```
public class Testing {

	public static void main(String[] args) {
	int [] originalHL = new int[1000000];
	int [] originalLH = new int[1000000];
	int [] random = new int[1000000];
	int [] extra = new int[1000000];
	int mil=1000000;
	long start;
	long end;

	for(int i = 0;i<originalLH.length;i++){
		originalLH[i]=i+1;
	}

	for(int i = 0;i<originalHL.length;i++){
		originalHL[i]=mil;
		mil--;
	}
	System.arraycopy(originalLH, 0, extra, 0, extra.length);

   int index = 0;
   while (index < 1000000) {
      int x = (int) (Math.random() * 1000000);
      if (extra[x] != 0) {
         random[index] = extra[x];
         extra[x] = 0;
         index++;
      }
   }
   /*
    * This is for sorting 100 elements using the Bubble Sort
    */
   int[] bubble100LH = new int[100];
   int[] bubble100HL = new int[100];
   int[] bubble100R = new int[100];

   System.arraycopy(originalLH, 0, bubble100LH, 0, bubble100LH.length);
   start = System.nanoTime();
   Sorts.bubbleSort(bubble100LH);
   end = System.nanoTime() - start;
   System.out.println("bubble100LH time: " + end);

   System.arraycopy(originalHL, 0, bubble100HL, 0, bubble100HL.length);
   start = System.nanoTime();
   Sorts.bubbleSort(bubble100HL);
```

```
end = System.nanoTime() - start;
System.out.println("bubble100HL time: " + end);

System.arraycopy(random, 0, bubble100R, 0, bubble100R.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble100R);
end = System.nanoTime() - start;
System.out.println("bubble100R time: " + end + "\n");

/*
 * This is for sorting 100 elements using the insertion Sort
 */
int[] insertion100LH = new int[100];
int[] insertion100HL = new int[100];
int[] insertion100R = new int[100];

System.arraycopy(originalLH, 0, insertion100LH, 0, insertion100LH.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100LH);
end = System.nanoTime() - start;
System.out.println("insertion100LH time: " + end);

System.arraycopy(originalHL, 0, insertion100HL, 0, insertion100HL.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100HL);
end = System.nanoTime() - start;
System.out.println("insertion100HL time: " + end);

System.arraycopy(random, 0, insertion100R, 0, insertion100R.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100R);
end = System.nanoTime() - start;
System.out.println("insertion100R time: " + end + "\n");

/*
 * This is for sorting 100 elements using the Selection Sort
 */
int[] selection100LH = new int[100];
int[] selection100HL = new int[100];
int[] selection100R = new int[100];

System.arraycopy(originalLH, 0, selection100LH, 0, selection100LH.length);
start = System.nanoTime();
Sorts.selectionSort(selection100LH);
end = System.nanoTime() - start;
System.out.println("selection100LH time: " + end);
```

```
System.arraycopy(originalHL, 0, selection100HL, 0, selection100HL.length);
start = System.nanoTime();
Sorts.selectionSort(selection100HL);
end = System.nanoTime() - start;
System.out.println("selection100HL time: " + end);

System.arraycopy(random, 0, selection100R, 0, selection100R.length);
start = System.nanoTime();
Sorts.selectionSort(selection100R);
end = System.nanoTime() - start;
System.out.println("selection100R time: " + end + "\n");

/*
 * This is for sorting 100 elements using the heap Sort
 */
int[] heap100LH = new int[100];
int[] heap100HL = new int[100];
int[] heap100R = new int[100];

System.arraycopy(originalLH, 0, heap100LH, 0, heap100LH.length);
start = System.nanoTime();
Sorts.HeapSort(heap100LH,heap100LH.length-1);
end = System.nanoTime() - start;
System.out.println("heap100LH time: " + end);

System.arraycopy(originalHL, 0, heap100HL, 0, bubble100HL.length);
start = System.nanoTime();
Sorts.HeapSort(heap100HL,heap100HL.length-1);
end = System.nanoTime() - start;
System.out.println("heap100HL time: " + end);

System.arraycopy(random, 0, heap100R, 0, heap100R.length);
start = System.nanoTime();
Sorts.HeapSort(heap100R,heap100R.length-1);
end = System.nanoTime() - start;
System.out.println("heap100R time: " + end + "\n");

/*
 * This is for sorting 100 elements using the quick Sort
 */
int[] quick100LH = new int[100];
