




Welcome to the **Co**Grammar Sets, Variables, Functions and Order Complexity Lecture

The session will start shortly...

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



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

60 Guided Learning Hours

Data Science - **13 tasks**

Software Engineering - **13 tasks**

Web Development - **13 tasks**

Due Date: 28 April 2024

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

Completion: All mandatory tasks,
including Build Your Brand and
resubmissions by study period end
Interview Invitation: Within 4 weeks
post-course
Guided Learning Hours: Minimum of
112 hours by support end date
(10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship
Outcome: Document within 12
weeks post-graduation
Relevance: Progression to
employment or related
opportunity



**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Sets, Variables, Functions and Order Complexity

March 2024

Brief Overview of CIWs

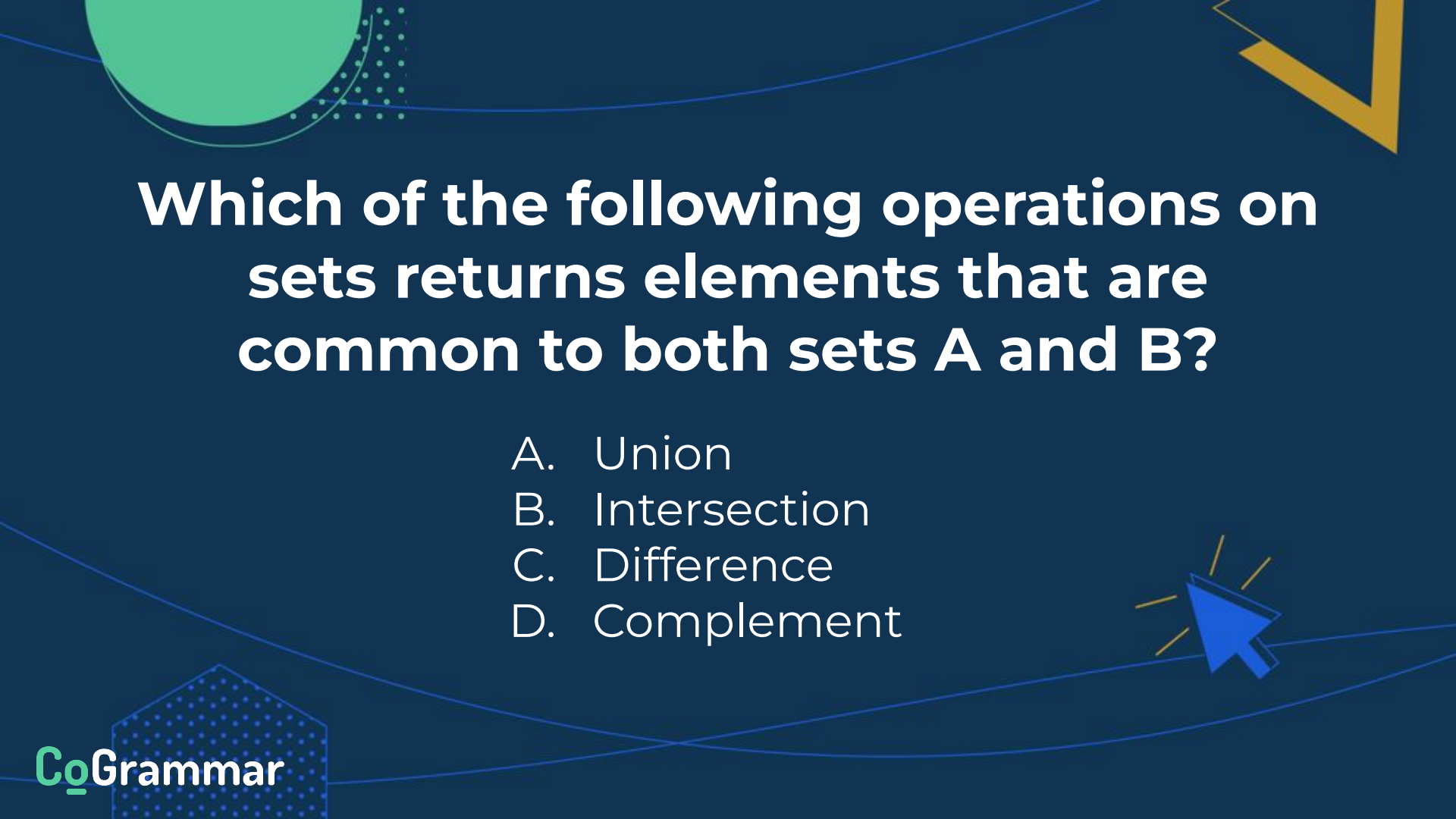


Learning Objectives

- ❖ Define and exemplify **different types of sets** and their **operations**
- ❖ Solve basic **number theory** problems.
- ❖ Identify and use various **data types** in Python and JavaScript, demonstrating **type conversion** and **variable assignment**.
- ❖ Define and invoke **functions** in Python and JavaScript, understanding **parameters**, **return values**, and **scope**.

