Mohamed Salah Abd-Allah

Sec_3

Task_2_ algorithm

Test on 1000 sized array

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL

```
4378018 1158110642 1165025640 1165306025 1165343250 1166015436 1166290799 1168736729 1169767415 1179103873 1184926246 1189027194 11
1200788918 1204202428 1204683280 1227482174 1228082298 1229244152 1229422349 1229795539 1247153106 1252038914 1252867356 125304873
07 1259062261 1259901763 1267972262 1268271557 1268865834 1286741184 1314656599 1321804025 1322014488 1329294834 1335045042 133713
24171 1346500731 1360418750 1364089133 1371794699 1373246183 1375938525 1381367957 1382507049 1382919727 1388028541 1389833466 139
99030400 1399594146 1401229800 1406183445 1406470479 1407462200 1415977861 1419108867 1419217689 1424070190 1430816803 1457780537 1
 1463587359 1468170736 1473836329 1476419873 1478012324 1490057371 1493167185 1496944722 1498212319 1498263348 1502651437 150312561
978 1509065687 1511517823 1511627974 1515534216 1516026116 1532849275 1538548816 1544509356 1549134866 1552317006 1554844319 155587
542624 1560763218 1562242212 1564065844 1567259292 1569648677 1570281163 1570632810 1570887015 1577259801 1579190077 1579366437 158
593410289 1593688972 1594614991 1600625972 1602183027 1603893742 1616859854 1618288619 1633195929 1638067433 1638874491 1645281352
9 1681359451 1683050893 1698856116 1701957394 1703456068 1705251065 1715635549 1717380650 1723718069 1725580076 1728180137 1729493
3912 1737603238 1741938658 1748050924 1749707348 1751012692 1755647685 1764108881 1769531094 1771808251 1773332742 1774037674 17757
9065835 1780657909 1782614348 1782897188 1784743687 1786962814 1787973431 1789932593 1796966588 1801959825 1806914751 1807975238 18
1811574332 1816630611 1821197791 1825910580 1826058165 1830524756 1836578787 1840034088 1845210135 1850795986 1858608609 187466795
88 1881691220 1889791957 1890852474 1892872598 1893540888 1905478351 1907646464 1909172018 1915407085 1915860664 1927041937 193298
62459 1945050139 1945627346 1949789125 1950503734 1952257473 1960905251 1962031202 1966950833 1967457738 1970227653 1970334771 1975
96198651 2000133924 2004982931 2007127611 2008170087 2017142698 2017469881 2023771643 2029352612 2031165501 2032154242 2038926603 2
 2043712229 2044124014 2045966947 2052552492 2055683587 2059045451 2073121177 2076190844 2080529222 2080541238 2091919073 209304681
318 2099110000 2100148643 2102335515 2104823342 2111617640 2113160867 2120161879 2126712231 2144640286

 Time elapsed for insertion algoritm is: 8.000000

 Time elapsed for merge algoritm is: 1.000000

 insertion counter: 259261

 merge counter: 8722
PS G:\JAVA 2022>
```

Merge sort algorithm

main.java 9 | Merge_sort.java 1 | global_counter.java | Insertion_sort.java 1

Algorithms_Mohamed_Salah > Merge_sort.java > Merge_sort > mergeSort(int[], int)

```java
package Algorithms_Mohamed_Salah;



public class Merge_sort {
 global_counter mergeCounter= new global_counter();

    public static void merge(int[] left_arr,int[] right_arr, int[] arr,int left_size, int right_size){

        int i=0,l=0,r = 0;
        //The while loops check the conditions for merging
        while(l<left_size && r<right_size){

            if(left_arr[l]<right_arr[r]){
                arr[i++] = left_arr[l++];
            }
            else{
                arr[i++] = right_arr[r++];
            }
        }
        while(l<left_size){
            arr[i++] = left_arr[l++];
        }
        while(r<right_size){
          arr[i++] = right_arr[r++];
        }
    }



    public  void mergeSort(int [] arr, int len){
        int counter=0;

        if (len < 2){return;}

        int mid = len / 2;
        int [] left_arr = new int[mid];
        int [] right_arr = new int[len-mid];

      //Dividing array into two and copying into two separate arrays
        int k = 0;
        for(int i = 0;i<len;++i){
            counter++;
            mergeCounter.setMerge_counter(counter);

            if(i<mid){
                left_arr[i] = arr[i];
            }
            else{
                right_arr[k] = arr[i];
                k = k+1;
            }
        }
      // Recursively calling the function to divide the subarrays further
        mergeSort(left_arr,mid);
        mergeSort(right_arr,len-mid);
      // Calling the merge method on each subdivision
        merge(left_arr,right_arr,arr,mid,len-mid);
    }
  /* ////for testing//////
    public static void main( String args[] ) {
          int [] array = {12,1,10,50,5,15,45};
          mergeSort(array,array.length);
          for(int i =0; i< array.length;++i){
              System.out.print(array[i]+ " ");
          }
      }*/
  }
```

main.java 9 | Merge_sort.java 1 | global_counter.java | Insertion_sort.java 1

Algorithms_Mohamed_Salah > Insertion_sort.java > Insertion_sort > Insertion(int[])

