Welcome to this CoGrammar session:

Searching and Sorting

The session will start shortly...

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

Software Engineering Session Housekeeping

- 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**

Software Engineering Session Housekeeping cont.

- 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**

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

- ***Timeframe:*** First 2 Weeks
- ***Guided Learning Hours (GLH):*** Minimum of 15 hours
- ***Task Completion:*** First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

- ***Guided Learning Hours (GLH):*** **60**
- ***Task Completion:*** 13 tasks

Due Date: 28 April 2024

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar

Searching and Sorting

April 2024

Learning Objectives

- Define what an algorithm is.
- Define complexity order and determine the complexity order of different algorithms.
- Explain how different data structures are used for different algorithms.
- Recognize and implement common searching and sorting algorithms.

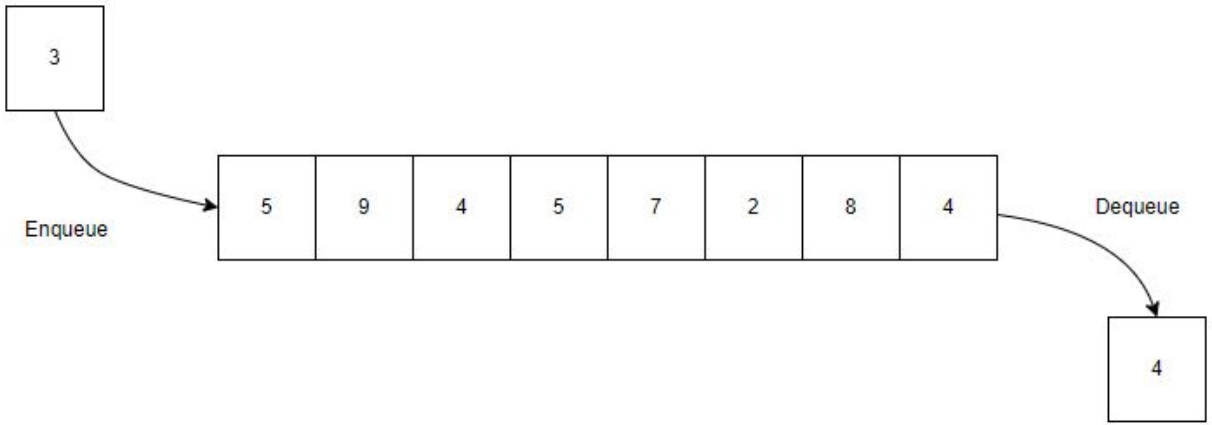
Data Structures

CoGrammar

Queues

- Data structure that works with a first-in-first-out protocol(FIFO)
- Each item getting added to a queue gets added to the back and has to wait its turn to exit. Similar to a queue in real life say at an ATM.
- The first item added to our list will be the first item to leave the list with each consecutive item being allowed to exit as the value after the value in front of it has exited.

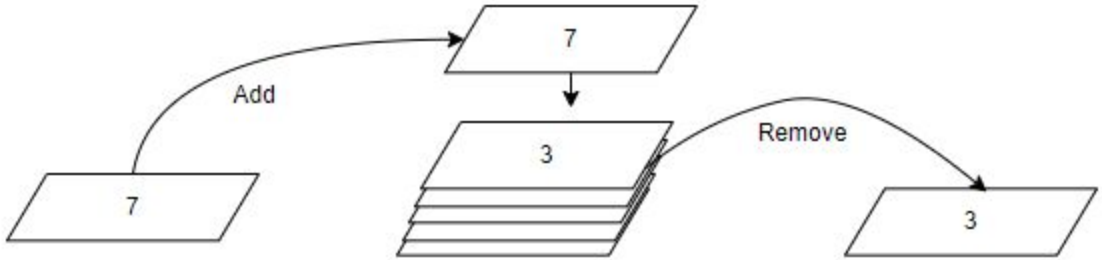
Queues



Stacks

- Stacks are similar to queues but use a last-in first-out protocol(LIFO)
- When multiple items get added to the stack the first item entering the stack can only exit when the item before it has been removed from the stack.
- Similar to a deck of cards in real life where we pick up each card from the stack of cards one by one. We first have to remove the top card to reveal the next one.

