

Implementing QuickSort

CSE 204 – Week 13

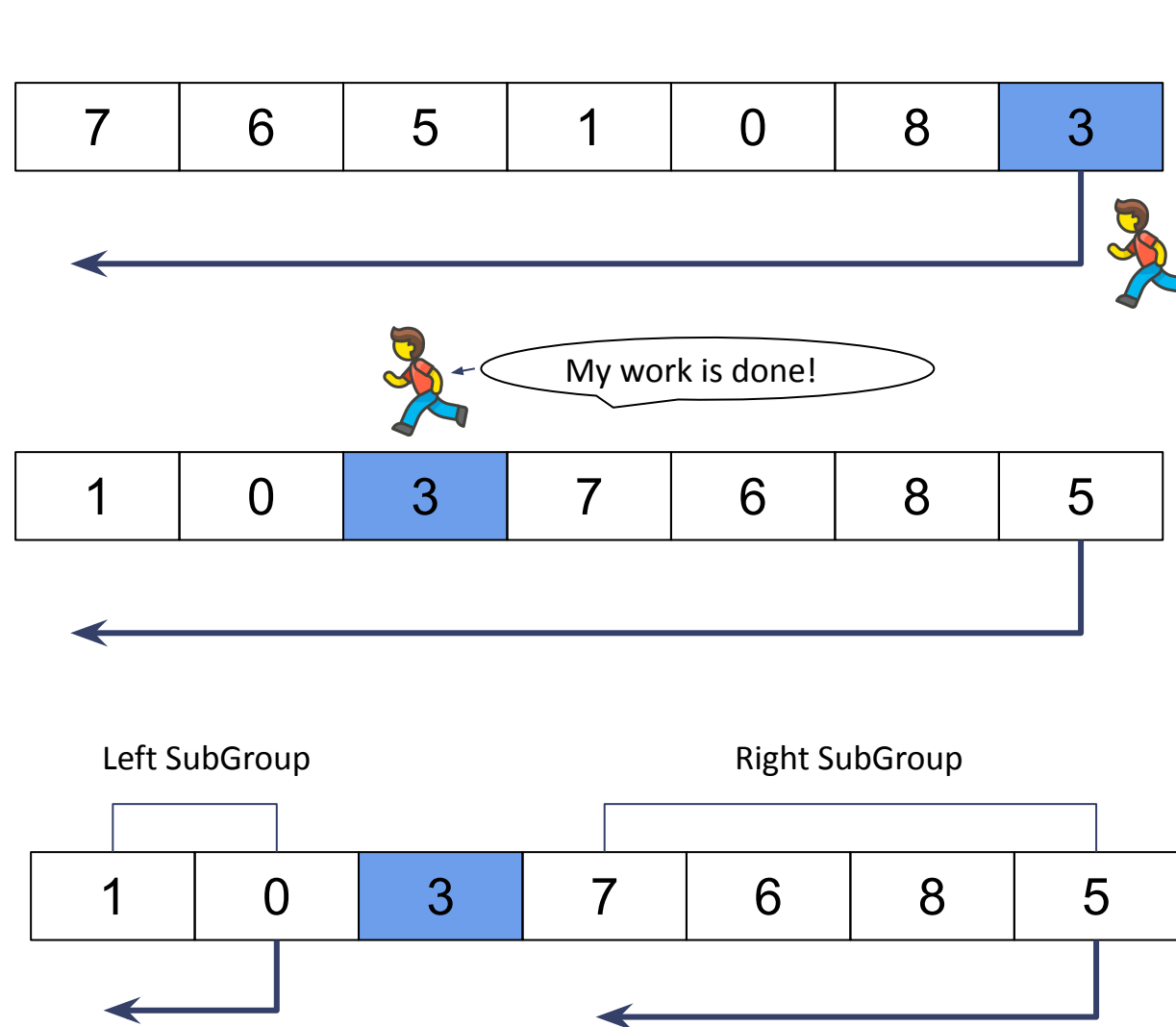
LEC RAIYAN RAHMAN


Dept of CSE, MIST

raihan@cse.mist.ac.bd



How QuickSort Works - A Metaphor



 Current Runner

