Time Complexity report

Experiment

This report consists of comparisons between two different sets of algorithms, string appending v. concatenating and insertion v. merge sort.

For measurement between appending and concatenation, each comparison calls a static method and appends/concatenates the string a certain amount of time. This returns the time taken before and after using *System.nanoTime*. This process repeats until the runtime is larger or equal to 1 second, with the increment of 100 calls per runtime. The comparison will be the same both string appending and concatenating, except a *toString* method will be call after the append. To further save time, the static method only generate random string once and them applied it

Comparison between insertion and merge sort for integers (primitive) also have the similar approach. Every run generates an ascending sorted array list length n, item range from 0 to n. This will be shuffled using *Collections.shuffle* and output to array. A static method is called and returns the time taken difference before and after using *System.nanoTime*. If the time is less than one second, 10 (for integer insertion sort), 100 (for integer string sort) or 10000 (for merge sort) items more is added to array list, shuffle and return back to array, then the whole process repeats.[1] Both of these comparisons for both algorithms will be exactly the same.

Here is the configuration of the machine:

CPU:	2.5 Ghz Intel Core i5
RAM:	8 GB 1600 Mhz DDR3
Java version:	10.0.2 2018-07-17
OS:	macOS El Capitan 10.11.5

All of other unnecessary background processes were killed to reduce the CPU load, especially the IDE, which hogged up a lot of resource. The program is compiled and ran directly from Terminal using command line to further reduce more interference.

[1] Rather than re-generating array with random numbers every time, a new element is added to the array list and then get shuffled and sent back to the array which then used for sorting.

Result

Based on the implementation and performing test, here is the result.

Run / Calls	1 character		80 characters	
	Appending	Concatenation	Appending	Concatenation
1	97,900	79,000,100	11,100	4,299,200
2	98,100	79,000,100	10,700	4,299,200
3	97,800	79,000,600	11,100	4,299,200
4	98,500	79,000,600	11,300	4,299,200
5	97,400	79,000,600	11,300	4,299,200
Average	97,940	79,000,400	11,100	4,299,200

Table 1: Comparison between appending and concatenating string. Time limit: 1 sec. Increment: 100 calls.

Run / Calls	Integers		Strings	
	Insertion Sort	Merge Sort	Insertion Sort	Merge Sort
1	75,510	3,560,000	5,100	260,000
2	75,820	4,940,000	5,500	240,000
3	75,570	3,560,000	5,500	230,000
4	75,800	3,560,000	5,500	230,000
5	75,590	3,560,000	5,500	220,000
Average	75,658	3,836,000	5,420	236,000

Table 2: Comparison between insertion and merge sort on array. Time limit: 1 sec. Increment: 100 items.

It is obvious that merge and appending are much faster by when their number of calls are 3 more digits. However, as stated before, there is always interference in the background. They consists of some system processes that cannot be terminated, otherwise the system will not work.