**To begin:**

1. **clarify the basic meaning**
2. **care about different input cases: illegal, empty/null/zero, negative/duplicate, corner cases.**

**To move on:**

1. **analysis and describe your idea: clear definition**

**To analysis:**

1. **basic data structures: hash table, hash set, stack, queue, priority queue, linked list, binary search.**
2. **advanced:**

**2.1) use tree to describe your idea**

**2.2) dp: make problem size smaller by make some decisions and assumptions**

Can you make the problem size smaller? If you make some decisions, and move on base on this decision, then you can make problem size smaller. Use memo!

Top down dp to solve win-game problems: define helper function(whether the current player wins the game), use memo.

**2.3) divide and conquer, greedy, reverse thinking**

**To test:**

1. **if you care about all the possible input cases.**
2. **Complexity**

**Weakness:**

1. **DFS can be either recursion and iteration.**

**Recursion: return from function once.**

**Iteration: use stack, you can stop at any time you like.**

1. **Object Oriented Programming: design data structures and API**
2. **DP problem**

**For some problems, like to generate a set, there is no faster way because you need to generate each element one by one. For others like calculate a sum, use dp instead of naïve solutions.**

**DP: 0 and 1 backpag problem. Give you certain sources, how do you make the maximum output?**

**Ans: make some decision(you try to choose something and move), make problem size smaller. If you use array, then array index becomes smaller. If you treat DP problem as array problem, our target is to calculate dp[m][n]. Then we should find the connection between dp[m][n] with other elements in the array.**

**You can use extra digit to represent state: Paint Fence**

**If you need to find all the answers satisfying some condition, it may means you need do a backtracking, no dp. If you encounter multiple choices, just choose one and move forward. dfs in a graph is a special form of backtracking.**

**Visit a tree:**

1. **Nested list: use stack**
2. **Pre-order, in-order, post-order: use stack**

**By using stack, you can visit the whole tree not only at one time and you can stop anytime and continue later. Stack helps to record where you are.**

**String window problem: find the max/min window satisfying a condition.**

**Keep a window[lo, hi], move hi to the right, use some data structure to keep information(stack, queue, etc), until you find the conditions already satisfied. Update the global optimal and then update lo.**

**Remove k digits: analysis the basic and simple example to find patterns.**

**494 Target Sum**

**dp[i][s] = dp[i-1][s-num[i]]+dp[i-1][s+num[i]]**

**This is the state function. If all the index becomes smaller, then you can turn recursion to iteration(bottom-up) method. Use state parameters as index. Else, use recursion because one index becomes smaller so it can always convergance. If you use recursion, the index is the function parameter.**

**Dp: in a state, you have multiple choices. You try to get the maximum output by choosing differently. To design dp, you can directly use your optimal solution. Or you can restrict the optimal with some choice, e.g the best solution if I choose x in state i. Then sum up will get the global optimal.**

**DP: to find the n-th xxx. Then you should find the connection between the n-th xxx with the 1~n-1 xxx. Or you could use bottom-up, how to find the second xxx according to the first one.**

**For math problem, carefully consider all the possible inputs first: Int.MaxValue, Int.MinValue, 0, negative, duplicate, overflow。 Math.Abs(int.min) will not get the right answer.**

**Binary search: the index is in a range or the value of answer is in a range. Both of them can use binary serach. e.g #287 find the duplicate number # find the kth smallest in a matrix.**

**Ask to clarify: illegal, empty, null input?**

**Analysis problem. Figure out and list some basic facts.**

**If no idea, think about can I use some data structures, can I use tree, can I use dp, can I use divide and conquer, can I use greedy, can I use reverse thinking?**

**Data structure: hash table, hash set, stack, queue, priority queue(max min), heap, linked list**

**Store information a graph, binary search tree, trie, segment tree.**

**Dp: do something and make size smaller. Deduce from n to n-1.**

**String and array: two pointers, dp, stack, use matrix to analysis.**

**Tree: dp, stack, queue, graph, dfs, bfs**

# **0\ general:**

start talking, ask about questions to clarify it. Take care about illegal input, zero, empty, null, duplicate, overflow, int/double. Fully understand what you need to do.

