Basic

There are a handful of core **patterns** that appear again and again in coding interviews, and learning them (with 5–10 problems each) is one of the fastest ways to improve.[1][2]

Must-know patterns

- Two pointers (for arrays/strings where the data is sorted or you move from both ends).[3][1]
- Sliding window (for subarray/substring questions like “longest”, “smallest”, “at most K”).[1][3]
- Fast & slow pointers (cycle detection, middle of linked list, etc.).[1]
- Binary search / modified binary search (on sorted arrays, answer space, or monotonic functions).[3][1]
- DFS and BFS on trees/graphs (traversal, counting islands, shortest paths in unweighted graphs).[4][1]
- Recursion + backtracking (subsets, permutations, combinations, N-Queens, word search).[2][1]
- Merge intervals (merging/adding intervals, meeting rooms, calendar/time range problems).[2][1]
- Subsets / combinatorics (generate all subsets, combinations, partitions).[1]
- Top K elements with heaps (kth largest, k closest points, top K frequent).[5][1]
- Basic dynamic programming patterns like 0/1 knapsack, subset sum, and simple DP on sequences.[6][1]

Slightly more advanced but common

- Cyclic sort (arrays with numbers in 1..n, find missing/duplicate).[3][1]
- In-place reversal of linked list (reverse list, reverse in k-groups).[1]
- Bit manipulation / XOR tricks (find single non-duplicate, swap without temp, subsets via bits).[4][1]
- Topological sort (course schedule, dependency ordering in DAGs).[4][1]
- Monotonic stack (next greater element, largest rectangle in histogram).[7][1]

How to practice these patterns

- Pick one pattern (say sliding window), read 2–3 short explanations, then solve 8–10 LeetCode or similar problems only from that pattern.[6][1]
- After solving, write a 3–5 line “template” for that pattern in your preferred language, so you can quickly recall the structure in interviews.[6][3]

- Rotate through 4–5 high-ROI patterns first: two pointers, sliding window, DFS/BFS, binary search, backtracking.[2][4]

If you share your current level (beginner / intermediate / advanced) and preferred language, a focused 4–6 week pattern-based plan can be outlined next.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

When to stop collecting patterns
If you can recognize and implement the patterns listed in Grokking's 20 patterns and ByteByteGo's chapters (two pointers, hash maps, linked lists, sliding window, binary search, stacks, heaps, intervals, prefix sums, trees, tries, graphs, backtracking, DP, greedy, sort & search, bit manipulation, math/geometry), you already have more than enough for typical product-company interviews.

After that point, your improvement comes mainly from practice and speed, not from discovering brand-new patterns.

Beginner

Being a **beginner** is actually the best time to learn patterns properly, as long as you follow a simple, focused roadmap.roadmap+1

Step 1: Get language comfort

- Pick one language for interviews (Java, Python, C++ or JavaScript) and stick to it for all practice.techinterviewhandbook+1
- Spend 2–3 weeks writing small programs (loops, functions, arrays, strings, classes) so syntax never distracts you during DSA.dev+1

Step 2: Learn basics + big-O

- Learn what arrays, linked lists, stacks, queues, hash maps, and trees are, and roughly how fast common operations are (insert, delete, search).scholarhat+1
- Study time and space complexity (big-O) just enough to compare two solutions and talk about them in interviews.roadmap+1

Step 3: Start with easy patterns

For the first 1–2 months, you only need a small set of beginner-friendly patterns.designgurus+1

- Arrays/strings + **two pointers**
- **Sliding window** on strings/arrays
- Basic **hash map** usage (frequency counting, two-sum)
- Simple **DFS/BFS** on grids (islands) and trees

Step 4: Practice the right way

- For each pattern, learn a 10–15 line “template”, then solve 5–10 easy problems that fit that pattern (LeetCode Easy / basic GeeksforGeeks).designgurus+1
- Always code fully, run, debug, then write a short note: idea, pattern used, and complexity.github+1

Step 5: Gradually add harder topics

After you feel comfortable with the easy ones, add:scaler+1

- Recursion + simple backtracking (subsets, permutations of small sets).
- Binary search variations.
- Basic dynamic programming (Fibonacci, climbing stairs, 0/1 knapsack-lite).

If you share your current language and how many hours per day you can study, a very concrete 4–6 week beginner plan (with what to do each week and which patterns to start with) can be outlined next.

