CoGrammar

Welcome to this session:

Coding Interview Workshop - Linear Data Structures

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

- **Explain the fundamental characteristics** of arrays, linked lists, stacks, queues, and hash tables.
- **Implement insertion, deletion, searching, and traversal operations** for each structure.
- **Solve interview problems** that require efficient use of stacks, queues, and hash tables, such as valid parentheses, next greater element, queue reconstruction, and anagram grouping.
- **Analyse the time and space complexity** of different data structure operations.

You are building a text editor with an undo feature. Which data structure would be the best choice?

A. Array

B. Stack

C. Queue

D. Linked List

CoGrammar

Which data structure would be best for fast lookups in a dictionary application?

A. Array
B. Linked List
C. Hash Table
D. Queue

CoGrammar

Lecture Overview

→ Linked Lists
→ Stacks and Queues
→ Hash Tables

CoGrammar

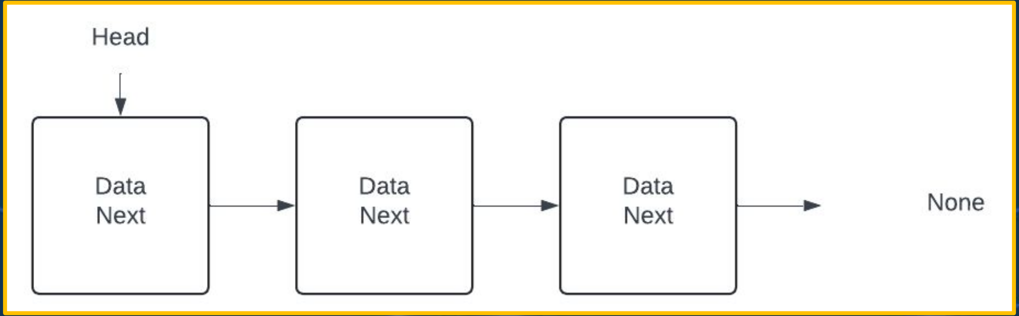
Linked Lists

A linear data structure consisting of a sequence of nodes, each node contains data and a reference (link) to the next node in the sequence

- Components of a node:
  - **Data:** The actual information stored in the node
  - **Next pointer:** A reference to the next node in the linked list
  - **Previous pointer (for doubly linked lists):** A reference to the previous node in the linked list
- Nodes are **connected to form a linked list**, allowing for efficient insertion and deletion of elements at any position in the sequence

Singly Linked Lists

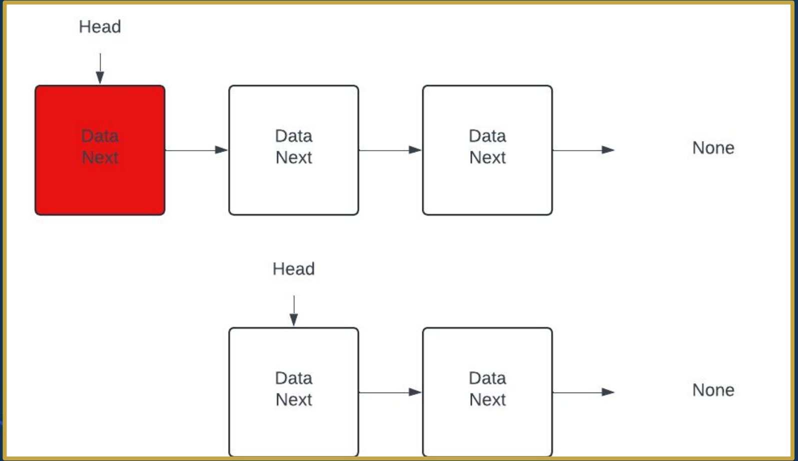
- Pros:
  - **Dynamic size:** Linked lists can grow or shrink as needed during runtime
  - **Efficient insertion and deletion:** O(1) time complexity for inserting or deleting elements at the beginning of the list

