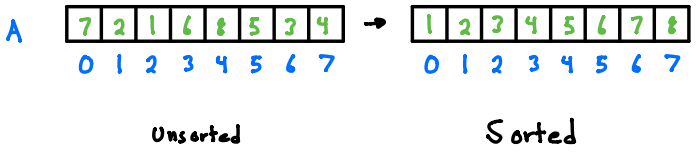
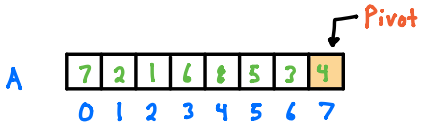
## Quick Sort

- Divide and Conquer
- $O(n \log n)$ - Average case running time
- $O(n^2)$ - Worst case running time
- In-place

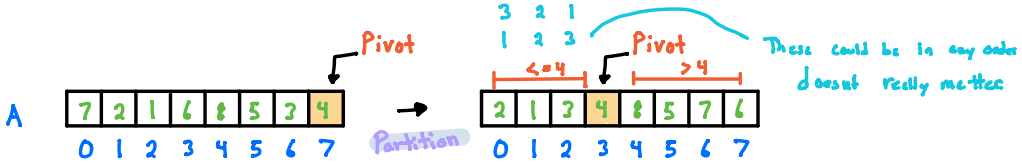



## Quick Sort Overview

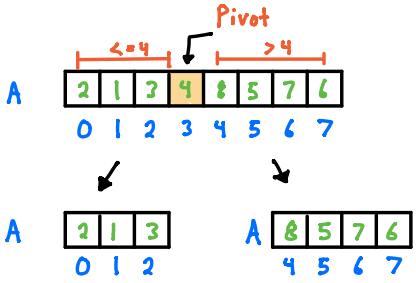



1) We first select one elements in the list (can be any element), we call this selection number, Pivot.

2) Now we rearrange the list such that all the elements lesser than the Pivot are towards the left of it and all the elements greater are towards the right of it (Partitioning of a list)



