Data Structures & Algorithms

Ryan Hou

2025-02-17

Table of contents

Preface

This is my notes on Data Structures & Algorithms in C++.

Resources

Some relevant resources:

- EECS 280 - Programming and Intro Data Structures (University of Michigan)
- EECS 281 - Data Structures and Algorithms (University of Michigan)
- EECS 376 - Foundations of Computer Science
- Data Structures & Algorithms - Google Tech Dev Guide
- EECS 281 References
- The Algorithms

Practice Resources:

- LeetCode
- NeetCode
  - Blind 75
- Codeforces

Interview Resources:

- Leetcode Patterns
- Learning Resources - Reddit-wiki-programming
- AlgoMonster
- Tech Interview Handbook

C++ Guides:

- learncpp.com

Books:

- Algorithms - Jeff Erickson
- Cracking the Coding Interview

1 Introduction

1.1 Perspective

> Note 1: Definition - Definition goes here
>
> **Definition** is defined by: definition.

This notebook contains programming basics, data structures, and algorithms. The language of choice is C++, and concepts from C++ STL are also covered.

1.2 Subheading

2 Foundations

2.1 Resources

3 Pointers

4 Arrays

5 Strings

6 Streams and IO

7 Abstract

8 Object-Oriented Programming

9 Dynamic Memory

10 Linked List

11 Iterators

12 Recursion

13 Function Objects, Functors

14 Error Handling & Exceptions

15 Stacks & Queues

16 Complexity Analysis

16.1 References

- AlgoMonster - Runtime Summary

16.2 Runtime Overview

16.3 Common Time Complexities

16.3.1 O(1) - Constant

Constant time complexity. Could be

- Hashmap lookup
- Array access and update
- Pushing and popping elements from a stack
- Finding and applying math formula

16.3.2 O(log(N)) - Logarithmic

$log(N)$ grows very slowly

In coding interviews, log(N) typically means:

- Binary search or variant
- Balanced binary search tree lookup
- Processing the digits of a number

Unless specified, typically log(N) refers to $log_2(N)$

Example C++:

```cpp
int N = 100000000;
while (N > 0) {
  // some constant operation
  N /= 2;
}
```

Many mainstream relational databases use binary trees for indexing by default, thus lookup by primary key in a relational database is log(N).

16.3.3 O(N) - Linear

Linear time typically means looping through a linear data structure a constant number of times. Most commonly, this means:

- Going through array/linked list
- Two pointers
- Some types of greedy
- Tree/graph traversal
- Stack/Queue

Example C++:

```cpp
for (int i = 1; i <= N; i++) {
  // constant time code
}

for (int i = 1; i < 5 * N + 17; i++) {
  // constant time code
}

for (int i = 1; i < N + 538238; i++) {
  // constant time code
}
```

16.3.4 O(K log(N))

- Heap push/pop K times. When you encounter problems that seek the "top K elements", you can often solve them by pushing and popping to a heap K times, resulting in an O(K log(N)) runtime. e.g., K closest points, merge K sorted lists.
- Binary search K times.

Since K is constant this kind of isn't its own time complexity and can be grouped with O(log(N))

16.3.5 O(N log(N)) - Log-Linear

- Sorting. The default sorting algorithm's expected runtime in all mainstream languages is N log(N). For example, java uses a variant of merge sort for object sorting and a variant of Quick Sort for primitive type sorting.
- Divide and conquer with a linear time merge operation. Divide is normally log(N), and if merge is O(N) then the overall runtime is O(N log(N)). An example problem is smaller numbers to the right.

16.3.6 O(N^2) - Quadratic

- Nested loops, e.g., visiting each matrix entry
- Many brute force solutions

```
for (int i = 1; i <= N; i++) {
  for (int j = 1; j <= N; j++) {
    // constant time code
  }
}
```

16.3.7 O(2^N) - Exponential

Grows very rapidly. Often requires memoization to avoid repeated computations and reduce complexity.

- Combinatorial problems, backtracking, e.g. subsets
- Often involves recursion and is harder to analyze time complexity at first sight

E.g.: A recursive Fibonacci algorithm is $O(2^N)$

```
int Fib(int n) {
  if (n == 0 || n == 1) {
    return 1;
  }
  return Fib(n - 1) + Fib(n - 2);
}
```

16.3.8 O(N!) - Factorial

Grows very very rapidly. Only solvable by computers for small N. Often requires memoization to avoid repeated computations and reduce complexity.

