GRAB A BYTE

QUICK SORT!

WHAT WE'VE COVERED?

Since the second week of this semester we've covered Linear Search, Binary Search, Bubble Sort, Selection Sort, Insertion Sort and Merge Sort!

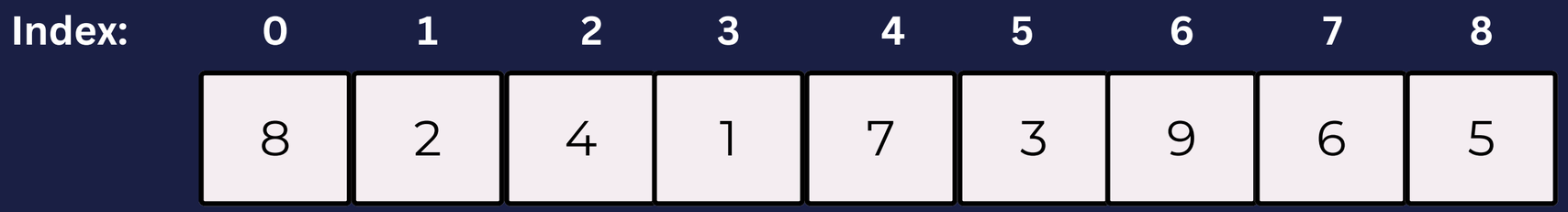
WHAT WE ARE LEARNING TODAY

Today we are covering Quick Sort!
The Quick Sort Algorithm is a sorting algorithm that utilizes “divide and conquer” strategy by selecting a “pivot” element, partitioning the array into elements smaller and larger, and recursively sorting

WHICH QUICK SORT?

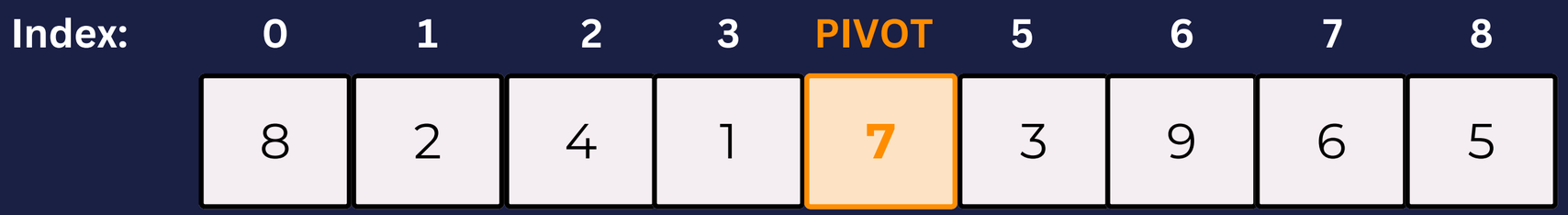
There seem to be many different versions and ways to do the Quick Sort algorithm. I've chosen the one that I beleive will cause the least amount of issues for most lists and is a better practice to use.

Lets consider an array of integer values:



We need to choose a “pivot” element. You can choose anywhere in the array, but the best practice is the middle element (or the starting index plus the ending index divided by 2)

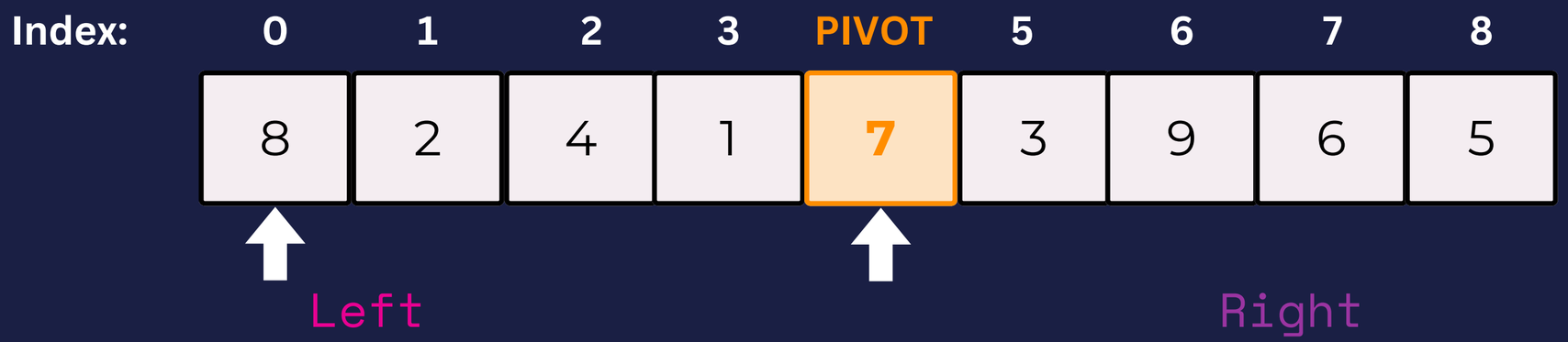
Our pivot index is 4 and the value is 7.



We will check the elements in the array and if the element is less than the value at the pivot (7) it will go in the Left partition, or if it is larger, it will go into the Right partition.

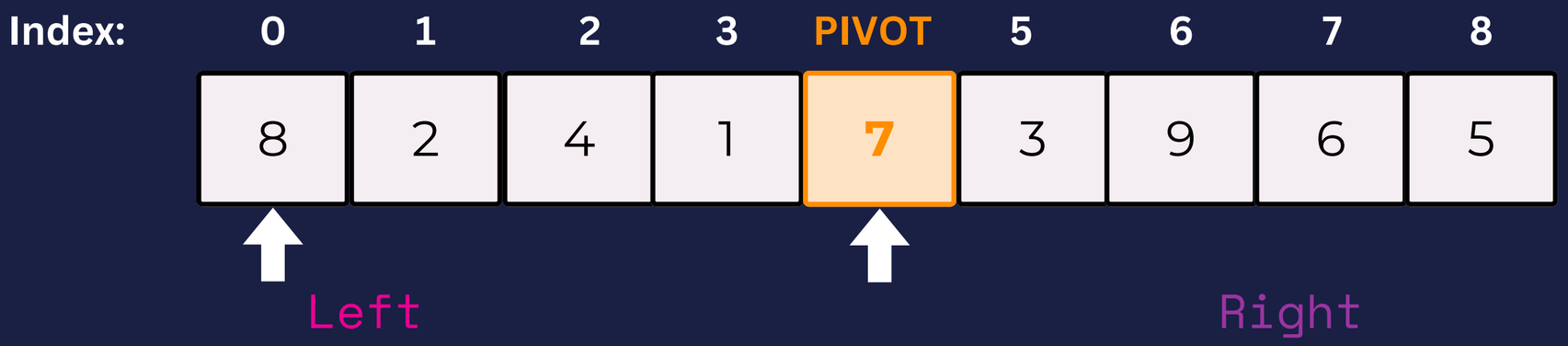
Our pivot index is 4 and the value is 7.

Our pivot index is 4 and the value is 7.



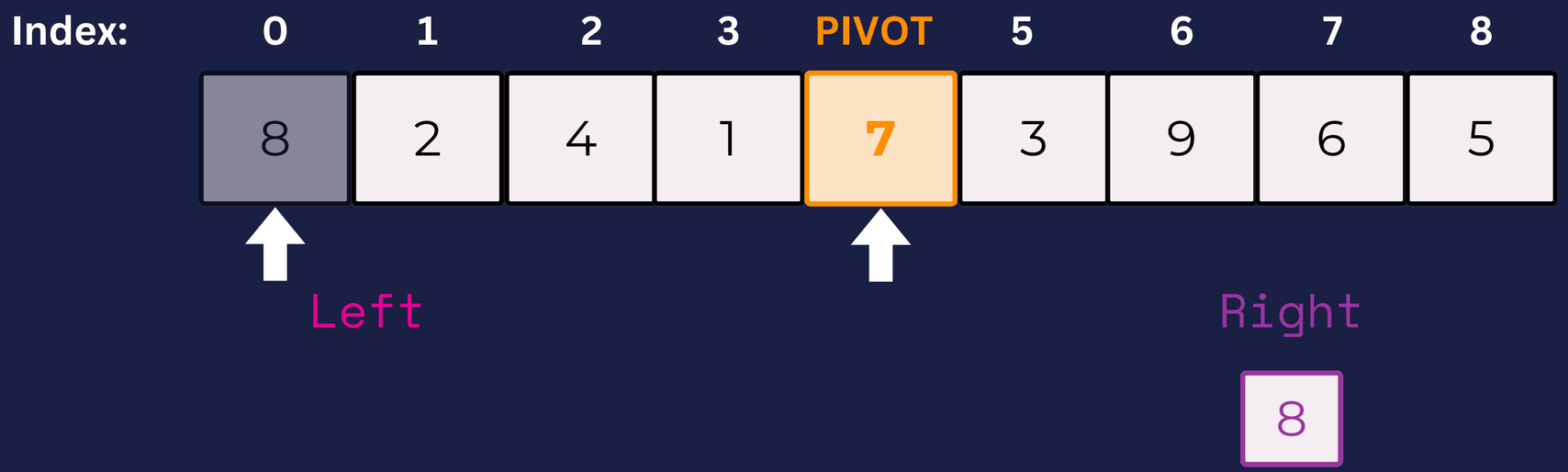
We check if 8 is less than or greater than our pivot value

Our pivot index is 4 and the value is 7.



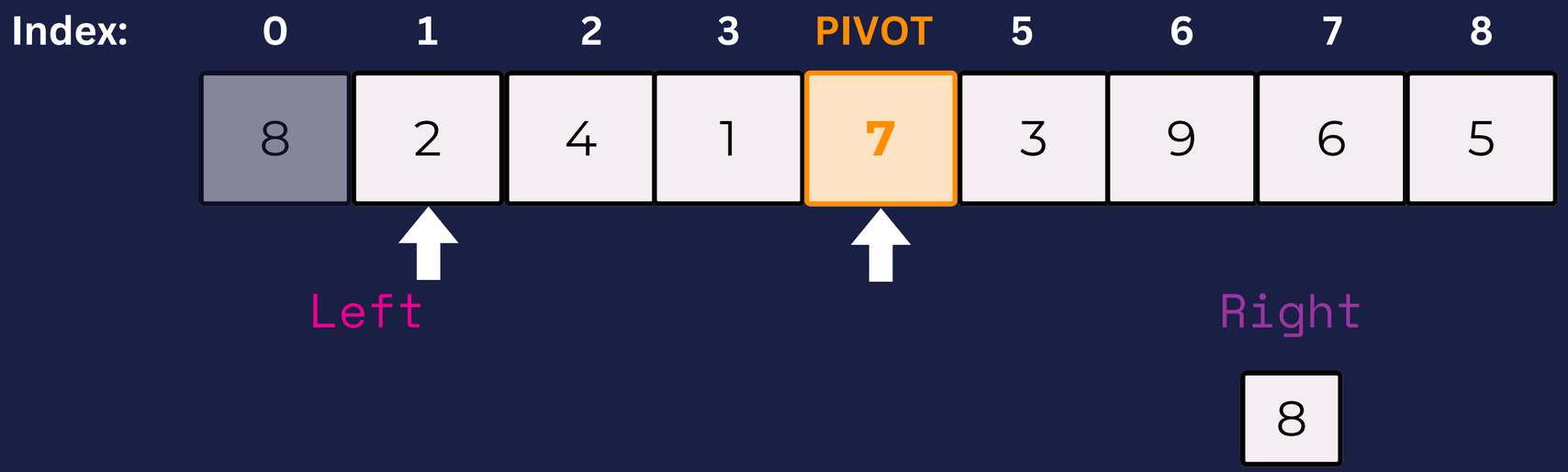
It is greater, so it gets placed in the Right partition

Our pivot index is 4 and the value is 7.



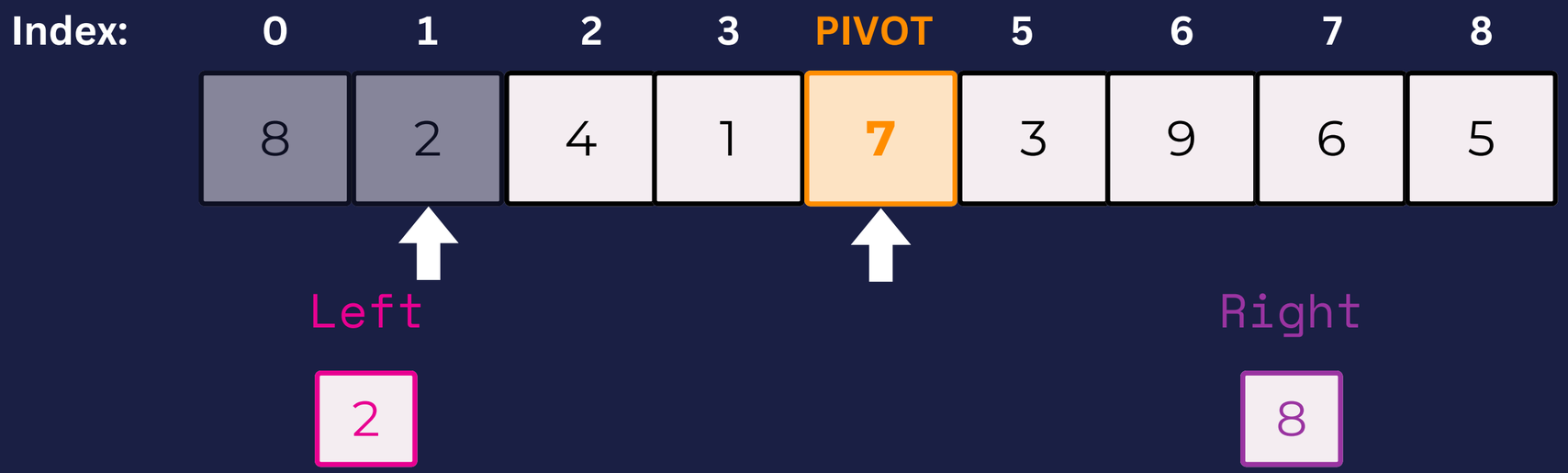
It is greater, so it gets placed in the Right partition

Our pivot index is 4 and the value is 7.