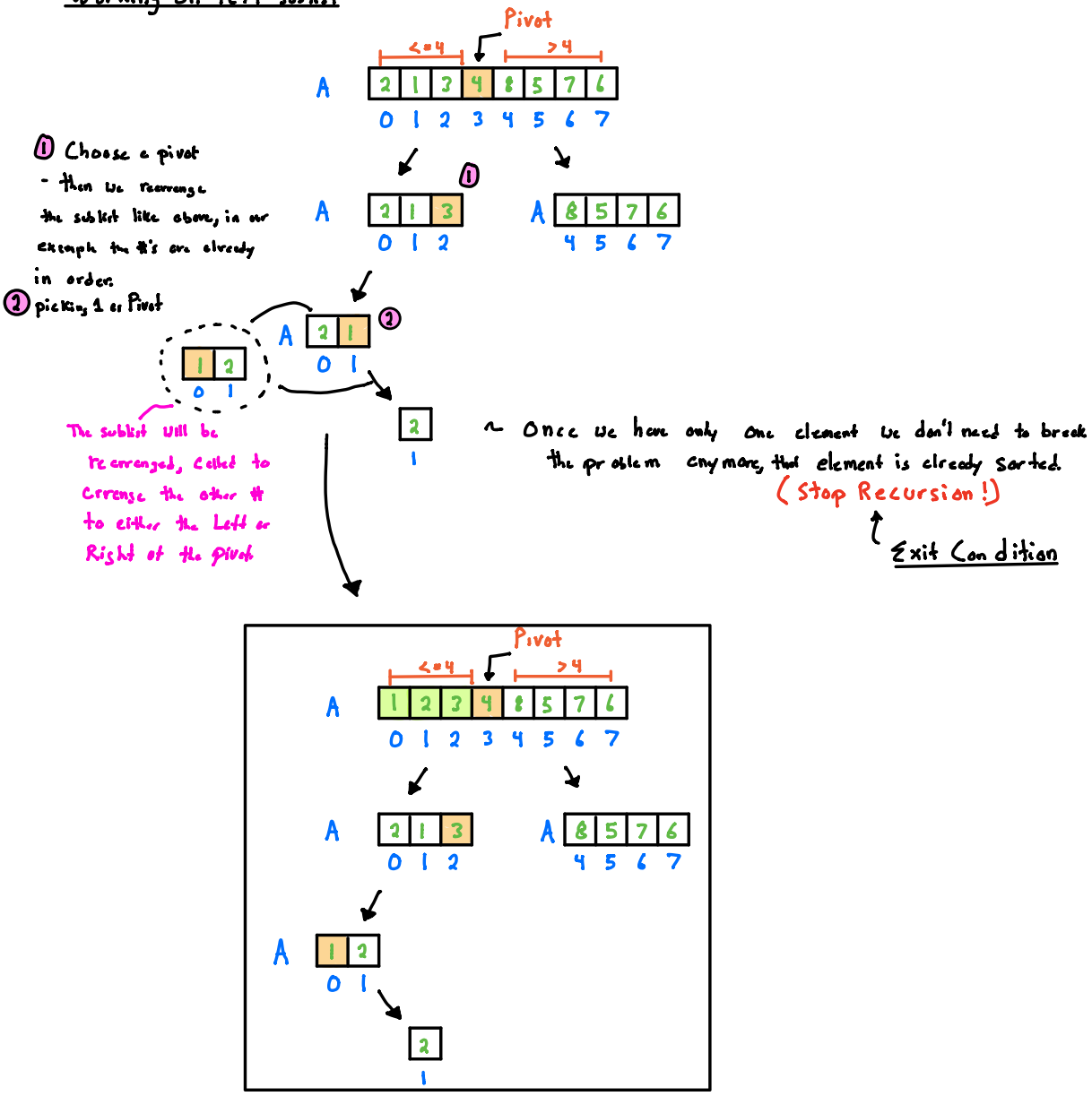

- We can break this problem into two sub-problems, these being sorting the array to the Left of the pivot and sorting the array to the right of the pivot.




### Different than Merge Sort

- Unlike merge sort we do not need to create auxilary arrays entirely, We can work on the same array. We just have to keep track of the start and end index of the segments

## Working on left sublist:







- Then we work back through the problem.

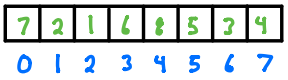
# QuickSort Function Sudo Code

```
def QuickSort( A, start, end):          # Index
    if start < end:                     # ] exit condition of recursive function
        return
    pIndex = partition (A, start, end)
    QuickSort ( A, start, pIndex-1)     # Recursive Calls
    QuickSort ( A, pIndex +1, end)
```

```
def Partition (A, Start, End):
    pivot = A[end]                      ~ Selecting the Pivot
    Partition_index = start
    for i from start to end-1 :
        if (A[i] <= pivot):
            A[i] = A[ partition_index]  ~ Swap
        partition_index = partition_index + 1
    A [ Partition Index] = A [end]      ~ Swap
    Return Partition_ Index
```

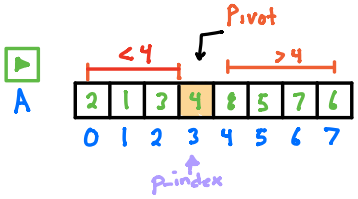
# Start

① QS(A,0,7) A




← Unsorted Array

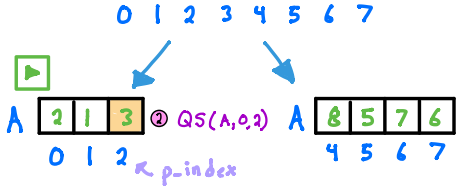
① Since this satisfies our condition Start < end (0<7), we will call the Partition function, which will find our Pivot and swap values Left or Right depending on if they're higher or lower then the Pivot point it then will return a p_index, which will be index 3, for this step

