Algorithm and Problem Solving Quick Guide in C++

Data Structures, Algorithms and Coding Interview Problem Patterns in C++

rfdavid, 2025

rfdavid.com

INTRODUCTION

Motivation

The tech industry hiring standard is based on algorithm and data structure.

There are plenty of free resources available around algorithms and data structures. The purpose of this project is to be a quick guide where you can learn and review algorithms and data structures.

Some of the intended **key features:**

- Non-verbose, short-structured, and easy to follow descriptions
- Slide-based, practical for reviewing
- Free and open-source

⭐ **If you like, please add a star at *github.com/rfdavid/cpp-algo-cheatsheet***

How do I created and use this document?

Creating this document was a great exercise to learn and review data structures and algorithms.
I encourage anyone to create its own notes

- Have sections based on the type of the problems (string, array, tree, dynamic programming)
- Start solving the problems randomly on Leetcode
- Create one slide for each question, including: problem, solution, code
- Write with your own words
- If you are not confident you have fully captured a problem, write the problem and leave the solution and code to another day when you redo it.

⭐ **If you like, please add a star at *github.com/rfdavid/cpp-algo-cheatsheet***

Some Useful Links

Tech Interview Handbook
https://www.techinterviewhandbook.org
A very well-structured resource for interview preparation

Blind 75 Leetcode Questions
https://leetcode.com/discuss/general-discussion/460599/blind-75-leetcode-questions

Common problems

- Blind 75 is a popular list of algorithm problems that intends to cover the main data structures and patterns.
- It is a curated list of 75 popular coding questions created by an ex-Meta Staff Engineer

Array

✓ Two Sum

✓ Contains Duplicate

✓ Product of Array Except Self

✓ Best Time to Buy and Sell Stock

✓ Maximum Subarray

✓ Maximum Product Subarray

✓ Find Minimum in Rotated Sorted Array

✓ Search in Rotated Sorted Array

✓ 3 Sum

✓ Container With Most Water

Binary

✓ Sum of Two Integers

✓ Number of 1 Bits

✓ Counting Bits

✓ Missing Number

✓ Reverse Bits

Dynamic Programming

✓ Climbing Stairs

Coin Change

Longest Increasing Subsequence

✓ Longest Common Subsequence

Word Break

Combination Sum

✓ House Robber

House Robber II

Decode Ways

✓ Unique Paths

Jump Game

Matrix

✓ Set Matrix Zeroes

✓ Spiral Matrix

✓ Rotate Image

Word Search

Common Problems

Tree

- ✓ Maximum Depth of Binary Tree
- ✓ Same Tree
- ✓ Invert/Flip Binary Tree
- ✓ Path Sum
- ✓ Binary Tree Level Order Traversal
- ✓ Serialize and Deserialize Binary Tree
- ✓ Subtree of Another Tree
- ✓ Construct Binary Tree from Preorder and Inorder Traversal
- ✓ Validate Binary Search Tree
- ✓ Kth Smallest Element in a Binary Search Tree
- Lowest Common Ancestor of Binary Search Tree
- ✓ Implement Trie (Prefix Tree)
- Add and Search Word
- Word Search II
- Balanced Binary Tree

Heap

- ✓ Top K Frequent Elements
- Find Median from Data Stream

String

- ✓ Longest Substring Without Repeating Characters
- ✓ Longest Repeating Character Replacement
- ✓ Minimum Window Substring
- ✓ Valid Anagram
- ✓ Group Anagrams
- ✓ Valid Parentheses
- ✓ Valid Palindrome
- Longest Palindromic Substring
- Palindromic Substrings
- Encode and Decode Strings ⭐

Linked List

- ✓ Reverse a Linked List
- ✓ Detect Cycle in a Linked List
- ✓ Merge Two Sorted Lists
- ✓ Merge K Sorted Lists
- ✓ Remove Nth Node From End Of List
- ✓ Reorder List

Graph

- ✓ Clone Graph
- ✓ Course Schedule
- ✓ Pacific Atlantic Water Flow
- ✓ Number of Islands
- ✓ Longest Consecutive Sequence
- Alien Dictionary ⭐
- ✓ Graph Valid Tree ⭐
- ✓ Number of Connected Components In an Undirected Graph ⭐

Interval

- ✓ Insert Interval
- ✓ Merge Intervals
- ✓ Non-overlapping Intervals
- ✓ Meeting Rooms ⭐
- Meeting Rooms II ⭐

⭐ = leetcode premium

Other problems

- ✓ Maximum Level Sum of a Binary Tree
- ✓ Minimum Number of Increments on Subarrays to Form a Target Array
- ✓ Leaf-Similar Trees
- ✓ Count Good Nodes in Binary Tree
- Min Cost Climbing Stairs
- Longest Palindromic Subsequence
- ✓ Minimum Cost for Tickets
- ✓ Webcrawler
- ✓ Network Delay Time
- ✓ Rotated Digits
- ✓ Ransom Note
- ✓ String to Integer (atoi)
- ✓ Middle of the Linked List

⭐ = leetcode premium

Time Complexity

If the input gets bigger, how many steps does the algorithm take?

- Measure of how much the execution time of an algorithm grows relative to the size of its input (usually called n)
- Expressed in **Big-O notation** (e.g. $O(1)$, $O(n)$, $O(n^2)$, etc) to describe the **upper bound** of how fast the algorithm's runtime grows
- Asymptotic notations

  Big-O (O) – Upper Bound (**Worst-case**): Describes the maximum amount of time/memory an algorithm could take.

  Theta Θ – Tight Bound (**Exact**): describes both the **upper** and **lower bound** (the **exact** growth rate)

  Omega (Ω) – Lower Bound (**Best-case**): describes the minimum time/space the algorithm needs.

Time Complexity

Examples

O(1)

Examples	**Problems**
Accessing an array element (`arr[i]`)	Hash table lookups
Swapping two variables	Checking if a number is even/odd
Stack/Queue `push` or `pop`	Returning first element of a list

O(log n)

Binary Search	Search in sorted array
Balanced BST insert/find (AVL, Red-Black Tree)	Find k-th smallest in BST
Finding floor/ceil in sorted array	Finding square root with binary search

O(n)

Linear Search	Maximum subarray sum (Kadane's algo)
Finding min/max in an array	Counting frequencies with hash map
Traversing linked list or array	One-pass string processing problems

Time Complexity

$O(n \log n)$

Merge Sort / Heap Sort	Sorting an array
Efficient algorithms for Closest Pair	Finding inversion count in array
Heapify operations	Kth largest element with heap

$O(n^2)$

Bubble Sort / Insertion Sort	Two Sum (brute force)
Checking all pairs in array	Longest Palindromic Substring (DP)
Floyd-Warshall algorithm	Edit Distance (DP)

$O(n^3)$

Matrix multiplication (naive)	Boolean matrix multiplication
DP on subsequences of length 3	Some DP path-finding problems
Floyd-Warshall for dense graphs	Counting triangles in graph

$O(2^n)$

Recursive Fibonacci (no memo)	Subset sum (brute force)
Backtracking for combinations/permutations	N-Queens
Traveling Salesman (brute force)	All subsets of array (power set)

Time Complexity

V = vertices (nodes)
E = edges

O(n!)

Generating all permutations	Traveling Salesman (brute force)
Brute-force anagram check	Word ladder with all transformations
Solving puzzles with all arrangements	Hamiltonian Path

O(V + E)

DFS / BFS (adjacency list)	DFS / BFS (adjacency list)

O(E log V)

Dijkstra with priority queue	Dijkstra with priority queue

O(VE)

Bellman-Ford

O(V^3)

Floyd-Warshall

O(E log E)

Kruskal's MST algorithm

Space Complexity

As the input size n grows, how much extra memory does the algorithm need to run?

- Measure of how much memory an algorithm uses relative to **input size**
- Expressed in **Big-O notation** (e.g. $O(1)$, $O(n)$, $O(n^2)$, etc)
- It includes auxiliary space (extra memory used by the algorithm, not counting the input itself) and sometimes considers input space depending on the context
- Count only extra space needed **(exclude output)**
- The space complexity of a recursive tree traversal is **O(h),** where h is the height of the tree. This is because each recursive call adds a frame to the call stack, and in the worst case, the maximum stack depth is proportional to the tree's height

Space Complexity

Examples

Algorithm / Operation	Space Complexity	Explanation
Swap two integers	O(1)	Only uses constant space
Iterate through array and sum values	O(1)	No extra memory used besides accumulator
Store array copy	O(n)	Needs space to store the copied array
Recursive factorial (factorial(n))	O(n)	n stack frames in the call stack
Binary search (recursive)	O(log n)	Recursive depth is log(n) for sorted array
Binary search (iterative)	O(1)	No extra space beyond a few variables
Merge sort	O(n)	Needs temp arrays to merge subarrays
Quick sort (in-place)	O(log n)	Call stack for recursive calls
Depth-first search (recursive) in tree	O(h)	h = height of the tree (stack frames)
Breadth-first search (using queue)	O(n)	Stores all nodes at current level in queue
DP with full 2D table (e.g., LCS)	O(m*n)	Stores results of all subproblems
Optimized Fibonacci with two variables	O(1)	Only tracks last two results
Memoized Fibonacci (top-down DP)	O(n)	Memoization table + recursion stack
Using a hash map to count frequencies	O(n)	Stores one count per element
Storing all substrings of a string	O(n^2)	Total number of substrings is ~n^2
Adjacency list for graph with V nodes, E edges	O(V + E)	One list per node, total edges stored