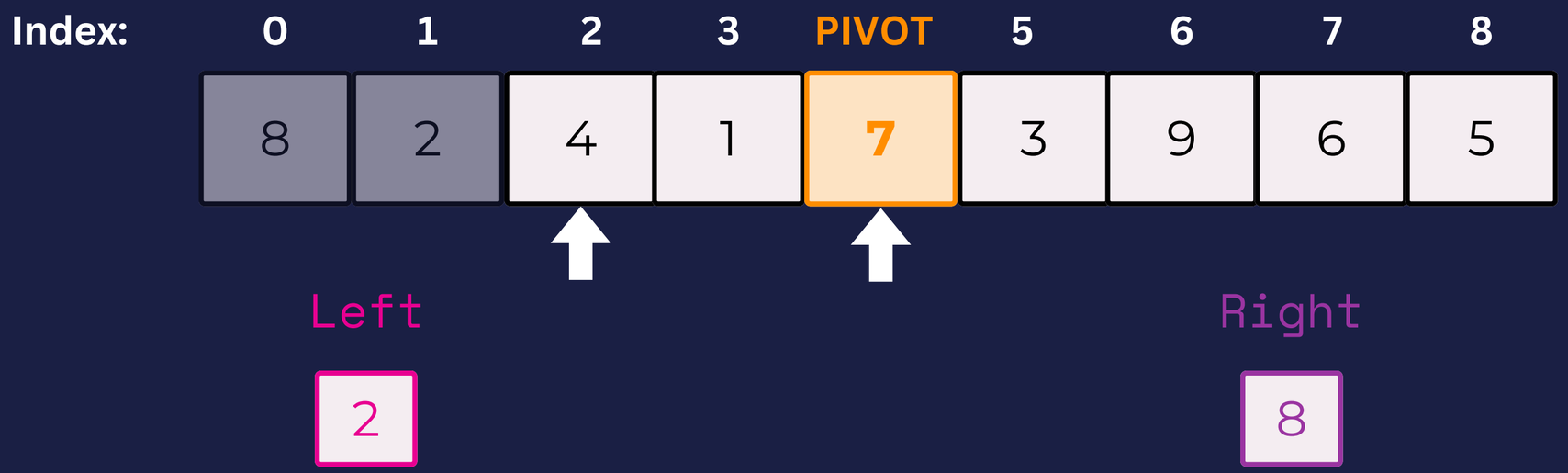
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



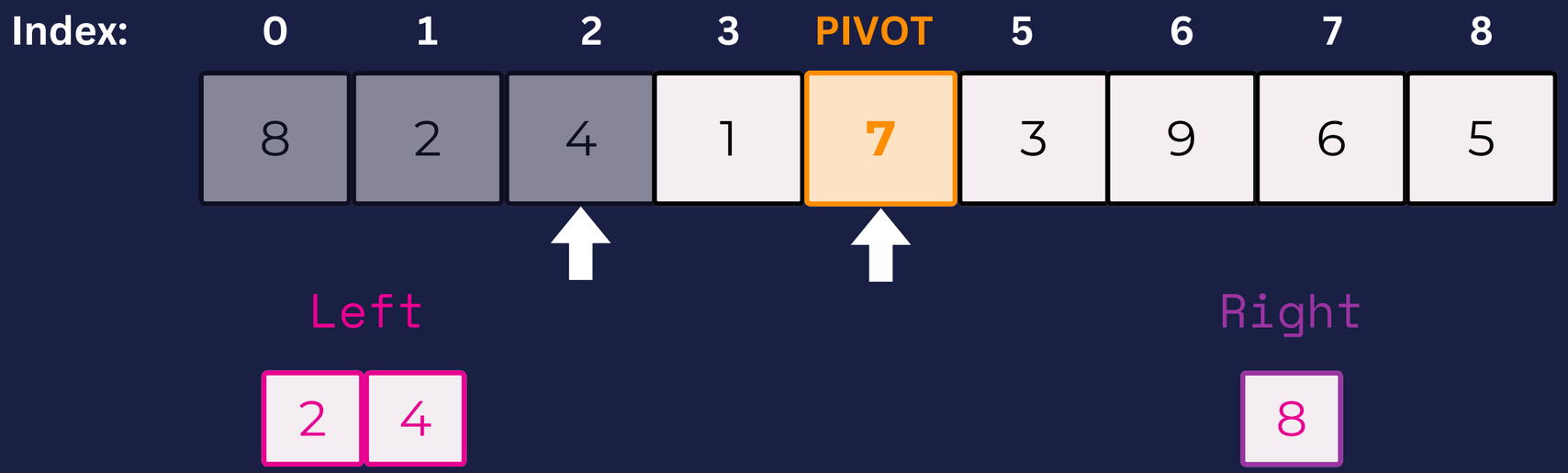
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



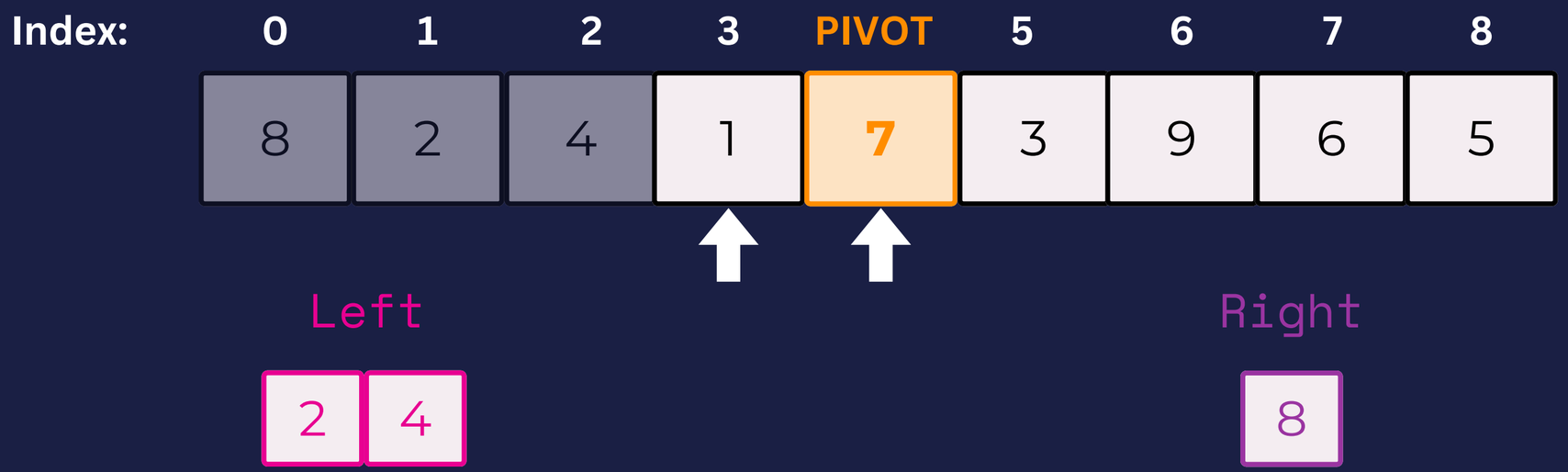
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



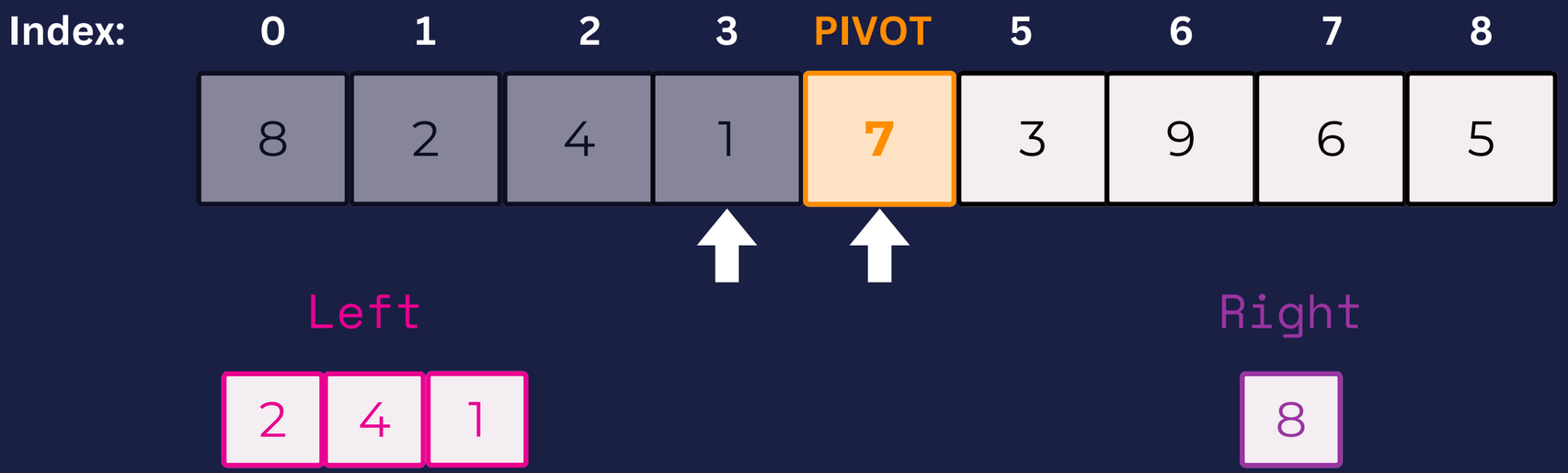
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



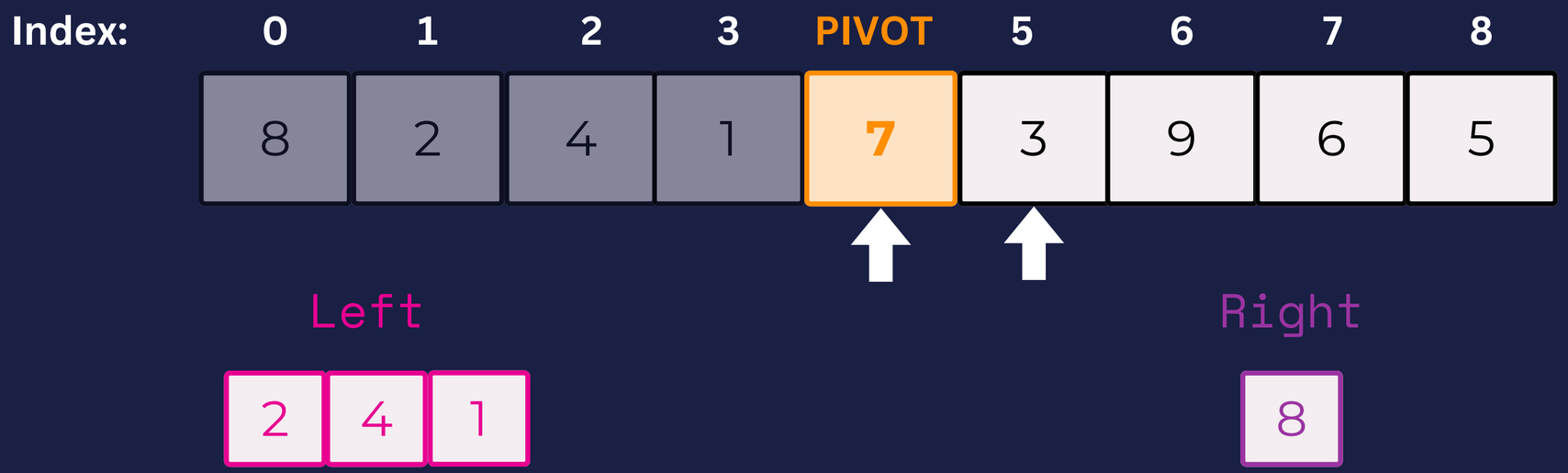
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



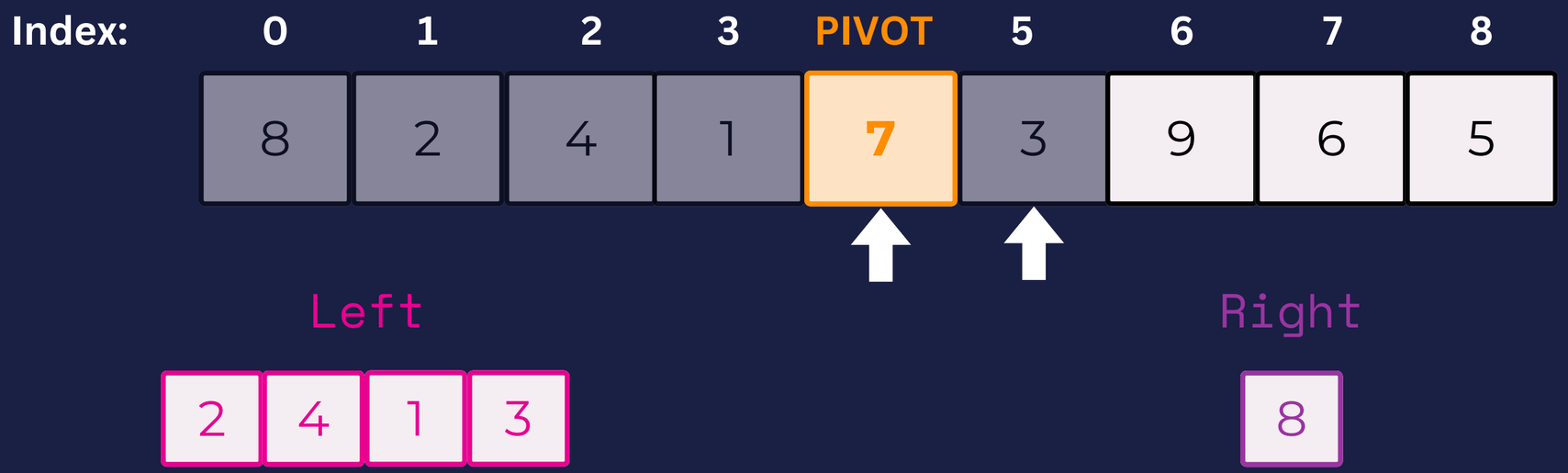
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



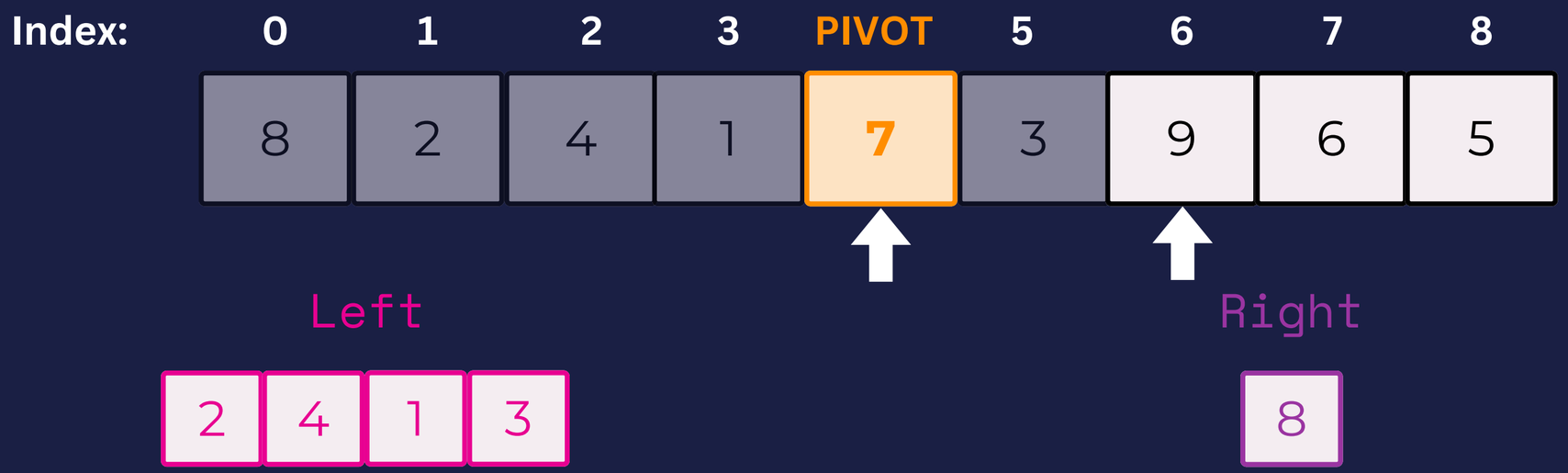
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



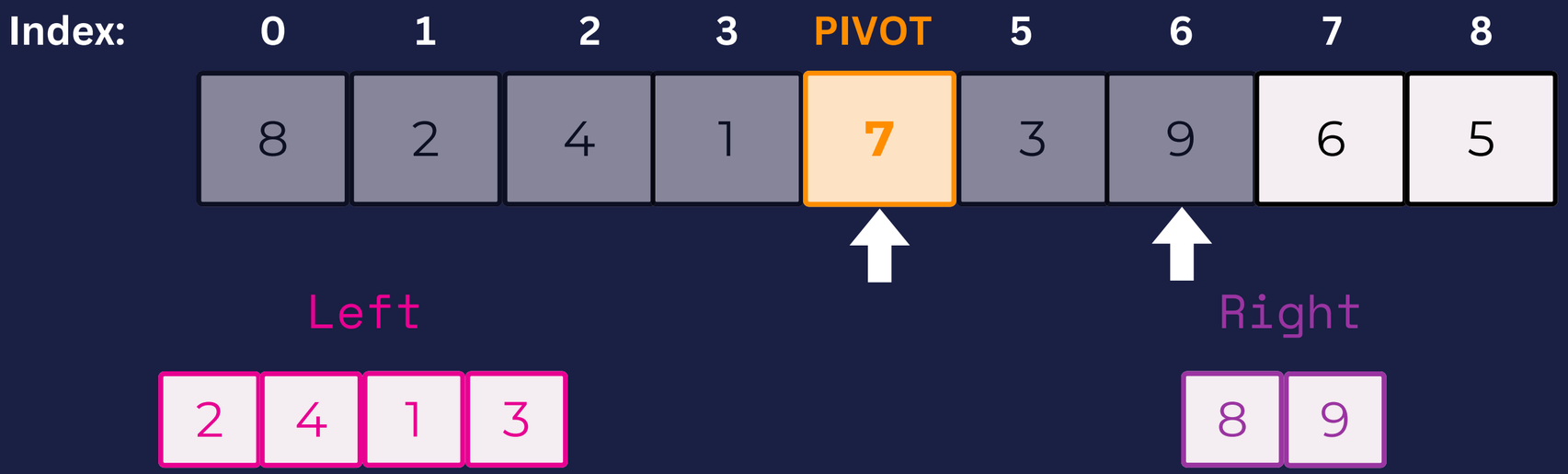
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



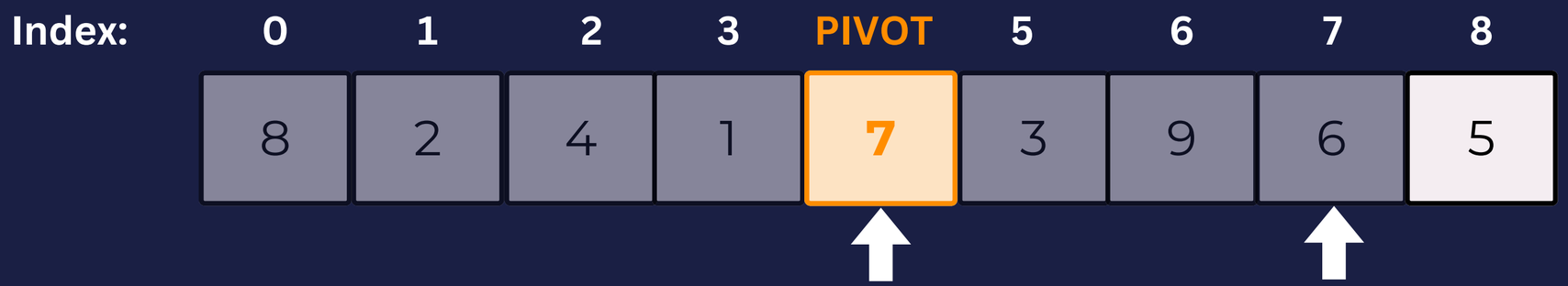
Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.



Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.

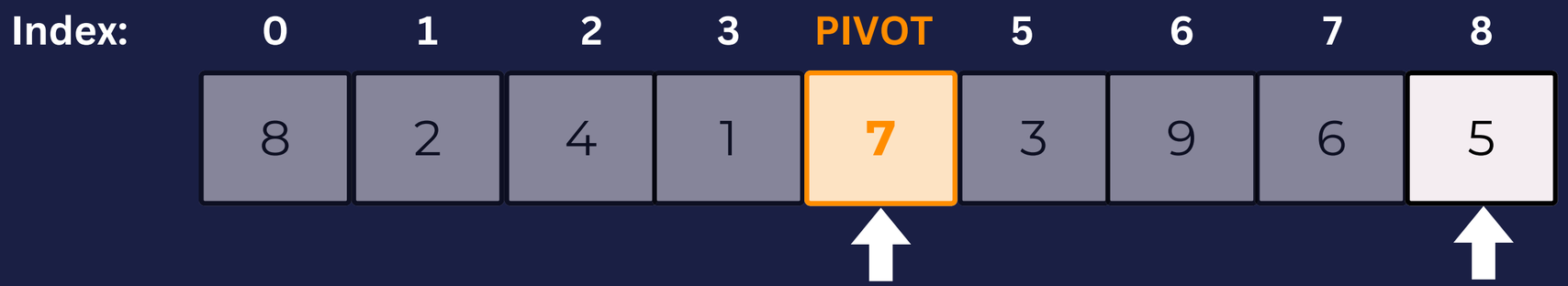


Left

Right

Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.

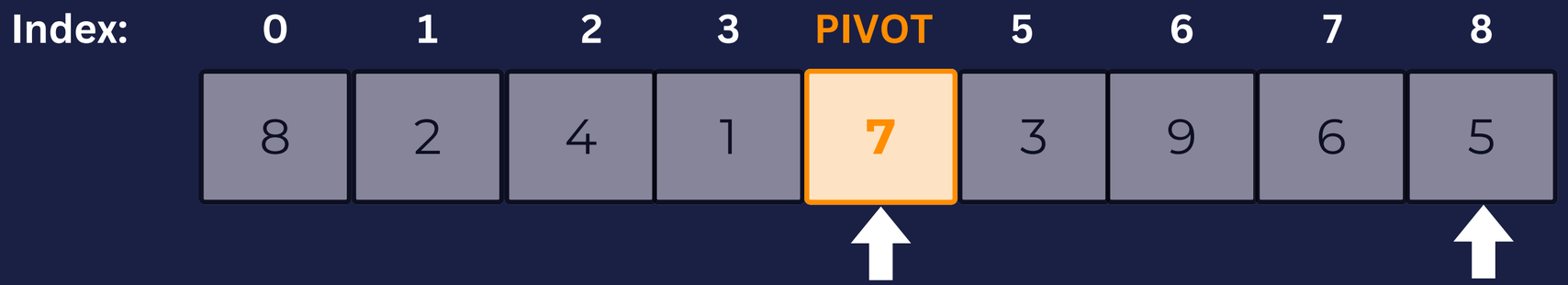


Left

Right

Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

Our pivot index is 4 and the value is 7.

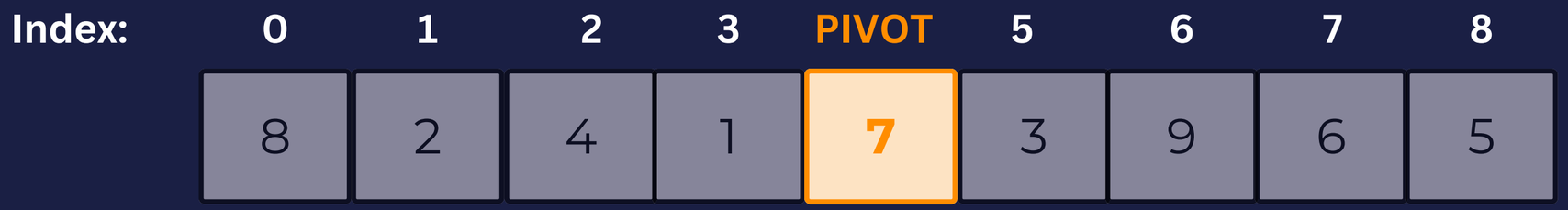


Left

Right

Now we move on to the other values
and place them in the Left (lower) or
Right (higher)

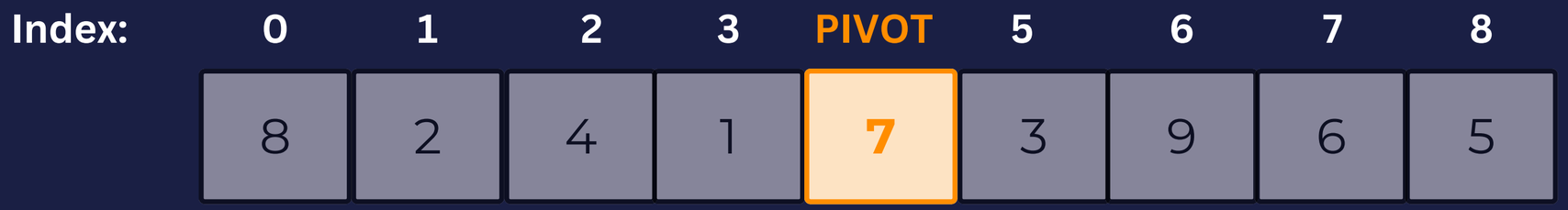
Now all values < the pivot are in the left and all values > the pivot are in the right.



Quick sort is a recursive algorithm so it calls itself when returning. ex:

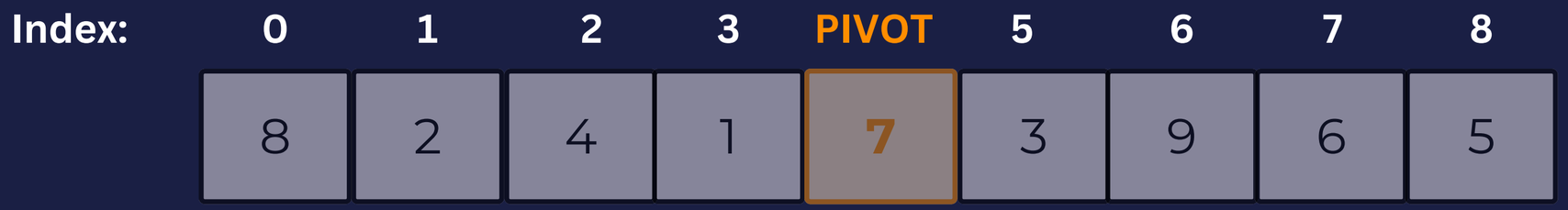
```
return quickSort(left) + pivot + quickSort(right)
```

Before it actually returns, it calls the functions again, starting with the first call: `quickSort(left)`




```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```

Before it actually returns, it calls the functions again, starting with the first call: quickSort(left)



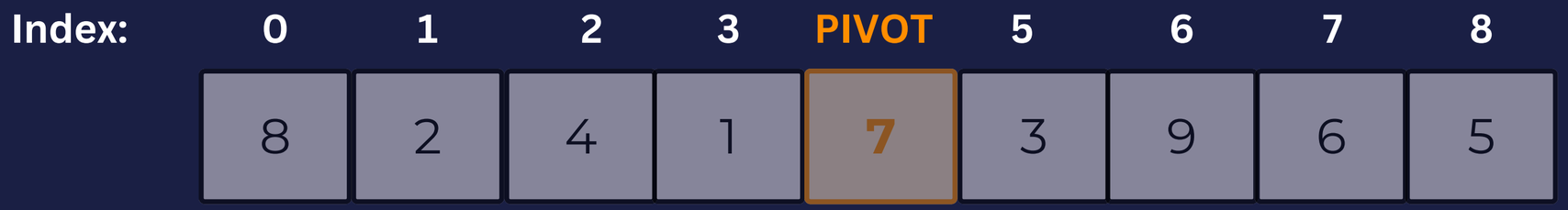
return quickSort(left) + pivot + quickSort(right)

Left

Right

We need to choose a new pivot, which is first index, plus last index divided by 2. In our case index 0 plus index 5 divided by two is 2.5, so we round down. Index 2 will be our new pivot.

Before it actually returns, it calls the functions again, starting with the first call: quickSort(left)



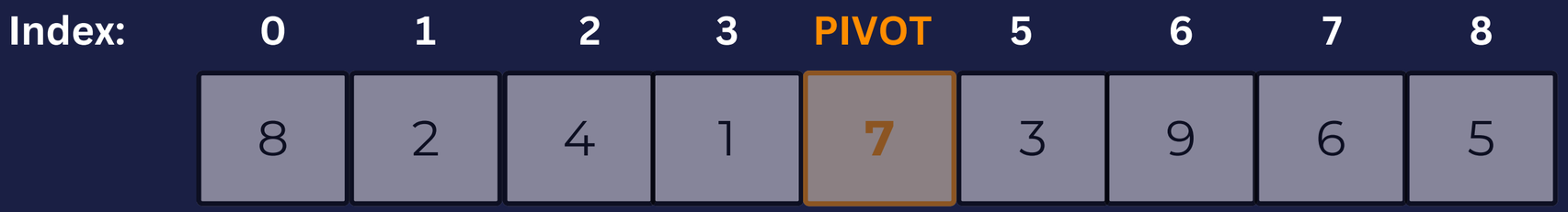
return quickSort(left) + pivot + quickSort(right)

Left

Right

We need to choose a new pivot, which is first index, plus last index divided by 2. In our case index 0 plus index 5 divided by two is 2.5, so we round down. Index 2 will be our new pivot.

And now we have a new left and right



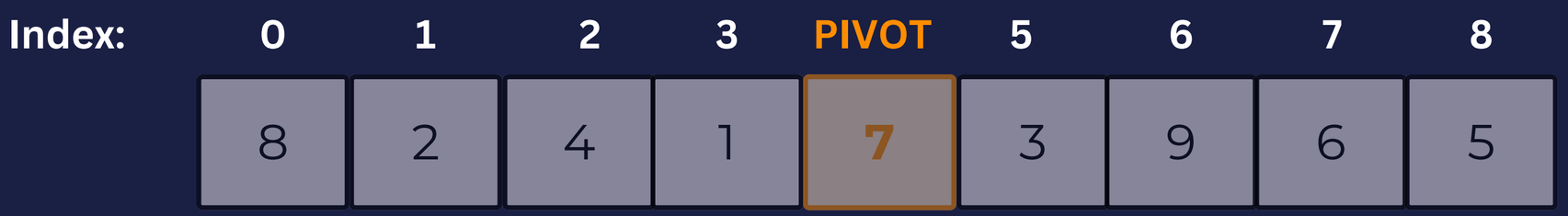

```
return quickSort(left) + pivot +  quickSort(right)
```

Left

Left Right

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

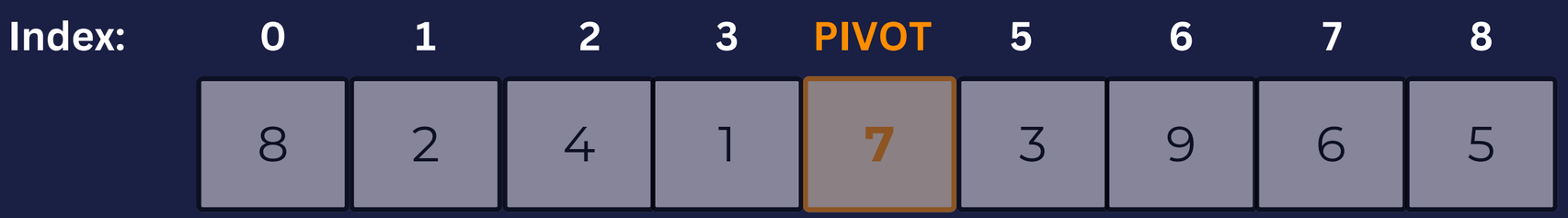
Left

Right

Left

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

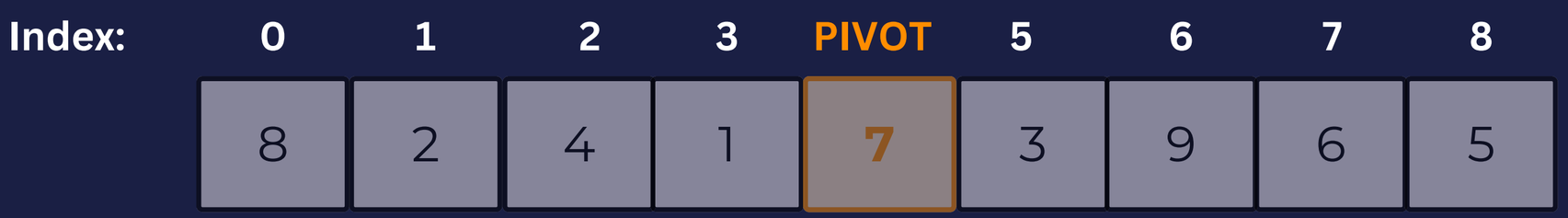
Left

Right

Left

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

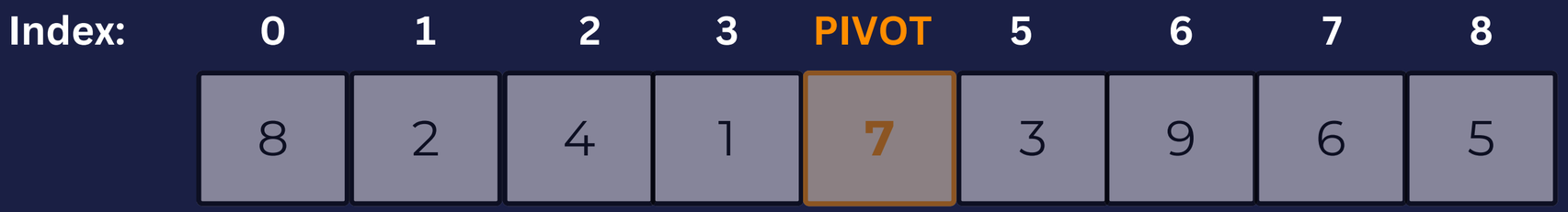
Left

Right

Left

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

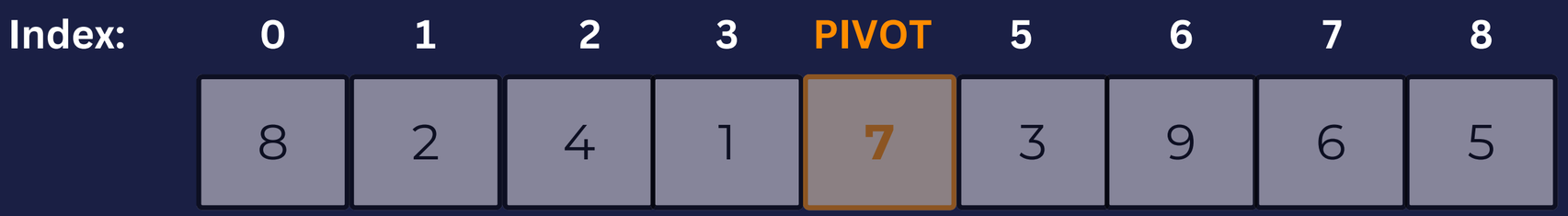
Left

Right

Left

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

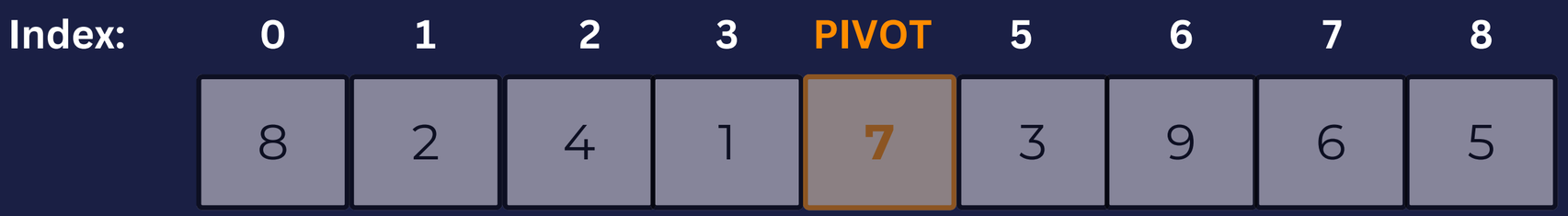
Left

Left

Right

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

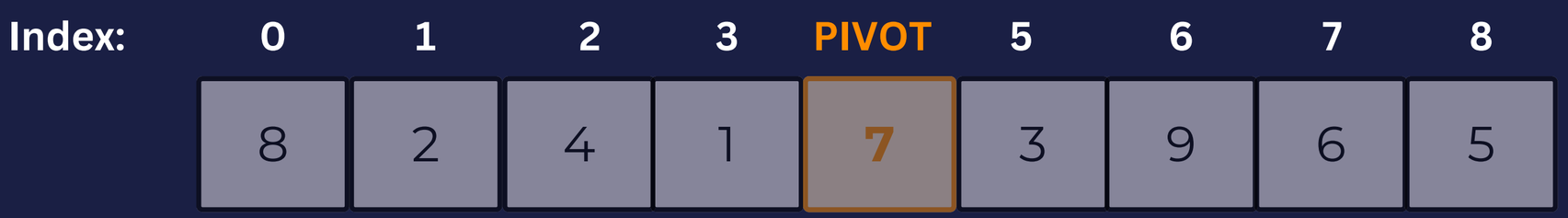
Left

Left

Right

Right

And we have to check the values and put them in the new left and right

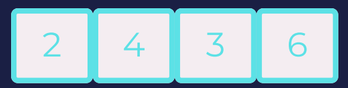



```
return quickSort(left) + pivot +  quickSort(right)
```

Left

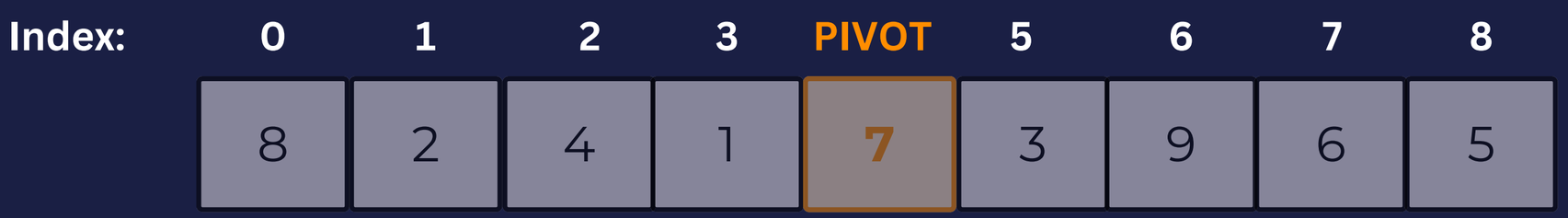
Left

Right



Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

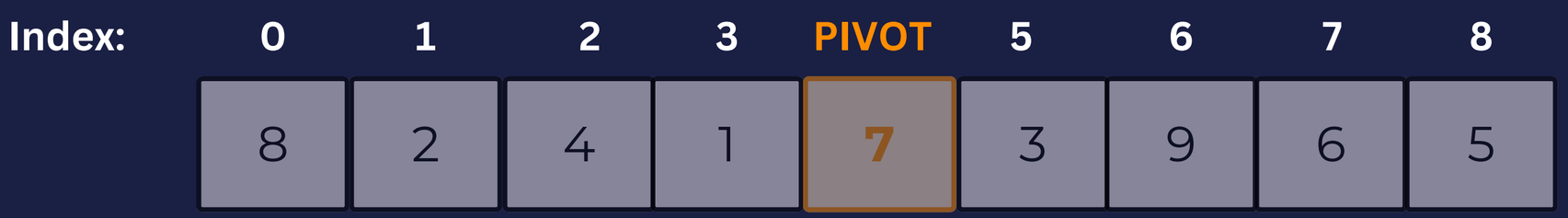
Left

Right

Left

Right

And we have to check the values and put them in the new left and right




```
return quickSort(left) + pivot +  quickSort(right)
```

Left

Right

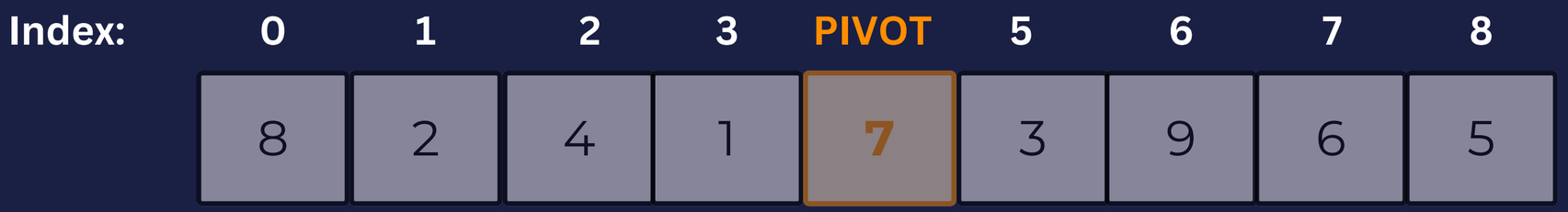
Left

Right

And it returns the values, calling the functions again.

```
return quickSort(left) + pivot + quickSort(right)
```

There is no values in left so it returns no values.




```
return quickSort(left) + pivot +  quickSort(right)
```

Left

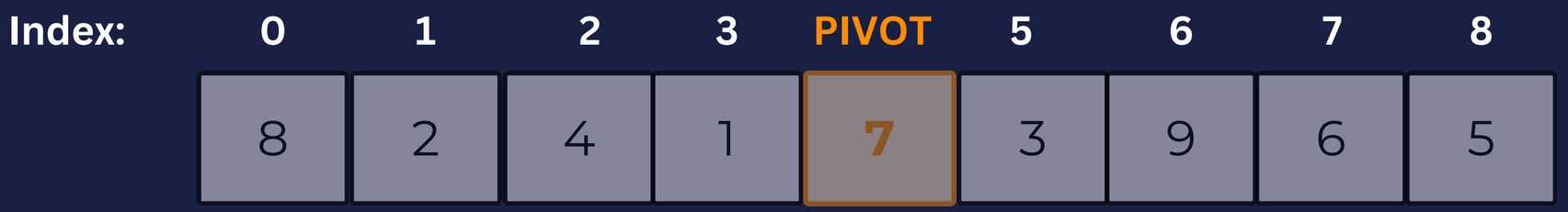
Right

Left

Right


```
return quickSort(left) + pivot + quickSort(right)
```

It will replace pivot with the current value (1)




```
return quickSort(left) + pivot +  quickSort(right)
```

Left

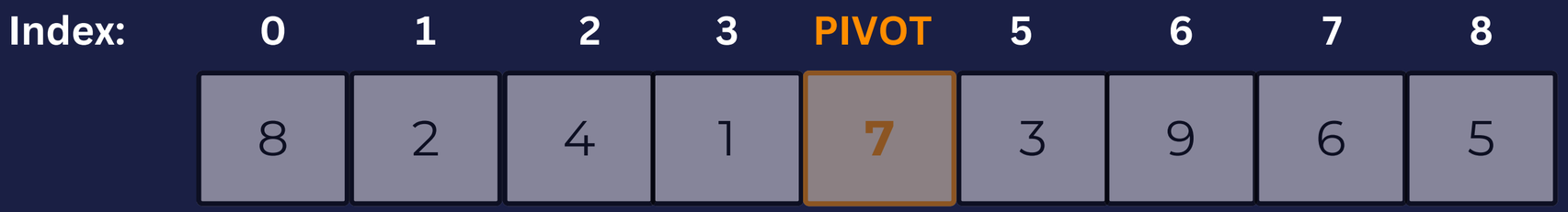
Right

Left

Right


```
return pivot + quickSort(right)
```

It will replace pivot with the current value (1)



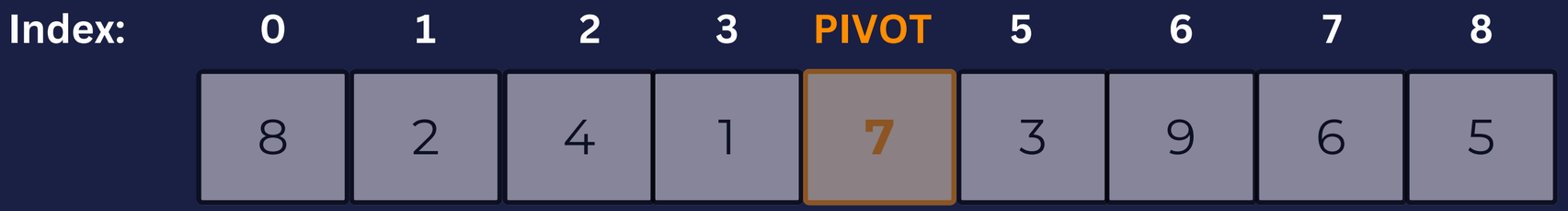

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```

Left

Right


```
return [1] + quickSort(right)
```

Now it will run the Right side, and create a new pivot

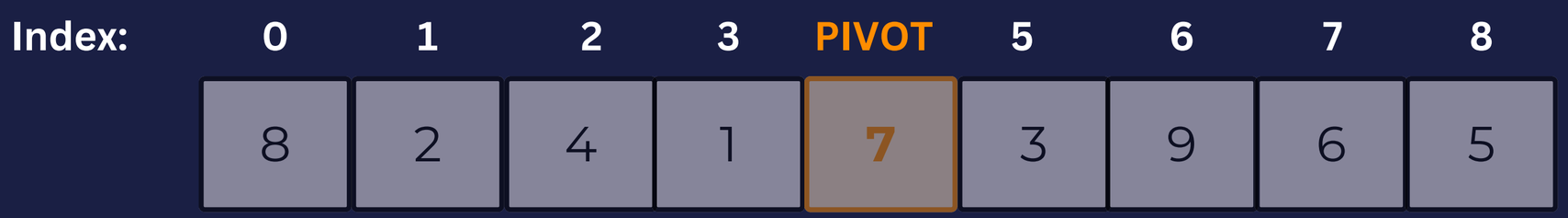



```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```


```
return [1]  + quickSort(right)
 Left                Right
```

The new pivot is now 3.

And now we make a new Left and a new Right

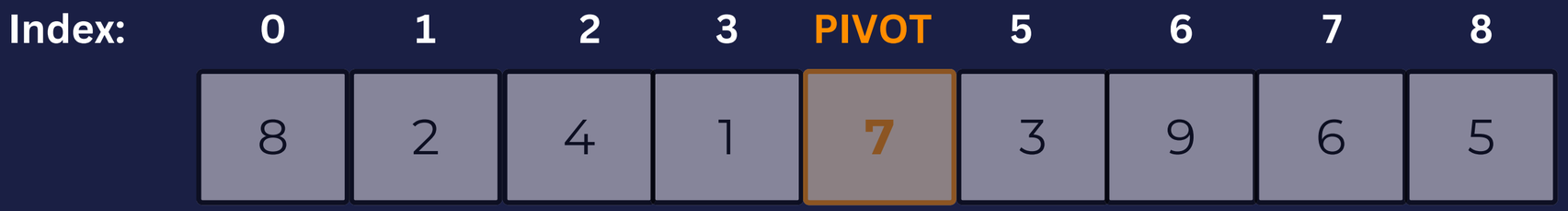



```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

Left Right

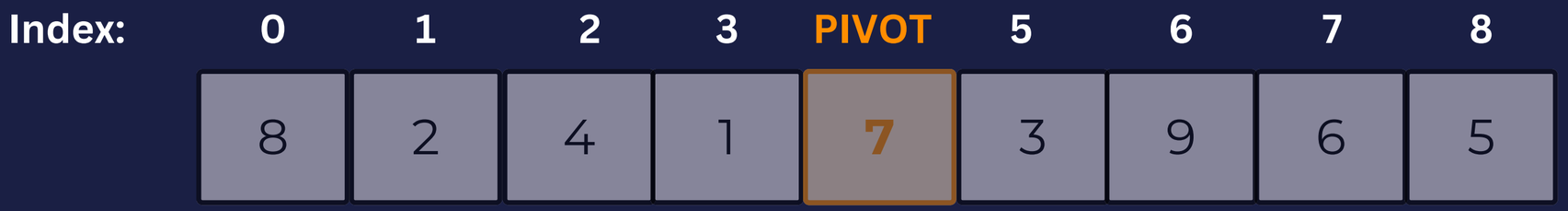
And we run check the values again




```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```

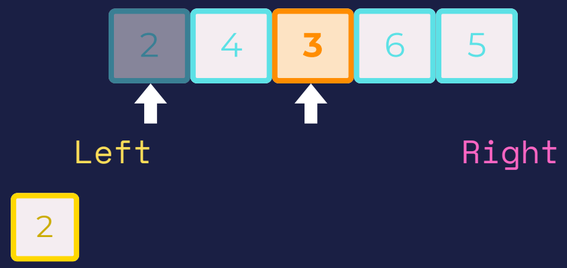

```
return [1]  + quickSort(right)
 Left              Right
```

And we run check the values again

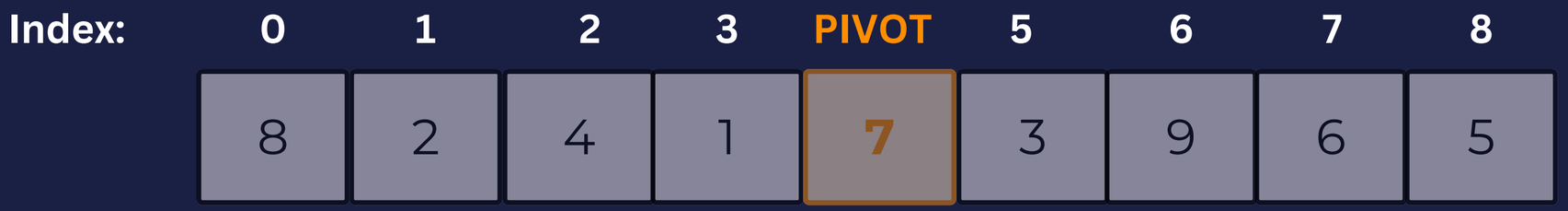



```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```



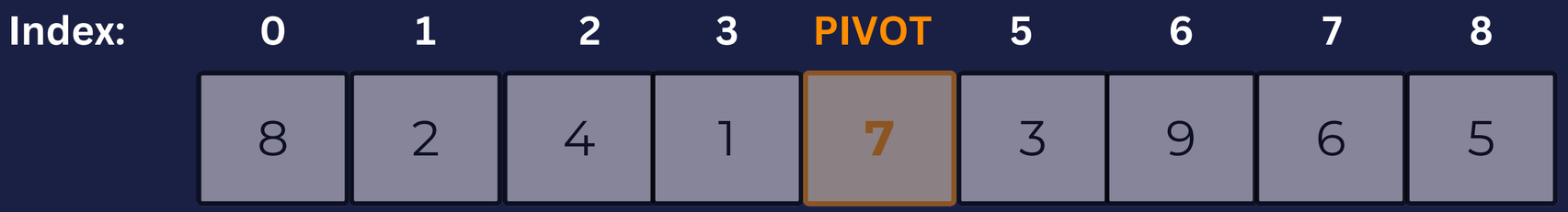
And we run check the values again




```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

And we run check the values again



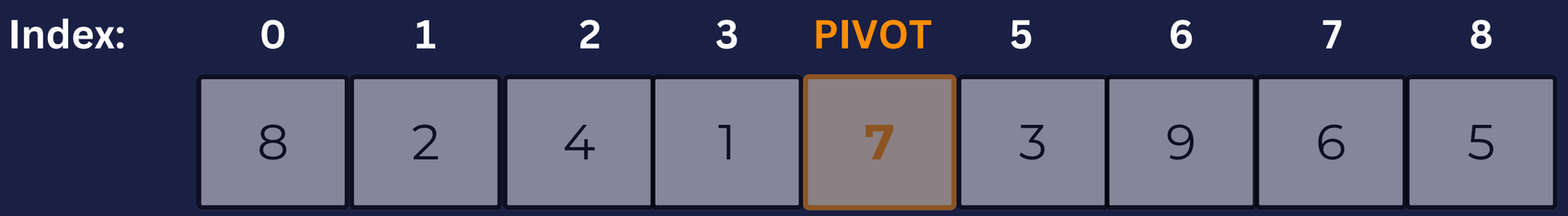

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

Left

Right

And we run check the values again



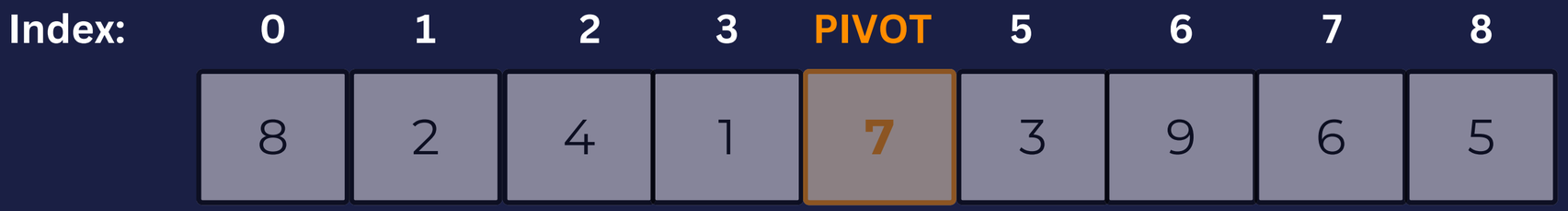

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

Left

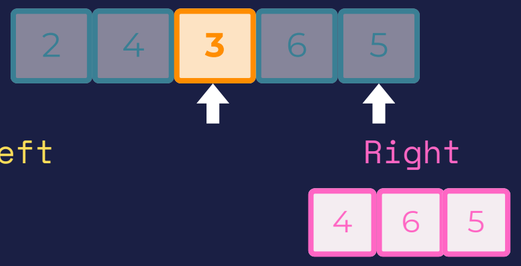
Right

And we run check the values again

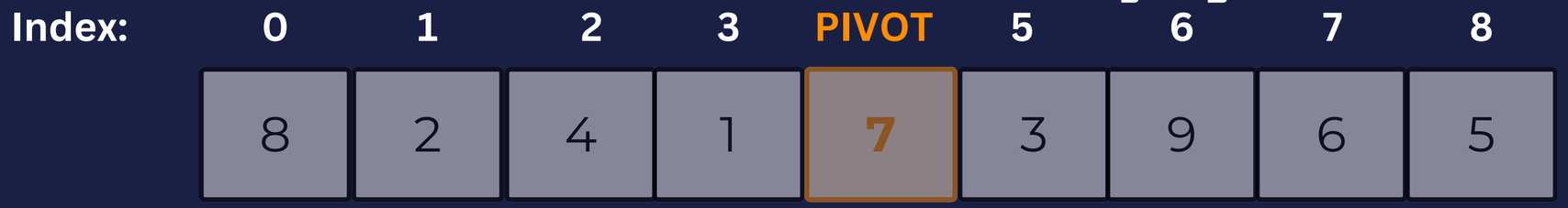



```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```



And now we have another completed run, and we return the functions recurisvely again




```
return quickSort(left) + pivot +  quickSort(right)
```

Left

Right


```
return [1] + quickSort(right)
```

Left

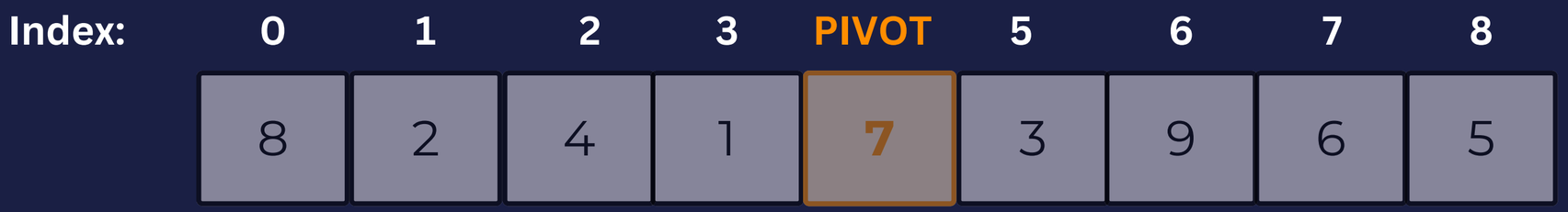
Right

Left

Right


```
return quickSort(left) + pivot + quickSort(right)
```

Left returns and array of 1 value



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

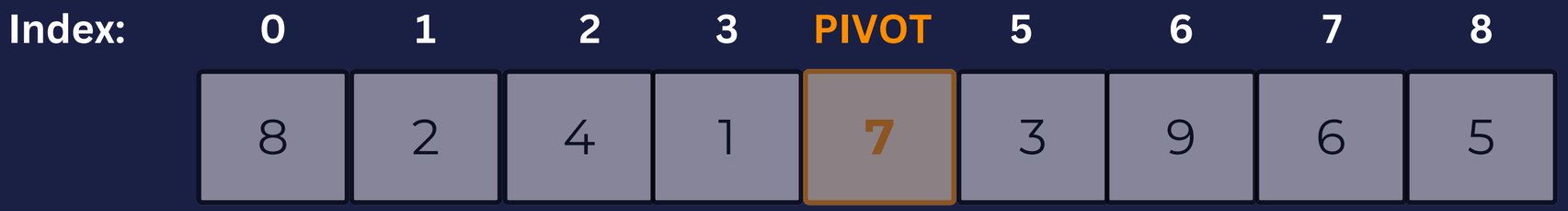
Right

Left

Right

return quickSort(left) + pivot + quickSort(right)

Left returns and array of 1 value



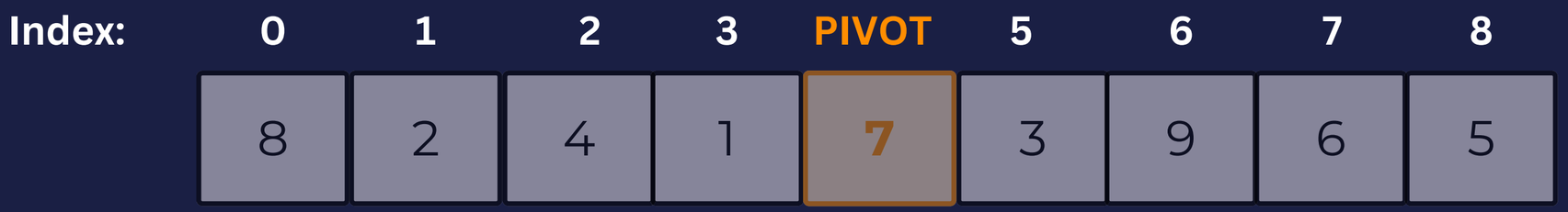

```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

Right


```
return [2] + pivot + quickSort(right)
```

And we return the array that is the pivot.



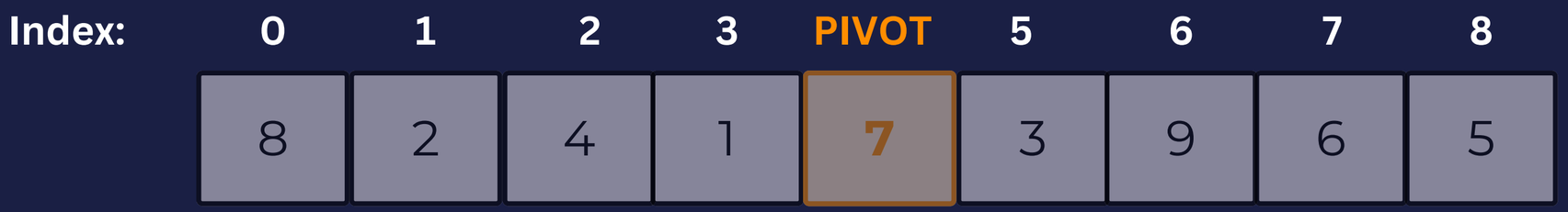

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

Right


```
return [2] + pivot + quickSort(right)
```

And we return the array that is the pivot.



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

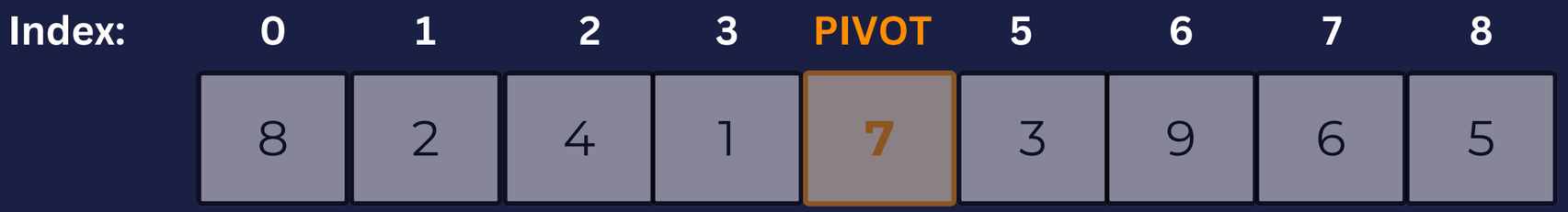
Left

Right

Right

return [2] + [3] + quickSort(right)

And then we run the Right



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

Right

return [2] + [3] + quickSort(right)

And then we run the Right

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```

2	4	1	3	6	5

```
return [1]  + quickSort(right)
 Left              Right
```

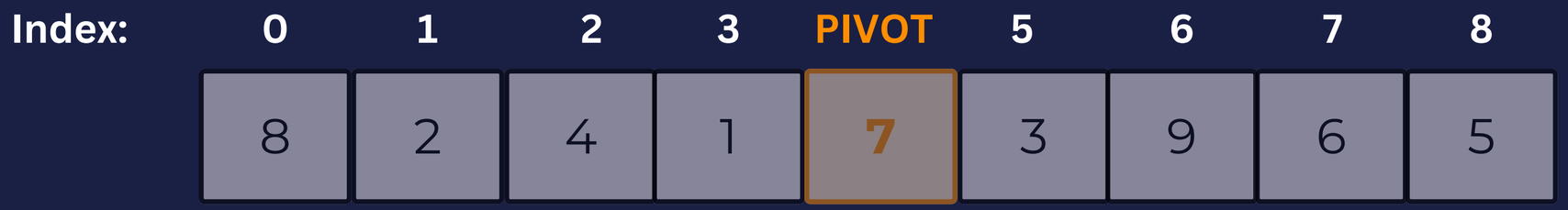
8	9

2	4	3	6	5

```
return [2] + [3] + quickSort(right)
                      Right
```

4	6	5

We create a new pivot and a new left and right



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

Right

Left

Right

And we check the values again

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```

2	4	1	3	6	5

```
return [1]  + quickSort(right)
 Left              Right
```

8	9

2	4	3	6	5

```
return [2] + [3] + quickSort(right)
                     Right
```

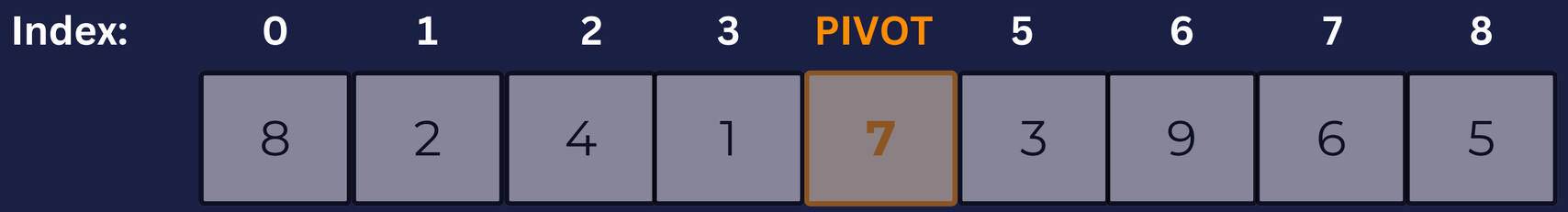
4	6	5

Left

4

Right

And we check the values again




```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```


```
return [1]  + quickSort(right)
 Left              Right
```


```
return [2] + [3] + quickSort(right)
                          Right
```

Left

Right

And we return the functions again

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```


```
return [2] + [3] + quickSort(right)
                      Right
```

Left

Right

```
return quickSort(left) + pivot + quickSort(right)
```

And choose a new pivot and a new left and right

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

return quickSort(left) + pivot + quickSort(right)

Left

Right

2	4	1	3	6	5

8	9

return [1] + quickSort(right)

Left

Right

2	4	3	6	5

return [2] + [3] + quickSort(right)

Right

4	6	5

Left

Right

4	5

Left

Right

return quickSort(left)
+ pivot + quickSort(right)

And Check the values again

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

`return quickSort(left) + pivot + quickSort(right)`

Left — Right

2	4	1	3	6	5

`return [1] + quickSort(right)`

Left — Right

8	9

2	4	3	6	5

`return [2] + [3] + quickSort(right)`

Right

4	6	5

Left — Right

4	5

Left

Right

`return quickSort(left) + pivot + quickSort(right)`

And Check the values again

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

return quickSort(left) + pivot + quickSort(right)

Left

Right

2	4	1	3	6	5

8	9

return [1] + quickSort(right)

Left

Right

2	4	3	6	5

return [2] + [3] + quickSort(right)

Right

4	6	5

Left

Right

4	5

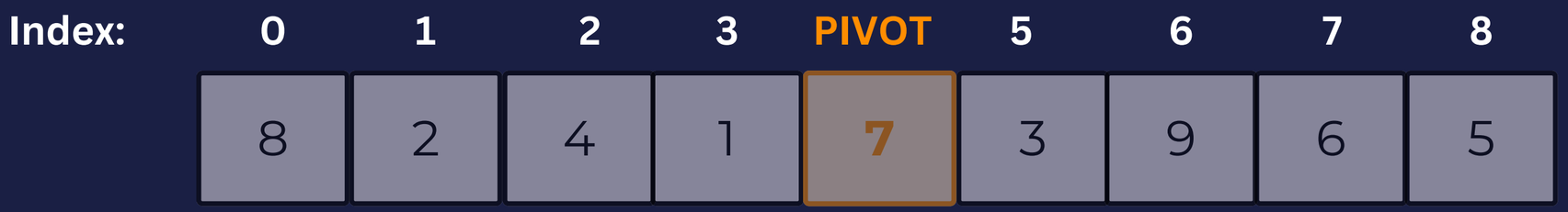
Left

Right

5

return quickSort(left)
+ pivot + quickSort(right)

And return the values.




```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```


```
return quickSort(left) + pivot + quickSort(right)
```

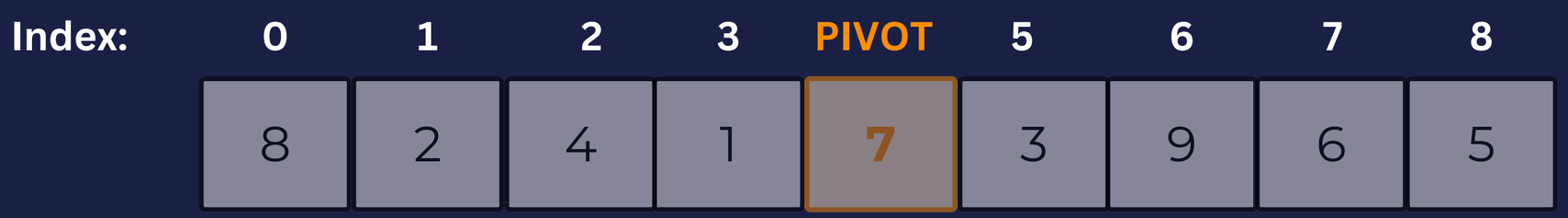

```
return [2] + [3] + quickSort(right)
                      Right
```

Left

Right


```
return quickSort(left)
+ pivot +quickSort(right)
```

Left returns an empty array, so we move on to pivot



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return pivot + quickSort(right)

return [2] + [3] + quickSort(right)

Right

Left

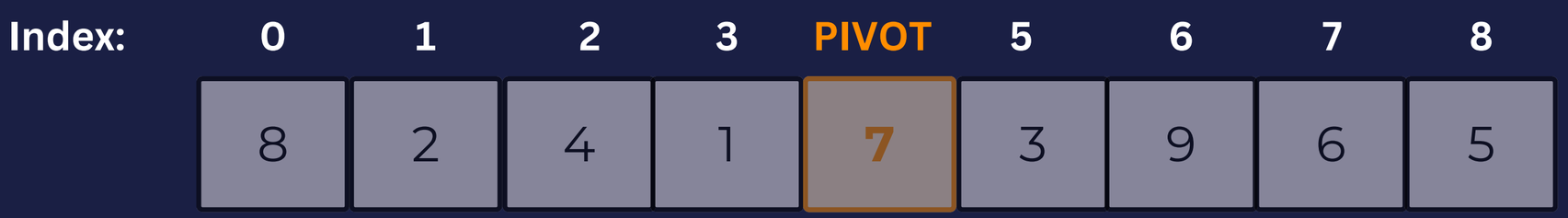
Right

Left

Right

return quickSort(left)
+ pivot + quickSort(right)

Left returns an empty array, so we move on to pivot



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [4] + quickSort(right)

return [2] + [3] + quickSort(right)

Right

Left

Right

Left

Right

return quickSort(left)
+ pivot + quickSort(right)

Right returns it's array

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

return quickSort(left) + pivot + quickSort(right)

Left

Right

2	4	1	3	6	5

8	9

return [1] + quickSort(right)

Left

Right

return [4] + quickSort(right)

2	4	3	6	5

return [2] + [3] + quickSort(right)

Right

Left

Right

4	6	5

5

Left

4	5

Right

return quickSort(left)

+ pivot + quickSort(right)

Right returns it's array

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

return quickSort(left) + pivot + quickSort(right)

Left

Right

2	4	1	3	6	5

8	9

return [1] + quickSort(right)

Left

Right

return [4] + [5]

2	4	3	6	5

return [2] + [3] + quickSort(right)

Right

Left

Right

4	6	5

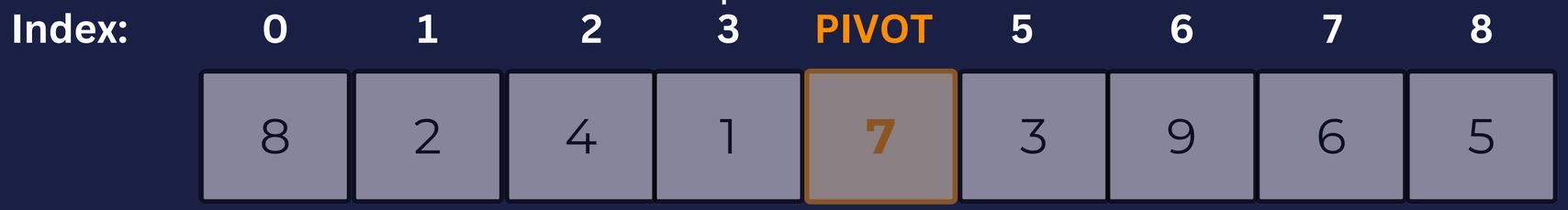
4	5

5

return quickSort(left)

+ pivot + quickSort(right)

and [4] + [5] is now [4, 5], and is what will get returned in the previous return statement.



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

Right

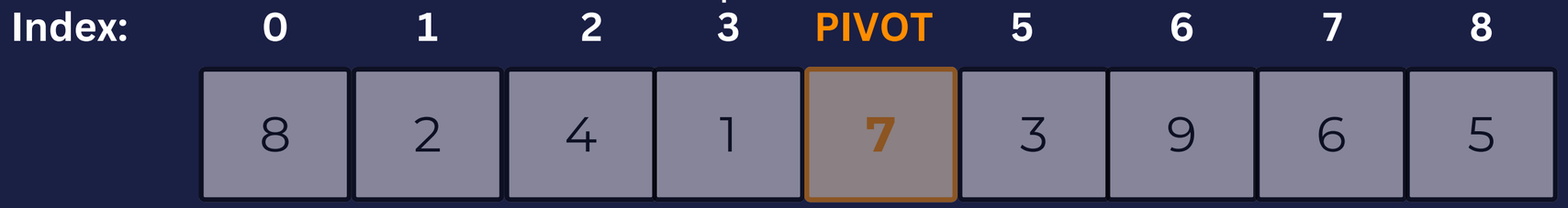
Left

Right

return [4] + [5]

return quickSort(left)
+ pivot + quickSort(right)

and [4] + [5] is now [4, 5], and is what will get returned in the previous return statement.



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

Right

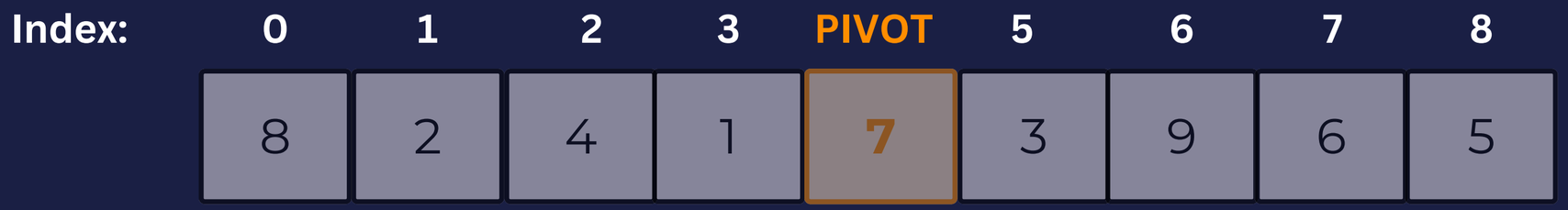
Left

Right

return [4, 5]

return quickSort(left)
+ pivot + quickSort(right)

It returns and replaces the function that called it.




```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```


```
return [1]  + quickSort(right)
 Left              Right
```


```
return [2] + [3] + quickSort(right)
                      Right
```

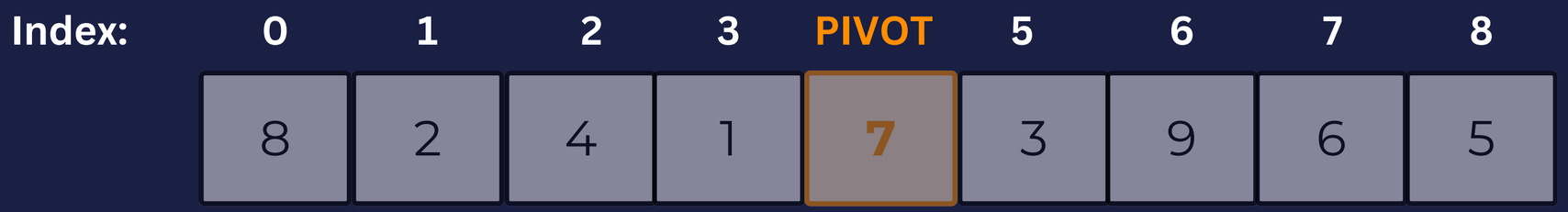
Left

Right


```
return [4, 5]
```

```
return quickSort(left)
+ pivot + quickSort(right)
```

It returns and replaces the function that called it.



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

Right

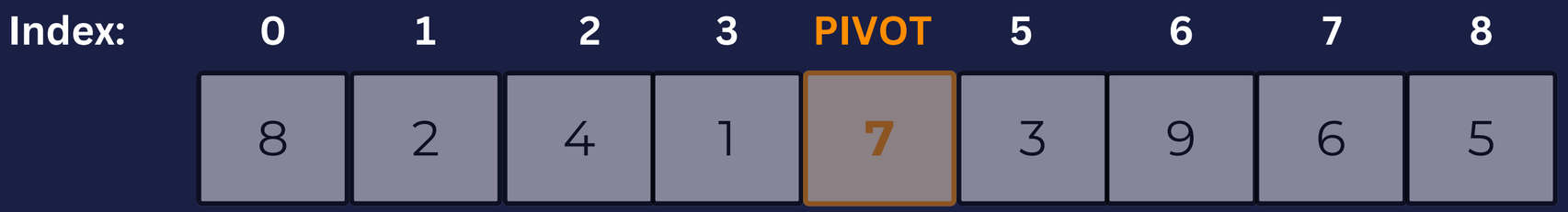
Left

Right

return [4, 5]

+ pivot + quickSort(right)

And we add the pivot




```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```


```
return [1]  + quickSort(right)
 Left              Right
```


```
return [2] + [3] + quickSort(right)
                          Right
```

Left

Right

```
return [4, 5]
+ pivot + quickSort(right)
```

And we add the pivot

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return quickSort(left) + pivot +  quickSort(right)
         Left                       Right
```

2	4	1	3	6	5

```
return [1]  + quickSort(right)
 Left              Right
```

8	9

2	4	3	6	5

```
return [2] + [3] + quickSort(right)
                      Right
```

4	6	5

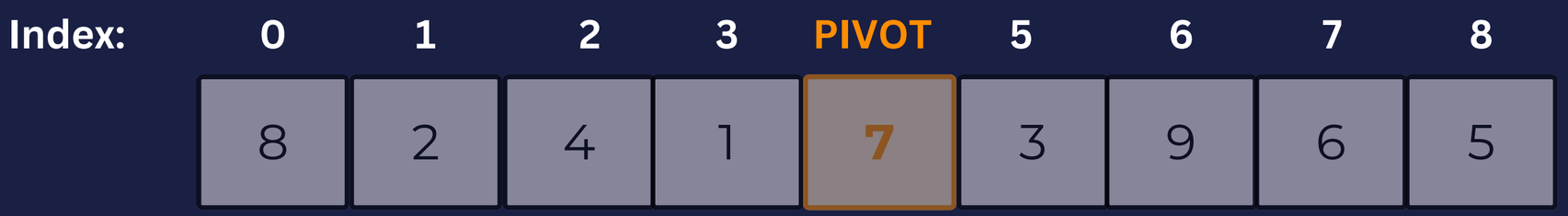
Left

4	5

Right

```
return [4, 5]
+ [6] + quickSort(right)
```

And Right is empty so it returns an empty array




```
return quickSort(left) + pivot +  quickSort(right)
         Left                          Right
```


```
return [1]  + quickSort(right)
 Left              Right
```

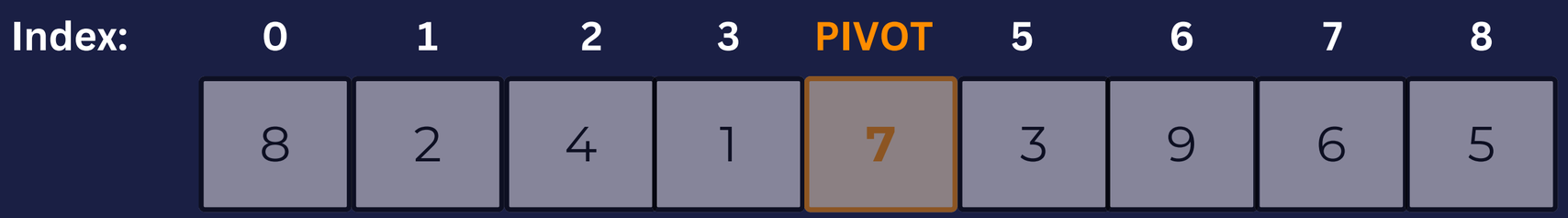

```
return [2] + [3] + quickSort(right)
                          Right
```

Left

Right

```
return [4, 5] + [6] +
  quickSort(right)
```

And [4, 5] + [6] turns into [4, 5, 6] and returns



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

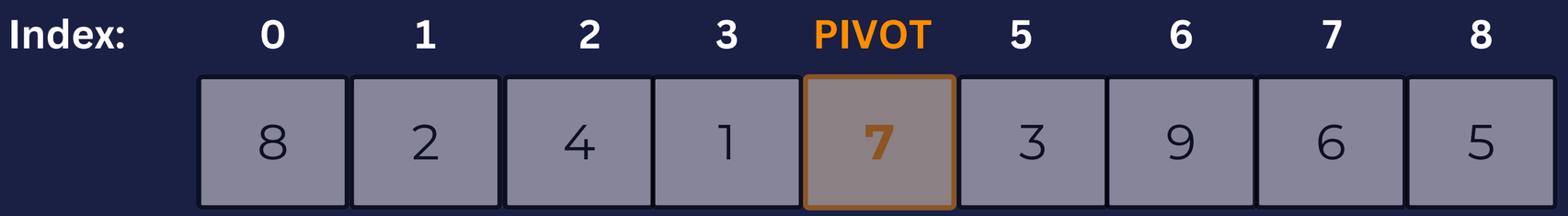
Right

Left

Right

return [4, 5] + [6]

And [4, 5] + [6] turns into [4, 5, 6] and returns



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

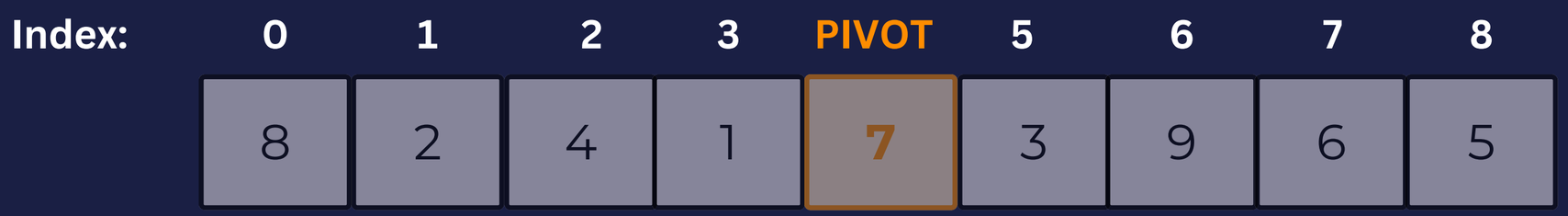
Right

return [2] + [3] + quickSort(right)

Right

return [4, 5, 6]

Now that this quickSort is done, it returns [4, 5, 6]



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

Left

Right

return [2] + [3] + quickSort(right)

Right

return [4, 5, 6]

And replaces `quickSort(right)` with `[4, 5, 6]`


```
return quickSort(left) + pivot +  quickSort(right)
         Left                      Right
```

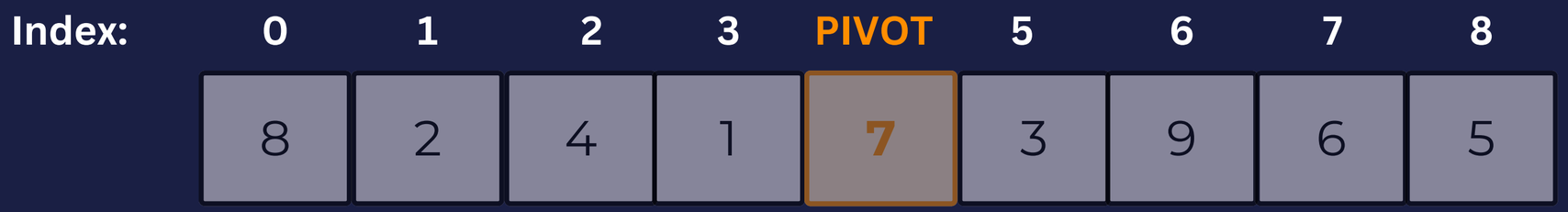

```
return [1]  + quickSort(right)
 Left              Right
```


```
return [2] + [3] + quickSort(right)
```

```
return [4, 5, 6]
```

And [2] + [3] + [4, 5, 6] becomes [2, 3, 4, 5, 6]

and returns



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

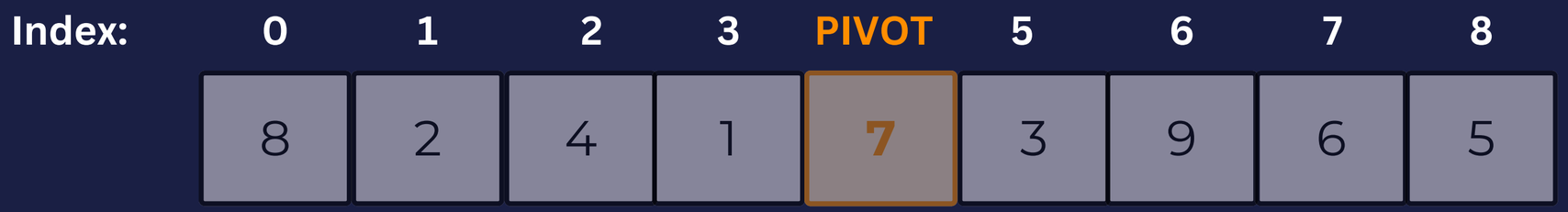
Left

Right

return [2] + [3] + [4, 5, 6]

And [2] + [3] + [4, 5, 6] becomes [2, 3, 4, 5, 6]

and returns



return quickSort(left) + pivot + quickSort(right)

Left

Right

return [1] + quickSort(right)

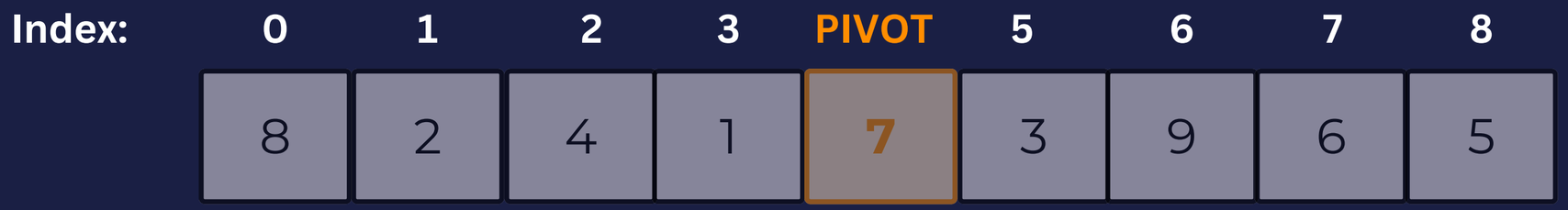
Left

Right

return [2, 3, 4, 5, 6]

And [2] + [3] + [4, 5, 6] becomes [2, 3, 4, 5, 6]

and returns




```
return quickSort(left) + pivot +  quickSort(right)
```

Left

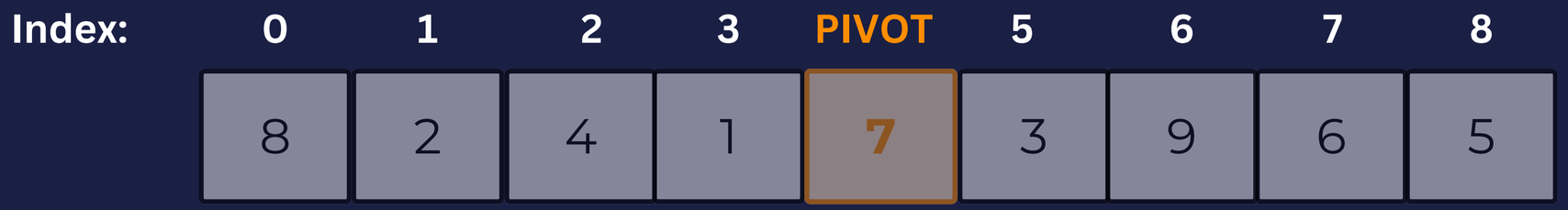