Learning Objectives

- ❖ Understand and explain the concept of **order complexity (Big O notation)** to analyze the **time and space efficiency** of algorithms.
- ❖ Apply **set and number theory** concepts to solve practical problems, like **data filtering and organization**.
- ❖ Develop and analyze simple algorithms in Python and JavaScript to demonstrate understanding of **variables, functions, and complexity analysis**.



Which of the following operations on sets returns elements that are common to both sets A and B?

- A. Union
- B. Intersection
- C. Difference
- D. Complement

Correct Answer: B

- ❖ **Sets** are a fundamental concept in both Mathematics and Computer Science.
- ❖ They are a **data structure** used to store **unique elements** and perform **set operations**.
- ❖ There are various **set operations** which we need to be familiar with since they are useful in various algorithms and tasks like **data filtering and processing**.
- ❖ The **intersection** of two sets is a new set containing only the elements that are **common to both sets**.

Correct Answer: B

- ❖ The definitions of the other incorrect operations are:
 - **Union of set A and set B:** A new set containing all unique elements from from both sets.
 - **Difference between set A and set B:** A new set containing elements from set A but not from set B.
 - **Complement of set A:** A set which contains all the elements that are not in set A.



In computer science, which of the following is a valid variable name convention?

- A. Using spaces between words
- B. Starting variable names with a number
- C. Using camelCase
- D. Using special characters

Correct Answer: C

- ❖ When it comes to naming variables there **rules** and **guidelines**.
- ❖ This is to ensure that our code adheres to **programming principles**, and is **clean** and **readable**.
- ❖ For this question, the correct answer is a **guideline**, not a strict rule.
- ❖ **camelCase** is a way of name variables where the first letter of every word in the name is **capitalised**, expect for the first word e.g. myVariableName.

Correct Answer: C

- ❖ The incorrect options in this questions are all rules that need to be adhered to, except for the last option:
 - Variable names **cannot contain spaces**
 - Variable names **cannot start with numbers**
 - Variable names **cannot contain special characters** except “_” (both Python and JavaScript) or “\$” (only JavaScript), but this is generally avoided to ensure readability.

What is the output of the following Python and JavaScript code?

```
// JavaScript code
function addNumbers(a, b) {
    return a + b;
}
console.log(addNumbers(3, 4));
```

```
# Python Code
def add_numbers(a, b):
    return a + b

print(add_numbers(3, 4))
```

- A. 7
- B. Nothing
- C. "34"
- D. "7"

Correct Answer: A

- ❖ This question tests your understanding of **functions definitions, function calls** and **function parameters**.
- ❖ It is common in coding interviews that you will be tasked with **predicting the outcome** of various functions.
- ❖ This demonstrates an understanding of the **flow of code**, and various **functions** and **operations**.
- ❖ In both code snippets, we define a **function** which takes in **two parameters** and returns their **sum**. We then call this function with the **arguments 3 and 4**.

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
```

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

Variables

- ❖ There are conventions when it comes to naming variables in your code:
 - Variables names **cannot start with a number**
 - Variable names **cannot contain spaces**
 - Variable names **can only contain letters, numbers** and **underscores (_) [and dollar signs in the case of JavaScript]**
 - Variable names **cannot be reserved words**
 - Variable names should be **meaningful** and should indicate their **purpose** and **type**
 - **CamelCase** is suggested where the first letter of each word is **capitalised** except for the first word e.g. myVariableName

Variables

- ❖ Variables in Mathematics are placeholders for **numbers**.
- ❖ Variables in Computer Science can store **various data types**:

JavaScript	Python	Example
Numbers	Integers and Floats	1, 7098, 192.90
Strings	Strings	"Hello World"
Booleans	Booleans	True, False
Arrays	Lists	[1, 2, 3]

Variables

- ❖ We can use **functions** to convert variables from one type to another. This is known as **type-casting**.

