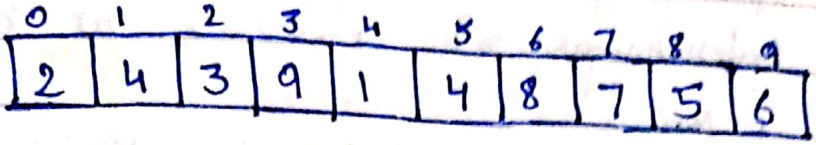
Quicksort Algorithm:

→ The Quicksort Algorithm is quite different ~~from the ones~~. Here, we use the divide & conquer method to sort our array in pieces reducing our effort and space complexity of the algorithm. There are two new concepts. First is the DIVIDE AND CONQUER method. As the name suggests, Divide & Conquer divides a problem into subproblems and solves them at their levels, giving the output as a result of all these subproblems. The second is the partition method in sorting. In the partition method, we choose an element as a pivot and try pushing all the elements smaller than the pivot element to its left and all the greater elements to its right. We thus finalize the position of the pivot element. Quicksort is implemented using both these concepts.

Suppose we are given an array of integers, and we are asked to sort them using the quicksort algorithm. Then the very first task you would do is to choose a pivot. Pivots are chosen in various ways, but for now, we'll consider the first element of every imported

subarray as the pivot. Remember this while we proceed.



US

→ In the QuickSort algorithm, every time you get a fresh unsorted subarray, you do a partition on it. Partition asks you to first choose an element as a pivot. And as already decided, we would choose the first element of the unsorted subarray as the pivot. We would need two more index variables, i & j. Below enlisted is the flow of our partition algorithm we must adhere to. We always start from step 1 with each fresh partition call.

1) Define i as the low index, which is the index of the first element of the subarray, and j as the high index, which is the index of the last element of the subarray.
2) Set the pivot as the element at the low index i since that is the first index of unsorted subarray.
3) Increase i by 1 until you reach an element greater than the pivot element.
4) Decrease j by 1 until you reach an element smaller than or equal to the

pivot element.

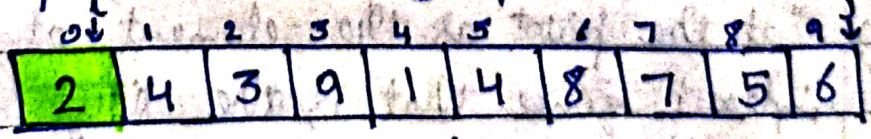
5) Having fixed the values of i & j, interchange the elements at indices i and j.

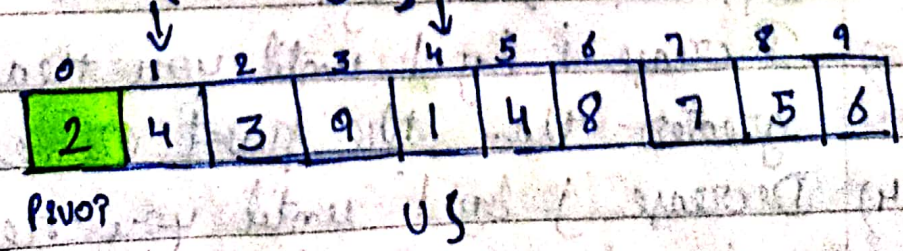
6) Repeat steps 3, 4 & 5 until j becomes less than or equal to i.