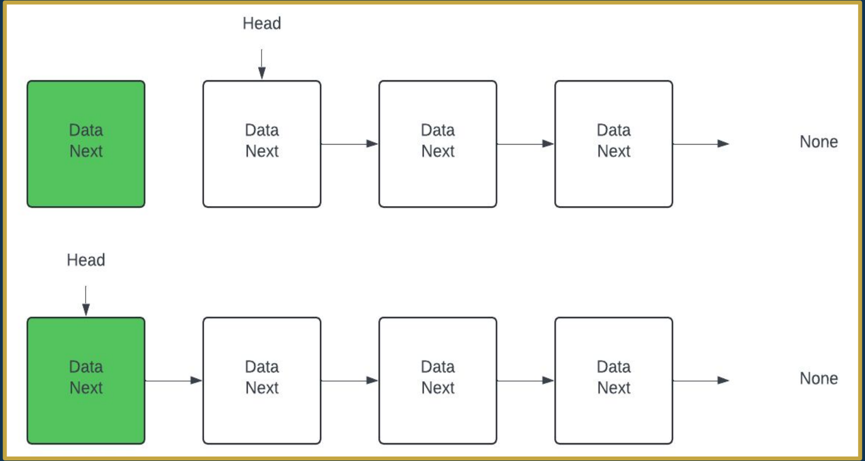


CoGrammar

Singly Linked Lists

❖ Why O(1)? Glad you asked, so 🤓...





❖ For removal we only take **1 step no matter the size of the list**, and the same for insertion, thus O(1) for worst case

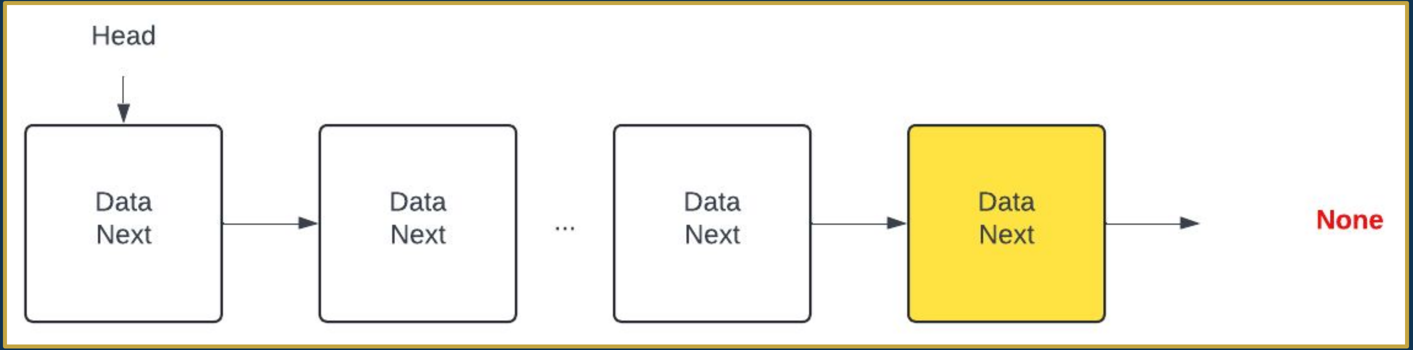
CoGrammar

Singly Linked Lists

- Cons:
  - **No random access:** Accessing an element at a specific index requires traversing the list from the beginning, resulting in O(n) time complexity
  - **Extra memory:** Linked lists require additional memory for storing the next pointers

Singly Linked Lists

❖ Why O(n)? You flatter me ☺️...

