*QON:1* Implement Binary Search and Linear Search ?

*Binary Search*

*binary\_search(arr, target):*

*low, high = 0, 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*

*# Example usage:*

*arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]*

*target = 5*

*result = binary\_search(arr, target)*

*if result != -1:*

*print(f"Target {target} found at index {result}")*

*else:*

*print(f"Target {target} not found")*

*Linear Search*

*def linear\_search(arr, target):*

*for i, num in enumerate(arr):*

*if num == target:*

*return i*

*return -1*

*# Example usage:*

*arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]*

*target = 5*

*result = linear\_search(arr, target)*

*if result != -1:*

*print(f"Target {target} found at index {result}")*

*else:*

*print(f"Target {target} not found")*

QON 2: How to run the program ?

ANS: To run a program, you'll need to provide more context about the program itself, such as the programming language it's written in and the environment it's intended to run in. Are you trying to run a program on a specific device or platform

To run a program on Windows, you'll typically need to compile or interpret the code, depending on the programming language. Do you have the program's source code, or is it a pre-compiled executable file

QON3: The purpose of the code ?

ANS: In general, the purpose of programming code is to instruct a computer to perform a specific task or set of tasks. This can include things like processing data, performing calculations, controlling hardware, or providing a user interface. What kind of task are you trying to accomplish with code

For example:

As a business assistant, I can suggest some business-related project ideas. How about building a simple chatbot to help with customer service, or creating a program to help manage inventory or track sales

QSN4: The time complexity, also tell the worst case of the binary search O(n), and print that numbers ?

ANS:

The time complexity of binary search is actually O(log n), not O(n).

To clarify, 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.

The worst-case scenario for binary search is when the item is not in the list, and you have to search the entire list. Even in this case, the time complexity is still O(log n), because you're dividing the search space in half with each step.

Example of binary search works:

list of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9]

If we're searching for the number 5, the binary search algorithm would work as follows:

1. Start with the middle element of the list: 5

2. Compare the target number (5) to the middle element (5). They're equal, so we're done!

If we were searching for a number that's not in the list, the algorithm would work as follows:

1. Start with the middle element of the list: 5

2. Compare the target number (e.g. 10) to the middle element (5). The target number is greater, so we know it's in the second half of the list.

3. Repeat the process with the second half of the list: [6, 7, 8, 9]

4. Compare the target number (10) to the middle element (7). The target number is greater, so we know it's in the second half of the list.

5. Repeat the process with the second half of the list: [8, 9]

6. Compare the target number (10) to the middle element (8). The target number is greater, so we know it's not in the list.

As you can see, even in the worst-case scenario, the binary search algorithm only needs to perform a logarithmic number of comparisons to find the target number (or determine that it's not in the list).

Here's some sample Python code to illustrate the binary search algorithm:

def binary\_search(arr, target):

low, high = 0, 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 # not found

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]

target = 5

result = binary\_search(arr, target)

print(result) # output: 4