Program Structures and Algorithms
Spring 2023(SEC –8)

NAME: Junlong Qiao
NUID: 002784609

Task: Assignment 6(Hits as time predictor)

Determine the best predictor: that will mean the graph of the appropriate observation will match the graph of the timings most closely.

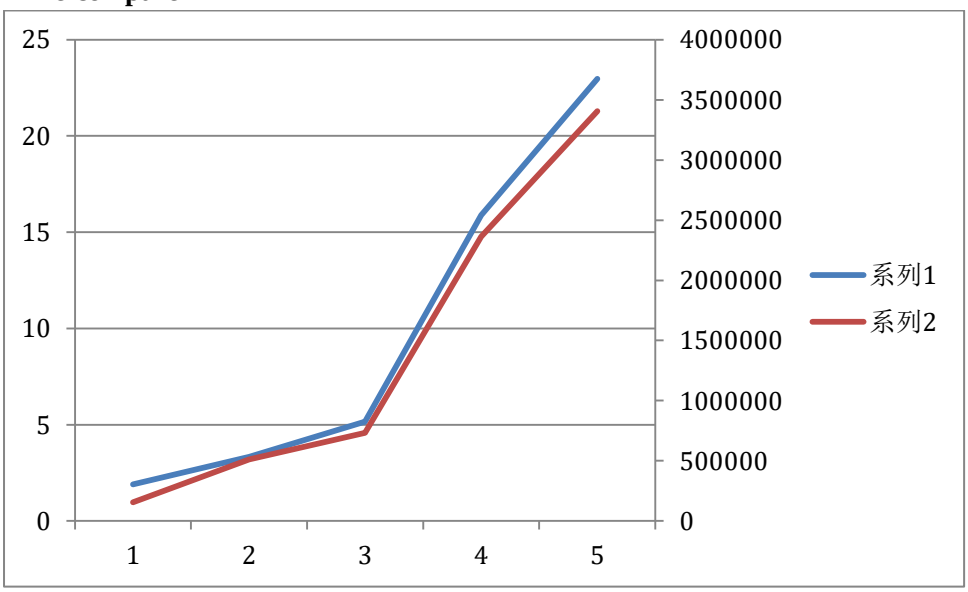
Runtie Relationship Conclusion:

Compare is the best predictor. Because in three sort methods, compare match time best.

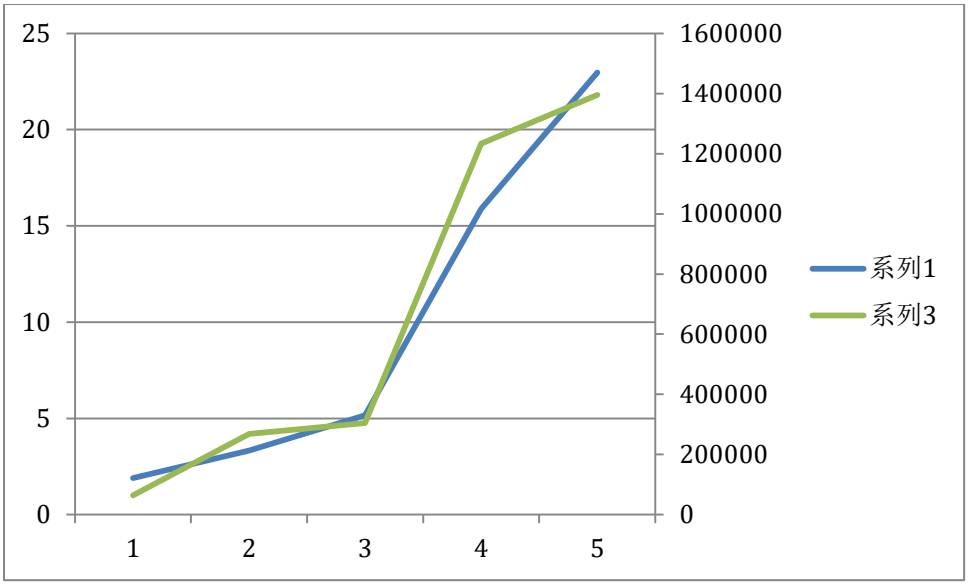
QuickSort_DualPivot:

arraylength	compares	swaps	copies	hits	time
10000	154776.05	64504.74	0	415552	1.91
20000	510764.96	268429	0	2095246	3.33
40000	732222.89	304685.38	0	1962052	5.16
80000	2362945.39	1233591.28	0	9660256	15.89
160000	3407063.83	1395780.93	0	9034389	22.96