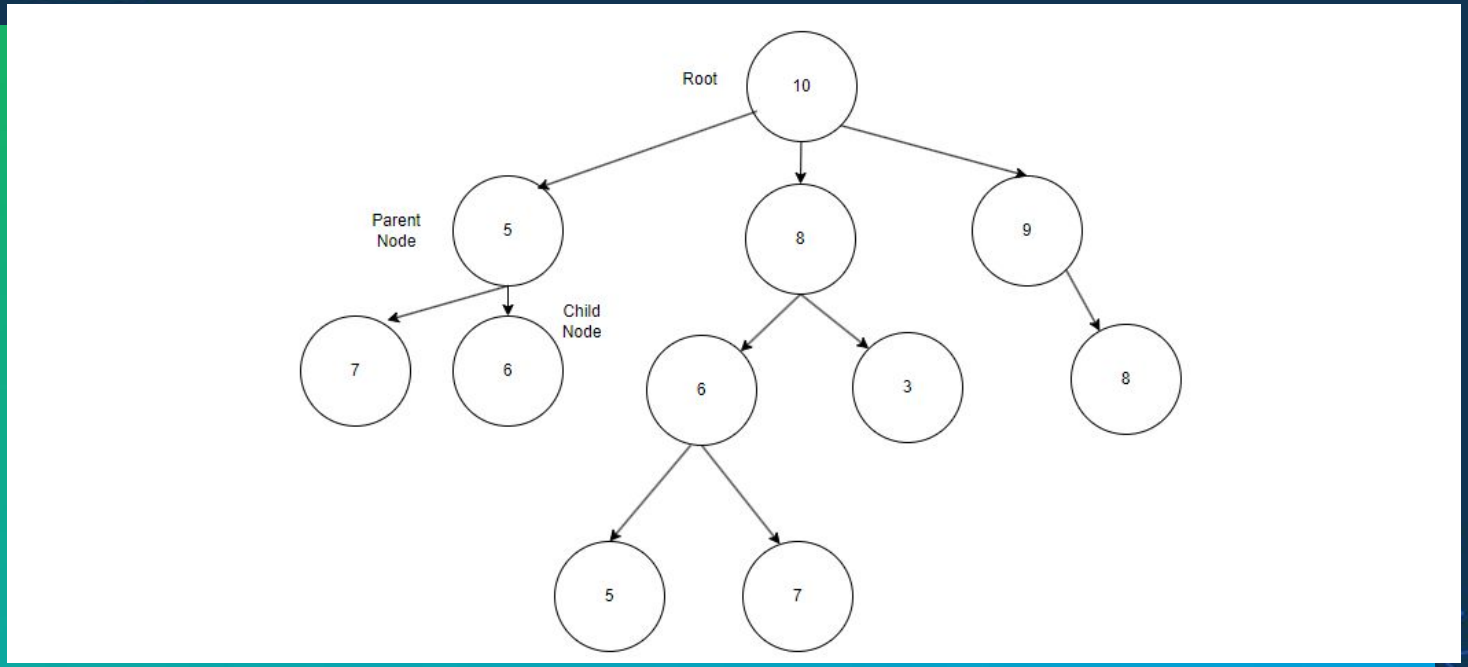
Stacks



Trees

- Trees are a type of structure that has a main node called a root with subsequent nodes connected to it as parents and more nodes connected to those nodes as children.
- This creates a branch-like structure resembling a tree.

Trees



Algorithms

CoGrammar

What is an Algorithms?

- Set of instructions that can solve a problem.
- Every time you write code you are creating an algorithm.
- Provides us with a systematic way to solve problems and automate tasks.

Algorithm Characteristics

- **Input:** Algorithms take input data, which can be in various forms, and process it to produce an output.
- **Deterministic:** Algorithms will always produce the same output for a given input. There is no randomness or uncertainty in how they operate.
- **Finite:** Algorithms must have a finite number of steps or instructions. They cannot run forever, and should produce an output or terminate.

