Welcome to this CoGrammar Lecture: Searching and Sorting

The session will start shortly...

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

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Searching and Sorting

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**

CoGrammar

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:

  www.hyperiondev.com/support

- We would love your **feedback** on lectures: **Feedback on Lectures**

Safeguarding & Welfare

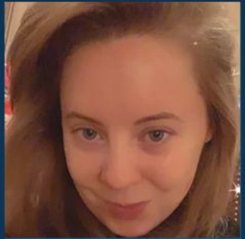
We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

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

CoGrammar HyperionDev

Skills Bootcamp Progression Overview

To be eligible for a certificate of completion, students must fulfil three specific criteria. These criteria ensure a high standard of achievement and alignment with the requirements for the successful completion of a Skills Bootcamp.

✅ Criterion 1 - Meeting Initial Requirements

Criterion 1 involves specific achievements within the first two weeks of the program. To meet this criterion, students need to:

- Attend a minimum of 7-8 hours per week of guided learning (lectures, workshops, or mentor calls) within the initial two-week period, for a total minimum of 15 guided learning hours (GLH), by no later than 15 September 2024.
- Successfully complete the Initial Assessment by the end of the first 14 days, by no later than 15 September 2024.

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 2 - Demonstrating Mid-Course Progress

Criterion 2 involves demonstrating meaningful progress through the successful completion of tasks within the first half of the bootcamp.
To meet this criterion, students should:

- Complete 42 guided learning hours and the first half of the assigned tasks by the end of week 7, no later than 20 October 2024.

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3 - Demonstrating Post-Course Progress

Criterion 3 involves showcasing students' progress after completing the course. To meet this criterion, students should:

- Complete all mandatory tasks before the bootcamp's end date. This includes any necessary resubmissions, no later than 22 December 2024.
- Achieve at least 84 guided learning hours by the end of the bootcamp, 22 December 2024.

CoGrammar

Poll

In Python, which data structure is commonly used to implement a Last In, First Out (LIFO) ordering?

A. Queue

B. Linked list

C. Stack

D. array

CoGrammar

Poll

What is an algorithm?

a. A type of programming language

b. A step-by-step procedure for solving a problem

c. A hardware component in a computer

d. A type of data structure

Learning Objectives & Outcomes

- Define what an algorithm is.
- Define order of complexity and determine the complexity order of different algorithms.
- Recognise and implement common searching and sorting algorithms.

CoGrammar

Algorithms

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.

CoGrammar

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.

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.

CoGrammar

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.