int[] quick100HL = new int[100];
int[] quick100R = new int[100];

System.arraycopy(originalLH, 0, quick100LH, 0, quick100LH.length);
```

```
start = System.nanoTime();
Sorts.QuickSort(quick100LH,0,quick100LH.length-1);
end = System.nanoTime() - start;
System.out.println("quick100LH time: " + end);

System.arraycopy(originalHL, 0, quick100HL, 0, quick100HL.length);
start = System.nanoTime();
Sorts.QuickSort(quick100HL,0,quick100HL.length-1);
end = System.nanoTime() - start;
System.out.println("quick100HL time: " + end);

System.arraycopy(random, 0, quick100R, 0, quick100R.length);
start = System.nanoTime();
Sorts.QuickSort(quick100R,0,quick100R.length-1);
end = System.nanoTime() - start;
System.out.println("quick100R time: " + end + "\n");

/*
 * This is for sorting 100 elements using the Merge Sort
 */
int[] merge100LH = new int[100];
int[] merge100HL = new int[100];
int[] merge100R = new int[100];

System.arraycopy(originalLH, 0, merge100LH, 0, merge100LH.length);
start = System.nanoTime();
Sorts.MergeSort(merge100LH,0,merge100LH.length-1);
end = System.nanoTime() - start;
System.out.println("merge100LH time: " + end);

System.arraycopy(originalHL, 0, merge100HL, 0, merge100HL.length);
start = System.nanoTime();
Sorts.MergeSort(merge100HL,0,merge100HL.length-1);
end = System.nanoTime() - start;
System.out.println("merge100HL time: " + end);

System.arraycopy(random, 0, merge100R, 0, merge100R.length);
start = System.nanoTime();
Sorts.MergeSort(merge100R,0,merge100R.length-1);
end = System.nanoTime() - start;
System.out.println("merge100R time: " + end + "\n");



/*
```

```
 * This is for sorting 1000 elements using the Bubble Sort
 */
int[] bubble1000LH = new int[1000];
int[] bubble1000HL = new int[1000];
int[] bubble1000R = new int[1000];

System.arraycopy(originalLH, 0, bubble1000LH, 0, bubble1000LH.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble1000LH);
end = System.nanoTime() - start;
System.out.println("bubble1000LH time: " + end);

System.arraycopy(originalHL, 0, bubble1000HL, 0, bubble1000HL.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble1000HL);
end = System.nanoTime() - start;
System.out.println("bubble1000HL time: " + end);

System.arraycopy(random, 0, bubble1000R, 0, bubble1000R.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble1000R);
end = System.nanoTime() - start;
System.out.println("bubble1000R time: " + end + "\n");

/*
 * This is for sorting 1000 elements using the insertion Sort
 */
int[] insertion1000LH = new int[1000];
int[] insertion1000HL = new int[1000];
int[] insertion1000R = new int[1000];

System.arraycopy(originalLH, 0, insertion1000LH, 0, insertion1000LH.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000LH);
end = System.nanoTime() - start;
System.out.println("insertion1000LH time: " + end);

System.arraycopy(originalHL, 0, insertion1000HL, 0, insertion1000HL.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000HL);
end = System.nanoTime() - start;
System.out.println("insertion1000HL time: " + end);

System.arraycopy(random, 0, insertion1000R, 0, insertion1000R.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000R);
```

```
end = System.nanoTime() - start;
System.out.println("insertion1000R time: " + end + "\n");

/*
 * This is for sorting 1000 elements using the Selection Sort
 */
int[] selection1000LH = new int[1000];
int[] selection1000HL = new int[1000];
int[] selection1000R = new int[1000];

System.arraycopy(originalLH, 0, selection1000LH, 0, selection1000LH.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000LH);
end = System.nanoTime() - start;
System.out.println("selection1000LH time: " + end);

System.arraycopy(originalHL, 0, selection1000HL, 0, selection1000HL.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000HL);
end = System.nanoTime() - start;
System.out.println("selection1000HL time: " + end);

System.arraycopy(random, 0, selection1000R, 0, selection1000R.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000R);
end = System.nanoTime() - start;
System.out.println("selection1000R time: " + end + "\n");

/*