Algorithm Characteristics

- **Clear and Unambiguous:** Algorithms are expressed in a way that leaves no room for interpretation or ambiguity. Each step must be precisely defined and understandable
- **Effective:** Algorithms are designed to be effective, meaning they solve the problem efficiently. This often involves optimizing for factors like time complexity and space complexity
- **General Applicability:** Algorithms can be applied to a range of instances of a problem. They are not tied to a specific set of inputs but can handle a variety of cases within the problem domain

Algorithms

- Play a big role in everyday life.
- Used in various fields such as computer science, mathematics and engineering.
- They are the building blocks for computer programs and are used to perform tasks like searching and sorting.

Complexity Order

CoGrammar

What is complexity order?

- In computer science, the order of complexity is used to describe the relative representation of complexity of an algorithm.
- It describes how an algorithm performs and scales, and is the upper bound of the growth rate of a function.
- Time complexity and Space complexity.
- We use Big O notation to express the complexity of an algorithm.

Time Complexities

- **O(1) Constant Time Complexity**
  - Remains constant regardless of input size
  - Accessing a list element with it's index or performing a basic arithmetic calculation
- **O(log n) Logarithmic Time Complexity**
  - Execution time grows logarithmically with input size
  - Very Efficient
  - Binary search
- **O(n) Linear Time Complexity**
  - Execution time grows linearly with input size
  - Iterating over each element in a list

Time Complexities

- **O(n^2) Quadratic Time Complexity**
  - Execution time grows quadratically with input size
  - Nested iterations over input data
  - Bubble sort
- **O(2^n) Exponential Time Complexity**
  - Execution time grows exponentially with input size
  - Highly inefficient
  - Brute force algorithms
  - Brute force algorithm solves problems by going through every possible option until a solution is found

Time Complexities

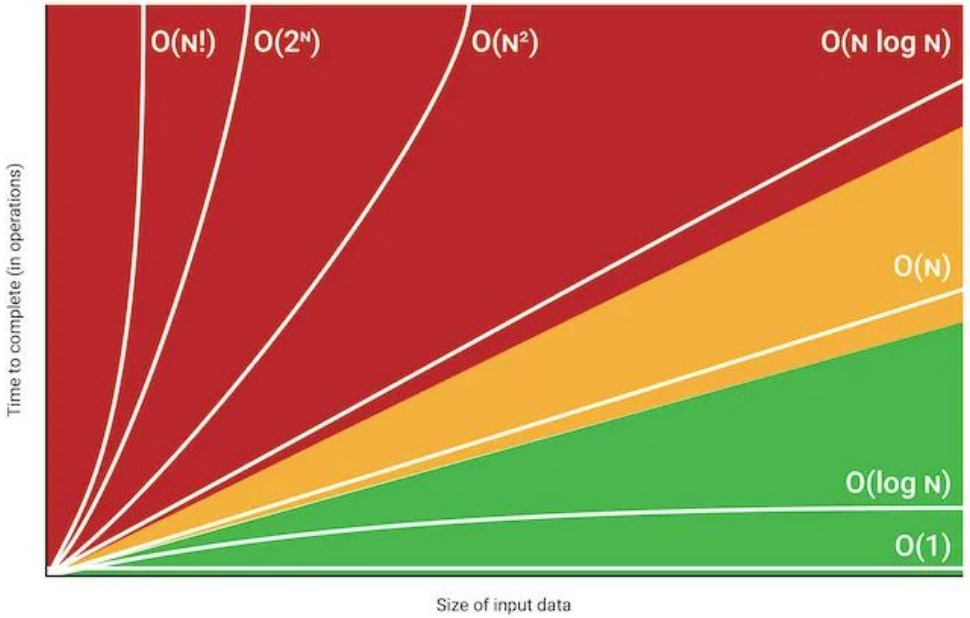


Image source: https://pub.towardsai.net/big-o-notation-what-is-it-69cfd9d5f6b8

CoGrammar

Analyzing Algorithm complexity

- Focus on the dominant term of n(input size)
- When input becomes very large the other term become negligible to the dominant term
- Consider the number of operations your function performs with regards to the input size
- Try to identify the factor that influences the growth rate the most
- Express this factor using big o notation

