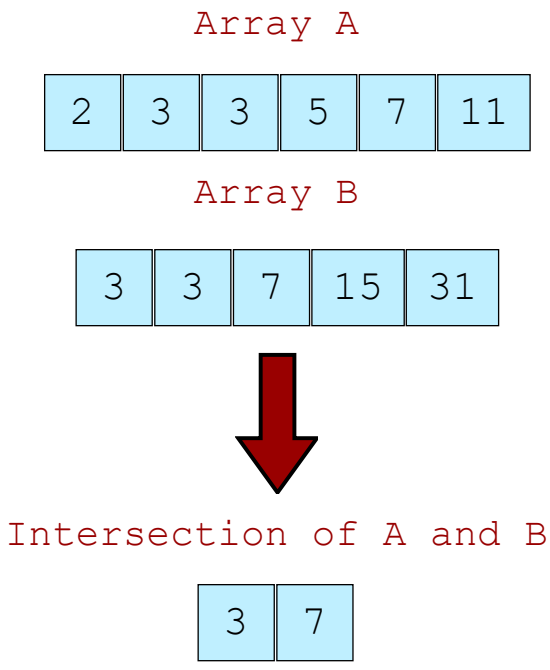
Intersection of Two Sorted Arrays

In this lesson, you will learn how to find the intersection of two sorted arrays in Python.

WE'LL COVER THE FOLLOWING

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

Given two sorted arrays, `A` and `B`, determine their intersection. What elements are common to `A` and `B`?



There is a one-line solution to this problem in Python that will also work in the event when the arrays are not sorted.

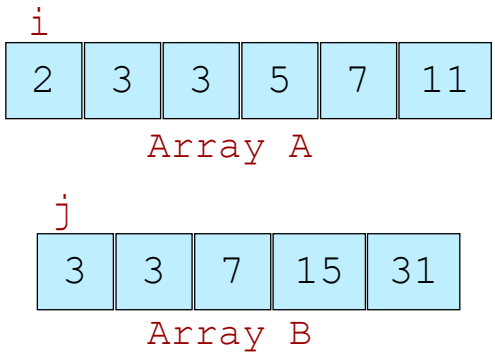
```
A = [2, 3, 3, 5, 7, 11]
B = [3, 3, 7, 15, 31]


print(set(A).intersection(B))
```

The `set` function in Python operates on a list and returns a set object that contains the unique elements of that list. `A` is converted into a set on **line 4** by using `set(A)`. Now there is another built-in function in Python called `intersection` which operates on sets. In the code above, `intersection` returns a set which is the intersection of `A` and `B`. The method `intersection` is a member of the `Set` class, but it can accept lists as arguments too. This solution is fine considering it works on unsorted arrays as well. However, since we are aware that the arrays `A` and `B` are sorted, we can use this to our advantage and solve the problem in a way that leverages this and gives us a slightly better runtime.

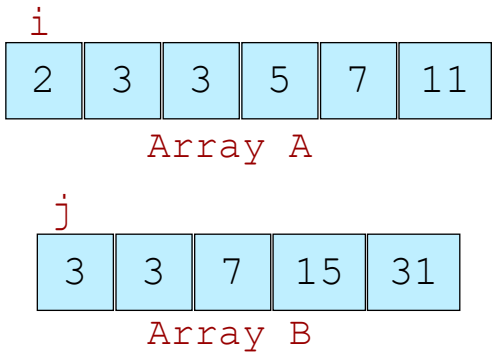
Algorithm

In the algorithm that we'll use to solve this problem more efficiently, we will make use of two iterators, `i` and `j`, one for each array. If the ith element in `array1` is less than the jth element in `array2`, we increment the `ith` iterator to increase the value so that we can reach a match with `array2`. On the other hand, if `array1[i]` is greater than `array2[j]`, `j` will be incremented so that we can get a larger value from `array2`. If both the elements pointed to by the iterators are equal, we have found an intersection and both the iterators are incremented.

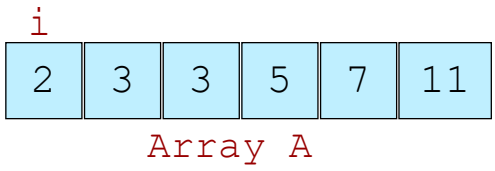
Additionally, we need to cater for an edge case of duplicates. If an intersection is recorded for some elements, we need to make sure that we don't count another intersection of their duplicates. In the slides below, this algorithm has been illustrated for you.