1. https://roadmap.sh/datastructures-and-algorithms
2. https://dev.to/bonaogeto/introduction-to-data-structures-and-algorithms-roadmap-akk
3. https://www.techinterviewhandbook.org/coding-interview-prep/
4. https://www.freecodecamp.org/news/coding-interviews-for-dummies-5e048933b82b/
5. https://www.scholarhat.com/tutorial/datastructures/roadmap-to-learn-dsa
6. https://www.designgurus.io/blog/grokking-the-coding-interview-patterns
7. https://www.reddit.com/r/leetcode/comments/1dzcduj/ive_created_a_free_course_to_help_you_visualize/
8. https://github.com/muhammadfarhankt/Data-Structures-Algoithms-Resource-for-Beginners
9. https://www.scaler.com/blog/dsa-roadmap/
10. https://www.reddit.com/r/learnprogramming/comments/12dappg/roadmap_to_learn_data_structures_and_algorithms/
11. https://www.reddit.com/r/learnprogramming/comments/l13dsp/how_to_effectively_prepare_for_coding_interviews/
12. https://roadmap.sh/pdfs/roadmaps/datastructures-and-algorithms.pdf
13. https://dev.to/arslan_ah/20-essential-coding-patterns-to-ace-your-next-coding-interview-32a3
14. https://dev.to/thecoderishi/roadmap-to-learn-data-structures-and-algorithms-4mce
15. https://www.youtube.com/watch?v=JPMsNSDNows
16. https://dev.to/somadevtoo/coding-interviews-was-hard-until-i-learned-these-patterns-2ji7
17. https://www.youtube.com/watch?v=ijrukuGMjKo
18. https://capd.mit.edu/resources/coding-interview-prep-relax-start-here/
19. https://hackernoon.com/14-patterns-to-ace-any-coding-interview-question-c5bb3357f6ed
20. https://www.linkedin.com/posts/minal-pandey-39763b27_complete-roadmap-to-learn-dsa-in-30-days-activity-7255085113414193152-cG-T

With Python and 1 hour daily, you can make very good progress if you focus on just a few patterns at a time and stay consistent for a few months.codechef+1

Overall 6-week structure (1 hr/day)

- Weeks 1–2: Arrays, strings, and two pointers in Python.geeksforgeeks+1
- Weeks 3–4: Sliding window, hash maps, and basic recursion.codingshanks.substack+1
- Weeks 5–6: Intro to DFS/BFS on trees/grids and simple dynamic programming (like climbing stairs).youtubegeeksforgeeks

What to do each day (1 hour)

- 10–15 min: Review notes/templates from previous problems (no new

content).algocademy+1

- 30–40 min: Learn or revise one small concept/pattern and solve 1 problem on that topic (HackerRank/LeetCode Easy).techinterviewhandbook+1
- 5–10 min: Write down the idea, pattern used, and time complexity in a notebook.bosscoderacademy+1

Pattern order for you (Python beginner)

Follow this order so you don't feel overwhelmed.codechef+1

- Start: Arrays, strings, loops, functions in Python (1–2 weeks if not fully comfortable).
- Then:
  - Two pointers on arrays/strings.
  - Sliding window (longest/shortest subarray or substring questions).
  - Hash map basics (two-sum, frequency counting, anagram check).
  - Recursion basics → subsets/permutations of small lists.
  - DFS/BFS for "number of islands" style problems and tree traversals.

How to practice patterns in Python

- For each pattern, keep a small Python "template" file (for example, a generic sliding window or DFS structure) and reuse it across problems.youtube+1
- Aim for at least 1 fully solved problem per day; on weekends, if you have more time, try 2–3 easy ones to build momentum.datacamp+1

If you want, the next reply can be a concrete Week 1 schedule (Day 1–7) with exact topics and example problem types to search for on LeetCode/HackerRank, all in Python.

1. https://www.codechef.com/roadmap/python-dsa
2. https://www.geeksforgeeks.org/dsa/dsa-roadmap-for-python/
3. https://codingshanks.substack.com/p/complete-dsa-in-python-from-beginner
4. https://www.youtube.com/watch?v=pkYVOmU3MgA
5. https://algocademy.com/blog/how-to-make-a-study-schedule-for-coding-interview-prep-a-comprehensive-guide/
6. https://www.bosscoderacademy.com/blog/complete-dsa-roadmap
7. https://www.techinterviewhandbook.org/coding-interview-study-plan/
8. https://www.datacamp.com/blog/dsa-roadmap
9. https://www.youtube.com/watch?v=wmEAJ6JVCaE
10. https://www.reddit.com/r/learnprogramming/comments/14e52ia/learning_dsa_from_scratch_the_ultimate_guide/
11. https://www.linkedin.com/posts/harshita-verma-57a5b7229_if-i-had-to-start-dsa-again-from-scratch-activity-7360267851594649600-ZwtO
12. https://www.getbridged.co/insights/daily-schedule-for-coding-interview-prep

13. https://www.youtube.com/watch?v=DjYZk8nrXVY
14. https://www.reddit.com/r/cscareerquestions/comments/grvgv9/schedule_for_prep_leetcode_cs_topics/
15. https://www.youtube.com/watch?v=g6TLB_tAaCI
16. https://www.youtube.com/watch?v=5EHv2LOR7Tg&vl=en
17. https://www.reddit.com/r/leetcode/comments/1b44wpo/from_puzzles_to_patterns_sharing_my_leetcode/
18. https://www.designgurus.io/blog/things-to-do-one-day-before-coding-interview
19. https://www.reddit.com/r/leetcode/comments/1d31ksp/leetcode_patternstechniques_cheat_sheet/
20. https://www.scribd.com/document/893095846/Python-DSA-Placement-Roadmap

---

Here is a simple Week 1 plan tailored for Python, 1 hour per day, focusing on arrays and strings (no advanced stuff yet).leetcode+1

Week 1 focus

- Goal: Get comfortable with Python syntax, arrays (lists), and strings, and solve easy problems using loops and conditions.leetcode+1
- Platforms you can use:
  - LeetCode “Array and String” explore card.leetcode
  - HackerRank “Python” and “Warmup” challenges.hackerrank+1

Day 1–2: Python lists and loops

- Learn/revise:
  - Lists, indexing, slicing, `for` loops, `len`, `range`, basic input/output.youtube leetcode
- Practice ideas (search on LeetCode/HackerRank):
  - Sum of array elements (like “Simple Array Sum” on HackerRank).youtube hackerrank
  - Find maximum element in a list.