```
// Casting variables to other types [JavaScript]
console.log("John is " + String(age) + " years old.")
```

```
# Casting variables to other types [Python]
print("John is " + str(age) + " years old.")
```

- ❖ We can also **check the type** of data values in our code.

```
// Checking the type of a variable [JavaScript]
console.log(typeof age)
```

```
# Checking the type of a variable [Python]
print(type(name))
```

Sets

A collection of distinct, unordered objects also known as elements or members

- ❖ Objects in a set can be anything from **numbers** to **physical entities**, but each element must be **unique**.

$\{4, 8, 12, 16\}$
 $\{\text{dog, cat, rabbit, pig}\}$
 $\{1, 2, \text{buckle, my, shoe}\}$

- ❖ A set can have **any number of elements**, from 0 to infinite elements.
- ❖ We use sets to represent numbers which share **various common traits**.

prime numbers = $\{2, 3, 5, 7, 11, 13, 17, \dots\}$
factors of 24 = $\{1, 2, 3, 4, 6, 8, 12, 24\}$

Sets

- ❖ Sets have defined **operations** which can be performed on them.
- ❖ Here are a few of the most commonly used ones:
 - **Union of set A and set B ($A \cup B$):** A new set containing all unique elements from from both sets.
 - **Intersection of set A and set B ($A \cap B$):** A new set containing all elements that are in both set A and set B.
 - **Difference between set A and set B ($A \setminus B$):** A new set containing elements from set A but not from set B.
 - **Complement of set A (A'):** A set which contains all the elements that are not in set A.

Sets

- ❖ Sets can be represented in our coding using the **set data type** in both **Python** and **JavaScript**. **Set operations** are only native to Python.

```
# Python Code
# Creating sets
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Union of sets
union_set = set1.union(set2)
print("Union:", union_set)

