Asymptotic Analysis

Advanced Notes for Competitive Programming

December 22, 2025

Contents

1 Complexity Analysis for Competitions

1.1 Time Limits and Constraints

Constraint Analysis

Rule of thumb: Modern judges execute $\sim 10^8$ to 10^9 simple operations per second.
Safe complexity bounds per time limit:

Time Limit	Max n	Complexity
1 sec	10^8	$O(n), O(n \lg n)$
1 sec	10^7	$O(n \lg n)$
1 sec	10^4	$O(n^2)$
1 sec	500	$O(n^3)$
1 sec	100	$O(n^4)$
1 sec	25	$O(2^n)$
1 sec	11	$O(n!)$

Competitive Trick

Given constraint, choose algorithm:

- $n \leq 10$: $O(n!)$, $O(2^n \cdot n^2)$ acceptable
- $n \leq 20$: $O(2^n)$, $O(n^2 \cdot 2^n)$
- $n \leq 100$: $O(n^4)$
- $n \leq 500$: $O(n^3)$
- $n \leq 10^4$: $O(n^2)$
- $n \leq 10^6$: $O(n \lg n)$
- $n \leq 10^8$: $O(n)$, $O(\lg n)$

1.2 Exact Operation Counts

Don't just know Big-O — know the constant!

Optimization

Examples of hidden constants:

- Merge sort: $\sim 2n \lg n$ comparisons
- Quick sort (avg): $\sim 1.39n \lg n$ comparisons
- Heap sort: $\sim 2n \lg n$ comparisons
- std::sort (C++): Hybrid, $\sim 1.5n \lg n$ on average

If time limit is tight, $O(n \lg n)$ algorithms can differ by 2x in practice!

2 Advanced Asymptotic Relations

2.1 Key Inequalities for Bounding

For proving upper/lower bounds:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$
$$\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$
$$\sum_{i=0}^{\lg n} 2^i = 2^{\lg n+1} - 1 = 2n - 1 = \Theta(n)$$
$$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1} = \Theta(r^n) \quad (r > 1)$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\lg n)$$

Competitive Trick

Telescoping sums: If loop counter changes by division/multiplication:

$$\text{Iterations} = \lg(\text{range})$$

Example: for i=1 to n, i*=2 $\rightarrow \Theta(\lg n)$ iterations

2.2 Master Theorem (Extended)

For recurrences $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$:
Let $c_{crit} = \log_b a$
Case 1: If $f(n) = O(n^c)$ for some $c < c_{crit}$, then

$$T(n) = \Theta(n^{c_{crit}})$$

Case 2: If $f(n) = \Theta(n^{c_{crit}} \log^k n)$ for some $k \geq 0$, then

$$T(n) = \Theta(n^{c_{crit}} \log^{k+1} n)$$

Case 3: If $f(n) = \Omega(n^c)$ for some $c > c_{crit}$, and $af(n/b) \leq \delta f(n)$ for some $\delta < 1$ and large n, then

$$T(n) = \Theta(f(n))$$

Optimization

Common applications:

$$T(n) = 2T(n/2) + O(n) = O(n \lg n) \quad \text{(Merge sort)}$$
$$T(n) = 2T(n/2) + O(1) = O(n) \quad \text{(Binary search on sorted array)}$$
$$T(n) = T(n/2) + O(1) = O(\lg n) \quad \text{(Binary search)}$$
$$T(n) = 2T(n/2) + O(n^2) = O(n^2)$$
$$T(n) = 8T(n/2) + O(n^2) = O(n^3)$$

2.3 Amortized Analysis Techniques

Three methods:

1. **Aggregate method:** Total cost / number of operations
2. **Accounting method:** Assign costs to operations, some pay for future ops
3. **Potential method:** Define potential function Φ, amortized cost = actual cost + $\Delta\Phi$

Competitive Trick

Classic example: Dynamic array doubling
Each insertion: $O(1)$ amortized, despite occasional $O(n)$ resize
Proof (aggregate):

- After n insertions, total cost $= n + (1 + 2 + 4 + \cdots + 2^{\lg n})$
- Geometric series sums to $2n - 1 = O(n)$
- Amortized: $O(n)/n = O(1)$

3 Logarithm Tricks and Identities

3.1 Essential Identities

$$\begin{aligned}
\log(ab) &= \log a + \log b \\
\log(a/b) &= \log a - \log b \\
\log(a^k) &= k \log a \\
a^{\log_b c} &= c^{\log_b a} \\
\log_b a &= \frac{\log_c a}{\log_c b} = \frac{1}{\log_a b} \\
b^{\log_b a} &= a
\end{aligned}$$

Optimization

Bit manipulation shortcut:

$$\lg n = \lfloor \log_2 n \rfloor = \text{MSB position}$$

In C++: `__builtin_clz(n)` gives leading zeros, so $\lg n \approx 31 -$ `__builtin_clz(n)`

3.2 Approximations

For positive x:

$$\begin{aligned}
\ln(1+x) &\approx x - \frac{x^2}{2} \quad (|x| < 1) \\
e^x &\approx 1 + x + \frac{x^2}{2} \quad (|x| \ll 1) \\
(1+x)^n &\approx 1 + nx \quad (|nx| \ll 1)
\end{aligned}$$

Stirling's approximation:

$$\ln(n!) = n \ln n - n + O(\ln n) = \Theta(n \ln n)$$

More precisely:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Competitive Trick