 * This is for sorting 1000 elements using the heap Sort
 */
int[] heap1000LH = new int[1000];
int[] heap1000HL = new int[1000];
int[] heap1000R = new int[1000];

System.arraycopy(originalLH, 0, heap1000LH, 0, heap1000LH.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000LH,heap1000LH.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000LH time: " + end);
System.arraycopy(originalHL, 0, heap1000HL, 0, bubble1000HL.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000HL,heap1000HL.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000HL time: " + end);
```

```
System.arraycopy(random, 0, heap1000R, 0, heap1000R.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000R,heap1000R.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000R time: " + end + "\n");

/*
 * This is for sorting 1000 elements using the quick Sort
 */
int[] quick1000LH = new int[1000];
int[] quick1000HL = new int[1000];
int[] quick1000R = new int[1000];

System.arraycopy(originalLH, 0, quick1000LH, 0, quick1000LH.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000LH,0,quick1000LH.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000LH time: " + end);

System.arraycopy(originalHL, 0, quick1000HL, 0, quick1000HL.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000HL,0,quick1000HL.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000HL time: " + end);

System.arraycopy(random, 0, quick1000R, 0, quick1000R.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000R,0,quick1000R.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000R time: " + end + "\n");

/*
 * This is for sorting 1000 elements using the Merge Sort
 */
int[] merge1000LH = new int[1000];
int[] merge1000HL = new int[1000];
int[] merge1000R = new int[1000];

System.arraycopy(originalLH, 0, merge1000LH, 0, merge1000LH.length);
start = System.nanoTime();
Sorts.MergeSort(merge1000LH,0,merge1000LH.length-1);
end = System.nanoTime() - start;
System.out.println("merge1000LH time: " + end);

System.arraycopy(originalHL, 0, merge1000HL, 0, merge1000HL.length);
start = System.nanoTime();
```

```
Sorts.MergeSort(merge1000HL,0,merge1000HL.length-1);
end = System.nanoTime() - start;
System.out.println("merge1000HL time: " + end);

System.arraycopy(random, 0, merge1000R, 0, merge1000R.length);
start = System.nanoTime();
Sorts.MergeSort(merge1000R,0,merge1000R.length-1);
end = System.nanoTime() - start;
System.out.println("merge1000R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the Bubble Sort
 */
int[] bubble10000LH = new int[10000];
int[] bubble10000HL = new int[10000];
int[] bubble10000R = new int[10000];

System.arraycopy(originalLH, 0, bubble10000LH, 0, bubble10000LH.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble10000LH);
end = System.nanoTime() - start;
System.out.println("bubble10000LH time: " + end);

System.arraycopy(originalHL, 0, bubble10000HL, 0, bubble10000HL.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble10000HL);
end = System.nanoTime() - start;
System.out.println("bubble10000HL time: " + end);

System.arraycopy(random, 0, bubble10000R, 0, bubble10000R.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble10000R);
end = System.nanoTime() - start;
System.out.println("bubble10000R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the insertion Sort
 */
int[] insertion10000LH = new int[10000];
int[] insertion10000HL = new int[10000];
int[] insertion10000R = new int[10000];

System.arraycopy(originalLH, 0, insertion10000LH, 0, insertion10000LH.length);
start = System.nanoTime();
Sorts.insertionSort(insertion10000LH);
end = System.nanoTime() - start;
```

```
System.out.println("insertion100000LH time: " + end);

System.arraycopy(originalHL, 0, insertion10000HL, 0, insertion10000HL.length);
start = System.nanoTime();
Sorts.insertionSort(insertion10000HL);
end = System.nanoTime() - start;
System.out.println("insertion10000HL time: " + end);

System.arraycopy(random, 0, insertion10000R, 0, insertion10000R.length);
start = System.nanoTime();
Sorts.insertionSort(insertion10000R);
end = System.nanoTime() - start;
