



## Welcome to this session: Coding Interview Workshop - Order Complexity

**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



Rafiq Manan



Charlotte Witcher



Nurhaan Snyman



Ronald Munodawafa



Tevin Pitts

Scan to report a  
safeguarding concern



or email the Designated  
Safeguarding Lead:  
Ian Wyles

[safeguarding@hyperiondev.com](mailto:safeguarding@hyperiondev.com)

# 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](http://www.hyperiondev.com/support)**
- **Report a safeguarding incident:** **[www.hyperiondev.com/safeguardreporting](http://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

---


- ❖ **Define functions and function properties** in the context of Mathematics
- ❖ **Outline function notations** and **demonstrate how to visualise functions** based on function equations.
- ❖ **Explain the concept of order complexity (Big O notation)** to analyse the time and space efficiency of algorithms.
- ❖ **Differentiate between best, worst and average case** in order complexity.
- ❖ **Develop and analyse simple algorithms** in Python to demonstrate understanding of variables, functions, and complexity analysis.

# What is a function in mathematics?

- A. A set of ordered pairs.
- B. A rule that assigns exactly one output to each input in a specific domain.
- C. A sequence of instructions executed in a program.
- D. A process to find the limit of a value.



# In which scenario is average-case complexity most important to consider?

- A. When analyzing the best-case scenario.
  - B. When analyzing the typical performance for random inputs.
  - C. When only the worst-case performance matters.
  - D. When analyzing the space complexity exclusively.
- 



# Lecture Overview

---

- Variables
- Functions
- Worked Example 1
- Algorithmic Complexity
- Asymptotes
- Application of Complexities
- Worked Example 2





# Variables

**Symbols that represent values in mathematical expressions or algorithms.**

- ❖ Symbols are usually letters like **x**, **y** or **z**, but they can be any symbol.

$$\begin{aligned}x &= 3 \\ y + 11 &= 6z - 51\end{aligned}$$

- ❖ Variables are used in the place of an **unspecified** or **unknown** value.
- ❖ In our code, **variables** are symbols used to represent values stored in the computer's memory.
- ❖ The function like **containers** which store different types of information.

```
x = 3
y = 8
z = x + y
```

# Functions

A relation between a set of inputs and a set of permissible outputs with the property that each input is related to at most one output.

- ❖ The **sets** that make up the input and output of a function are known as the **domain** and the **range** respectively.
- ❖ The standard notation for a function is:

$$f : A \rightarrow B$$

where  $f$  is the name of the function,  $A$  the domain and  $B$  the range.

- ❖ The elements in the domain of the function are referred to with the **variable  $x$**  and the corresponding element in the range is referred to with  **$y$** . This relationship is denoted as:

$$f(x) = y$$

## Univariate and Multivariate Functions

- ❖ Univariate functions relate one input to at most one output.

$$f(x) = y$$

- ❖ Multivariate functions relate multiple inputs to at most one output.

$$f(x, y) = z$$

## Function Representations

- ❖ Functions can be represented as an algebraic equation using variables to refer to the input.