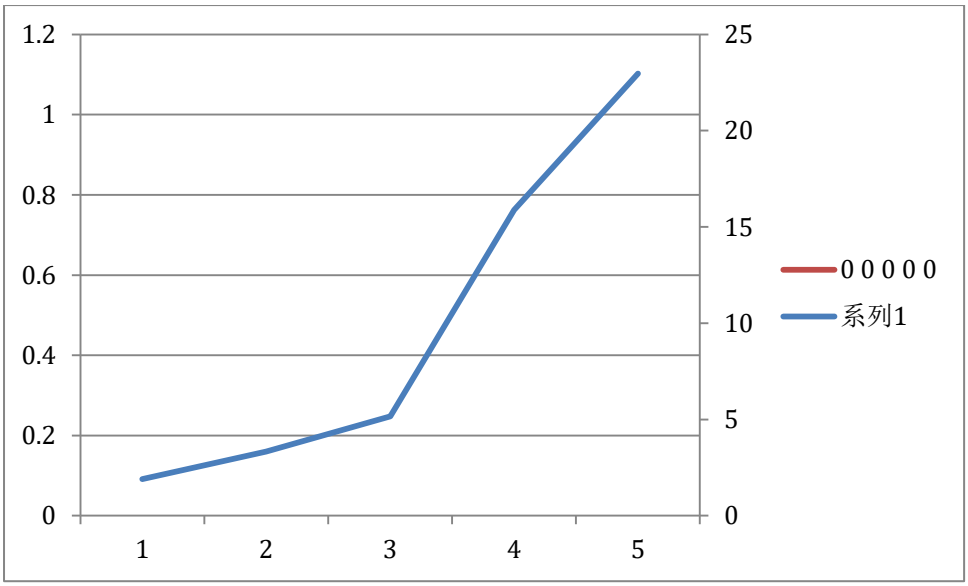
Time-compare



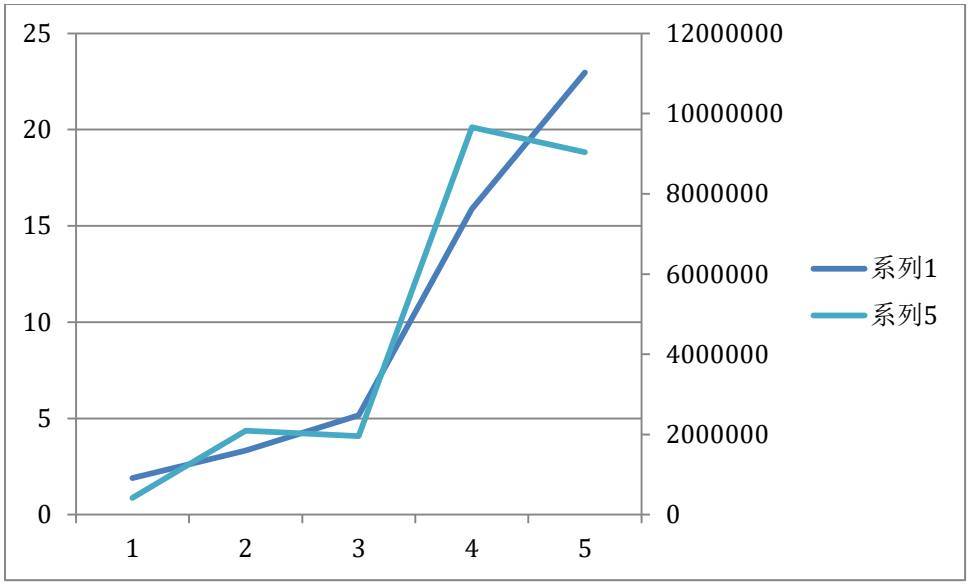
Time-swap



Time-copy



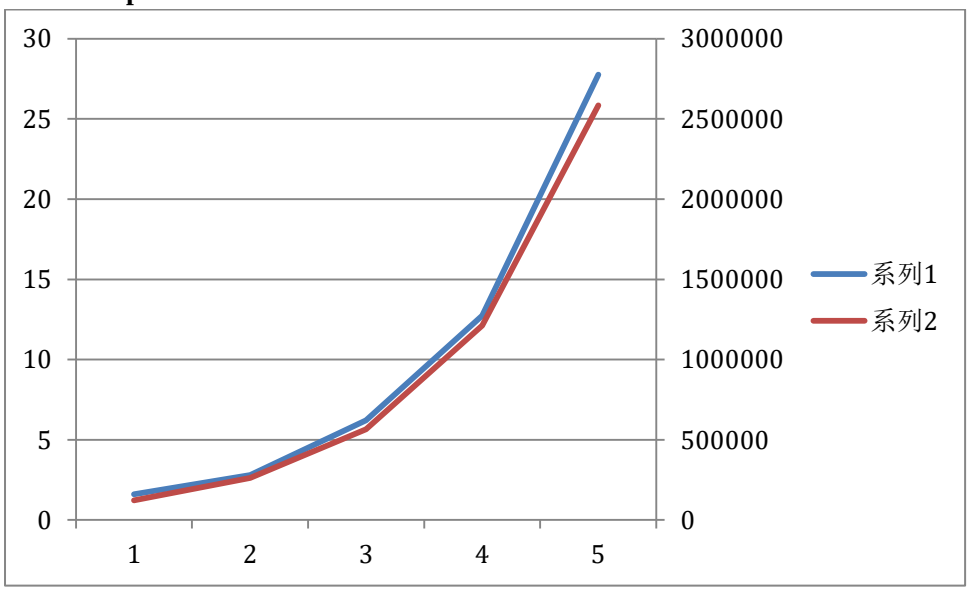
Time-hit



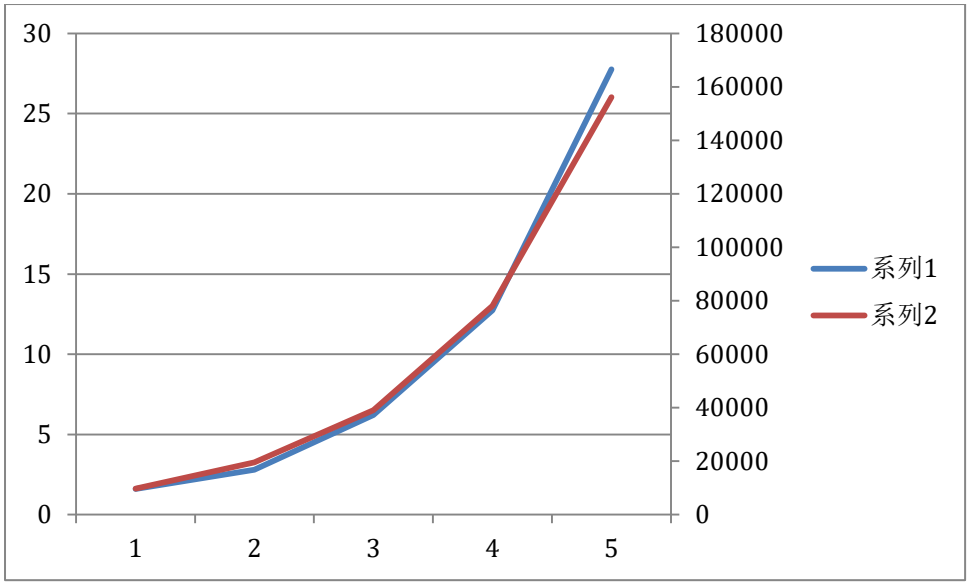
MergeSort:

arraylength	compares	swaps	copies	hits	time
