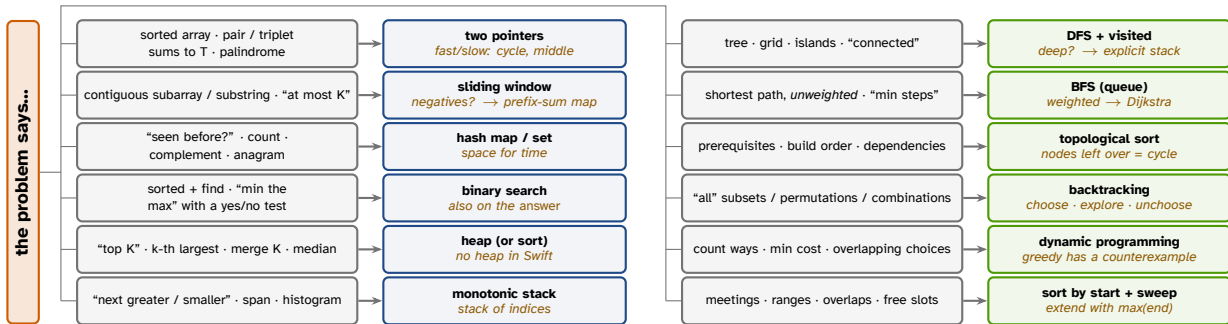


In one line: Don't invent — **recognise**. Read the constraints first: $n \leq 20 \rightarrow$ exponential/backtracking is fine; $n \leq 10^4 \rightarrow O(n^2)$ ok; $n \geq 10^5 \rightarrow$ you need $O(n \log n)$ or $O(n)$; " $O(\log n)$ " or $n \approx 10^9 \rightarrow$ binary search. Then match the *wording* to a pattern below, say the brute force, name the repeated work, and swap in the pattern that removes it.



Two pointers $O(n) \cdot O(1)$

cue: sorted, pair/triplet, palindrome, in-place dedupe. Ends move inward; fast/slow finds a cycle or the middle.

```
var l = 0, r = a.count - 1
while l < r {
  let s = a[l] + a[r]
  if s == t { return (l, r) }
  if s < t { l += 1 } else { r -= 1 } }
```

Sliding window $O(n)$ amortised

cue: contiguous + constraint. **Fixed k:** sum += a[i] - a[i-k]. **Variable:** grow right, shrink left while invalid — each index enters and leaves once.

```
var l = 0
for r in 0..
```

Hashing $O(n) \cdot O(n)$ space

cue: complement: look up *before* inserting (no self-pair) · **frequency** · **canonical key** (sorted chars) to group anagrams.

```
if let j = seen[t - x] { return [j, i] }
seen[x] = i
freq[c, default: 0] += 1
```

Prefix-sum map (sum = k, negatives ok), seed [0: 1]:
p += x
count += seen[p - k, default: 0]
seen[p, default: 0] += 1

Binary search $O(\log n)$

cue: sorted, first/last position, insert point. One template (half-open lower bound), never mix:
var lo = 0, hi = a.count // [lo, hi)
while lo < hi {
 let m = lo + (hi - lo) / 2 // no overflow
 if a[m] < t { lo = m + 1 } else { hi = m } }
// lo = first index with a[lo] >= t

On the answer (min speed / capacity, "minimise the max"): ok(x) flips once $\rightarrow$ lo = minAns, hi = maxAns, then if ok(m) { hi = m } else { lo = m + 1 }. $O(n \log \text{range})$.

Heap / top-K $O(n \log k)$

cue: top K, k-th largest, merge K lists, running median (two heaps). A **min**-heap of size k keeps the k **largest**. **Swift has no heap** — say so, then: sort $O(n \log n)$, a small binary heap, or quickselect (avg $O(n)$). Top-K frequent: bucket by count, $O(n)$.

```
let top = freq.sorted { $0.value > $1.value }
.prefix(k).map { .key }
```

Monotonic stack $O(n)$ amortised

cue: next greater/smaller, stock span, daily temperatures, largest rectangle. Keep *indices*; pop while the new value beats the top.

```
for i in 0..
```

BFS / DFS + visited $O(V + E)$

cue: BFS = levels = shortest *unweighted* path; DFS = connectivity, flood fill, cycles. Mark visited **on enqueue**.

```
var q = [s], head = 0, seen: Set = [s]
while head < q.count { // head: O(1) pop
  let u = q[head]; head += 1
  for v in adj[u] where seen.insert(v).inserted {
    q.append(v) } }
```

Grid: 4 neighbours, bounds-check *before* subscribing. Directed cycle: an onPath set, separate from visited.

Topological sort (Kahn) $O(V + E)$

cue: "must come before". In-degree $\rightarrow$ queue; pop, decrement neighbours. order.count < n $\rightarrow$ **cycle**. [a, b] = "a needs b" = edge b $\rightarrow$ a.

Backtracking $O(n \cdot 2^n) / O(n \cdot n!)$

cue: "all" subsets / permutations / combos / N-Queens. **Choose** $\rightarrow$ **explore** $\rightarrow$ **unchoose**; prune invalid branches early.

```
func bt(_ start: Int) {
  res.append(path)
  for i in start..
```

Reuse allowed $\rightarrow$ bt(i); permutations $\rightarrow$ a used array, loop from 0.

Dynamic programming states $\times$ transition

cue: count ways / min / max / feasible, and the brute force re-solves the same sub-inputs. Recipe: **state in one sentence** $\rightarrow$ recurrence $\rightarrow$ base cases $\rightarrow$ iteration order $\rightarrow$ answer cell. **Memo** (top-down: recursion + cache, only reachable states, deep stack) vs **tabulation** (bottom-up loop, easy to shrink to one row / two scalars). Same big-O.

The 5 classic shapes

1D linear: dp[i] = max(dp[i-1], dp[i-2]+x) — stairs, robber

Unbounded choice: dp[a] = min(dp[a], dp[a-c]+1), a ascending — coin change

0/1 knapsack: one row, capacity **descending** $\Rightarrow$ each item once

Two sequences: dp[i][j], table (m+1) $\times$ (n+1), compare a[i-1] — LCS, edit distance

Grid paths: dp[r][c] = dp[r-1][c] + dp[r][c-1], first row/col = 1

```
var memo: [Int: Int] = [:]
func ways(_ i: Int) -> Int {
  if i <= 1 { return 1 }
  if let v = memo[i] { return v }
  let v = ways(i - 1) + ways(i - 2)
  memo[i] = v; return v }
```

Intervals $O(n \log n)$

cue: sort by start, sweep; extend with max(end, iv.end) — a contained interval must not shrink it. Touching ([1, 4], [4, 5]): decide and say it. Meeting rooms = min-heap of end times, or sorted starts vs sorted ends.

Swift gotchas in a coding round

- String has no s[i] — let c = Array(s) once ([Character], $O(1)$ index). s.count is $O(n)$.
- Character $\neq$ String: "a" as Character; c.asciiValue! (UInt8); c.isLetter, c.isNumber.
- dict[k, default: 0] += 1; plain lookup returns an *Optional*.
- No heap, no deque: removeFirst() is $O(n)$ $\rightarrow$ keep a head index.
- Tuples aren't Hashable $\rightarrow$ struct P: Hashable { let r, c: Int }.
- Int overflow **traps**: midpoint lo + (hi-lo)/2; ∞ sentinel amount + 1, not Int.max.
- 2D: Array(repeating: Array(repeating: 0, count: n), count: m).
- a.max() is Optional; stride(from: n-1, through: 0, by: -1); swapAt; sorted(by:) returns a copy.
- Deep recursion crashes (iOS main-thread stack is 1 MB, secondary 512 KB) $\rightarrow$ explicit stack for degenerate trees / long paths.

Cost of the Swift basics

append · removeLast · a[i]	$O(1)$ (append amortised)
insert(at:0) · removeFirst	$O(n)$
Array.contains · firstIndex	$O(n)$
Set / Dictionary ops	$O(1)$ avg, $O(n)$ worst
sort · sorted	$O(n \log n)$
String.count · Array(s)	$O(n)$
recursion	$O(\text{depth})$ stack

Interview traps

- ! Window with negative numbers — shrinking no longer restores validity; use the prefix-sum map.
- ! BFS marking visited on *dequeue* $\rightarrow$ a node queued many times.
- ! Greedy that "looks right": coins [1, 3, 4], 6 $\rightarrow$ greedy 4+1+1, DP 3+3.
- ! Backtracking without the unchoose $\rightarrow$ paths leak into each other; res.append(path) copies (value type) — good.
- ! Claiming $O(n)$ and forgetting the $O(n)$ **space** of the map or the recursion stack.

Remember: sorted $\rightarrow$ pointers / binary · contiguous $\rightarrow$ window · seen $\rightarrow$ hash · levels $\rightarrow$ BFS · "all" $\rightarrow$ backtrack · repeated sub-answers $\rightarrow$ DP · top K $\rightarrow$ heap.

Likely questions

- Why is the window $O(n)$ despite a nested while? — each index enters and leaves once.
- Why seed the prefix map with [0: 1]? — subarrays that start at index 0.
- Memo or table? — same big-O; memo mirrors the recurrence, table avoids deep recursion and shrinks space.
- Two-sum sorted vs unsorted? — pointers $O(1)$ space vs hash $O(n)$ keeping original indices.