**Programming and Data Structure**

**1DV507**

**Name: Noorie Esmaili**

**Time Measurements**

Exercise 6:

This exercise was to measure how many string concatenations and StringBuilder appends can be performed in one second.

Method The experiment was done by writing an application that performs string concatenations until one second has passed storing the results and then performs StringBuilder appends until one second has passed and stored the results. The application was designed to perform multiple runs and calculate the average results. The results were measured, and the average was calculated at nine runs.

The application was run on a machine with an Intel Core i5-8th Gen running Windows 10.

prosses:

1. one variable has been created, which returns the time in milliseconds. ThisvariableusedjavalibraryclassmethodnamedSystem.currentTimeMills() for returning time.
2. a while loop has been established, which runs until current milliseconds - variable (start) is less than 1000 milliseconds. Inside this loop, a short string or long string was added using + operator.
3. one more variable has been declared, which also returns time by using the System.currentTimeMills() methods. This variable helped to ﬁnd the runtime. For instance, when the while loop ends, the difference between this variable (end) and variable (start) is the runtime.
4. During the short string case, the length of the string and the number of concatenation was the same. However, during the long string, divided the string length by 80 in order to ﬁnd the number of concatenations.

String with length 1

|  |  |  |
| --- | --- | --- |
| Time(milliseconds) | Concatenations | length |
| 1000 | 59488 | 59488 |
| 1000 | 70175 | 70175 |
| 1000 | 69183 | 69183 |
| 1000 | 73347 | 73347 |
| 1000 | 70301 | 70301 |
| 1000 | 68040 | 68040 |
| 1000 | 71421 | 71421 |

String with length 80

|  |  |  |
| --- | --- | --- |
| Time(milliseconds) | Concatenations | length |
| 1000 | 4109 | 328720 |
| 1000 | 5462 | 436960 |
| 1000 | 5524 | 441920 |
| 1000 | 5567 | 445360 |
| 1000 | 5592 | 447360 |
| 1000 | 5289 | 423120 |
| 1000 | 5271 | 421680 |

Buffer reader process:

1. Firstly, one string builder has been created.
2. Then, one variable has been created, which returns the time in milliseconds. This variable used java library class method named System.currentTimeMills() for returning time.
3. Furthermore, a while loop has been established, which runs until the current milliseconds - variable (start) is less than 1000. Inside this loop, a short string or long string was added using + operator.
4. After this loop, one more variable has been declared, which also return time by using the System.currentTimeMills() methods. This variable helped to ﬁnd the runtime.
5. Then, convert the string builder to string using the toString() method and save time in a variable
6. Again, run the while loop, but this time it will run until it is smaller than 1000 milliseconds. Inside this loop, again, added short or long string in a new string builder.
7. During the short string case, the length of string and number of concatenations is the same. However, during the long string, divided the string length by 80 in order to ﬁnd the number of concatenations.
8. The problem during the long string case was that it usually takes more time than 1000 milliseconds since adding long string takes more time than adding short ones.
9. And another problem was getting outOfMemory Error for BufferReader of 80 chars.

BufferReader with length 1

|  |  |  |
| --- | --- | --- |
| Time(milliseconds) | Concatenations | length |
| 1000 | 66828924 | 66828924 |
| 1000 | 67756706 | 67756706 |
| 1000 | 58040297 | 58040297 |
| 1000 | 76570868 | 76570868 |
| 1000 | 75497471 | 75497471 |
| 1000 | 78071113 | 78071113 |
| 1000 | 75497471 | 75497471 |

Exercise 7:

The exercise was to measure the size of an array that can be sorted by the author of this report’s insertion sort algorithms (Integer and String).

An application was written that creates integer arrays and sorts them, the size changes until an array that takes ≈1 second to sort is produced. This is repeated for string arrays. The results were measured, and the average was calculated at seven runs.

The application was run on a machine with an Intel Core i5-8th Gen running Windows 10.

Insertion Sort

|  |  |  |
| --- | --- | --- |
| Insertion Sort | ArraySize | Time(seconds) |
| Int | 70794 | 1.001 |
| String | 22092 | 1.006 |
| Int | 73584 | 0.996 |
| String | 24273 | 1.003 |

Why is string builder faster than string concatenation using the + operator?

However, string concatenation also uses a string builder to add another string. The reason why string builder is faster is that string concatenation using+operator makes a copy during each concatenation, which requires memory to save and time to make that copy. The string builder adds a new string at the end position and only makes a copy in some cases. Such as during resize or if inserting an element in the middle.