Chapter 9: Coding

30문제 패턴 분류

면접에서 반복 출제되는 7가지 핵심 패턴

문제 전체 난이도 분류

난이도	문제 번호	출제 기업
Easy	9.1 ~ 9.6	Amazon, D.E. Shaw, Facebook, Google, Akuna
Medium	9.7 ~ 9.24	Google, AQR, Amazon, Facebook, LinkedIn, Yelp, Goldman Sachs, Two Sigma, Workday, Palantir, Citadel, Uber
Hard	9.25 ~ 9.30	Citadel, Bloomberg, Google, Two Sigma

7가지 핵심 패턴 맵

패턴	대표 문제	핵심 도구
Array / String	9.1, 9.2, 9.5, 9.7, 9.12, 9.14	Set, Heap, Sliding Window, Sorting
Hash Map	9.1, 9.12, 9.14, 9.20	Dict 빈도수, Anagram 비교
Linked List	9.16	Two-pointer, 재귀 카운팅
Tree / Graph	9.6, 9.9, 9.11, 9.15, 9.26	BFS, DFS, 재귀
Dynamic Programming	9.21, 9.22, 9.23, 9.25, 9.29	Memoization, Bottom-up DP
Recursion / Backtracking	9.19, 9.24, 9.18	Permutation, Combination, Stack
Sorting / Searching	9.3, 9.4, 9.7, 9.13, 9.20	Heap, Binary Search, Interval sort

Pattern 1

Array / String 조작

Set, Heap, Kadane 알고리즘, Sliding Window

9.1 Amazon: 두 배열의 교집합

- **문제**: 두 배열 A, B의 교집합 반환
- **Brute force**: 이중 루프 → O(N*M)
- **최적 해법**: **Set 변환** 후 짧은 쪽 순회하며 lookup
- O(1) lookup이 핵심 — 전체 O(N+M)

```
def intersection(a, b):
    set_a, set_b = set(a), set(b)
    if len(set_a) < len(set_b):
        return [x for x in set_a if x in set_b]
    else:
        return [x for x in set_b if x in set_a]
```

- **Time**: O(N+M) / **Space**: O(N+M)

9.2 D.E. Shaw: 세 수의 최대 곱

- **핵심 함정**: 음수 2개의 곱이 양수가 될 수 있음
- 면접관에게 "음수 포함 여부" 반드시 확인할 것
- **전략**: 가장 큰 3개 vs 가장 작은 2개 * 가장 큰 1개 비교

```python
import heapq
def max_three(arr):
    a = heapq.nlargest(3, arr)
    b = heapq.nsmallest(2, arr)
    return max(a[0]*a[1]*a[2], b[0]*b[1]*a[0])
```

- Heap 사용 → **Time**: O(N) / **Space**: O(1)
- 정렬 기반도 가능 (O(N log N))

9.5 Akuna Capital: 최대 연속 부분배열 합 (Kadane)

- Kadane's Algorithm: 단일 순회로 해결하는 고전 패턴
- 핵심 아이디어: `curr_sum < 0` 이면 리셋 (이전 구간 버림)
- 모든 원소가 음수면 결과 = 0 (빈 부분배열)

```
def max_subarray(arr):
    max_sum, curr_sum = arr[0], 0
    for x in arr:
        curr_sum += x
        max_sum = max(max_sum, curr_sum)
        if curr_sum < 0:
            curr_sum = 0
    return max_sum
```

- Time: O(N) / Space: O(1) — 면접 단골 문제

9.7 Google: Peak Element 찾기 (변형 이진탐색)

Brute Force

- 전체 순회 O(N)
- 각 원소의 좌우 비교

Binary Search 변형

- **중간값의 오른쪽이 더 크면** → 우측에 반드시 peak 존재
- O(log N)으로 개선

```
def
```

Pattern 2

Hash Map 활용

빈도수 카운팅, Anagram 매칭, Sliding Window

9.12 LinkedIn: 문자열 내 Anagram 위치 찾기

- A 안에서 B의 anagram인 부분문자열의 시작 인덱스 반환
- **Sliding Window + Dictionary 비교**가 핵심
- 정렬 비교(O(K log K)) 대신 빈도 딕셔너리 비교(O(K))

알고리즘 흐름:

1. B의 문자 빈도 딕셔너리 생성
2. A에서 크기 K 윈도우의 빈도 딕셔너리 유지
3. 윈도우 이동 시 좌측 문자 제거, 우측 문자 추가
4. 두 딕셔너리 일치 시 인덱스 기록

- **Time**: O(NK) / **Space**: O(K)

9.12 LinkedIn: Anagram 코드

```
def total_anagrams(A, B):
    n, k = len(A), len(B)
    if n < k: return []

    def add(char, d):
        d[char] = d.get(char, 0) + 1
        return d

    d_a, d_b = {}, {}
    for i in range(k):
        d_a = add(A[i], d_a)
        d_b = add(B[i], d_b)

    start, res = 0, []
    for i in range(k, n + 1):
        if d_a == d_b:      # O(K) 비교
            res.append(start)
        if i < n:
            d_a = add(A[i], d_a)
            d_a[A[start]] -= 1
            start += 1
    return res
```

9.14 Goldman Sachs: Anagram 그룹핑

- 문자열 배열 → 같은 anagram끼리 그룹화
- **핵심**: 문자 빈도 딕셔너리를 **정렬된 문자열 키**로 변환
  - `"abc"` → `"a1b1c1"`, `"cab"` → `"a1b1c1"` (같은 키)
- 최종 딕셔너리에서 키별 리스트 수집

```
def anagram_group(str_list):
    groups = {}
    for s in str_list:
        cfd = {}
        for c in s:
            cfd[c] = cfd.get(c, 0) + 1
        key = ''.join(f'{k}{v}' for k, v
              in sorted(cfd.items()))
        groups.setdefault(key, []).append(s)
    return list(groups.values())
```

- Time: O(NK log K) / Space: O(NK)

Pattern 3

Sorting & Searching

Heap, Binary Search, Interval 정렬

9.3 Facebook: K-최근접점 (Min-Heap)

- N개 좌표 중 원점에서 가장 가까운 K개 반환
- 정렬 O(N log N) 대신 **Min-Heap**으로 O(N log K)

```
from heapq import heappush, heappop

def closest(points, k):
    heap = []
    for x, y in points:
        heappush(heap, (x**2 + y**2, [x, y]))
    return [heappop(heap)[1] for _ in range(k)]
```

- **Time**: O(N log K) / **Space**: O(K)
- 유사 패턴: Top-K 문제 전반에 적용 가능

9.4 Google: 정렬된 행렬에서 K번째 작은 수

- 행/열 모두 정렬된 행렬에서 K번째 최솟값 찾기
- **핵심 관찰**: K번째 최솟값은 상위 K개 행/열 안에 존재
- Min-Heap에 해당 범위만 넣고 K번 pop

```
from heapq import heappush, heappop

def kth_smallest(m, k):
    n = len(m)
    heap = []
    for i in range(min(k, n)):
        for j in range(min(k, n)):
            heappush(heap, m[i][j])
    res = -1
    for _ in range(k):
        res = heappop(heap)
    return res
```

- Time: O(K^2 log K) / Space: O(K)

9.13 Yelp: 최소 구간 제거 (Interval Scheduling)

- 구간 리스트에서 겹침 없도록 최소 제거 수 계산
- **전략**: 시작-종료 기준 정렬 후 겹침 감지

```
def interval_removal(intervals):
    if not intervals: return 0
    intervals.sort(key=lambda k: (k[0], k[1]))
    low, count = 0, 0
    for high in range(1, len(intervals)):
        if intervals[low][1] > intervals[high][0]:
            count += 1
            if not (intervals[high][0] < intervals[low][1]
                    < intervals[high][1]):
                low = high  # merge
    return count
```

- Time: O(N log N) 정렬 / Space: O(1)

9.20 Two Sigma: 가중 확률 샘플링

단순 접근

- 가중치만큼 원소 복제 후 `random.choice`
- Space: O(N) — 가중치 합이 크면 비효율

최적 접근 (Cumulative Sum + Binary Search)

- 누적합 배열 생성: `[5, 15, 30, 50]`
- 0~총합 사이 난수 → 이진탐색으로 카테고리 결정
- Time: O(K) / Space: O(K)
  - K = 카테고리 수 (가중치 합 N과 무관)

Pattern 4

Linked List 조작

재귀 카운팅, 딕셔너리 캐싱

9.16 Workday: 뒤에서 K번째 노드 제거

- 연결 리스트의 끝에서 K번째 노드를 제거하고 head 반환
- **전략**: 재귀로 노드 순번을 딕셔너리에 캐싱

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def remove(head, k):
    def count(node, d):
        if node.next is None:
            d[1] = node
            return 1
        res = count(node.next, d)
        d[1 + res] = node
        return 1 + res

    d = {}
    n = count(head, d)
    if k == n:
        return d[k-1] if n > 1 else None
```

Pattern 5

Tree & Graph 순회

재귀, BFS, DFS, 인접행렬

9.6 Facebook: 트리의 거울 대칭 판별

- 이진 트리가 자기 자신의 거울인지 판별
- **재귀 조건**: 좌측의 왼쪽 == 우측의 오른쪽, 좌측의 오른쪽 == 우측의 왼쪽

```
def is_mirror(root):
    if root is None: return True
    return helper(root.left, root.right)

def helper(x, y):
    if x is None and y is None: return True
    if x is None or y is None: return False
    return (x.val == y.val and
            helper(x.left, y.right) and
            helper(x.right, y.left))
```

- Time: O(N) / Space: O(N) — 재귀 스택

9.9 Amazon: 트리의 지름 (Diameter)

- 임의의 두 노드 사이 **가장 긴 경로**의 길이
- **핵심**: 각 노드에서 지름 = 좌측 깊이 + 우측 깊이
- 깊이 계산 중 지름을 동시에 갱신하는 헬퍼 함수

```
def calc_diameter(root):
    def depth(root, diameter):
        if root is None:
            return 0, diameter
        left, diameter = depth(root.left, diameter)
        right, diameter = depth(root.right, diameter)
        diameter = max(diameter, left + right)
        return max(left, right) + 1, diameter

    _, diameter = depth(root, 0)
    return diameter
```

- Time: O(N) / Space: O(N)

9.11 Facebook: 소셜 그래프 최단 거리 (BFS)

- 두 유저 간 최소 친구 관계 수 = **최단 경로** → BFS
- 거리 배열 + 방문 체크 + 큐 사용

```
import queue
def friendship_distance(n, edges, x, y):
    distance = [0] * n
    processed = [False] * n
    q = queue.Queue()
    q.put(x)
    processed[x] = True
    while not q.empty():
        curr = q.get()
        if curr not in edges: continue
        for neighbor in edges[curr]:
            if not processed[neighbor]:
                distance[neighbor] = distance[curr] + 1
                q.put(neighbor)
                processed[neighbor] = True
    return distance[y]
```

9.15 Two Sigma: 친구 그룹 수 세기 (DFS)

- 인접행렬에서 연결된 컴포넌트(친구 그룹) 수 구하기
- **패턴**: DFS로 한 그룹 전체 탐색 → 미방문 노드에서 새 DFS

```
def count_groups(N):
    def dfs(friends, i, N):
        friends.add(i)
        for j in range(len(N[i])):
            if N[i][j] == 1 and j not in friends:
                dfs(friends, j, N)

    num = len(N[0])
    groups, friends = 0, set()
    for i in range(num):
        if i not in friends:
            dfs(friends, i, N)
            groups += 1
    return groups
```

9.26 Bloomberg: 행렬 최장 증가 경로 (DFS + Memoization)

- m x n 행렬에서 인접 셀로만 이동하는 최장 증가 경로 길이
- **DFS + 테이블 캐싱** — 방문 셀 결과 재사용

```
def longest_increasing_path(mx):
    table = {}
    m, n = len(mx), len(mx[0])

    def dfs(i, j, prev):
        if (i < 0 or i >= m or j < 0 or j >= n
            or mx[i][j] <= prev):
            return 0
        if (i, j) in table:
            return table[(i, j)]
        curr = 1 + max(dfs(i-1,j,mx[i][j]),
                       dfs(i+1,j,mx[i][j]),
                       dfs(i,j-1,mx[i][j]),
                       dfs(i,j+1,mx[i][j]))
        table[(i, j)] = curr
```

Pattern 6

Dynamic Programming

Bottom-up 테이블, 부분문제 재활용

DP 문제 접근 프레임워크

1. 부분문제 정의

- `dp[i]` 또는 `dp[i][j]` 가 무엇을 의미하는지 명확히

2. 점화식 도출

- `dp[i]` 와 `dp[i-1]` (또는 이전 상태들) 사이 관계

3. Base Case

- `dp[0]` , `dp[0][0]` 등 초기값 설정

4. 순회 방향

- Bottom-up (반복) 또는 Top-down (재귀+메모)

9.21 Amazon: 최장 공통 부분배열 (LCS 변형)

- 두 배열에서 **연속된** 공통 부분배열의 최대 길이
- `dp[i][j]` = `a[:i]`와 `b[:j]`의 최장 공통 부분배열 길이
- `a[i-1] == b[j-1]`이면 `dp[i][j] = 1 + dp[i-1][j-1]`

