# **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. check about overflow especially you add, subtract, divide or times two int.
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.

1. #354 Russain Doll Envelopes

1) when you encounter some problems, first think about is there any way to walk around it. If not, try to solve it directly.

**2) 1. divide and conquer -> what is the longest increasing sequence ending with ith element?**

**2. dp -> if we already have an increasing sequence like 1,3,6,7,9, what if the next number we meet is 3, 4, 8, 10?**

**3) examplify some problems so that you can find a clue.**

1. #297 Serialize and Deserialize Binary Tree

Decouple your solution. Make it modular so that keep your solution in a clear way.

1. #115 Disctinct Subsequence

recursion is trivail: but still need improvement to avoid unnecessary recursion.

**iteration: build a map for two string problem. map[m, n] how it connects to map[m-1,n], map[m-1,n-1], map[m, n-1]**

Similar problem: longest common subsequence.

1. #363 Max Sum of Reactangle No Larger Than k

The maximum subarray sum:

1)divide and conquer: what is the maximum subarray sum ending with i-th element? cum[i] = Math.Max(cum[i-1]+arr[i], arr[i])

2)sum[i, j] = sum[j]-sum[i];

cum += arr[i];

ret = Math.Max(ret, cum-min\_cum);

min\_cum = Math.Min(min\_cum, cum); // min\_cum initialize to be 0.

subarray sum is sum[j]-sum[i]. Suppose we already have sum[0], sum[1], ..., sum[i-1]. Now we have sum[i], we only need to subtract sum[i] using the smallest number in the set.

The maximum subarray sum no more than k:

store cum[] in a set, for a new cum: find lower\_bound of cum-k, ret = Math.Max(ret, cum-\*lower\_bound). Insert cum into set.

This is actually implemented by BST-> To find a min number in a set more than k. Use BST, O(lgN)

subarray sum is sum[j]-sum[i]. Suppose we already have s = {sum[0], sum[1], ..., sum[i-1]}. Now we have sum[i], we need to find a number x in s, so that sum[i]-x <= k, e.g x >= sum[i]-k. x is the smallest number bigger than sum[i]-k in set s. So we use BST to store the set so that we can find x fast.

similar problem: count numbers smaller than self in the right

THe maximum sub-matrix:

1) For each row i, calculate the sum of l-th col to r-th col elements. For each i, apply one-D solution. THe complexity should be O(n^2\*m): n is smaller one of # rows and cols.

The maximum sub-matrix no more than k:

sum[i, j]=the first i element of column j. For each i, apply one-D solution.The complexity should be O(n^2\*m\*lgm)

**What is the optimal subsequence in a sequence?**

**1) what is the optimal subsequence containing ith element / containing ith element as the biggest value / ...**

**2) if we already have a optimal subsequence like X, X, X, X, what if our next number is A,B, C, D, ....? How to adapt the optimal subsequence.**

1. #293 Sliding Window Maximum

Dig some truth and fact from the problem: if we have 2 following 3. Then 2 will never be used because any subarray containing 2 will contain 3. We just pop 2 out.

1. #115 Distinct Subsequence

when asking problems about two string s,t, think about using dp tp solve problems.

Memo[i,j] indicates a subproblem, memo[m-1,n-1] is the answer.

String problem:

1. scan the string, use stack/set/map to store information

#227 basic calculator #3 Longest Substring Without Repeating Characters #32 Longest valid Parenthese #76 Minimum Window Substring

1. use two pointers from the same side or two sides

#76 Minimum Window Substring

1. use dp

# 91 Decode ways #5 Longest Palindromic Substring #115 Distinct Subsequence

1. #25 Reverse Nodes in k-Groups

Decouple you op: count the # of loops required to avoid judge null condition.

1. #164 Maximum Gap
2. radix sort in time O(N)
3. bucket sort: **dig some facts about the problem. If the max value is max, min value is min, the number of elements is n, then the maximum gap would be more than (max-min)/n.** So we do not need to care about numbers in a small range, we put them into the same bucket. We only need to know the max/min value in each bucket. The maximum gap must be between successive buckets.
4. #327 Count of Range Sum

Analyse the answer: some ranges are only about the left part, some are about the right part. So you divide the original problem into two similar subproblems: deal with left and right individually. Then deal with the answers that lie between these two parts.

Similar problem: #307 Range Sum Query: segment tree. Easy to maintain and modify info about range/segment.

1. #123 Best Time to Buy and Sell Stock III
   1. You can easily come up with the first version of solution: buy1[i] means the maximum profit you can get if you buy on day i. sell1[i] means the max\_profit if you sell on day i. You compute buy1 first, then use buy1 array to build sell1 array, sell1->buy2->sell2.
   2. Then from the first solution , you notice that in buy1, what really cares is only the max-profit you can get until day i. So in your new solution, buy1[i] means the max\_profit you can get until day i using only one buy.
   3. From the second solution, only one element in array matters, so you do not need to declare an array, just use single variable.
2. #316 Remove Duplicate Letters

**For such generic problems, make it concrete by listing examples. Analysis the real example and notice the point when things make some sense.**

**From detailed example, then you can generic it to abstract rules.**

See our first input is c, the next letter would have two kinds: smaller than c or bigger than c.

If smaller than c, say we have a: should we drop c? It depends on whether we have c afterwards. If we have, then we drop c, if not we keep c.If bigger than c, say we have d, we just push d in.

See we have cedcd. The stack is c->ce->ced, when we meet c again, although c is smaller than d and we have d afterwards, but c is already in the stack, so we just continue.

The above problem can be found if you generalize your rule from detailed example.

….x, the next is y. if y is in the stack, continue. Compare x and y, and also consider if we still have x afterwards.

1. #56 Merge Intervals

**It’s very similar to above problems. We analysis the process step by step, when we come to some point, we should combine other conditions to make some judgement, deal with it and move on.**

1. #282 Expression Add Operators

Your every single analysis values. Do not give up or just give it not deep deduction.

If we only need to return the total number of possibilities, then we can use divide and conquer by returning Llist<int> in helper function.

But now we need to return the string, so we should consider about scan from left to right, since for each step we have multiple options, we use bp.

What different? Heading zero and when we want to \*. So we deal with head zero apartly and see later if we can merge.

**Left to right: dp(iteration, what we can get unitl the ith element) or bp(recurion), or use other data structure.**

**From the whole: divide and conquer, dp.**

1. Overflow

#282 Expression Add operators #327 Count of Range Sum #164 Maximum Gap