




Welcome to the **Co**Grammar Datasets and DataFrames

The session will start shortly...

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



Data Science 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](#)

Data Science 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](#)

CoGrammar

Divide and Conquer Algorithms

April 2024




What is your familiarity with the concept of a divide and conquer algorithm?

1. A. Very familiar
2. B. Somewhat familiar
3. C. Not familiar at all





What is an algorithm?

1. A step-by-step procedure for solving a problem or accomplishing a task.
 2. A set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
 3. A sequence of instructions that performs a specific task or calculation.
 4. A method or process for solving a problem efficiently.
 5. All of the above
- 



Which of the following are divide and conquer algorithms??

1. Merge Sort
2. Linear Search
3. Quick Sort
4. Binary Search
5. Bubble Sort
6. All of the above

Learning objectives

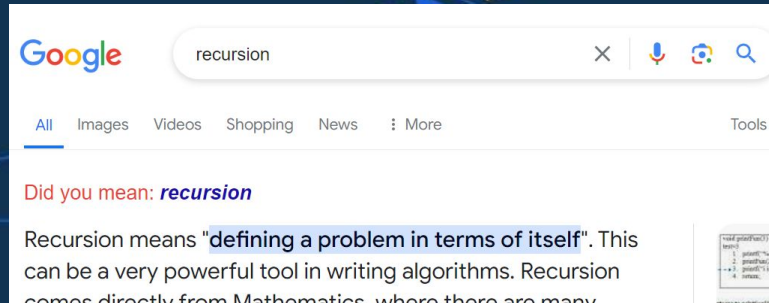
- ❖ Understand the divide and conquer paradigm and its relevance in problem-solving.
- ❖ Understand the Benefits of divide and conquer algorithms
- ❖ Confidently and accurately describe how to apply a divide and conquer algorithm to interview-type questions.

Divide and Conquer Algorithms



Divide and Conquer Algorithms

- ❖ **Algorithms** - a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.
- ❖ **Divide and Conquer** :A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.



Why Use Divide and Conquer Algorithms?

Resource Constraints

- ❖ **Real-Time Applications:** These often face limitations in processing power and memory.

Time Complexity

- ❖ **Linear Search:** The time taken to find information increases proportionally with the dataset size.
- ❖ **Potential Delays:** This can lead to unacceptable delays in real-time applications.

