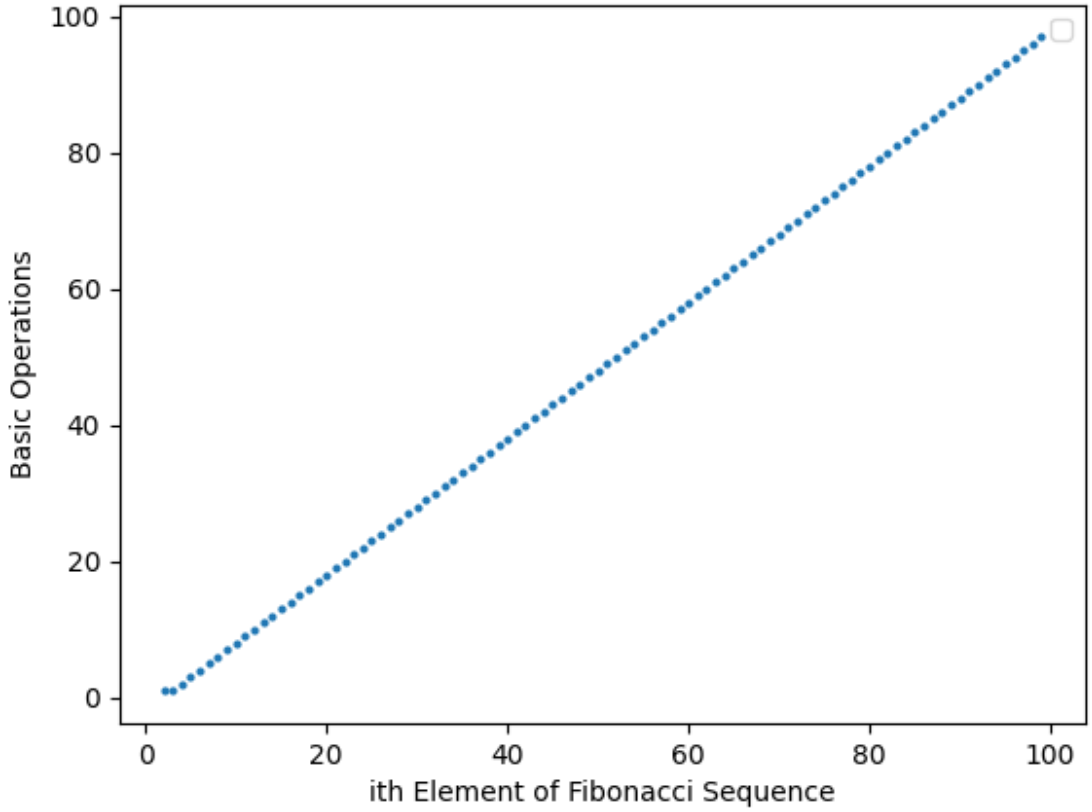
Amit Deb & Luke Demeter-Willison

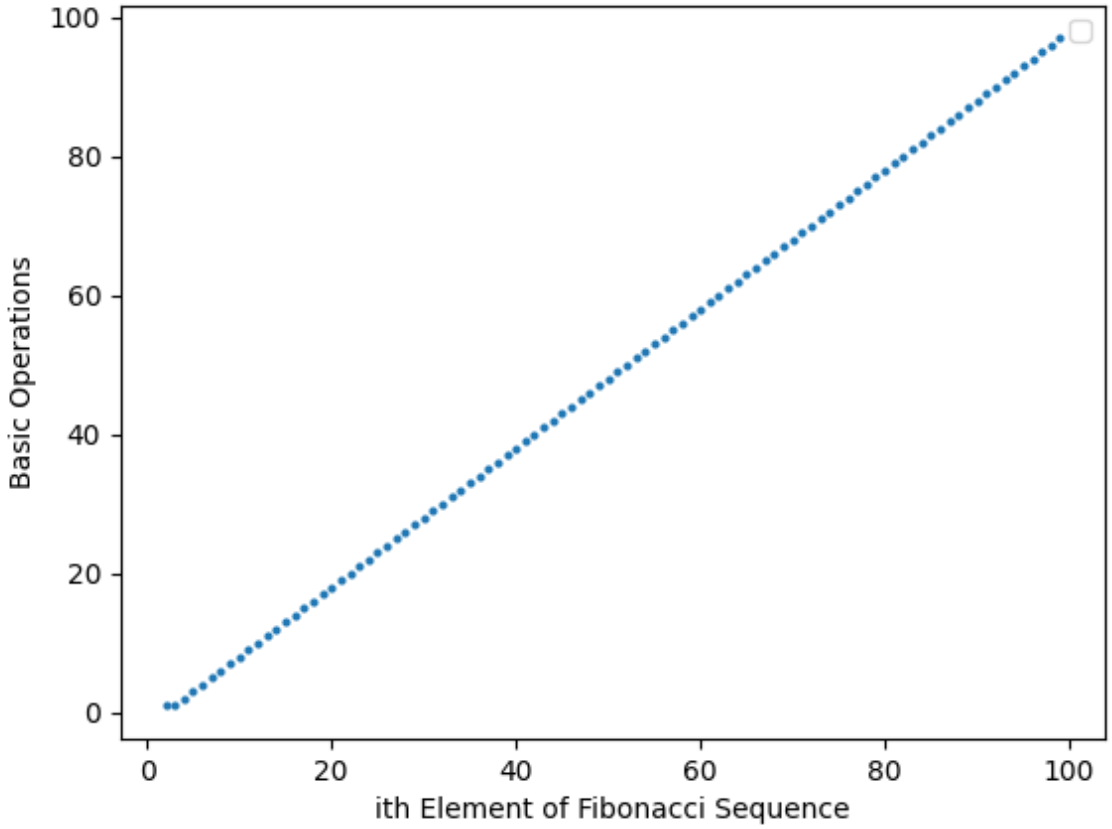
Asymptotic Analysis of Algorithms for computing GCD