System.out.println("insertion10000R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the Selection Sort
 */
int[] selection10000LH = new int[10000];
int[] selection10000HL = new int[10000];
int[] selection10000R = new int[10000];

System.arraycopy(originalLH, 0, selection10000LH, 0, selection10000LH.length);
start = System.nanoTime();
Sorts.selectionSort(selection10000LH);
end = System.nanoTime() - start;
System.out.println("selection10000LH time: " + end);

System.arraycopy(originalHL, 0, selection10000HL, 0, selection10000HL.length);
start = System.nanoTime();
Sorts.selectionSort(selection10000HL);
end = System.nanoTime() - start;
System.out.println("selection10000HL time: " + end);

System.arraycopy(random, 0, selection10000R, 0, selection10000R.length);
start = System.nanoTime();
Sorts.selectionSort(selection10000R);
end = System.nanoTime() - start;
System.out.println("selection100R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the heap Sort
 */
int[] heap10000LH = new int[10000];
int[] heap10000HL = new int[10000];
int[] heap10000R = new int[10000];
```

```
System.arraycopy(originalLH, 0, heap10000LH, 0, heap10000LH.length);
start = System.nanoTime();
Sorts.HeapSort(heap10000LH,heap10000LH.length-1);
end = System.nanoTime() - start;
System.out.println("heap10000LH time: " + end);

System.arraycopy(originalHL, 0, heap10000HL, 0, heap10000HL.length);
start = System.nanoTime();
Sorts.HeapSort(heap10000HL,heap10000HL.length-1);
end = System.nanoTime() - start;
System.out.println("heap10000HL time: " + end);

System.arraycopy(random, 0, heap10000R, 0, heap10000R.length);
start = System.nanoTime();
Sorts.HeapSort(heap10000R,heap10000R.length-1);
end = System.nanoTime() - start;
System.out.println("heap10000R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the quick Sort
 */
int[] quick10000LH = new int[10000];
int[] quick10000HL = new int[10000];
int[] quick10000R = new int[10000];

System.arraycopy(originalLH, 0, quick10000LH, 0, quick10000LH.length);
start = System.nanoTime();
Sorts.QuickSort(quick10000LH,0,quick10000LH.length-1);
end = System.nanoTime() - start;
System.out.println("quick10000LH time: " + end);

System.arraycopy(originalHL, 0, quick10000HL, 0, quick10000HL.length);
start = System.nanoTime();
Sorts.QuickSort(quick10000HL,0,quick10000HL.length-1);
end = System.nanoTime() - start;
System.out.println("quick10000HL time: " + end);

System.arraycopy(random, 0, quick10000R, 0, quick10000R.length);
start = System.nanoTime();
Sorts.QuickSort(quick10000R,0,quick10000R.length-1);
end = System.nanoTime() - start;
System.out.println("quick10000R time: " + end + "\n");

/*
 * This is for sorting 10000 elements using the Merge Sort
 */
```

```
int[] merge10000LH = new int[10000];
int[] merge10000HL = new int[10000];
int[] merge10000R = new int[10000];

System.arraycopy(originalLH, 0, merge10000LH, 0, merge10000LH.length);
start = System.nanoTime();
Sorts.MergeSort(merge10000LH,0,merge10000LH.length-1);
end = System.nanoTime() - start;
System.out.println("merge10000LH time: " + end);

System.arraycopy(originalHL, 0, merge10000HL, 0, merge10000HL.length);
start = System.nanoTime();
Sorts.MergeSort(merge10000HL,0,merge10000HL.length-1);
end = System.nanoTime() - start;
System.out.println("merge10000HL time: " + end);

System.arraycopy(random, 0, merge10000R, 0, merge10000R.length);
start = System.nanoTime();
Sorts.MergeSort(merge10000R,0,merge10000R.length-1);
end = System.nanoTime() - start;
System.out.println("merge10000R time: " + end + "\n");



/*
 * This is for sorting 10000 elements using the Bubble Sort
 */
int[] bubble100000LH = new int[100000];
int[] bubble100000HL = new int[100000];
int[] bubble100000R = new int[100000];

System.arraycopy(originalLH, 0, bubble100000LH, 0, bubble10000LH.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble100000LH);
end = System.nanoTime() - start;
System.out.println("bubble100000LH time: " + end);

System.arraycopy(originalHL, 0, bubble100000HL, 0, bubble100000HL.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble100000HL);
end = System.nanoTime() - start;