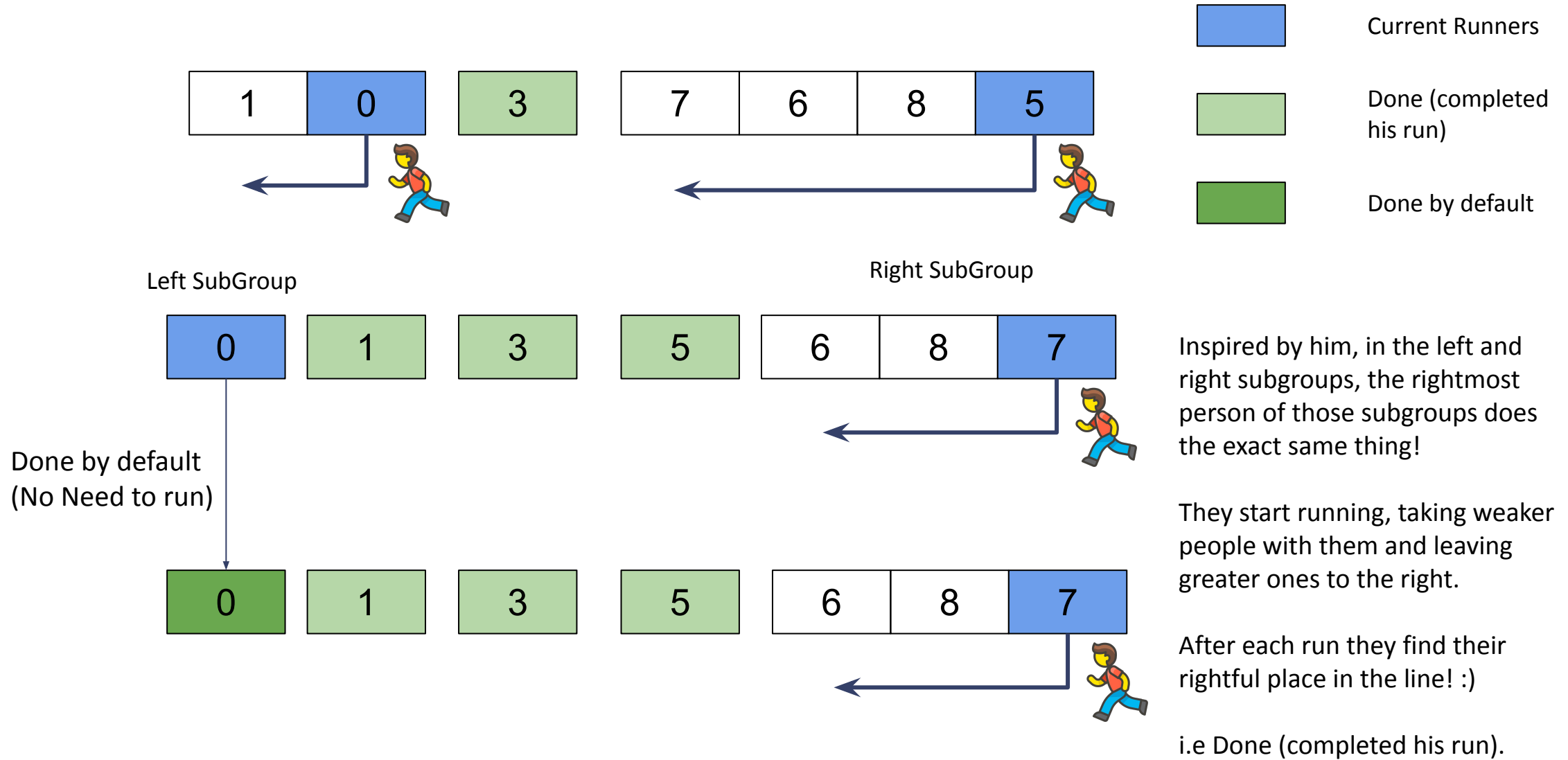
Consider, people are standing in a line. The **rightmost** guy decides to run from his end to the other end. While running, he can take only other guys who are weaker than him (they have lesser value).