For computing large binomial coefficients:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Use logarithms to avoid overflow:

$$\log \binom{n}{k} = \log(n!) - \log(k!) - \log((n-k)!)$$

4 Optimization Techniques

4.1 Constant Factor Optimizations

Optimization

1. **Bit operations over arithmetic:**
   - $n * 2 \to n \ll 1$
   - $n/2 \to n \gg 1$
   - $n \bmod 2^k \to n\&((1 \ll k) - 1)$
2. **Minimize memory accesses:**
   - Cache locality matters: access arrays sequentially
   - 2D arrays: row-major order (C++) or column-major (Fortran)
3. **Avoid function calls in tight loops:**
   - Inline functions
   - Macro expansion (use carefully)

4.2 Algorithmic Optimizations

Competitive Trick

Reduce complexity class:
Problem: Find pair summing to target in array
Naive: $O(n^2)$ - check all pairs
Optimized: $O(n \lg n)$ - sort + two pointers OR $O(n)$ - hash set

Problem: Range sum queries
Naive: $O(n)$ per query
Optimized: $O(n)$ precomputation + $O(1)$ per query (prefix sums)

4.3 Space-Time Tradeoffs

Optimization

Common patterns:
1. **Memoization / DP:**
   - Time: $O(\text{exponential}) \to O(\text{polynomial})$
   - Space: $O(1) \to O(\text{state space})$
2. **Precomputation:**
   - Compute once, query many times
   - Example: Factorials mod p, prefix sums, LCA preprocessing
3. **Hash tables:**
   - Time: $O(n) \to O(1)$ lookup
   - Space: $O(1) \to O(n)$

5 Analysis of Common Algorithms

5.1 Sorting

Algorithm	Best	Avg	Worst
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
Quick Sort	$O(n \lg n)$	$O(n \lg n)$	$O(n^2)$
Heap Sort	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
Counting Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$
Radix Sort	$O(d(n + k))$	$O(d(n + k))$	$O(d(n + k))$

Common Pitfall

Quick sort worst case:
Occurs on already sorted input if pivot is always first/last element.
Fix: Randomized pivot selection $\rightarrow$ expected $O(n \lg n)$

5.2 Graph Algorithms

Let V = vertices, E = edges

Algorithm	Time	Space
BFS	$O(V + E)$	$O(V)$
DFS	$O(V + E)$	$O(V)$
Dijkstra (binary heap)	$O((V + E) \lg V)$	$O(V)$
Dijkstra (Fib heap)	$O(E + V \lg V)$	$O(V)$
Bellman-Ford	$O(VE)$	$O(V)$
Floyd-Warshall	$O(V^3)$	$O(V^2)$
Kruskal	$O(E \lg E)$	$O(V)$
Prim (binary heap)	$O((V + E) \lg V)$	$O(V)$
Topological Sort	$O(V + E)$	$O(V)$

Competitive Trick

When to use which shortest path:

- Single source, non-negative: Dijkstra
- Single source, negative edges: Bellman-Ford
- All pairs, dense graph: Floyd-Warshall
- All pairs, sparse graph: Dijkstra from each vertex

5.3 Data Structures

Structure	Access	Search	Insert/Delete
Array	$O(1)$	$O(n)$	$O(n)$
Sorted Array	$O(1)$	$O(\lg n)$	$O(n)$
Linked List	$O(n)$	$O(n)$	$O(1)^*$
Stack/Queue	-	-	$O(1)$
Hash Table	-	$O(1)$ avg	$O(1)$ avg
BST (balanced)	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
Heap	$O(1)$ min	-	$O(\lg n)$
Segment Tree	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
Fenwick Tree	-	$O(\lg n)$	$O(\lg n)$

* Given pointer to position

6 Common Pitfalls and Edge Cases

Common Pitfall

1. Integer overflow:
When $n \approx 10^9$, n^2 overflows 32-bit int!
Solutions:

- Use 64-bit: `long long` in C++
- Modular arithmetic when appropriate

Common Pitfall

2. Off-by-one errors in complexity:
$\sum_{i=0}^{n-1}$ vs $\sum_{i=1}^{n}$ - both are $O(n)$, but exact count differs by 1
Critical when time limit is tight!

Common Pitfall

3. Hidden logarithmic factors:
`std::set::find()`, `std::map::operator[]` are $O(\lg n)$, not $O(1)$!
Loop with set operations: actually $O(n \lg n)$, not $O(n)$

Common Pitfall

4. Amortized vs worst-case:
`std::vector::push_back()` is $O(1)$ amortized, but $O(n)$ worst-case
If real-time guarantees needed, consider `std::deque` or pre-reserve

7 Advanced Bounds

7.1 Comparison-Based Sorting Lower Bound

Constraint Analysis

Theorem: Any comparison-based sorting algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.
Proof idea: Decision tree has $n!$ leaves, height $\geq \lg(n!) = \Theta(n \lg n)$

Implication: To beat $O(n \lg n)$, must use non-comparison-based algorithm (counting sort, radix sort) with assumptions about input.

7.2 3SUM Lower Bound Conjecture

Constraint Analysis

3SUM Problem: Given array, find three elements summing to 0.
Best known: $O(n^2)$
Conjecture: No $O(n^{2-\epsilon})$ algorithm exists (for any $\epsilon > 0$)
Many problems are 3SUM-hard, meaning they're at least as hard as 3SUM.

8 Final Competitive Tips

Competitive Trick

1. Quick estimation:
Input size $\rightarrow$ complexity $\rightarrow$ algorithm choice
This should take ¡ 10 seconds in your head!
2. Worst-case analysis:
Always analyze worst case unless problem explicitly asks for average
3. Constant factors:
If $O(n \lg n)$ TLEs, try optimizing constant (e.g., iterative vs recursive)
4. Space limits:
10^6 ints $\approx$ 4 MB, 10^6 longs $\approx$ 8 MB
5. Pre-written templates:
Have complexity analysis of your library code memorized
6. Profile, don't guess:
If optimization needed, profile to find bottleneck