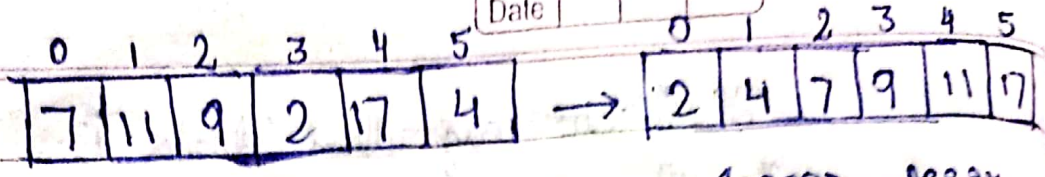
# BUBBLE SORT ALGORITHM



→ We can sort with any algorithm, the result will always be same just the different lies in the algorithm we follow. With Bubble sort, we intend to ensure that the largest element of the segment reaches the last position at each iteration.

Page No.
Date




UNSORTED ARRAY SORTED ARRAY

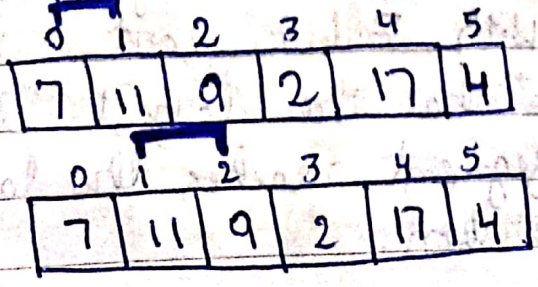
→ Bubble sort intends to sort an array using $(n-1)$ passes where $n$ is the array's length. And in one pass, the largest element of the current unsorted part reaches its final position, and our unsorted part of the array reduces by 1, and the sorted part rises by 1.

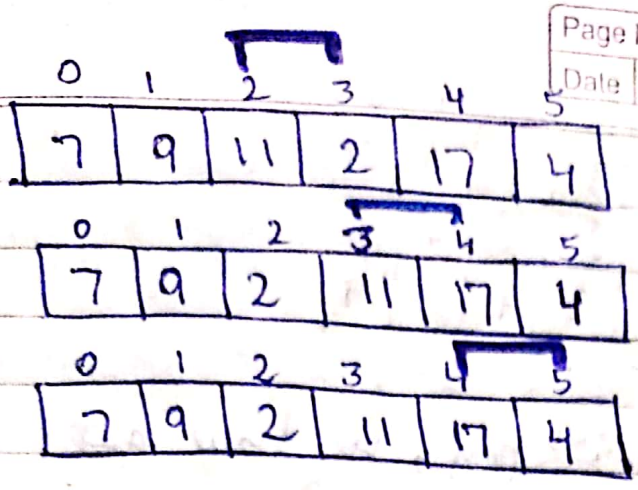
→ At each pass, we will iterate through the unsorted part of the array and compare every adjacent pair. We move ahead if the adjacent pair is sorted; otherwise, we make it sorted by swapping their positions. And doing this at every pass ensures that the largest element of the unsorted part of the array reaches its final position at the end.

→ Since our array is of length 6, we will make 5 passes.

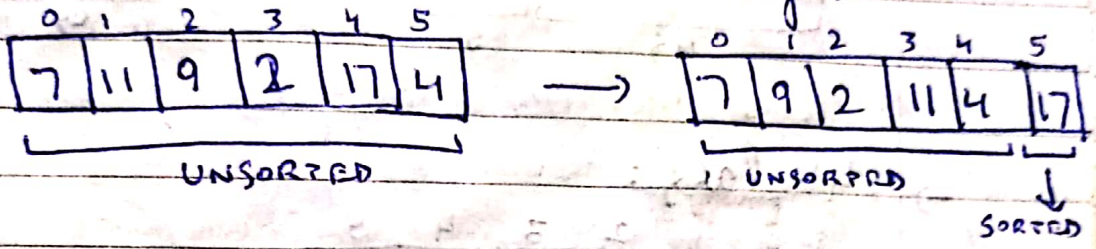
↳ MEANS CHANGE

→ 1st pass

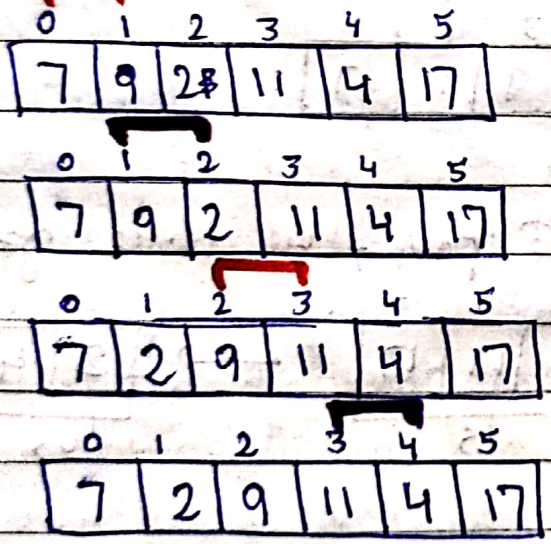




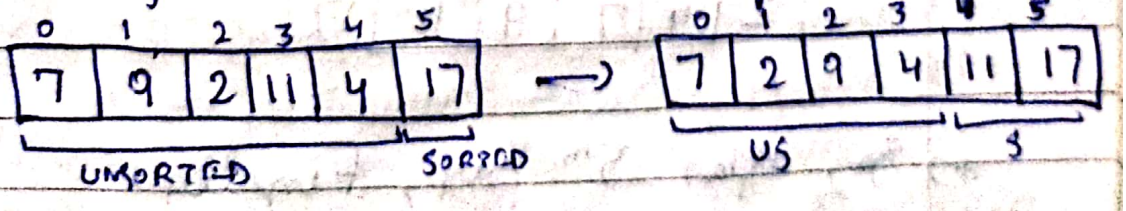

*) So, after 1st Pass, we have got:



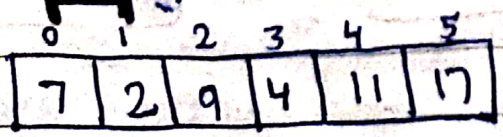

→ 2ND Pass:

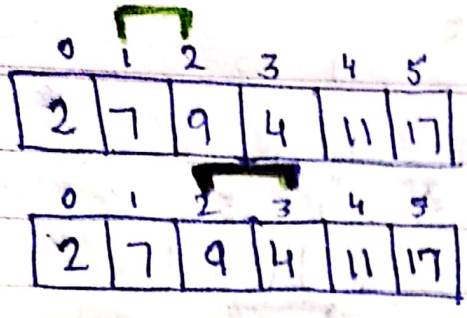



*) So, after 2ND Pass, we have got:

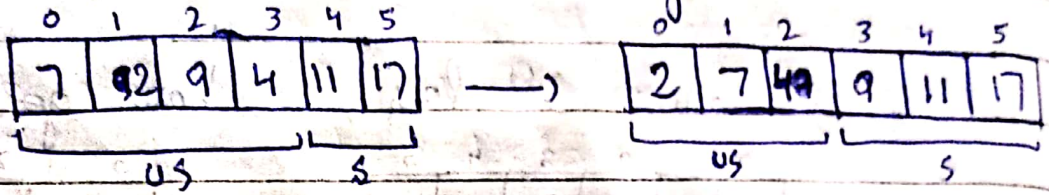



→ 3RD Pass:

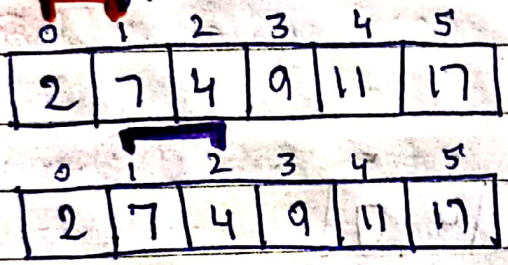




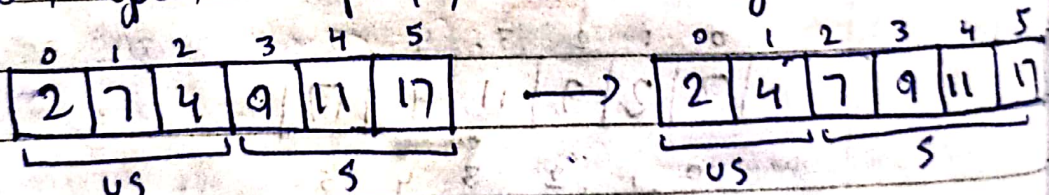

*) So, after 3RD pass, we have got:



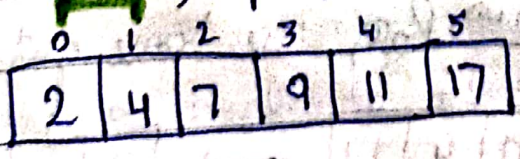

→ 4th Pass:



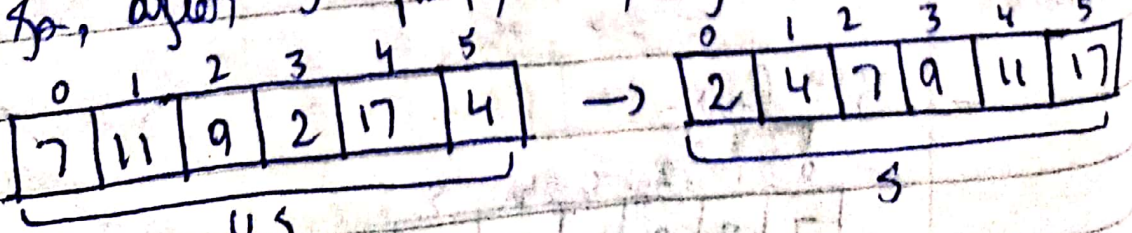

*) So, after 4th pass, we have got:




→ 5th (last) pass:




*) So, after 5th pass, Finally:



→ And this is what the Bubble Sort algorithm looks like:.

→ Time Complexity of Bubble Sort:

→ If you count the number of comparisons we made, there were (5+4+3+2+1), that is, a total of 15 comparisons. And every time we compared, we had a fair probability of making a swap. So, 15 comparisons intend to make 15 possible swaps. For length 6, we had 5+4+3+2+1 number of comparisons and possible swaps, Therefore, for an array of length n, we would have (n-1) + (n-2) + (n-3) + (n-4) + ... + 1 comparison and possible swaps.
→ This is a high school thing to find the sum from 1 to n-1, which is n(n-1)/2, and hence our complexity of runtime become $O(n^2)$
→ And if you could observe, we never made a swap when two elements of a pair become equal. Hence the algorithm is a stable algorithm.
→ It is not a recursive algorithm since we didn't use recursion here.
→ This algorithm has no adaptive aspect since every pair will be compared,

even if the array given has already been sorted. So, no adaptiveness. Although it can be modified to make it adaptive, it is not adaptive by default.

NOTE→ Bubble sort is called bubble because it bubbles up lighter elements to the left and stores larger elements towards the right.