```
def longest_common(a, b):
    m, n = len(a), len(b)
    dp = [[0]*(n+1) for _ in range(m+1)]
    max_val = 0
    for i in range(1, m+1):
        for j in range(1, n+1):
            if a[i-1] == b[j-1]:
                dp[i][j] = 1 + dp[i-1][j-1]
                max_val = max(max_val, dp[i][j])
    return max_val
```

- Time / Space: O(MN)

9.22 Uber: 최대 증가 부분수열 합 (MISS)

- 증가하는 부분수열 중 **합이 최대**인 것
- `res[i]` = 인덱스 i까지의 MISS
- `arr[j] < arr[i]` 이고 `res[i] < res[j] + arr[i]` 이면 갱신

```
def max_subseq_sum(arr):
    n = len(arr)
    res = arr[:]            # 초기값: 자기 자신
    for i in range(1, n):
        for j in range(i):
            if arr[j] < arr[i] and res[i] < res[j] + arr[i]:
                res[i] = res[j] + arr[i]
    return max(res)
```

- 예: `[3, 2, 5, 7, 6]` → 3+5+7 = **15**
- **Time**: O(N^2) / **Space**: O(N)

9.23 Palantir: 최소 제곱수 합 (Perfect Squares)

- 양의 정수 n을 제곱수 합으로 표현할 때 최소 개수
- **Greedy 함정**: 41 = 36+4+1 (3개) vs 25+16 (2개)
- 점화식: `res[i] = min(res[i], res[i - j^2] + 1)`

```
def square_count(n):
    res = list(range(n + 1))  # 최악: 모두 1의 합
    for i in range(2, n + 1):
        for j in range(1, int(i**0.5) + 1):
            res[i] = min(res[i], res[i - j**2] + 1)
    return res[n]
```

- **Time**: O(N sqrt(N)) / **Space**: O(N)

9.25 Citadel: 최장 유효 괄호 부분문자열 (Hard)

- 올바르게 매칭된 괄호의 **최장 연속 길이**
- **Stack에 인덱스 저장** — 현재 위치와 스택 top의 차이 = 길이

```
def longest_parens(s):
    stack = [-1]    # 더미 값 (첫 ')' 처리)
    longest = 0
    for i in range(len(s)):
        if s[i] == '(':
            stack.append(i)
        else:
            stack.pop()
            if not stack:
                stack.append(i)
            else:
                longest = max(longest, i - stack[-1])
    return longest
```

- 예: `")(())("` → `"(())"` → 4

9.29 Two Sigma: Wildcard 패턴 매칭 (Hard)

- `?` = 아무 문자 1개, `*` = 0개 이상 아무 문자
- **2D DP 테이블**: `dp[i][j]` = `string[:i]`가 `regex[:j]`에 매칭?

점화식:

1. `r[j-1]`이 `?` 또는 `s[i-1]`과 같으면: `dp[i][j] = dp[i-1][j-1]`
2. `r[j-1]`이 `*`이면: `dp[i][j] = dp[i-1][j-1] or dp[i][j-1] or dp[i-1][j]`

```
dp = [[False]*(n+1) for _ in range(m+1)]
dp[0][0] = True
# '*'로만 구성된 regex prefix → True
for j in range(1, n+1):
    if regex[j-1] != '*': break
    dp[0][j] = True
```

- **Time / Space**: O(MN)

Pattern 7

Recursion & Backtracking

순열, 조합, 스택 기반 유효성

9.18 Palantir: 최소 괄호 제거로 유효 문자열 만들기

- 스택으로 유효한 괄호 쌍 추적, 나머지 제거
- `(` → 스택에 인덱스 push / `)` → pop 후 양쪽 인덱스 기록

```
def split_paren(s):
    stk = []
    res = [''] * len(s)
    for idx, val in enumerate(s):
        if val == '(':
            stk.append(idx)
        elif val == ')':
            if stk:
                latest = stk.pop()
                res[latest] = s[latest]
                res[idx] = val
        else:
            res[idx] = val
    return ''.join(res)
```

9.19 Citadel: 모든 순열 생성 (Permutation)

- 핵심: 각 원소를 고정하고, 나머지로 재귀 호출
- Base case: 길이 1이면 자기 자신 반환

