Time Measurements part 2

Tomas Marx-Raacz von Hidvég

StudentId code: tendn09
Linnaeus University, 1dv507
tendn09.student@lnu.se

Abstract

This report is intended to investigate time measurement of two types implementations of InsertionSort. Our experiment will test how many elements of Integers and Strings can be sorted using the authors own implementation of the method. We aim to investigate the difference between the two and also see to what extent these can be used when run for a full second.

Our experiment shows that the speed of which the Integers are sorted far exceeds the ones of the Strings . With Integers ranging at the Integer[].length*2 giving a result of 100000 sorted elements under 1 second, the String containing 10 random characters showed a result of 19800.

Introduction

In the Java programming language there are a lot of sorting algorithms. One of them is InsertionSort that uses a recursive sorting mechanism.

In this experiment we will compare sorting Strings and Integers in above mentioned implementation of InsertionSort. The Integer test is going to be run with random numbers with a max interval of twice the size of the Integer array that is going to be sorted. The String array is going to be filled with Strings containing 10 random characters. This is done to get a wide range of Strings and Integers and reduce the chance of doubles.

The Experiment

The test setup for these runs is a heavily modified 5 year old Asus G751jy:
https://www.asus.com/us/ROG-Republic-Of-Gamers/ROG-G751JY/specifications/
It is equipped with Intel I7 4710HQ mobile 4 core @2,5ghz, 32GB of memory and equipped with SATA SSDs. It is running on Windows 10 Pro and we used Java 14 and changed the Java virtual machine with the following parameters to increase the heap size as well as the stack size:

-Xmx4096, -Xms16384, -Xss16m(This was very important as a lower number got Stack Overflow from the Integer test)

We also closed all non system critical applications to reduce interference from the system as well as making sure to the best of our knowledge to make the application for running the test does not interfere with the actual test. In that respect we isolated the actual run of the test from the rest of the code and made sure to use garbagecollector before each run. The following code ran the test:

```
public static int timeTestIntInsertionSort (int[] unsortedArray, Runtime runtime) {

        runtime.gc();

        long before = System.currentTimeMillis();

        insertionSort(unsortedArray);

        long after = System.currentTimeMillis();

        return (int) (after - before);}
```

And:

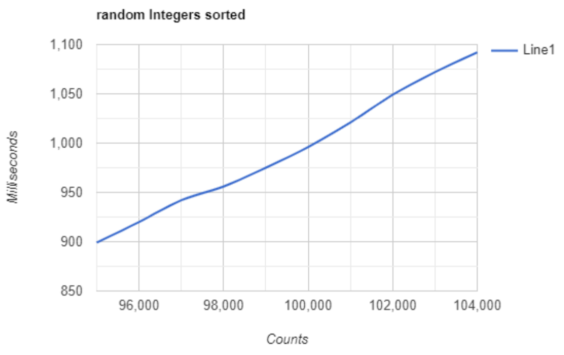
```
public static int timeTestStringInsertionSort(String[] arrayToSort, Runtime runtime){

        runtime.gc();

        long before = System.currentTimeMillis();

        insertionSort(arrayToSort, new IntComparator<>());
        long after = System.currentTimeMillis();
        return (int) (after - before); }
```

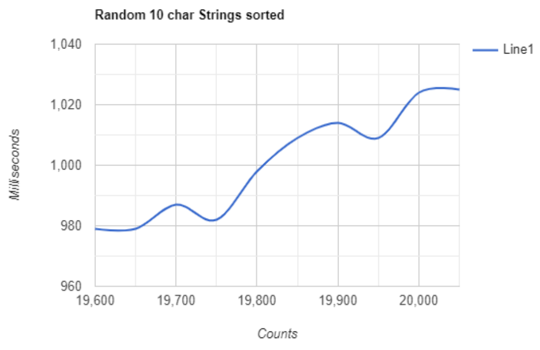
The above mentioned code is what the author considered to be the bare minimum to run per test.
It consists of a garbagecollection, an initiation of the Insertion sort with the provided array of either Integers or Strings, and the actual test using `System.currentTimeMillis()`.Each of these tests received their own class with various helper methods for random automating the test suites.

In all four test suites the experiment started of with a try-and-error search for a number of sorts close to, but not over 1000 milliseconds. After this an interval was chosen from said initial test phase. These depended largely on the process of reaching the said number of sorts. Each of these tests were run in 10 repetitions with an average calculated.
The numbers of concatenations found were:

	Integer InsertionSort	String InsertionSort
Found repetitions	100000	19800
Start of interval	95000	19600
Increase per step	1000	50

We can already see a pattern emerging, but further tests and results was required to try to spot the closest we could get to a second and what it yielded as a result.
The tests were as mentioned before done in 10 repetition of which an average was calculated.
Prior to the tests a warmup session of 50 repetitions of the "found repetitions" value was run after which 10 steps of 10 tests were conducted, starting at the interval start found in the table above and each consecutive with the increase counted in. Each average was then saved and plotted into a curve.





As we can see the Integer tests yielded largely the same result as the trial and error type initial investigation at 100000 sorted elements. The String tests showed far more fluctuations but yielded largely the same result as the initial investigation as well. We can also see that the Integer sorting is about five times as fast as that of the String. This comes down to the fact that Integers have in itself a way of sorting it, it has a value, while Strings on the other hand have to be somehow converted to a tangible number for the method to use for sorting.

Conclusion

As explained prior these tests really do not offer that many surprises in effectiveness due to Integers being naturally sortable and Strings not, thus the method which sorts the elements has to do a recalculation of sorts for each and every element that comes its way. In either way it is very interesting to see just how large the difference is.

Possible Improvements

This was a second attempt into researching time measurement, and without doubt it did not provide perfect results. It did however give better results than the one before which in turn means that something in the code must have become better. There is however always room for improvements and as the teachers of the course have showed, it might be a good idea to also measure memory along with these tests to see if there is a memory leakage somewhere that interferes with the result.