CoGrammar

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

CoGrammar

Time Complexities

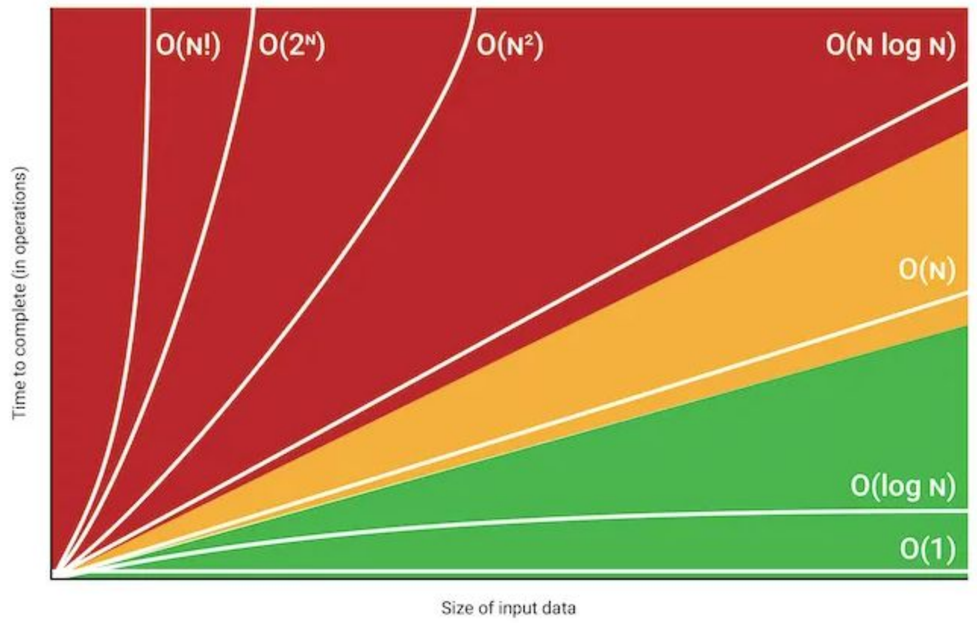


Image source: https://pub.towardsai.net/big-o-notation-what-is-it-69cfd9d5f6b8

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

CoGrammar

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

CoGrammar

Sorting

CoGrammar

Bubble sort

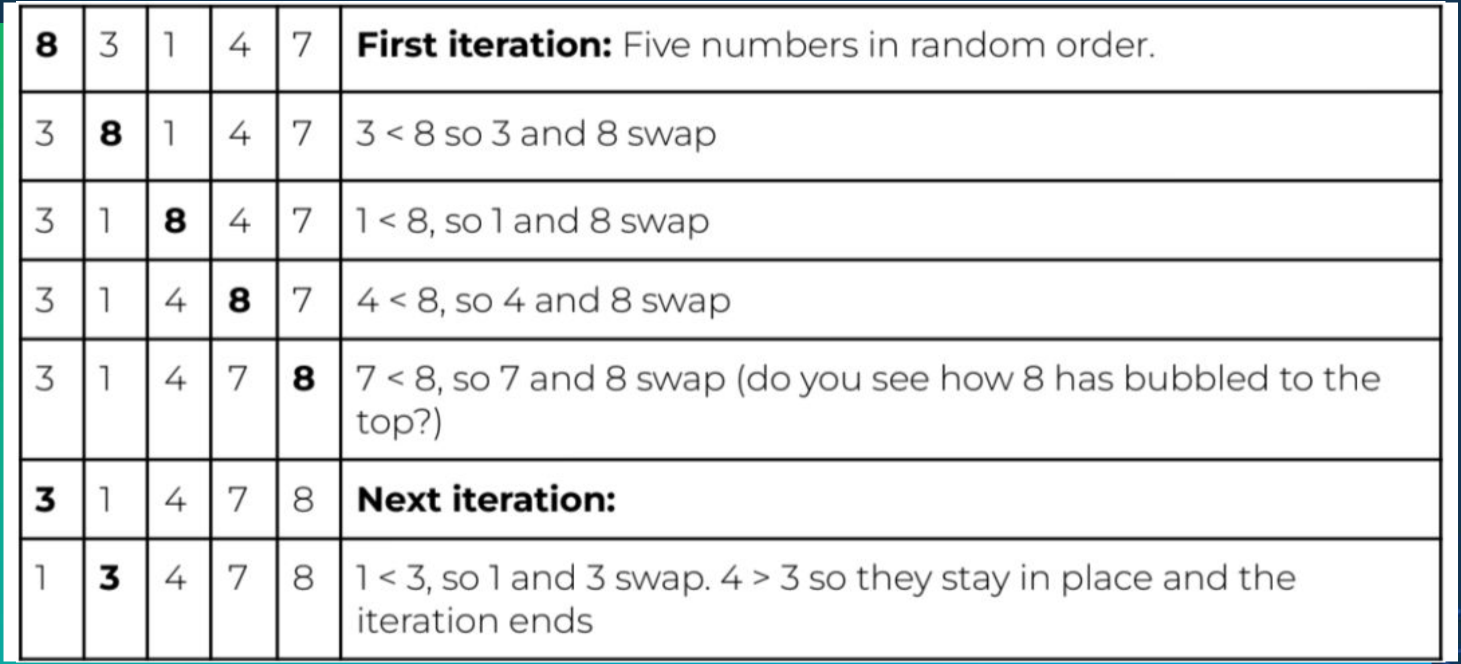
- Larger values tend to bubble up to the top of the list
- Compare an item to the item next to it
- If the first value is larger than the second value swap places
- This process repeats until all values are compare and starts the process again
- This will run for as many times as 1 less than the length of the list to ensure enough passes were made
- Time complexity is **$O(n^2)$**

CoGrammar

Bubble sort

```
function bubble_sort(array):
    length = len(array)
    swapped = True
    while swapped:
        swapped = False
        for i = 0 to length - 1:
            if array[i] > array[i + 1]:
                swap array[i] and array[i + 1]
                swapped = True
    return array
```