Task 1a: Recursive Fibonacci



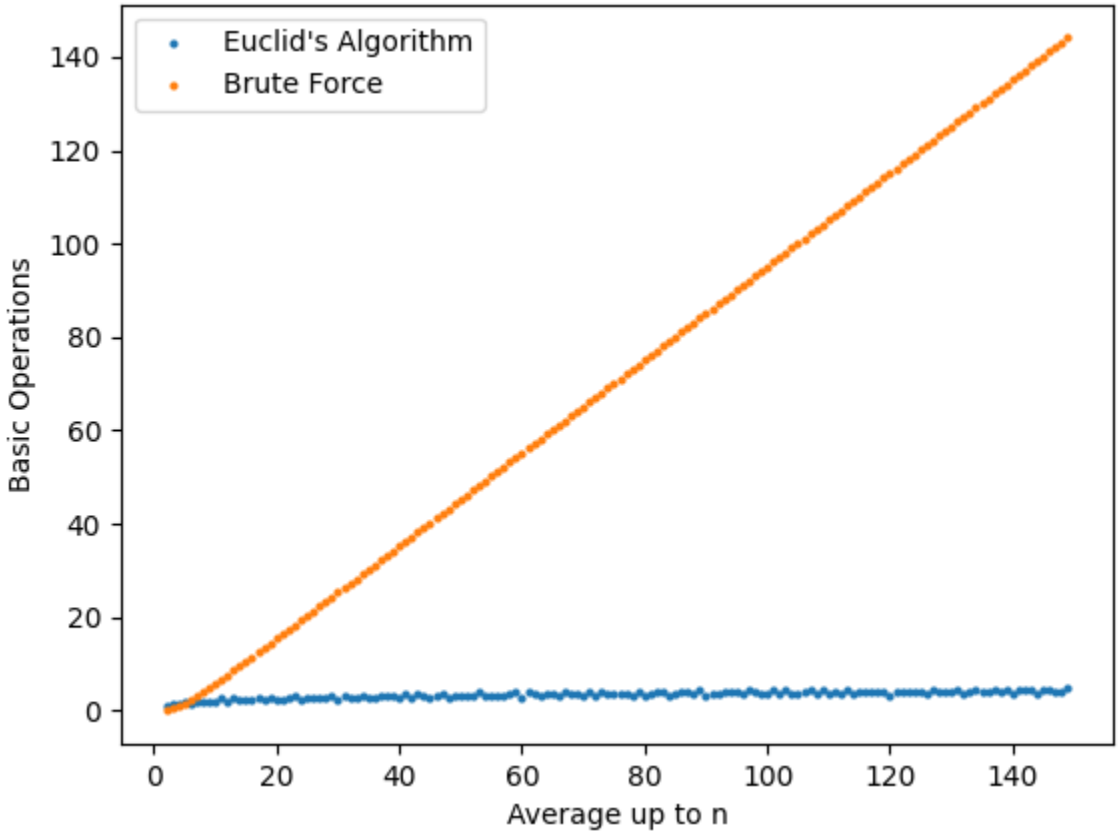
Bounds used: 0-100
Estimated time complexity: Θ(i)

Task 1b: GCD(m,n) where m = Fib(i+1), n = Fib(i)



Bounds used: 0-100
Estimated time complexity: Θ(i)