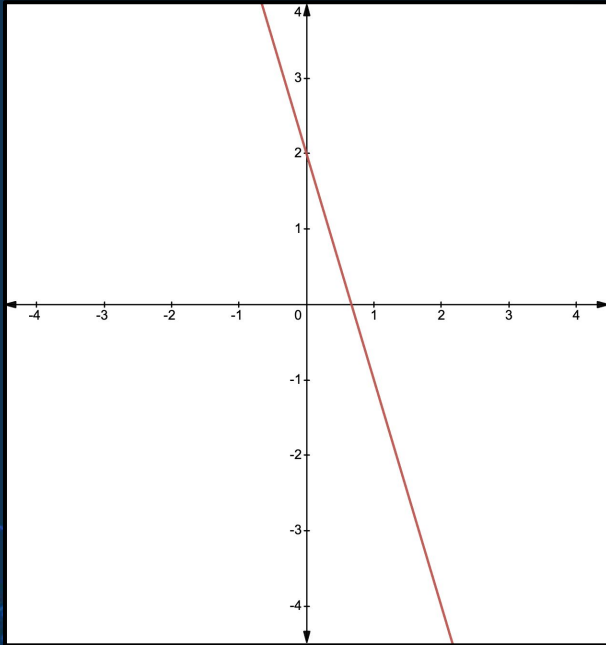
$$f(x) = 3x + 9$$

$$f(x) = \log(x)$$

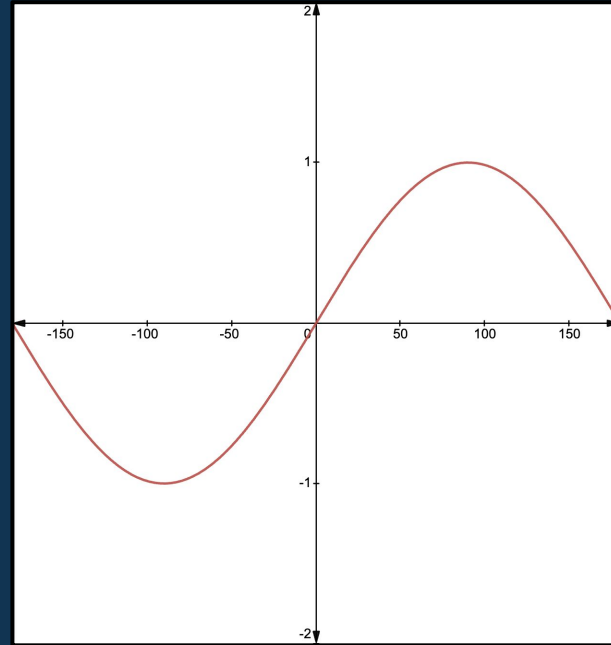
$$f(x, y) = 4x^2 + 3xy - 6$$

- ❖ Functions can also be represented as graphs by using the horizontal x-axis to represent the input to the function and the vertical y-axis to represent the output of the function.