```
return [1]  + quickSort(right)
```

Right


```
return [2, 3, 4, 5, 6]
```

And [2, 3, 4, 5, 6]

replaces quickSort(right)



return quickSort(left) + pivot + quickSort(right)

Left

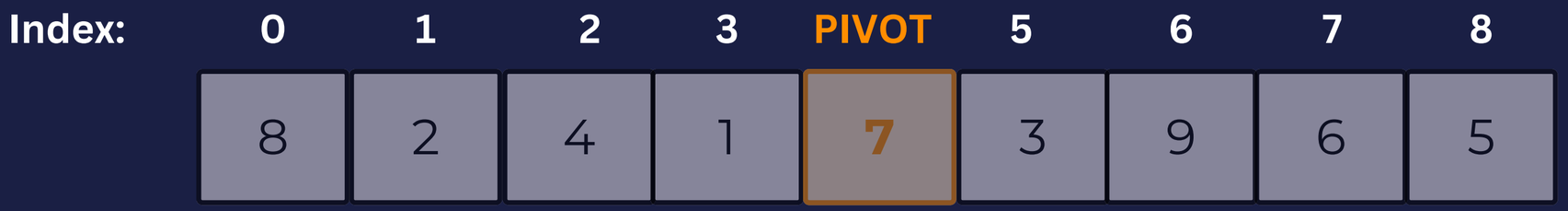
Right

return [1] + quickSort(right)

return [2, 3, 4, 5, 6]

And [2, 3, 4, 5, 6]

replaces quickSort(right)



return quickSort(left) + pivot + quickSort(right)

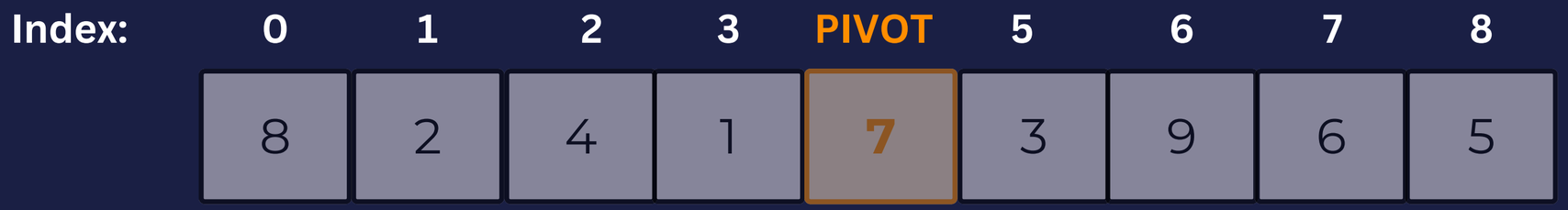
Left

Right

return [1] + [2, 3, 4, 5, 6]

And [1] + [2, 3, 4, 5, 6]

becomes [1, 2, 3, 4, 5, 6] and returns.



return quickSort(left) + pivot + quickSort(right)

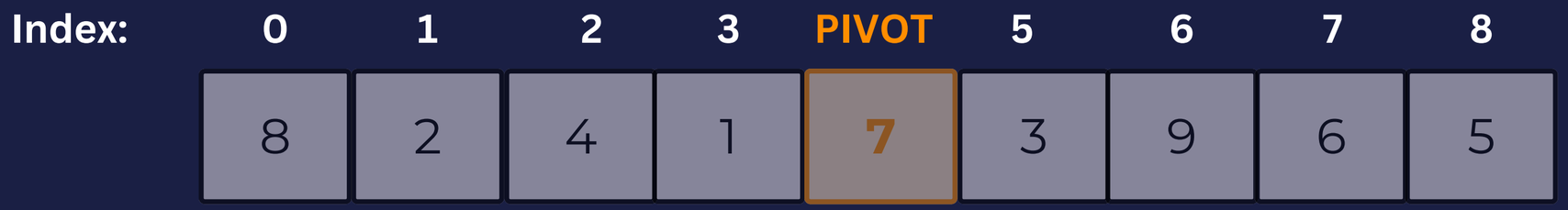
Left

Right

return [1] + [2, 3, 4, 5, 6]

And [1] + [2, 3, 4, 5, 6]

becomes [1, 2, 3, 4, 5, 6] and returns.




```
return quickSort(left) + pivot +  quickSort(right)
```

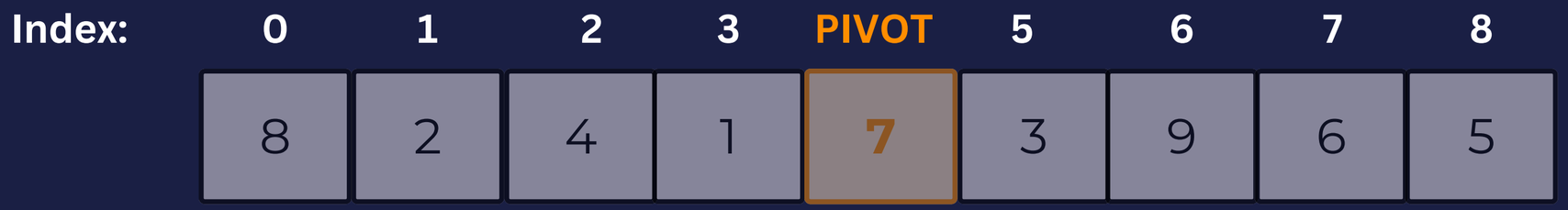
Left

Right

return [1, 2, 3, 4, 5, 6]

And [1] + [2, 3, 4, 5, 6]

becomes [1, 2, 3, 4, 5, 6] and returns.

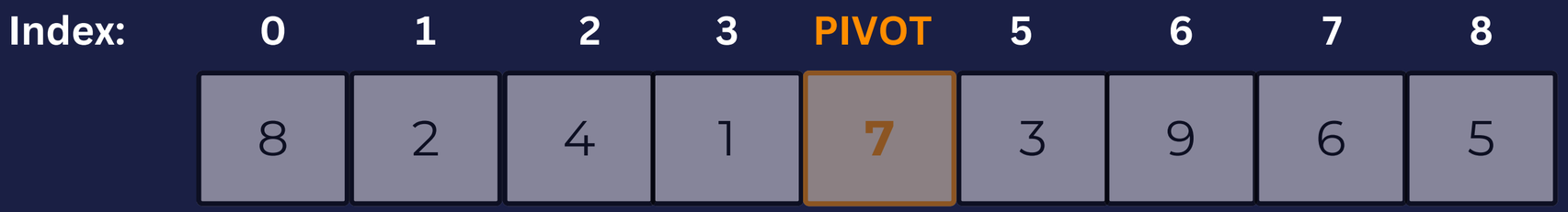


return quickSort(left) + pivot + quickSort(right)

return [1, 2, 3, 4, 5, 6]

Right

And it replaces `quickSort(left)`



`return quickSort(left) + pivot + quickSort(right)`

`return [1, 2, 3, 4, 5, 6]`

Right

And it assigns the value at pivot


```
return [1, 2, 3, 4, 5, 6] + pivot +  quickSort(right)
```

Right

And it assigns the value at pivot

return [1, 2, 3, 4, 5, 6] + [7] + quickSort(right)

Right

And it assigns the value at pivot

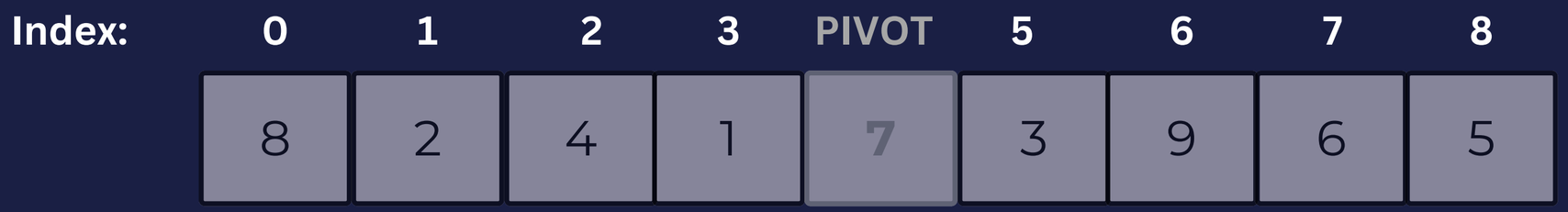
Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right

8	9

And we do the same to the Right side




```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

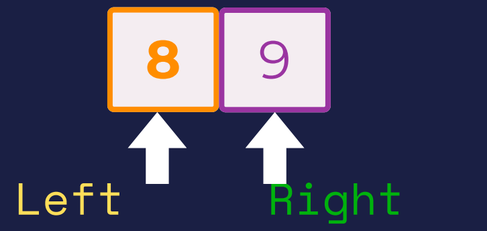
Right

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right



And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right


```
return quickSort(left) + pivot + quickSort(right)
```

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
                                        Right
```


```
return ~~quickSort(left)~~ + pivot + quickSort(right)
```

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right


```
return pivot + quickSort(right)
```

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
                                        Right
```


```
return [8] + quickSort(right)
```

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
```

Right


```
return [8] + [9]
```

And we do the same to the Right side

Index:	0	1	2	3	PIVOT	5	6	7	8
	8	2	4	1	7	3	9	6	5

return [1, 2, 3, 4, 5, 6] + [7] + quickSort(right)

Right

return [8, 9]

And we do the same to the Right side

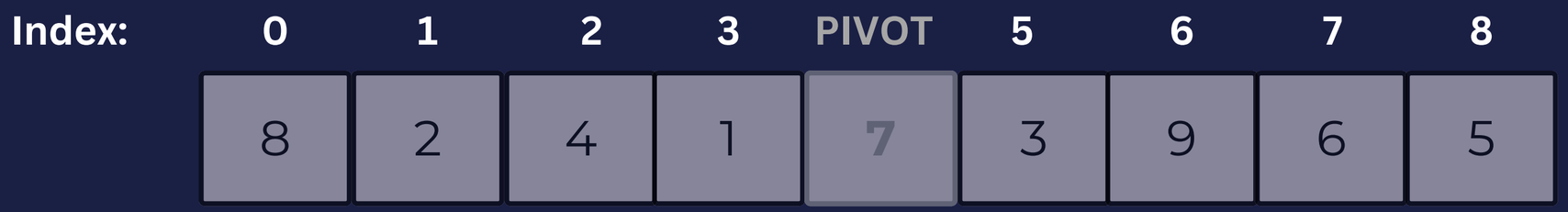

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
                                          Right
```

```
return [8, 9]
```

And we do the same to the Right side

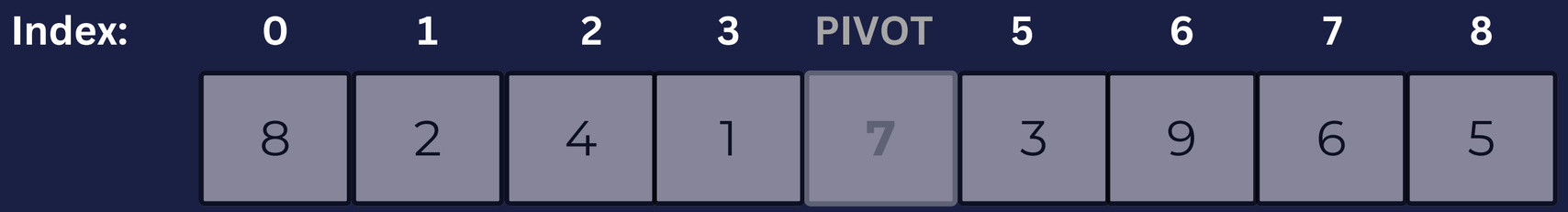

```
return [1, 2, 3, 4, 5, 6] + [7] +    quickSort(right)
                                       return [8, 9]
```

And we do the same to the Right side



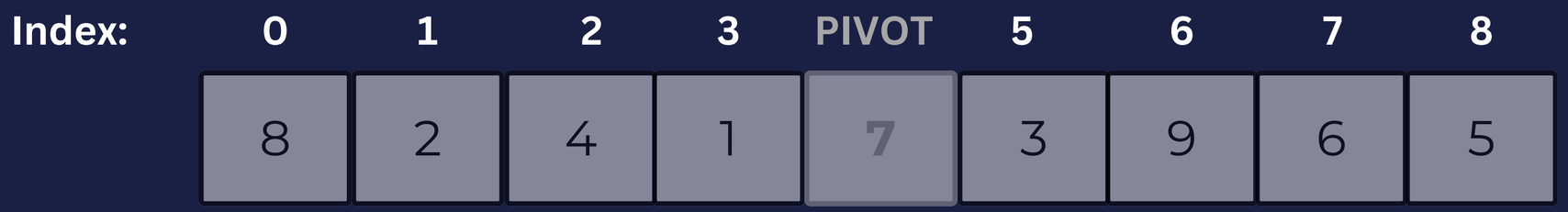

```
return [1, 2, 3, 4, 5, 6] + [7] + [8, 9]
```

And we combine [1, 2, 3, 4, 5, 6] + [7] + [8, 9] and return the finished array



return [1, 2, 3, 4, 5, 6] + [7] + [8, 9]

And we combine [1, 2, 3, 4, 5, 6] + [7] + [8, 9] and return the finished array



return [1, 2, 3, 4, 5, 6, 7, 8, 9]

And here is our sorted array!

Index:	0	1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8	9

THE MIDDLE PIVOT

The Three Main Steps:

- Finding the pivot value
- Sorting based on the pivot
  - Numbers less than the pivot are put to the left
  - Numbers greater than the pivot are put to the right
- Sorting the left and right side of the pivot number

THE PSEUDOCODE

```
function QUICK_SORT(array):
    IF length(array) <= 1:
        RETURN array

    pivot = (firstIndex + lastIndex) // 2
    left = (for num in array if num is < pivot)
    middle = (for num in array if num is == pivot)
    right = (for num in array if num is > pivot)

    // Recursive return statement
    return QUICK_SORT(left) + middle + QUICK_SORT(right)
```

EXAMPLES REPLIT! PLEASE GO TO:

HTTPS://REPLIT.COM/@RIKKIEHRHART/GRABABYTE

UP NEXT

SPRING BREAK!

Mar 26 - Breadth-First Search (BFS)
Apr 2 - Depth-First Search (DFS)
Apr 9 Hashing
Apr 16 - Dijkstra's Algorithm
Apr 23 - Dynamic Programming (Knapsack Problem)
Apr 30 - Union-Find
May 7 - Kruskal's Algorithm
May 14 - Prim's Algorithm

Questions? - rikki.ehrhart@g.ausitncc.edu

If you'd like the opportunity to run a Grab a Byte algorithm workshop, please let me know!