1. Sliding window

三步：

1. Update states, like count[end]++
2. While condition does not satisfy, count[start]--, start++, until conditions satisfy again
3. Ans = Math.max(ans, end - start + 1)

Example: 1004. Max Consecutive Ones III

2. Two pointers (fast & slow)

a. Slow: first element after all elements satisfying conditions, ex. Remove duplicates, slow is the first element after all unique elements
b. Fast: iterate through whole list
c. If current element does not satisfy, only move fast, keep slow at same position
d. If current element satisfy, num[slow] = num[fast], slow++

Example: 26. Remove Duplicates from Sorted Array

3. Merge Interval

a. Sort all the intervals based on start
b. Make the first interval as comparator
c. Compare each interval with the first interval, if having overlap, merge it together as new comparator interval, otherwise, add comparator interval to result list, make that interval as new comparator

Ex. merge interval / insert interval / interval intersection / minimum meeting room

4. Cyclic Sort

An array with length = n, it contains 1 - n, and it is not sorted, sort it in-place

[3, 1, 5, 4, 2] -> [1, 2, 3, 4, 5]
The key is put number in the index corresponding to this number itself, ex. 3 should be index = 2

If current number is not in right index, switch it with the one in its right index
[3, 1, 5, 4, 2]
[5, 1, 3, 4, 2]
[2, 1, 3, 4, 5]
If current number is in right index, move to next number
[1, 2, 3, 4, 5]

Ex. Find missing number(s) / find duplicate number(s)

5. In-place reversal of linkedlist

prev = None

```
    while head != None:
        temp = head.next
        head.next = prev
        prev = head
        head = temp

    return prev
```

6. Binary Tree level order traversal

Just apply BFS to tree,

Ex. Level order successor / right view of binary tree

7. Tree depth first search

Just apply DFS to tree, usually ask you to find a path

Ex. binary tree path sum / tree diameter / path with maximum sum

8. Two heaps

Pattern is ask you find median

Left heap is max-heap, right heap is min-heap, keep size of both heap same, the number in the middle is the median, when the size are not the same, move number around to keep it balanced

Ex. find median of number stream / sliding window median

8. Subsets

Pattern: permutation / combination

DFS is well-known to solve this, but you should understand the BFS solution, which is more generic to be applied to different follow-ups

9. Modified binary search

Different kinds of binary search

Ex. Given an array of numbers sorted in ascending order, find the element in the array that has the minimum difference with the given 'key'.

[4, 6, 10], key = 7, output = 6

Usually, for this kind of problem, you need to examine both nums[start] and nums[end], see which one is closer to the answer

10. Top ‘k’ elements

The most important data-structure you need to use is Heap, which can improve from O(nlgn) to O(nlgk)
Ex. K closest numbers
Another pattern is you need to choose based on some priority, like frequency, you can use heap as a tool
Ex. LT621 Task Scheduler

11. K way merge

Whenever we are given ‘K’ sorted arrays, we can use a **Heap** to efficiently perform a sorted traversal of all the elements of all arrays. We can push the smallest (first) element of each sorted array in a **Min Heap** to get the overall minimum. While inserting elements to the **Min Heap** we keep track of which array the element came from. We can, then, remove the top element from the heap to get the smallest element and push the next element from the same array, to which this smallest element belonged, to the heap. We can repeat this process to make a sorted traversal of all elements.

12. 0/1 knapsack (背包问题)

You can start with brute-force recursion solution, for each element, you have two choices, choose it or not to choose it, which is O(2^n), but there are many states are duplicated, if you use array to record it, like dp[i][sum], then you can have approach 1: top-down with memorization, which becomes O(N * C) N is for i, C is for sum
If you do it in opposite way, which is to come up with dp formula, you can have bottom-up solution. Dp[i][c] = Math.max(dp[i - 1][c]/* not choose */, dp[i - 1][c -

weight[i]] + profit[i] /* choose */), also, notice you only used dp[i - 1], you can improve it to dp[(i - 1) % 2][c], saved space.

13. Topological sort

In-degree: how many nodes comes into current node, [2, 3], means 2 can reach 3, then in-degree of 3 is 1
Graph: adjacency lists representing all the edges, [1, 4] [2, 3] [1, 3], 1: [4, 3], 2: [3]
Source: nodes with in-degree is 0

To find the topological sort of a graph we can traverse the graph in a **Breadth First Search (BFS)** way. We will start with all the sources, and in a stepwise fashion, save all sources to a sorted list. We will then remove all sources and their edges from the graph. After the removal of the edges, we will have new sources, so we will repeat the above process until all vertices are visited.