```
##########################################

Student Name = Jitong Ding

##########################################



CSE017 Grading sheet for Jitong Ding

Homework Assignment MergeSortComparison

Total points                          maximum: 100

Completeness: All class/methods included (40)   [  40  ]

Compilation: Program compiles (20)               [  20  ]

Execution: Program executes properly (30)        [  30  ]

Style: Program obeys style rules (10)            [  10  ]

Subtotal                                            100

Late Penalty                                     [   0  ]

Total Points                                        100


##########################################
##########################################
```

```
/*
CSE 17
Jitong Ding
jid221
Homework #5 DEADLINE: November 29, 2018
Program: Merge Sort Comparison
*/

import java.util.Random;

public class MergeSortComparison{

  /** Private data filed */
  private static Comparable[] tempList;
  private static long first;
  private static long second;

  /** A generic sort method to sort a list using merge sort. */
  public static <E extends Comparable<E>> void mergeSort(E[] list){
    if(list.length>1){
      /** merge sort the first half of the list */
      E[] listFirst = (E[])new Comparable[list.length/2];
      System.arraycopy(list,0,listFirst,0,list.length/2);
      mergeSort(listFirst);
      /** merge sort the second half of the list */
      int secondHalfLength = list.length - list.length/2;
      E[] listSecond = (E[]) new Comparable[secondHalfLength];
      System.arraycopy(list,list.length/2, listSecond,0,secondHalfLength);
      mergeSort(listSecond);
      /** merge the two halves lists */
      merge(listFirst,listSecond,list);
    }
}
  /** The recursive helper generic method for the merge. */
  public static <E extends Comparable<E>> void merge(E[] list1, E[] list2, E[] t
emp){
    int index1 = 0; /** index int the list1*/
    int index2 = 0; /** index int the list2*/
    int index3 = 0; /** index int the temp*/

    /** as long as neither index is at the end, compare them and copy the smalle
r value to temp */
    while(index1 < list1.length && index2 < list2.length){
      if(list1[index1].compareTo(list2[index2])<0){
        temp[index3++] = list1[index1++];
    }
      else{
        temp[index3++] = list2[index2++];
  }
    }
    /** copy remaining values from list1 to temp. */
    while(index1 < list1.length){
      temp[index3++] = list1[index1++];
    }
    /** copy remaining values from list1 to temp. */
    while(index2 < list2.length){
      temp[index3++] = list2[index2++];
    }
}
  /** A recursive genric sort method for newMergeSort method*/
  public static <E extends Comparable<E>> void newMergeSort(E[] list){
    tempList = new Comparable[list.length];
    newMergeSort(list, 0, list.length-1);
  }

  /** A recursive helper genric sort method for newMergeSort method*/
  public static <E extends Comparable<E>> void newMergeSort(E[] list, int startP
oint , int endPoint){
    if(endPoint-startPoint > 0){
        newMergeSort(list, startPoint,(startPoint+endPoint)/2);
        newMergeSort(list, (startPoint+endPoint)/2+1,endPoint);
        newMerge(list,startPoint,(startPoint+endPoint)/2+1,endPoint);
    }
```



```
  }

  /** The recursive helper generic method for the merge. */
  public static <E extends Comparable<E>> void newMerge(E[] list, int start, int
 mid, int end){
    int index3 = 0;
    int index2 = mid;
    int index1 = start;

    while(start < index2 && mid <end+1){
      if(list[start].compareTo(list[mid])<0){
        tempList[index3++] = list[start++];
      }
      else{
        tempList[index3++] = list[mid++];
      }
    }
    while(start < index2){
      tempList[index3++] = list[start++];
    }
    while(mid < end+1){
      tempList[index3++] = list[mid++];
    }
    System.arraycopy(tempList,0,list, index1, (end-index1)+1);
  }
  /** A method to return the time difference of the first method and second meth
od. */
  public static <E extends Comparable<E>> double getTime(E[] list1, E[] list2){
    long start = System.currentTimeMillis();
    mergeSort(list1);
    long mid = System.currentTimeMillis();
    first =mid - start;
    newMergeSort(list2);
    long end = System.currentTimeMillis();
    second = end - mid;
    return ((double)first - second);
  }

   /** The main method. */
  public static void main(String[] args){
    Random rand = new Random();
    /** A while loop for the Integer array for four different sizes. */
    int i = 0;
    System.out.println("Integers:");
    System.out.println("Test Size    Orig      New   Improve(%)");
    while(i<4){
      int size;
      double time;
      if(i==0){
        Integer[] i1 = new Integer[100000];
        for(int j =0; j< 100000;++j){
          i1[j] = rand.nextInt();
        }
        size = 100000;
        Integer[] i5 = new Integer[size];
        System.arraycopy(i1,0,i5,0,size);
        time = getTime(i1,i5);
    }
      else if(i == 1){
        Integer[] i2 = new Integer[500000];
        for(int j =0; j< 500000;++j){
          i2[j] = rand.nextInt();
        }
        size = 500000;
        Integer[] i5 = new Integer[size];
        System.arraycopy(i2,0,i5,0,size);
        time = getTime(i2,i5) ;
    }
      else if(i == 2){
        Integer[] i3 = new Integer[1000000];
        for(int j =0; j< 1000000;++j){
          i3[j] = rand.nextInt();
        }
```

```
            size = 1000000;
