Time Measurements

Student Marcus Johansson

Department of Computer Science, Linnaeus University
Magna floor 4, 391 82 Kalmar Sweden
mj223vn@student.lnu.se

Abstract - **The task handled in this report is to found out how many integers and strings can be sorted in one second using the two algorithms insertion sort and merge sort. The integers will be sorted in size and the strings containing ten randomly picked characters is sorted in alphabetical order. We use our own implementation of insertion sort and merge sort.**

Our experiment shows that with insertion sort we can sort about 90000 integers and 22000 strings in one second contrary to merge sort we can sort about 6 million integers and 1,25 million strings in one second.

I. Exercises

The problems we handle in this report is the following: a) How many integer can we sort in one second using insertion sort b) How many integer can we sort in one second using merge sort c) How many strings can we sort in one second using insertion sort d) How many strings can we sort in one second using merge sort.

II. Experimental setup

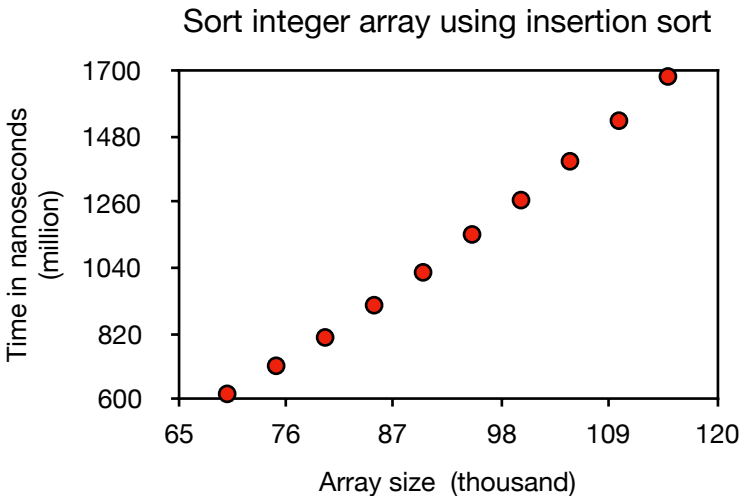
All experiments was done on a MacBook Pro with an Intel core i5 processor 2 GHz with 8GB of memory. We used Java 8 201. To get as accurate measurements as possible we closed all other applications and nothing else then the sorting is running during the time measurement. We are using the same approach in all four measurements. An garbage collection is run before each test to to avoid having memory problems. We start with one warm up run of the program to optimizing the JVM for our code. Then we measure the time for the sorting ten times and get the average time by dividing with ten. We use our own method timeSortIntegerArray(**int** size, **int** randomSize). The input size is the size of the array to be randomly generated using our own method randomInArray(**int** length, **int** n), randomSize is the integers generated from zero to Integers.**MAX_VALUE**. For strings we use our own method timeSortingString(**int** numberOfWords). The input numberOfWords is the size of an String array created by owner own method genarateRandomStrings(**int** numberOfWords) that generates an random string array containing words with ten characters. Characters from the English alphabet. A code example from the method timeSortingStrings is represented at top right:

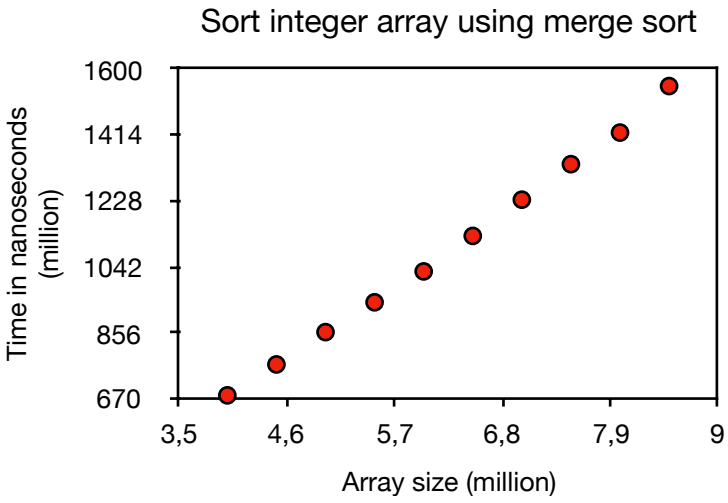
```
private static long timeSortingString(int numberOfWords) {
    InsertionSort sort = new InsertionSort();
    Comparator<String> c = (s1, s2) -> s1.compareTo(s2);
    String[] arr = genarateRandomStrings(numberOfWords);
     Timer timer = new Timer();
     timer.Start();
     sort.insertionSort(arr, c);
     timer.Stop();
     long returnTime = timer.getElapsedTime();
     return returnTime;
}
```

Timer is our own class to handle the time measurement. And the InsertionSort class is implemented to handle the sorting. For each experiment we started out with an try and error approach to find array sizes to sort around one second. Then we pick an interval around that number to get a more accurate result from our test runs. The interval for strings using insertion sort is [18, 27] thousand.

