Selection Sort Algorithm :

0	1	2	3	4
8	0	7	1	3

UNSORTED ARRAY

$\longrightarrow$

0	1	2	3	4
0	1	3	7	8

SORTED ARRAY

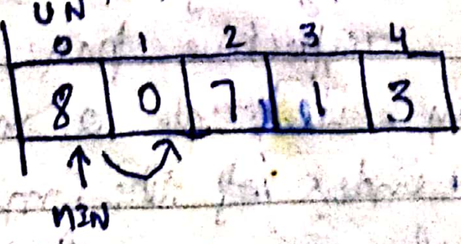
→ In selection sort, at each pass, we make sure that the smallest element of the current unsorted subarray reaches its final position. And this is pursued by finding the smallest element in the unsorted subarray and replacing it at the end with the element at the first index of the unsorted subarray. This algorithm reduces the size of the unsorted part by 1 and ↑ses the size of the sorted part by 1 at each respective pass. ~~Let~~
At each pass, we create a variable min to store the index of the minimum element. We start by assuming that the first element of the unsorted subarray is

the minimum. We will iterate through the unsorted part of the array, and compare every element to the element at min index. If the element is less than the element at min index, we replace min by the current index and move ahead. Else, we keep going. And when we reach the end of the array, we replace the first element of the unsorted subarray with the element at min index. And doing this at every pass ensures that the smallest element of the unsorted part of the array reaches its final position at the end.

Since our array is of length 5, we will make 4 passes. ~~You~~

1st Pass:

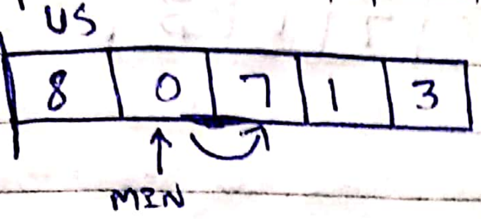
At first pass, our whole array comes under the unsorted part. We will start by assuming 0 as the min index. Now, we'll have to check among the remaining 4 elements if there is still a lesser element than the first one



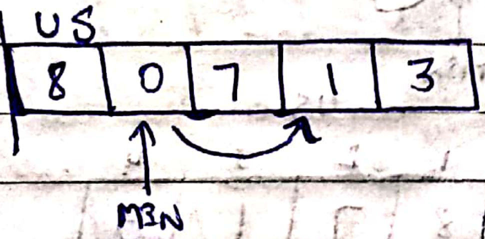
And when we compared the element at

Page No.
Date

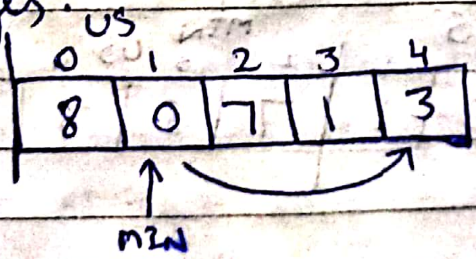
min index with the element at index 1, we found that 0 is less than 8 and hence we update our min index to 1.



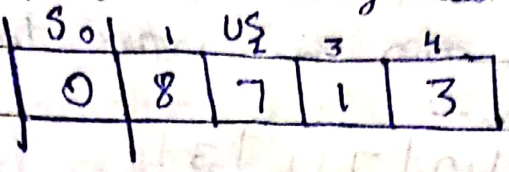
And now we keep checking with the updated min. Since 7 is not less than 0, we move ahead.



And now we compared the element at index 1 and 3, and 0 is still lesser than 1, so we move ahead without making any changes.

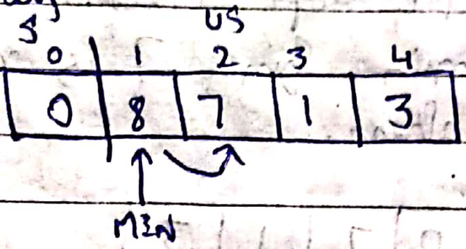


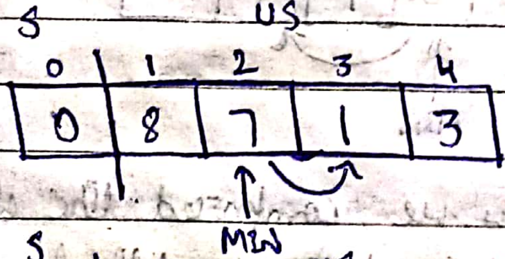
And now we compared the element at the min index with the last element. Since there ~~is~~ is nothing to change, we end our 1st pass here, Now we simply replace the element at 0th index with the element at the min index. And this gives us our first sorted ~~or~~ subarray of size 1. And this is where our first pass finishes. ~~We should make~~

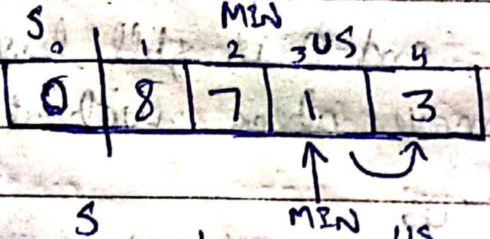


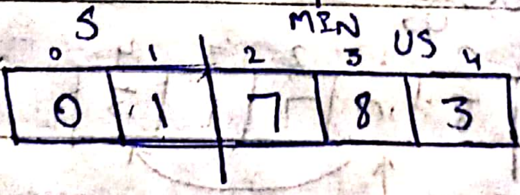
→ Similarly :

