Complete List of Algorithmic Patterns

🎯 FUNDAMENTAL PATTERNS (Must Learn First)

1. Traversal Pattern

When: Need to visit all elements

```python
# Linear traversal
for i in range(len(arr)):
    process(arr[i])

# Linked list traversal
current = head
while current:
    process(current.data)
    current = current.next
```

Examples: Print all elements, find sum, count elements

2. Search Pattern

When: Finding specific elements

```python
# Linear search
for i in range(len(arr)):
    if arr[i] == target:
        return i

# Binary search (sorted array)
left, right = 0, len(arr) - 1
while left <= right:
    mid = (left + right) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] < target:
        left = mid + 1
    else:
        right = mid - 1
```

3. Build-As-You-Go Pattern

When: Constructing new data structures

```python
result = []
for element in input:
    processed = process(element)
    result.append(processed)
return result
```

🔄 TWO-POINTER PATTERNS

4. Opposite Ends Pattern

When: Working with sorted arrays, palindromes

```python
left, right = 0, len(arr) - 1
while left < right:
    if condition_met(arr[left], arr[right]):
        return True
    elif arr[left] + arr[right] < target:
        left += 1
    else:
        right -= 1
```

Examples: Two Sum (sorted array), Valid Palindrome, Container With Most Water

5. Same Direction Pattern

When: Finding subarrays, removing elements

```python
slow = fast = 0
while fast < len(arr):
    if condition(arr[fast]):
        arr[slow] = arr[fast]
        slow += 1
    fast += 1
```

Examples: Remove Duplicates, Move Zeros

6. Fast & Slow Pointers (Floyd's)

When: Detecting cycles, finding middle

```python
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
    if slow == fast:  # Cycle detected
        return True
```

Examples: Linked List Cycle, Find Middle Node

🪟 SLIDING WINDOW PATTERNS

7. Fixed Size Window

When: Problems ask for "subarray of size k"

```python
window_sum = sum(arr[:k])
max_sum = window_sum
for i in range(k, len(arr)):
    window_sum = window_sum - arr[i-k] + arr[i]
    max_sum = max(max_sum, window_sum)
```

8. Variable Size Window

When: "Find smallest/largest subarray with condition"

```python
left = 0
for right in range(len(arr)):
    # Add arr[right] to window
    while window_condition_violated():
        # Remove arr[left] from window
        left += 1
    # Update result with current window
```

Examples: Longest Substring Without Repeating Characters, Minimum Window Substring

---

🔄 RECURSION PATTERNS

9. Divide & Conquer

When: Problem can be broken into similar subproblems

```python
def solve(arr, start, end):
    if start >= end:  # Base case
        return base_result

    mid = (start + end) // 2
    left_result = solve(arr, start, mid)
    right_result = solve(arr, mid + 1, end)

    return combine(left_result, right_result)
```

Examples: Merge Sort, Quick Sort, Binary Search

10. Backtracking Pattern

When: Exploring all possibilities, need to "undo" choices

```python
def backtrack(path, options):
    if is_solution(path):
        result.append(path[:])  # Found solution
        return

    for option in options:
        # Make choice
        path.append(option)

        # Recurse
        backtrack(path, new_options)

        # Undo choice (backtrack)
        path.pop()
```

Examples: Generate Parentheses, Permutations, N-Queens

11. Tree Recursion Pattern

When: Working with trees

python

```python
def tree_function(root):
  if not root:  # Base case
    return base_value

  left_result = tree_function(root.left)
  right_result = tree_function(root.right)

  return process(root.val, left_result, right_result)
```

📊 DYNAMIC PROGRAMMING PATTERNS

12. 1D DP Pattern

When: Current state depends on previous states

python

```python
dp = [0] * (n + 1)
dp[0] = base_case
for i in range(1, n + 1):
  dp[i] = function_of(dp[i-1], dp[i-2], ...)
```

Examples: Fibonacci, Climbing Stairs, House Robber

13. 2D DP Pattern

When: Two changing variables

python

```python
dp = [[0] * (n + 1) for _ in range(m + 1)]
for i in range(1, m + 1):
  for j in range(1, n + 1):
    dp[i][j] = function_of(dp[i-1][j], dp[i][j-1], ...)
```

Examples: Unique Paths, Edit Distance, Longest Common Subsequence

14. DP with State Compression

When: Only need previous row/column

python

```python
prev = [0] * n
for i in range(m):
    curr = [0] * n
    for j in range(n):
        curr[j] = function_of(prev[j], curr[j-1])
    prev = curr
```

🌳 TREE PATTERNS

15. Tree Traversal Patterns

python

```python
# Preorder (Root → Left → Right)
def preorder(root):
    if root:
        process(root)
        preorder(root.left)
        preorder(root.right)

# Inorder (Left → Root → Right)
def inorder(root):
    if root:
        inorder(root.left)
        process(root)
        inorder(root.right)

# Postorder (Left → Right → Root)
def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        process(root)
```

16. Level-Order (BFS) Pattern

python

```python
from collections import deque
queue = deque([root])
while queue:
    level_size = len(queue)
    for _ in range(level_size):
        node = queue.popleft()
        process(node)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
```

🕸 GRAPH PATTERNS

17. DFS Pattern

python

```python
def dfs(node, visited):
    if node in visited:
        return
    visited.add(node)
    process(node)
    for neighbor in graph[node]:
        dfs(neighbor, visited)
```

18. BFS Pattern

python

```python
from collections import deque
queue = deque([start])
visited = set([start])
while queue:
    node = queue.popleft()
    process(node)
    for neighbor in graph[node]:
        if neighbor not in visited:
            visited.add(neighbor)
            queue.append(neighbor)
```

19. Union-Find Pattern

When: Finding connected components, cycle detection

```python
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        px, py = self.find(x), self.find(y)
        if px != py:
            self.parent[px] = py
```

⚡ SORTING PATTERNS

20. Comparison-Based Sorting

```python
# Bubble Sort Pattern
for i in range(n):
    for j in range(n - 1 - i):
        if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Quick Sort Pattern
def quicksort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quicksort(arr, low, pi - 1)
        quicksort(arr, pi + 1, high)
```

21. Counting/Bucket Sort Pattern

When: Limited range of values

```python
```

```
count = [0] * (max_val + 1)
for num in arr:
    count[num] += 1

result = []
for i, freq in enumerate(count):
    result.extend([i] * freq)
```

🎯 GREEDY PATTERNS

22. Greedy Choice Pattern

When: Local optimal leads to global optimal

```python
arr.sort(key=lambda x: some_criteria(x))
result = []
for item in arr:
    if can_add(item, result):
        result.append(item)
```

Examples: Activity Selection, Huffman Coding

🔧 SPECIALIZED PATTERNS

23. Monotonic Stack Pattern

When: Finding next/previous greater/smaller element

```python
stack = []
result = []
for i, num in enumerate(arr):
    while stack and arr[stack[-1]] < num:
        idx = stack.pop()
        result[idx] = num
    stack.append(i)
```

24. Trie Pattern

When: Working with prefixes, word searches

```python
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        current = self.root
        for char in word:
            if char not in current.children:
                current.children[char] = TrieNode()
            current = current.children[char]
        current.is_end = True
```

25. Binary Search Pattern

When: Finding boundaries, peak elements

```python
def binary_search_template(arr):
    left, right = 0, len(arr) - 1
    while left < right:
        mid = (left + right) // 2
        if condition(arr[mid]):
            right = mid  # Search left half
        else:
            left = mid + 1  # Search right half
    return left
```

26. Prefix Sum Pattern

When: Range sum queries

```python
```

```
prefix_sum = [0]
for num in arr:
    prefix_sum.append(prefix_sum[-1] + num)

# Range sum from i to j
range_sum = prefix_sum[j + 1] - prefix_sum[i]
```

27. Bit Manipulation Patterns

```python
# Check if bit is set
if n & (1 << i):
    # i-th bit is set

# Set bit
n |= (1 << i)

# Clear bit
n &= ~(1 << i)

# Toggle bit
n ^= (1 << i)
```

🏆 MASTER THESE FIRST (Priority Order)

Week 1-2: Patterns 1-3 (Traversal, Search, Build-As-You-Go) **Week 3-4:** Patterns 4-6 (Two Pointers) **Week 5-6:** Patterns 7-8 (Sliding Window) **Week 7-8:** Patterns 9-11 (Basic Recursion) **Week 9-10:** Patterns 15-16 (Tree Traversal) **Week 11-12:** Patterns 17-18 (Graph DFS/BFS)

Advanced: Patterns 12-14 (DP), 19-27 (Specialized)

🎯 How to Use This List

1. **Pick ONE pattern per week**
2. **Solve 5-10 problems using that pattern**
3. **Don't move to next pattern until you can solve problems without help**
4. **Practice writing the pattern template from memory**

Remember: These patterns solve 95% of all coding interview problems. Master the first 18 patterns and you'll be unstoppable! 🚀