Quick-Sort Performance
C# vs Python

Michael Ejdal Lundsgaard

December 18, 2020

Abstract

There's many situation where performance means everything for a program to be successful. Therefore it's important to choose the right programming language for the task. In this paper we're going to investigate the difference in execution time between C# and Python. I'll conclude which is faster based on execution times of an quick-sort algorithm, finally I'll discuss their use cases based on the results and features.

1 The Problem

The problem is to figure out the performance difference between C# and Python by running a quick-sort algorithm. Since we finished our Algorithms and Datastructures course, I've wanted to figure out how big the difference actually is.

The reason for using a quick-sort is that it's one of the more commonly used algorithms for larger data sets[2],

2 Specs and Information

The method is benchmarked by running it 250 times, using C#'s Stopwatch and Python's time.time. The reason for this number of iterations is to get a good sample size.

Spec The program is being run on a Windows 10 pro 64-bit machine, with an Intel Core i5-9600KF CPU at 3.70GHz, and 16GB RAM 1063MHz.

Environment The program is being run from inside C#'s Jet-Brains Rider and Visual Studio Code for Python.

3 Comparison

We're going to compare the execution speeds of Python and C#. Additionally I decided to test the execution speed of PyPy. You can view the source code in section 3.3.

3.1 Results

The file used for this experiment is taken from the science fiction book series The Foundation. Apparently the original file was too large for python to handle. Therefore I decided to make it into three separate files;

- Large: 218.335 words.
- Medium: 82.952 words
- Small: 41.807 words

You will notice that, there's a ? in the table, this is due to the program being unable to run the method. The measurements are in seconds.

Language	File	Min	Max	Median	Mean	Average	Std.dev.
C#	L	2.4620	2.8295	2.4783	2.6413	2.4998	0.0529
	M	0.3709	0.3917	0.3748	0.3813	0.3755	0.0032
	S	0.0967	0.1052	0.0977	0.1010	0.0980	0.0011
Python	L	?	?	?	?	?	?
	M	?	?	?	?	?	?
	S	1.5232	1.6001	1.5341	1.5616	1.5473	0.0231

Table 1: Comparisons between C# and Python Quick-Sort.

Looking at the average execution time, we can conclude that C# is roughly 14x faster then Python. This is not surprising since native Python doesn't have a JIT[3]. Therefore I've decide to run the code using PyPy, which is an alternative implementation for CPython. The reason that it would execute the code faster, is that it's a JIT compiler.

Language	File	Min	Max	Median	Mean	Average	Std.dev.
C#	L	2.4620	2.8295	2.4783	2.6413	2.4998	0.0529
	M	0.3709	0.3917	0.3748	0.3813	0.3755	0.0032
	S	0.0967	0.1052	0.0977	0.1010	0.0980	0.0011
Python	L	?	?	?	?	?	?
	M	?	?	?	?	?	?
	S	1.5232	1.6001	1.5341	1.5616	1.5473	0.0231
PyPy	L	?	?	?	?	?	?
	M	0.5133	0.5477	0.5212	0.5305	0.5212	0.0038
	S	0.1254	0.2214	0.1274	0.1734	0.1290	0.0069

Table 2: Comparisons between C#, Python and PyPy Quick-Sort.

You'll notice that PyPy is significantly faster then native Python, 11x faster to be specific. However C# is still 31% faster then PyPy. That being said PyPy still had issues handling the larger file. It could be interesting to try and rewrite the method in Cython, however I've never had any practice with that before.

3.2 Analysing the results

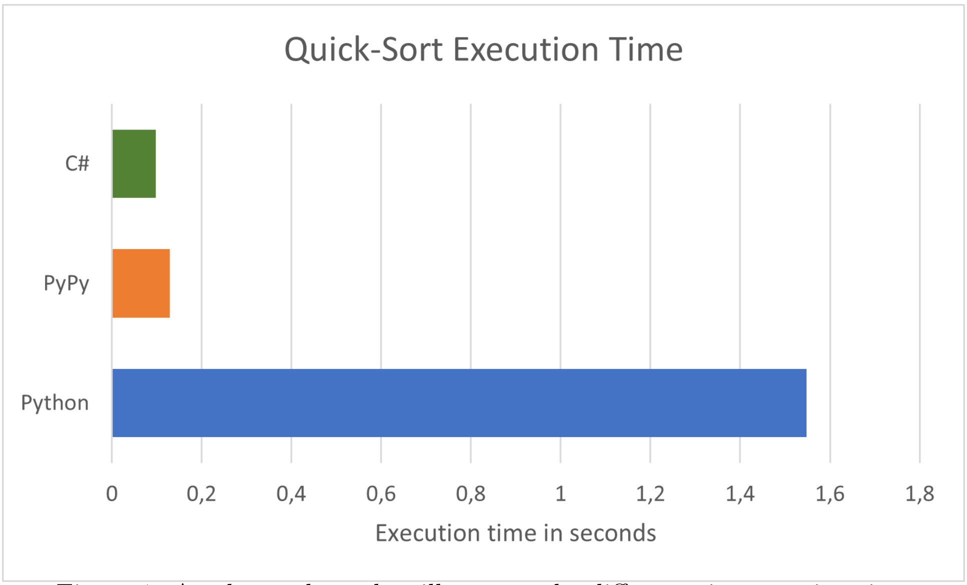

Figure 1: A column chart that illustrates the difference in execution time

In the above diagram you can see the significant difference in execution time. You might wonder why PyPy is not standard if it's so much faster then native Python. The reason for this is that PyPy isn't supported by many of the popular packages, such as Pandas, SciPy and Matplotlib.[5].