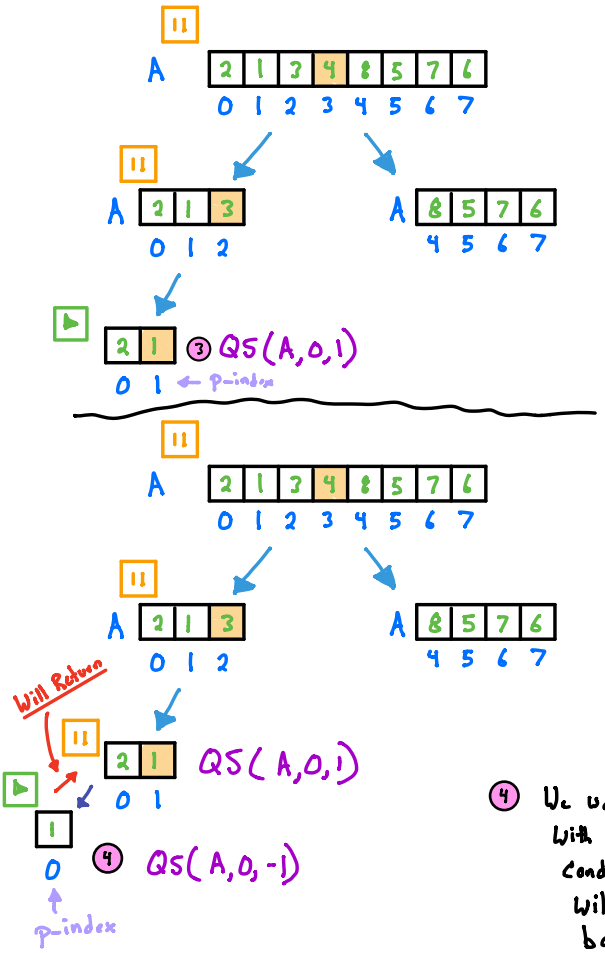



← We have 4 as our Pivot
- The Partition function will rearrange the array to this, it will return the p_index

II ← Function Call is paused.
- Machine says it will come back once im finished

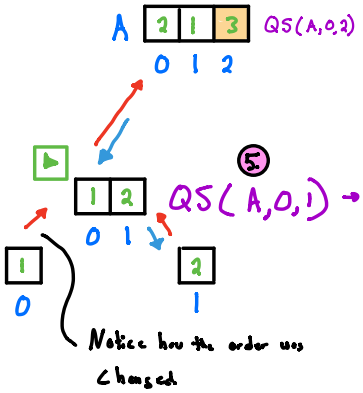
② With our p_index returned we will call the QuickSort(A, Start, pIndex-1), which will move our "End" index (3-1) = 2., With our starting being 0, it will still satisfy our Condition Start < End (0<2), so we call the Partition function it takes in our array, and index values 0 to 2, finds a Pivot point of 3 and rearranges things, and returns p_index = 2





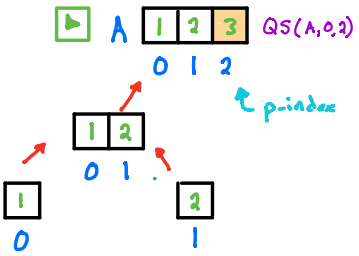

③ After the p-index is returned (QuickSort(A, Start, pIndex-1)), is called. Our p-index goes (2-1) = 1. (QS(A,0,1)), This still satisfies the condition Start > End (1>0). So we call the Partition function again. It will make the pivot become 1. and the p-index will become 0.

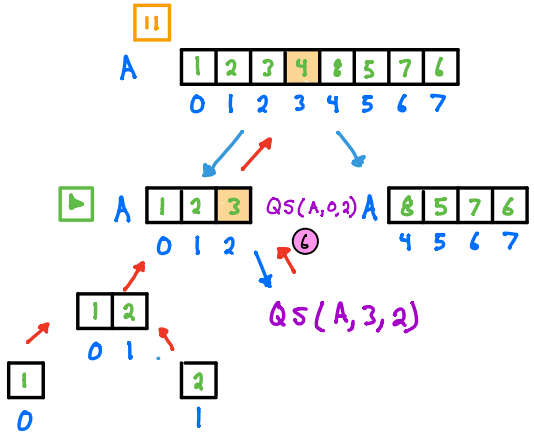
④ We we will call the (QuickSort(A, Start, pIndex-1)), With QS(A,0,0-1) = QS(A,0,-1), our condition start > End is false, so we will Return, and control will return back to QS(A,0,1).