2ND Pass :

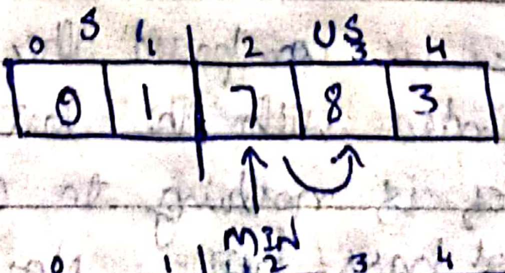


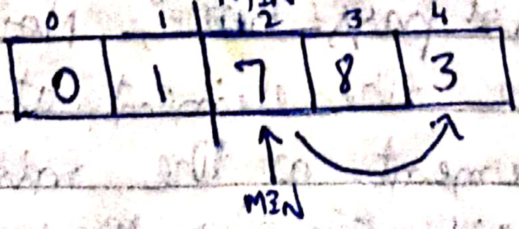


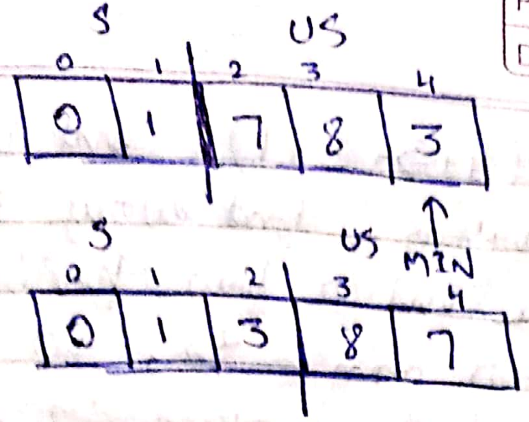




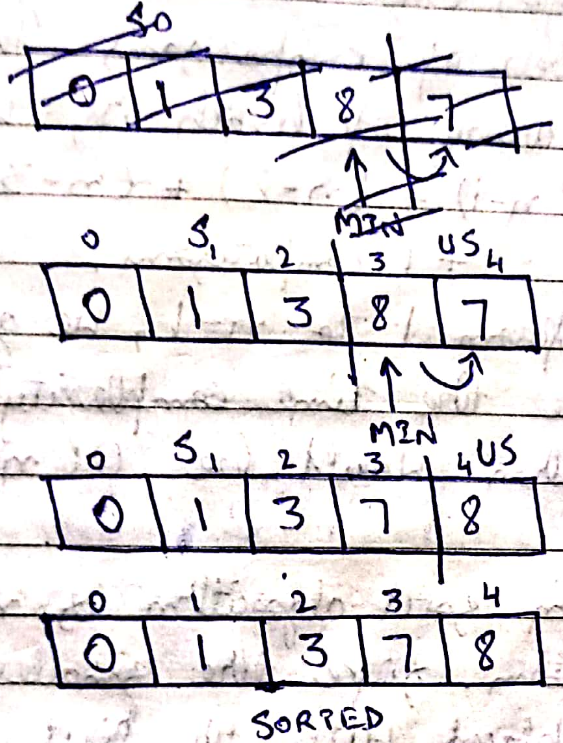
3RD Pass:







4th Pass:



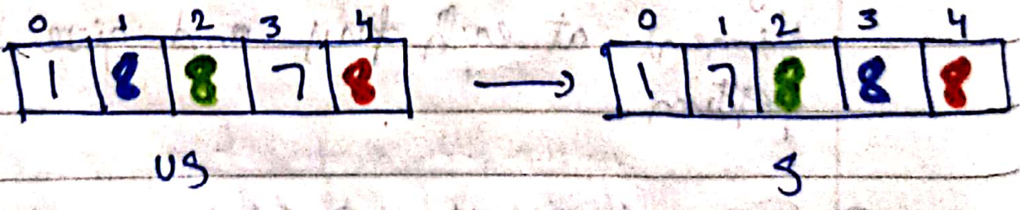
→ And this is why the Selection Sort algorithm got its name. We select the minimum element at each pass and give it its final position.

① Time Complexity of Selection Sort:

→ We made 4 passes for an array of length 5. Therefore, for an array of length n we would have to make n-1 passes. And if you count the number of comparisons we made at each pass,

there were $(4+3+2+1)$, that is, a total of 10 comparisons. And every time we compared; we had a fair possibility of updating our min. So, 10 comparisons are equivalent to making 10 updates. So, for length 5, we had $4+3+2+1$ number of comparisons. Therefore, for an array of length n, we would have $(n-1)+(n-2)+(n-3)+(n-4)+$ ---- $+1$ comparisons.
Sum from 1 to $n-1$, we get, and hence the time complexity of the algorithm would be $O(n^2)$.

② Selection sort algorithm is not a stable algorithm. Since the smallest element is replaced with the first element at each pass, it may jumble up positions of equal elements very easily. Hence, unstable.



③ It is not a recursive algorithm, since, we didn't use recursion here.

④ Selection sort would anyways compare every element with the min element,

Page No.
Date

regardless of the fact if the array is sorted or not, hence selection sort is not an adaptive algorithm by default.

⑤ This algorithm offers the benefit of making the least number of swaps to sort an array. We don't make any redundant swaps here.