- Combinatorial problems, backtracking, e.g. permutations
- Often involves recursion and is harder to analyze time complexity at first sight

16.4 Amortized Time Complexity

17 STL

18 Heaps

18.1 References

- AlgoMonster - Heap Fundamentals

18.2 Heap? Priority Queue?

Priority Queue is an **Abstract Data Type**, and Heap is the concrete data structure we use to implement a priority queue. Source

18.3 Heap in C++

19 Trees

20 Searching

21 Sorting

22 Hash Maps

23 Graphs

24 Brute-Force & Greedy Algorithms

25 Divide and Conquer, Dynamic Programming

26 Backtracing, Branch and Bound Algorithms

27 Summary

In summary...

28 Tips on Solving DS&A Questions

28.1 Problem Solving Flowchart

28.2 DS&A Roadmap

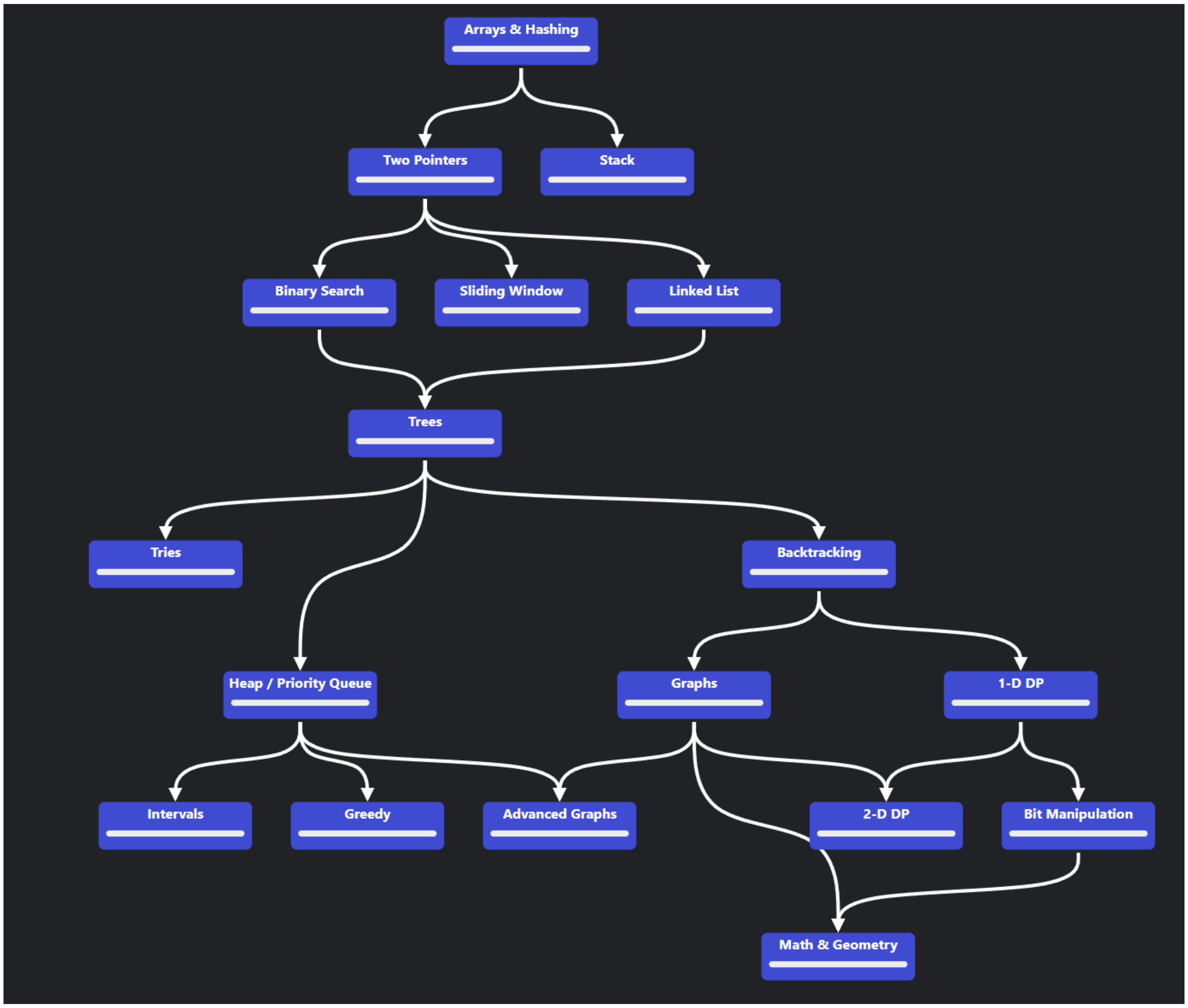

Figure 28.2: A Roadmap for studying. Source

```
If input array is sorted then
- Binary search
- Two pointers

If asked for all permutations/subsets then
- Backtracking

If given a tree then
- DFS
- BFS

If given a graph then
- DFS
- BFS

If given a linked list then
- Two pointers

If recursion is banned then
- Stack

If must solve in-place then
- Swap corresponding values
