ANALYSIS OF ALGORITHM COMPLEXITY PROJECT

REPORT: Comparative Analysis of Algorithm Complexity

Iterative vs. Recursive Solutions for the k-Step Climbing Stairs Problem

Group Members:

HAFIZH AZRIAL – 103012440017

Course:

CAK2BAB2 - Analysis of Algorithm Complexity

PROGRAM STUDI S1 INFORMATIKA FAKULTAS INFORMATIKA UNIVERSITAS TELKOM

2025

Contents

1. Problem Description

The ***k*-Step Climbing Stairs** problem is a combinatorial optimization challenge. Given a staircase with n steps, a person can climb either $1, 2$, ..., or k steps at a time. The goal is to determine the total number of distinct ways to reach the top.

- **Input:** n (total steps), k (maximum steps per stride).
- **Output:** Total distinct paths to reach step n.
- **Mathematical Recurrence:** $T(n) = \sum_{j=1}^{k} T(n-j)$ for $n > 0$, with base case $T(0) = 1$.

2. Description of Algorithms

A. Recursive Algorithm

The recursive approach breaks the problem into sub-problems by exploring every possible last step taken (from 1 to k).

- **Logic:** To reach step n, the algorithm recursively sums the ways to reach $n-1, n-2, \dots, n-k$.
- **Technique:** Exhaustive search via a recursion tree.

B. Iterative Algorithm

The iterative approach builds the solution from the ground up, starting from step 1.

- **Logic:** It uses an array to store the number of ways to reach each step i. To calculate $[i]$, it simply sums the previously calculated values in $[i-1 \dots i-k]$.
- **Technique:** Linear iteration with a nested loop for k.

3. Complexity Analysis

Recursive Complexity

As seen in the Week 10 slides on Homogeneous Linear Recurrence Relations, the characteristic equation for this problem is:

$$r^k - r^{k-1} - r^{k-2} - \cdots - 1 = 0$$

The growth is determined by the largest root (r). As k increases, r approaches 2.

- For $k = 3, r \approx 1.839$.
- For $k = 4, r \approx 1.927$.
- For $k = 5, r \approx 1.966$.

For $k = 2$, this results in the Golden Ratio. As k increases, the base of the exponent increases.

Complexity Class: $O(k^n)$ — **Exponential Time**.

Iterative Complexity

Following the rules for **Iterative Algorithm Analysis** (Week 5 & 7):

- There is an outer loop running n times.
- There is an inner loop running k times.
- The basic operation (addition) is executed $n \times k$ times.

Complexity Class: $O(nk)$ — **Linear Time** (relative to n).

Comparison Table

Feature	**Recursive Approach**	**Iterative Approach**
Memory	Stack memory ($O(n)$)	Array memory ($O(n)$)
Time Complexity	$O(k^n)$	$O(nk)$
Scalability	Very slow for $n > 40$	Handles $n = 10{,}000 +$

4. Running Time Comparison

Experimental Determination of c_{op}

Using the approximation $T(n) \approx c_{op} \cdot C(n)$ from Week 1, we calculated the constant of operation (c_{op}) using the C++ execution data at $n = 30$. We use the "Total Ways" as a proxy for the basic operations executed.

k	Result$\left(C(n)\right)$	Measured Time $\left(T(n)\right)$	$c_{op} = T(n)/C(n)$
3	53,798,080	$20529\ ms$	3.816×10^{-4} ms
4	201,061,985	$88618\ ms$	4.407×10^{-4} ms
5	345,052,351	$186206\ ms$	5.396×10^{-4} ms

Observation: As k increases, c_{op} increases because the loop inside each recursive call becomes longer.