System.out.println("bubble100000HL time: " + end);

System.arraycopy(random, 0, bubble100000R, 0, bubble100000R.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble100000R);
```

```
end = System.nanoTime() - start;
System.out.println("bubble100000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the insertion Sort
 */
int[] insertion100000LH = new int[100000];
int[] insertion100000HL = new int[100000];
int[] insertion100000R = new int[100000];

System.arraycopy(originalLH, 0, insertion100000LH, 0, insertion100000LH.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100000LH);
end = System.nanoTime() - start;
System.out.println("insertion100000LH time: " + end);

System.arraycopy(originalHL, 0, insertion100000HL, 0, insertion100000HL.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100000HL);
end = System.nanoTime() - start;
System.out.println("insertion100000HL time: " + end);

System.arraycopy(random, 0, insertion100000R, 0, insertion100000R.length);
start = System.nanoTime();
Sorts.insertionSort(insertion100000R);
end = System.nanoTime() - start;
System.out.println("insertion100000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the Selection Sort
 */
int[] selection100000LH = new int[100000];
int[] selection100000HL = new int[100000];
int[] selection100000R = new int[100000];

System.arraycopy(originalLH, 0, selection100000LH, 0, selection100000LH.length);
start = System.nanoTime();
Sorts.selectionSort(selection100000LH);
end = System.nanoTime() - start;
System.out.println("selection100000LH time: " + end);

System.arraycopy(originalHL, 0, selection100000HL, 0, selection100000HL.length);
start = System.nanoTime();
Sorts.selectionSort(selection100000HL);
end = System.nanoTime() - start;
System.out.println("selection100000HL time: " + end);
```

```
System.arraycopy(random, 0, selection100000R, 0, selection100000R.length);
start = System.nanoTime();
Sorts.selectionSort(selection100000R);
end = System.nanoTime() - start;
System.out.println("selection100000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the heap Sort
 */
int[] heap100000LH = new int[100000];
int[] heap100000HL = new int[100000];
int[] heap100000R = new int[100000];

System.arraycopy(originalLH, 0, heap100000LH, 0, heap100000LH.length);
start = System.nanoTime();
Sorts.HeapSort(heap100000LH,heap100000LH.length-1);
end = System.nanoTime() - start;
System.out.println("heap100000LH time: " + end);

System.arraycopy(originalHL, 0, heap100000HL, 0, heap100000HL.length);
start = System.nanoTime();
Sorts.HeapSort(heap100000HL,heap100000HL.length-1);
end = System.nanoTime() - start;
System.out.println("heap100000HL time: " + end);

System.arraycopy(random, 0, heap100000R, 0, heap100000R.length);
start = System.nanoTime();
Sorts.HeapSort(heap100000R,heap100000R.length-1);
end = System.nanoTime() - start;
System.out.println("heap100000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the Merge Sort
 */
int[] merge100000LH = new int[100000];
int[] merge100000HL = new int[100000];
int[] merge100000R = new int[100000];

System.arraycopy(originalLH, 0, merge100000LH, 0, merge100000LH.length);
start = System.nanoTime();
Sorts.MergeSort(merge100000LH,0,merge100000LH.length-1);
end = System.nanoTime() - start;
System.out.println("merge100000LH time: " + end);

System.arraycopy(originalHL, 0, merge100000HL, 0, merge100000HL.length);
```

```
start = System.nanoTime();
Sorts.MergeSort(merge100000HL,0,merge100000HL.length-1);
end = System.nanoTime() - start;
System.out.println("merge100000HL time: " + end);

System.arraycopy(random, 0, merge100000R, 0, merge100000R.length);
start = System.nanoTime();
Sorts.MergeSort(merge100000R,0,merge100000R.length-1);
end = System.nanoTime() - start;
System.out.println("merge100000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the quick Sort
 */
int[] quick100000LH = new int[100000];
int[] quick100000HL = new int[100000];
int[] quick100000R = new int[100000];

System.arraycopy(originalLH, 0, quick100000LH, 0, quick100000LH.length);
start = System.nanoTime();
Sorts.QuickSort(quick100000LH,0,quick100000LH.length-1);