Insertion sort algorithm

```java
package Algorithms_Mohamed_Salah;

public class Insertion_sort {

  global_counter insertCounter= new global_counter();
    public  void Insertion(int [] rand_arr){
      int insert_Counter=0;
        for(int i=0;i<rand_arr.length;++i){

          int j = i;

          while(j > 0 && rand_arr[j-1]>rand_arr[j]){
            insert_Counter++;
            insertCounter.setInsertion_counter(insert_Counter);
            int key = rand_arr[j];
            rand_arr[j] = rand_arr[j-1];
            rand_arr[j-1] = key;
            j = j-1;


          }
        }
    }
/********* for test********
      public static void main( String args[] ) {
          int [] arr = {5,2,12,12,1};
          Insertion(arr);

          for(int i=0;i<arr.length;++i){
            System.out.print(arr[i] + " ");
          }

          }
          */
}
```

main.java 9 | Merge_sort.java 1 | global_counter.java | Insertion_sort.java 1

Algorithms_Mohamed_Salah > global_counter.java > global_counter

Global counter implementation

```java
package Algorithms_Mohamed_Salah;

public  class global_counter{
    public static int merge_counter;
    public static void setMerge_counter(int merge_counter) {
        global_counter.merge_counter = merge_counter;
    }
    public static int getMerge_counter() {
        return merge_counter;
    }
    public  static int insertion_counter;

    public static void setInsertion_counter(int insertion_counter) {
        global_counter.insertion_counter = insertion_counter;
    }
    public static int getInsertion_counter() {
        return insertion_counter;
    }
}
```

main.java 9 | Merge_sort.java 1 | global_counter.java | Insertion_sort.java 1

Algorithms_Mohamed_Salah > main.java > main > main(String[])

Main project

```java
package Algorithms_Mohamed_Salah;
import java.util.Scanner;

import Algorithms_Mohamed_Salah.Insertion_sort;
import Algorithms_Mohamed_Salah.Merge_sort;
import Algorithms_Mohamed_Salah.random_arr_gen;
public class main {



    Run | Debug
    public static void main(String[] args) {
global_counter Counter= new global_counter();

        Scanner xScanner=new Scanner(System.in);
        System.out.println(x: "Enter the size of the random array you need to test on: ");
        int arr_size= xScanner.nextInt();

        Merge_sort mSort= new Merge_sort();
        Insertion_sort iSort= new Insertion_sort();
        random_arr_gen rArr_gen= new random_arr_gen();


        System.out.printf(format: "%n The random array to test the insertion sort: %n");
        int[] arr_insertion=rArr_gen.random_arr(arr_size);
        long startTime_insertion = System.nanoTime(); // start a timer to  get time of the insertion algoritm
        iSort.Insertion(arr_insertion);
        double elapsedTime_insertion = (System.nanoTime() - startTime_insertion)/1000000;// calculating the elapsed time for the insertion algorit
         System.out.println(x: "Insertion sort: ");
         for(int i=0;i<arr_insertion.length;++i){
            System.out.print(arr_insertion[i] + " ");
          }

         System.out.printf(format: "%n%n The random array to test the merge sort: %n");
         int[] arr_merge=rArr_gen.random_arr(arr_size);
         long startTime_merge = System.nanoTime(); // start a timer to  get time of the merge algoritm
         mSort.mergeSort(arr_merge,arr_size);
         double elapsedTime_merge = (System.nanoTime() - startTime_merge)/1000000;// calculating the elapsed time for the merging algorithm

         System.out.println(x: "Merge sort: ");
         for(int i=0;i<arr_merge.length;++i){
            System.out.print(arr_merge[i] + " ");
          }


          System.out.printf(format: "%n %n Time elapsed for insertion algoritm is: %f",elapsedTime_insertion);
          System.out.printf(format: "%n %n Time elapsed for merge algoritm is: %f",elapsedTime_merge);
          System.out.printf(format: "%n %n insertion counter: %d",Counter.getInsertion_counter());
          System.out.printf(format: "%n %n merge counter: %d",Counter.getMerge_counter());
```