: Competitive Programming:

Learning:

**Day**−∞−∞**to 0:**Stick to a programming language like C or C++. Make sure that you are comfortable with pointers/objects.

**Day 1:** Understand the concept of [**Algorithmic complexity**](https://en.wikipedia.org/wiki/Algorithmic_complexity). Skip the theory for now, but for every piece of code you write, you should be able to derive both time and space complexity.

**Day 2 - 10:** Let’s start with some simple data structures,

1. *Arrays*
2. *Linked Lists*
3. *Strings*
4. *Stacks*
5. *Queues*

Understand their basic operations (*insert, delete, search, traversal*) and their complexity - [Big-O Algorithm Complexity Cheat Sheet](http://bigocheatsheet.com/), and code them all.

**Day 11 - 25:**Let’s now learn some simple algorithms,

1. *Sorting* - [Insertion sort](https://en.wikipedia.org/wiki/Insertion_sort), [Merge sort](https://en.wikipedia.org/wiki/Merge_sort), [Quick sort](https://en.wikipedia.org/wiki/Quicksort), [Heap sort](https://en.wikipedia.org/wiki/Heapsort), [Bucket sort](https://en.wikipedia.org/wiki/Bucket_sort), [Counting sort](https://en.wikipedia.org/wiki/Counting_sort), [Radix sort](https://en.wikipedia.org/wiki/Radix_sort), [External sorting](https://en.wikipedia.org/wiki/External_sorting)
2. *Search* - [Linear search](https://en.wikipedia.org/wiki/Linear_search), [Binary Search](https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/) (along with its variants).
3. *Prime Numbers* - [Sieve of Eratosthenes](https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes), [Primality test](https://en.wikipedia.org/wiki/Primality_test)
4. *Strings* - [String searching](https://en.wikipedia.org/wiki/String_searching_algorithm), [LCS](https://en.wikipedia.org/wiki/Longest_common_subsequence_problem), [Palindrome detection](https://www.rosettacode.org/wiki/Palindrome_detection)
5. *Miscellaneous* - [Euclidean algorithm](https://en.wikipedia.org/wiki/Euclidean_algorithm), [Matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication), [Fibonacci Numbers](https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Fibonacci_Number_Program), [Pascal's Triangle](http://www.geeksforgeeks.org/pascal-triangle/), [Max Subarray problem](https://en.wikipedia.org/wiki/Maximum_subarray_problem)

**Day 26 - 50:** Once you are comfortable with everything above, start doing problems from,

1. [Cracking the Coding Interview](https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/0984782850)
2. [Elements of Programming Interviews](https://www.amazon.com/Elements-Programming-Interviews-Insiders-Guide/dp/1479274836)
3. [Programming Interviews Exposed: Secrets to Landing Your Next Job](https://www.amazon.com/Programming-Interviews-Exposed-Secrets-Landing/dp/1118261364)
4. [GeeksforGeeks](http://www.practice.geeksforgeeks.org/)
5. [HackerRank](https://www.hackerrank.com/)
6. [InterviewBit](https://www.interviewbit.com/invite/afaf)

Stick to chapters of arrays, linked lists, strings, stacks, queues and complexity.

**Day 51 - 60:** Let’s learn some non-linear data structures,

1. *Tree*
   1. *Binary Tree, Binary Search Tree* - [Tree traversals](https://en.wikipedia.org/wiki/Tree_traversal), [Lowest common ancestor](https://en.wikipedia.org/wiki/Lowest_common_ancestor), [Depth, Height & Diameter](http://stackoverflow.com/questions/2603692/what-is-the-difference-between-tree-depth-and-height), [Finding k-th smallest element](http://www.geeksforgeeks.org/find-k-th-smallest-element-in-bst-order-statistics-in-bst/)
   2. *Heaps*
2. *Hash table* - [4 sum problem](http://www.sigmainfy.com/blog/4sum-problem-analysis-different-time-complexity.html), [Checking if sudoku solution is valid](http://stackoverflow.com/questions/5484629/check-if-sudoku-solution-is-valid)
3. *Graph* - [Breadth-first search](https://en.wikipedia.org/wiki/Breadth-first_search), [Depth-first search](https://en.wikipedia.org/wiki/Depth-first_search), [Topological sorting](https://en.wikipedia.org/wiki/Topological_sorting), [Minimum spanning tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree), [Shortest path problem](https://en.wikipedia.org/wiki/Shortest_path_problem),

**Day 61- 90:** Refer to the previous resources and start doing problems from trees, hash tables, heaps and graphs.

**Day 91 - 100:** Understand [Computational complexity theory](https://en.wikipedia.org/wiki/Computational_complexity_theory) and [NP-completeness](https://en.wikipedia.org/wiki/NP-completeness), [Knapsack problem](https://en.wikipedia.org/wiki/Knapsack_problem), [Travelling salesman problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem), [SAT problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem) and so on.

Date: 18/01/2023

1. What is <bits/stdc++.h> file in cpp?

* In contests, using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive.
* For example to use sqrt( ) function, in <bits/stdc++.h> header file we need not have to

write <cmath> header file in the code.’

#include <bits/stdc++.h>

using namespace std;

int main() {

cout << sqrt(25);

return 0;

}

Cpp STL:

Cpp contains algorithms and containers.

Inside containers:

1)Sequence containers

2)container Adaptors

3)Associative Containers

4)Unordered Associative