5 Counting Inversions

arr: [1, 3, 4, 2, 7, 0]

Total inversions: 2 + 5 = 7

[1, 3, 4, 2, 7, 0]

[1, 3, 4] [2, 7, 0]

[1, 3] [4] [2, 7] [0]

[1] [3] [2] [7]

Divide Steps

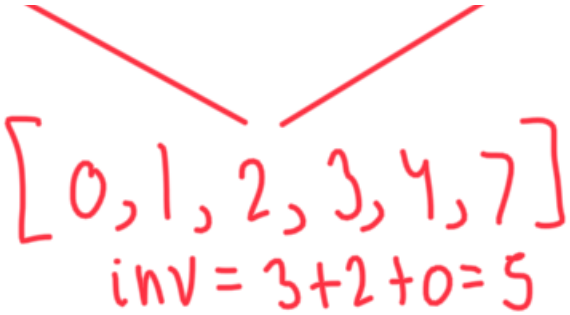
Merge Steps

[1, 3]
inv = 0

[2, 7]
inv = 0

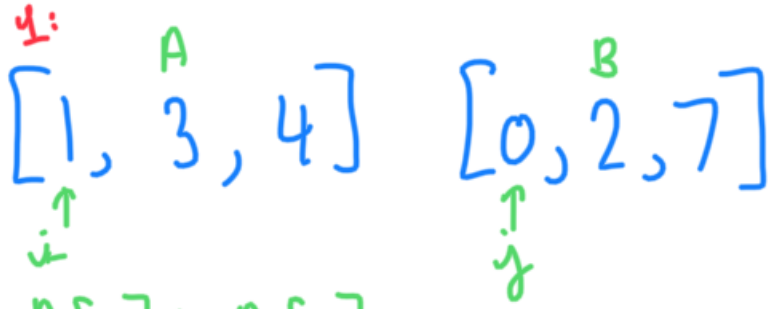
[1, 3, 4]
inv = 0

[0, 2, 7]
inv = 1 + 1 = 2



Merge:

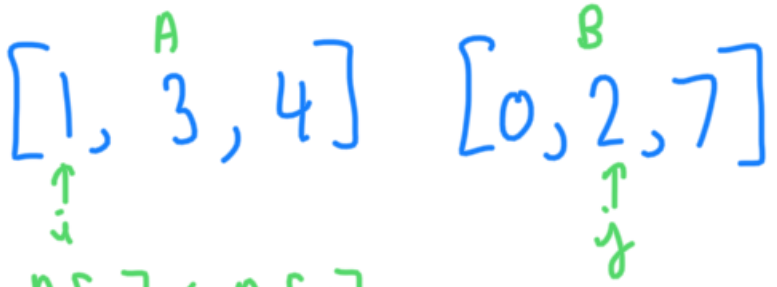
1:



A[i] > A[j]:

inv += 3, res.add(A[j]), j++

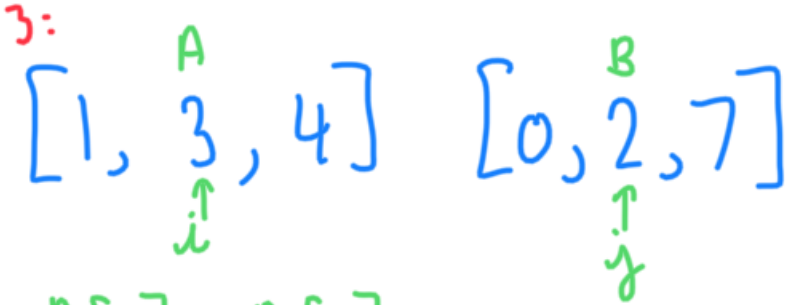
2:



A[i] ≤ A[j]:

res.add(A[i]), i++

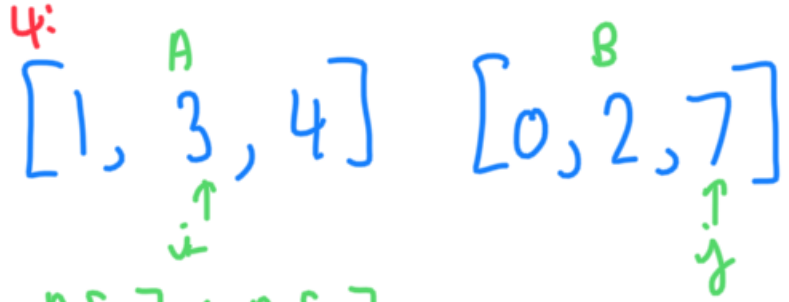
3:



