CoGrammar

Welcome to this session:

Coding Interview Workshop - Algorithmic Paradigms

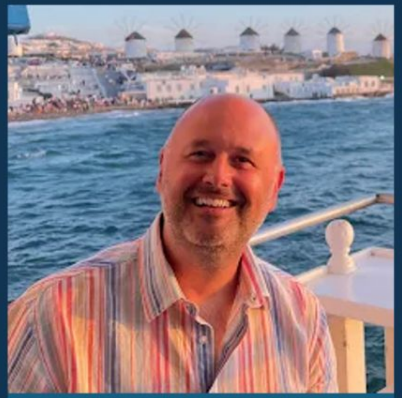
The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Coding Interview Workshop

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Skills Bootcamp Coding Interview Workshop

- For all **non-academic questions**, please submit a query: ***www.hyperiondev.com/support***
- **Report a safeguarding incident: *www.hyperiondev.com/safeguardreporting***
- We would love your feedback on lectures: **Feedback on Lectures**
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- **Differentiate between greedy algorithms**, **divide and conquer**, and **dynamic programming**, and recognize their ideal use cases.
- **Implement classic problems** such as **Knapsack**, **Fibonacci sequence**, and **QuickSort** using the appropriate paradigm.
- **Compare the trade-offs between different paradigms** in terms of time and space complexity.
- **Use algorithmic paradigms** to help solve some common interview questions.

Which algorithmic paradigm do you feel most comfortable with?

A. Greedy algorithms
B. Divide and conquer
C. Dynamic programming
D. None of the above

CoGrammar

When faced with a complex coding problem, do you typically:

A. Try to solve it step by step using an intuitive approach
B. Break it into smaller subproblems and solve each one
C. Store previously computed results to avoid redundant calculations
D. Guess and check until something works

CoGrammar

Lecture Overview

- ➔ Key Techniques
- ➔ Steps to Success
- ➔ Practice the Plan

Shortest Path Algorithms

Imagine you are **planning trip from City A to City B**. There are many different ways to get from A to B but you are trying to find the **shortest one**. You can **go through other cities** as well, the path does not have to be direct. You are given a list of all the neighbouring cities and paths connecting them.

- Would you choose the shortest path at each point and hope that the total path is the shortest as well?
- Would you split the journey into parts and find the shortest path for each part?
- Would you consider each path and dynamically update the shortest path between any two cities?

Shortest Path Algorithm

Algorithms are at the core of computer science, and **different algorithmic paradigms** provide **different approaches** to solving problems efficiently. Whether optimizing network routing, compressing files, or planning resource allocation, understanding these paradigms allows developers to make informed decisions when designing solutions.

Algorithmic Paradigms

General strategies or frameworks for designing algorithms

- As we work through more problems, you'll realise that the way we approach certain problems are similar.
- Algorithmic Paradigms are the **different approaches** we use to solve **certain types** of problems.
- This involves providing **a blueprint for solving a class of problems**, rather than a specific algorithm itself.
- There are many algorithmic paradigms, we'll be looking at 3.

Algorithmic Paradigms

- **Greedy Algorithm**
  - Make locally optimal choices at each step.
  - Works well when local optimization leads to a global optimum.
  - Examples: Activity Selection, Huffman Encoding, Dijkstra's Algorithm.

Algorithmic Paradigms

- **Divide-and-Conquer**
  - Breaks a problem into smaller subproblems, solves them independently, and merges results.
  - Efficient for problems that can be split into independent parts.
  - Examples: Merge Sort, QuickSort, Binary Search.

Algorithmic Paradigms

- **Dynamic Programming (DP)**
  - Solves problems by breaking them into overlapping subproblems and storing results to avoid redundant computations.
  - Useful when a problem has optimal substructure and overlapping subproblems.
  - Examples: Knapsack Problem, Fibonacci Sequence, Longest Common Subsequence.

Algorithmic Paradigms

Paradigm	Time Complexity	Space Complexity	Best Use Cases
Greedy	Usually O(n) or O(n log n)	Low (O(1) or O(n))	Problems with greedy-choice property
Divide & Conquer	Varies	Medium	Problems that can be divided into independent subproblems
Dynamic Programming	Typically $O(n^2)$ or $O(n^3)$	High (O(n) to $O(n^2)$)	Problems with overlapping subproblems

Practice the Paradigm

Let's practice the algorithmic paradigms by solving some classic problems for each paradigm.

Then we'll do the following problems together:

- Majority Elements
- Container With Most Water
- Longest Palindromic Substring

Which paradigm do you feel most confident with after this lecture?

A. Greedy algorithms
B. Divide and conquer
C. Dynamic programming
D. Still unsure

CoGrammar

What was the most valuable part of today's lecture?

A. Understanding the trade-offs between paradigms
B. Seeing real-world applications of algorithms
C. Implementing classic problems in Python
D. Practicing with an interview-style question

CoGrammar

Homework

Practise the skills we've developed by completing the rest of the LeetCode questions:

- Practise speaking through your solutions and explaining how you approached each problem.
- In the next lecture we'll be covering the topic: "Linear Data Structures"
- You can have a look at the following LeetCode questions to prepare:
  - Example 1
  - Example 2
  - Example 3

Summary

- ★ **Greedy algorithms:** best for problems where local decisions lead to a global optimum.
- ★ **Divide and conquer:** breaks a problem into independent subproblems.
- ★ **Dynamic programming:** useful when subproblems overlap and results can be stored.
- ★ Recognizing which paradigm to apply is critical in interviews and real-world.

Paradigm	Time Complexity	Space Complexity	Best Use Cases
Greedy	Usually O(n) or O(n log n)	Low (O(1) or O(n))	Problems with greedy-choice property
Divide & Conquer	Varies	Medium	Problems that can be divided into independent subproblems
Dynamic Programming	Typically $O(n^2)$ or $O(n^3)$	High (O(n) to $O(n^2)$)	Problems with overlapping subproblems

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education