# **1\ data structure**

1. array:
   1. dynamic programming, the ith element is built on 0…n-1. If you already know all the elements ahead, how would you get the next one?
   2. two pointers: move from two sides or the same side.
   3. binary search: the answer is in a range
2. hash table, hash set：
   1. use set instead of array for quick insert, query and delete. use hash table when key is not number in a range.
   2. for non-duplicate situation.
3. binary search tree:
   1. insert and delete in BST is O(lgN)

# **2\ logical thinking**

1. think about all the possible data structures to help you solve the problem:

hash map for key-value pair insert and query.

hash set for quick insert, query and delete.

heap for finding min/max values

stack, queue for dynamic processing

tree: organize your data in a tree for quick search and query.

1. Think about the process you generate the solution. Especially the point when things make sense. Also focus on the structure and feature of the solutions. Find some facts about the problem/model.
2. dynamic programming: sometimes, the original problem show a good dp pattern, e.g find the n-th XXX. But sometimes the original problem is not clear enough. You should pose some limitations/assumptions on it , so that you can divide a big problem into subproblems(e.g. find the optimal XXX => the optimal XXX containing ith element, the optimal XXX ending with ith element, etc). The key is to define your subproblem in state i. Find connections between state i and state 0,…,i-1 and make use of what you already produce. **Remember to make memo!!!**

E.g. the shortest path with limitations.

1. Recursion.
2. Divide and conquer: solve its left, solve its right. Combine solutions of subproblems.
3. Binary search: If your solution is in a limited range, use binary search to find it.
4. Tree problem: the result of root depends on the result of root.left, root.right. use helper function and make the return result helpful.
5. Back tracking: in a situation you meet multiple choice, you choose one and consider the others choice later. You use function frame in stack to store local information.
6. Examplify and Simplify: from high level to low level by analysis the specific example, especially when your solution possibility is limited. Make assumptions, give certain conditions(if…) to list examples. Generalize: from low level to high level.
7. pre-checking to speed up.

# 3\ how to elegantly code

1. ask questions to clarify problem and corner case:

when deal with number: what type? int or double? what range? negative, 0, positive? will it overflow? Do not assume int by yourself.

when deal with character: how encoded?

cate about corner case: negative, zero, null, overflow, invalid, min, max

1. Design an algorithm: if you cannot do it in one step, analysis it and find some facts about it.

it’s also helpful to come up with a brute force solution.

1. Code:
   1. Care about data structure in your code. Always design your own class when possible. Good object-oriented sense. Think about the problem you meet in MS interview(class Point, class Polygon, your problem should be a method in Polygon class.)
   2. Efficiently design your API. Ignore the specific implementation first and only care for API first. This is also part of modular programming and object-oriented programming.
   3. Give clear definition to your variable and method. Write comments to help clarify.
   4. Encapsulate common ops into functions.
   5. Modular: outline your thoughts clearly.
   6. if some situation is hard to satisfy, discuss with your interviewee and see if you can ignore it for now and concentrate on other parts. Simplify.
2. test your code: check your concerning list to see if you deal with them carefully.
3. Never give up analysis!!! Interviews are supposed to be hard. Try your best!!

Calm down and really analysis it. Do it like in your life.

# 4\ problems

1. #52 N Queens
2. #312 Burst Balloons

**Think about the process the other way around**

**use MEMO in dp**

1. #145 Binary Tree Postorder Traversal

consider about reverse postorder, which is very similar to preorder.

1. #329 Longest Increasing Path in a Matrix

fix logical bug: it’s kind of like dfs in a tree. you are able to iterate one subtree each time, so in main program, you need to do multiple iteration.

1. #301 Remove Invalid Parenthese

when you get confused, you can solve a sub problem first.

1. #315 Count smaller numbers in the right

when the problem is about range, comparision, you should think about sorted array and binary search OR binary search tree.

How are you going to organize your data/info ? in a tree, in a map, in a set, in order and so on.

Three solutions: brute force(O(N^2)), sort while query(O(NlgN) for query), BST(NlgN)

1. #330 Patching Array

Generalize: our target is k, it means we can get all the numbers from 1 to k-1. If our next number is k, then we can get all the numbers from 1 to 2\*k-1, if it is less than k, we can get from 1 to k-1+nums[i]. So the next target will be k+nums[i] if nums[i] <= k, else we patch a k.

tail recursion: change it to iteration.