A[i] > A[j]:

inv += 2, res.add(A[j]), j++

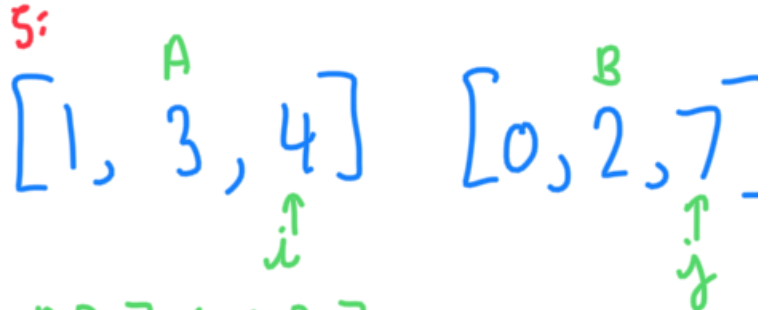
4:



A[i] ≤ A[j]:

res.add(A[i]), i++

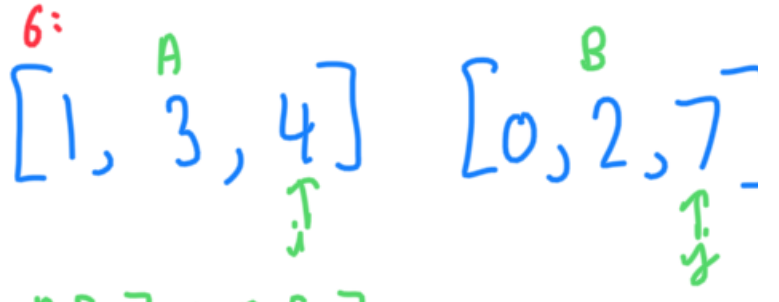
5:



A[i] ≤ A[j]:

res.add(A[i]), i++

6:



A[i] ≤ A[j]:

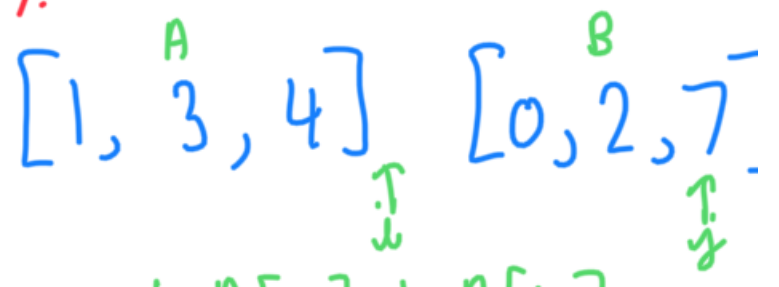
res.add(A[i]), i++

When A[i] > A[j], inv += len(A) − j + 1

if start idx=1

Total inversions in merge(A,B) = 3+2 = 5

7:



res + A[i:] + B[j:]

We can modify the Merge sort to count all possible inversions from a given list.

From the example,
    arr: [1, 3, 4, 2, 7, 0]

We can see that there are
    7 total inversions: (1,0), (3,2), (3,0), (4,2), (4,7), (2,0), (7,0)

