Analysis of Algorithm

Short Notes based on ***Exit Exam***

Zemzem labs

1. Introduction to Algorithm Analysis

What is Algorithm Analysis?

The process of evaluating an algorithm in terms of:

- **Time complexity** (speed)
- **Space complexity** (memory usage)

Why Analyze?

- Predict performance
- Compare algorithms
- Choose the best approach

2. Asymptotic Notations

Describes algorithm performance as input size **n** $\to \infty$.

1. Big-O (O) — Upper Bound

Worst-case time.
Example: Bubble sort $\to O(n^2)$

2. Big-Ω (Ω) — Lower Bound

Best-case time.
Example: Linear search best $\to \Omega(1)$

3. Big-Θ (Θ) — Tight Bound

Exact / average-case.
Example: Merge sort $\to \Theta(n \log n)$

4. Little-o (o) & Little-ω (ω)

Strict inequalities; used in theoretical proofs.

3. Growth Rates (Order of Complexity)

List from fastest to slowest:

1. **O(1)** — constant
2. **O(log n)** — logarithmic
3. **O(n)** — linear
4. **O(n log n)**
5. **O(n²)**
6. **O(n³)**
7. **O(2ⁿ)** — exponential
8. **O(n!)** — factorial

Important Rules

- Ignore constants: 5n → O(n)
- Ignore lower-order terms: n² + n → O(n²)

4. Recurrence Relations

Used to analyze **recursive algorithms**.

Common Methods

1. **Substitution Method**
2. **Recursion Tree Method**
3. **Master Theorem**

5. Master Theorem

Used for divide-and-conquer recurrences:

T(n) = aT(n/b) + f(n)

Cases

1. **If f(n) = O(n^c) and a > b^c**
   → T(n) = Θ(n^log_b(a))
2. **If f(n) = Θ(n^c * log^k(n)) and a = b^c**
   → T(n) = Θ(n^c * log^{k+1}(n))
3. **If f(n) = Ω(n^c) and a < b^c**
   AND regularity condition holds
   → T(n) = Θ(f(n))

Examples

- Merge Sort: $T(n) = 2T(n/2) + n \rightarrow \Theta(n \log n)$
- Binary Search: $T(n) = T(n/2) + 1 \rightarrow \Theta(\log n)$

6. Amortized Analysis

Used when occasional expensive operations exist.

Methods

- **Aggregate analysis**
- **Accounting method**
- **Potential method**

Examples

- Stack with dynamic array resizing
- Incrementing binary counters
- Hash table insertion

7. Time Complexity of Common Operations

Array

- Access: O(1)
- Insert/Delete at middle: O(n)

Linked List

- Access: O(n)
- Insert/Delete at head: O(1)

Stack/Queue

- All operations: O(1)

BST

- Average: O(log n)
- Worst (skewed): O(n)

Hash Table

- Average: O(1)
- Worst: O(n) (many collisions)

8. Sorting Algorithm Complexities

Algorithm	Best	Average	Worst	Space
Bubble	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)

9. Complexity Analysis Patterns

1. Nested Loops

Multiply work inside loops.

```
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    // work
```

Complexity = $O(n^2)$

2. Logarithmic Pattern

Divide input by a constant.

Examples:

- Binary search → O(log n)
- Tree height operations

3. Linear Pattern

Single loop → O(n)

4. Divide & Conquer

Divide problem into smaller parts.

Examples:

- Merge sort → O(n log n)
- Quick sort → O(n log n)

5. Exponential Pattern

Recursive calls expanding.

Example:

$T(n) = 2T(n-1) + O(1)$

→ $O(2^n)$

10. Space Complexity

Measures memory usage.

Categories

- Input space
- Auxiliary space
- Recursion stack space

Examples:

- Iterative factorial → O(1)
- Recursive factorial → O(n)

11. Best, Worst & Average Case

Example: Linear Search

- Best → O(1)
- Worst → O(n)
- Average → O(n/2) = O(n)

Example: Binary Search

- Best → O(1)
- Worst → O(log n)
- Average → O(log n)

12. NP, NP-Complete, NP-Hard (Basic AoA View)

P

Problems solvable in polynomial time.

NP

Solutions verifiable in polynomial time.

NP-Complete

Hardest problems in NP.
Examples:

- SAT
- 3-SAT
- Clique

NP-Hard

At least as hard as NP-Complete.

13. Algorithm Design Techniques

Greedy

Local best → global best (Huffman, Prim, Kruskal)

Divide and Conquer

Split–Solve–Combine (Merge Sort, Quick Sort)

Dynamic Programming

Overlapping subproblems (Knapsack, LCS)

Backtracking

Search all possibilities (N-Queens)

Branch & Bound

Prunes search space (TSP)

✨ Ultra-Condensed Exam Summary

- Asymptotic notations describe growth rate
- Big-O = worst case; Θ = average case; Ω = best case
- Master theorem solves divide & conquer recurrences
- Arrays: O(1) access; Linked list: O(n) access
- Merge sort = O(n log n), Quick sort worst = $O(n^2)$
- BFS/DFS = O(V + E)
- Greedy, DP, Recursion, Divide & Conquer = core design strategies