end = System.nanoTime() - start;
System.out.println("quick100000LH time: " + end);

System.arraycopy(originalHL, 0, quick100000HL, 0, quick100000HL.length);
start = System.nanoTime();
Sorts.QuickSort(quick100000HL,0,quick100000HL.length-1);
end = System.nanoTime() - start;
System.out.println("quick100000HL time: " + end);

System.arraycopy(random, 0, quick100000R, 0, quick100000R.length);
start = System.nanoTime();
Sorts.QuickSort(quick100000R,0,quick100000R.length-1);
end = System.nanoTime() - start;
System.out.println("quick100000R time: " + end + "\n");


/*
 * This is for sorting 1000000 elements using the Bubble Sort
 */
int[] bubble1000000LH = new int[1000000];
int[] bubble1000000HL = new int[1000000];
int[] bubble1000000R = new int[1000000];

System.arraycopy(originalLH, 0, bubble1000000LH, 0, bubble1000000LH.length);
start = System.nanoTime();
```

```
Sorts.bubbleSort(bubble1000000LH);
end = System.nanoTime() - start;
System.out.println("bubble1000000LH time: " + end);

System.arraycopy(originalHL, 0, bubble1000000HL, 0, bubble1000000HL.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble1000000HL);
end = System.nanoTime() - start;
System.out.println("bubble1000000HL time: " + end);

System.arraycopy(random, 0, bubble1000000R, 0, bubble1000000R.length);
start = System.nanoTime();
Sorts.bubbleSort(bubble1000000R);
end = System.nanoTime() - start;
System.out.println("bubble1000000R time: " + end + "\n");

/*
 * This is for sorting 1000000 elements using the insertion Sort
 */
int[] insertion1000000LH = new int[1000000];
int[] insertion1000000HL = new int[1000000];
int[] insertion1000000R = new int[1000000];

System.arraycopy(originalLH, 0, insertion1000000LH, 0, insertion1000000LH.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000000LH);
end = System.nanoTime() - start;
System.out.println("insertion10000000LH time: " + end);

System.arraycopy(originalHL, 0, insertion1000000HL, 0, insertion1000000HL.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000000HL);
end = System.nanoTime() - start;
System.out.println("insertion1000000HL time: " + end);

System.arraycopy(random, 0, insertion1000000R, 0, insertion1000000R.length);
start = System.nanoTime();
Sorts.insertionSort(insertion1000000R);
end = System.nanoTime() - start;
System.out.println("insertion1000000R time: " + end + "\n");

/*
 * This is for sorting 1000000 elements using the Selection Sort
 */
int[] selection1000000LH = new int[1000000];
int[] selection1000000HL = new int[1000000];
```

```
int[] selection1000000R = new int[1000000];

System.arraycopy(originalLH, 0, selection1000000LH, 0, selection1000000LH.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000000LH);
end = System.nanoTime() - start;
System.out.println("selection1000000LH time: " + end);

System.arraycopy(originalHL, 0, selection1000000HL, 0, selection1000000HL.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000000HL);
end = System.nanoTime() - start;
System.out.println("selection1000000HL time: " + end);

System.arraycopy(random, 0, selection1000000R, 0, selection1000000R.length);
start = System.nanoTime();
Sorts.selectionSort(selection1000000R);
end = System.nanoTime() - start;
System.out.println("selection1000000R time: " + end + "\n");

/*
 * This is for sorting 1000000 elements using the heap Sort
 */
int[] heap1000000LH = new int[1000000];
int[] heap1000000HL = new int[1000000];
int[] heap1000000R = new int[1000000];

System.arraycopy(originalLH, 0, heap1000000LH, 0, heap1000000LH.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000000LH,heap1000000LH.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000000LH time: " + end);

System.arraycopy(originalHL, 0, heap1000000HL, 0, heap1000000HL.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000000HL,heap1000000HL.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000000HL time: " + end);

System.arraycopy(random, 0, heap1000000R, 0, heap1000000R.length);
start = System.nanoTime();
Sorts.HeapSort(heap1000000R,heap1000000R.length-1);
end = System.nanoTime() - start;
System.out.println("heap1000000R time: " + end + "\n");

/*
```

```
 * This is for sorting 1000000 elements using the quick Sort
 */
int[] quick1000000LH = new int[1000000];
int[] quick1000000HL = new int[1000000];
int[] quick1000000R = new int[1000000];

System.arraycopy(originalLH, 0, quick1000000LH, 0, quick1000000LH.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000000LH,0,quick1000000LH.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000000LH time: " + end);

System.arraycopy(originalHL, 0, quick1000000HL, 0, quick1000000HL.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000000HL,0,quick1000000HL.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000000HL time: " + end);

System.arraycopy(random, 0, quick1000000R, 0, quick1000000R.length);
start = System.nanoTime();
Sorts.QuickSort(quick1000000R,0,quick1000000R.length-1);
end = System.nanoTime() - start;
System.out.println("quick1000000R time: " + end + "\n");

/*
 * This is for sorting 100000 elements using the Merge Sort
 */
int[] merge1000000LH = new int[1000000];
int[] merge1000000HL = new int[1000000];
int[] merge1000000R = new int[1000000];

System.arraycopy(originalLH, 0, merge1000000LH, 0, merge1000000LH.length);
start = System.nanoTime();
Sorts.MergeSort(merge1000000LH,0,merge1000000LH.length-1);
end = System.nanoTime() - start;
System.out.println("merge1000000LH time: " + end);

System.arraycopy(originalHL, 0, merge1000000HL, 0, merge1000000HL.length);
start = System.nanoTime();
Sorts.MergeSort(merge1000000HL,0,merge1000000HL.length-1);
end = System.nanoTime() - start;
System.out.println("merge1000000HL time: " + end);

System.arraycopy(random, 0, merge1000000R, 0, merge1000000R.length);
start = System.nanoTime();
Sorts.MergeSort(merge1000000R,0,merge1000000R.length-1);
```

```
    end = System.nanoTime() - start;
    System.out.println("merge1000000R time: " + end + "\n");

        }


}



public class Sorts {

        public static int[] bubbleSort(int[] arr) {
                int val;
                //bubble sort starts from the end of list and moves towards the front
                for (int i = 0; i < arr.length - 1; i++) {
                        for (int j = arr.length - 1; j > i; j--) {
                                //exchanges A[j] and A[j-1]
                                if (arr[j - 1] > arr[j]) {
                                        val = arr[j - 1];
                                        arr[j - 1] = arr[j];
                                        arr[j] = val;
                                }
                        }
                }
                return arr;
        }

        public static void insertionSort(int[]arr){
                for(int j= 0;j<arr.length;j++){
                int key =arr[j];
                int i = j-1;

                while(i>=0 && arr[i]>key){
                        arr[i+1] = arr[i];
                        i=i-1;
                }
                arr[i+1]=key;
                }
        }
        /**
         * This is a selection sort that sorts an array from smallest to largest
         *
         * @param A is the array being sorted
         */
```

```
public static void selectionSort(int[] A) {
	for (int j = 0; j <= A.length - 2; j++) {
		int smallest = A[j];
		int index = j;
		//goes through the list to obtain the smallest value
		for (int i = j + 1; i <= A.length - 1; i++) {
			if (A[i] < smallest) {
				smallest = A[i];
				index = i;
			}
		}
		//swapping of value occurs
		int temp = A[index];
		A[index] = A[j];
		A[j] = temp;
	}
}


/**
 * This is a Merge Sort that sorts an array from smallest to largest
 *
 * @param A is the array to be sorted
 * @param p is the initial position of the array
 * @param r is the terminal position of the array
 */
public static void MergeSort(int[] A, int p, int r) {
	// if p > r... returns -- it's the base case for this recursion
	if (p < r) {
		// casting the flooring to be an integer
		int q = ((int) Math.floor((p + r) / 2));
		MergeSort(A, p, q);
		MergeSort(A, q + 1, r);
		Merge(A, p, q, r);
	}
}

// The Merge Sort uses this method as part of its recurrence calls.