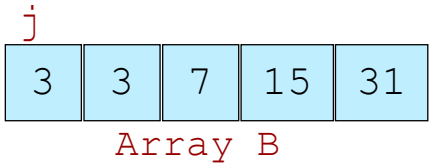


We'll traverse these arrays according to the rules defined in the algorithm.




```
if A[i] < B[j]
        i++
if A[i] > B[j]
        j++
if A[i] == B[j] and A[i] != A[i-1]
        Intersection Found!
        i++; j++
    if A[i] == B[j]
        i++; j++
```

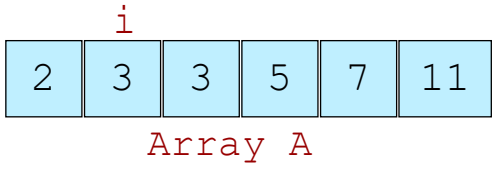


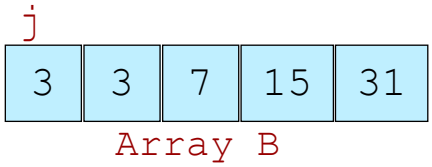


if A[i] < B[j]

i++

```
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

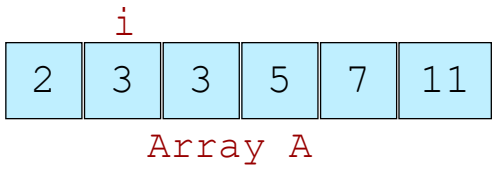



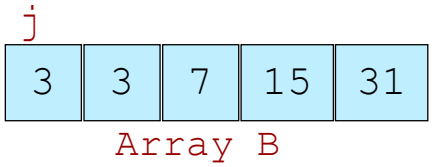


if A[i] < B[j]

i++

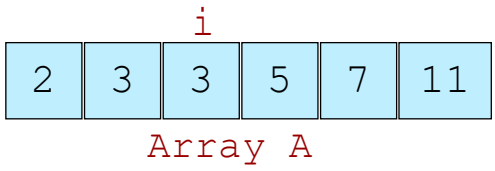
```
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

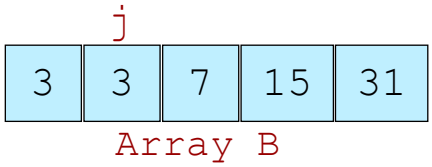





```
if A[i] < B[j]
      i++
if A[i] > B[j]
      j++
if A[i] == B[j] and A[i] != A[i-1]
      Intersection Found!
      i++; j++
if A[i] == B[j]
    i++; j++
```

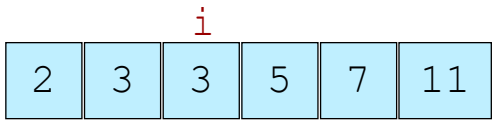
Intersection:
[3]



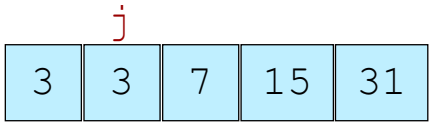



```
if A[i] < B[j]
      i++
if A[i] > B[j]
      j++
if A[i] == B[j] and A[i] != A[i-1]
      Intersection Found!
      i++; j++
if A[i] == B[j]
    i++; j++
```

Intersection:
[3]



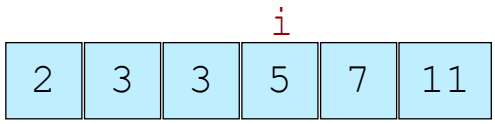
Array A



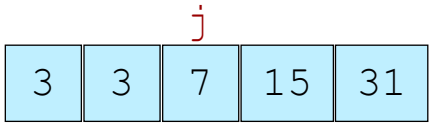
Array B

```
if A[i] < B[j]
        i++
if A[i] > B[j]
        j++
if A[i] == B[j] and A[i] != A[i-1]
        Intersection Found!
        i++; j++
    if A[i] == B[j]
        i++; j++
```

Intersection:
[3]



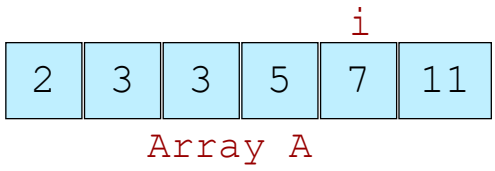
Array A

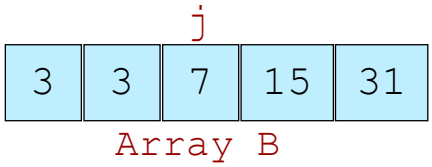


Array B

```
if A[i] < B[j]
        i++
if A[i] > B[j]
        j++
if A[i] == B[j] and A[i] != A[i-1]
        Intersection Found!
        i++; j++
    if A[i] == B[j]
        i++; j++
```

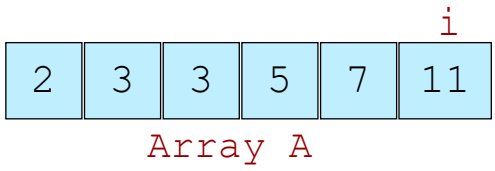
Intersection:
[3]