Advantages of Complexity order

- We can compare algorithms and determine which ones are better than others
- We have an estimate runtime for an algorithm helping us determine if it will be useful for the task at hand
- Helps us determine the areas in our algorithms that have the highest time complexity. This allows us to optimize and improve our algorithms

Sorting

CoGrammar

Bubble sort

- Larger values tend to bubble up to the top of the list
- Compare an item to the item next to it
- If the first value is larger than the second value swap places
- This process repeats until all values are compare and starts the process again
- This will run for as many times as 1 less than the length of the list to ensure enough passes were made

Bubble sort

```
Bubble Sort(arr, size)
        for i=0 to n-i-1
                for j=0 to n-i-2
                        if arr[j]>arr[j+1]
                                Swap arr[j] and arr[j+1]
```

Bubble sort

8	3	1	4	7	**First iteration:** Five numbers in random order.
3	**8**	1	4	7	3 < 8 so 3 and 8 swap
3	1	**8**	4	7	1 < 8, so 1 and 8 swap
3	1	4	**8**	7	4 < 8, so 4 and 8 swap
3	1	4	7	**8**	7 < 8, so 7 and 8 swap (do you see how 8 has bubbled to the top?)
3	1	4	7	8	**Next iteration:**
1	**3**	4	7	8	1 < 3, so 1 and 3 swap. 4 > 3 so they stay in place and the iteration ends

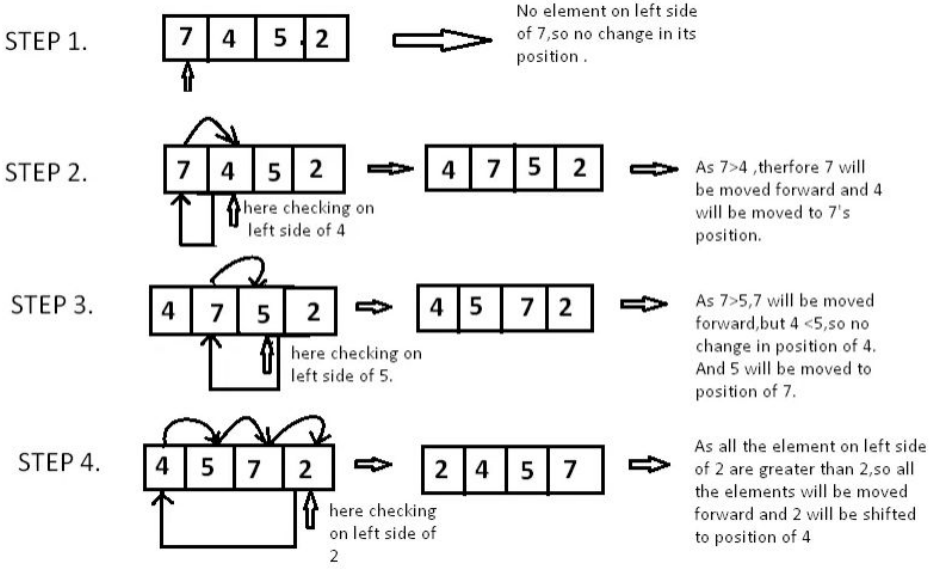
Insertion Sort

- Sorts an array of values one item at a time by comparison.
- Looks at one item at a time and compares it to the items in the sorted array.
- The item gets swapped with the items in the sorted array until it reaches the correct position.

Insertion Sort

```
procedure insertion_sort(array : list of sortable
items, n : length of list)
    i ← 1
    while i < n
        j ← i
        while j > 0 and array[j - 1] > array[j]
            swap array[j - 1] and array[j]
            j ← j - 1
        end while
        i ← i + 1
    end while
end procedure
```