Task 2: Average-Case efficiency of Euclid's algorithm vs Brute Force



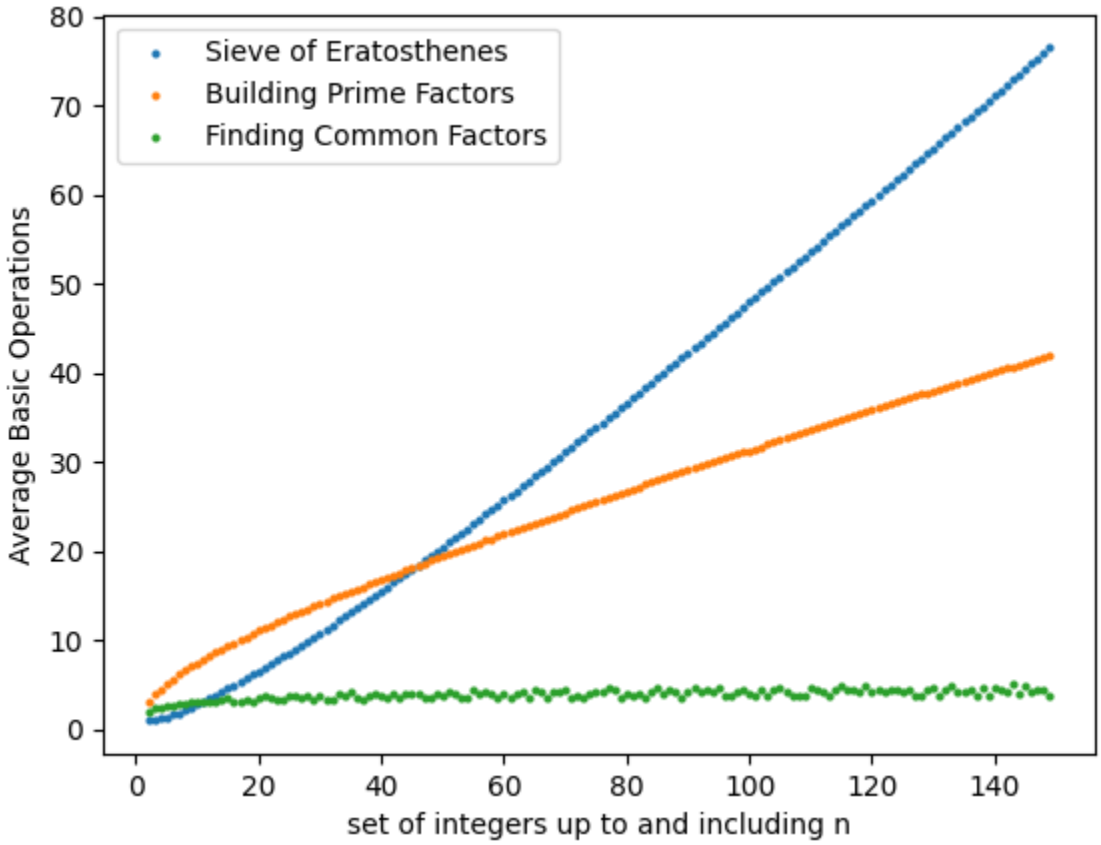
Bounds used: 2-150
Estimated time complexity:
Brute Force: **Θ**(n)
Euclid's Algorithm: **Θ**(log(n))

Notes:
Clearly, Euclid's Algorithm smokes the brute force approach. The brute force approach does an unbelievable amount of unnecessary calculations that make it unsuited as a competitor in any regard.

Because the next step of Euclid's Algorithm is less than the previous step divided by some integer >= 1, that means the parameter is divided each step. As seen in the Master's Theorem, an algorithm with its recursive parameter divided each step ends up being **Θ**(log(n)) (provided no other part of the algorithm is growing faster).

It's worth noting that the worst-case efficiency of these algorithms ends up being the same, but only by virtue of the Brute Force approach never taking advantage of anything. So, in almost any case, Euclid's Algorithm is going to be better, and even if exclusively the worst-case is being worked with, Euclid's still just as good.

Task 3: Middle-School Method



Bounds used: 2-150
Estimated time complexity:
Sieve of Eratosthenes: **Θ**(n)
Building Prime Factors: **Θ**(log(n))
Finding Common Factors: **Θ**(len(PF(min(m,n)))

Notes:
As felt obvious when implementing this algorithm, it is not efficient. There is perhaps some merit to being able to pre-calculate many parts of it, but building the list of prime numbers each time makes this unbelievably slow. As a narrative, it is a shame to be introduced to Eulcid's Algorithm so early, as its beautiful simplicity makes algorithms like this shameful to bear witness to.

The Sieve of Eratosthenes's time complexity may intuitively seem like $\sqrt{n}$, but that ignores the inner loop within the sieve that iterates throughout the multiples. This means that no matter what, every element of A is interacted with by the sieve, and given that A spans from 1..n, that makes the whole algorithm at least **Θ**(n).

The time complexity for Finding Common Factors is linear, certainly, but along a measure of input size not seen elsewhere. It's based on the number of prime factors that each value has, which, unless there is a huge breakthrough in the underlying mathematics, is not wholly predictable.

Amit Deb & Luke Demeter-Willison

Final Remarks

For all graphs, the primary balance struck when charting the axes was making the individual points distinguishable and providing enough room to see the growth rates plateau.

As a competition, Euclid's Algorithm wins handedly, as even in its worst-case it performs better than/equal to the other algorithms' average-case time complexity.