Day 3–4: Strings and basic operations

- Learn/revise:
  - String indexing/slicing, `in` operator, `lower()`, `upper()`, `strip()`, `split()`, `join()`.leetcode+1
- Practice ideas:
  - Reverse a string.
  - Count vowels in a string.

- Check if a string is a palindrome (ignore case).

Day 5–6: Easy array/string problems

- Try 1 LeetCode easy array problem and 1 easy string problem on these days.leetcode+1
- Example types to search:
  - “Remove duplicates from sorted array” (array).
  - “String Matching in an Array” is a bit harder but good to read/try.leetcodeyoutube

Day 7: Review and mini-test

- Revisit all problems you solved this week, re-code 1–2 from scratch without looking.algomaster+1
- Write a half-page note: what you learned about lists, strings, loops, and common mistakes you made.

If you want, the next step can be a Week 2 plan where you start two pointers in Python, with a simple template and the kind of problems you should search for.

1. https://leetcode.com/explore/learn/card/array-and-string/
2. https://www.hackerrank.com/domains/python
3. https://leetcode.com/discuss/explore/fun-with-arrays/1848144/arrays-101-in-python-all-examples-and-solutions/
4. https://www.hackerrank.com/interview/interview-preparation-kit/warmup/challenges
5. https://www.youtube.com/watch?v=3EgfFdpGR6k
6. https://www.youtube.com/watch?v=d-_OUee4Yvo
7. https://leetcode.com/problem-list/string/
8. https://leetcode.com/problem-list/array/
9. https://leetcode.com/problems/string-matching-in-an-array/
10. https://www.youtube.com/watch?v=7K2BjgjCFDo
11. https://blog.algomaster.io/p/15-leetcode-patterns
12. https://leetcode.com/discuss/post/1688903/solved-all-two-pointers-problems-in-100-z56cn/
13. https://www.youtube.com/watch?v=ZHJDwbfqoa8
14. https://aman.ai/code/two-pointers/
15. https://www.youtube.com/watch?v=QzZ7nmouLTI
16. https://www.reddit.com/r/leetcode/comments/180qheh/strings_and_array_easy_study_suggestion/
17. https://leetcode.com/problem-list/two-pointers/
18. https://github.com/TannerGilbert/HackerRank-Solutions/blob/master/ProblemSolving/Python/Warmup/staircase.py
19. https://www.reddit.com/r/leetcode/comments/18g9383/twopointer_technique_an_indepth_guide_concepts/

20. https://www.youtube.com/playlist?list=PL_8jNcohs27XQfEmWAHCgLFqpsNaWxUSe

---

Here is a simple Week 2 plan for you: learn and practice the **two pointers** pattern in Python with 1 hour per day.geeksforgeeks+1

What is two pointers (for you)?

- Two pointers = use two indices (like `left` and `right`) to walk through a list/string, usually from start & end or both from start.designgurus+1
- It often turns nested loops O(n2)O(n^2)O(n2) into a single loop O(n)O(n)O(n), and is used in many interview problems (pair sum, reverse, move zeros, etc.).algodaily+1

Week 2 daily plan (1 hr/day)

Day 1: Concept + very easy examples

- Learn the basic idea with a sorted array and target sum (concept only, no stress).geeksforgeeks+1
- In Python, implement:
  - Reverse an array/list using two pointers (`left` , `right` , swap, move inward).towardsdatascience+1
  - Check if a string is a palindrome using two pointers.

Day 2: Practice same-end pointers

- Focus on problems where both pointers start at the **ends**:hellointerview+1
  - Reverse list (again, from scratch, no looking).
  - Given a sorted list and a target, check if any pair sums to target (return True/False).

Day 3: Practice same-direction pointers

- Learn variation: pointers moving in same direction (like `slow` and `fast` , but on arrays).algomaster+1
- Practice ideas:
  - Given a sorted list, remove duplicates **in place** and return the new length.
  - Merge two sorted lists into one sorted list (two pointers, one on each list).

Day 4: Apply to LeetCode easy problems

- Search and pick 1–2 “Two Pointers” easy problems on LeetCode.leetcodethehardway+1
- Take your time:

- First, identify positions (where do pointers start?).
- Then, write the loop and pointer movement logic.

Day 5: Mix with strings

- Use two pointers on strings:sharpener+1
  - Check if two strings are “almost equal” (e.g., differ in at most one index).
  - Remove all vowels from a string by building a new string with a read pointer and a write pointer idea.

Day 6: Intuition day

- Read one short article/guide just about two pointers (with diagrams) and connect it to the problems you solved.reddit+1
- Re-implement your favorite 2 problems without looking at old code.