$$f(x) = y$$
$$y = -3x + 2$$

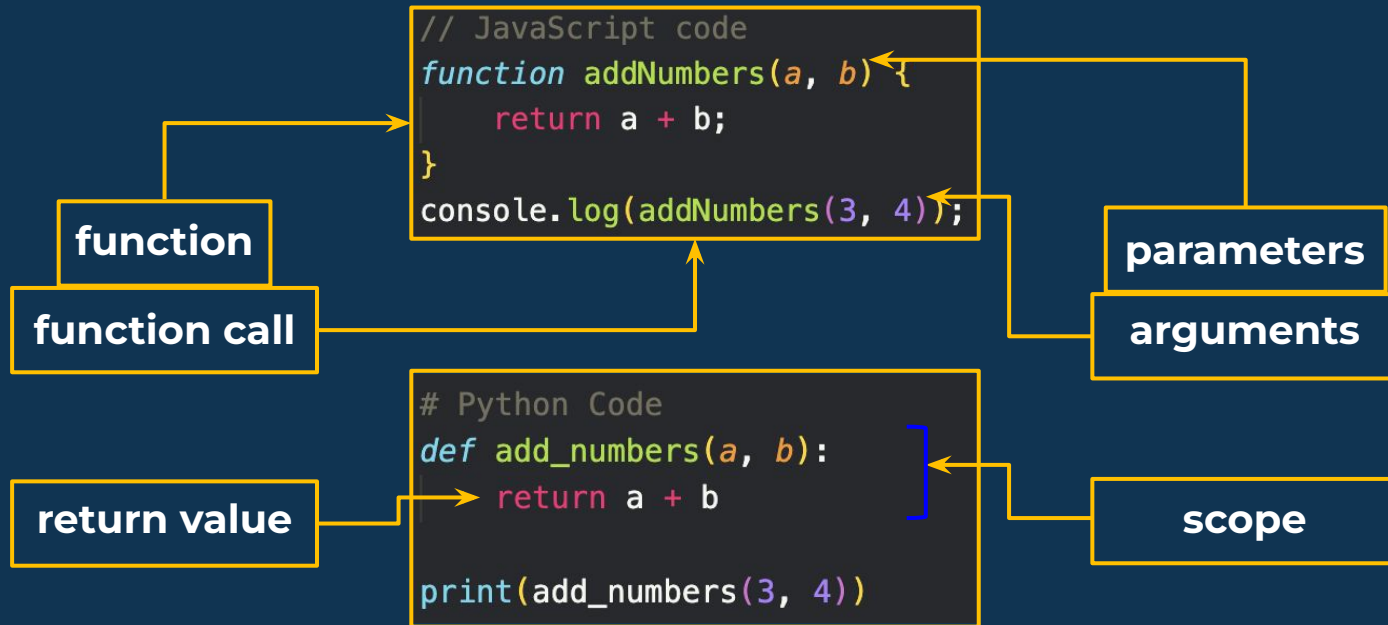


$$g(x) = y$$
$$y = \sin(x)$$



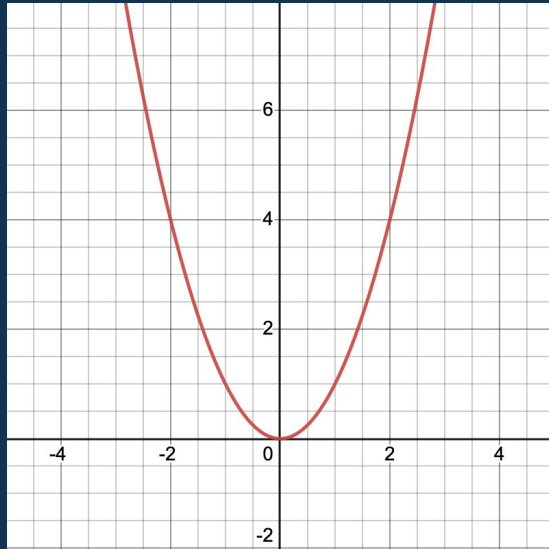
# Functions

- ❖ Functions can be defined in our code as well:



## Worked Example

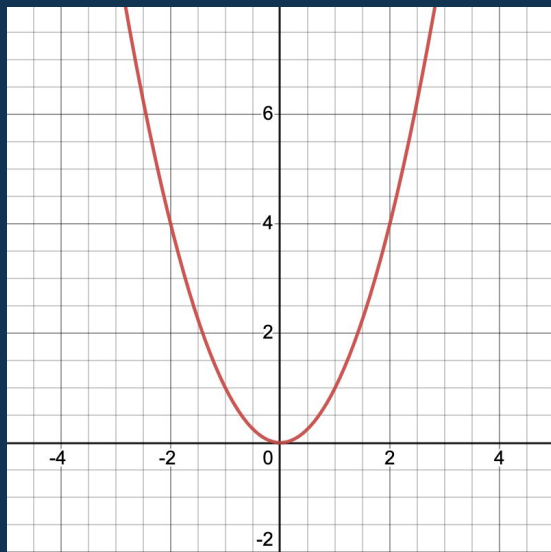
Determine the range, domain and equation of the function represented in the graph below.



1. Determine the range and domain of the function.
2. Determine the equation of the function.

## Worked Example

Determine the range, domain and equation of the function represented in the graph below.



1. Determine the range and domain of the function.

**Domain =  $(-\infty, \infty)$**

**Range =  $[0, \infty)$**

2. Determine the equation of the function.

**Choose some points and evaluate whether there is a relationship.**

**Point A: (1, 1)**

**Point B: (2, 4)**

**Note some of the qualities of the graph: No negative values, symmetrical about the y-axis...**

**Equation  $\rightarrow f(x) = x^2$**



# Algorithmic Complexity

The measure of the amount of resources, such as time and/or space, required by an algorithm to solve a problem as a function of the size of the input.

- ❖ This definition tells us that the amount of resources can be **quantified** and **changes** as the **input size changes**.
- ❖ **Algorithms** are sets of instructions to solve **specific problems**. We can compare algorithms by comparing their **complexity**.
- ❖ **Space complexity**: measures the amount of **memory space** required by an algorithm as a function of the input size.
- ❖ **Time complexity**: measures the amount of **time** an algorithm takes to complete as a function of the input size.

# Algorithmic Complexity

- ❖ In both cases, the complexity (which we can represent as the **output of a function**), is **dependent** on the **size of the input**.
- ❖ **Big O notation** helps us understand the behaviour of an algorithm as the input size approaches infinity.
- ❖ When evaluating the complexity of an algorithm, the algorithm may perform differently given specific conditions e.g.
  1. When finding an item in a list ->
    - **Best case scenario:** Item is the first element in the list
    - **Worst case scenario:** Item is not in the list
- ❖ This is why usually when we evaluate complexity of an algorithm, we note the **best, worst and average case**.

