CS 3339

Insertion Sort Analysis Report

Youssef Abdeltawab, Suman Mallah, Tanner McGee, Farrah Omar, Quetzin Pimentel

https://git.txstate.edu/tbm54/archRepo

In our project, we wanted to evaluate the time efficiency of the insertion sort algorithm written in both C++ and Python between several different architectures. In order to keep external variables at a minimum, each algorithm was executed in the same IDE (VS Code) and the exact same unsorted list of 2000 values was used.

The architectures we used to in this project include: Quetzin's macOS 14.4.1, Apple M1 Max with 10 cores @ 2.06 Ghz, 16GB; macOS Sonoma 14.4.1, Intel i7 9750H with with 6 cores @ 2.6 GHz, 16GB; Windows 11 Intel i7 11700K with 8 cores @ 3.6 GHz, 32GB; Windows 11, Intel i9 10850K with 10 cores @ 3.6 GHz, 32GB. Tanner's Windows, Intel i7 4790k with 4 cores @ 4.0 GHz, 16GB; Windows, AMD Ryzen 7 5800H with 8 cores @ 3201 MHz, 16GB. Tanner’s Windows 10, Intel i7 4790k with 4 cores @ 4.0 GHz, 16 GB; Windows 11, AMD Ryzen 7 5800H with 8 cores @ 3201 MHz, 16 GB. Suman’s Windows 11, Intel i7 9750H with 6 cores @ 2.60 GHz, 16 GB. Youssef's Windows 11, Intel i7 11800H with 8 cores @ 2.30 GHz, 2.304 Ghz, 32 GB

We predicted that C++ would be the most time efficient language since it is a compiled language especially when compared to Python which is an interpreted language. We also predicted that the architecture with the highest core count, clock rate

frequency, and RAM would be the most time efficient architecture. We predicted that this architecture was the Windows 11, Intel i9 10850K with 10 cores @ 3.6 GHz, 32 GB. Conversely, we predicted that the least time efficient architecture will be the architecture with the lowest core count, clock rata, and RAM. macOS Sonoma 14.4.1, Intel i7 9750H with 6 cores @ 2.6 GHz, 16 GB. Interestingly, we have the Intel i7 9750H with 6 cores @ 2.6 GHz, 16 GB architecture running on two separate machines running on two different operating systems Windows 11 and macOS Sonoma 14.4.1. We predict that out of these two Windows 11 will be the most time efficient operating system.

Id	OS	CPU Year	Core Count	Base Clock	Memory	Python Time	C++ Time	Average Time
1	0	2021	10	2.06	16	84.77	35	59.885
2	0	2019	6	2.6	16	92.311	29	60.6555
3	1	2021	8	3.6	32	69.477	10	39.7385
4	1	2020	10	3.6	32	70.729	11	40.8645
5	1	2014	4	4	16	89.846	25	57.423
6	1	2020	8	3.201	16	172.707	30	101.3535
7	1	2019	6	2.6	16	31.246	5.623	18.4345
8	1	2021	8	2.304	32	86.48	24.561	55.5205