            Integer[] i5 = new Integer[size];
            System.arraycopy(i3,0,i5,0,size);
            time = getTime(i3,i5);
        }
            else{
            Integer[] i4 = new Integer[5000000];
            for(int j =0; j< 5000000;++j){
              i4[j] = rand.nextInt();
            }
            size = 5000000;
            Integer[] i5 = new Integer[size];
            System.arraycopy(i4,0,i5,0,size);
            time = getTime(i4,i5);
        }
            System.out.printf("%9d%9s%10s%11.1f%%\n",size,first,second,(time/first)*100
    );
            i++;
    }
        /** A while loop for the Double array for four different sizes. */
        int j = 0;
        System.out.println("Doubles:");
        System.out.println("Test Size    Orig      New   Improve(%)");
        while(j<4){
          int size;
          double time;
          if(j==0){
            Double[] d1 = new Double[100000];
            for(int m =0; m< 100000;++m){
              d1[m] = rand.nextDouble();
            }
            size = 100000;
            Double[] d5 = new Double[size];
            System.arraycopy(d1,0,d5,0,size);
            time = getTime(d1,d5);
         }
          else if(j == 1){
            Double[] d2 = new Double[500000];
            for(int m =0; m< 500000;++m){
              d2[m] = rand.nextDouble();
            }
            size = 500000;
            Double[] d5 = new Double[size];
            System.arraycopy(d2,0,d5,0,size);
            time = getTime(d2,d5) ;
        }
            else if(j == 2){
            Double[] d3 = new Double[1000000];
            for(int m =0; m< 1000000;++m){
              d3[m] = rand.nextDouble();
            }
            size = 1000000;
            Double[] d5 = new Double[size];
            System.arraycopy(d3,0,d5,0,size);
            time = getTime(d3,d5);
        }
            else{
            Double[] d4 = new Double[5000000];
            for(int m =0; m< 5000000;++m){
              d4[m] = rand.nextDouble();
            }
            size = 5000000;
            Double[] d5 = new Double[size];
            System.arraycopy(d4,0,d5,0,size);
            time = getTime(d4,d5);
        }
            System.out.printf("%9d%9s%10s%11.1f%%\n",size,first,second,(time/first)*100
    );
            j++;
    }
    }
    }
```

```
##########################################
##########################################


############# Compiled Result ###############


Source Code Compilation:

MergeSortComparison.java:14: warning: [rawtypes] found raw type: Comparable
  private static Comparable[] tempList;
                 ^
  missing type arguments for generic class Comparable<T>
  where T is a type-variable:
    T extends Object declared in interface Comparable
MergeSortComparison.java:22: warning: [rawtypes] found raw type: Comparable
      E[] listFirst = (E[])new Comparable[list.length/2];
                           ^
  missing type arguments for generic class Comparable<T>
  where T is a type-variable:
    T extends Object declared in interface Comparable
MergeSortComparison.java:22: warning: [unchecked] unchecked cast
      E[] listFirst = (E[])new Comparable[list.length/2];
                          ^
  required: E[]
  found:    Comparable[]
  where E is a type-variable:
    E extends Comparable<E> declared in method <E>mergeSort(E[])
MergeSortComparison.java:27: warning: [rawtypes] found raw type: Comparable
      E[] listSecond = (E[]) new Comparable[secondHalfLength];
                               ^
  missing type arguments for generic class Comparable<T>
  where T is a type-variable:
    T extends Object declared in interface Comparable
MergeSortComparison.java:27: warning: [unchecked] unchecked cast
      E[] listSecond = (E[]) new Comparable[secondHalfLength];
                             ^
  required: E[]
  found:    Comparable[]
  where E is a type-variable:
    E extends Comparable<E> declared in method <E>mergeSort(E[])
MergeSortComparison.java:60: warning: [rawtypes] found raw type: Comparable
    tempList = new Comparable[list.length];
                   ^
  missing type arguments for generic class Comparable<T>
  where T is a type-variable:
    T extends Object declared in interface Comparable
6 warnings



##########################################


##########################################
############ Execution Result ############
##########################################


Original Program

Integers:
Test Size      Orig       New    Improve(%)
   100000       100       101        -1.0 %
   500000       357       254        28.9 %
  1000000       832       742        10.8 %
  5000000      5588      3849        31.1 %
```



```
Doubles:
Test Size      Orig       New    Improve(%)
   100000        82        49        40.2 %
   500000       348       280        19.5 %
  1000000       943       616        34.7 %
  5000000      4500      4383         2.6 %


##########################################

Test Program

Test 1:          [83, 73, 56, 57, 52, 15, 37, 28, 72, 21, 9, 58, 75, 93, 82]
Expected:        [9, 15, 21, 28, 37, 52, 56, 57, 58, 72, 73, 75, 82, 83, 93]
Original:        [9, 15, 21, 28, 37, 52, 56, 57, 58, 72, 73, 75, 82, 83, 93]
New:             [9, 15, 21, 28, 37, 52, 56, 57, 58, 72, 73, 75, 82, 83, 93]

Test 2:          [42]
Original:        [42]
New:             [42]

Test 3:          [g, c, e, a, h, i, b, f, d, j]
Original:        [a, b, c, d, e, f, g, h, i, j]
New:             [a, b, c, d, e, f, g, h, i, j]
```

Output is good