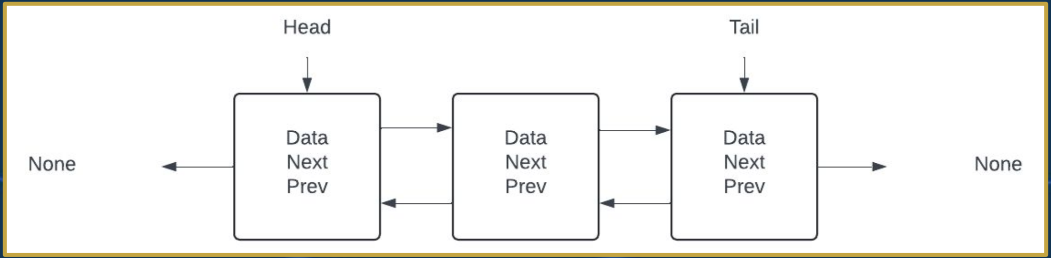


❖ This list has n nodes, to get to the yellow (last) one we need to pass n nodes, so in worst case O(n)

CoGrammar

Doubly Linked Lists

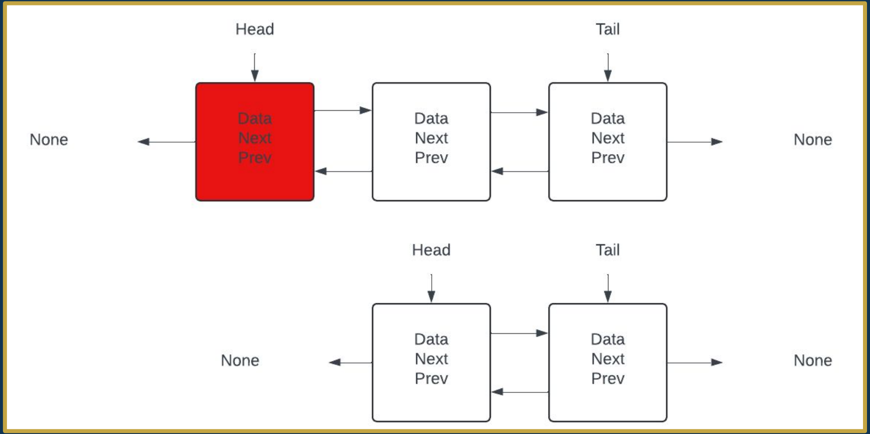
- Pros:
  - **Efficient insertion and deletion:** O(1) time complexity for inserting or deleting elements at both ends of the list
  - **Backward traversal:** Doubly linked lists allow traversal in both forward and backward directions

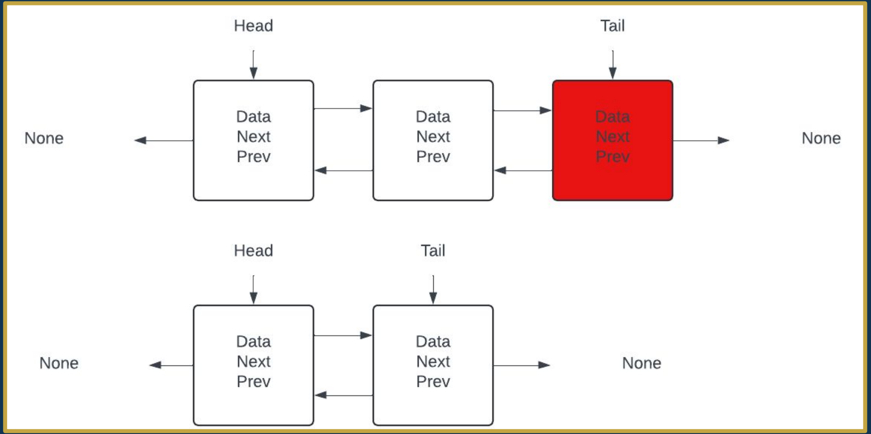


CoGrammar

Doubly Linked Lists

❖ Why O(1)?





❖ Thanks to the tail pointer deletion or addition on either sides **take 1 step at worst**, thus O(1)

CoGrammar

Doubly Linked Lists

- Cons:
  - **Extra memory:** Doubly linked lists require more memory than singly linked lists due to the additional previous pointers
  - **More complex implementation:** Maintaining the previous pointers requires more careful management of node connections

Stacks and Queues

Types of linear data structures that allow for storage and retrieval of data based on a specific method of ordering.