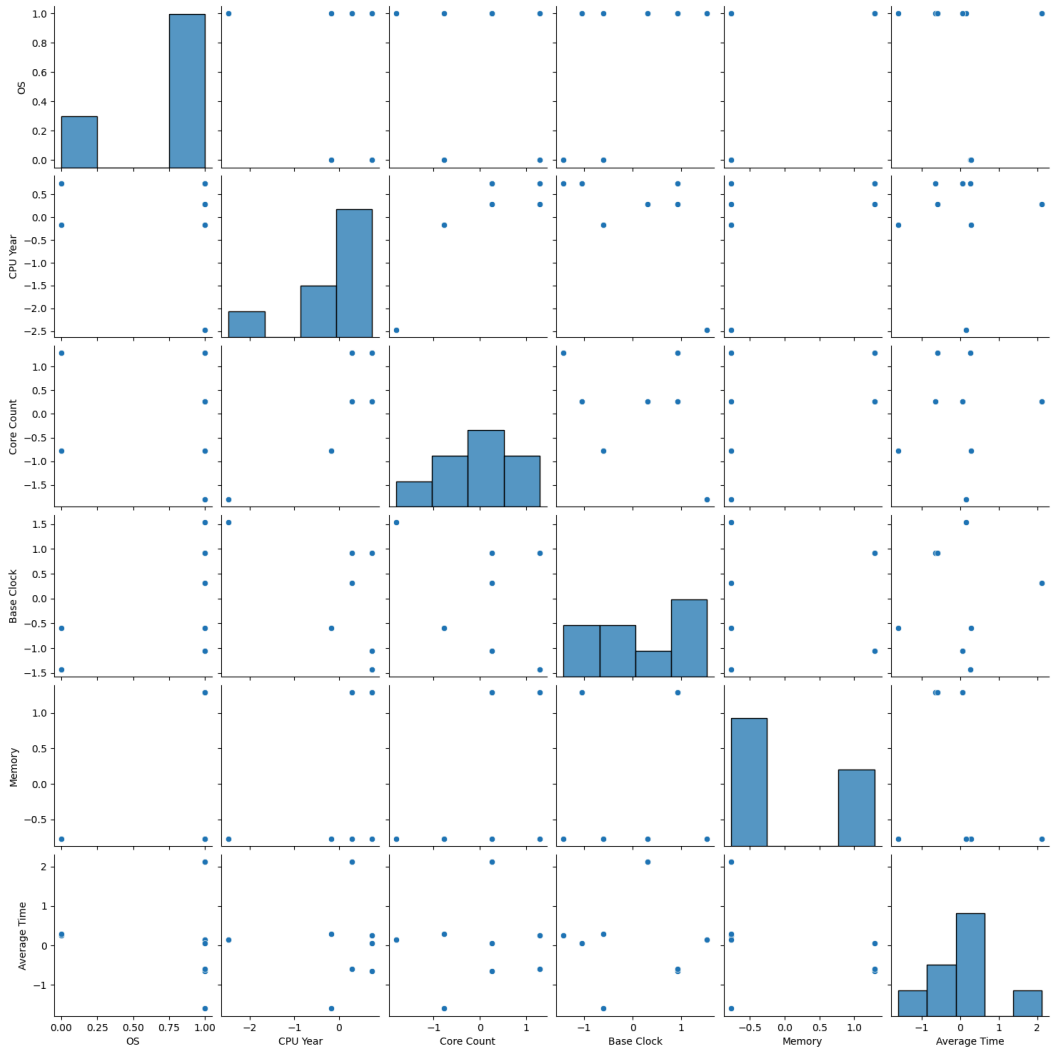
Chart of the systems we used. OS=1 is Windows, OS=0 is MacOS. Time is in milliseconds. Clock speed is in gigahertz.

When Quetzin executed the insertion sort algorithms on his i7 Mac, he yielded the following results: 92.311 ms for the Python implementation and 29 ms for the C++ implementation. When executed on his M1 Max Mac, he yielded the following results: 84.770 ms for the Python implementation and 35 ms for the C++ implementation. When executed on his i7 Windows 11, he yielded the following results: 69.477 ms for the Python implementation and 10 ms for the C++ implementation. When executed on his i9 Windows 11, he yielded the following results: 70.729 ms for the Python implementation and 11 ms for the C++ implementation. When Tanner executed the insertion sort

algorithm on his i7 Windows 10, he yielded the following results: 89.846 ms for the Python implementation and 25 ms for the C++ implementation. When executed on his AMD Ryzen 7 Windows 11, he yielded the following results: 172.707 ms for the Python implementation and 30 ms for the C++ implementation. When Suman executed the insertion sort algorithms on his i7 Windows 11, he yielded the following results: 31.246 ms for the Python implementation and 5.623 ms for the C++ implementation. When Youssef executed the insertion sort algorithms on his i7 Windows 11, he yielded the following results: 86.480 ms for the Python implementation and 24.5611 ms for the C++ implementation.

	Id	OS	Core Count	Base Clock	Memory	Python Time	C++ Time	Average Time
count	8.00000	8.00000	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
mean	4.50000	0.75000	7.500000	2.995625	22.000000	87.195750	21.273000	54.234375
std	2.44949	0.46291	2.070197	0.701861	8.280787	39.762226	10.866308	23.831878
min	1.00000	0.00000	4.000000	2.060000	16.000000	31.246000	5.623000	18.434500
25%	2.75000	0.75000	6.000000	2.526000	16.000000	70.416000	10.750000	40.583000
50%	4.50000	1.00000	8.000000	2.900500	16.000000	85.625000	24.780500	56.471750
75%	6.25000	1.00000	8.500000	3.600000	32.000000	90.462250	29.250000	60.077625
max	8.00000	1.00000	10.000000	4.000000	32.000000	172.707000	35.000000	101.353500

Chart of each data's attributes.



Normalized data showing correlation between each attribute.

In conclusion our prediction about C++ being more time efficient than Python was correct. In every machine that we tested, the C++ implementation always resulted in a faster execution time when compared to the Python execution time. When comparing the i7 9750H with 6 cores @ 2.6 GHz architecture running on Windows 11 and macOS Sonoma 14.4.1, Windows 11 ended up being the more time efficient operating system. After running the algorithms an all of our machines we concluded that the most time efficient architecture is: Windows 11, Intel i7 9750H with 6 cores @ 2.60 GHz, 16 GB; and the worst is: Windows 11, AMD Ryzen 7 5800H with 8 cores @ 3201 MHz, 16 GB.