- Store one or more different values in the same pointer

If asked for maximum/minimum subarray/subset/options then
- Dynamic programming

If asked for top/least K items then
- Heap
- QuickSelect

If asked for common strings then
- Map
- Trie

Else
- Map/Set for O(1) time & O(n) space
- Sort input for O(nlogn) time and O(1) space
```

Figure 28.1: Tips on problem approach.Image Source

28.3 Problem Flowchart

28.4 ROI

28.5 "Academic" Algorithms

According to AlgoMonster, some **algorithms** that are very rarely/almost never asked in interviews:

- Minimal spanning tree: Kruskal's algorithm and Prim's algorithm
- Minimum cut: Ford-Fulkerson algorithm
- Shortest path in weight graphs: Bellman-Ford-Moore algorithm
- String search: Boyer-Moore algorithm

28.6 Keyword to Algo

AlgoMonster provides a convenient "Keyword to Algorithm" summary:

"Top k"

- Heap
  - E.g. K closest points

"How many ways.."

- DFS
  - E.g. Decode ways
- DP
  - E.g. Robot paths

"Substring"

- Sliding window
  - E.g. Longest substring without repeating characters

"Palindrome"

- two pointers: Valid Palindrome
- DFS: Palindrome Partitioning
- DP: Palindrome Partitioning II