He takes the ones weaker than him and sits at the rightmost position of all those weaker ones, thus creating a **left subgroup** on the left of him.

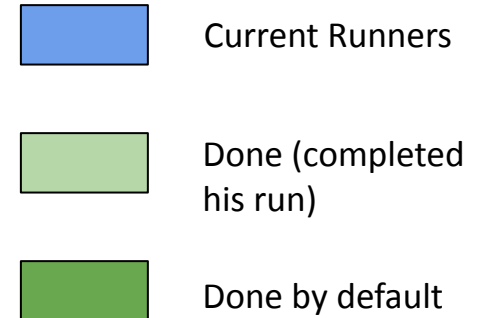
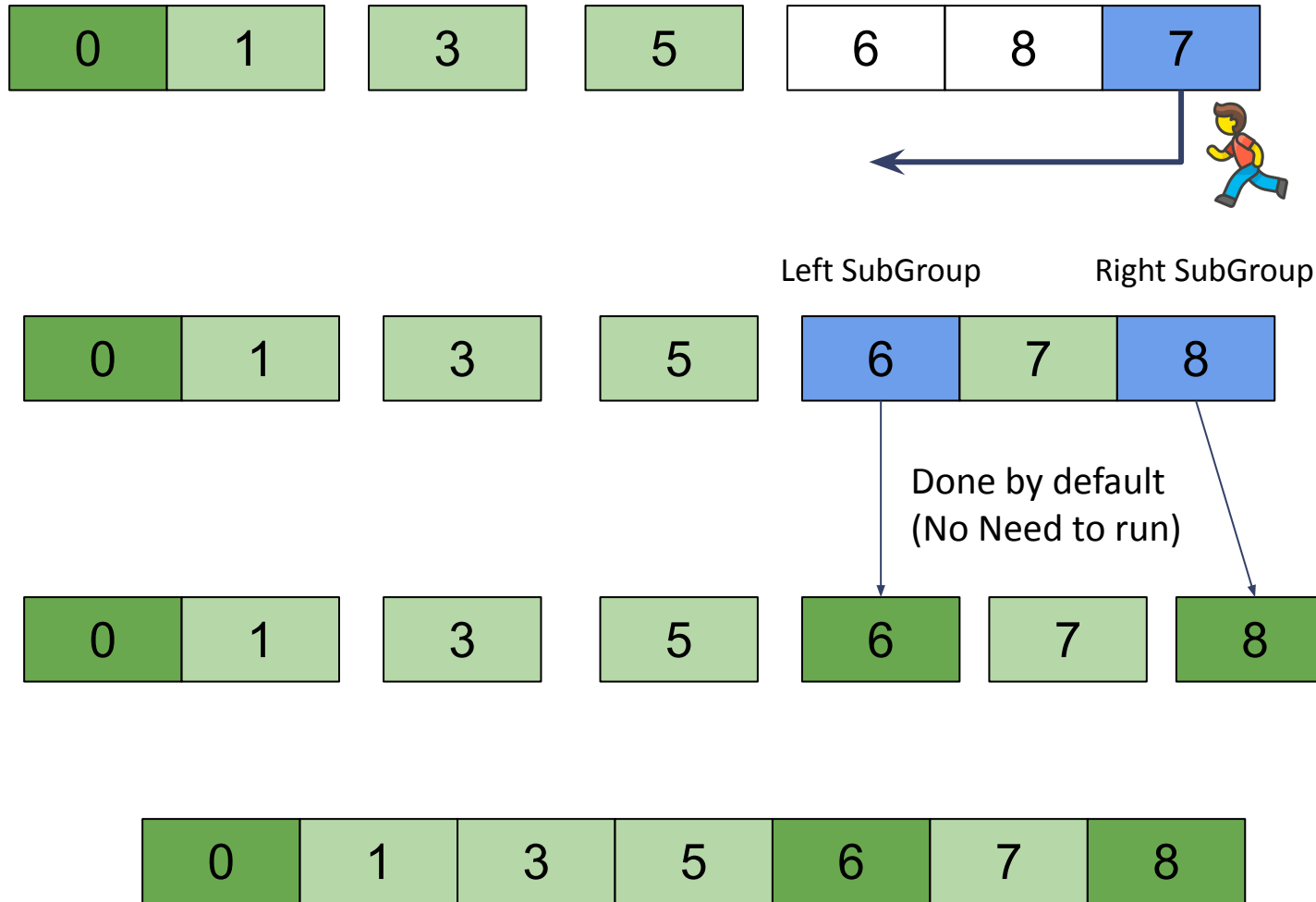
The ones greater than him will be on the right to that person, creating a **right subgroup**.

