Strategy:

* Easy: 30mins
* Medium: 45mins
* Hard: 1hr

If close to the solution throw in extra 10 mins

Arrays and Strings

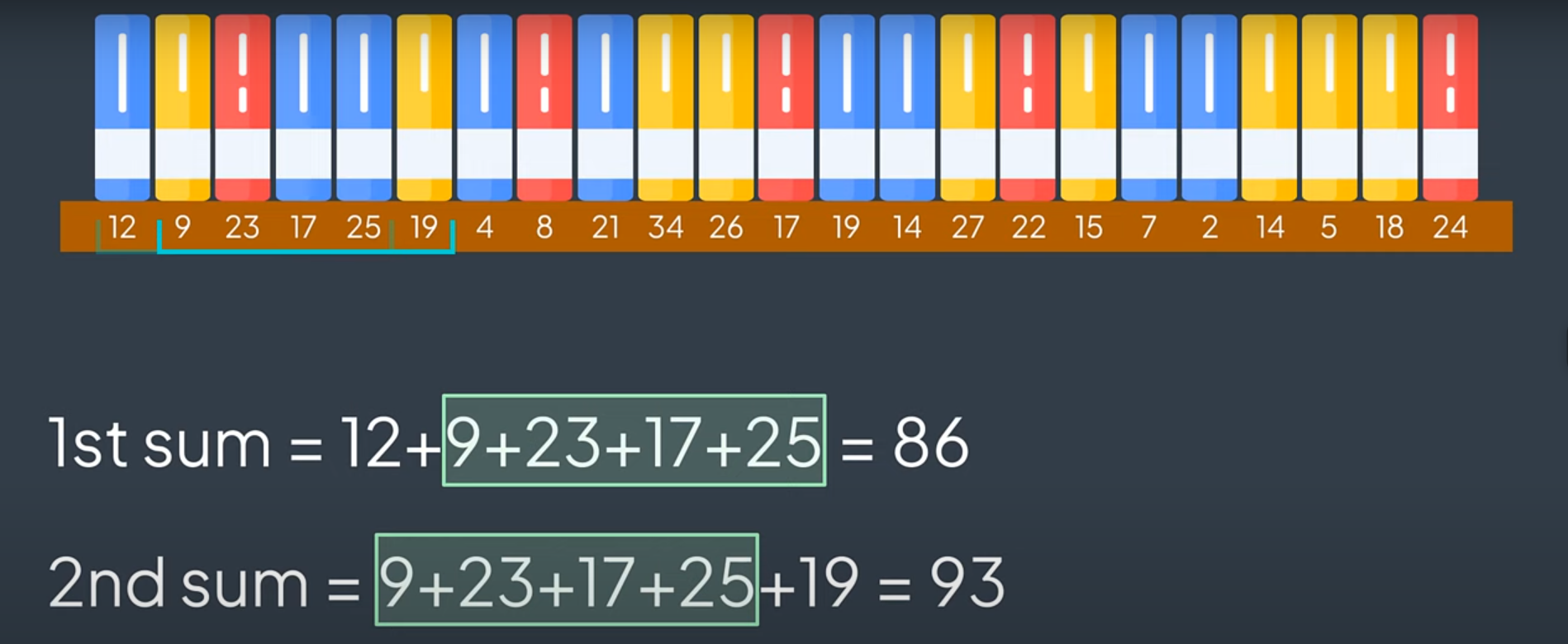
You should return in the conditions (if) to end the loop if find a valid value🡪 stopping loops.

1. Two Pointers: Used for finding pairs or elements that meet specific criteria.

* List elements increasing
* Use two pointer I pointer is head of array; another pointer is tail of array.
* Two pointer i and j should only change when satisfy the condition.

1. Sliding Window: Maintains a subset of elements within a larger dataset. Then we keep moving that window by extending it and shrinking it while respecting constraints until we finish the hold input.

* The size of the window can be change.
* Don’t need array to iterate the binary search.
* if a sliding window problem is related to characters, it almost always uses a frequency map or array.



1. Prefix Sum: Precompute cumulative sums for quick range queries.
2. Bit Manipulation: XOR

* A XOR A = 0; A XOR 0 = A;
* If you see a duplicate number and don’t use extra place.

Hash Map

1. Find number of subarrays
2. Sliding window

LinkedList

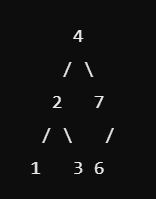
* In situations you want create a new list that don’t change anything in original lists use dummy node as a starting point.
* If you want to merge two list, you must create a pointer to keep track the dummy list.
* In situations change original list like reverse list you can use current node is equal to head
* If you want to copy from the original list, you must create heads of two list and two pointers to keep track two lists.
* If you want to make change the list you must use at least 3 pointers

Trees

* Finding one or more base cases
* Calling the same function on the left subtree
* Calling the same function on the right subtree
* Joining the results

Note:

* The height of a node: 1+ max (leftHeight, rightHeight).
* The recursive must store in value except modify tree.
* Don’t use root.left.val and root.right.val

1. Depth-First Search (DFS): Preorder, inorder, and postorder traversals.
2. Breadth-First Search (BFS): Level-order traversal.
3. Binary Search Tree (BST) operations: Insertion, deletion, and validation.
4. Tree construction: From preorder/inorder or postorder/inorder traversals.

Hash Tables

1. Frequency counting: Track occurrences of elements.
2. Two Sum pattern: Find pairs with a specific sum.
3. Anagram detection: Compare character frequencies.
4. Caching: Store computed results for quick lookup.

Graphs

1. Depth-First Search (DFS): Explore paths deeply before backtracking.
2. Breadth-First Search (BFS): Explore nodes level by level.
3. Topological Sort: Order nodes in a directed acyclic graph.
4. Union Find: Detect cycles and connect components.

Stacks

1. Parentheses matching: Validate balanced brackets.
2. Monotonic stack: Maintain increasing/decreasing order for next greater/smaller element problems.
3. Expression evaluation: Evaluate arithmetic expressions.

Queues

1. BFS implementation: Level-order traversal in graphs and trees.
2. Task scheduling: Manage order of operations.
3. Sliding window problems: Maintain a window of elements.

Heaps

1. Top K Elements Pattern: Find or manipulate the K largest/smallest elements in a collection.
2. Merge K Sorted Pattern: Combine K sorted lists or arrays into a single sorted list.
3. Two Heaps Pattern: Use two heaps to track median or balance elements in a stream.
4. Sliding Window Median Pattern: Calculate median in a sliding window over a stream of numbers.
5. Scheduling Pattern: Manage tasks or intervals using a heap for efficient scheduling.