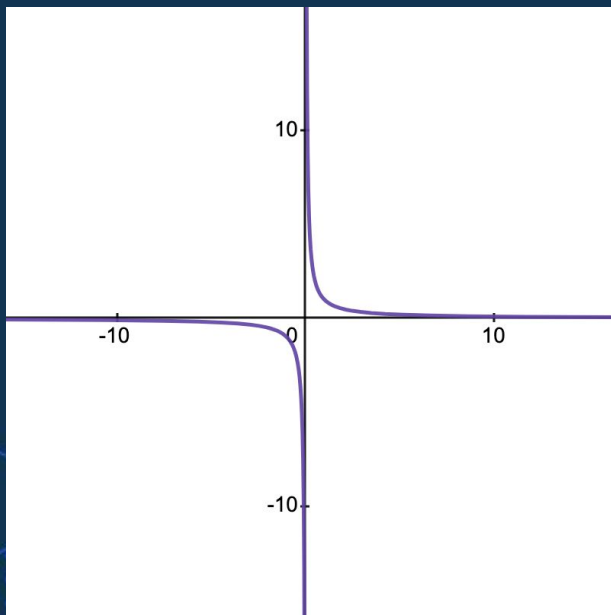
# Asymptotes

An asymptote is a line that a curve approaches as it heads towards infinity.

- ❖ We learn about asymptotes to **understand and predict** the behavior of algorithms as the size of their **input grows**, which is crucial for evaluating their **efficiency** and **scalability**.
- ❖ **Vertical asymptotes** illustrate points where an algorithm fails or becomes highly inefficient.
- ❖ **Horizontal asymptotes** represent algorithms whose performance stabilizes as input size grows.
- ❖ **Oblique asymptotes** indicate algorithms with complexity increasing non-linearly with input size.