List an example to analysis it. When you cannot think clear, try to generalize. Or just code first.

Design an algorithm to solve it: generalize or exemplify. Figure out basic facts/patterns and list it out.

1. Is there brute force or direct solution?
2. Do not jump to a direct solution. See it from a bigger picture. You should analysis from the very beginning, figure out some basic facts: clarify the whole meaning, clarify the input(eg. Suppose the len of array is N, if N==0, if N==1 xxx. What is the range? Int or double?), figure some basic facts between input and output. Special cases include and we are going to check if our algorithm cover them correctly.
3. Think about its process: analysis the whole process. Use stack/queue to simulate. If in a state, we have multiple choice, use backtracking to simulate. Use greedy and design a strategy. No shame on brute force which is natural for complex problem.
4. Think about its feature: data structure, hash table, hash set, stack, heap, priority-queue, binary search tree(smaller, bigger, rank, etc. Let some key words spire you.), graph to simulate relations(bfs or dfs), etc. Use a data structure to describe your data , to summarize your data relations will spur your idea. Can I use graph to describe the connection? Especially about hard problems, do not do it like manually. Find its features and properties. See it from a higher level. Figure out basic facts.
5. Think about dp:

Top down:

Can you make the problem size smaller? If you make some decisions, and move on base on this decision, then you can make problem size smaller. Use memo!

If the optimal solution in left/right/pre-state is XXX, can you solve the cur-state? Deduce. Make assumptions and consider different situations. Use generic function to describe your idea.

Do you know how to solve a simple case? Try to build up from the bottom.

Make assumptions and Classify cases(settle some variables and change others.) to make problem size smaller.

If I know XXX, can I deduce the next XXX.

use dp(memo). Break big problems into similar sub-problems and solve the sub ones to form the answers for big one. try to reduce the size of problem. Find connections between different states.

How to think toward dp? Find a clue like find the kth, remove k digits, total amount is k and see if you can reduce the size of problem. Make assumption to simplify problems if we already know XXX, what is the optimal XXX.

1. Others: divide and conquer, recursion, dp(if you do not know which decision is optimal, or if you are in a situation with several states, several choices,just try each. This is exactly what computers do.), brute force, reverse thinking, etc.

Code it. Before you start to code, ask yourself this is the efficient solution? Like you should compare between dfs and bfs. While coding, handle invalid input. While coding, find if there are ways to improve it.

Test it.

You must give very CLEAR definition to your variable and method. It helps you think clearly.

Do it like a BOSS from a big picture. Do not get yourself in a mess

Reverse problem. See the opposite side of it.

Some times you need to think about from the beginning, from detailed example so that you can get a clue.

# **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
   4. care about overflow!!
   5. Think about addition method. Range Addition.
   6. Queue, stack and list may be better than array if the range is dynamically changing.
2. Stack and Queue: important on how to manipulate the push and pop.
3. hash table, hash set：
   1. Hash table and hash set is used to store information for quick insert, quick query and quick delete later.
   2. use set instead of array for quick insert, query and delete. use hash table when key is not number in a range.
   3. for non-duplicate situation.
   4. Scan once using hash table to store info, scan second so that you can query info in hash table.
   5. If there is a range in keys, use array is better than hash table.
4. binary search tree:
   1. insert and delete in BST is O(lgN)
5. priority queue: hold information about min/max, the kth largest xxx
6. Tree: binary tree, binary search tree, segment tree(hold range information for query like min, max, sum, etc), dictionary tree. To organize data in a ordered way so that you can query later.

How to handle tree problems:

1\ recursion: answer of node = f(answer of node.left, answer of node.right)+some\_judgements

2\ bfs: find the shortest path. Dfs. Use queue or stack. Use recursion or iteration.

3\ Find connections between parents and children. If parent is xxx, then its children should be xxx.

# **2\ logical thinking**

1. Do it like BOSS. Do not sink into detail. Analysis patiently but see it from bigger picture.
2. Care about details: overflow, like pow(x, n) to compute the –n of integer n, list to represent big data.
3. think about all the possible data structures to help you solve the problem:

stack, queue for dynamic processing

heap for finding min/max values

hash map for key-value pair insert and query.

hash set for quick insert, query and delete.