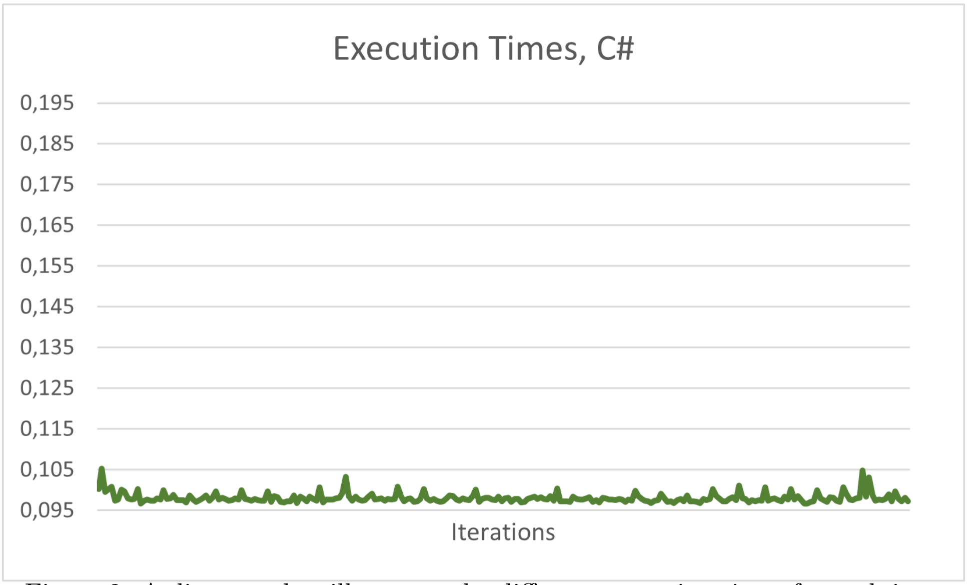

Figure 2: A diagram that illustrates the different execution times for each iteration, in C# for the small text file.

As seen on table 2 in section 3.1, the standard deviation for C# is 0.0011 seconds, which is illustrated in the above line graph, where the execution times spans between 0.105 and 0.095.

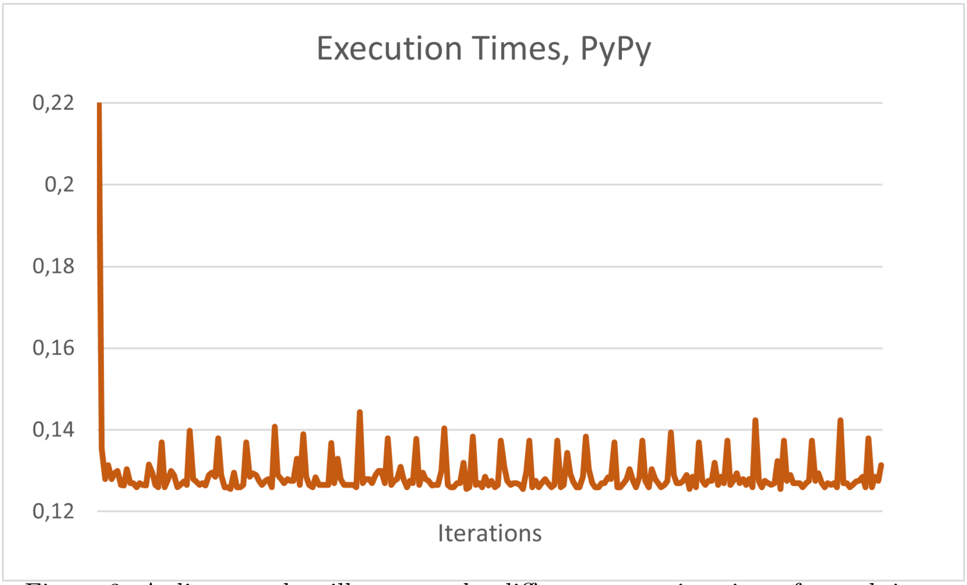

Figure 3: A diagram that illustrates the different execution times for each iteration, in PyPY for the small text file.

We can see that the line graph for PyPy looks a lot like the one for C#. The only difference is the large spike in the beginning. The reason for the spike is that PyPy is both an interpreter and compiler. The first time it runs a program it has to execute it line by line. Where after the JIT compiler comes in to optimize the run time[1].

