```java
package ordenacao;

interface Ordenador {
   void ordena(int[] A);
}

class HeapSort implements Ordenador {

   static void refazHeapMax(int A[], int i, int compHeap) {
      int esq, dir, maior, temp;

      esq = 2 * i + 1; // esquerda(i) => 2 * i + 1
      dir = 2 * i + 2; // direita (i) => 2 * i + 2

      if (esq < compHeap && A[esq] > A[i]) {
         maior = esq;
      } else {
         maior = i;
      }

      if (dir < compHeap && A[dir] > A[maior]) {
         maior = dir;
      }

      if (maior != i) {
         // trocar A[i] <==> A[maior]
         temp = A[i];
         A[i] = A[maior];
         A[maior] = temp;
         // Ajusta a posicao onde estava o maior
         refazHeapMax(A, maior, compHeap);
      }
   }

   static void constroiHeapMaxIterativo(int[] A) {
      int compHeap = A.length;
      for (int i = (A.length) / 2 - 1; i >= 0; i--) {
         refazHeapMax(A, i, compHeap);
      }
   }

   static void constroiHeapMaxRec(int[] A, int i) {
      int n = A.length;
      if (i < n/2){
         constroiHeapMaxRec(A, 2*i+1);
         constroiHeapMaxRec(A, 2*i+2);
         refazHeapMax(A, i, n);
      }
   }
```

```
   public void ordena(int A[]) {
      int i, compHeap, temp;
      // Constroi o heap maximo do arranjo todo
      compHeap = A.length;
      //constroiHeapMax(A);
      constroiHeapMaxRec(A,0);

      for (i = A.length - 1; i > 0; --i) {
         temp = A[0];
         A[0] = A[i];
         A[i] = temp;
         // Diminui o heap, pois A[i] esta posicionado
         compHeap--;
         refazHeapMax(A, 0, compHeap);
      }
   }
}
```

```java
class QuickSort implements Ordenador {

   public void ordena(int[] A) {
      quickSort(A, 0, A.length - 1);
   }

   private void quickSort(int[] A, int p, int r) {
      if (p < r) {
         int q = particao(A, p, r);
         quickSort(A, p, q - 1);
         quickSort(A, q + 1, r);
      }
   }

   protected int particao(int[] A, int p, int r) {
      int x, i, j, temp;

      x = A[r]; // pivo
      i = p - 1;

      for (j = p; j <= r - 1; ++j) {
         if (A[j] <= x) {
            i++;
            // trocar
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
         }
      }

      // reposicionar o pivo
      temp = A[i + 1];
      A[i + 1] = A[r];
      A[r] = temp;
      return (i + 1);
   }
}
```

```
class QuickSortAleatorio extends QuickSort {

   protected int particao(int[] A, int p, int r) {
      int i, temp;
      double f;

      // Escolhe um numero aleatorio entre p e r

      f = java.lang.Math.random(); // retorna um real f tal que 0 <= f < 1
      i = (int) (p + (r - p) * f);

      // i eh tal que p <= i < r

      // Troca de posicao A[i] e A[r]

      temp = A[r];
      A[r] = A[i];
      A[i] = temp;

      return super.particao(A, p, r);
   }
}

class MergeSort implements Ordenador {

   private void merge(int[] A, int p, int q, int r) {
      // A subsequencia A[p...q] esta ordenada
      // A subsequencia A[q+1...r] esta ordenada
      int i, j, k;

      // Faz copias das subsequencia seq1 = A[p...q] e seq2 = A[q+1...r]
      int tamseq1 = q - p + 1; // tamanho da subsequencia 1
      int tamseq2 = r - q; // tamanho da subsequencia 2

      int[] seq1 = new int[tamseq1];

      for (i = 0; i < seq1.length; ++i) {
         seq1[i] = A[p + i];
      }

      int[] seq2 = new int[tamseq2];

      for (j = 0; j < seq2.length; ++j) {
         seq2[j] = A[q + j + 1];
      }

      // Faz a juncao das duas subsequencias
      k = p;
      i = 0;
      j = 0;
```

```
      while (i < seq1.length && j < seq2.length) {
         // Pega o menor elemento das duas sequencias

         if (seq2[j] <= seq1[i]) {
            A[k] = seq2[j];
            j++;
         } else {
            A[k] = seq1[i];
            i++;
         }
         k++;
      }

      // Completa com a sequencia que ainda nao acabou

      while (i < seq1.length) {
         A[k] = seq1[i];
         k++;
         i++;
      }

      while (j < seq2.length) {
         A[k] = seq2[j];
         k++;
         j++;
      }

      // A subsequencia A[p...r] esta ordenada
   }

   private void mergeSort(int[] A, int p, int r) {
      if (p < r) {
         int q = (p + r) / 2;
         mergeSort(A, p, q);
         mergeSort(A, q + 1, r);
         merge(A, p, q, r);
      }
   }

   public void ordena(int[] A) {
      mergeSort(A, 0, A.length - 1);
   }
}
```

```java
class SelectionSort implements Ordenador {

   public void ordena(int[] A) {
      int i, j, indiceDoMinimo, temp;
      int fim = A.length;

      for (i = 0; i < fim - 1; i++) {

         indiceDoMinimo = i;
         for (j = i + 1; j < fim; j++) {
            if (A[j] < A[indiceDoMinimo]) {
               indiceDoMinimo = j;
            }
         }

         temp = A[i];
         A[i] = A[indiceDoMinimo];
         A[indiceDoMinimo] = temp;
      }
   }
}
```

```
class InsertionSort implements Ordenador {

   public void ordena(int[] A) {
      int i, j, v;
      int fim = A.length;

      for (i = 1; i < fim; i++) {
         v = A[i];
         j = i;
         while ((j > 0) && (A[j - 1] > v)) {
            A[j] = A[j - 1];
            j = j - 1;
         }
         A[j] = v;
      }
   }
}
```

```java
class BubbleSort implements Ordenador {

   public void ordena(int[] A) {
      int i, j, temp;
      int fim = A.length;

      for (i = fim - 1; i > 0; i--) {

         for (j = 1; j <= i; j++) {
            if (A[j - 1] > A[j]) {
               // Troca os dois de lugar
               temp = A[j - 1];
               A[j - 1] = A[j];
               A[j] = temp;
            }
         }
      }
   }
}
```