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:

Try if you can make the problem’s size smaller by reducing sth.

Or you can find the connections between different states.

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. Greedy: just design a strategy
7. Brute force: use code to describe the logic.
8. 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.
9. pre-checking to speed up.
10. Design problem: clarify every API: think about all possible kinds of input? What fields you should maintain and how to update these fields in each API? Find where your code is inefficient and upgrade it. Think from anther side: what is O(1)? Hash table, hash set, insert and remove from end of array, insert and remove from linked list.

# 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? Duplicate numbers?

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. Test a real object:

Who will use it for what purpose? What are the use cases? What is the boundary? What is failure condition?

Ask questions. List the facts you know about the object.

Basic facts check / intended use: even it’s running, you can specify it by adding different conditions like running in different weather, different ground, by people with different height and weight. Drawing on different surface, the surface is dry or wet, flat or not, cold or hot.

Unintended use case

1. Find lower bound of x in an array. Make[lo, hi] unchecked. On the other range, it’s either valid answer of invalid answer.
2. Find magic number: nums[i] = i
3. Make changes: how many different ways to make a change. For all the possible situations, how to classify them.
4. Stack boxes: if there is only one box, we can solve it. So how do we narrow n boxes to one box? We try each box at the bottom, and bottom which could be placed upon are limited. So the number of valid boxes is reduced.
5. Kth largest number in an array: heap, quick-sort(analyze complexity). Modify quick sort to help in the question of k-th …..
6. #52 N Queens
7. #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

1. #Composition: the minimum deletion so that the left subsequence satisfy some condition.

**String problem:**

1. scan the string, use stack/set/map to store information
2. two pointers from one side or both side
3. **dp: if you find dp hard, maybe you do not define good state, try to define some other state. Especially the problem is about optimal solution.**

x[i]: the minimum deletion **if** the left subsequence is ending with str[i] ->good

the minimum deletion after dealing with str[i] -> bad

because the bad one does not tell us any information about the cur string we have, so we cannot associate it with pre-state correctly.

1. Divide and conquer
2. DP:
3. **If one position has multiple states/decisions, try to desperate them as the optimal with state 0, state 1….**
4. **The optimal solution in state N: how you can arrive N from lower 1, 2, 3…N-1.**
5. **The optimal solution: what’s the optimal solution ending with i/ if i is the xxxx:**

**The if condition makes your statement more concrete and you will be able to connect different states.**

1. **If our current optimal solution is xxxx, the next number you meet is yyyy, How would you adapt your optimal solution.**
2. **List two string as a 2D map.**
3. Round\_robin and Registration Day

At the first look, you should think about this is priority queue problem.

Analysis this kind of problem from the basis situation. Who will be served first, who will be served second. Give clear definition.

1. Is T a subsequence of S?

Use binary search because we only care about the first appearance of this character.

1. #410 Split Array Largest Sum
2. greedy solution:

given a non-negative array, a valid k, a M. can you find a way to split the array into k subarrays so that each sum of subarray is equal or less to M.

1. binary search:

think about if the answer is in a range so that you know you can use binary search.

1. #233 Number of Digit One

How many XXXX? Split into subproblems, constraint conditions to more detailed ones. Under each sub condition, how many XXX and then sum them up.

Consider about 1 in each digit.

Think about general and simple case first(2,9) then consider about corner case(0, 1)

1. #416 Partition Equal Subset Sum

How to reduce recursion depth? Try to use loop in one recursion

1. #45 Jump Game II

Analysis the problem. Demonstrate your idea in an organize way(use some variable or example). Think about corner case and take care of them when coding and test them afterwards.

1. #23 Merge K sorted Lists

ask questions to specify: what is the element? Int or double? Sort in what order?

1. #41 Frist Missing Positive

logical thinking: what’s the range/possibility of the answer? You can think up the first version of solution based on the range. Then you notice it matches the length of array so use original array to store it by swapping.

1. #10 regular expression matching

fully consider different situations and care for details.

Classify possible situations in an elegant way. TO THINK ORGANIZED.