And he has found his place, will not moving ever again!

How QuickSort Works - A Metaphor



How QuickSort Works - A Metaphor



Or if they're part of a subgroup where they're all alone, they don't run. They've already found their place.

They didn't have to run to find their place, other runners did it for them in the process.

i.e Done by default (No Need to run)

Look! We have sorted the runners!

From Metaphor to the Algorithm



Current Runners = Current Pivot



Done (completed his run) = Sorted

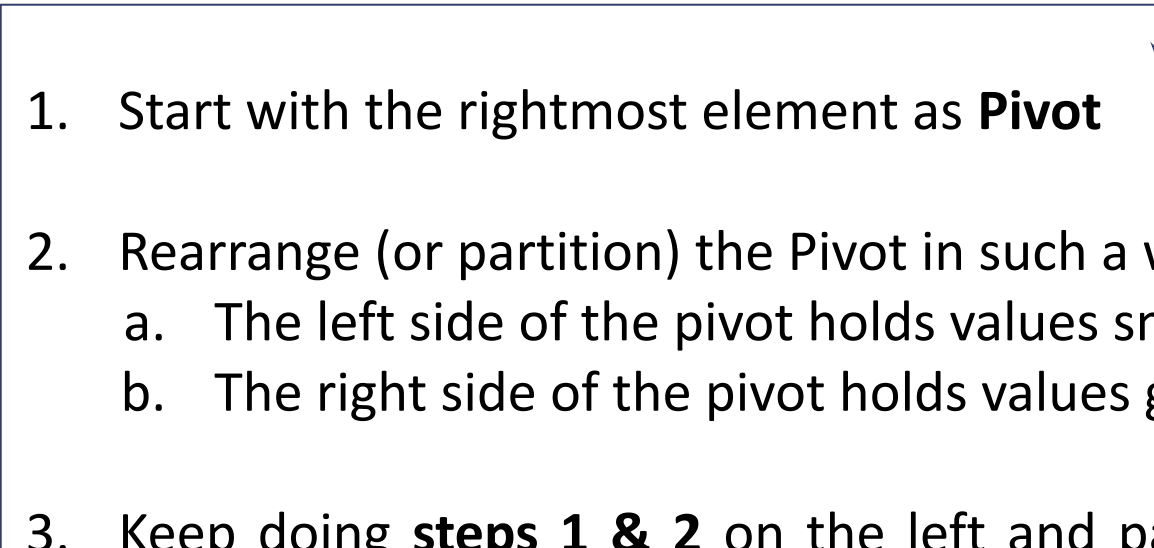


Done by default = Sorted

Left Subgroup = Left Partition, contains values smaller than the pivot. To be Quick-sorted again.

Right Subgroup = Right Partition, contains values greater than the pivot. To be Quick-sorted again.

What we Need to do (The Algorithm)

- 
1. Start with the rightmost element as **Pivot**
 2. Rearrange (or partition) the Pivot in such a way that-
 - a. The left side of the pivot holds values smaller than the pivot (left partition).
 - b. The right side of the pivot holds values greater than the pivot (right partition).
 3. Keep doing **steps 1 & 2** on the left and partitions again and again **till all the partitions reduce to just one element.**

But how do we do step 2 (the partitioning)?