Running Time Comparison Table ($n = 1$ to $10{,}000$)

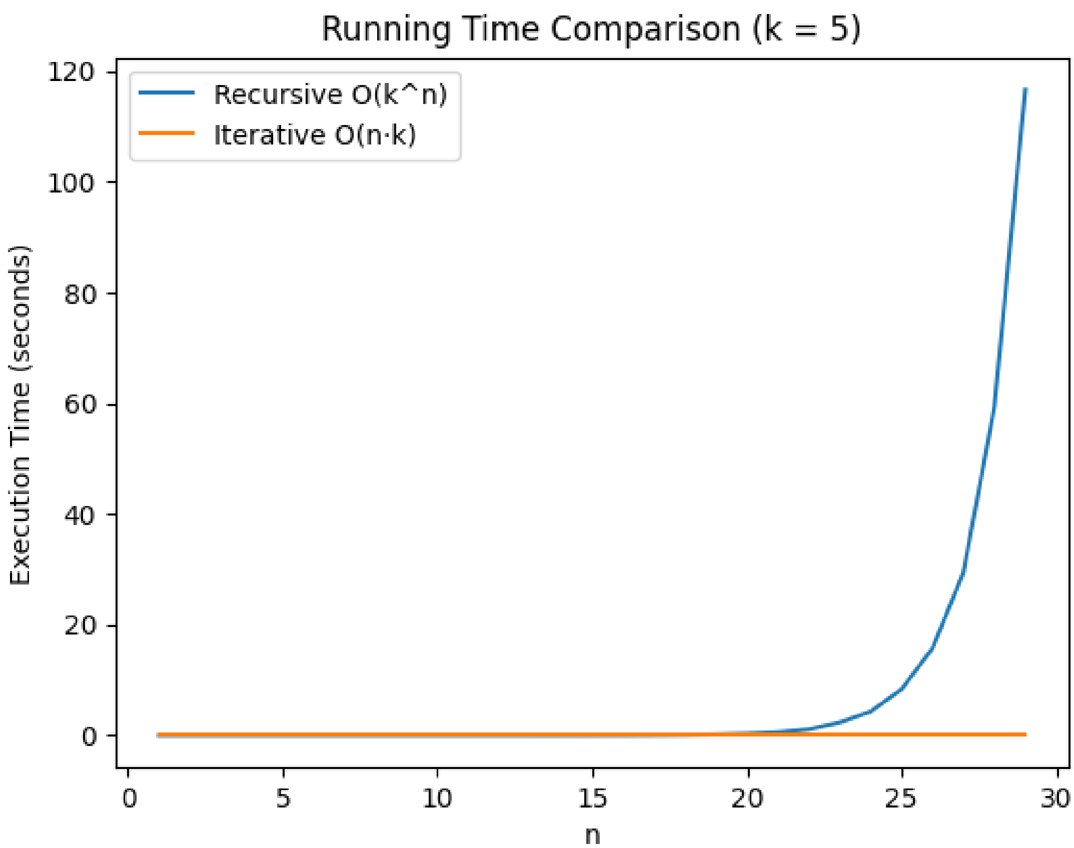
Since calculating $n = 10{,}000$ for the recursive algorithm is impossible, we use our c_{op} to provide a mathematical estimation.

Case Study: $k = 5$

Input Size (n)	Iterative Time	Recursive Time
1	$0.236711\ ms$	$0.000463701\ ms$
10	$0.24191\ ms$	$0.25318\ ms$
20	$0.239018\ ms$	$209\ ms$
30	$0.248546\ ms$	$186206\ ms$
40	$0.253222\ ms$	$*1.60 \times 10^{8}\ ms$
50	$0.25559\ ms$	$*1.38 \times 10^{11}\ ms$
100	$0.285518\ ms$	$*6.61 \times 10^{25}\ ms$
500	$0.544313\ ms$	$*1.77 \times 10^{143}\ ms$
1000	$2\ ms$	$*1.08 \times 10^{290}\ ms$
5000	$11\ ms$	$*2.10 \times 10^{1468}\ ms$
10000	$39\ ms$	$*1.52 \times 10^{2941}\ ms$

*Estimated time calculated using $T(n) \approx c_{op} \cdot C(n)$

Graph Visualization with Python3 For $n \leq 30$ and $k = 5$



5. Calculation Steps

This section details the derivation of the constant of operation (c_{op}) and the estimated running times for the recursive algorithm.