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

CoGrammar

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.
- Time complexity is **$O(n^2)$**

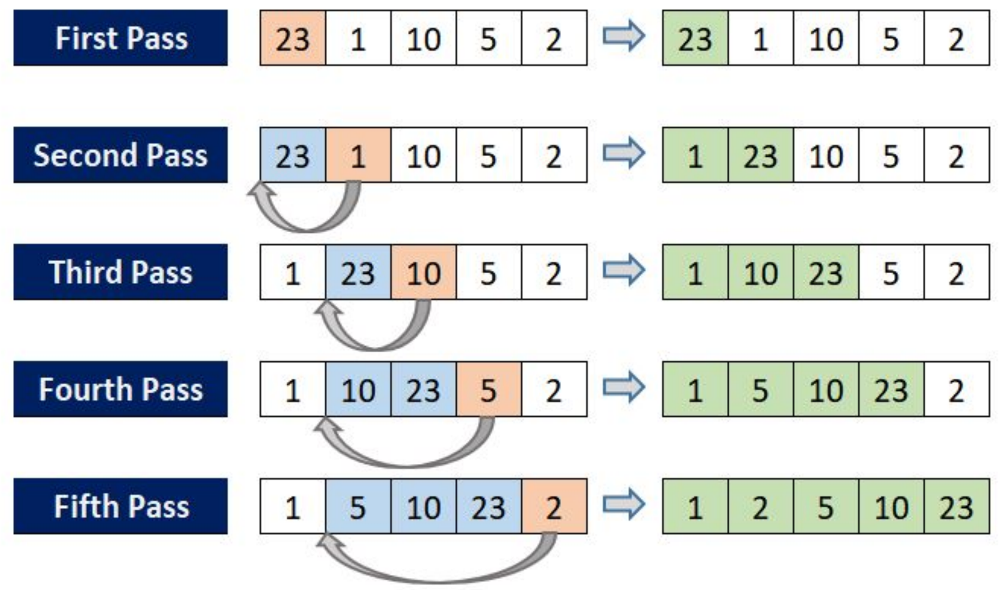
CoGrammar

Insertion Sort

```
function insertion_sort(array):
    i = 1
    length = len(array)
    while i < length:
        j = i
        while j > 0 and array[j - 1] > array[j]:
            swap array[j - 1] and array[j]
            j = j - 1
        i = i + 1
    return array
```