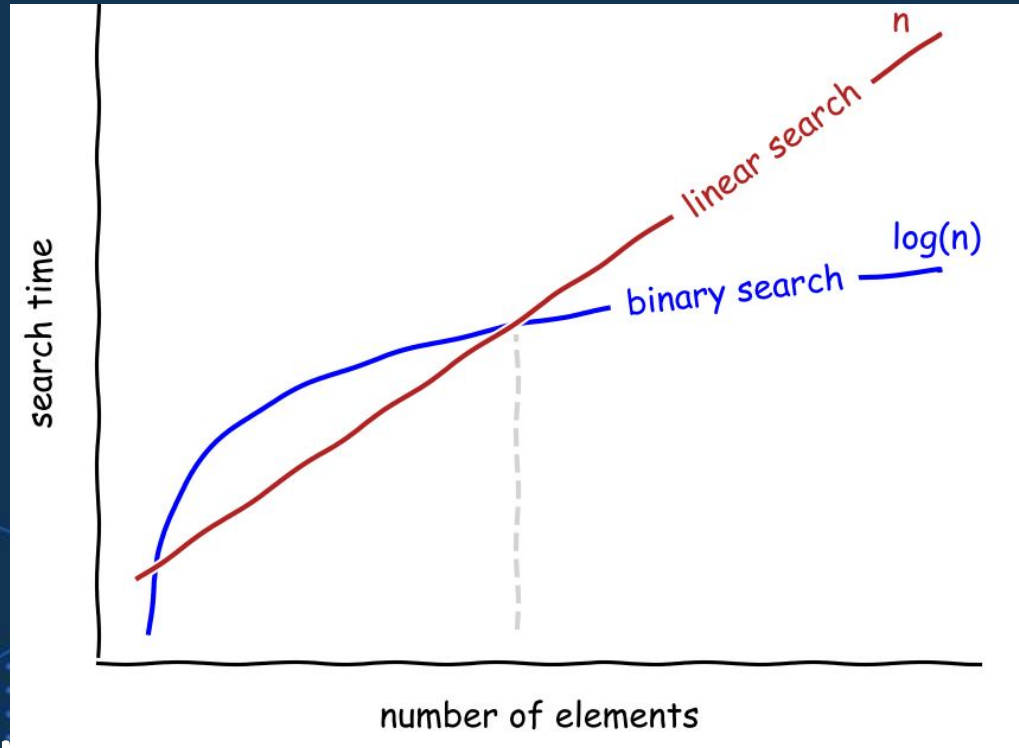
Comparing Approaches

- ❖ **Linear/Iterative Approach:** Slower as the dataset grows.
- ❖ **Divide and Conquer Algorithm:** Faster and more efficient, breaking down the problem and solving it in smaller, manageable parts.

Divide and Conquer Methodology

1. **Divide** – Break the problem into smaller, more manageable subproblems. These subproblems should be of the same type as the original problem.
2. **Conquer** - Solve the subproblems independently. If the subproblems are still too large, recursively apply the divide and conquer method to them.
3. **Combine** - Integrate the solutions of the subproblems to form the solution to the original problem.

Binary Search Approach vs Linear Search



Source: Binary Search variant

Linear Search



=
33

Source : [Search Visualisations](#)

Linear search is straightforward but becomes **inefficient** with long arrays, as it checks each element one by one. For larger datasets, this method's **time complexity of $O(n)$** leads to **significant performance** issues, highlighting the need for more efficient algorithms like binary search or hash tables to improve search speed and efficiency.

Divide and Conquer Algorithms

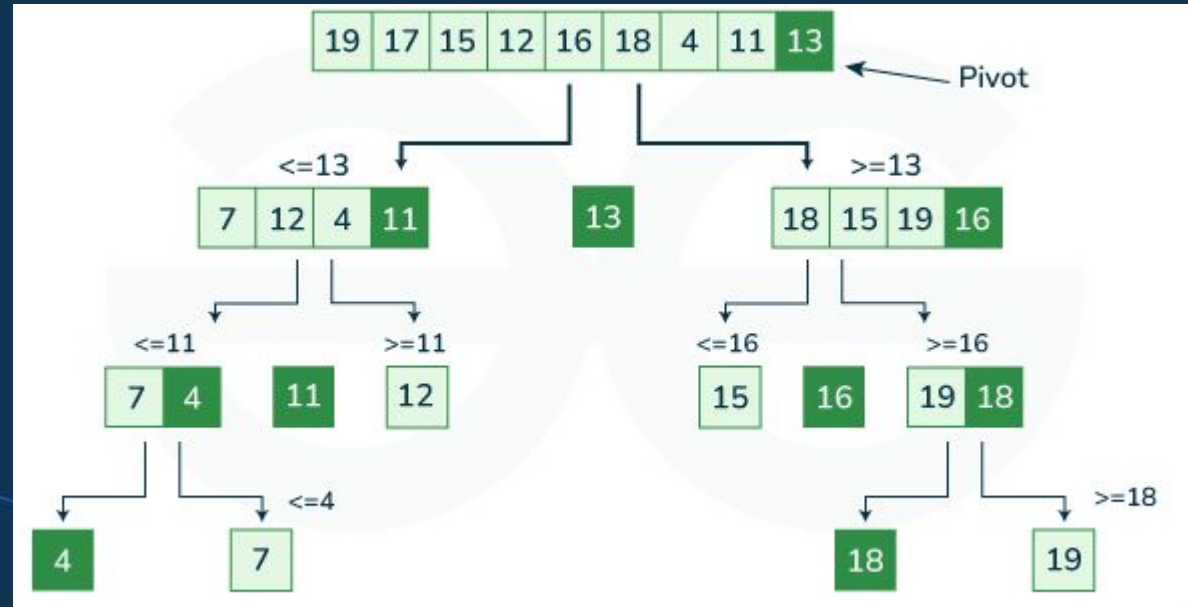
Quick Sort



Quick Search

Quick Sort: Quicksort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, one with elements that are less than a chosen pivot and another with elements that are greater.