// This method sorts the two divided arrays from smallest to biggest all
// into one new array.
private static void Merge(int[] A, int p, int q, int r) {
	int n1 = q - p + 1;
	int n2 = r - q;
	int[] L = new int[n1];
	int[] R = new int[n2];
```

```
for (int i = 1; i <= n1; i++) {
        // i-1 because we're using i to talk about the position... position
        // in java starts at 0
        L[i - 1] = A[p + i - 1];
}

for (int j = 1; j <= n2; j++) {
        // j-1 because we're using j to talk about the position... position
        // in java starts at 0
        R[j - 1] = A[q + j];
}

int i = 1;
int j = 1;

for (int k = p; k <= r; k++) {
        /**
         * This handles how the author does it where he puts infinity at the
         * end of the arrays to know when it's at the end of that array...
         * i'm doing the same thing but in a different way
         */
        if ((i - 1) >= L.length) {
                A[k] = R[j - 1];
                j = j + 1;
        }

        // Handles when it reaches the end of the array for array R
        else if ((j - 1) >= R.length) {
                A[k] = L[i - 1];
                i = i + 1;
        }

        /**
         * Both arrays are sorted into smallest to largest at this point...
         * combines the 2 arrays into one sorted array by comparing each
         * values of the arrays one by one
         */
        else if (L[i - 1] <= R[j - 1]) {
                A[k] = L[i - 1];
                i = i + 1;
        }

        else {
                A[k] = R[j - 1];
                j = j + 1;
```

```
                }

        }
}
/**
 * This is a Heap Sort that sorts an array from smallest to largest
 *
 * @param A is the array to be sorted
 * @param n is the terminal position of the array
 */
public static void HeapSort(int[] A, int n) {
        // Keep in mind that BuildMaxHeap is not in for-loop so it is only
        // called once.
        BuildMaxHeap(A, n);
        for (int i = n; i >= 1; i--) {
                int temp = A[0];
                A[0] = A[i];
                A[i] = temp;
                MaxHeapify(A, 0, i - 1);
        }
}

/**
 * This code is used for the HeapSort, it is only ran once in the beginning
 * to build the MaxHeap tree
 */
private static void BuildMaxHeap(int[] A, int n) {
        for (int i = (int) Math.floor((n - 1) / 2); i >= 0; i--) {
                MaxHeapify(A, i, n);
        }
}

// This code is used by the HeapSort
private static void MaxHeapify(int[] A, int i, int n) {
        // L = position left of the parent (remember that the tree starts at
        // position 1 for java)
        int L = (2 * i) + 1;
        // R = position right of the parent
        int R = (2 * i) + 2;
        int largest;

        // If the left child is larger than the parent...
        if ((L <= n) && (A[L] > A[i])) {
                largest = L;
        } else {
                largest = i;
```

```
		}

		// If the right child is larger than the left child and parent...
		if ((R <= n) && (A[R] > A[largest])) {
			largest = R;
		}

		// If the parent had at least one children that was larger, run
		if (largest != i) {
			int temp = A[i];
			A[i] = A[largest];
			A[largest] = temp;
			MaxHeapify(A, largest, n);
		}
	}
	/**
	 * @param A Takes in an array A which will be sorted
	 * @param p pivot position where the sorted value  will be placed
	 * @param r value which will be sorted
	 */
	public static void QuickSort(int[] A, int p, int r) {
		if (p < r) {
			int q = Partition(A, p, r);
			QuickSort(A, p, q - 1);
			QuickSort(A, q + 1, r);
		}

	}

	/**
	 *
	 * @param A Takes in an array A which will be sorted
	 * @param p pivot position where the sorted value will be placed
	 * @param r value which will be sorted
	 * @return an Array where r is sorted in its proper postition
	 */
	public static int Partition(int[] A, int p, int r) {
		int x = A[r];
		int i = p - 1;
		for (int j = p; j < r; j++) {
			// Goes into this statement if number stored in A[j] is less then x
			if (A[j] <= x) {
				// exchanging values

				i = i+1;
				int temp = A[i];
```

```
                        A[i] = A[j];
                        A[j] = temp;
                    }
                }
                // exchanges  A[i+1] with A[j] --- places pivot in sorted order.
                int temp2 = A[i + 1];
                A[i + 1] = A[r];
                A[r] = temp2;
                return i + 1;
        }

}
```