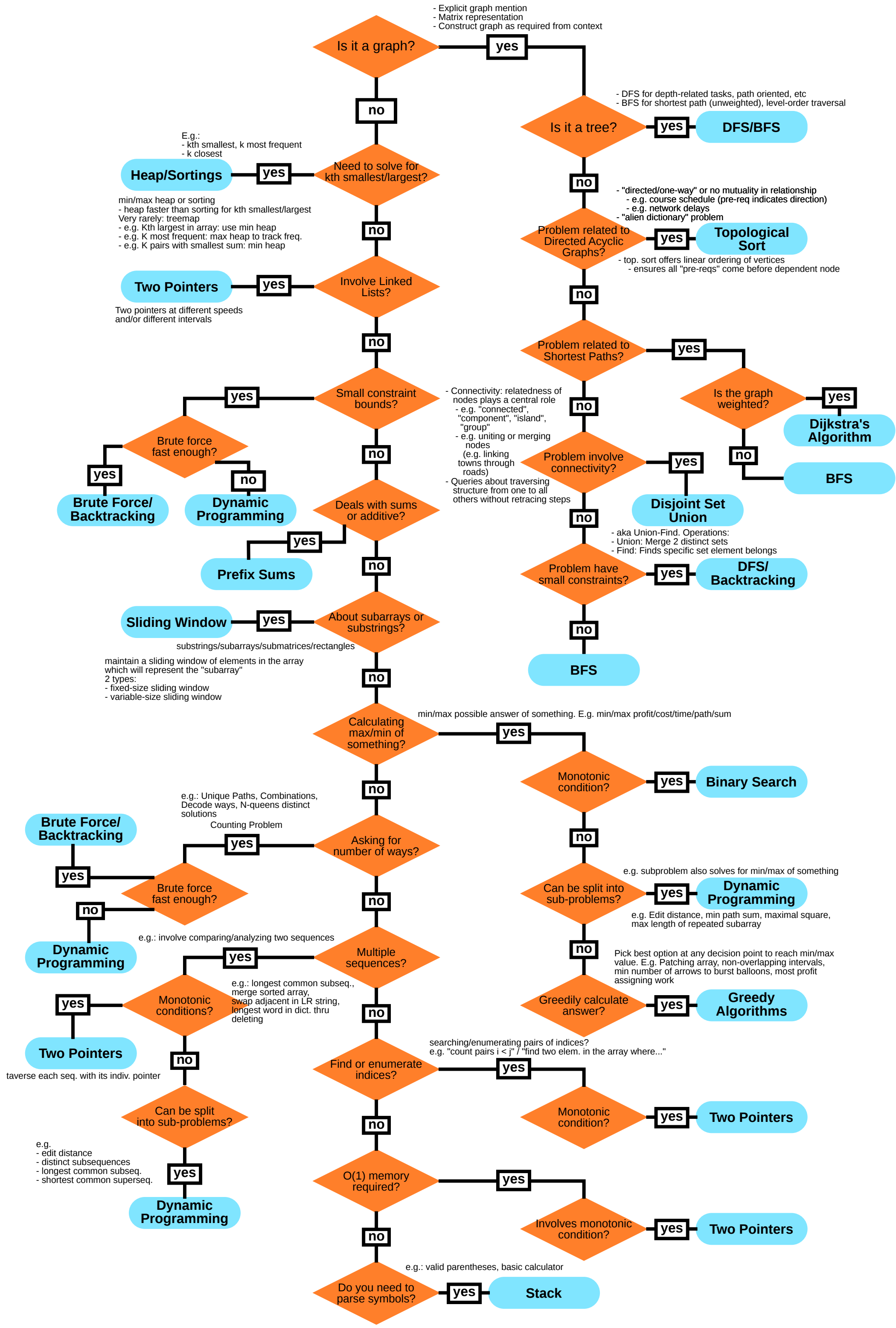

Figure 28.3: A problem solving flowchart based on AlgoMonster's flowchart. Open the SVG in a new tab and zoom in/out for better viewing.

Topic	Difficulty to Learn
Two Pointers	Easy
Sliding Window	Easy
Breadth-First Search	Easy
Depth-First Search	Medium
Backtracking	High
Heap	Medium
Binary Search	Easy
Dynamic Programming	High
Divide and Conquer	Medium
Trie	Medium
Union Find	Medium
Greedy	High

Figure 28.4: Studying to Maximizing ROI according to AlgoMonster.

"Tree"

- shortest, level-order
  - BFS: Binary Tree Level-Order Traversal
- else: DFS: Max Depth

"Parentheses"

- Stack: Valid Parentheses

"Subarray"

- Sliding window: Maximum subarray sum
- Prefix sum: Subarray sum
- Hashmap: Continuous subarray sum

Max subarray

- Greedy: Kadane's Algorithm

"X Sum"

- Two pointer: Two sum

"Max/longest sequence"

- Dynamic programming, DFS: Longest increasing subsequence
- mono deque: Sliding window maximum

"Minimum/Shortest"

- Dynamic programming, DFS: Minimal path sum
- BFS: Shortest path

"Partition/split … array/string"

- DFS: Decode ways

"Subsequence"

- Dynamic programming, DFS: Longest increasing subsequence
- Sliding window: Longest increasing subsequence

"Matrix"

- BFS, DFS: Flood fill, Islands
- Dynamic programming: Maximal square

"Jump"

- Greedy/DP: Jump game

"Game"

- Dynamic programming: Divisor game, Stone game

"Connected component", "Cut/remove" "Regions/groups/connections"

- Union Find: Number of connected components, Redundant connections

Transitive relationship

- If the items are related to one another and the relationship is transitive, then chances are we can build a graph and use BFS or Union Find.
  - string converting to another, BFS: Word Ladder
  - string converting to another, BFS, Union Find: Sentence Similarity
  - numbers having divisional relationship, BFS, Union Find: Evaluate Division

"Interval"

- Greedy: sort by start/end time and then go through sorted intervals Interval Pattern

29 Problems & Explanations