Source: [Geeks](#)



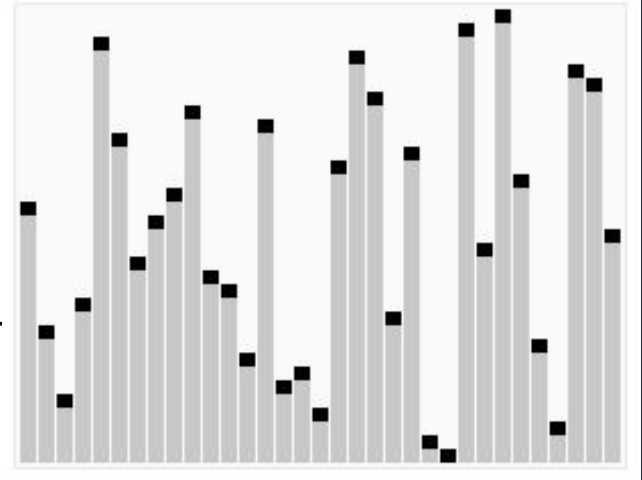
How Quick Sort Works

Partitioning

1. Select a pivot from the list.
2. Move elements:
 3. Less than the pivot to the left.
 4. Greater than or equal to the pivot to the right

Recursion

5. Apply the process to the left sub-array.
6. Apply the process to the right sub-array.



Source : [Wikipedia](https://en.wikipedia.org/wiki/Quick_sort)

Divide and Conquer Algorithms

Binary Search



Binary Search

Binary search : Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

Using binary search,
we will search for the
position of the value
K = 56

Source: [Binary search](#)

We use formula to calculate mid of array $\text{mid} = (\text{beg} + \text{end})/2$
Where $\text{beg} = 0$, $\text{end} = 8$ therefore $\text{mid} = (0+8)/2 = 4$

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

↑

$A[mid] = 39$
 $A[mid] < K$ (or, $39 < 56$)
 So, $beg = mid + 1 = 5$, $end = 8$
 Now, $mid = (beg + end) / 2 = 13 / 2 = 6$

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

↑

$A[mid] = 51$
 $A[mid] < K$ (or, $51 < 56$)
 So, $beg = mid + 1 = 7$, $end = 8$
 Now, $mid = (beg + end) / 2 = 15 / 2 = 7$

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

↑

$A[mid] = 56$
 $A[mid] = K$ (or, $56 = 56$)
 So, $location = mid$
 Element found at 7th location of the array

Problem Statement

Scenario 1: **Book Sorting**

You're working at a bookstore where books are delivered out of order. Your task is to quickly organise the books on the shelves in alphabetical order by title. You need an efficient sorting algorithm to accomplish this task, ensuring that customers can easily find the books they're looking for.

Scenario 2: **Book Search System**

After sorting the books in the bookstore, you're tasked with developing a search system for customers to find books quickly. The system should allow users to enter a book title, and it should efficiently search through the sorted list of books to find the requested title. You need to implement a search algorithm that can handle this task efficiently to provide a smooth experience for bookstore customers

The Bookstore Dataset

```
books = [  
    "To Kill a Mockingbird", "1984", "The Great Gatsby", "The Catcher in the Rye",  
    "Moby-Dick", "War and Peace", "Pride and Prejudice", "The Lord of the Rings",  
    "The Hobbit", "Jane Eyre", "Wuthering Heights", "Brave New World",  
    "The Odyssey", "Crime and Punishment", "The Brothers Karamazov",  
    "Anna Karenina", "Madame Bovary", "The Adventures of Huckleberry Finn",  
    "Alice's Adventures in Wonderland", "Don Quixote", "The Divine Comedy",  
    "Ulysses", "One Hundred Years of Solitude", "The Iliad", "The Aeneid",  
    "The Canterbury Tales", "Les Misérables", "Great Expectations", "Dracula",  
    "Frankenstein", "The Count of Monte Cristo", "The Scarlet Letter",  
    "Heart of Darkness", "Catch-22", "Lord of the Flies", "The Sound and the Fury",  
    "Gone with the Wind", "Slaughterhouse-Five", "The Grapes of Wrath",  
    "A Tale of Two Cities", "The Old Man and the Sea", "The Stranger",  
    "The Sun Also Rises", "The Metamorphosis", "The Picture of Dorian Gray",  
    "A Clockwork Orange", "The Bell Jar", "Beloved", "Invisible Man", "Fahrenheit 451",  
    "Things Fall Apart", "No Longer at Ease", "Arrow of God", "A Man of the People",  
    "Anthills of the Savannah", "When Rain Clouds Gather", "Maru", "A Question of Power",  
    "Petals of Blood", "Weep Not, Child", "The River Between", "A Grain of Wheat",  
    "Season of Migration to the North", "The Wedding of Zein", "Half of a Yellow Sun",  
    "Purple Hibiscus", "Americanah", "So Long a Letter", "The Dark Child",  
    "Chaka", "The Joys of Motherhood", "Nervous Conditions", "The Book of Not",  
    "The Hairdresser of Harare", "The Beautiful Ones Are Not Yet Born",  
    "Waiting for the Barbarians", "Disgrace", "Cry, the Beloved Country",  
    "Mine Boy", "Burger's Daughter", "July's People"  
]
```


