Quick-Sort Performance
C# vs Python

Michael Ejdal Lundsgaard

December 11, 2020

Abstract

In this paper we're going to investigate why there's such a significant difference in execution time between C# and Python. There's many situation where performance means everything for a program to be successful. However there's ways to improve the performance of Python, and we're going look at one of them.

1 The Problem

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

2 Specs and Information

The method is benchmarked by running it 250 times, using C#'s Stopwatch and Python's time.time.

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 progam is being run from inside C#'s JetBrains 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

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 ? in the table, this is due to the program being unable to run the method.

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[2]. 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

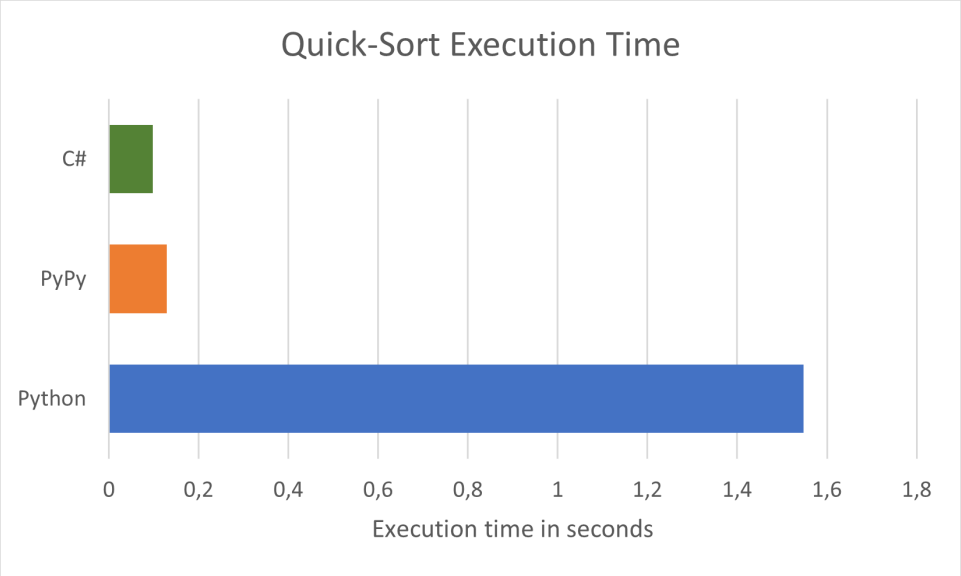

Figure 1: A column chart that illustrates the difference in execution time

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.[3].

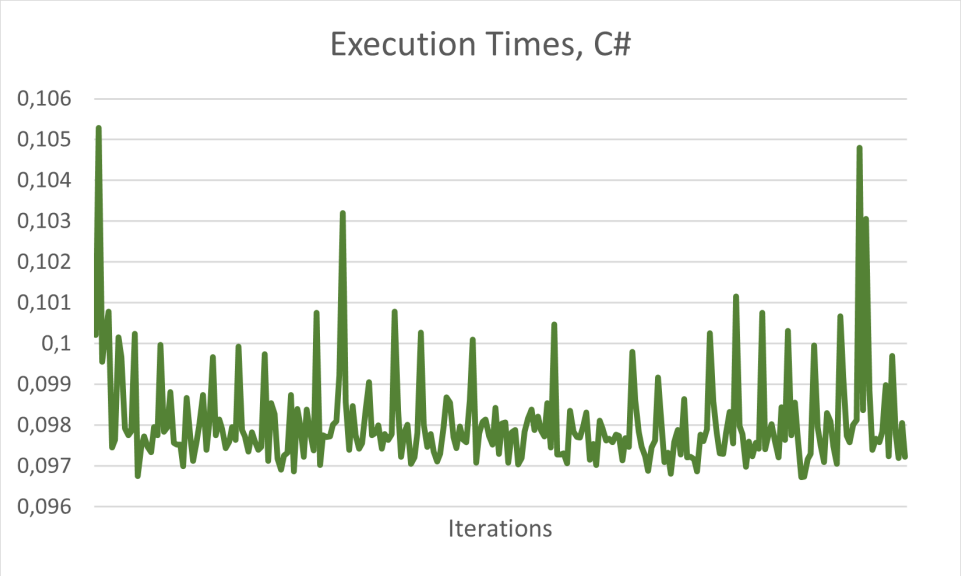

Figure 2: A diagram that illustrates the different execution times for each iteration, in C# for the small text file.

These results are more or less understandable. However, the spikes now and then boggles me. It could have something to do with a background program running on my PC. However the high spike at the beginning and the lower execution times throughout the iterations, are caused by the JIT compiler.

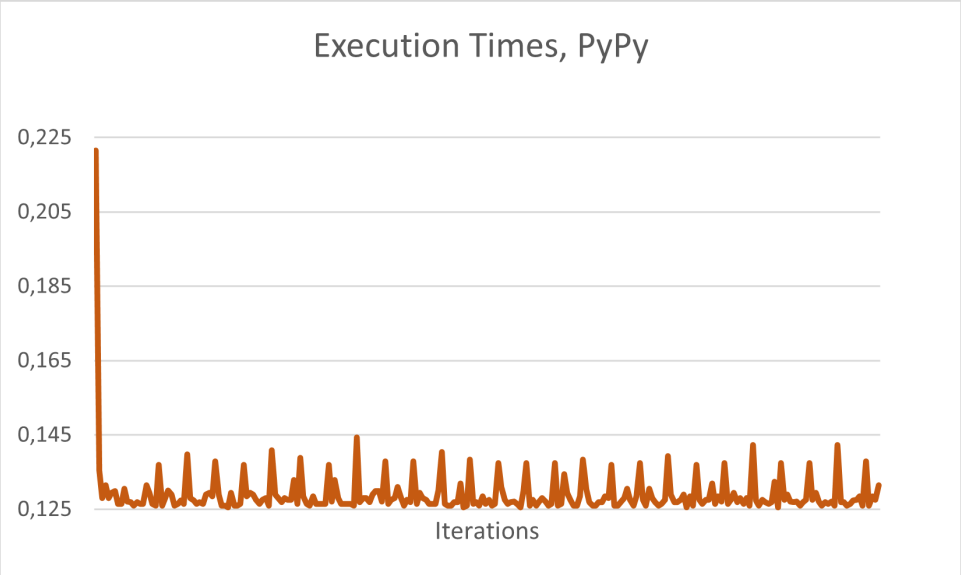

Figure 3: A diagram that illustrates the different execution times for each iteration, in PyPY for the small text file.

Here we can see the JIT compiler at work. This is more along the lines, of what i expected from the execution times in C#.

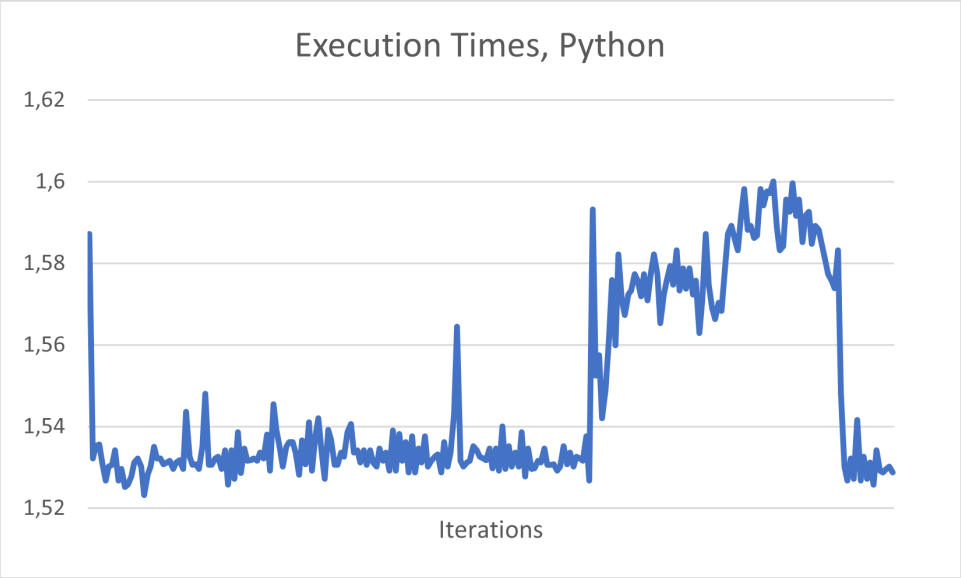

Figure 4: A diagram that illustrates the different execution times for each iteration, in Python for the small text file.

The last group of executions times must have something to do with a background-program updating on my PC. Nonetheless it doesn't effect the comparison much. Since it would still have a way higher, minimum and maximum execution time.

3.3 Code

Here you can see my implementation for the quick sort algorithm in both C# and Python. Note that if you want to access the runner code and text processor, you'll have to find it in the gitrepo.

```python
def sort(arr):
    n = len(arr)
    _sort(arr, 0, n-1)

def _sort(arr, left, right):
    if len(arr) == 1:
        return arr
    if left < right:
        pivot = partition(arr, left, right)
        _sort(arr, left, pivot - 1)
        _sort(arr, pivot + 1, right)

def partition(arr, left, right):
    pivot = arr[right]
    i = left - 1 # Starting from last element -1
    for j in range(left, right):
        # If the current element is small then
        # the pivot, increment i and swap(i, j)
        if arr[j] <= pivot:
            i+=1
            arr[i], arr[j] = arr[j], arr[i]

    # swap arr[i+1] and arr[right] (or pivot)
    arr[i+1], arr[right] = arr[right], arr[i+1]
    return i + 1
```

Listing 1: Code snippet for Python Quick-Sort.

Important note, when you run this you'll most likely need to increase Pythons standard 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

```
public class Quick {
        public static void Sort(string[] arr) {
            var n = arr.Length;
            Sort(arr, 0, n - 1);
        }
        private static void Sort(string[] arr, int left, int right) {
            if (left < right) {
                var pivot = Partition(arr, left, right);
                Sort(arr, left, pivot - 1);
                Sort(arr, pivot + 1, right);
            }
        }

        private static int Partition(string[] arr, int left, int right) {
            var pivot = arr[right];
            var i = left - 1; // Starting from last element -1
            for (var j = left; j < right; j++) {
                // If the current element is small then
                // the pivot, increment i and swap(i, j)
                if (Utils.Less(arr[j], pivot)) {
                    i++;
                    Utils.Swap(arr, i, j);

                }
            }
            // swap arr[i+1] and arr[right] (or pivot)
            Utils.Swap(arr,i+1,right);
            return i + 1;
        }
    }
```

Listing 2: Code snippet for C# Quick-Sort.

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.

Discussion and Future Work

From this experiment I've figured out how much faster the C language is, compared to other. 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.

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[1]. 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://stackoverflow.com/questions/30486513/why-is-my-quicksort-so-slow-in-python`. (accessed: 10.12.2020).

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

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