Fibonacci Numbers: A Deep Dive

Stanislav Ostapenko

October 14, 2025

Abstract

TODO

Contents

Listings

$$
\begin{aligned}
\mathcal{O}(1) &= \mathcal{O}(\text{yeah}) \\
\mathcal{O}(\log n) &= \mathcal{O}(\text{nice}) \\
\mathcal{O}(n) &= \mathcal{O}(\text{k}) \\
\mathcal{O}(n^2) &= \mathcal{O}(\text{my}) \\
\mathcal{O}(2^n) &= \mathcal{O}(\text{no}) \\
\mathcal{O}(n!) &= \mathcal{O}(\text{mg}) \\
\mathcal{O}(n^n) &= \mathcal{O}(\text{sh*t!})
\end{aligned}
$$

1 Recursion and Mathematical Induction

2 Naive Recursion

2.1 Algorithm in Java

2.2 Binet's formula

2.2.1 Intuitive Explanation

2.2.2 Formal Derivation

2.3 Time Complexity (Big $\mathcal{O}$)

2.4 Empirical Validation of Time Complexity

2.5 Introduction to JVM Memory Structures

2.6 Method Execution in Java

2.6.1 Core concepts

2.6.2 Pass-by-Value in Java

2.6.3 Tail Recursion

2.7 Depth and `StackOverflow`

2.8 Recursion Tree

2.9 Space Complexity

2.9.1 Call Stack Analysis

2.9.2 Clarification on Exponential Misconception

2.10 JVM Debugger view

2.11 Conclusion

3 Optimizing Recursion

3.1 Memoization

3.2 Dynamic Programming

3.3 Linear algebra and Fibonacci

3.3.1 Matrices and Transformations

3.3.2 Algorithm implementation in Java

3.3.3 Time and space complexity

3.4 Computer Algebra Systems

3.5 Binet formula in real life

3.6 Conclusion

4 Limitations of Primitive Types for Large Fibonacci Numbers

4.1 Showcase : from int to double

4.2 When double goes wild

4.2.1 $0.1 + 0.2 \neq 0.3$

4.2.2 IEEE 754 Representation