Quick sort Pseudo code

1. **Base Case:**

If the array has 0 or 1 elements, it is already sorted. Return the array as is.

2. **Choose Pivot:**

Select the middle element of the array as the pivot.

3. **Partitioning:**

Create three sub-arrays:

left: elements less than the pivot.

middle: elements equal to the pivot.

right: elements greater than the pivot.

4. **Recursion:**

Recursively apply Quick Sort to the left and right sub-arrays.

Concatenate the sorted left array, middle array, and sorted right array to get the final sorted array.

Quick sort Pseudo code

1. Initialize Pointers:

Set left to the beginning of the array (0).

Set right to the end of the array (length of array - 1).

2. Loop Until Pointers Cross:

Continue the loop while left is less than or equal to right.

3. Calculate Middle Index:

Compute mid as the average of left and right.

4. Check Middle Element:

If array[mid] is equal to target, return mid (target found).

If array[mid] is less than target, move left to mid + 1 (ignore left half).

If array[mid] is greater than target, move right to mid - 1 (ignore right half).

5. Return -1 if Not Found:

If the loop exits without finding the target, return -1 (target not found).

Code Implementation



Quick Search Code

```
32 def binary_search(book_list, target):
33     # Initialize left and right pointers to the beginning and end of the list
34     left, right = 0, len(book_list) - 1
35
36     # Continue searching as long as left pointer does not exceed right pointer
37     while left <= right:
38         # Calculate the middle index of the current segment
39         mid = (left + right) // 2
40
41         # Check if the middle element is the target
42         if book_list[mid] == target:
43             return mid # Target found at index mid
44
45         # If target is greater than the middle element, search in the right half
46         elif book_list[mid] < target:
47             left = mid + 1 # Update left pointer to mid + 1
48
49         # If target is less than the middle element, search in the left half
50         else:
51             right = mid - 1 # Update right pointer to mid - 1
52
53     # If target is not found, return -1
54     return -1
```

Binary Search Code

```
32 def binary_search(book_list, target):
33     # Initialize left and right pointers to the beginning and end of the list
34     left, right = 0, len(book_list) - 1
35
36     # Continue searching as long as left pointer does not exceed right pointer
37     while left <= right:
38         # Calculate the middle index of the current segment
39         mid = (left + right) // 2
40
41         # Check if the middle element is the target
42         if book_list[mid] == target:
43             return mid # Target found at index mid
44
45         # If target is greater than the middle element, search in the right half
46         elif book_list[mid] < target:
47             left = mid + 1 # Update left pointer to mid + 1
48
49         # If target is less than the middle element, search in the left half
50         else:
51             right = mid - 1 # Update right pointer to mid - 1
52
53     # If target is not found, return -1
54     return -1
```

How confident are you in explaining and implementing Binary Search during an interview?

1. Very confident: I can explain and implement Binary Search without any issues.
2. Confident: I understand Binary Search well but might need to brush up before an interview.
3. Neutral: I have a basic understanding of Binary Search but need more practice.
4. Not very confident: I find Binary Search somewhat challenging and need additional practice.
5. Not confident at all: I do not feel prepared to explain or implement Binary Search in an interview setting.

Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