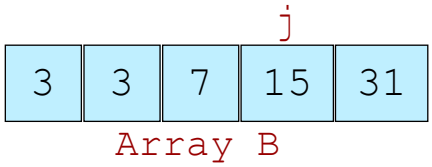





```
if A[i] < B[j]
    i++
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

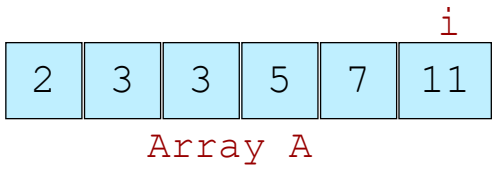
Intersection:
[3]

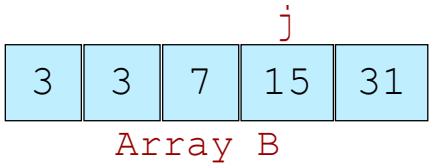





```
if A[i] < B[j]
    i++
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

Intersection:
[3,7]

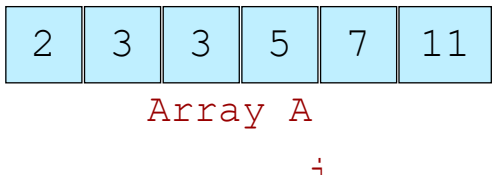


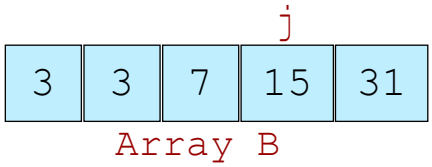


if A[i] < B[j]
i++

```
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

Intersection:
[3,7]

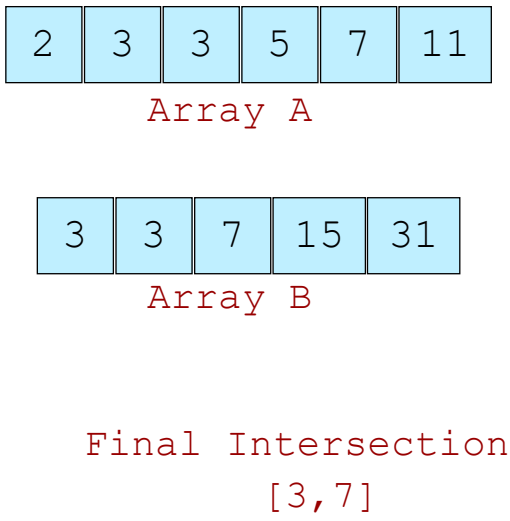




if A[i] < B[j]
i++

```
if A[i] > B[j]
    j++
if A[i] == B[j] and A[i] != A[i-1]
    Intersection Found!
    i++; j++
if A[i] == B[j]
    i++; j++
```

Intersection:
[3,7]



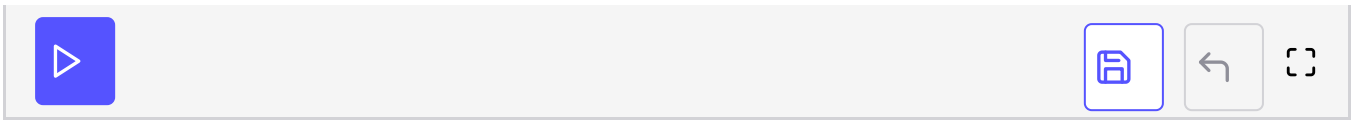
13 of 13

Implementation

As evident from the slides above, the solution will be complete once any iterator is done traversing its respective array. Now all that is left for us is to code the solution in Python. Let's go!

```
def intersect_sorted_array(A, B):
    i = 0
    j = 0
    intersection = []

    while i < len(A) and j < len(B):
        if A[i] == B[j]:
            if i == 0 or A[i] != A[i - 1]:
                intersection.append(A[i])
            i += 1
            j += 1
        elif A[i] < B[j]:
            i += 1
        else:
            j += 1
    return intersection

A = [2, 3, 3, 5, 7, 11]
B = [3, 3, 7, 15, 31]
```

```
print(intersect_sorted_array(A, B))
```



Explanation

`i` and `j` have been set to `0` on **lines 2-3**. `intersection` has been initialized to an empty list on **line 4**. The `while` loop on **line 6** is where the crux of the algorithm lies. The `while` loop will terminate as soon as `i` or `j` becomes equal or greater than the length of the arrays they are iterating on. The conditions from **lines 7-15** are reflective of the conditions discussed in the algorithm with some slight variations.

One of these conditions is when we check for duplicates using the condition (`A[i] != A[i - 1]`) on **line 8**. In that case, we need to handle the code when `i` equals `0`. Therefore, we have an additional check on **line 8**, i.e., `i == 0`. Therefore, if `i` equals `0`, there is no need to check for the duplicate condition (`A[i] != A[i - 1]`) given on **line 8**. Instead, `A[i]` is appended to `intersection` if `A[i]` equals `B[j]` and `i` equals `0`.

After the `while` loop terminates, `intersection` is returned from the function on **line 16**.

I hope you have enjoyed the chapter so far. In the next lesson, we have an interesting challenge waiting for you!