Insertion Sort



CoGrammar

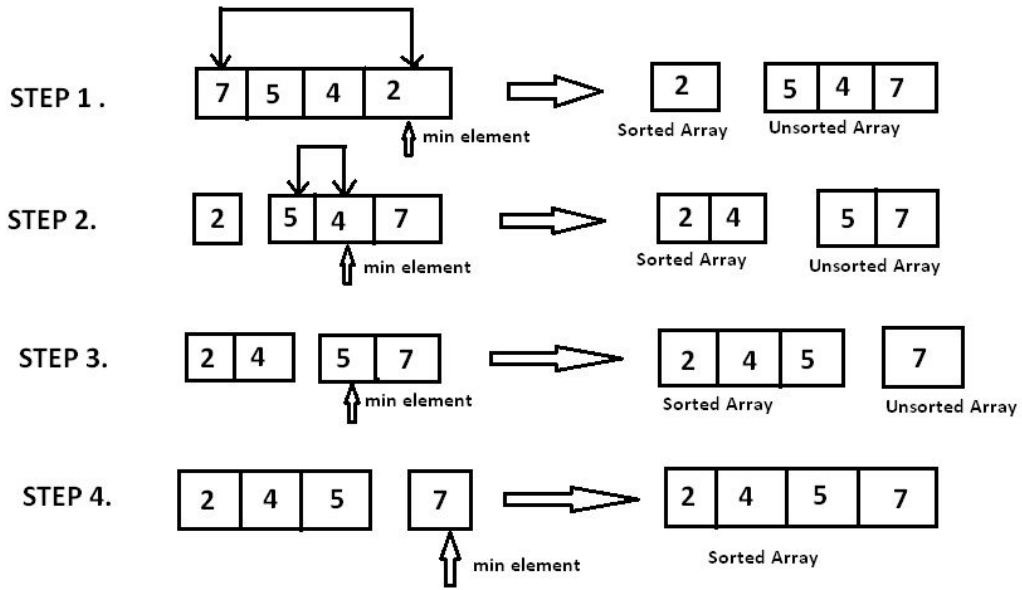
Selection Sort

- Starts by taking the first position and moving the smallest number in the array into this position.
- Now the value in the first position is in the correct order we can move to the second position.
- Again we compare all the values to get the smallest value and move it into the second position.
- Continue this process until all values are moved to the correct position.

Selection Sort

```
procedure selection_sort(array : list of sortable items, n : length of list)
        i ← 0
        while i < n - 1
            min_index ← i
            j ← i + 1
            while j < n
                if array[j] < array[min_index]
                    min_index ← j
                end if
                j ← j + 1
            end while
            swap array[i] and array[min_index]
            i ← i + 1
        end while
    end procedure
```

Selection Sort



Searching

CoGrammar

Searching Algorithms

- Two main sorting algorithms
  - Linear Search
  - Binary Search
- Linear search is closest to how we as humans would look for something
- If we have a set of sorted values we can use Binary search to achieve a much quicker result

Linear Search

- Start by knowing what element we want
- We then look at each of the other elements and compare them to the one we are looking for
- Once we get the correct element or reach the end of the list the process stops
- O(n)

Linear Search

```
Function linear search(array, target_value)
    For value in array
        If value == target_value:
            return item
    Else:
        return -1
```

Binary Search

- Can only be used if the values in the list are in order
- We know what value we are looking for but instead of looking at every value in the list we go straight to the middle of the list
- We then check if the value we are looking for is bigger or smaller than the middle value
- The middle value being bigger or smaller will determine where we cut the list to get rid of the unnecessary values
- We keep repeating these steps until we find the correct value or list cannot be divided further.

Binary Search

```
function binary_search(list, target):
    left = 0
    right = length(list) - 1
    while left <= right:
        mid = (left + right) // 2
        If list[mid] == target:
            return mid
        elif list[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Questions and Answers

CoGrammar

Let’s take a short break

CoGrammar

Summary

CoGrammar

Summary

1. Data structures
   - ➢ We have different data structures that allow us to crete very powerful algorithms. Queues, Stacks, Trees and many more.
2. Complexity Order
   - ➢ We can determine how a algorithm will scale with the input by calculating the complexity order of an algorithm.
3. Searching and Sorting
   - ➢ We don't have to reinvent the wheel. The are common search and sort patterns we can learn and use within our own project. Bubble, merge and quick sort alongside linear and binary search.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar