Find Fixed Number

In this lesson, you will learn how to find a fixed number in a list using a binary search in Python.

We'll cover the following

In this lesson, we will be solving the following problem:

Given an array of n distinct integers sorted in ascending order, write a function that returns a **fixed point** in the array. If there is not a fixed point, return `None`.

> A fixed point in an array `A` is an index `i` such that `A[i]` is equal to `i`.

The naive approach to solving this problem is pretty simple. You iterate through the list and check if each element matches its index. If you find a match, you return that element. Otherwise, you return `None` if you don't find a match by the end of the `for` loop. Have a look at the code below:

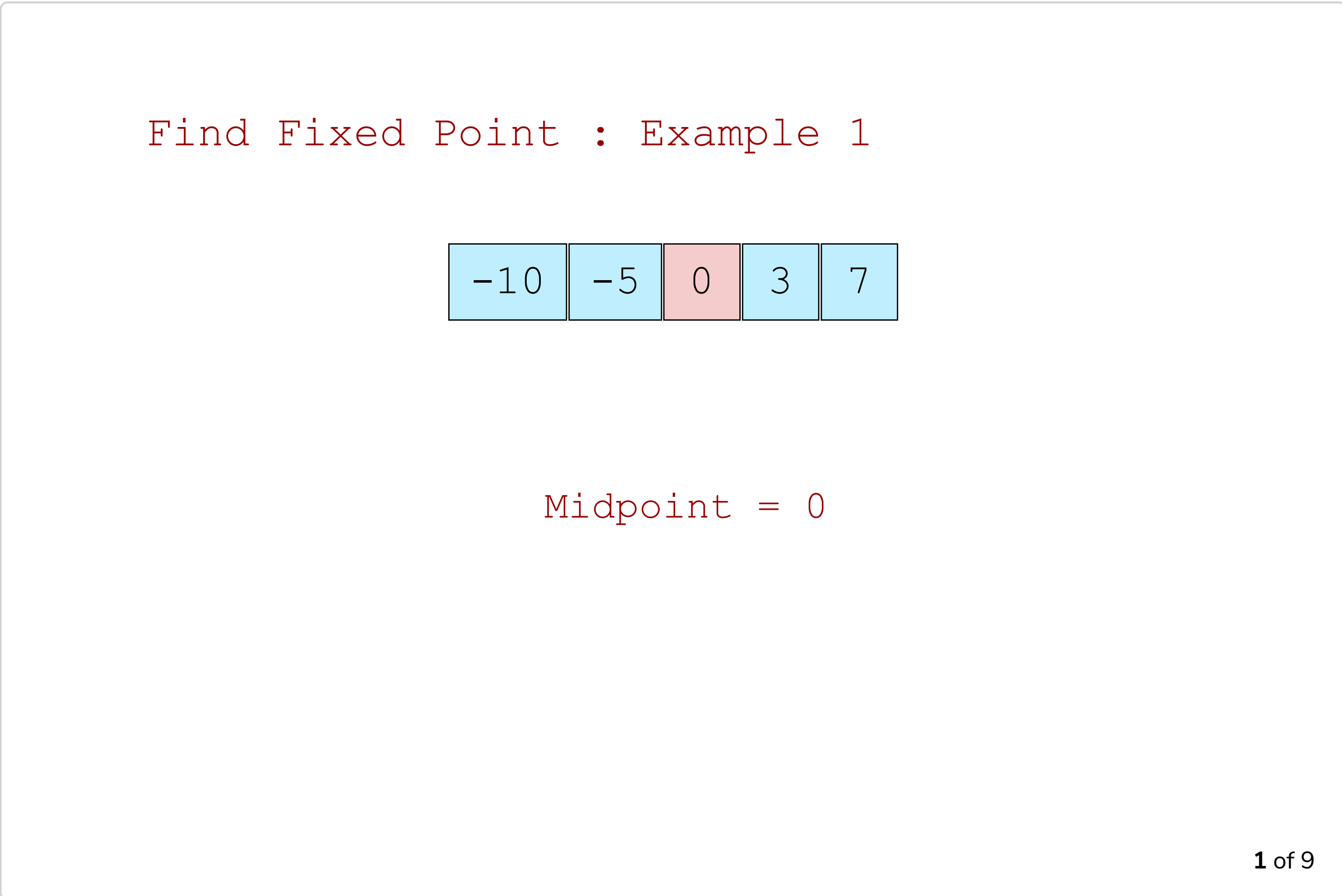
```python
# Time Complexity: O(n)
# Space Complexity: O(1)
def find_fixed_point_linear(A):
    for i in range(len(A)):
        if A[i] == i:
            return A[i]
    return None
```

