Algorithm: Binary Search

Binary Search

Index is returned

Key Aspects:

- Very efficient search algorithm.
- Also called "Half-interval Search".
- The list or tuple has to be sorted in ascending order for the algorithm to work correctly.
- Reduces the search space by half on every iteration.

Algorithm:

- Start with an interval that covers the whole list.
- Find the middle index.
- Check if the item in the middle of the list is the one that you are looking for.
- If it is, return the index.
- If it isn't, compare the element in the middle of the list with the element that you are looking for.
    - If it's larger, assign the middle_index - 1 as the upper bound to discard the upper half of the list.
    - If it's smaller, assign the middle_index + 1 as the lower bound to discard the lower half of the list.
- Repeat the process until the item is found or until the interval not valid.
- If the item is not found, return -1.

Time Complexity:

- Worst-Case Time Complexity: O(log(n)) (Logarithmic).
- Average-Case Time Complexity: O(log(n)) (Logarithmic).
- Best-Case Time Complexity: O(1) (Constant).

Binary Search

Code (Iterative):

```python
# The Binary Search Algorithm takes:
# data - A list or tuple.
# item - The item that you wish to find in the list (data).
def binary_search(data, item):

    # Set the initial bounds for the interval:
    # The lower bound is the first index in the list.
    # The upper bound is the last index in the list.
    low = 0
    high = len(data) - 1

    # While the interval is valid
    while low <= high:
        # Find the item in the middle of the interval
        middle = (low + high)//2
        # If that item is equal to the target item,
        # return the index.
        if data[middle] == item:
            return middle
        # If the item is not equal to the target item,
        # check if it's greater or smaller and reassign
        # the bounds appropriately.
        elif data[middle] > item:
            high = middle - 1
        else:
            low = middle + 1

    # Else, if the item is not found in the list, return -1
    return -1
```