Day 7: Mini test + notes

- Pick 1 new easy two-pointer problem from LeetCode’s “Two Pointers” list and attempt it from scratch.leetcode+1
- Write a one-page note for yourself answering:
  - When do I think of two pointers?
  - What are the 2–3 typical pointer setups (start/end, both from start, fast/slow)?

If you want, the next step can be a compact Python “template” for two pointers (for arrays and strings) plus a similar Week 3 plan for **sliding window**.

1. https://www.geeksforgeeks.org/dsa/two-pointers-technique/
2. https://www.educative.io/courses/grokking-coding-interview/introduction-to-two-pointers
3. https://www.designgurus.io/answers/detail/what-is-two-pointers-coding-pattern
4. https://algodaily.com/lessons/using-the-two-pointer-technique
5. https://www.geeksforgeeks.org/python/python3-program-for-two-pointers-technique/
6. https://towardsdatascience.com/two-pointer-approach-python-code-f3986b602640/
7. https://www.hellointerview.com/learn/code/two-pointers/overview
8. https://algomaster.io/learn/dsa/two-pointers-introduction
9. https://leetcodethehardway.com/tutorials/basic-topics/two-pointers
10. https://leetcode.com/problem-list/two-pointers/
11. https://www.sharpener.tech/blog/two-pointers-technique-in-algorithms/
12. https://www.reddit.com/r/leetcode/comments/18g9383/twopointer_technique_an_indepth_guide_concepts/
13. https://leetcopilot.dev/blog/two-pointers-intuition-for-leetcode-beginners-step-by-step
14. https://vetswhocode.io/blogs/two-pointers-a-practical-technique-for-code-challenges
15. https://www.youtube.com/watch?v=syTs9_w-pwA

16. https://aman.ai/code/two-pointers/
17. https://www.youtube.com/watch?v=pMnGp0bHNCk
18. https://www.youtube.com/watch?v=QzZ7nmouLTI
19. https://bytebytego.com/courses/coding-patterns/two-pointers/introduction-to-two-pointers?fpr=javarevisited
20. https://www.desiqna.in/7302/two-pointers-list-all-good-problems-list-tricks-concepts-2022
21. https://www.youtube.com/watch?v=MQmUVokbTrs

Here is what you asked for: a compact Python **two pointers template**, a **sliding window template**, and a simple **Week 3 plan** around sliding window, all suitable for your 1-hour-per-day schedule.geeksforgeeks+1

Two pointers: when and template

Use two pointers when:

- Data is in an array/string.
- You are dealing with pairs, palindromes, or need to move from both ends or compare neighbors.geeksforgeeks+1

Basic two-pointer template (array, both ends):geeksforgeeks+1

python

```
def two_pointers_example(nums, target): nums.sort() # often needed left, right = 0, len(nums) - 1 while left < right: curr = nums[left] + nums[right] if curr == target: return True # or store pair, etc. elif curr < target: left += 1 else: right -= 1 return False
```

Basic two-pointer template (string palindrome):designgurus+1

python

```
def is_palindrome(s: str) -> bool: left, right = 0, len(s) - 1 while left < right: if s[left] != s[right]: return False left += 1 right -= 1 return True
```

Sliding window: idea and when to use

Use sliding window when the problem talks about:

- Subarrays or substrings (continuous).
- “Maximum/Minimum/Longest/Shortest subarray/substring with condition X”.designgurus+1

Idea: keep a **window** `[left, right]`, move `right` to expand, and move `left` to shrink while maintaining some condition (sum, count, distinct chars, etc.).builtin+1

Sliding window templates in Python

1. Fixed-size window (easier)

Example: maximum sum of any subarray of size `k`.favtutor+1

python

```
def max_subarray_sum_k(nums, k): window_sum = sum(nums[:k]) max_sum = window_sum for right in range(k, len(nums)): window_sum += nums[right] # add new element window_sum -= nums[right - k] # remove old element max_sum = max(max_sum, window_sum) return max_sum
```

2. Variable-size window (more common in interviews)

Example pattern for "longest substring with condition X".algodaily+1

python

```
def longest_substring_example(s: str) -> int: left = 0 best = 0 freq = {} # or set, or counters depending on problem for right in range(len(s)): # include s[right] into window c = s[right] freq[c] = freq.get(c, 0) + 1 # shrink window while condition is violated while some_bad_condition(freq): # define based on problem left_char = s[left] freq[left_char] -= 1 if freq[left_char] == 0: del freq[left_char] left += 1 # update answer using current window [left, right] best = max(best, right - left + 1) return best
```

Week 3 plan (1 hr/day, sliding window)

Focus: fixed-size and simple variable-size sliding window on arrays and strings.blogs.skillcomb+1

- **Day 1:**
  - Learn fixed-size sliding window (like the `max_subarray_sum_k` example).
  - Implement that function in Python and test with a few arrays.
- **Day 2:**
  - Do 1–2 fixed-size problems: "maximum sum subarray of size k", "average of each subarray of size k" (search on LeetCode/GeeksforGeeks).geeksforgeeks+1
- **Day 3:**
  - Learn variable-size sliding window idea (expand with `right`, shrink with `left`).builtin+1

    - Implement a simple function: longest substring with at most `k` distinct characters (even if you follow a tutorial).
- **Day 4–5:**
    - Try 1 easy variable-size problem each day: e.g., “longest substring without repeating characters” style.youtubeelements-of-leetcode.readthedocs
    - Focus on using the same template and only changing the condition.
- **Day 6:**
    - Re-code 1 fixed-size and 1 variable-size solution from memory.
    - Write down in a notebook: “When do I use sliding window vs two pointers?”
- **Day 7:**
    - Pick any new sliding window problem, try for 30 minutes.
    - Then read a solution and map it back to your template (where is expand, where is shrink, where is answer update?).reddityoutube

If you want next, a similar Week 4 plan can be made for **hash maps + basic recursion** in Python, again assuming 1 hour per day.