# Intersection of sets
intersection_set = set1.intersection(set2)
print("Intersection:", intersection_set)

# Difference of sets
difference_set = set1.difference(set2)
print("Difference:", difference_set)
```

```
// JavaScript Code
// Creating sets
const set1 = new Set([1, 2, 3, 4, 5]);
const set2 = new Set([4, 5, 6, 7, 8]);
```

Let's Breathe!

Let's take a small break
before moving on to
the next topic.



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 makes up the input and output of a function is 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:

$$f(x) = y$$

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

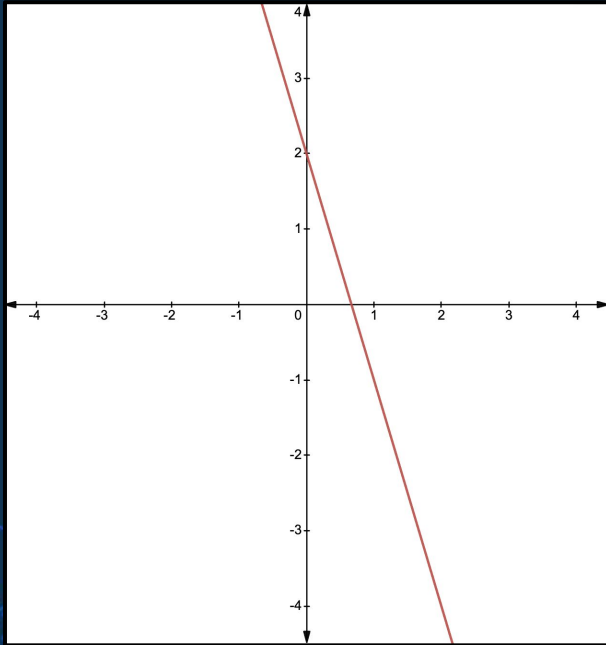
$$f : A \rightarrow B$$

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

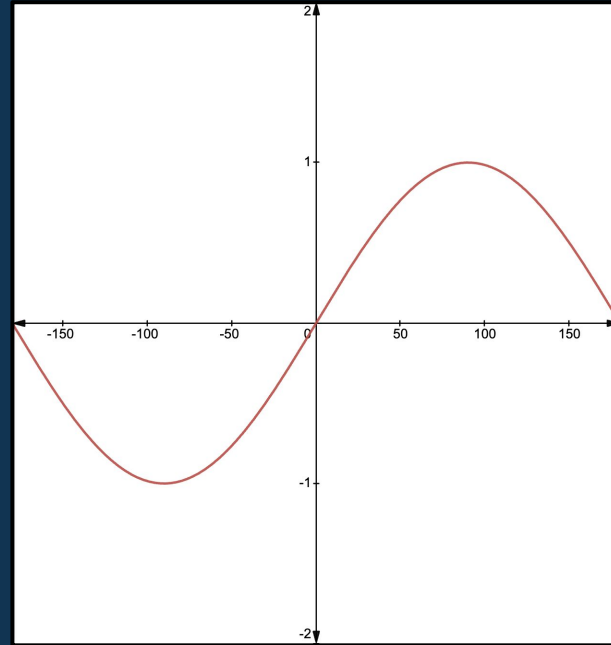
- ❖ The elements in the domain of the function are referred to with the **variable x** and the corresponding element in the range is referred to with **y** . This relationship is denoted as:

$$f(x) = y$$

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

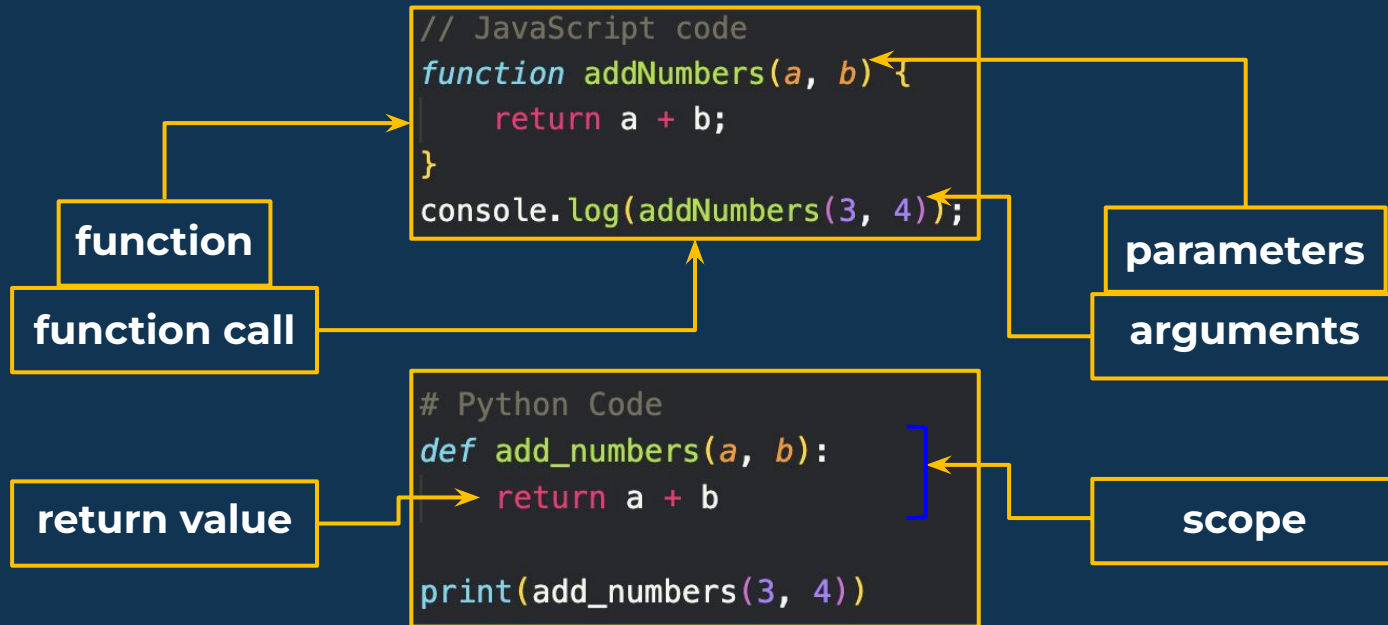


$$g(x) = y$$
$$y = \sin(x)$$



Functions

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



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.