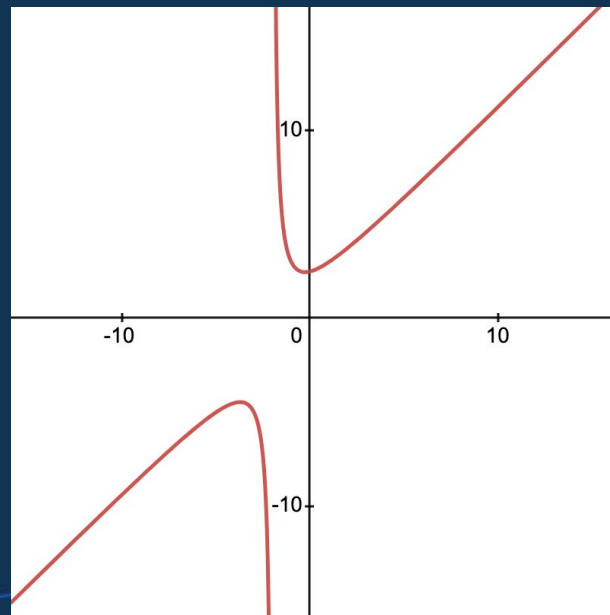
## Vertical and Horizontal Asymptotes

$$f(x) = \frac{1}{x}$$

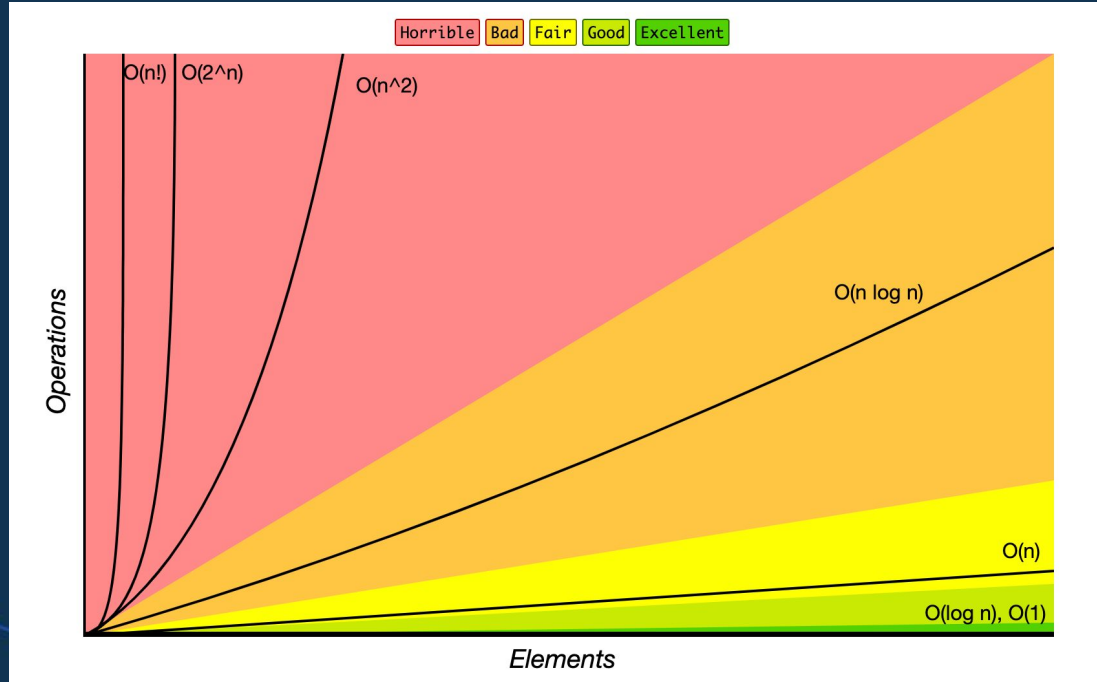


## Oblique Asymptotes

$$f(x) = \frac{x^2 + 3x + 5}{x + 2}$$



# Common Complexities



# Example: Linear Search

# Python Code

```
def linear_search(arr, target):  
    for i in range(len(arr)):  
        if arr[i] == target:  
            return i # Return the index if the target is found  
    return -1 # Return -1 if the target is not found
```

// JavaScript Code

```
function linearSearch(arr, target) {  
    for (let i = 0; i < arr.length; i++) {  
        if (arr[i] === target) {  
            return i; // Return the index if the target is found  
        }  
    }  
    return -1; // Return -1 if the target is not found  
}
```

- ❖ The worst case is if the target is the **last element**, meaning we **iterate over all  $n$  elements** of the input array, thus the **worst-case time complexity is  $O(n)$** .
- ❖ No matter the input, only space to **store  $i$**  and **target** are needed, so the **worst-case space complexity is  $O(1)$** .

# Example: Fibonacci Sequence

```
# Python Fibonacci Code
def fibonacci(n):
    if n <= 1:
        return n # T0 = 0, T1 = 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

```
// JavaScript Fibonacci Code
function fibonacci(n) {
    if (n <= 1) {
        return n; // T0 = 0, T1 = 1
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

- ❖ Each call generates two more calls, which results in an **exponential growth** in complexity, in fact the **worst-case time complexity is  $O(2^n)$** .
- ❖ This is a **recursive function**, so it is stored as a **“stack”** in memory, which we won't get into here. For each input it adds a layer to the stack, making the **worst-case space complexity  $O(n)$** .



# Comparing Complexities

```
# Python Binary Search Code
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1
```

- ❖ Linear search has  **$O(n)$  time complexity**. Binary search has  **$O(\log(n))$  time complexity**.
- ❖ Both have  **$O(1)$  space complexities**.
- ❖ We already graphed the difference between  $O(n)$  and  $O(\log(n))$  time complexities.
- ❖ Looking at the previous example we see that if the **input is 1 billion**  $O(n)$  will take **1 billion time units** where  $O(\log(n))$  would take **9 time units at most**.

# Common Sorting and Searching Algorithms

## Common sorting algorithms:

1. **Bubble Sort** - Time Complexity:  $O(n^2)$ , Space Complexity:  $O(1)$
2. **Selection Sort** - Time Complexity:  $O(n^2)$ , Space Complexity:  $O(1)$
3. **Insertion Sort** - Time Complexity:  $O(n^2)$ , Space Complexity:  $O(1)$
4. **Merge Sort** - Time Complexity:  $O(n \log(n))$ , Space Complexity:  $O(n)$
5. **Quick Sort** - Time Complexity:  $O(n \log(n))$  average,  $O(n^2)$  worst-case, Space Complexity:  $O(\log(n))$
6. **Heap Sort** - Time Complexity:  $O(n \log(n))$ , Space Complexity:  $O(1)$

# Common Sorting and Searching Algorithms

## Common searching algorithms:

1. **Linear Search** - Time Complexity:  $O(n)$ , Space Complexity:  $O(1)$
2. **Binary Search** - Time Complexity:  $O(\log(n))$ , Space Complexity:  $O(1)$
3. **Jump Search** - Time Complexity:  $O(\sqrt{n})$ , Space Complexity:  $O(1)$
4. **Interpolation Search** - Time Complexity:  $O(\log(\log(n)))$  average,  $O(n)$  worst-case, Space Complexity:  $O(1)$

## Worked Example

You are optimising a program that sorts large datasets. You have two sorting algorithms at your disposal: **Bubble Sort** and **Quicksort**. You need to decide which one to use based on their time and space complexities.