1. https://www.geeksforgeeks.org/dsa/window-sliding-technique/
2. https://www.geeksforgeeks.org/dsa/two-pointers-technique/
3. https://www.designgurus.io/answers/detail/what-is-two-pointers-coding-pattern
4. https://www.geeksforgeeks.org/python/python3-program-for-two-pointers-technique/
5. https://towardsdatascience.com/two-pointer-approach-python-code-f3986b602640/
6. https://www.designgurus.io/answers/detail/what-is-sliding-window-coding-pattern
7. https://builtin.com/data-science/sliding-window-algorithm
8. https://aman.ai/code/sliding-window/
9. https://favtutor.com/blogs/sliding-window-algorithm
10. https://algodaily.com/lessons/a-birds-eye-view-into-sliding-windows
11. https://blogs.skillcomb.com/sliding-window-technique-using-python/
12. https://www.youtube.com/watch?v=hNMMSEGWoJk
13. https://elements-of-leetcode.readthedocs.io/en/latest/sliding_window/sw.html
14. https://www.reddit.com/r/leetcode/comments/15a7ezt/sliding_window_technique/
15. https://www.youtube.com/watch?v=y2d0VHdvfdc
16. https://dev.to/biswasprasana001/sliding-window-pattern-in-javascript-a-beginner-friendly-guide-2kpo
17. https://justanr.blogspot.com/2014/08/implementing-sliding-windows-in-python.html
18. https://www.youtube.com/watch?v=X4xwDUP_h8w
19. https://www.youtube.com/watch?v=dOonV4byDEg
20. https://www.youtube.com/watch?v=W9Mi4r4lmkI
21. https://www.reddit.com/r/leetcode/comments/yuy5ap/sliding_window_guide/
22. https://hackernoon.com/a-beginners-guide-to-the-sliding-window-algorithm-with-javascript

23. https://www.youtube.com/watch?v=GaXwHThEgGk
24. https://www.codeintuition.io/courses/array/tFob_VIXJQJ8bUN3hlSMB

Here is a compact Python **hash map template**, a **recursion template**, and a **Week 4 plan** (1 hour/day) to learn them both.youtubegeeksforgeeks

Hash map: when and template

Use hash map (`dict` in Python) when:

- You need fast lookups (O(1) average).
- Counting frequencies, checking existence, mapping keys to values, or solving “two sum”/anagram problems.interviewnoodle+1

Basic hash map template (frequency counting):stratascratchyoutube

python

```
def freq_count_example(s: str) -> dict: freq = {} # or collections.Counter(s) for char in s: if char in freq: freq[char] += 1 else: freq[char] = 1 return freq # Or shorter: def freq_count_short(s: str) -> dict: return {char: s.count(char) for char in set(s)}
```

Two-sum style template:youtubeinterviewnoodle

python

```
def two_sum(nums: list, target: int) -> list[int]: seen = {} # key: number, value: index for i, num in enumerate(nums): complement = target - num if complement in seen: return [seen[complement], i] seen[num] = i return []
```

Recursion: idea and when

Use recursion when:

- The problem can be broken into a smaller version of itself (trees, subsets, permutations).
- Base case + recursive case. Always have a base case to avoid infinite loops.geeksforgeeks+1

Basic recursion template:w3schools+1

python

```
def recursive_example(n: int) -> int: # BASE CASE (stops recursion) if n <= 1: return 1 # or n, or 0, etc. # RECURSIVE CASE (calls itself with smaller input) return n * recursive_example(n - 1) # factorial example
```

List/tree traversal example:codecademy+1

python

```
def sum_list(nums: list) -> int: if not nums: # base case: empty list return 0
return nums[0] + sum_list(nums[1:]) # head + recursion on tail
```

Week 4 plan (1 hr/day)

Focus: Hash maps first (Days 1–4), then simple recursion (Days 5–7).geeksforgeeks+1

Days 1–2: Hash map basics

- Learn `dict` operations: `d[key] = val`, `if key in d`, `d.get(key, default)`.techinterviewhandbook+1
- Practice:
  - Count character frequencies in a string.
  - Check if two strings are anagrams (same freq counts).

Days 3–4: Hash map in problems

- Solve “two sum” (LeetCode #1, very common).leetcodeyoutube
- Try 1 more: group anagrams or find duplicate elements (search LeetCode “Hash Table” easy).geeksforgeeks+1

Days 5–6: Recursion basics

- Implement factorial, Fibonacci (simple version), and sum of list using recursion.programiz+1
- Understand call stack visually (draw 3 calls for factorial(4)).geeksforgeeks+1

Day 7: Mix + review

- Try a recursive problem: reverse a string using recursion (no loops).
- Notebook: “Hash map vs list?”, “Base case vs recursive case?”.
- Re-code one hash map and one recursion problem from memory.

