = node.next;  
 }  
 }  
  
 //小于key的键的数量  
 public int rank(Key key) {  
 int i = 0;  
 Node temp = first;  
 while (temp.key.compareTo(key) <= 0) {  
 i++;  
 if (temp.next != null)  
 temp = temp.next;  
 else  
 break;  
 }  
 return i;  
 }  
  
 public void put(Key key, Value value) {  
 if (first == null) {  
 first = new Node(key, value, null);  
 return;  
 }  
  
 int rank = rank(key);  
 Node temp = first;  
 if (rank == 0) {  
 first = new Node(key, value, temp);  
 }  
 while (rank != 1) {  
 rank--;  
 temp = temp.next;  
 }  
 if (temp.key.compareTo(key) == 0) {  
 temp.value = value;  
 } else {  
 temp.next = new Node(key, value, temp.next);  
 }  
 }  
  
 public Value get(Key key) {  
 int rank = rank(key);  
 Node temp = first;  
 while (rank != 1) {  
 rank--;  
 temp = temp.next;  
 }  
 if (temp.key.compareTo(key) == 0) {  
 return temp.value;  
 } else {  
 return null;  
 }  
 }  
  
 public void delete(Key key) {  
 int rank = rank(key);  
 Node temp = first;  
 if (rank == 1 && temp.key.compareTo(key) == 0) {  
 first = null;  
 }  
 while (rank != 2) {  
 rank--;  
 temp = temp.next;  
 }  
 if (temp.next.key.compareTo(key) == 0) {  
 temp.next = temp.next.next;  
 }  
 }  
  
 public boolean contains(Key key) {  
 int rank = rank(key);  
 Node temp = first;  
 if (rank == 0) {  
 return false;  
 }  
 while (rank != 1) {  
 rank--;  
 temp = temp.next;  
 }  
 return temp.key.compareTo(key) == 0;  
 }  
  
 public boolean isEmpty() {  
 return first == null;  
 }  
  
 public Key max() {  
 Node temp = first;  
 if (temp == null)  
 return null;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
 return temp.key;  
 }  
  
 public Key min() {  
 return first.key;  
 }  
  
 private class Node {  
 Key key;  
 Value value;  
 Node next;  
  
 public Node(Key key, Value value, Node next) {  
 this.key = key;  
 this.value = value;  
 this.next = next;  
 }  
 }  
}

