**Report for Time Measurements – 1DV507 Assignment 4**

**Exercise 2:**

For this exercise, we are asked to use the additive operator + and StringBuilder append to produce repeated concatenations. The main objective is to find the fastest approach by measuring the amount of concatenations and the length of the final string each method can compute in 1 second when adding short strings (ones containing only one character) and adding long strings (row with 80 characters).

I divided my work into two parts, first I use the additive operator method to concatenate the short string and then use the StringBuilder append method to concatenate the short strings. After that, I used the additive operator to concatenate the long string and finally the append method to concatenate the long string.

To carry out the calculations, I created start point using System.currentTimeMillis(); allowing us to get the current time. After this a while loop has been created to run from a new current time minus the previous start time is less than 1000 ms and inside the while loop is where the concatenation using additive operator or append method happens. This method is repeated three more times for the different experiments that I mentioned previously.

**Results for concatenation using + for short string**

|  |  |  |
| --- | --- | --- |
| **Trial number** | **Concatenation** | **Length** |
| 1 | 58644 | 58644 |
| 2 | 59756 | 59756 |
| 3 | 57498 | 57498 |
| 4 | 58329 | 58329 |
| 5 | 60276 | 60276 |
| **Average** | 58900.6 | 58900.6 |

**Results for concatenation using append for short string**

|  |  |  |
| --- | --- | --- |
| **Trial number** | **Concatenation** | **Length** |
| 1 | 77903664 | 77903664 |
| 2 | 77700736 | 77700736 |
| 3 | 77801592 | 77801592 |
| 4 | 75497471 | 75497471 |
| 5 | 77801235 | 77801235 |
| **Average** | 77340939.6 | 77340939.6 |

**Results for concatenation using + operative for long string**

|  |  |  |
| --- | --- | --- |
| **Trial number** | **Concatenation** | **Length** |
| 1 | 943721 | 75497711 |
| 2 | 982955 | 78636460 |
| 3 | 954319 | 76503732 |
| 4 | 945267 | 75537438 |
| 5 | 978177 | 77622395 |
| **Average** | 960887.8 | 76759547.2 |

**Results for concatenation using append for long string**

|  |  |  |
| --- | --- | --- |
| **Trial number** | **Concatenation** | **Length** |
| 1 | 943721 | 75497711 |
| 2 | 982955 | 78636460 |
| 3 | 954319 | 76503732 |
| 4 | 945267 | 75537438 |
| 5 | 978177 | 77622395 |
| **Average** | 960887.8 | 76759547.2 |

**Exercise 3:**

For this exercise, we are asked to test the 4 different algorithms we had created in Assignment 3. These include insertion sort for integers and strings and merge sort for integers and strings. Our task is to find out how many integers and strings can be sorted using these algorithms in 1 second. For the strings, we are also supposed to have 10 character strings that are generated randomly and are sorted alphabetically.

To carry out this task, I have first made a new class in my time sub-package with the different sort methods. I have then created 2 methods for integers (one for insertion sort and one for merge sort) and only merge sort for the strings. I then have 3 additional helper methods, one to generate a random integer array and one to generate a random string array (the 3rd method mentioned is used to create the random string array method).

I have used similar concept as in exercise 2 with the while loop but instead of printing out everything inside the main, I have created static methods named intInsertionSort, intMergeSort and stringMergeSort that I call from the main.

**Results for integer insertion sort**

|  |  |
| --- | --- |
| **Trial number** | **Number of integers sorted in one second** |
| 1 | 817 |
| 2 | 773 |
| 3 | 789 |
| 4 | 739 |
| 5 | 874 |
| **Average** | 798.4 |

**Results for integer merge sort**

|  |  |
| --- | --- |
| **Trial number** | **Number of integers sorted in one second** |
| 1 | 625 |
| 2 | 769 |
| 3 | 425 |
| 4 | 487 |
| 5 | 427 |
| **Average** | 546.6 |

.

**Results for string merge sort**

|  |  |
| --- | --- |
| **Trial number** | **Number of integers sorted in one second** |
| 1 | 16 |
| 2 | 24 |
| 3 | 44 |
| 4 | 56 |
| 5 | 23 |
| **Average** | 32.6 |

There are some anomalies in both the datas above and that could be due to the background applications running on my laptop during the testing and data collection. This can be easily solved by minimizing the amount of background data running.

**Why StringBuilder is faster than additive operation:**

This is because the additive operation + makes a new copy during each of the concatenations whereas StringBuilder appends (adds) the new concatenated string to the end.