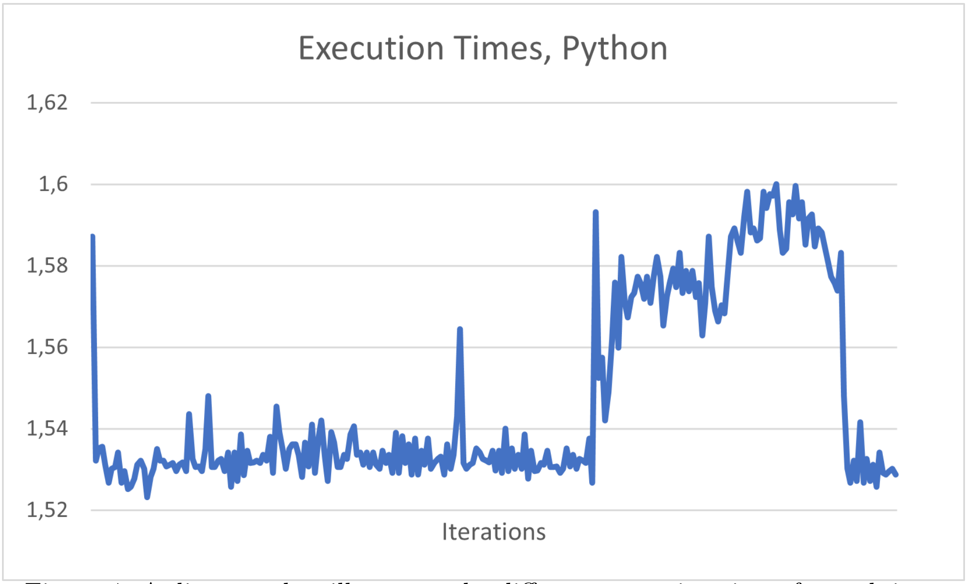

Figure 4: A diagram that illustrates the different execution times for each iteration, in Python for the small text file.

The last group of executions times, are consistent through out my multiple attempts at running the 250 iterations. This proves that it can't be caused by a background program updating. Therefore it either has something to do with my CPU or the way Python handles recursion.

3.3 Code

You can find my implementation for the quick-sort algorithm in both C# and Python on this GitHub repository: https://github.com/Gonron/Investigation-and-Reporting/tree/main/Exam.

Note, when you're running the Python implementation remember to increase the recursion limit. You can achieve this by importing sys and adding the following line of code `sys.setrecursionlimit(<limit>)`. I suggest setting the limit to 5.000

4 Conclusion

By comparing C# and Python, I concluded that C# is roughly 14x faster then Python. This was not a surprise. Following these result I decided to try and run the Python code with PyPy, which resulted in a measly 31% difference from C#. However you don't always want to utilize PyPy since it doesn't have support for some of the more popular packages.

5 Discussion and Future Work

From this experiment I've figured out how much faster the C language is, compared to Python. Even though PyPy got relatively close to matching the execution times of C#, it has some drawbacks. Python accelerates at scientific computing, data science and machine learning, which most of the packages needed for that, isn't supported by PyPy.

Also I implemented my quicksort with recursion, which is not ideal for Python[4].

As mentioned in section 3.1 I mentioned, that I would've liked to test the execution time of Cython, which is a Python-to-C compiler[6]. However I was unable to achieve this because of the time constraint.

In addition it would have been interesting to test other implementations of a quick-sort algorithm. There's different ways you can choose your pivot, one that interests me is Hoares' partition scheme, which has two pivots that start at both ends and move towards each other until they notice an inversion.

References

[1] URL: `https://www.pypy.org/performance.html`. (accessed: 18.12.2020).

[2] Abir Bhushan. *A Guide to Choosing The Best Sorting Algorithm*. Feb. 2020. URL: `https://abirbhushan.com/blog/2020/02/04/A-Guide-to-Choosing-The-Best-Sorting-Algorithm/`. (accessed: 18.12.2020).

[3] Dev Kumar. *Why is Python slower then other languages?* Jan. 2019. URL: `https://www.tutorialspoint.com/why-is-python-slower-than-other-languages#`. (accessed: 10.12.2020).

[4] Christopher Tao. *Don't Use Recursion In Python Any More*. Nov. 2020. URL: `https://towardsdatascience.com/dont-use-recursion-in-python-any-more-918aad95094c`. (accessed: 18.12.2020).

[5] Samuel S. Watson. *Why shouldn't I use PyPy over CPython if PyPy is 6.3 times faster?* Oct. 2019. URL: `https://www.quora.com/Why-shouldnt-I-use-PyPy-over-CPython-if-PyPy-is-6-3-times-faster`. (accessed: 10.12.2020).

[6] Serdar Yegulap. *What is Cython? Python at the speed of C*. Apr. 2020. URL: `https://www.infoworld.com/article/3250299/what-is-cython-python-at-the-speed-of-c.html`. (accessed: 17.12.2020).