If you want next, a Week 5 plan can be made for **DFS/BFS** on grids and trees in Python (still 1 hour/day).

1. https://www.youtube.com/watch?v=RcZsTI5h0kg
2. https://www.geeksforgeeks.org/python/recursion-in-python/
3. https://interviewnoodle.com/a-gentle-introduction-to-using-hashmaps-in-coding-interviews-8cbc422c9913

4. https://www.techinterviewhandbook.org/algorithms/hash-table/
5. https://www.stratascratch.com/blog/easy-guide-to-python-hashmaps/
6. https://www.youtube.com/watch?v=KLIXCFG5TnA
7. https://interviewnoodle.com/path-to-conquer-leetcode-easy-part-1-hashmap-and-two-sums-baba53e36496
8. https://www.w3schools.com/python/python_recursion.asp
9. https://www.programiz.com/python-programming/recursion
10. https://www.codecademy.com/learn/learn-recursion-python/modules/recursion-python/cheatsheet
11. https://www.geeksforgeeks.org/dsa/top-50-problems-on-hash-data-structure-asked-in-sde-interviews/
12. https://leetcode.com/problem-list/hash-table/
13. https://www.interviewbit.com/hashmap-interview-questions/
14. https://www.youtube.com/watch?v=bGw2-Pdg_78
15. https://www.reddit.com/r/leetcode/comments/1ea7akj/some_arrayhash_table_questions/
16. https://www.intervue.io/top-coding-questions/python/design-hashmap
17. https://leetcode.com/problems/design-hashmap/
18. https://www.youtube.com/watch?v=JlbhUhTec-s
19. https://www.youtube.com/watch?v=72uVUU1boKo
20. https://www.youtube.com/watch?v=CPrY1LdDppQ

Here is a compact Python **DFS and BFS template**, and a **Week 5 plan** (1 hour/day) to learn graph/tree traversals on simple grids and trees.deercode.github+1

DFS vs BFS: quick idea

- **DFS** (Depth First): Go deep first (like recursion or stack), good for paths, connected components.datacamp+1
- **BFS** (Breadth First): Level by level (queue), good for shortest path in unweighted graphs, level order.geeksforgeeks+1
  Use both for grids (like islands) and trees.algomaster+1

DFS template (recursive, simplest for beginner)

python

```
def dfs(grid, i, j, visited): rows, cols = len(grid), len(grid[0]) # BASE CASE: out of bounds, visited, or invalid cell if (i < 0 or i >= rows or j < 0 or j >= cols or (i, j) in visited or grid[i][j] == 0): # 0 = water/island boundary return 0 # or False, etc. visited.add((i, j)) # RECURSIVE CASE: explore 4
```

```
directions size = 1 # count this cell size += dfs(grid, i+1, j, visited) # down size += dfs(grid, i-1, j, visited) # up size += dfs(grid, i, j+1, visited) # right size += dfs(grid, i, j-1, visited) # left return size
```

Usage: `visited = set(); dfs(grid, start_row, start_col, visited)` github+1

BFS template (iterative with queue)

python

```
from collections import deque def bfs(grid, start_i, start_j): rows, cols = len(grid), len(grid[0]) queue = deque([(start_i, start_j)]) # (row, col) visited = set([(start_i, start_j)]) while queue: i, j = queue.popleft() # dequeue # PROCESS current cell (e.g., add to path) # EXPLORE neighbors (4 directions) for di, dj in [(0,1), (1,0), (0,-1), (-1,0)]: ni, nj = i + di, j + dj if (0 <= ni < rows and 0 <= nj < cols and (ni, nj) not in visited and grid[ni][nj] == 1): queue.append((ni, nj)) visited.add((ni, nj)) return len(visited) # or shortest path, etc.
```

Week 5 plan (1 hr/day)

Focus: DFS and BFS on grids (like “number of islands”) and simple trees.leetcode+1

Days 1–2: DFS on grids

- Learn DFS idea: mark visited, recurse on 4 neighbors.leetcode+1
- Practice:
  - Implement DFS to find size of one island in a 2D grid (1=land, 0=water).
  - Use the template above.

Days 3–4: BFS on grids

- Learn BFS: queue for level-by-level, `deque.popleft()`.datacamp+1
- Practice:
  - BFS to find shortest path from start to end in a grid (no obstacles first).
  - “Number of islands” using BFS (mark visited as you queue).

Days 5–6: Trees (simpler)

- Tree as nested dict or class: `node.children = [child1, child2]`.github
- Practice:
  - Level order traversal of tree (BFS).
  - Inorder/preorder traversal (DFS recursive).

Day 7: Review + mix

- Pick one LeetCode easy “Breadth-First Search” or “DFS” problem (e.g., islands, rotting oranges lite).leetcode
- Notebook: “DFS vs BFS? When shortest path?”
- Re-code one DFS and one BFS from memory.

If you want next, a Week 6 plan can be made for **binary search + simple DP** in Python (1 hour/day).

1. https://deercode.github.io/en/dfs-by-python
2. https://github.com/BruceWeng/Leetcode-Python/blob/master/BFS%20and%20DFS%20template.py
3. https://www.datacamp.com/tutorial/depth-first-search-in-python
4. https://leetcode.com/discuss/study-guide/1072548/A-Beginners-guid-to-BFS-and-DFS/
5. https://www.geeksforgeeks.org/python/python-program-for-breadth-first-search-or-bfs-for-a-graph/
6. https://www.datacamp.com/tutorial/breadth-first-search-in-python
7. https://blog.algomaster.io/p/15-leetcode-patterns
8. https://leetcode.com/discuss/post/5039797/bfs-and-dfs-graph-problems-easy-to-mediu-8mz5/
9. https://leetcode.com/problem-list/breadth-first-search/
10. https://www.freecodecamp.org/news/graph-algorithms-in-python-bfs-dfs-and-beyond/
11. https://blog.algorithmexamples.com/graph-algorithm/how-to-guide-python-graph-traversal-algorithms-implementation/
12. https://thita.ai/blog/dsa/graph-traversal-patterns-dfs-bfs
13. https://www.youtube.com/watch?v=cS-198wtfj0
14. https://www.w3schools.com/dsa/dsa_algo_graphs_traversal.php
15. https://gist.github.com/wanderingstan/4b8ae84dd77eadf716b012f2a5e74790
16. https://www.reddit.com/r/leetcode/comments/viwzya/any_resources_or_suggestions_to_for_solving_bfs/
17. https://www.tutorialspoint.com/python_data_structure/python_graph_algorithms.htm
18. https://labuladong.online/algo/en/essential-technique/bfs-framework/
19. https://www.youtube.com/watch?v=VTH21EqzECk
20. https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

Here is a compact Python **binary search template**, **simple DP template**, and a **Week 6 plan** (1 hour/day) to wrap up your beginner patterns.w3schools+1

Binary search: when and template

Use binary search when:

- Data is **sorted** (array or can be sorted).
- You need to find an element, first/last occurrence, or position where a condition becomes true.geeksforgeeks+1

Basic template (find element):realpython+1

python

```
def binary_search(arr: list, target: int) -> int: 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 return -1 # not found
```

Find first true condition (more advanced but common):leetcode+1

python

```
def first_true(arr: list) -> int: left, right = 0, len(arr) - 1 while left < right: mid = (left + right) // 2 if condition(arr[mid]): # your condition here right = mid else: left = mid + 1 return left if condition(arr[left]) else -1
```

Simple DP: idea and template

Use simple DP when:

- Optimal substructure (answer depends on smaller subproblems).
- Overlapping subproblems (reuse answers). Start with classics like Fibonacci.w3schools+1

1D DP template (like Fibonacci/climbing stairs):youtubew3schools

python

```
def climbing_stairs(n: int) -> int: if n <= 2: return n prev2, prev1 = 1, 2 # dp[i-2], dp[i-1] for i in range(3, n + 1): curr = prev1 + prev2 # dp[i] = dp[i-1] + dp[i-2] prev2, prev1 = prev1, curr return prev1
```

Full array DP (subset sum lite):favtutor

python

```
def can_sum(target: int, nums: list) -> bool: dp = [False] * (target + 1) dp[0] = True # base case for num in nums: for t in range(num, target + 1): dp[t] = dp[t] or dp[t - num] return dp[target]
```

Week 6 plan (1 hr/day)

Focus: Binary search (Days 1–3), simple DP (Days 4–7).leetcode+1

Days 1–2: Binary search basics

- Learn the template, test on sorted array search.youtubew3schools
- Practice:
  - LeetCode #704 “Binary Search” (exact match).
  - Find first position of a number in sorted array (duplicates).

Day 3: Binary search variations

- Try “search in rotated sorted array” lite or find peak element.leetcode
- Notebook: “left <= right vs left < right?”

Days 4–5: DP basics (Fibonacci style)

- Implement Fibonacci and “climbing stairs” (1 or 2 steps).w3schoolsyoutube
- Understand space optimization (2 variables instead of array).

Days 6–7: Simple DP problems

- “House robber” lite or “min cost climbing stairs”.leetcodeyoutube
- Try 1D subset sum or coin change (small inputs).
- Review: Re-code one binary search and one DP from memory.

Congrats on completing beginner patterns!

You now know the top 8 patterns (arrays/strings, two pointers, sliding window, hash maps, recursion, DFS/BFS, binary search, simple DP) that cover 80%+ of easy/medium interview problems.algomaster

Next could be a **mock interview schedule** (how to practice full problems under time) or **Week 7–8: harder variations**. Let me know!

1. https://www.w3schools.com/python/python_dsa_binarysearch.asp
2. https://favtutor.com/blogs/dynamic-programming
3. https://www.geeksforgeeks.org/python/python-program-for-binary-search/
4. https://realpython.com/binary-search-python/
5. https://leetcode.com/discuss/post/786126/Python-Powerful-Ultimate-Binary-Search-Template.-Solved-many-problems/
6. https://dev.to/alexhanbich/binary-search-template-in-python-44jo
7. https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php
8. https://www.educative.io/blog/python-dynamic-programming-tutorial

9. https://www.youtube.com/watch?v=_i4Yxeh5ceQ
10. https://leetcode.com/problems/binary-search/
11. https://leetcode.com/problem-list/dynamic-programming/
12. https://www.youtube.com/watch?v=s4DPM8ct1pI
13. https://leetcode.com/problem-list/binary-search/
14. https://blog.algomaster.io/p/15-leetcode-patterns
15. https://www.reddit.com/r/leetcode/comments/1csos9p/i_still_struggle_with_binary_search_after_500/
16. https://towardsdatascience.com/powerful-ultimate-binary-search-template-and-many-leetcode-problems-1f850ef95651/
17. https://high-python-ext-3-algorithms.readthedocs.io/ko/latest/chapter8.html
18. https://stackoverflow.com/questions/60681085/binary-search-advanced-template-explain
19. https://www.datacamp.com/tutorial/binary-search-python
20. https://discuss.python.org/t/basic-binary-search-algorithm-wrangling-with-while-loops-and-their-conditions-x2/43104

Great! Now that you have the core patterns, here is a **mock interview practice schedule** (1 hour/day) for Weeks 7–8 to simulate real coding interviews.techinterviewhandbook+1

Mock interview structure (45–60 min per session)

Every mock follows this flow (practice out loud every time):anthonydmays+1

1. **Clarify** (2–5 min): Repeat problem, ask about input/output/edge cases/constraints.
2. **Examples** (5 min): 2–3 test cases, including edge cases.
3. **Pattern match** (5–10 min): “This looks like sliding window + hash map”.
4. **Code** (15–25 min): Write clean Python code, test mentally.
5. **Complexity + optimize** (5 min): Time/space big-O, any improvements?

Week 7–8 daily plan (1 hr/day)

Goal: 3–4 mocks per week + review. Alternate easy/medium problems from LeetCode Top 150 or HackerRank Interview Kit.leetcode+1

Days 1, 3, 5: Full mock interview

- Pick 1 LeetCode medium from a random pattern you know (use Top Interview 150).leetcode
- Time yourself: 45 min total, speak out loud (“I think two pointers here because...”).
- After: Note mistakes (syntax? missed edge case? wrong pattern?).techinterviewhandbook+1

Days 2, 4, 6: Review + weak spots

- Review 1–2 problems from previous mocks (re-code from scratch).
- Fix weak pattern: e.g., if BFS rusty, do 1 easy grid problem.

Day 7: Mock debrief + plan

- Do 1 full mock, record yourself (phone video) if possible.
- Self-score: Did you clarify? Communicate? Code cleanly?
- Update notebook: top 3 mistakes, patterns still weak.

Platforms for solo mocks (free/cheap)

Platform	Why good for you	Time per mock
LeetCode Mock Assessment leetcode	Timed, real questions, instant feedback	45 min
HackerRank Interview Kit hackerrank	Beginner-friendly, Python support	30–60 min
Pramp pramp	Free peer mocks (sign up, 1 hr sessions)	60 min
interviewing.io interviewing	Anonymous, FAANG-style (paid after free trials)	60 min

Extra tips for 1 hr/day success

- **Out loud always:** Even solo, explain like to an interviewer ("Now I'll use left and right pointers...").anthonydmays
- **No IDE:** Use LeetCode editor or notepad to simulate whiteboard.anthonydmays
- **Track progress:** After 2 weeks, you should solve easy/medium in 30–40 min with clean code.techinterviewhandbook
- **If stuck >5 min:** Hint, but note why you got stuck.

After Weeks 7–8, you'll be ready for real interviews or harder patterns. If you want a **company-specific plan** (e.g., Google, Amazon) or **system design intro**, just say techinterviewhandbook

1. https://www.techinterviewhandbook.org/coding-interview-study-plan/
2. https://anthonydmays.com/blog/2022/05/09/how-to-practice-leetcode-problems-the-right-way/
3. https://www.reddit.com/r/codinginterview/comments/1o6kqbe/preparing_for_a_coding_interview_for_one_hour/
4. https://leetcode.com/assessment/

5. https://www.hackerrank.com/interview/interview-preparation-kit
6. https://leetcode.com/studyplan/top-interview-150/
7. https://www.pramp.com
8. https://interviewing.io
9. https://www.algoexpert.io/mock-interviews
10. https://www.tryexponent.com/practice
11. https://www.getbridged.co/insights/daily-schedule-for-coding-interview-prep
12. https://www.reddit.com/r/leetcode/comments/1n6htih/daily_1hour_codinginterview_practice_group/
13. https://igotanoffer.com/en/mock-interviews/leetcode
14. https://www.freecodecamp.org/news/coding-interviews-for-dummies-5e048933b82b/
15. https://www.reddit.com/r/leetcode/comments/wvdpwq/can_anyone_share_your_leetcode_grindinterview/
16. https://www.youtube.com/watch?v=FNL61tvwBUg
17. https://www.interviewbit.com/practice/
18. https://www.designgurus.io/answers/detail/what-to-do-1-hour-before-a-coding-interview
19. https://www.senseicopilot.com/blog/how-to-prepare-for-coding-interviews
20. https://www.reddit.com/r/learnprogramming/comments/l13dsp/how_to_effectively_prepare_for_coding_interviews/