- ❖ **Big O notation** helps us understand the behaviour of an algorithm as the input size approaches infinity.

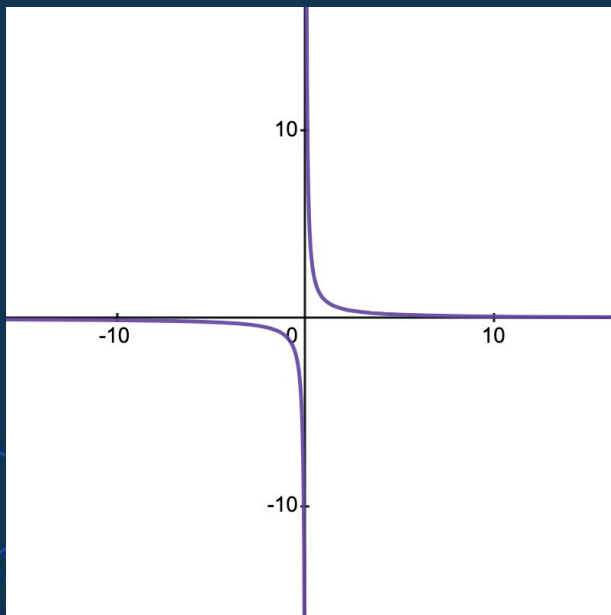
Asymptotes

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.

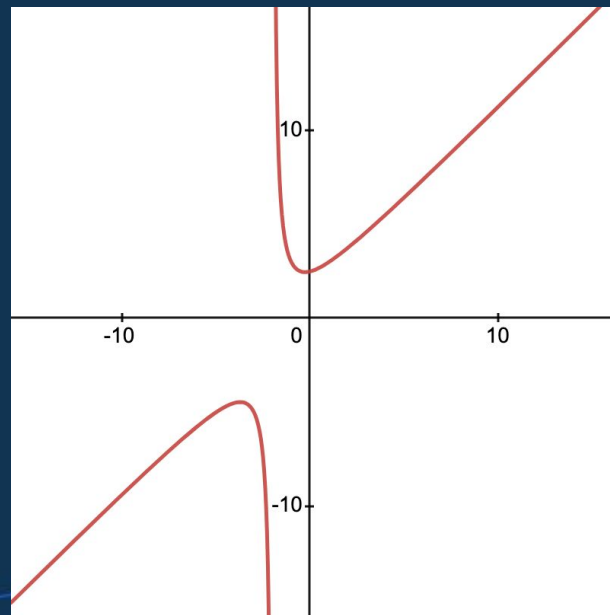
Vertical and Horizontal Asymptotes

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

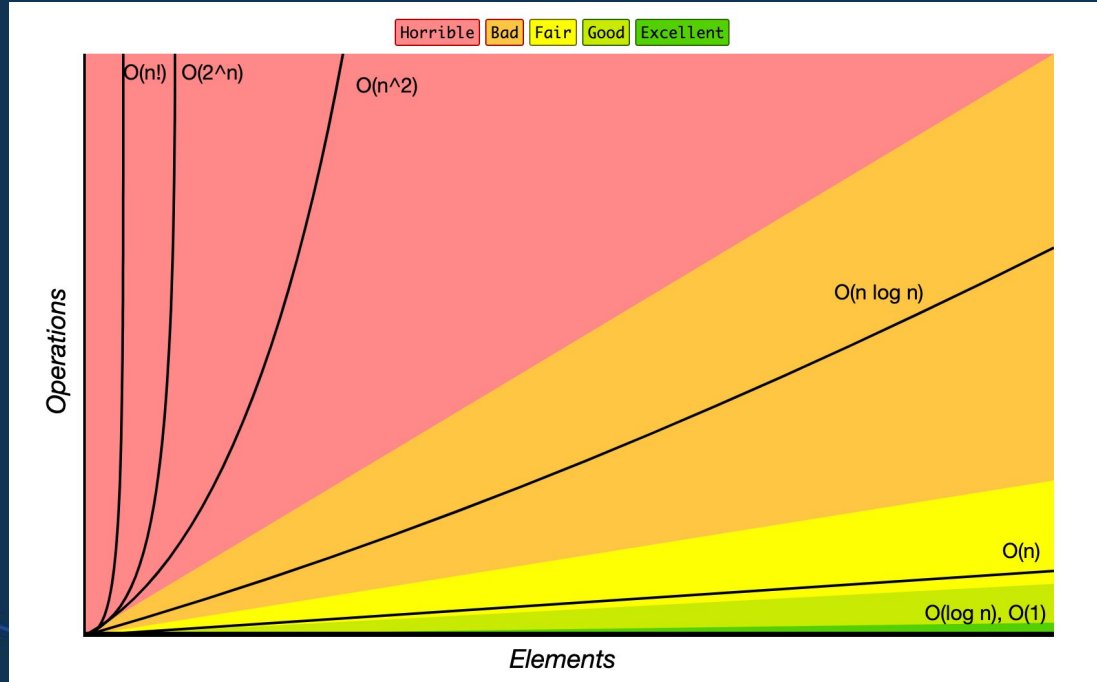


Oblique Asymptotes

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



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 $O(n)$** .

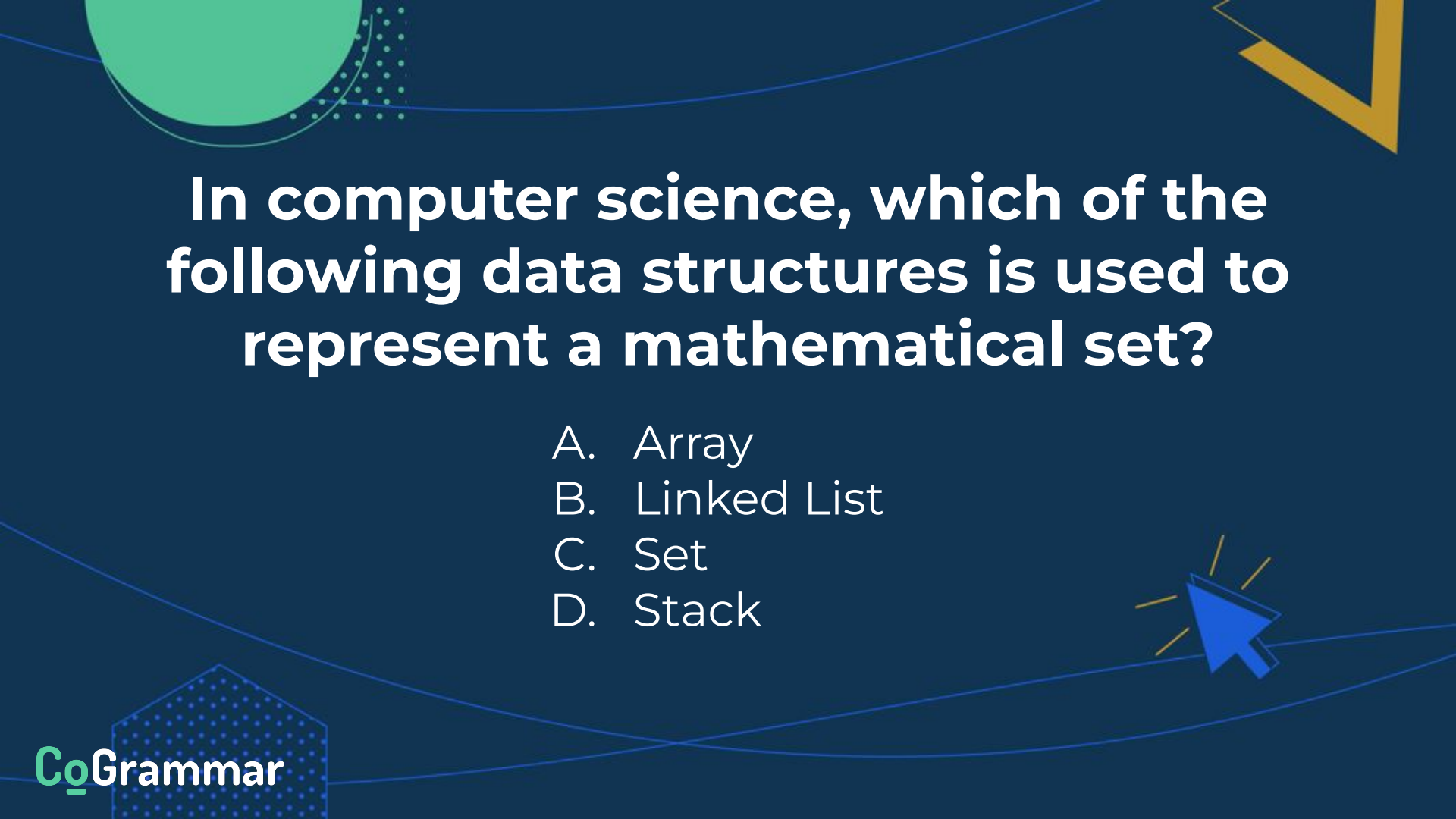
Comparing Complexities