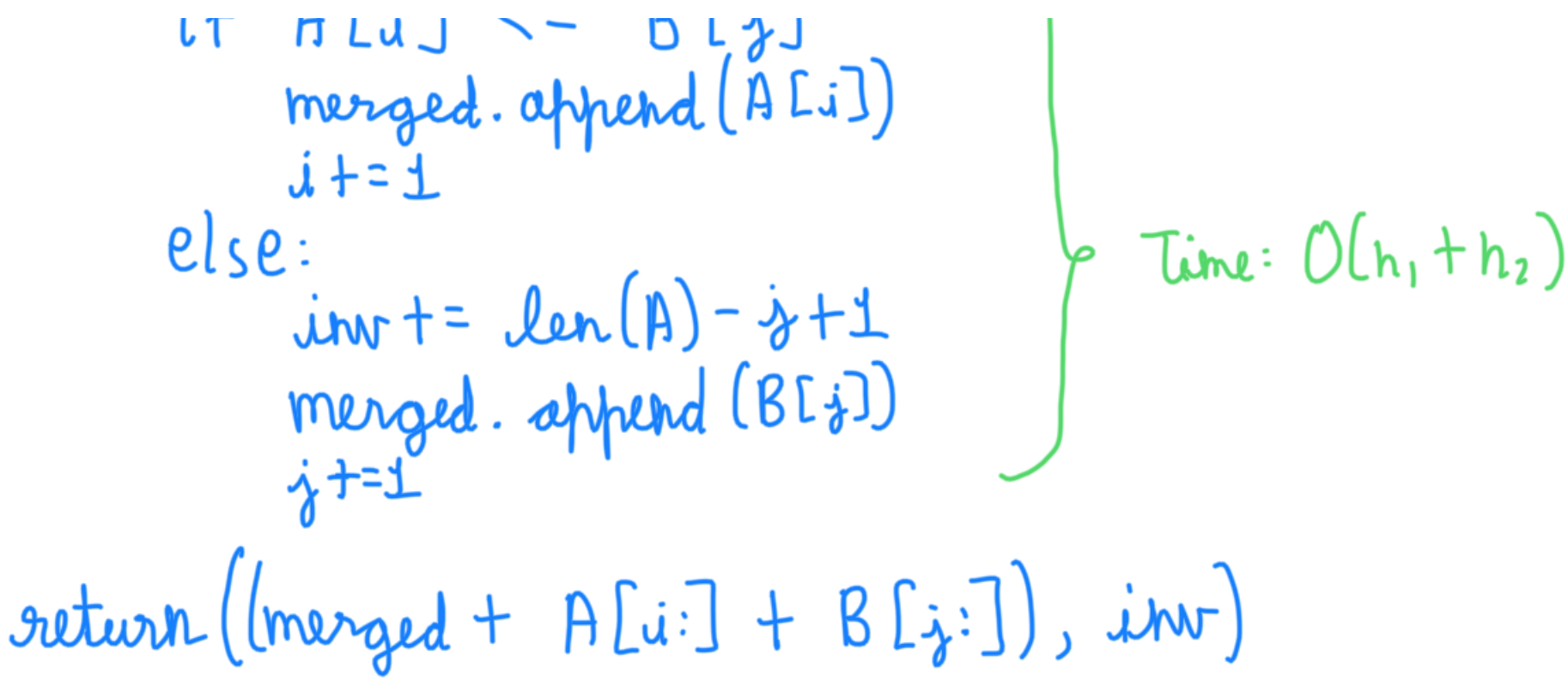
```
def merge_sort(arr[lo...hi]):
    if (len(arr) <= 1):          } b
        return (arr, 0)

    mid = ⌊(lo+hi)/2⌋

    A, inv_A = merge_sort(arr[lo... mid])      — T(n/2)
    B, inv_B = merge_sort(arr[mid+1...hi])     — T(n/2)
    M, inv_M = merge(A, B)  ———————  O(n)

    return (M , (inv_A + inv_B + inv_M))
```

$$T(n) = \begin{cases} b & , \text{if } n=1 \\ 2T(n/2)+n & , \text{if } n>1 \end{cases}$$

Time: $T(n) = O(n \log n)$

Space: $n + \log n = O(n)$

```
def merge(A[i... n1], B[j... n2]):
    merged, inv = [], 0
    while  i <= n1  and  j <= n2:
        if A[i] <= B[j]
```



Code:

```
def countInversions(arr):
    if len(arr) <= 1:
        return (arr, 0)

    mid = (len(arr)//2)

    A, invA = countInversions(arr[:mid])
    B, invB = countInversions(arr[mid:])
    M, invM = merge(A, B)

    return (M, (invA+invB+invM))


def merge(A, B):
    i, j = 0, 0
    merged, inv = [ ], 0

    while i < len(A) and j < len(B):
        if A[ i ] <= B[ j ]:
            merged.append(A[ i ])
            i+=1
```

```
    else:
        inv += len(A) - j
        merged.append(B[ j ])
        j+=1

return ((merged + A[ i: ] + B[ j: ]), inv)
```