A out print ruining the code for timeSortingString:

```
Number of words = 21000
Time = 1018183329
Memory = 9MB
```



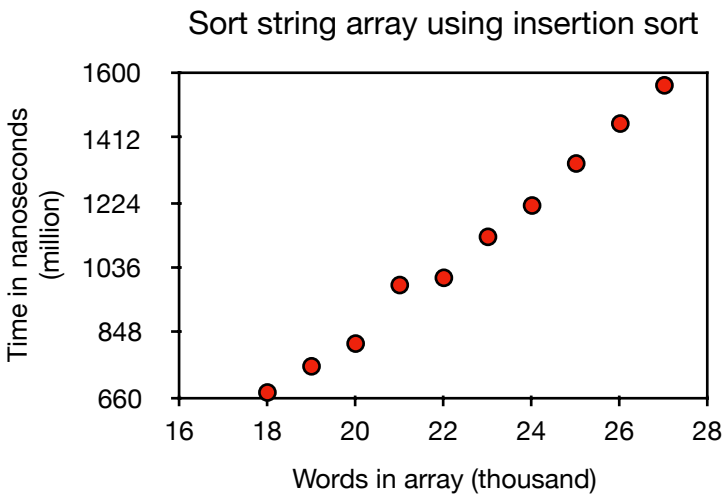


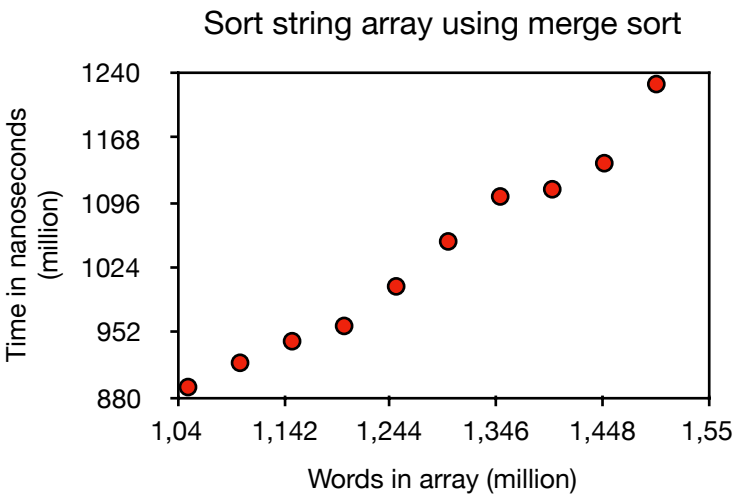
Sort integer array using insertion sort		
Size	**Time (nanoseconds) (million)**	**Memory (MB)**
70000	618	2
75000	712	2
80000	807	2
85000	915	3
90000	1025	2
95000	1152	1
100000	1267	3
105000	1397	2
110000	1533	1
115000	1681	3

Sort integer array using merge sort		
Size (million)	**Time (nanoseconds) (million)**	**Memory (MB)**
4	681	74
4,5	768	123
5	859	73
5,5	943	83
6	1030	95
6,5	1130	174
7	1232	135
7,5	1332	145
8	1421	152
8,5	1552	141

III. Integer sort

The tables and diagrams above show how the different measurements went by. The first thing to observe is that there is an steady increase in both insertion sort and merge sort. Memory usage in insertion sort is very low but on the contrary merge sort can handle an significantly larger sized array. Insertion sort can handle about 90000 versus merge sorts 6 million in one second.





Sort string array using insertion sort		
Number of words in array	**Time (nanoseconds) (million)**	**Memory (MB)**
18000	673	15
19000	749	34
20000	815	14
21000	985	35
22000	1006	17
23000	1125	40
24000	1216	24
25000	1338	8
26000	1454	34
27000	1565	21

Sort string array using merge sort		
Words in array (million)	**Time (nanoseconds) (million)**	**Memory (MB)**
1,05	891	228
1,1	918	405
1,2	942	423
1,25	959	595
1,3	1003	650
1,35	1053	834
1,4	1103	178
1,45	1111	418
1,5	1140	688
1,55	1228	941

IV. String sort

The tables and diagrams above show how the different measurements went by. The first thing to observe there is a not an completely steady increase in either insertion sort or merge sort. This make the measurements a bit less reliable. As in the test for integers we here also can see that merge sort can handle an significantly larger array in one second, and the memory use is larger for merge sort. Insertion sort can handle about 22000 words in one second versus 1,3 million using merge sort.

V. Conclusion

As we can see from the test results, using merge sort is much faster in sorting both integer and string arrays.

To improve our test further we can use regression to calculate an more precise position of the one second point. But for this experience we say its ”good enough” to do an estimation based on the graphs on where the one second point is.