- ❖ Data structures allow us to **organise storage** so that data can be accessed **faster and more efficiently**.
- ❖ Stacks and queues are simple, easy to implement and widely applicable.
- ❖ Each has defined methods of **ordering** and **operations** for adding and removing elements to and from the structure.
- ❖ Can be implemented using an Array, Linked List or the `deque` or `queue` modules in Python.

CoGrammar

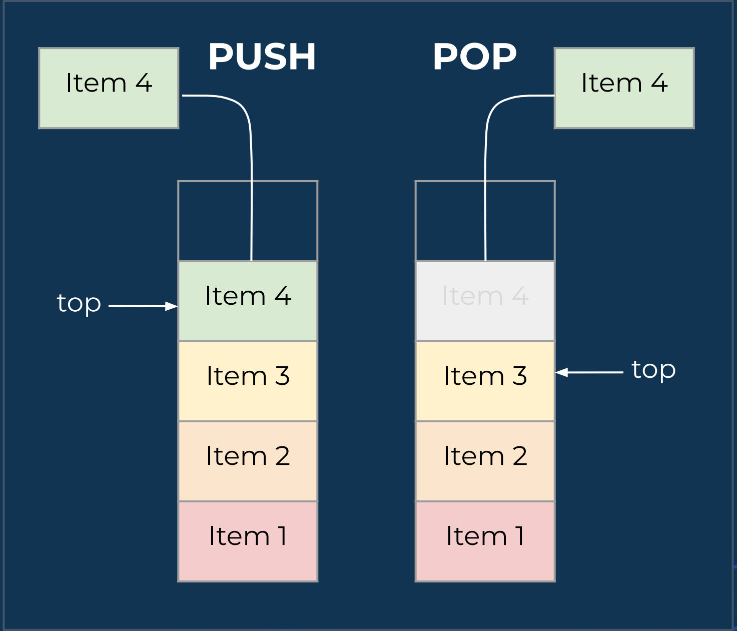
Stacks

Method of Ordering

- ❖ **LIFO:** Last element added to the stack is the first to be removed
- ❖ Elements are added on **top** of one another in a "stack"
- ❖ A pointer points to the element at the **top** of the stack

Operations

- ❖ **Push:** Adds an element to the top of the stack
- ❖ **Pop:** Removes the element from the top of the stack



CoGrammar

Stacks

Complexity Analysis

- **Push**
  - **Space: O(1)**
    No extra space is used
  - **Time: O(1)**
    A single memory allocation done in constant time

- **Pop**
  - **Space: O(1)**
    No extra space is used
  - **Time: O(1)**
    Pointer is decremented by 1

Common Uses

- Function and Recursive Calls
- Undo and Redo Mechanisms
- “Most recently used” features
- Backtracking algorithms
- Expression evaluations and syntax parsing

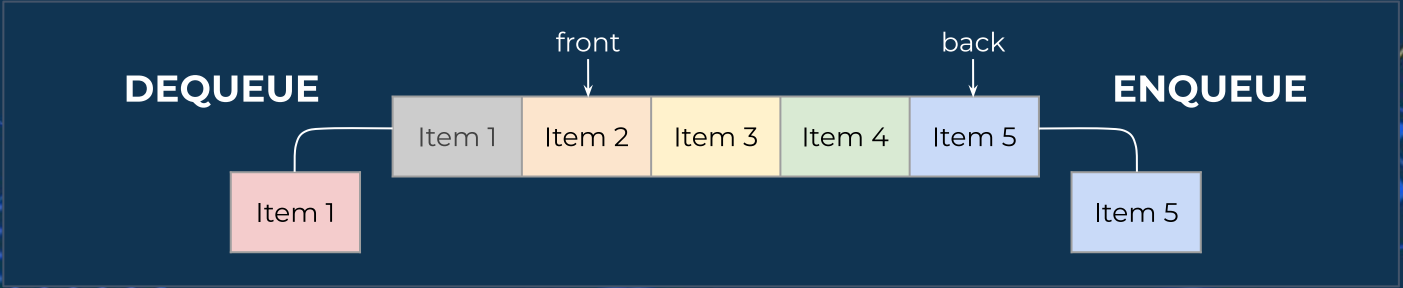
Queues

Method of Ordering

- **FIFO:** First element added to the stack is the first to be removed
- Elements added to **rear** of a “queue” and removed from **front**
- Two pointers point to the **front** and the **rear**