CoGrammar

Insertion Sort



CoGrammar

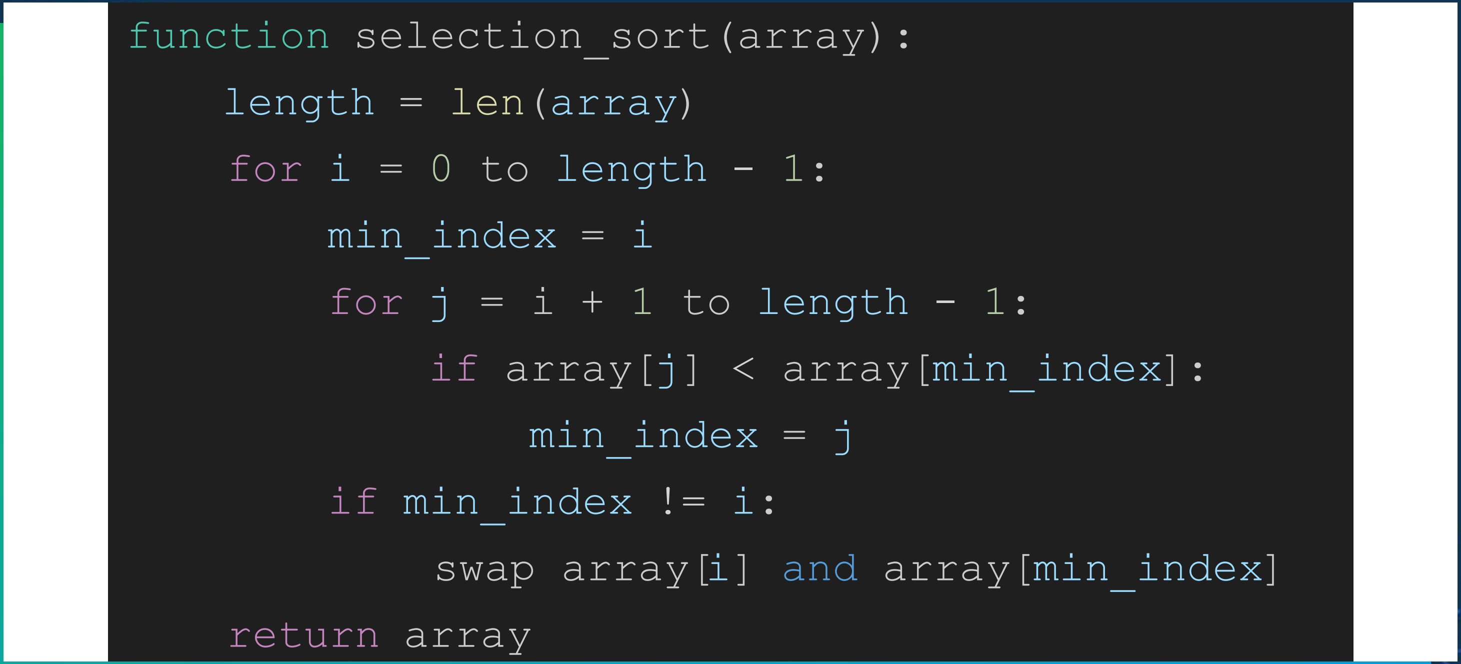
Source

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.
- Time complexity is **O(n^2)**

CoGrammar

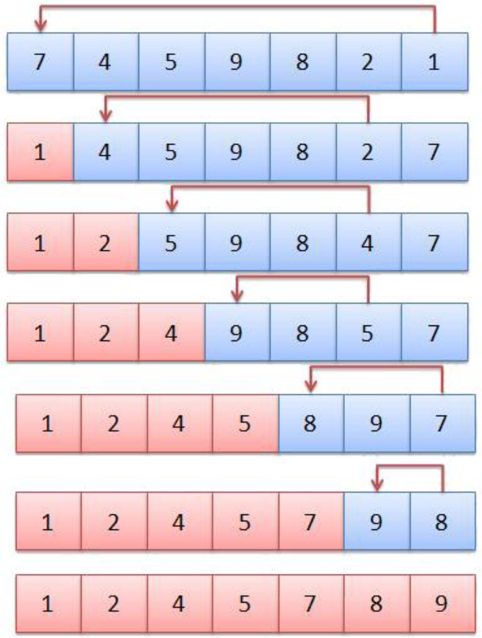
Selection Sort



```
function selection_sort(array):
    length = len(array)
    for i = 0 to length - 1:
        min_index = i
        for j = i + 1 to length - 1:
            if array[j] < array[min_index]:
                min_index = j
        if min_index != i:
            swap array[i] and array[min_index]
    return array
```