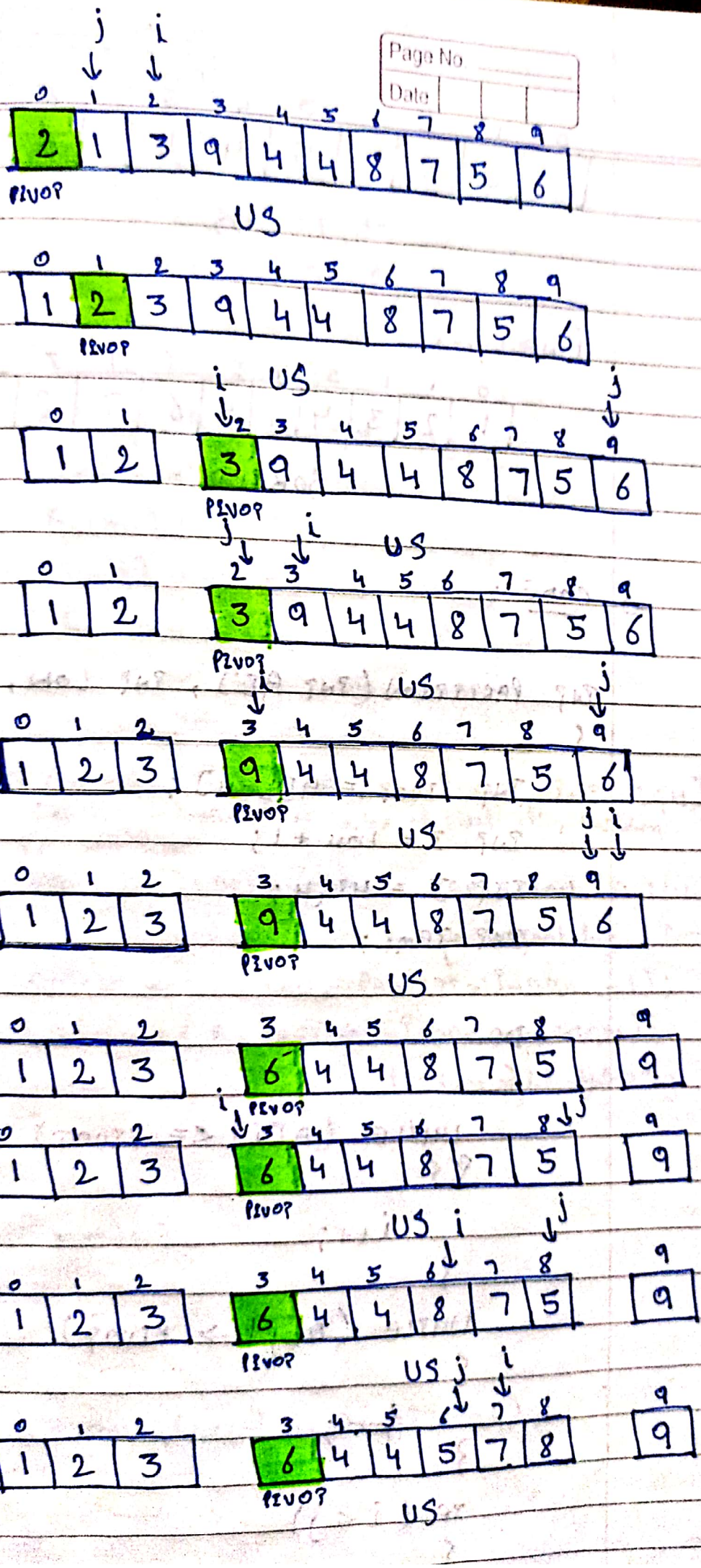
7) Finally, swap the pivot element and the element at the index.

→ This was the partition algorithm. Every time you call a partition, the pivot element gets its final position. A partition never guarantees a sorted array, but it does guarantee that all the smaller elements are located to the pivot's left, and all the greater elements are located to the pivot's right.

→ Now let's look at how the array we received at the beginning gets sorted using partitioning and divide and conquer recursively for smaller subarrays.

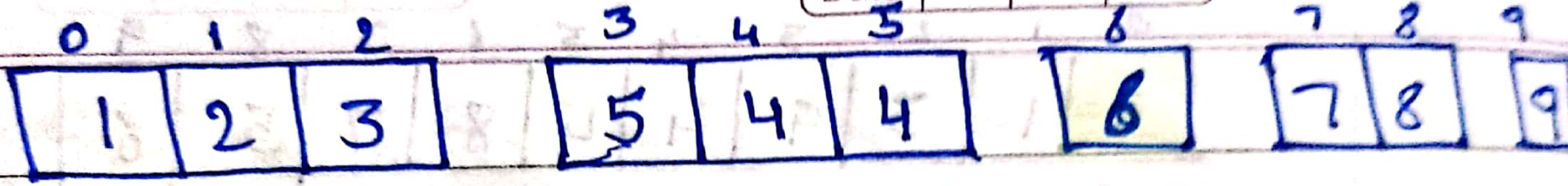






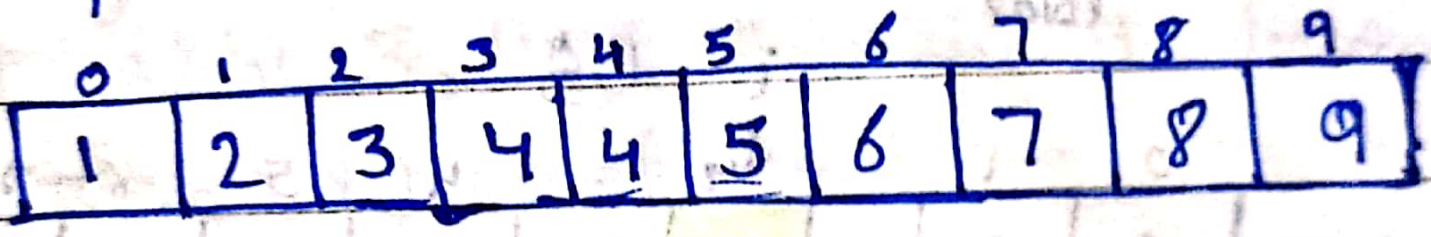
Page No.

Date



US US

FINALLY...



SORTED