Big-O Study Note

Name：黃以蔡

Student ID：1133313

Date: 9 / 16

Big-O Study Note

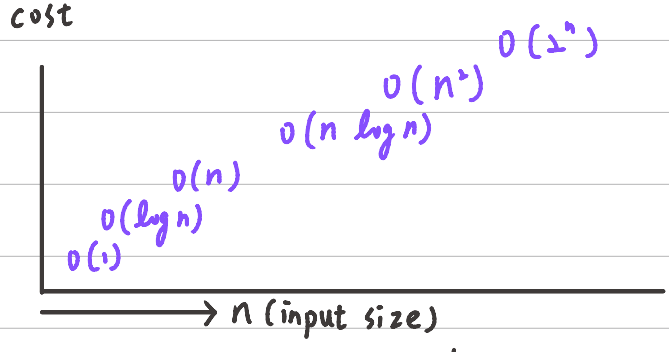
1. Definition

Big-O notation is a mathematical way to describe the asymptotic growth rate of an algorithm's time or space usage as the input size n becomes large.

It sloves the problem of comparing algorith efficiency without depending on hardware, programming language, or constant factors.

2. Visualization

Common Big-O growth curves:



A steeper curve means high complexity.

3. charaterristics

- Type : Describes asymtotic behavior of algorithms
- Focus : Usually expresses worst-case performance
- Ignores constants : O(2n) and O(100n) both simplify to O(n)
- Used for comparison : Helps determine which algorithm scales better
- Related notations : Θ (tight bound) . Ω (best case)

4. Limitations

- Doesn't reflect real execution time
- Not accurate for small input size
- Only describes worst-case unless paired with Θ/Ω
- High Big-O doesn't always mean slow in real case.

Date: /

5. Time / Space Complexity Table

Complexity	Name	Example
$O(1)$	Constant	Array index access
$O(\log n)$	Logarithmic	Binary search
$O(n)$	Linear	Loop through array
$O(n \log n)$	Linearithmic	Merge sort, Heap sort
$O(n^2)$	Quadratic	Double nested loops
$O(n^3)$	Cubic	Triple nested loops
$O(2^n)$	Exponential	Subset enumeration
$O(n!)$	Factorial	Brute-force TSP

Simplification Rules

- Drop constants : $O(3n+10) \to O(n)$
- keep the dominant term : $O(n^2+n) \to O(n^2)$
- Nested loops multiply : two loops $O(n \times n) = O(n^2)$
- Sequential steps take the maximum : $O(n) + O(\log n) = O(n)$

Date: /

6. Pros / Cons

Pros

- Standardized way to compare algorithm efficiency
- Helps evalute long-term scalability
- Independent of machine or implementation details
- Core tool in data structures . algorithm analysis

Cons :

- Ignores constant factors that may matter in practice
- Same Big-O can behave differently in real code
- Doesn't model CPU can effects, memory locality
- Only describes asymptotic behavior, not real timing

7. Use Cases

1. Comparing two algorithms
2. Analyzing data structure operations
3. Evaluating scalability