7	6	5	1	0	8	3
---	---	---	---	---	---	---



Let's initialize a variable, `int A = (current subgroups leftmost index) - 1`



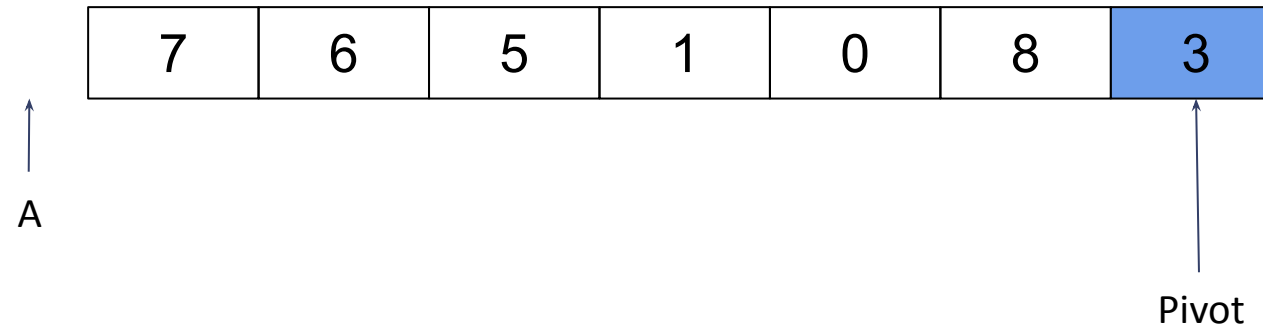
Pivot

So, When leftmost index = 0 (first runner's case), variable 'A' will be initialized to -1.

Step 2A

1. Store the value of the pivot (rightmost value of the group/subgroup).
2. Initialize another variable with the (subgroup's leftmost index - 1).

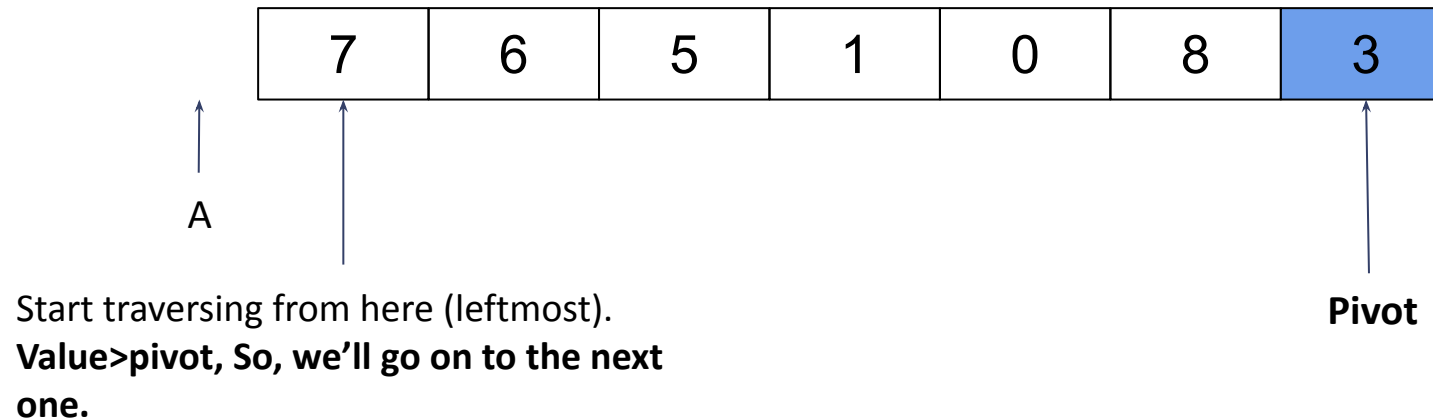
But how do we do step 2 (the partitioning)?



Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the element > pivot, go to the next value for comparison.
 - b. if the element < pivot-
 - i. increase value of the variable 'A' (do A++).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