- Which algorithm would you choose for sorting very large datasets and why?
- How do the time complexities of Bubble Sort and Quicksort differ for large datasets?

## Worked Example

You are optimising a program that sorts large datasets. You have two sorting algorithms at your disposal: **Bubble Sort** and **Quicksort**. You need to decide which one to use based on their time and space complexities.

- Which algorithm would you choose for sorting very large datasets and why?

**For sorting very large datasets, Quicksort would be the preferred algorithm over Bubble Sort. This is because Quicksort has a better average and worst-case time complexity than Bubble Sort. Quicksort generally operates in  $O(n \log n)$  time, whereas Bubble Sort operates in  $O(n^2)$  time. For large datasets, the  $n^2$  complexity of Bubble Sort makes it impractically slow compared to the more efficient  $n \log n$  complexity of Quicksort.**

## Worked Example

You are optimising a program that sorts large datasets. You have two sorting algorithms at your disposal: **Bubble Sort** and **Quicksort**. You need to decide which one to use based on their time and space complexities.

- How do the time complexities of Bubble Sort and Quicksort differ for large datasets?

**As mentioned, Bubble Sort has a time complexity of  $O(n^2)$ , meaning the time it takes to complete the sorting operation grows quadratically as the size of the dataset increases. Quicksort, on the other hand, has an average time complexity of  $O(n \log n)$ , indicating that it scales more effectively with larger datasets. The time required by Quicksort grows at a rate proportional to the size of the dataset multiplied by its logarithm, which is significantly more efficient than the quadratic growth of Bubble Sort for large datasets.**

## Worked Example

Imagine you are developing a mobile application that needs to sort small arrays of data. You are considering **Insertion Sort** and **Merge Sort** for this task. Given the constraints of mobile devices, such as limited memory, your choice should factor in both time and space complexities.

- For small datasets, which sorting algorithm would be more efficient and why?
- How does the space complexity of Merge Sort impact its suitability for memory-constrained environments?



## Worked Example

Imagine you are developing a mobile application that needs to sort small arrays of data. You are considering **Insertion Sort** and **Merge Sort** for this task. Given the constraints of mobile devices, such as limited memory, your choice should factor in both time and space complexities.

- For small datasets, which sorting algorithm would be more efficient and why?

**For small datasets, Insertion Sort might be more efficient than Merge Sort. This is because Insertion Sort has a lower overhead and is generally faster on small datasets due to its simpler algorithm. Although Merge Sort has a better overall time complexity  $O(n \log n)$  compared to  $O(n^2)$  for Insertion Sort, the constant factors and simpler operations of Insertion Sort often make it faster for small arrays.**

## Worked Example

Imagine you are developing a mobile application that needs to sort small arrays of data. You are considering **Insertion Sort** and **Merge Sort** for this task. Given the constraints of mobile devices, such as limited memory, your choice should factor in both time and space complexities.

- How does the space complexity of Merge Sort impact its suitability for memory-constrained environments?

**Merge Sort has a space complexity of  $O(n)$ , meaning it requires additional memory proportional to the size of the dataset. In memory-constrained environments, such as mobile devices, this additional memory requirement can be a significant drawback. The need for extra space to hold the divided arrays during the merge process might not be ideal for environments with limited memory availability. This consideration makes Merge Sort less suitable compared to algorithms with a constant space complexity, like Insertion Sort, which has a space complexity of  $O(1)$  and does not require additional memory beyond the input array.**



# Which statement best describes Big O notation?

- A. A measure of the exact runtime of an algorithm.
- B. A measure of the growth rate of an algorithm's time or space requirements relative to input size.
- C. A measure of an algorithm's success rate.
- D. A method of sorting functions in increasing order.



**What is the order complexity of a function that doubles in operations for every increase in input size?**

- A.  $O(n)$
- B.  $O(n^2)$
- C.  $O(\log n)$
- D.  $O(2^n)$

## Homework

---

Analyse algorithms and determine the space and time complexities for each of them. Determine whether there are any changes which can be made to the algorithm in order to make them more efficient.

- ❖ You can use the code files up on GitHub.
- ❖ You can take a look at the following LeetCode question.
- ❖ Imagine you are answering this question in a job interview, how can you communicate your skills and aptitudes by answering a question like this?

# 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**



  
Department  
for Education