实验二：几种排序算法的实验性能比较

张俊华 16030199025

一、实验内容

实现插入排序（Insertion Soxt，IS），自顶向下归并排序（Top-down Mergesort，TDM），自底向上归并排序（Bottom-up Mergesort，BUM），随机快速排序（Random Quicksort，RQ），Dikstra3-路划分快速排序（Quicksot with Dikstca 3-way Partition，QD3P）。在你的计算机上针对不同输入规模数据进行实验，对比上述排序算法的时间及空间占用性能。要求对于每次输入运行10次，记录每次时间/空间占用，取平均值。

二、实验环境

IntelliJ IDEA 2018.2.5 (Ultimate Edition)

JRE: 1.8.0_152-release-1248-b19 amd64

JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Windows 10 10.0

三、实验步骤

1. 几种排序的实现

1. 按照题目要求，设计题目要求的插入排序（Insertion Sort，IS），自顶向下归并排序（Top-down Mergesort，TDM），自底向上归并排序（Bottom-up Mergesort，BUM），随机快速排序（Random Quicksort，RQ），Dijkstra 3-路划分快速排序（Quicksort with Dijkstra 3-way Partition，QD3P）五种排序算法
2. 编写 `generateRandom()` 函数，实现产生指定大小的随机数组功能，用于排序
3. 使用 `LinkedHashMap` 数据结构，对每次排序的时间和空间开销进行记录
4. 使用 JVM 虚拟机提供的 `Runtime` 类，对排序算法执行期间的空间开销进行计算。

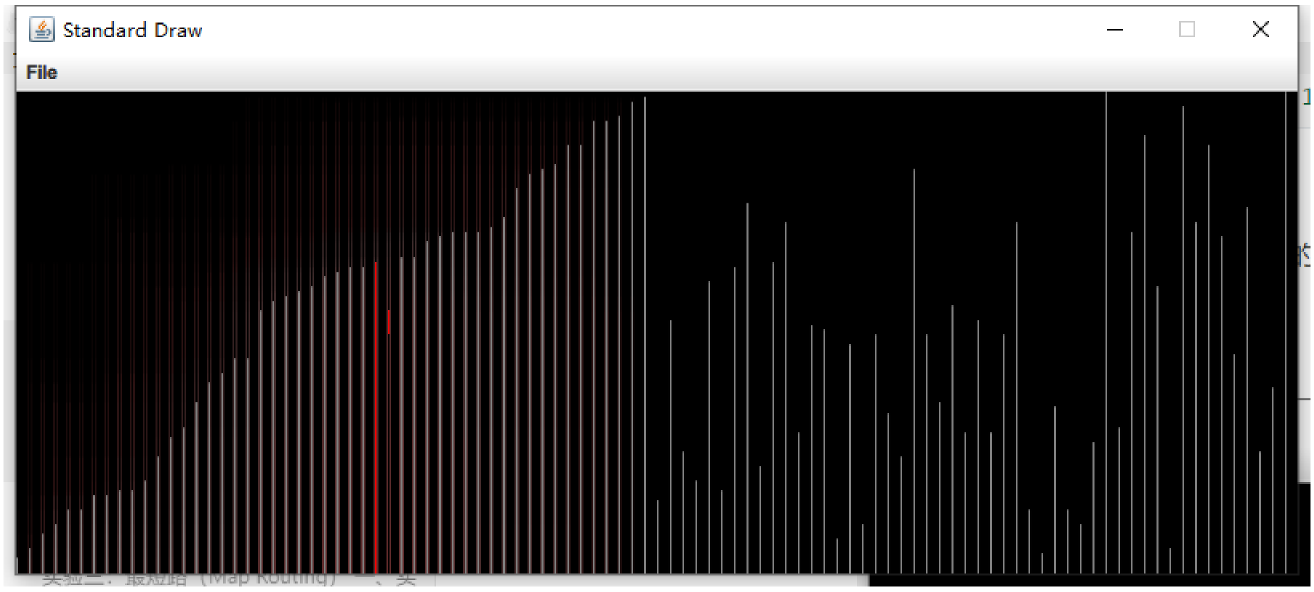
```java
        Comparable[] tlist = list.clone();
        LinkedHashMap<String,Double> timeResult = new LinkedHashMap<>();
        LinkedHashMap<String,Double> memResult = new LinkedHashMap<>();

        Stopwatch w1  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        long MemoryBefore = Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
        Insertion.sort(tlist);
        long MemoryNow = Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
        timeResult.put("IS",w1.elapsedTime());
        memResult.put("IS",1.0*(MemoryNow - MemoryBefore)/1000);
```

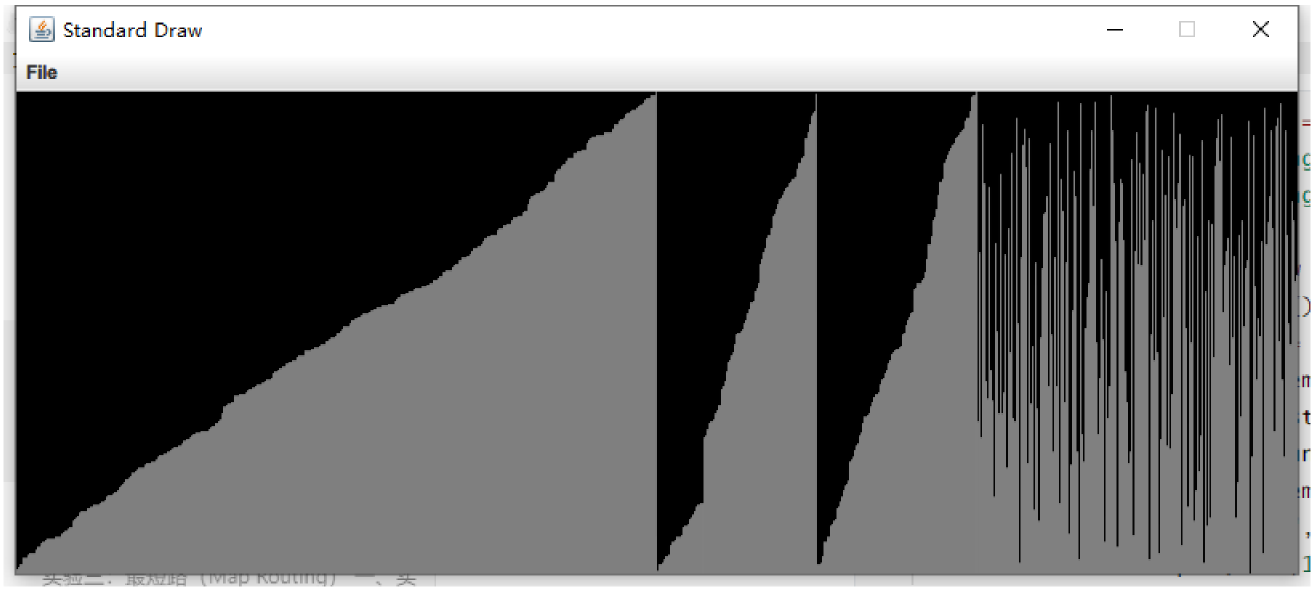
2. 排序算法的可视化

在实现了不同算法的时间空间开销记录的基础上，编写 `DrawGraph` 函数，可以将排序算法的执行过程可视化输出

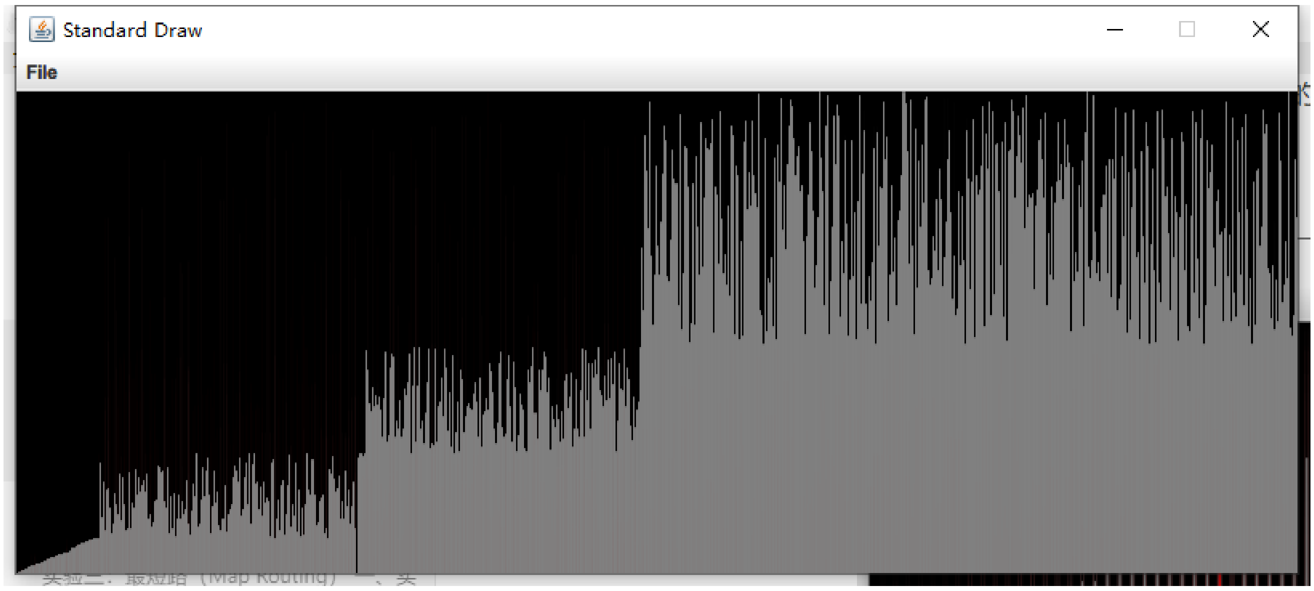
插入排序可视化中间过程：



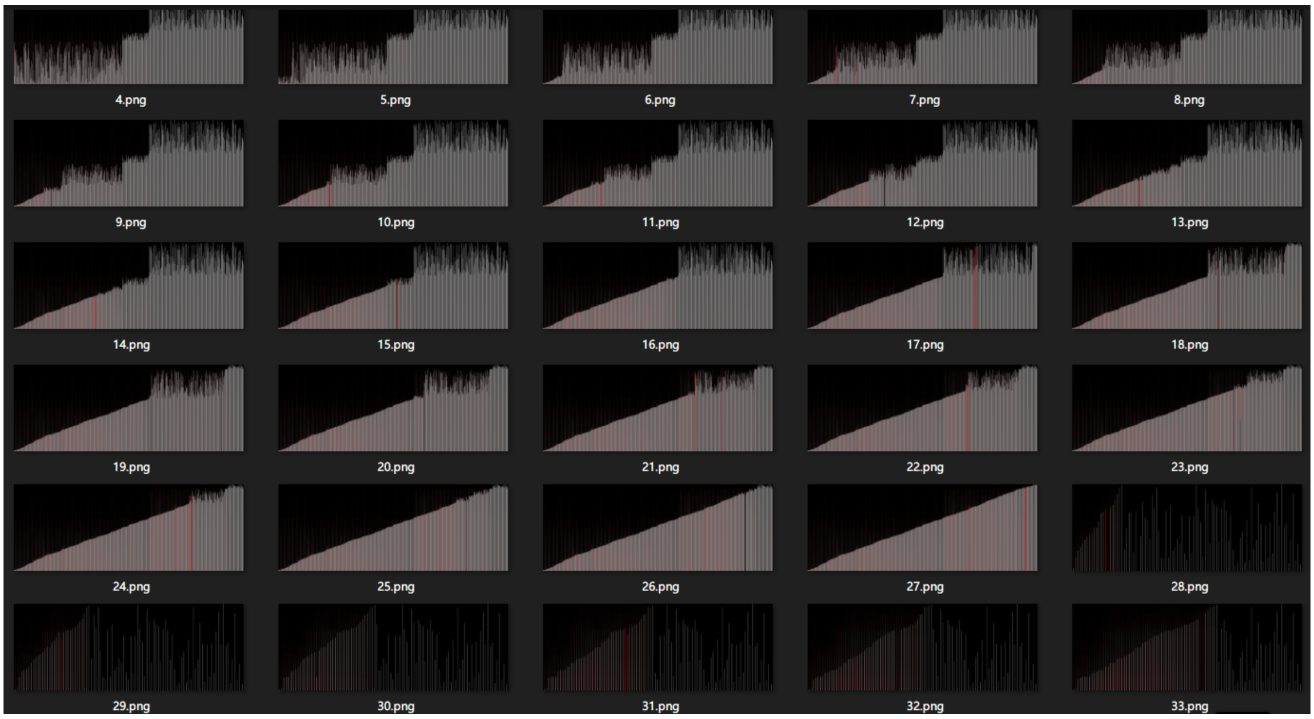
归并排序 1000 个数据元素 可视化中间过程：



快速排序 1000 个数据元素 可视化中间过程：



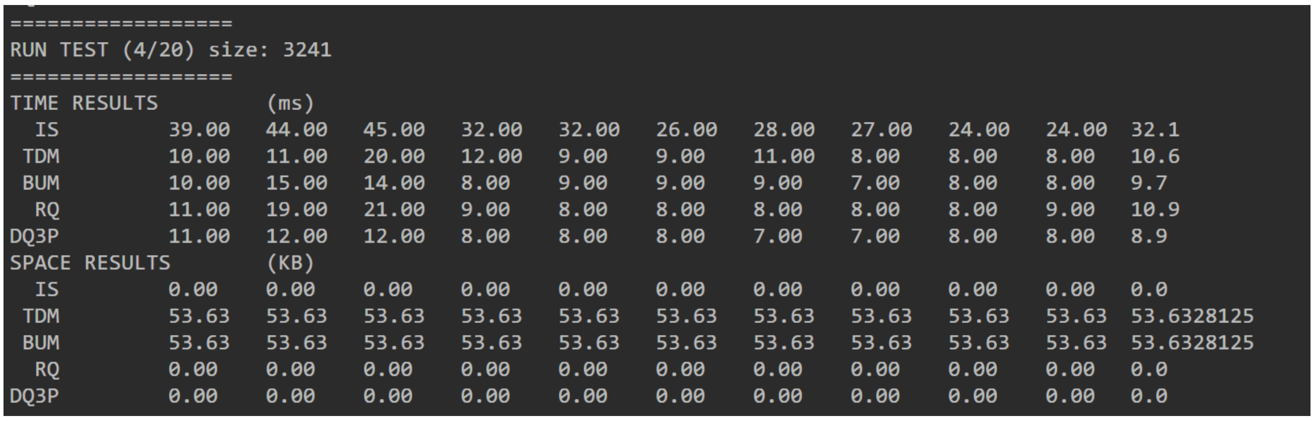
可视化的原理是重写了排序函数，在排序的关键部分，如元素交换，递归返回等位置，调用绘图函数，在改变数组元素的同时，重绘视窗中发生改变的区域，实现算法执行过程的动态展示。



四、实验结果

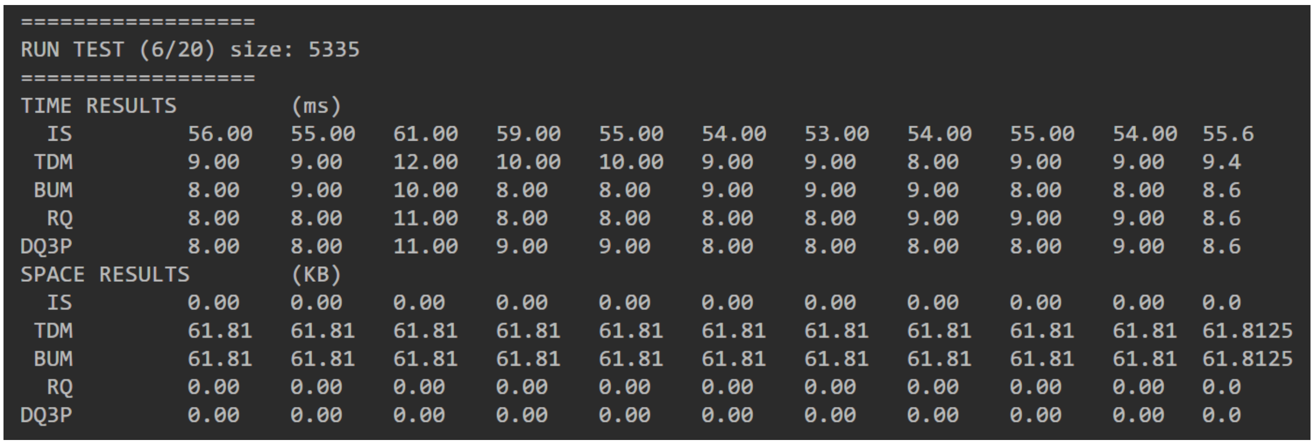
测试从 1000 到 20000 个元素的排序，从中选取部分测试结果如下所示

测试排序规模：3241 个元素



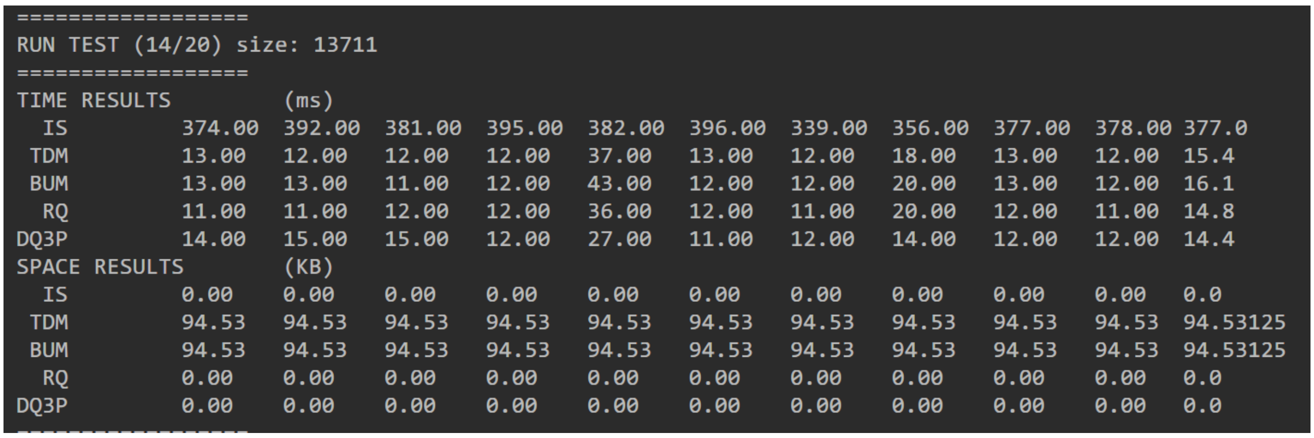
```
==================
RUN TEST (4/20) size: 3241
==================
TIME RESULTS        (ms)
  IS        39.00   44.00   45.00   32.00   32.00   26.00   28.00   27.00   24.00   24.00   32.1
 TDM        10.00   11.00   20.00   12.00   9.00    9.00    11.00   8.00    8.00    8.00    10.6
 BUM        10.00   15.00   14.00   8.00    9.00    9.00    9.00    7.00    8.00    8.00    9.7
  RQ        11.00   19.00   21.00   9.00    8.00    8.00    8.00    8.00    8.00    9.00    10.9
DQ3P        11.00   12.00   12.00   8.00    8.00    8.00    7.00    7.00    8.00    8.00    8.9
SPACE RESULTS       (KB)
  IS        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
 TDM        53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.6328125
 BUM        53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.63   53.6328125
  RQ        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
DQ3P        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
```

测试排序规模：5335 个元素



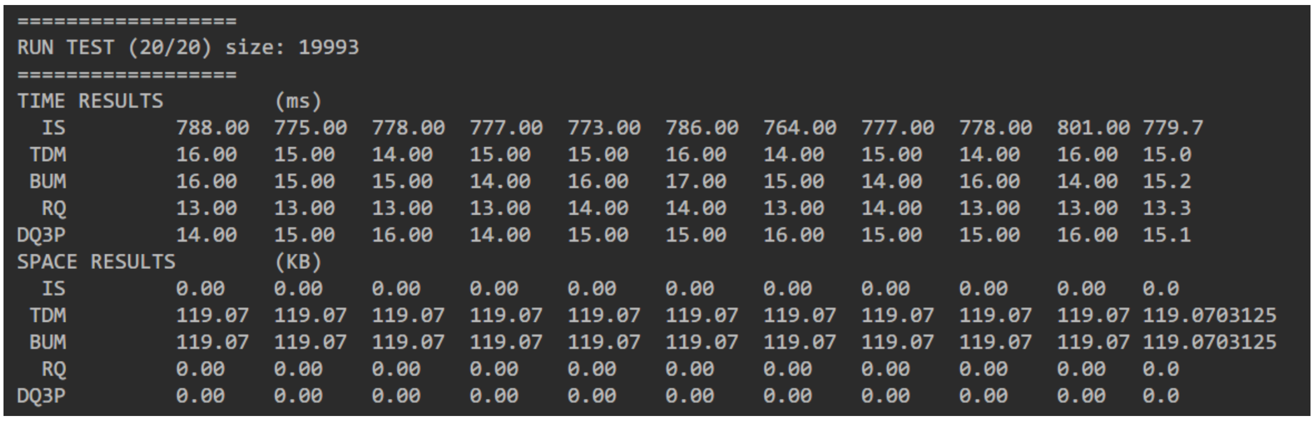
```
==================
RUN TEST (6/20) size: 5335
==================
TIME RESULTS        (ms)
  IS        56.00   55.00   61.00   59.00   55.00   54.00   53.00   54.00   55.00   54.00   55.6
 TDM        9.00    9.00    12.00   10.00   10.00   9.00    9.00    8.00    9.00    9.00    9.4
 BUM        8.00    9.00    10.00   8.00    8.00    9.00    9.00    9.00    8.00    8.00    8.6
  RQ        8.00    8.00    11.00   8.00    8.00    8.00    8.00    9.00    9.00    9.00    8.6
DQ3P        8.00    8.00    11.00   9.00    9.00    8.00    8.00    8.00    8.00    9.00    8.6
SPACE RESULTS       (KB)
  IS        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
 TDM        61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.8125
 BUM        61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.81   61.8125
  RQ        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
DQ3P        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
```

测试排序规模：13711 个元素



```
==================
RUN TEST (14/20) size: 13711
==================
TIME RESULTS        (ms)
  IS        374.00  392.00  381.00  395.00  382.00  396.00  339.00  356.00  377.00  378.00 377.0
 TDM        13.00   12.00   12.00   12.00   37.00   13.00   12.00   18.00   13.00   12.00   15.4
 BUM        13.00   13.00   11.00   12.00   43.00   12.00   12.00   20.00   13.00   12.00   16.1
  RQ        11.00   11.00   12.00   12.00   36.00   12.00   11.00   20.00   12.00   11.00   14.8
DQ3P        14.00   15.00   15.00   12.00   27.00   11.00   12.00   14.00   12.00   12.00   14.4
SPACE RESULTS       (KB)
  IS        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
 TDM        94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53125
 BUM        94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53   94.53125
  RQ        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
DQ3P        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
```

测试排序规模：19993 个元素



```
==================
RUN TEST (20/20) size: 19993
==================
TIME RESULTS        (ms)
  IS        788.00  775.00  778.00  777.00  773.00  786.00  764.00  777.00  778.00  801.00 779.7
 TDM        16.00   15.00   14.00   15.00   15.00   16.00   14.00   15.00   14.00   16.00   15.0
 BUM        16.00   15.00   15.00   14.00   16.00   17.00   15.00   14.00   16.00   14.00   15.2
  RQ        13.00   13.00   13.00   13.00   14.00   14.00   13.00   14.00   13.00   13.00   13.3
DQ3P        14.00   15.00   16.00   14.00   15.00   15.00   16.00   15.00   15.00   16.00   15.1
SPACE RESULTS       (KB)
  IS        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
 TDM        119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07 119.0703125
 BUM        119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07  119.07 119.0703125
  RQ        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
DQ3P        0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.0
```

观察实验结果可以发现，由于插入排序的算法时间复杂度最高，因此在不同规模的测试中所用时间均最久。快速排序在几种排序中所用时间最少，归并排序次之，但两者差距不大。

从空间占用情况看，因为归并排序需要额外的数组空间实现归并操作，因此其空间占用一直是所有排序算法中最高，且其空间占用随着测试规模的增大而增大。

附：部分源代码

```java
import edu.princeton.cs.algs4.Insertion;
import edu.princeton.cs.algs4.Merge;
import edu.princeton.cs.algs4.Quick;
import edu.princeton.cs.algs4.Quick3way;
import edu.princeton.cs.algs4.MergeBU;

import edu.princeton.cs.algs4.StdRandom;

import java.util.LinkedHashMap;


public class SortTest {
    public static Integer[] randomList;
    /**
     * 产生 size 大小的随机数组
     * @param size
     */
    public static Integer[] generateRandom(int size){
        randomList = new Integer[size];
        for (int j = 0; j<size;j++){
            randomList[j] = StdRandom.uniform(size);
        }
        StdRandom.shuffle(randomList);
        return randomList;
    }

    /**
     * 对 list 数组元素进行一次排序测试
     * @param list
     */
    public static LinkedHashMap[] runTest(Comparable[] list){
        Comparable[] tlist = list.clone();
        LinkedHashMap<String,Double> timeResult = new LinkedHashMap<>();
        LinkedHashMap<String,Double> memResult = new LinkedHashMap<>();

        Stopwatch w1  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        long MemoryBefore = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        Insertion.sort(tlist);
        long MemoryNow = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        timeResult.put("IS",w1.elapsedTime());
        memResult.put("IS",1.0*(MemoryNow - MemoryBefore)/1000);


        tlist = list.clone();
        Stopwatch w2  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        MemoryBefore = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        Merge.sort(tlist);
        MemoryNow = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        timeResult.put("TDM",w2.elapsedTime());
        memResult.put("TDM",1.0*(MemoryNow - MemoryBefore)/1024);

        tlist = list.clone();
        Stopwatch w3  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        MemoryBefore = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        MergeBU.sort(tlist);
        MemoryNow = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        timeResult.put("BUM",w3.elapsedTime());
        memResult.put("BUM",1.0*(MemoryNow - MemoryBefore)/1024);

        tlist = list.clone();
        Stopwatch w4  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        MemoryBefore = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        Quick.sort(tlist);
        MemoryNow = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        timeResult.put("RQ",w4.elapsedTime());
        memResult.put("RQ",1.0*(MemoryNow - MemoryBefore)/1024);

        tlist = list.clone();
        Stopwatch w5  = new Stopwatch();
        Runtime.getRuntime().gc(); //空间回收
        MemoryBefore = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        Quick3way.sort(tlist);
        MemoryNow = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
        timeResult.put("DQ3P",w5.elapsedTime());
        memResult.put("DQ3P",1.0*(MemoryNow - MemoryBefore)/1024);

        LinkedHashMap[] RESULT = new  LinkedHashMap[2];
        RESULT[0] = timeResult;
        RESULT[1] = memResult;
        return RESULT;
    }

    public  void runTests(int size){

        TestResult rec = new TestResult();
        for(int i = 0; i<10;i++){
            generateRandom(size);
            rec.record(runTest(randomList));
        }
        rec.printResult();
    }

}
```

```java
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdStats;

import java.util.ArrayList;
import java.util.Map;
import java.util.LinkedHashMap;


public class TestResult {
    LinkedHashMap<String, ArrayList<Double>> timeResults = new LinkedHashMap<>();
    LinkedHashMap<String, ArrayList<Double>> spaceResults = new LinkedHashMap<>();
    int testTimes;


    /**
     * 记录一次测试结果
     */
    public void recordTimeResult(LinkedHashMap<String,Double> result){
        recordResult(result, timeResults);
    }
    /**
     * 记录一次测试结果
     */
    public void recordSpaceResult(LinkedHashMap<String,Double> result){
        recordResult(result, spaceResults);
    }

    private void recordResult(LinkedHashMap<String, Double> result, 
LinkedHashMap<String, ArrayList<Double>> timeResults) {
        if (timeResults.isEmpty()){
            for (Map.Entry<String, Double> entry : result.entrySet()) {
                ArrayList<Double> time = new ArrayList<>();
                time.add(entry.getValue());
                timeResults.put(entry.getKey(),time);
            }
        }
        else {
            for (Map.Entry<String, Double> entry : result.entrySet()) {
                timeResults.get(entry.getKey()).add(entry.getValue());
            }
        }
    }

    public void record(LinkedHashMap<String, Double>[] result){
        recordTimeResult(result[0]);
        recordSpaceResult(result[1]);
    }

    /**
     * 打印测试结果
     */
    public void printResult(){
        StdOut.println("TIME RESULTS \t\t (ms)");
        prs(timeResults);
        StdOut.println("SPACE RESULTS \t\t (KB)");
        prs(spaceResults);

    }

    private void prs(LinkedHashMap<String, ArrayList<Double>> spaceResults) {
        for (Map.Entry<String, ArrayList<Double>> entry : spaceResults.entrySet())
{
            StdOut.printf("%4s\t\t",entry.getKey());
            double[] entryTime = new double[entry.getValue().size()];
            int i = 0;
            for (Double time : entry.getValue()){
                StdOut.printf("% 4.2f\t",time);
                entryTime[i++] = time;
            }
            StdOut.println(StdStats.mean(entryTime));
        }
    }
}
```