Tips And Tricks I

--------------------------------------------------------------------------------------------------------------

If input array is sorted then  
- Binary search  
- Two pointers  
  
If asked for all permutations/subsets then  
- Backtracking  
- Recursion  
  
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  
- Sliding Window  
  
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

Tips And Tricks II

----------------------------------------------------------------------------------------------------

1. While using HashMap use the containsKey() instead of keyset().contains()
2. When you wish to return and int[], do not create a new array and put values into it: just do return new int[]{elem1, elem2};
3. Try to avoid the Collections.contains() function as much as possible. It will take a considerable about of time for larger inputs
4. String take more time to create than lists of same length
5. Places where the time limit is exceeding, see if you can break the code down into smaller parts using recursion; this will make it simple to optimize
6. Instead of Math.abs(x) use if (x> 0) {x = -x;} this way, it is faster
7. If your code is looking lengthy most of the time, it is not optimized
8. Set.contains() / hashMap.contains() work in O(1) complexity
9. Check if the problem can be solved usinf a sliding window; it is not that obvious often
10. Char array can be directly converted to string by string constructor, instead of concatenating one by one..

Tips And Tricks III

--------------------------------------------------------------------------------------------------------------

**Decided to prepare for a top product based company**

Started with Leetcode. Tried easy questions. Failed. Learnt all the data structures. Fundamental algos. Solved easy questions. Tried medium questions. Failed. Implemented all the data structures and algorithms learnt. Tried medium questions. Failed. Solved 3 and 4 point questions from virtual contests. Slowly progressing. Still after one year of this I couldn't solve a given medium question. Back to drawing board.

**Change of approach and focus**

I was doing a full time job also. I needed more hours everyday to prepare. Stopped everything. Focused on health and diet. Dropped sugar completely. Started leetcode again with a fresh mindset and energy. Got myself a digital pen that helped a ton. Every question I fail I'd retry in 3 weeks. Always kept a timer and always talked as if I was at an interview to get used to the pressure. Never viewed the topic before attempting a question. I want to be able to figure out if this is a Dp question or greedy. Followed this random guy Elite Code on yt he has these raw videos of him approaching unseen problems. Then learnt everything about backend engineering and honestly fell in love with it.

Tips And Tricks IV

--------------------------------------------------------------------------------------------------------------

1. Understand Depth vs. Breadth
   1. The Breadth-Only approach has a problem—you don’t build a strong foundation. Interviews require deep problem-solving knowledge and the ability to code fast and accurately. You only develop these skills with focused preparation.
2. Start the Depth-First Approach—make a list of core questions
   1. Identify a list of ~100 core problems.
3. Master each data structure.
   1. Now that you have finalized your list, start with the basics. Know every data structure.
   2. Learn how to use each data structure in your language.
   3. Also, learn how to implement them.
4. Spaced Repetition
   1. The best technique we’ve seen – solve the problem again in 3 days. Then in a week. Then in a month. It will become second nature to you.
5. Isolate techniques that are reused. Isolate actual code blocks.
   1. This is where the Depth-First approach gets exciting. As you solve these problems, you’ll start to notice patterns.
   2. Let’s say you solved 5 problems that used Binary Search. You can isolate the Binary Search code and practice it over and over. You know it will be used in similar problems.
6. Now, it’s time for Breadth.
   1. Let’s say you’ve mastered your core problems. Using common data structures is second nature to you. You can now look beyond your core set. Because you’ve implemented so many techniques already, you don’t even have to code all the new questions.