```
def permute(nums):
    if len(nums) <= 1:
        return [nums]
    res = []
    for i in range(len(nums)):
        rest = nums[:i] + nums[i+1:]
        for combo in permute(rest):
            res.append([nums[i]] + combo)
    return res
```

- `permute([2,3,4])` → 6개 순열
- **Time**: O(N!) / **Space**: O(N * N!)
- 면접에서 "왜 N!인가?" 설명할 수 있어야 함

9.24 Facebook: 조합 생성 (Backtracking)

- 1~n에서 k개를 뽑는 모든 조합
- **Backtracking 3단계**: 추가 → 재귀 → 제거

```
def combos(n, k):
    def backtrack(res, combo, num, start):
        if num == k:
            res.append(list(combo))
            return
        if start > n or num >= k:
            return
        for i in range(start, n + 1):
            combo.append(i)             # 추가
            backtrack(res, combo, num+1, i+1)  # 재귀
            combo.remove(i)             # 제거

    res = []
    backtrack(res, [], 0, 1)
    return res
```

특수 패턴

통계, 시뮬레이션, 최적화

면접에서 수학+코딩 융합 문제

9.8 AQR: 상관계수 직접 구현

공식

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \cdot \sigma_Y}$$

$$Cov(X,Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

구현 포인트

- `mean()`, `sd()` 헬퍼 함수 분리
- 편차 곱의 평균 = 공분산
- Time: O(N) / Space: O(N)

```
def corr(x, y):
    x_m, y_m = mean(x), mean(y)
    cov = mean([(x[i]-x_m)*(y[i]-y_m)
                for i in range(len(x))])
    return cov / (sd(x) * sd(y))
```

9.10 D.E. Shaw: 조건부 랜덤 정수 생성

- 합 = target, 평균에서 c*sigma 이내의 n개 정수 생성
- **알고리즘**: 하한값으로 초기화 → 랜덤 인덱스 +1씩 증가

```
import random
def generate_nums(target, n, sigma):
    mean = target / n
    sd = int(sigma * mean)
    max_val, min_val = mean + sd, mean - sd
    results = [min_val] * n
    remaining = target - n * min_val
    while remaining > 0:
        a = random.randint(0, n - 1)
        if results[a] >= max_val:
            continue
        results[a] += 1
        remaining -= 1
    return results
```

9.17 Goldman Sachs: Monte Carlo로 pi 추정

- 단위 정사각형 안에서 (x, y) 랜덤 샘플링
- $x^2 + y^2 \leq 1$이면 원 내부 → 비율 * 4 = pi

```
import random, math
def estimate_pi(n=10_000_000):
    count = 0
    for _ in range(n):
        x = random.random()
        y = random.random()
        if math.sqrt(x**2 + y**2) < 1.0:
            count += 1
    return (count / n) * 4
```

- Time: O(N) / Space: O(1)
- 면접 포인트: 수렴 속도는 $O(1/\sqrt{N})$

9.27 Google: 연속 정수의 합으로 n 표현하기

- 연속 양의 정수 리스트 중 합이 n인 것의 개수
- **수학적 최적화**: k에서 시작, m+1개 항

$$k + (k+1) + \cdots + (k+m) = \frac{(2k+m)(m+1)}{2} = n$$

$$k = \frac{2n/(m+1) - m}{2}$$

- k가 양의 정수 ↔ 우변이 짝수인 양의 정수
- m의 범위: $1 \leq m < \sqrt{2n}$

```
import math
def consecutive_sum(n):
    upper = int(math.sqrt(2 * n))
    return sum(1 for m in range(upper)
               if (2*n) % (m+1) == 0
```

9.28 Citadel: 스트리밍 중앙값 (Dual Heap)

- 연속 입력 스트림에서 실시간 중앙값 계산
- Max-Heap(작은 절반) + Min-Heap(큰 절반) 유지