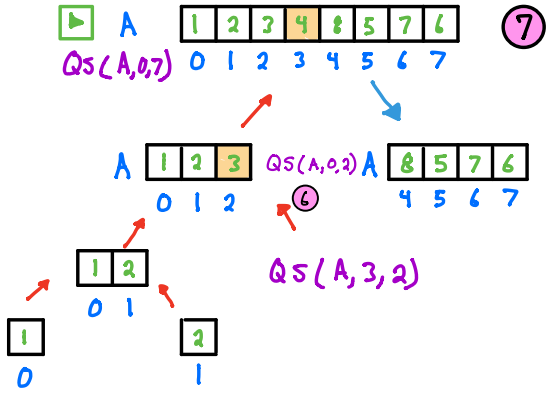

⑤ QuickSort(A, pIndex+1, end)
QS(A, 0+1, 1)
QS(A, 1, 1)

⑤ With the Control back, QuickSort(A, pIndex+1, end), is called, but since Start < end is false, it will Return and return Control to QS(A,0,2).

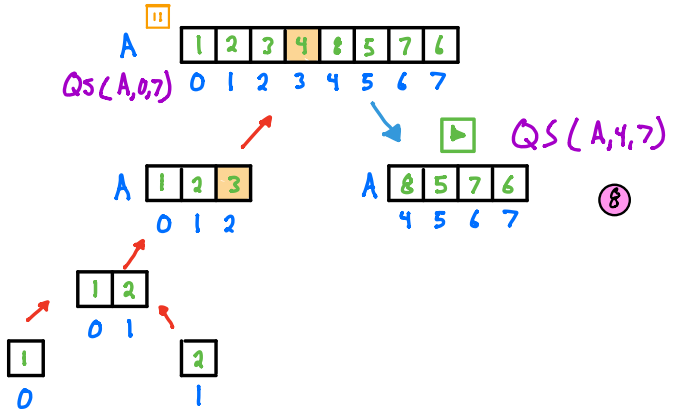


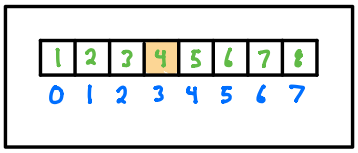

⑥ With Control back here, QuickSort(A, pIndex+1, end) is called and QS(A,3,2) is invalid, so QS will return (↗)




⑦ Our Original array at the top will now resume. QS(A,0,7). Now it will make a call right of the pivot, QuickSort(A, pIndex+1, end), which will be QS(A,4,7).

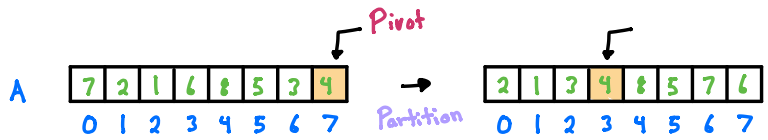



(8) Now this side of the array is solved.

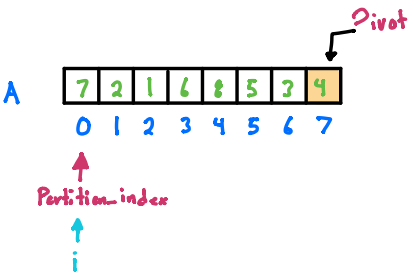



Sorted

## Partition Logic




- This what we went, to select a pivot and to rearrange a list such that all the elements lesser than the pivot are to the Left and all the elements greater than the pivot are to the right of it.




```
def Partition(A, Start, End):

    pivot = A[end]          ~ Selecting the Pivot

    Partition_index = start

    for i from start to end-1:
        if (A[i] <= pivot):
            A[i] = A[partition_index]      ~ Swap
        partition_index = partition_index + 1

    A[PartitionIndex] = A[end]      ~ Swap

    Return Partition_Index
```

**Running Time of Quicktime**

Actual Code