10000	121497.84	9751.39	110000	489763	1.61
20000	263017.4	19527.64	240000	1059611	2.81
40000	566022.57	39059.46	520000	2279202	6.22
80000	1212024.43	78073.95	1120000	4878257	12.74
160000	2584031.37	156184.78	2400000	1E+07	27.75

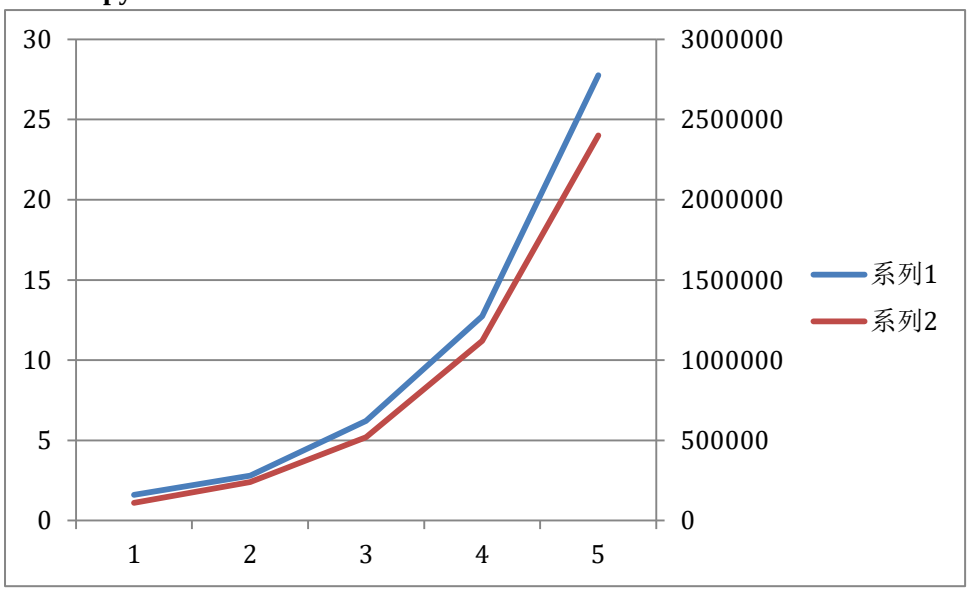
Time-compare:



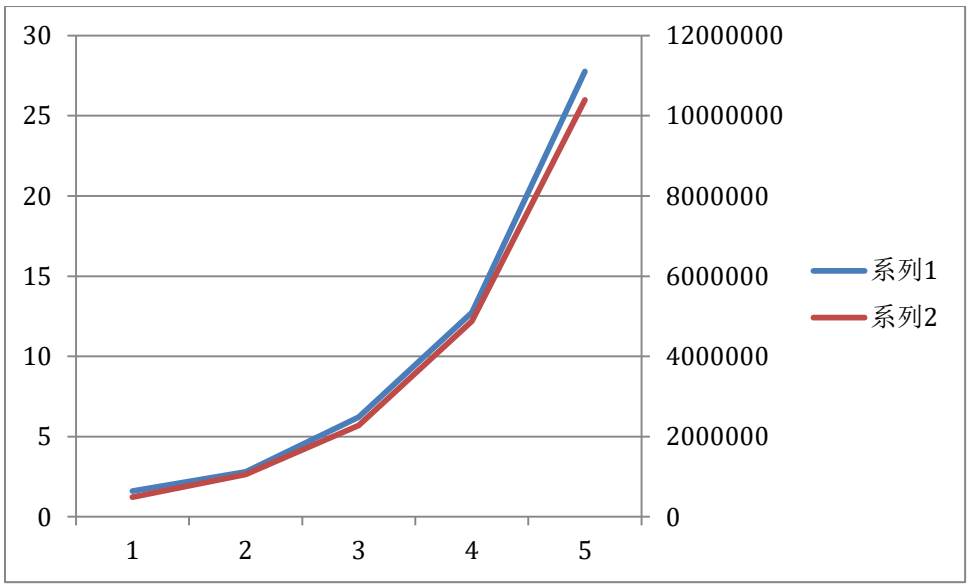
Time-swap:



Time-copy:



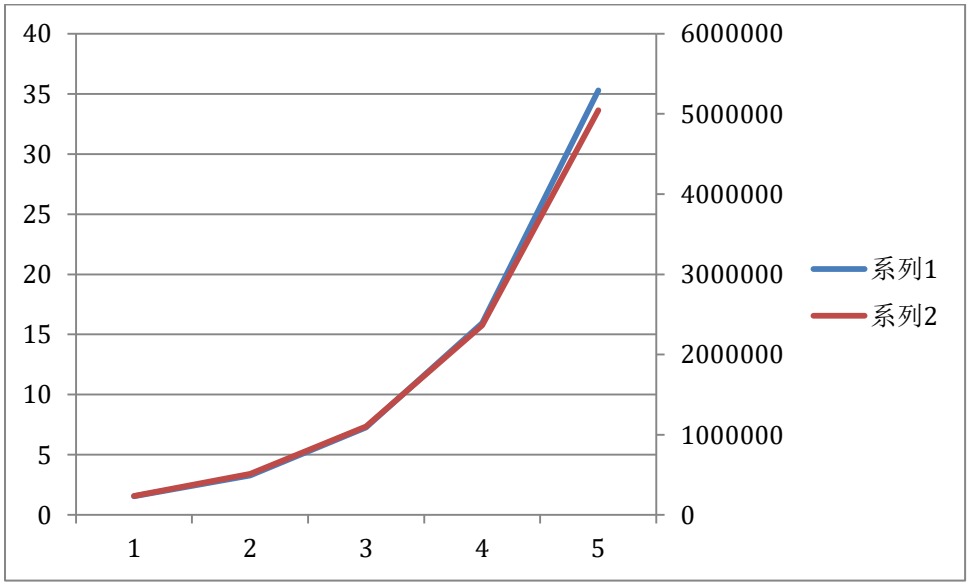
Time-hit



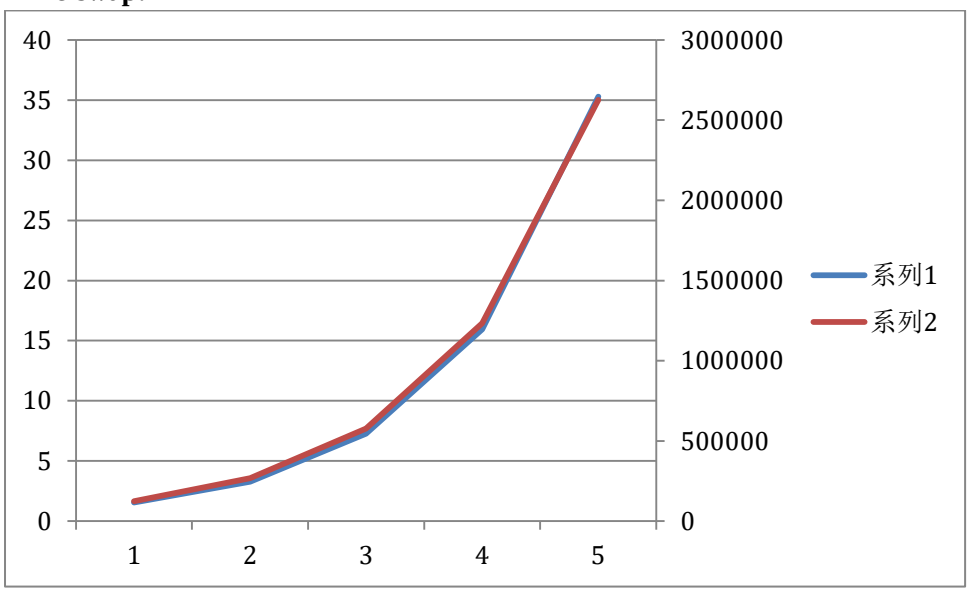
HeapSort:

arraylength	compares	swaps	copies	hits	time
10000	235367	124199.61	0	967532	1.54
20000	510764.96	268429	0	2095246	3.29
40000	1101477.97	576790.87	0	4510119	7.28
80000	2362945.39	1233591.28	0	9660256	15.95
160000	5045942.63	2627179.33	0	2.1E+07	35.27

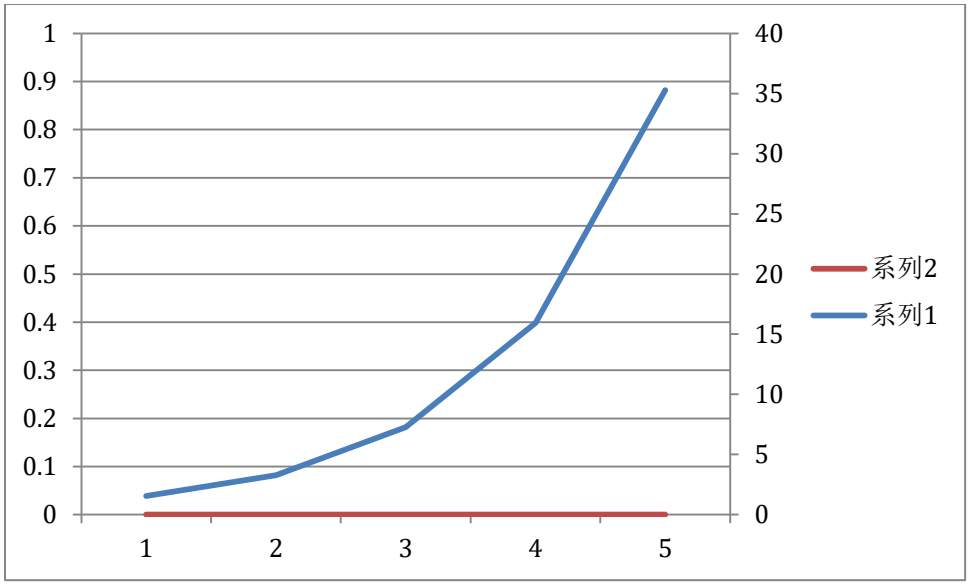
Time-compare:



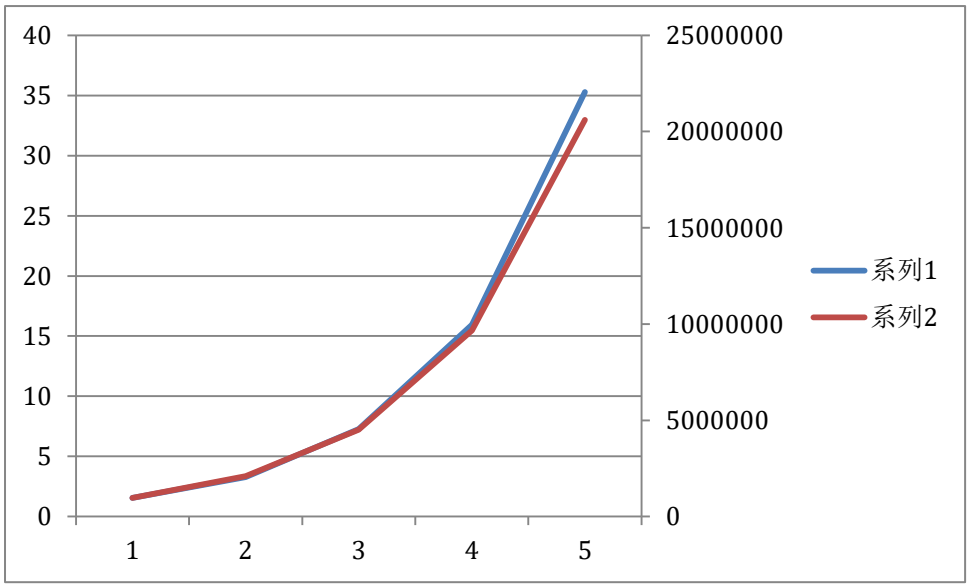
Time-swap:



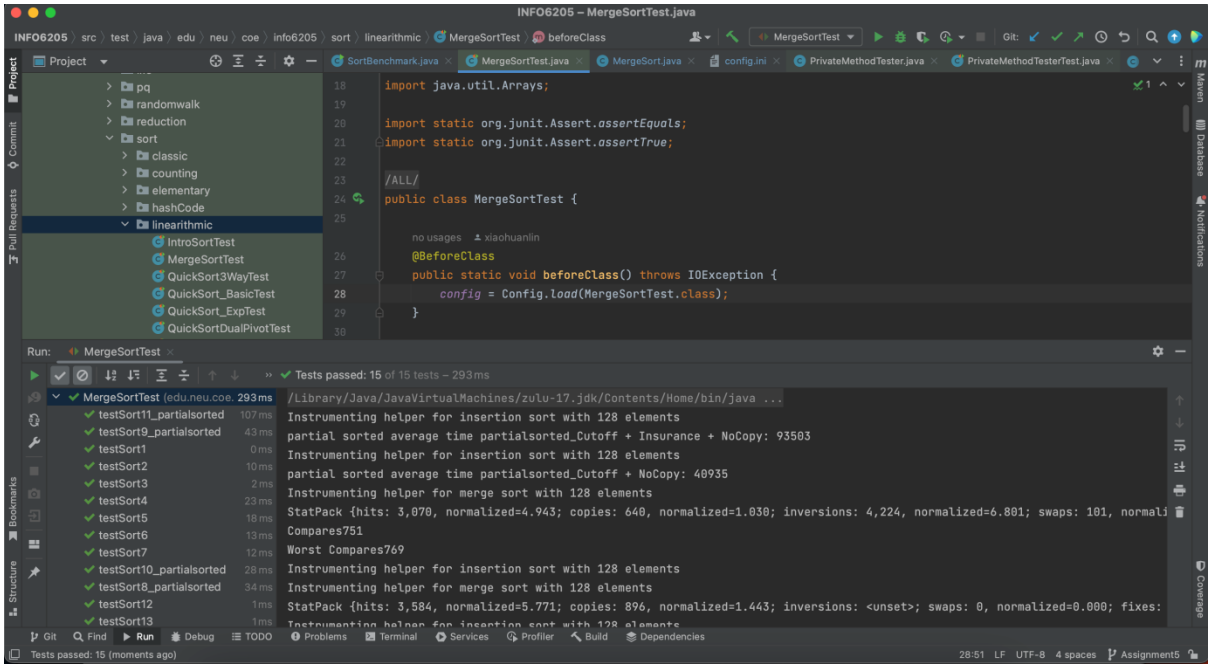
Time-copy:

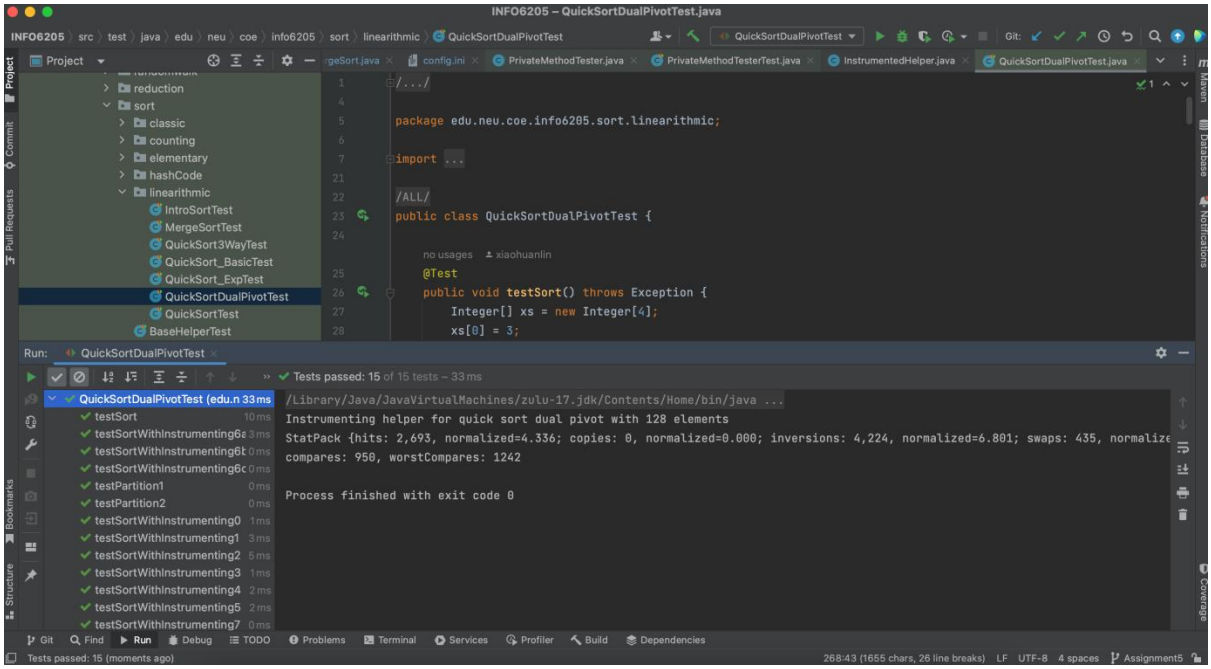


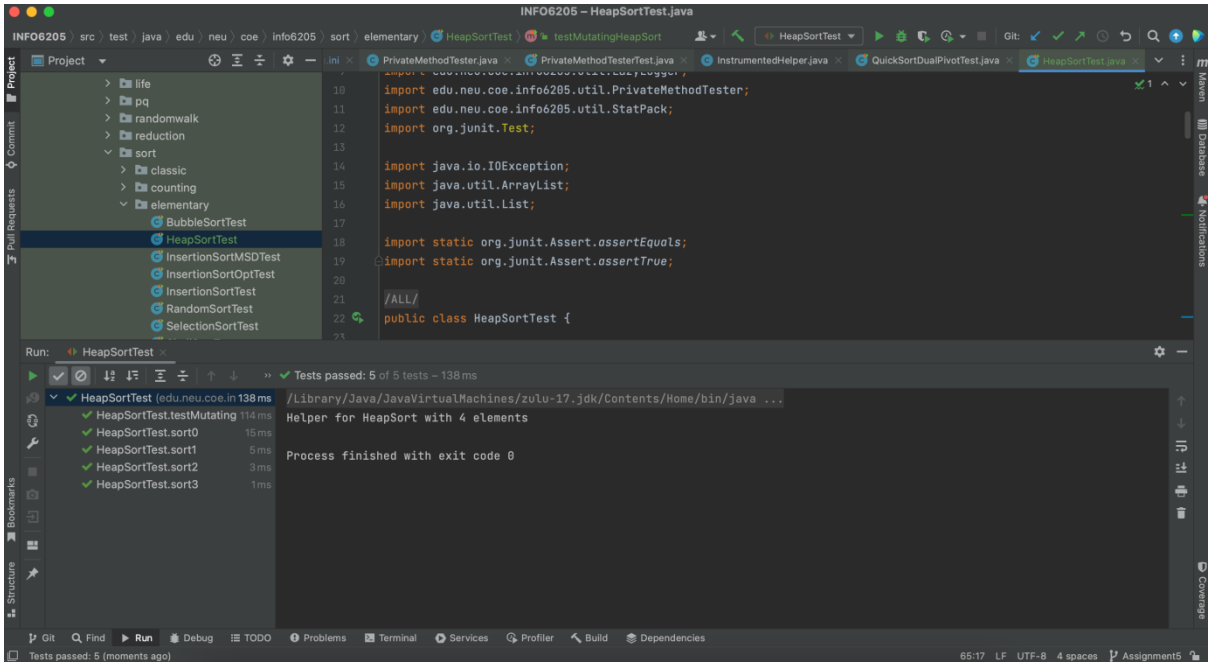
Time-hit



Unit Test Screenshots:


INFO6205 – MergeSortTest.java
import java.util.Arrays;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
/ALL/
public class MergeSortTest {
no usages xiaohuanlin
@BeforeClass
public static void beforeClass() throws IOException {
config = Config.load(MergeSortTest.class);
}
Tests passed: 15 of 15 tests – 293 ms
/Library/Java/JavaVirtualMachines/zulu-17.jdk/Contents/Home/bin/java ...
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance + NoCopy: 93503
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + NoCopy: 40935
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 3,070, normalized=4.943; copies: 640, normalized=1.030; inversions: 4,224, normalized=6.801; swaps: 101, normali
Compares751
Worst Compares769
Instrumenting helper for insertion sort with 128 elements
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 3,584, normalized=5.771; copies: 896, normalized=1.443; inversions: <unset>; swaps: 0, normalized=0.000; fixes:


INFO6205 – QuickSortDualPivotTest.java
/.../
package edu.neu.coe.info6205.sort.linearithmic;
import ...
/ALL/
public class QuickSortDualPivotTest {
no usages xiaohuanlin
@Test
public void testSort() throws Exception {
Integer[] xs = new Integer[4];
xs[0] = 3;
Tests passed: 15 of 15 tests – 33 ms
/Library/Java/JavaVirtualMachines/zulu-17.jdk/Contents/Home/bin/java ...
Instrumenting helper for quick sort dual pivot with 128 elements
StatPack {hits: 2,693, normalized=4.336; copies: 0, normalized=0.000; inversions: 4,224, normalized=6.801; swaps: 435, normalize
compares: 950, worstCompares: 1242
Process finished with exit code 0



Code:

SortBechmark.java

```
    public static void main(String[] args) throws IOException {
        Config config = Config.load(SortBenchmark.class);

        int size = 10000;
        int runs = 100;
        String des;
        int type = 3;
        while(size<256000){
            if(type==1){
                des = "QuickSort_DualPivot";
                Helper<Integer> helper = HelperFactory.create(des,size,true,config);
                SortWithHelper<Integer> sort = new QuickSort_DualPivot<>(helper);
                Integer[] ints = helper.random(Integer.class, r -> r.nextInt());
                SorterBenchmark sorterBenchmark = new SorterBenchmark(Integer.class, sort,
ints, runs, timeLoggersLinearithmic);
                sorterBenchmark.run(size);


System.out.println(((InstrumentedHelper)sort.getHelper()).getStatPack().mean("compares")
+ " , " + ((InstrumentedHelper)sort.getHelper()).getStatPack().mean("swaps") + " , "
+((InstrumentedHelper)sort.getHelper()).getStatPack().mean("copies") + " , " +
((InstrumentedHelper)sort.getHelper()).getStatPack().mean("hits"));
                size = size*2;
            }if(type==2){
                des = "MergeSort";
                Helper<Integer> helper = HelperFactory.create(des,size,true,config);
                SortWithHelper<Integer> sort = new MergeSort<>(helper);
                Integer[] ints = helper.random(Integer.class, r -> r.nextInt());
```

```
                SorterBenchmark sorterBenchmark = new SorterBenchmark(Integer.class,
(SortWithHelper) sort, ints, runs, timeLoggersLinearithmic);
                sorterBenchmark.run(size);

System.out.println(((InstrumentedHelper)sort.getHelper()).getStatPack().mean("compares")
+ " , " + ((InstrumentedHelper)sort.getHelper()).getStatPack().mean("swaps") + " , "
+((InstrumentedHelper)sort.getHelper()).getStatPack().mean("copies") + " , " +
((InstrumentedHelper)sort.getHelper()).getStatPack().mean("hits"));
                size = size*2;
            }else{
                des = "HeapSort";
                Helper<Integer> helper = HelperFactory.create(des,size,true,config);
                SortWithHelper<Integer> sort = new HeapSort<>(helper);
                Integer[] ints = helper.random(Integer.class, r -> r.nextInt());
                SorterBenchmark sorterBenchmark = new SorterBenchmark(Integer.class,
(SortWithHelper) sort, ints, runs, timeLoggersLinearithmic);
                sorterBenchmark.run(size);

System.out.println(((InstrumentedHelper)sort.getHelper()).getStatPack().mean("compares")
+ " , " + ((InstrumentedHelper)sort.getHelper()).getStatPack().mean("swaps") + " , "
+((InstrumentedHelper)sort.getHelper()).getStatPack().mean("copies") + " , " +
((InstrumentedHelper)sort.getHelper()).getStatPack().mean("hits"));
                size = size*2;
            }
        }
```