Operations

- **Enqueue:** Adds an element to the end of the queue
- **Dequeue:** Removes the element from the front of the queue



Queues

Complexity Analysis

- **Enqueue**
  - **Space: O(1)**
    No extra space used
  - **Time: O(1)**
    Single memory allocation done in constant time
- **Dequeue**
  - **Space: O(1)**
    No extra space used
  - **Time: O(1)**
    Front pointer incremented by 1 and node deallocated

Common Uses

- Task Scheduling
- Resource Allocation
- Network Protocols a
- Printing Queues
- Web Servers
- Breadth-First Search

CoGrammar

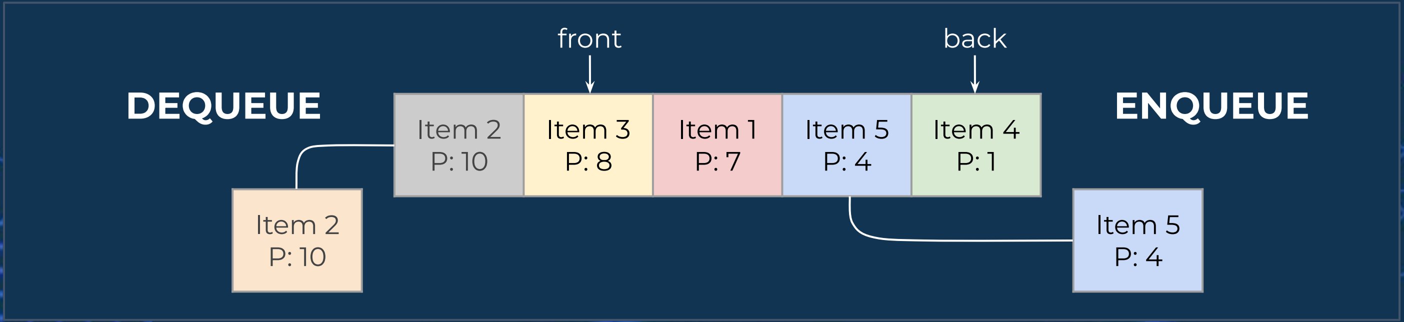
Priority Queues

Method of Ordering

- Arranged according to the assigned **priority** of elements
- Order direction doesn't matter, as long as **the highest priority elements are removed first**

Operations

- **Enqueue:** Adds an element to the queue based on its priority
- **Dequeue:** Removes the highest priority element from the queue



CoGrammar

Priority Queues

Complexity Analysis

- **Enqueue**
  - **Space:**
    - **List - O(1)**
      No extra space is used
    - **PriorityQueue - O(1)**
      No extra space is used
  - **Time:**
    - **List - O(n)**
      Each element's priority must be checked and compared
    - **PriorityQueue - O(log n)**
      Adding node to heap