Binary Search

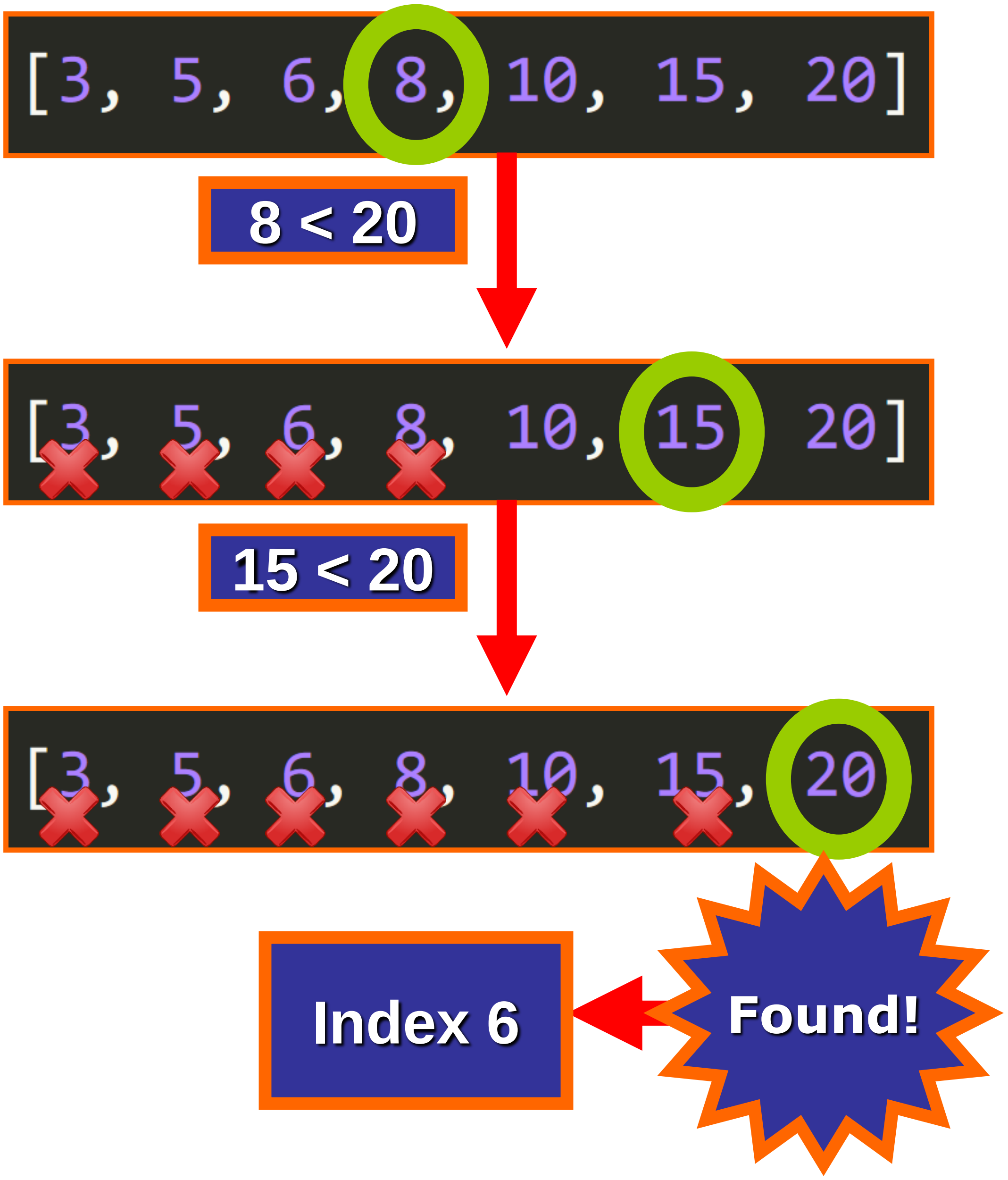
Code (Recursive):

```python
# This recursive version of the Binary Search
# takes four parameters:
# data - the list or tuple
# low  - the lower bound
# high - the upper bound
# item - the item that you are looking for
def binary_search(data, low, high, item):
    # If the interval is valid
    if low <= high:
        # Find the middle of the list (index)
        middle = (low + high)//2
        # If the item in the middle of the list
        # is the one that you are looking for,
        # return the index.
        if data[middle] == item:
            return middle
        # Else, if it's greater than the item that you
        # are looking for, make a recursive call passing
        # the middle index - 1 as upper bound to discard the
        # upper half of the list
        elif data[middle] > item:
            return binary_search(data, low, middle - 1, item)
        # Else, it it's smaller than the item that you
        # are looking for, make a recursive call passing
        # the middle index + 1 as lower bound to discard the
        # lower half of the list.
        else:
            return binary_search(data, middle + 1, high, item)
    # If the item is not found, return -1
    else:
        return -1
```

Binary Search

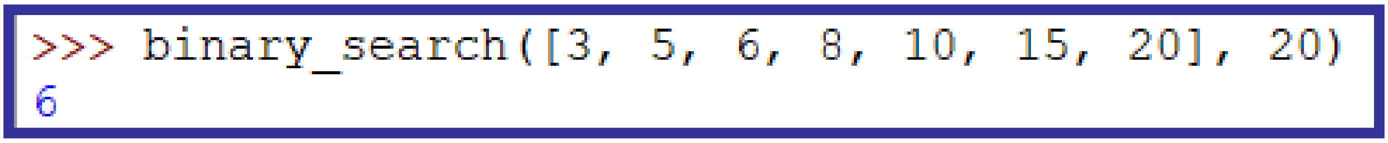
Example:

```
>>> binary_search([3, 5, 6, 8, 10, 15, 20], 20)
6
```