```
import heapq
class MedianFinder:
    def __init__(self):
        self.min_heap = []    # 큰 절반
        self.max_heap = []    # 작은 절반 (음수 저장)

    def add_num(self, num):
        heapq.heappush(self.max_heap, -num)
        heapq.heappush(self.min_heap,
                       -heapq.heappop(self.max_heap))
        if len(self.min_heap) > len(self.max_heap):
            heapq.heappush(self.max_heap,
                           -heapq.heappop(self.min_heap))

    def find_median(self):
        if len(self.max_heap) > len(self.min_heap):
```

9.30 Citadel: 최적 위치 (Gradient Descent)

- N개 집 좌표가 주어질 때, 총 유클리드 거리 최소인 소방서 위치
- Gradient Descent: 볼록 함수 → 전역 최적해 보장

편미분:

$$\frac{\partial L}{\partial x_c} = \sum_{i=1}^{n} \frac{x_c - x_i}{\sqrt{(x_c - x_i)^2 + (y_c - y_i)^2}}$$

하이퍼파라미터: 초기 학습률, 감쇠율, 종료 임계값, 감쇠 인자

- 초기값 = 좌표들의 무게중심 (centroid)
- Time: O(N * x_d * y_d) / Space: O(N)

복잡도 요약 — Easy 문제 (9.1 ~ 9.6)

#	기업	핵심 패턴	Time	Space
9.1	Amazon	Set lookup	O(N+M)	O(N+M)
9.2	D.E. Shaw	Heap (nlargest)	O(N)	O(1)
9.3	Facebook	Min-Heap	O(N log K)	O(K)
9.4	Google	Min-Heap	O(K^2 log K)	O(K)
9.5	Akuna	Kadane's	O(N)	O(1)
9.6	Facebook	Tree 재귀	O(N)	O(N)

복잡도 요약 — Medium 문제 (9.7 ~ 9.17)

#	기업	핵심 패턴	Time	Space
9.7	Google	Binary Search	O(log N)	O(1)
9.8	AQR	통계 구현	O(N)	O(N)
9.9	Amazon	Tree 재귀	O(N)	O(N)
9.10	D.E. Shaw	Random + 범위 제약	O(sigma*T)	O(N)
9.11	Facebook	BFS	O(N+M)	O(N)
9.12	LinkedIn	Sliding Window	O(NK)	O(K)
9.13	Yelp	Interval Sort	O(N log N)	O(1)
9.14	Goldman	Dict 그룹핑	O(NK log K)	O(NK)

복잡도 요약 — Medium/Hard 문제 (9.18 ~ 9.30)

#	기업	핵심 패턴	Time	Space
9.18	Palantir	Stack 괄호	O(N)	O(N)
9.19	Citadel	재귀 순열	O(N!)	O(N*N!)
9.20	Two Sigma	누적합+이진탐색	O(K)	O(K)
9.21	Amazon	2D DP (LCS)	O(MN)	O(MN)
9.22	Uber	1D DP (MISS)	O(N^2)	O(N)
9.23	Palantir	DP (제곱수)	O(N sqrt N)	O(N)
9.24	Facebook	Backtracking	O(C(n,k))	O(C(n,k))
9.25	Citadel	Stack DP	O(N)	O(N)

면접 전략 — 패턴별 출제 빈도

Array / HashMap (40%)

- 가장 자주 출제
- Set, Dict, Sliding Window 숙달 필수
- Brute force → Optimal 설명 패턴

DP (20%)

- 면접관에게 점화식 먼저 설명
- Bottom-up 선호 (스택 오버플로 방지)
- 공간 최적화 가능 여부 언급

Tree / Graph (25%)

- BFS = 최단경로, DFS = 완전탐색
- 재귀적 사고 + Base case 정의
- 인접행렬 vs 인접리스트 구분

Backtracking / 기타 (15%)

- 순열, 조합 = 재귀의 정석
- Monte Carlo, Gradient Descent 등 수학 기반
- 복잡도 분석이 곧 실력 증명

면접에서 꼭 기억할 5가지

❶ **Brute force부터 시작**: 최적해 전에 단순 해법을 먼저 제시하고, 개선 과정을 보여줌

❷ **Edge case 확인**: 빈 배열, 음수, 단일 원소, 모두 같은 값 등 코너 케이스를 면접관에게 질문

❸ **복잡도를 반드시 분석** : Time/Space 모두 설명 — "이 풀이의 시간복잡도는 O(N log N)이며..."

❹ **패턴 인식이 핵심**: 30문제 모두 7가지 패턴에 매핑됨 — 유형을 빠르게 파악하는 훈련

❺ **코드보다 소통**: 생각의 과정을 실시간으로 설명하며 구현 — 침묵하며 코딩하는 것은 감점