CoGrammar

Selection Sort



CoGrammar

Source

Let’s take a short break

CoGrammar

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

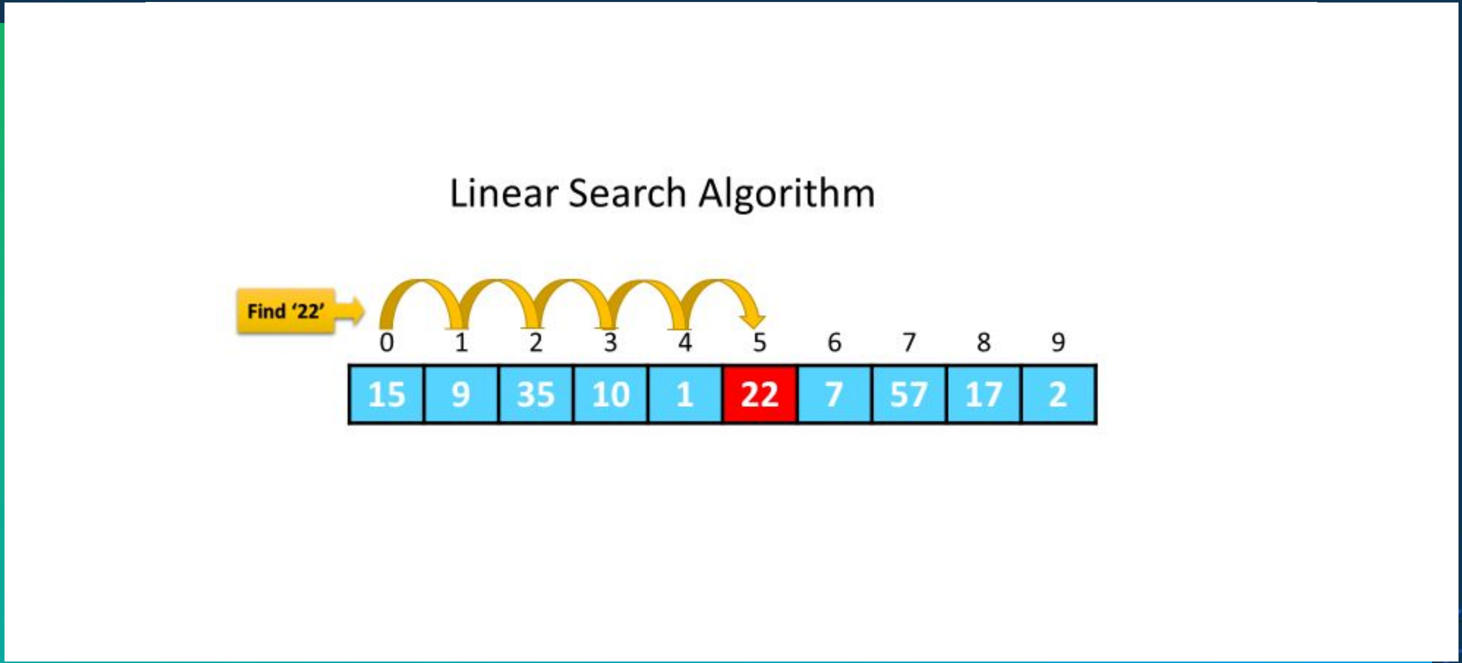
Linear Search

- 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
```

Linear Search



CoGrammar

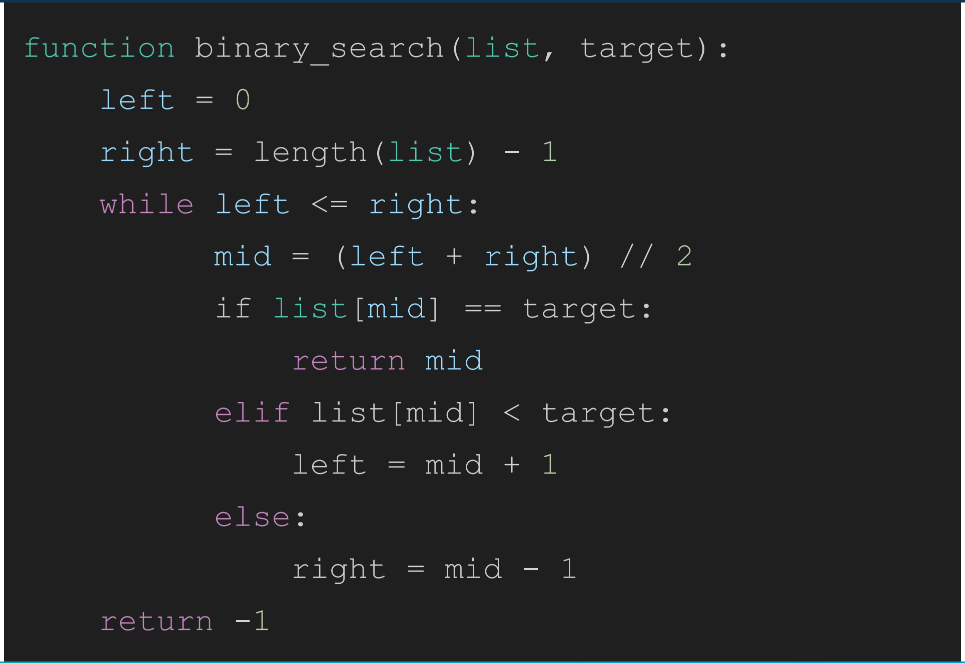
Source

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. This with a complexity of **O(log n)**

CoGrammar

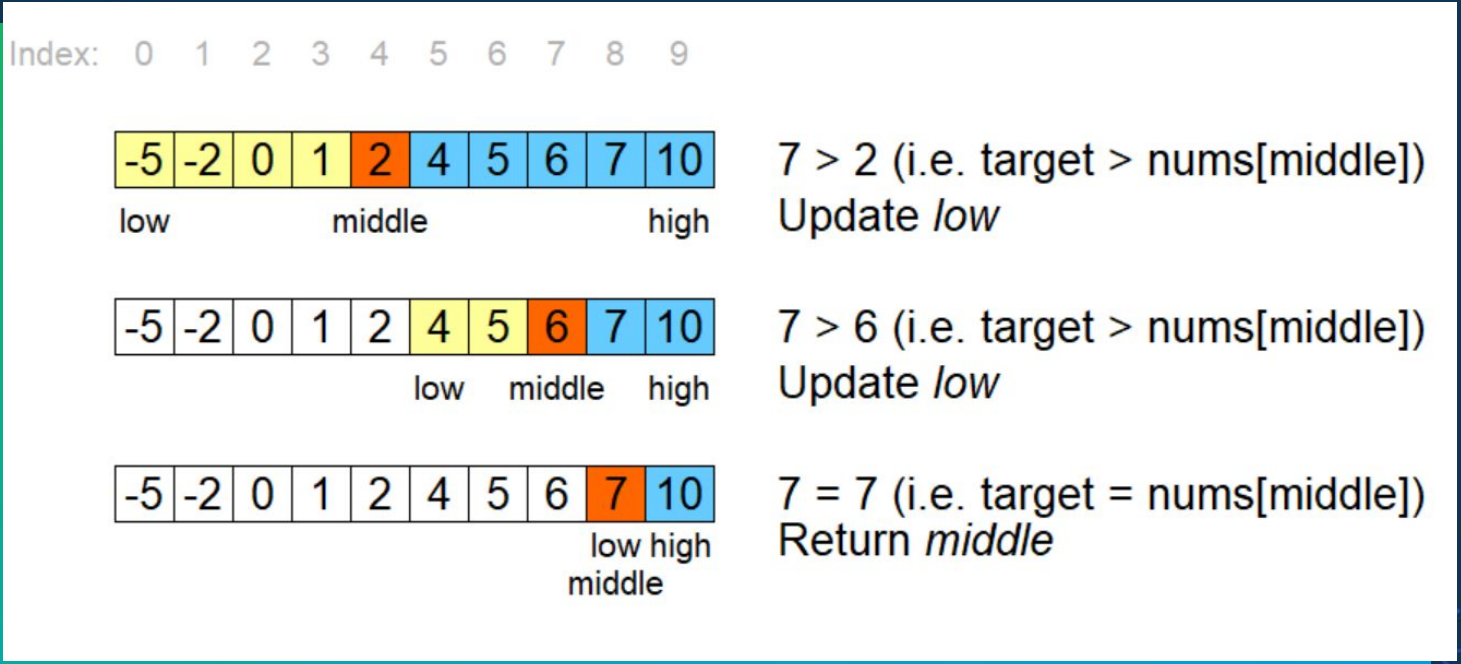
Binary Search



```
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
```

CoGrammar

Binary Search



7 > 2 (i.e. target > nums[middle])
Update *low*

7 > 6 (i.e. target > nums[middle])
Update *low*

7 = 7 (i.e. target = nums[middle])
Return *middle*

CoGrammar

Source

Questions and Answers

CoGrammar

Poll

What is the time complexity of the linear search algorithm?

A. $O(1)$

B. $O(n)$

C. $O(n^2)$

D. $O(\log(n))$

CoGrammar

Poll

Which sorting algorithm builds the final sorted array one element at a time by repeatedly selecting the smallest element from the unsorted part?

a. Bubble sort

b. Insertion sort

c. Selection sort

Summary

CoGrammar

Summary

- **Algorithms**
  - Set of instructions that can solve a problem, like searching and sorting.
- **Complexity Order**
  - We can determine how a algorithm will scale with the input by calculating the complexity order of an algorithm.
- **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, insertion and Selection Sort alongside linear and binary search.

Resources

- VisualGo
- Yongdanielliang
- Usfca
- Open Data Structures
- Data Structure Visualisations
- CS 1332 Data Structures and Algorithms Visualisation Tool

CoGrammar

Let’s get coding!

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar