Classification results

`1716.bases`	Team Bearland	yes	0.33 s
	Tresplans	yes	5.86 s
	other-awesome-team-name	yes/no	[seconds]
`252.bases`	Team Bearland	yes	0.007 s
	Tresplans	yes	0.13 s
`3432.bases`	Team Bearland	yes	1.276 s
	Tresplans	yes	23.36 s
`4092.bases`	Team Bearland	no	0.019 s
	Tresplans	no	4.99 s
`417.bases`	Team Bearland	no	0.002 s
	Tresplans	no	0.12 s
`462.bases`	Team Bearland	yes	0.021 s
	Tresplans	yes	0.43 s
`4639.bases`	Team Bearland	no	0.397 s
	Tresplans	no	2.33 s
`6435.bases`	Team Bearland	yes	4.506 s
	Tresplans	yes	84.63 s
`924.bases`	Team Bearland	yes	0.088 s
	Tresplans	yes	1.74 s

Execution times

Team Bearland

We implemented a mask-based algorithm. Each mask m represents a set. The i-th bit of the mask is set to 1 iff i belongs to the set. We can easily, and in a fast manner, access each position shifting bits, and unions, intersections and complements are solved using bitwise OR, AND, NOT, and XOR operators. The code implemented in PYTHON3, albeit leveraging the same principle, implements the masking operations differently.

In both cases, the complexity is the same. We take advantage of the fact that all values in all inputs are integers between 0 and $r - 1 = 19$. In every case, we build the masks in $\mathcal{O}(\sum_{B \in \mathcal{B}} |B|) = \mathcal{O}(r|\mathcal{B}|)$ time. Then, for each pair $(B_1, B_2) \in \mathcal{B} \times \mathcal{B}$, we check that the exchange conditions by taking all $\mathcal{O}(r)$ values of $x \in B_1$, and checking if the condition holds for each of the $\mathcal{O}(r)$ possible values of $y \in B_2$, and we also check that $x \in B_1 \setminus B_2$ and $y \in B_2 \setminus B_1$ in $\mathcal{O}(r^2)$ time. Finally, we check that the mask for $B_1 \setminus \{x\} \cup \{y\}$ belongs to the set of masks in $\mathcal{O}(\log(|\mathcal{B}|))$ time. Therefore, the total complexity is $\mathcal{O}(r|\mathcal{B}|) + \mathcal{O}(r^2|\mathcal{B}|^2|\log(|\mathcal{B}|)) = \mathcal{O}(r^2|\mathcal{B}|^2|\log(|\mathcal{B}|))$.

Figure [fig:team-bearland-times] summarizes the execution times for both programming languages. Note that we re-ordered the results to plot them in a,

more or less, increasing fashion.

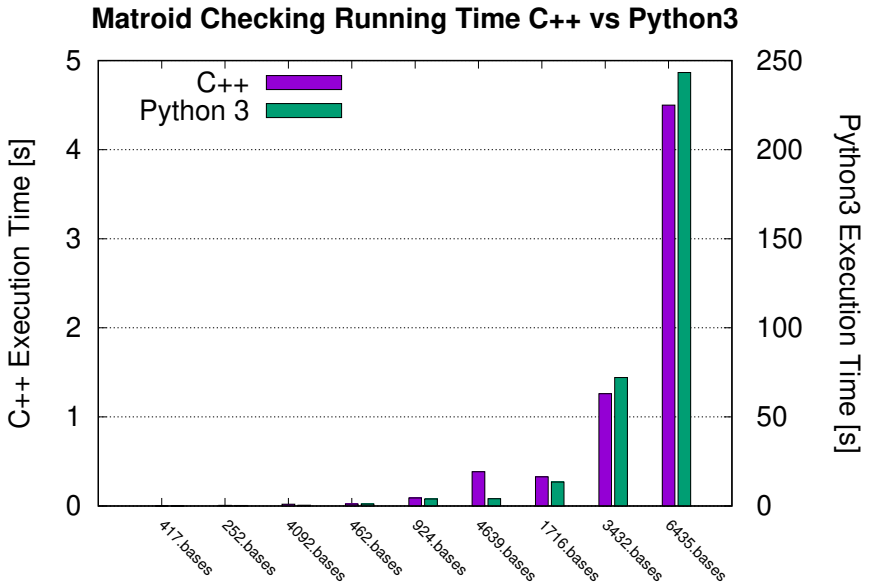

Figure 1: Execution time of the two different implementations in increasing order.[fig:team-bearland-times]

The results were to be expected, the compiled, performance-oriented, C++ implementation outruns the PYTHON3 one by two orders of magnitude. However, we do observe that the behaviour, the pattern described by execution times as we increase the load (*i.e.* the complexity) is the same for both algorithms. This is a great example not to underestimate the constants affecting our expected running time, usually disregarded by complexity theory and big $\mathcal{O}$ notation.

Tresplans

The complexity of our code is the same in both implementations. It can be computed as follows, where N is the number of bases:

- Process file: $O(N)$.
- Compare bases: $O(N^2)$.
- Compare each pair of subsets of the base: contsant time with a large constant.

Thus, overall complexity is $O(N^2)$. It is also output-sensitive, as it stops if it finds a pair of bases do not satisfy the exchange axiom.

other-awesome-team-name

another plot here