find_fixed_point_linear(A)

As the entire list is traversed once to find the fixed point, spending constant time on each element, the time complexity for the linear implementation above is $O(n)$. As we haven't used any additional space in the implementation above, the space complexity is $O(1)$. Now we need to think about how we can improve the solution above. We can use the following two facts to our advantage:

- The list is sorted.
- The list contains *distinct* elements.

Let's look at the slides below to get a rough idea of how we have taken advantage of the above facts.



Implementation

If you have gone through the slides, the implementation must be pretty clear to you. Let's jump to the implementation in Python:

```
# Time Complexity: O(log n)
# Space Complexity: O(1)
def find_fixed_point(A):
    low = 0
    high = len(A) - 1
```

```
    while low <= high:
        mid = (low + high)//2


        if A[mid] < mid:
            low = mid + 1
        elif A[mid] > mid:
            high = mid - 1
        else:
            return A[mid]
    return None
```

find_fixed_point(A)

Explanation

On **lines 4-5**, we define `low` and `high` in the same way we have always defined them for Binary Search. The next few lines (**lines 7-8**) are also the same as the code in a binary search. On **line 10**, we check if `A[mid]` is less than `mid` to decide which portion of the array to discard in further search. If the condition on **line 10** evaluates to `True`, execution jumps to **line 11** where `low` is set to `mid+1` to discard the portion to the left of `mid`. However, if the condition on **line 10** evaluates to `False`, the condition on **line 12** is evaluated. If `A[mid]` is greater than `mid`, i.e., `high` is set to `mid-1` to disregard the portion to the right of the midpoint. If both the conditions on **line 10** and **line 11** are `False`, it implies that `A[mid]` is equal to `mid`. We have found a fixed point! In this case, `A[mid]` is returned from the function on **line 15**. To cater to the case if there is no fixed point in the array, we return `None` on **line 16** after the `while` loop terminates.

As we have employed a binary search to write the above code, the time complexity for the code above is $O(logn)$ while the space complexity is $O(1)$.

The solution above was pretty straightforward. You can run the linear and binary search solution in the code widget below.

```
# Time Complexity: O(n)
# Space Complexity: O(1)
def find_fixed_point_linear(A):
    for i in range(len(A)):
        if A[i] == i:
            return A[i]
    return None


# Time Complexity: O(log n)
# Space Complexity: O(1)
```

```python
def find_fixed_point(A):
    low = 0
    high = len(A) - 1


    while low <= high:
        mid = (low + high)//2

        if A[mid] < mid:
            low = mid + 1
        elif A[mid] > mid:
            high = mid - 1
        else:
            return A[mid]
    return None

# Fixed point is 3:
A1 = [-10, -5, 0, 3, 7]

# Fixed point is 0:
A2 = [0, 2, 5, 8, 17]

# No fixed point. Return "None":
A3 = [-10, -5, 3, 4, 7, 9]
print("Linear Approach")
print(A1)
print(find_fixed_point_linear(A1))
print(A2)
print(find_fixed_point_linear(A2))
print(A3)
print(find_fixed_point_linear(A3))
print("Binary Search Approach")
print(A1)
print(find_fixed_point(A1))
print(A2)
print(find_fixed_point(A2))
print(A3)
print(find_fixed_point(A3))
```

You'll hopefully be getting the hang of binary search by now. Let's solve another problem using binary search in the next lesson.