1. Calculation of Constant of Operation ($\boldsymbol{c_{op}}$)

The constant of operation represents the time required to execute a single recursive step. It is calculated using the formula:

$$c_{op} = \frac{T(n)}{C(n)}$$

Where $T(n)$ is the measured execution time at $n = 30$ and $C(n)$ is the total number of operations (ways) at $n = 30$.

- **For $\boldsymbol{k = 3}$:**

  $$T(30) = 20{,}529\ ms,\ C(30) = 53{,}798{,}080,\ c_{op} = \frac{20{,}529}{53{,}798{,}080} \approx 3.816 \times 10^{-4}\ \boldsymbol{ms}$$

- **For $\boldsymbol{k = 4}$:**

  $$T(30) = 88{,}618\ ms, C(30) = 201{,}061{,}985, c_{op} = \frac{88{,}618}{201{,}061{,}985} \approx 4.407 \times 10^{-4}\ \boldsymbol{ms}$$

- **For $\boldsymbol{k = 5}$:**

  $$T(30) = 186{,}206\ ms, C(30) = 345{,}052{,}351, c_{op} = \frac{186{,}206}{345{,}052{,}351} \approx 5.396 \times 10^{-4}\ \boldsymbol{ms}$$

2. Estimated Recursive Running Times ($\boldsymbol{T_{est}}$) for $\boldsymbol{k = 5}$

Using the specific constant $c_{op} \approx 5.396 \times 10^{-4}\ ms$, we estimate the time for larger inputs using:

$$T_{est}(n) = c_{op} \times C(n)$$

- For $n = 40$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (2.97 \times 10^{11}) \approx 1.60 \times 10^{8}\ \boldsymbol{ms}$$

(Approx. 44.6 hours)

- For $n = 50$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (2.56 \times 10^{14}) \approx 1.38 \times 10^{11}\ \boldsymbol{ms}$$

(Approx. 4.4 years)

- For $n = 100$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (1.22 \times 10^{29}) \approx 6.61 \times 10^{25}\ \boldsymbol{ms}$$

- For $n = 500$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (3.28 \times 10^{146}) \approx 1.77 \times 10^{143}\ \boldsymbol{ms}$$

- For $n = 1000$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (2.00 \times 10^{293}) \approx 1.08 \times 10^{290}\ \boldsymbol{ms}$$

- For $n = 5000$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (3.90 \times 10^{1471}) \approx 2.10 \times 10^{1468}\ \boldsymbol{ms}$$

- For $n = 10000$:

$$T_{est} \approx (5.396 \times 10^{-4}) \times (2.82 \times 10^{2944}) \approx 1.52 \times 10^{2941}\ \boldsymbol{ms}$$

6. Conclusion

The analysis confirms that the iterative solution is vastly superior for the k-Step Climbing Stairs problem. While recursion is easier to implement and mathematically intuitive, the redundant calculation of sub-problems leads to exponential time growth ($O(k^n)$), making it infeasible for real-world applications. The iterative approach, reduces the complexity to $O(nk)$, proving the importance of algorithm design in performance.

7. Tools Used and References

Tools

- Zed Code Editor with Codex CLI
- Python 3.14 with matplotlib
- C++ 17 with GCC and optionally with boost package(boost.org)
- Microsoft Word for text and LaTeX formatting

References

- Course Slides: Weeks 1-13, Analysis of Algorithm Complexity, Telkom University.
- AfterAcademy: Climbing Stairs Problem Complexity Analysis.
- GitHub repository containing all of the codes. Source = https://github.com/includeMeXD/algorithmComplexityForKStepClimbingStairs