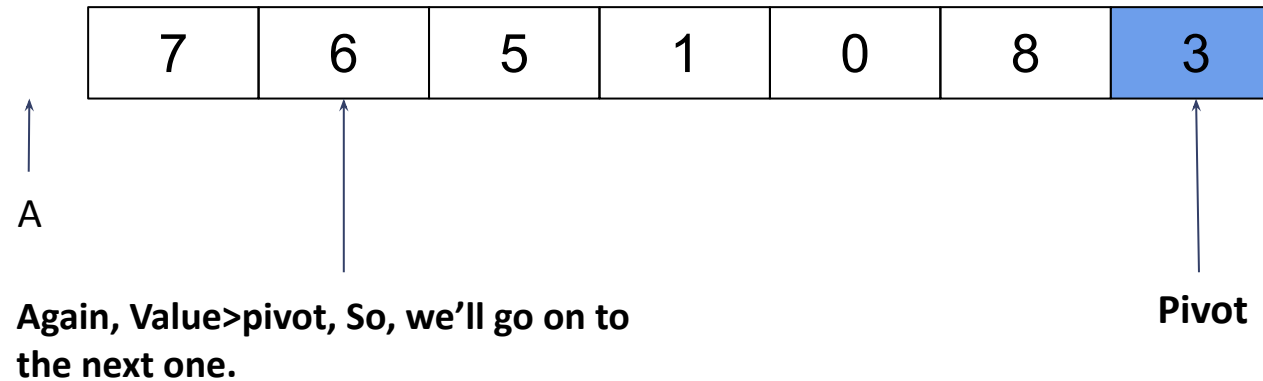
But how do we do step 2 (the partitioning)?



Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the element > pivot, go to the next value for comparison.
 - b. if the element < pivot-
 - i. increase value of the variable 'A' (do A++).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

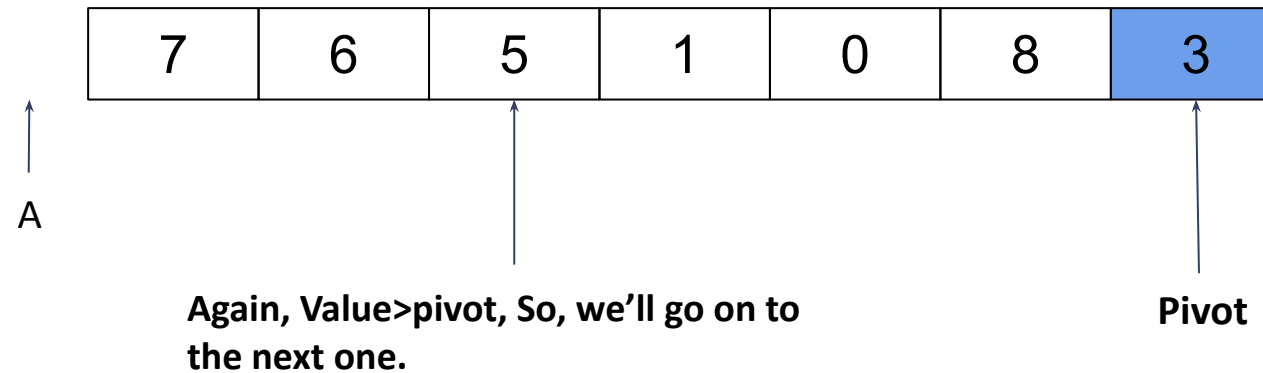
But how do we do step 2 (the partitioning)?



Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the element > pivot, go to the next value for comparison.
 - b. if the element < pivot-
 - i. increase value of the variable 'A' (do A++).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

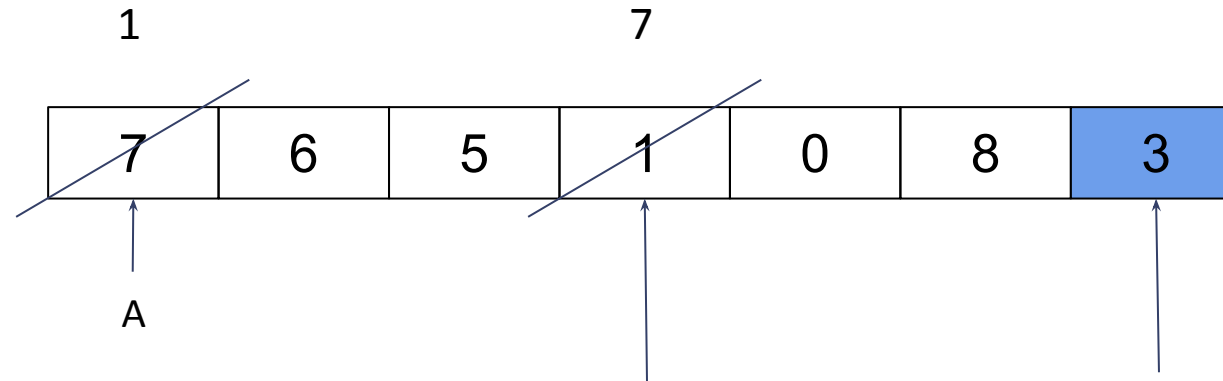
But how do we do step 2 (the partitioning)?



Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the element > pivot, go to the next value for comparison.
 - b. if the element < pivot-
 - i. increase value of the variable 'A' (do A++).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

But how do we do step 2 (the partitioning)?



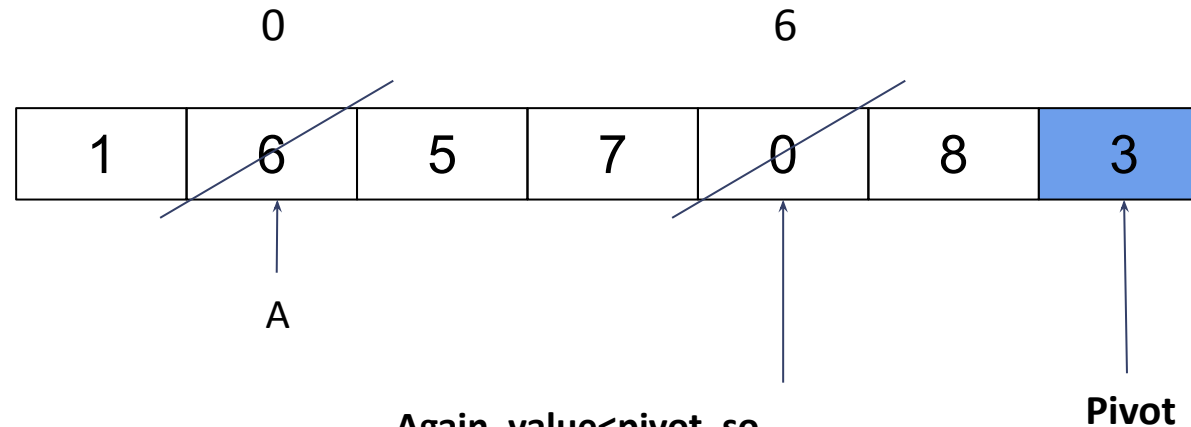
For the first time, $\text{value} < \text{pivot}$, so-

1. Increase value of A.
2. Swap current val with value at index 'A'.
3. Go to the next value.

Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the element $>$ pivot, go to the next value for comparison.
 - b. if the element $<$ pivot-
 - i. increase value of the variable 'A' (do $A++$).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

But how do we do step 2 (the partitioning)?



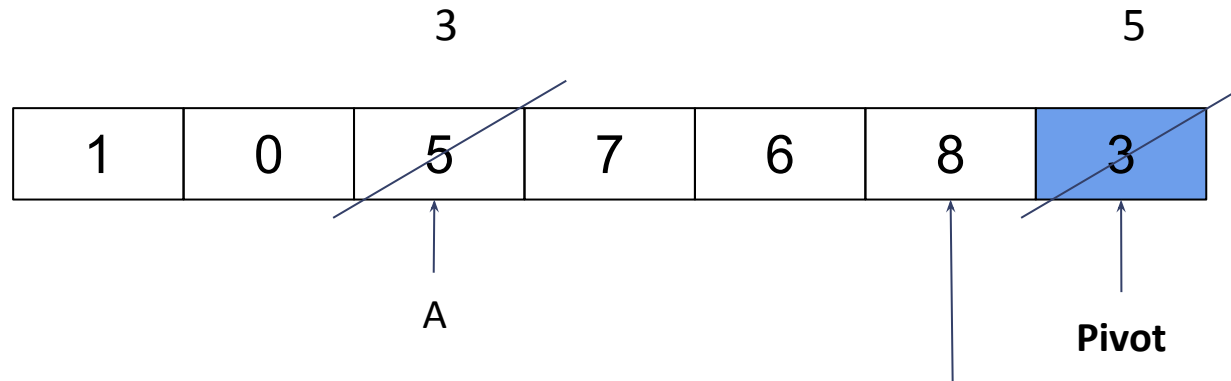
Again, $\text{value} < \text{pivot}$, so-

1. Increase value of A.
2. Swap current val with value at index 'A'.
3. Go to the next value.

Step 2B

1. Start comparing each element of the array (start traversing from the leftmost) with the pivot's value.
 - a. if the $\text{element} > \text{pivot}$, go to the next value for comparison.
 - b. if the $\text{element} < \text{pivot}$ -
 - i. increase value of the variable 'A' (do $A++$).
 - ii. Then swap the element at index 'A' with the current element in traversal.
 - iii. Then traverse to the next value.

But how do we do step 2 (the partitioning)?



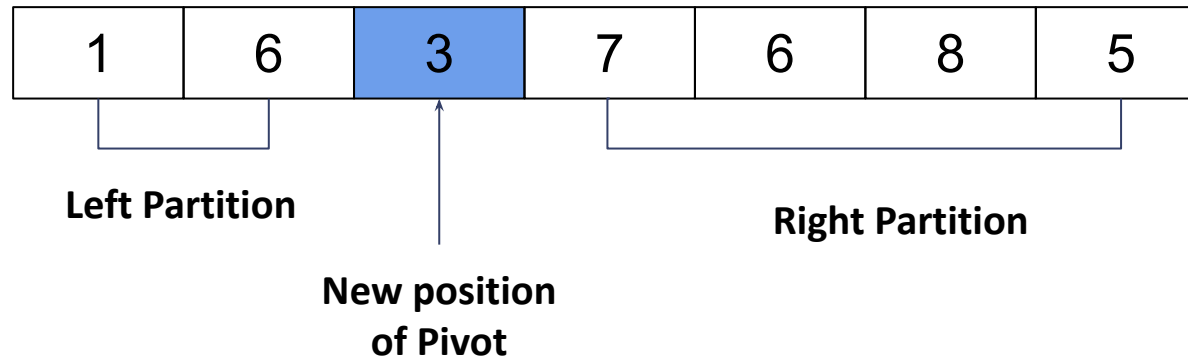
We've reached the second last element of the array, so -

1. Increase value of A.
2. Swap The pivot with value at index 'A'.
3. Return the index of the new position of the pivot.
4. The quicksort partitioning is **done** for this run.

Step 2C

1. While traversing, If we reach the second last element of the array -
 - a. Increase value of A.
 - b. Swap The pivot with value at index 'A'.
 - c. Return the index of the new position of the pivot.
 - d. The quicksort partitioning is **done** for this run.

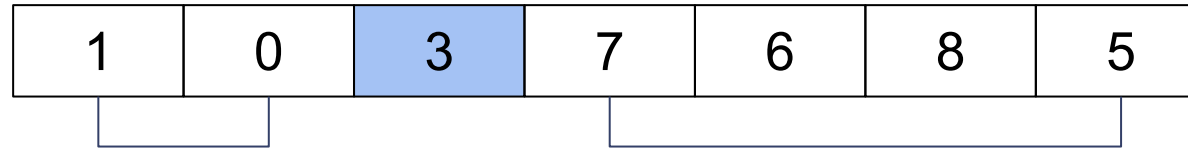
But how do we do step 2 (the partitioning)?



Step 2C

1. While traversing, If we reach the second last element of the array -
 - a. Increase value of A.
 - b. Swap The pivot with value at index 'A'.
 - c. Return the index of the new position of the pivot.
 - d. The quicksort partitioning is **done** for this run.

But how do we do step 2 (the partitioning)?



Quicksort again for this part

Quicksort again for this part

So, now we know how to do partitioning for each subgroup (we know how to do a single run).

Now, if we keep doing the exact same thing again and again the the left and right subgroups till we have just one element in each partition, our array will be sorted.

This can done using using recursion.

Now, lets implement this in C++

Solution

Try doing on your own first.

THANK YOU

All the best for the online and the term final. :)