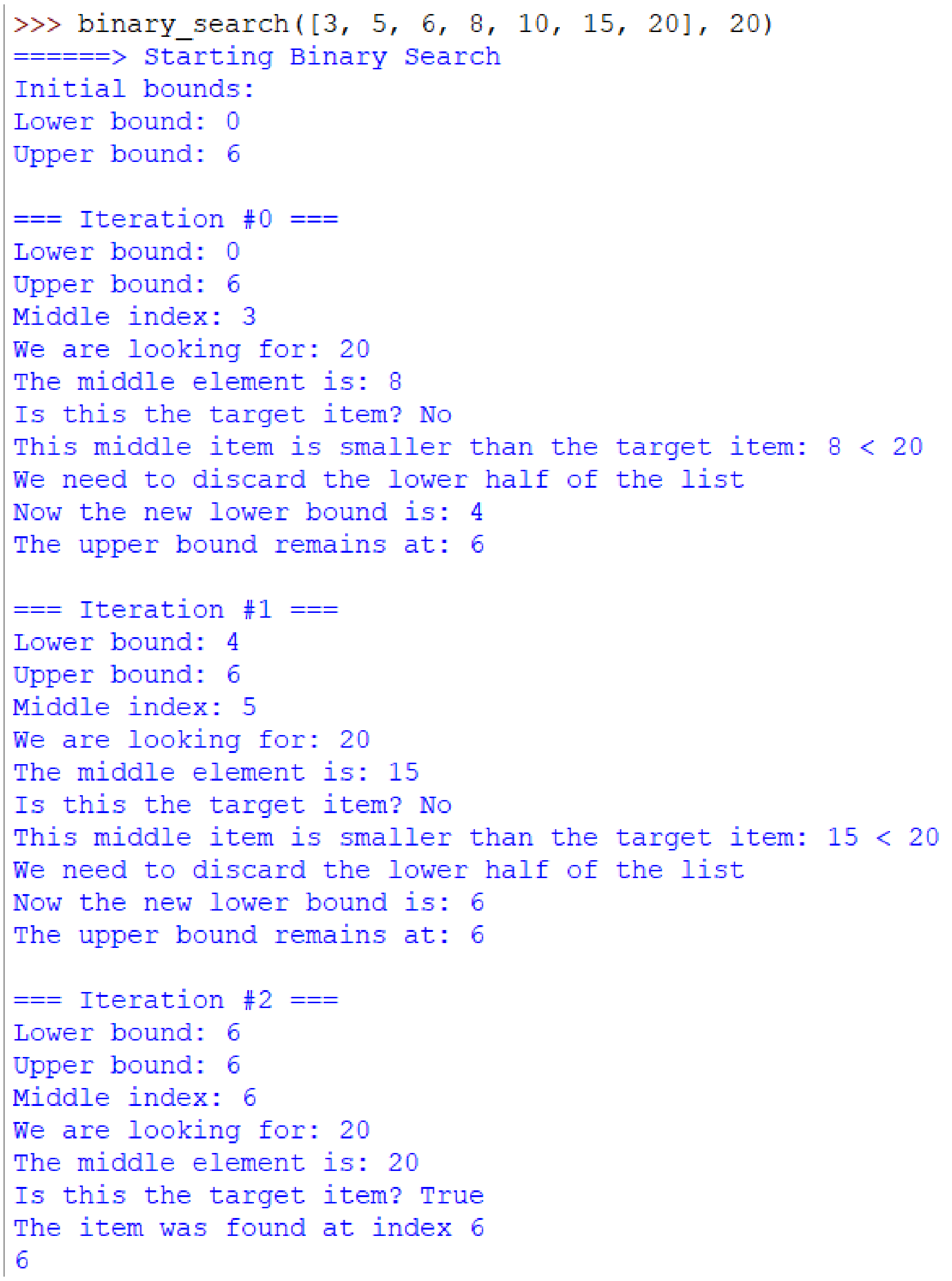


Binary Search

Example:



```
>>> binary_search([3, 5, 6, 8, 10, 15, 20], 20)
6
```



```
>>> binary_search([3, 5, 6, 8, 10, 15, 20], 20)
======> Starting Binary Search
Initial bounds:
Lower bound:  0
Upper bound:  6

=== Iteration #0 ===
Lower bound:  0
Upper bound:  6
Middle index:  3
We are looking for: 20
The middle element is: 8
Is this the target item? No
This middle item is smaller than the target item: 8 < 20
We need to discard the lower half of the list
Now the new lower bound is: 4
The upper bound remains at: 6

=== Iteration #1 ===
Lower bound:  4
Upper bound:  6
Middle index:  5
We are looking for: 20
The middle element is: 15
Is this the target item? No
This middle item is smaller than the target item: 15 < 20
We need to discard the lower half of the list
Now the new lower bound is: 6
The upper bound remains at: 6

=== Iteration #2 ===
Lower bound:  6
Upper bound:  6
Middle index:  6
We are looking for: 20
The middle element is: 20
Is this the target item? True
The item was found at index 6
6
```