Time Measurements

Mohammadali Rashidfarokhi

Software Technology Group

Department of Computer Science, Linnaeus University

mr223jp@student.lnu.se

Abstract— This report has been created to evaluate differents operations that were carried out by different algorithms in one Second. The total of four different algorithms was used for exercise 6 (Concatenation) such as String concatenation (short and long) and String concatenation using StringBuilder (short and long). On the other hand, for exercise 7, which is all about Insertion sort, both String array and Integer array was tested by different array sizes for at least 5 different executions to calculate the average.

I. Introduction

One of the main purposes of this report is to have the ability to report your code results and introducing the reader the results of your experiments. That is why one of the tasks in 1DV507 course, specifically, assignment 4, the developer is supposed to measure and state the outcome of experiments that were carried out by using different algorithms. Consequently, apart from reporting the results, the developer is supposed to identify either StringBuilder is faster and efficient to use or using + operator.

II. Experimental Setup

Experiments were carried out on an Acer Aspire A515-52 Intel(R) Core (TM) i7-8565U CPU @ 1.80GHz, 1.99 GHz, with 8.00GB (7.85 GB usable) memory (RAM).

We used JavaSE-1.8 and changed the Java Virtual Machine heap space to 4GB Using the arguments: -Xmx4096m -Xms4096m. Additionally, all other applications were closed while performing the experiments. New Date().getTime() was used to measure time. All experiments were made using the same approach. That is, each time measurement is the average of multiple executions. We also, run a garbage collection just before each time measurements to avoid having memory problems (and avoid) garbage collections just at the same time of performing Sorting. Also, for reducing the chance of not facing the OutOfMemoryError, System.gc() was used before each loop.

III. Results

A. Concatenations: String Builder and + Operator

The developer in exercise 6 is supposed to evaluate the performance of StringBuilder and + operator to concatenate Strings.

For completing this exercise, the + operator method was used to add short Strings containing one character then the long Strings with more than 80 characters were added by different methods (String Builder and + operator).

The below outcomes demonstrate adding both short and long Strings in 1 second.

Execution	String length	Number of operations
Short String of running using string concatenation + operator	73504	73503
Long String of running using string concatenation + operator	885212	8273
Short String of running using String Builder. Append	44914223	44914222
Long String of running using String Builder. Append	582926981	4778090

Execution2	String length	Number of operations
Short String of running using string concatenation + operator	70089	70088
Long String of running using string concatenation + operator	865417	8088
Short String of running using String Builder. Append	41551323	41551322
Long String of running using String Builder. Append	550311745	4510752

Execution3	String length	Number of operations
Short String of running using string concatenation + operator	74991	74990
Long String of running using string concatenation + operator	882430	8247
Short String of running using String Builder. Append	41023656	41023655
Long String of running using String Builder. Append	365385609	2994964

Execution4	String length	Number of operations
Short String of running using string concatenation + operator	69713	69712
Long String of running using string concatenation + operator	879006	8215
Short String of running using String Builder. Append	37216431	37216430
Long String of running using String Builder. Append	41341078	3388613

Execution5	String length	Number of operations
Short String of running using string concatenation + operator	73960	73959
Long String of running using string concatenation + operator	576624	5389
Short String of running using String Builder. Append	21214718	21214717
Long String of running using String Builder. Append	301989895	2475327

Execution6	String length	Number of operations
Short String of running using string concatenation + operator	79824	79823
Long String of running using string concatenation + operator	792336	7405
Short String of running using String Builder. Append	44243381	44243380
Long String of running using String Builder. Append	601053863	4926671

Average

Average	Number of Operations	String Length
Short String of running using string concatenation + operator	73679,16666	73680.1666
Long String of running using string concatenation + operator	7602.8333	813,504,1666
Short String of running using String Builder. Append	38360,621	38360,622
Long String of running using String Builder. Append	3845736.16666	407168195.16666

B. Sorting – Insertion sort

In this exercise, just like the previous exercise new Date().getTime() method was used to measure different time and size of insertion of both String array and Integer array. So, firstly, the time of insertion sort using Integer array will be calculated, then the average will be calculated. For making sure of our experiences the code will be run multiple times.

The below are the results of Integer array:

Run1

===== "EXECUTION" 1 =====

Integer Array Size= 49985, Time = 281

String Array Size= 49985, Time = 0

===== "EXECUTION" 2 =====

Integer Array Size= 99970, Time = 984

String Array Size= 99970, Time = 0

===== "EXECUTION" 3 =====

Integer Array Size= 199940, Time = 3217

String Array Size= 199940, Time = 0

===== "EXECUTION" 4 =====

Integer Array Size= 399880, Time = 15582

String Array Size= 399880, Time = 16

===== "EXECUTION" 5 =====

Integer Array Size= 799760, Time = 54294

String Array Size= 799760, Time = 0

Run2

===== "EXECUTION" 1 =====

Integer Array Size= 49985, Time = 312

String Array Size= 49985, Time = 16

===== "EXECUTION" 2 =====

Integer Array Size= 99970, Time = 984

String Array Size= 99970, Time = 16

===== "EXECUTION" 3 =====

Integer Array Size= 199940, Time = 3191

String Array Size= 199940, Time = 0

===== "EXECUTION" 4 =====

Integer Array Size= 399880, Time = 12595

String Array Size= 399880, Time = 0

===== "EXECUTION" 5 =====

Integer Array Size= 799760, Time = 50617

String Array Size= 799760, Time = 0

Run3

===== "EXECUTION" 1 =====

Integer Array Size= 49985, Time = 375

String Array Size= 49985, Time = 16

===== "EXECUTION" 2 =====

Integer Array Size= 99970, Time = 1047

String Array Size= 99970, Time = 16

===== "EXECUTION" 3 =====

Integer Array Size= 199940, Time = 3219

String Array Size= 199940, Time = 0

===== "EXECUTION" 4 =====

Integer Array Size= 399880, Time = 12455

String Array Size= 399880, Time = 0

===== "EXECUTION" 5 =====

Integer Array Size= 799760, Time = 50845

String Array Size= 799760, Time = 16

Run 4

===== "EXECUTION" 1 =====

Integer Array Size= 49985, Time = 266

String Array Size= 49985, Time = 0

===== "EXECUTION" 2 =====

Integer Array Size= 99970, Time = 969

String Array Size= 99970, Time = 0

===== "EXECUTION" 3 =====

Integer Array Size= 199940, Time = 2985

String Array Size= 199940, Time = 0

===== "EXECUTION" 4 =====

Integer Array Size= 399880, Time = 12173

String Array Size= 399880, Time = 16

===== "EXECUTION" 5 =====

Integer Array Size= 799760, Time = 50578

String Array Size= 799760, Time = 16

Run 5

===== "EXECUTION" 1 =====

Integer Array Size= 49985, Time = 308

String Array Size= 49985, Time = 0

===== "EXECUTION" 2 =====

Integer Array Size= 99970, Time = 985

String Array Size= 99970, Time = 0

===== "EXECUTION" 3 =====

Integer Array Size= 199940, Time = 3081

String Array Size= 199940, Time = 0

===== "EXECUTION" 4 =====

Integer Array Size= 399880, Time = 12409

String Array Size= 399880, Time = 0

===== "EXECUTION" 5 =====

Integer Array Size= 799760, Time = 50506

String Array Size= 799760, Time = 9

First, we execute the program to find the size of an array that will take one second (or 10000 milliseconds) to execute.

We observe that in execution 2, the time it takes is 985ms. After finding the size, we execute the program multiple times with that array size to measure how much time exactly it takes. The result of it's average is 993.8ms. That is, an Integer array with size of 99970 takes 993.8ms in average to be stored.

We can then calculate the size of the supposed array that takes precisely1000ms to run with this data. In this case, the size of such array is 1000,593,68

The same operation will occur for the String array. However, for speeding the process the calculation of array integer will be ignored and only String array will be calculated.

Run 1

===== "EXECUTION" 1 =====

String Array Size= 3900000, Time = 63

===== "EXECUTION" 2 =====

String Array Size= 7800000, Time = 62

===== "EXECUTION" 3 =====

String Array Size= 15600000, Time = 157

===== "EXECUTION" 4 =====

String Array Size= 31200000, Time = 344

===== "EXECUTION" 5 =====

String Array Size= 62400000, Time = 984

Run2

===== "EXECUTION" 1 =====

String Array Size= 3900000, Time = 78

===== "EXECUTION" 2 =====

String Array Size= 7800000, Time = 94

===== "EXECUTION" 3 =====

String Array Size= 15600000, Time = 187

===== "EXECUTION" 4 =====

String Array Size= 31200000, Time = 344

===== "EXECUTION" 5 =====

String Array Size= 62400000, Time = 1093

Run3

===== "EXECUTION" 1 =====

String Array Size= 3900000, Time = 63

===== "EXECUTION" 2 =====

String Array Size= 7800000, Time = 78

===== "EXECUTION" 3 =====

String Array Size= 15600000, Time = 156

===== "EXECUTION" 4 =====

String Array Size= 31200000, Time = 360

===== "EXECUTION" 5 =====

String Array Size= 62400000, Time = 937

Run4

===== "EXECUTION" 1 =====

String Array Size= 3900000, Time = 62

===== "EXECUTION" 2 =====

String Array Size= 7800000, Time = 63

===== "EXECUTION" 3 =====

String Array Size= 15600000, Time = 156

===== "EXECUTION" 4 =====

String Array Size= 31200000, Time = 328

===== "EXECUTION" 5 =====

String Array Size= 62400000, Time = 1031

First, we execute the program to find the size of an array that will take one second (or 10000 milliseconds) to execute.

We observe that in execution 5, the time it takes is 984ms. After finding the size, we execute the program multiple times with that array size to measure how much time exactly it takes. The result of it's average is 1011.25ms. That is, a String array with size of 62400000 takes 1011.25ms in average to be stored.

We can then calculate the size of the supposed array that takes precisely1000ms to run with this data. In this case, the size of such array is 61705,809

IV. Conclusion

The concept of the concatenation of String with operator + would be appealing and efficient if a few String are supposed to join each other. However, using operator + will become inefficient. That is, when carrying out string concatenation inside a for loop or joining a vast number of strings are the main reason that would make this operator inefficient. Consequently, since String builder is not synchronized, not thread-safe, much faster than operator + and String Buffer so, it will become the fastest one. All in all, the String builder is better at handling multiple String. On the other hand, operator + is the ideal one when it comes to handling a few String.

Regarding the insertion sort, this algorithm is very efficient and fast for the arrays that their size is less than 100000. Also, from the data, we can sort more strings than integers in one second with this algorithm.

On the other hand, the merge sort is very fast and recommended for the arrays with large sizes.