```
# 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**.



In computer science, which of the following data structures is used to represent a mathematical set?

- A. Array
- B. Linked List
- C. Set
- D. Stack

Correct Answer: C

- ❖ Understanding different **data structures** and their appropriate **use cases** is crucial in coding interviews.
- ❖ Knowing when to use certain data structures demonstrates **practical understanding** of various data structures and their **unique characteristics** and **properties**.
- ❖ A **set** is a data structure which represents a mathematical set which can only contain unique elements and has **set operations** defined for them.

Correct Answer: C

- ❖ The other data structures do not enforce the properties of no duplicates and defined set operations. Their definitions are as follows:
 - **Arrays:** Linear data structure used to store a collection of elements, referenced by their index.
 - **Linked Lists:** Linear data structure wherein each element points to the next element.
 - **Stacks:** Linear data structure that follows the Last In, First Out principle.



What is the time complexity of a binary search algorithm?

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

Correct Answer: B

- ❖ Time complexity is a very common coding interview topic since it demonstrates the candidate's ability to **analyse** and **optimise algorithms**.
- ❖ This shows employers that writing **efficient code** is a priority for this candidate and that they are able to recognise what parts of code could result in **inefficiencies** and **bottlenecks**.
- ❖ In each step of the **binary search algorithm**, the algorithm **halves** the search space. This leads to a **logarithmic time complexity $O(\log(n))$** .

Correct Answer: B

- ❖ The other time complexities do no account for the halving of the search space. Their definitions are:
 - **$O(n)$** : Indicates that the performance is proportional to the input size. Since the search space is constantly halved, this cannot be true.
 - **$O(n^2)$** : Indicates a quadratic performance, growing with input size.
 - **$O(2^n)$** : Indicates exponential growth, where performance doubles with input. This is the opposite of the expected performance of the algorithm.

Portfolio Assignment: SE

Prime Number Finder Application

Objective: Create a simple application in Python that identifies prime numbers within a specified range. The application should allow users to input two numbers, representing the start and end of a range, and then output all the prime numbers within that range.

Portfolio Assignment: SE

Requirements:

- Use Python to develop the application.
- Incorporate functions to identify prime numbers and to handle user input.
- Demonstrate the use of integer data types and basic operations (unions, intersection).
- Provide comments to explain the logic behind prime number determination and any usage of set operations.
- Include a README file that explains how to run the application, the purpose of the program, and a brief description of the algorithm's time complexity.

Portfolio Assignment: DS

Data Filtering and Organization Tool

Objective: Develop a Python script that filters and organizes a dataset based on specific criteria. For instance, given a CSV file containing information about various individuals (e.g., name, age, occupation), the script should allow filtering for specific age ranges and occupations, then organizing the output by name or age.

Portfolio Assignment: DS

Requirements:

- Utilize Python, demonstrating the use of lists, sets, and dictionaries to manage and process data.
- Implement functions to filter data based on criteria and to sort the filtered data.
- Showcase the ability to read from and write to files, enabling the script to process input data and output results.
- Explain the use of modular arithmetic or greatest common divisors if applicable to data organization.
- Accompany the script with a README file detailing how to use the tool, the types of data it can process, and a simple example with a sample dataset.

Portfolio Assignment: WD

Interactive Number Theory Playground

Objective: Create a web-based interactive tool that allows users to explore various number theory concepts (prime numbers, greatest common divisors, modular arithmetic) and set operations (union, intersection, difference, complement). This tool should enable users to input numbers and select the operation or concept they wish to explore, displaying the results on the webpage.

Portfolio Assignment: WD

Requirements:

- Use HTML/CSS to design a user-friendly interface for inputting values and selecting operations.
- Implement JavaScript to handle user inputs, perform the operations, and display the results dynamically on the page.
- Demonstrate type conversion and variable assignment.
- Include explanations or tooltips that briefly describe each number theory concept or set operation for educational purposes.
- Provide a README file with instructions on how to setup and use the web tool, along with an overview of the implemented features.

Summary

Variables

- Represent unspecified values in algebraic expressions and equations.

Sets

- Store a collection of distinct objects known as elements.

Functions

- Relations between a set of inputs and a set of permissible outputs.
- Each input of a function is related to at most one output.

Summary

Asymptotes

- A line that a curve approaches as it heads towards infinity.

Complexity Analysis

- A 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.
- Measured by Big O Notation, which measures the worst-case complexity and allows comparison of algorithms.

CoGrammar

Q & A SECTION

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

Thank you for attending



Department
for Education

CoGrammar