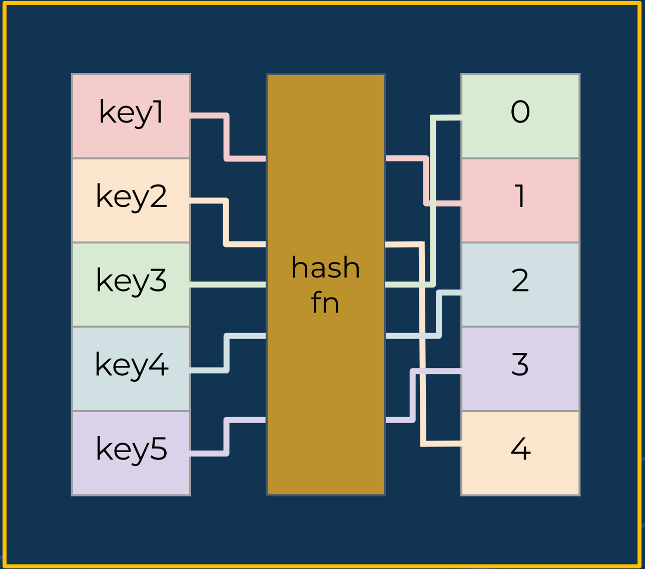
Dequeue

- **Space:**
  - **List - O(1)**
    No extra space is used
  - **PriorityQueue - O(1)**
    No extra space is used
- **Time:**
  - **List - O(1)**
    Pointer is updated or last element is removed
  - **PriorityQueue - O(log n)**
    Removing node from heap

Hash Tables

A data structure that is used to store key-value pairs which uses a hash function to transform the key into the index of an associated array element where the data will be stored.

- Hash tables allow for **efficient and fast** insertions, deletions and look-ups due to the manner in which values are stored.
- Since the **index where the values is stored is a function of the value**, the position where the value must be stored is always known.



Click here to visualize this data structure!

CoGrammar

- ❖ **Hashing** refers to using the hash function to calculate **the hash** which is the index of the array element where the key-value pair will be stored.
- ❖ Each array element is called a **bucket or slot** and can store one or more key-value pairs.
- ❖ The **hash function** can be any **deterministic** function which maps a key to the range of indices but the aim is that the function is **efficient**, it distributes keys **uniformly** and **multiple keys being mapped to the same index** is avoided.
- ❖ The **load factor** is the ratio of the number of stored elements to the total number of buckets and is used to determine whether the table can be downsized to improve performance.

CoGrammar

Hash Collisions

When two or more distinct keys hash to the same array index.

- ❖ Although we try to avoid hash collisions, they are inevitable so we have to implement a **collision-resolution** process.
- ❖ When searching or storing data, a hash collision **increases the time complexity** of the task.

Operation	Average Case	Worst Case
Insert	O(1)	O(n)
Lookup/Search	O(1)	O(n)
Delete	O(1)	O(n)

Collision-Resolution

- **Chaining**: Linked Lists are used for each bucket, so multiple key-value pairs can be stored in the same bucket.
  - Pros: Simple to implement, dynamic resizing
  - Cons: Memory overhead implications, space inefficiencies
- **Linear Probing**: If there is a collision, place the pair in the next available slot in the hash table (linearly).
  - Pros: Simple to implement, memory-efficient
  - Cons: Primary clustering (large blocks of occupied elements), clustering results in more time inefficiencies

Built-in Hash Tables

- **Dictionaries** (Python) and **Maps** (JavaScript) are data structures which store key-value pairs.
- They are implemented internally using a **hash table**, using built-in hash functions.
- They are the easiest way to implement a hash table, since all the complexities involved in ensuring efficiency is abstracted away.
- A custom hash table would be primarily used when a **custom hash function** needs to be implemented.
- This is useful in the case where keys are of **custom or non-hashable data types.**

CoGrammar

Practice the Structures

Let’s practice using different linear data structures by solving some classic problems for each paradigm.

Then we’ll do the following problems together:

- ➢ LRU Cache
- ➢ First Unique Character
- ➢ Group Anagrams

CoGrammar

You’re implementing an autocomplete feature that must store and retrieve words efficiently. Which data structure would you use?

A. Stack

B. Queue

C. Hash Table

D. Linked List

CoGrammar

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

A. Stack

B. Queue

C. Linked List

D. Hash Table

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: “Non-Linear Data Structures”
- ❖ You can have a look at the following LeetCode questions to prepare:
  - ➢ Example 1
  - ➢ Example 2
  - ➢ Example 3

Summary

★ **Arrays** allow fast indexing but slow insertions.

★ **Linked Lists** enable dynamic memory allocation but have slow searches.

★ **Stacks** handle undo/redo operations and recursion.

★ **Queues** are best for processing tasks in order.

★ **Hash Tables** provide fast lookups for key-value pairs.

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