```java
/**
 * @author Amar Bessedik
 * This class implements the heapSort algorithm. It is adapted to sort an array
 * of n Edges in increasing order of weight.
 */
public class HeapSort
{
   /**
    * Constructor
    */
   public HeapSort(){}



   /**
    * @param E and array of graph edges
    * @param n length of the array E
    */
   public void heapSort(Edge[] E, int n)
   {
      //Build a heap where root is the largest
      buildHeap(E, n);
      //Extract edges from heap one by one
      for (int i = n - 1; i >= 0; i--)
      {
         //Swap root (largest weight) to the end of array
         swap(E, 0, i);
         //Rebuild the reduced heap.
         heapify(E, 0, i);
      }
   }//end heapSort

   /**
    * @param E array of a graph edges
    * @param n length of the array E
    */
   private void buildHeap(Edge[] E, int n)
   {
      for (int i = n/2; i >= 0; i--)
      {
         heapify(E, i, n);
      }
   }//end heapify

```

```
45    /**
46     * @param E array of a graph edges
47     * @param i index from which start make a heap.
48     * @param n length of the array E.
49     */
50    private void heapify(Edge[] E, int i, int n)
51    {
52       int largest = i;// Initialize largest as i
53       int l = 2*i + 1; //left child
54       int r = 2*i + 2; //right child
55
56       // If left child's weight is greater than root's weight
57       if (l < n && E[l].getWeight() > E[largest].getWeight())
58       {
59          largest = l;
60       }
61       // If right child's weight is greater than root's weight
62       if (r < n && E[r].getWeight() > E[largest].getWeight())
63       {
64          largest = r;
65       }
66       // If largest is not i
67       if (largest != i)
68       {
69          //swap i with largest
70          swap(E, i, largest);
71          //heapify the reduced heap recursively
72          heapify(E, largest, n);
73       }
74    }//end heapify
75
76   //=========== HELPER METHODS ===============
77    /**
78     * @param E array of a graph edges
79     * @param i the edge at index i goes to index j.
80     * @param j the edge at index j goes to index i.
81     */
82    private void swap(Edge[] E, int i, int j)
83    {
84       Edge tmp = E[i];
85       E[i] = E[j];
86       E[j] = tmp;
87    }//end swap
88 }//end class
89
```