

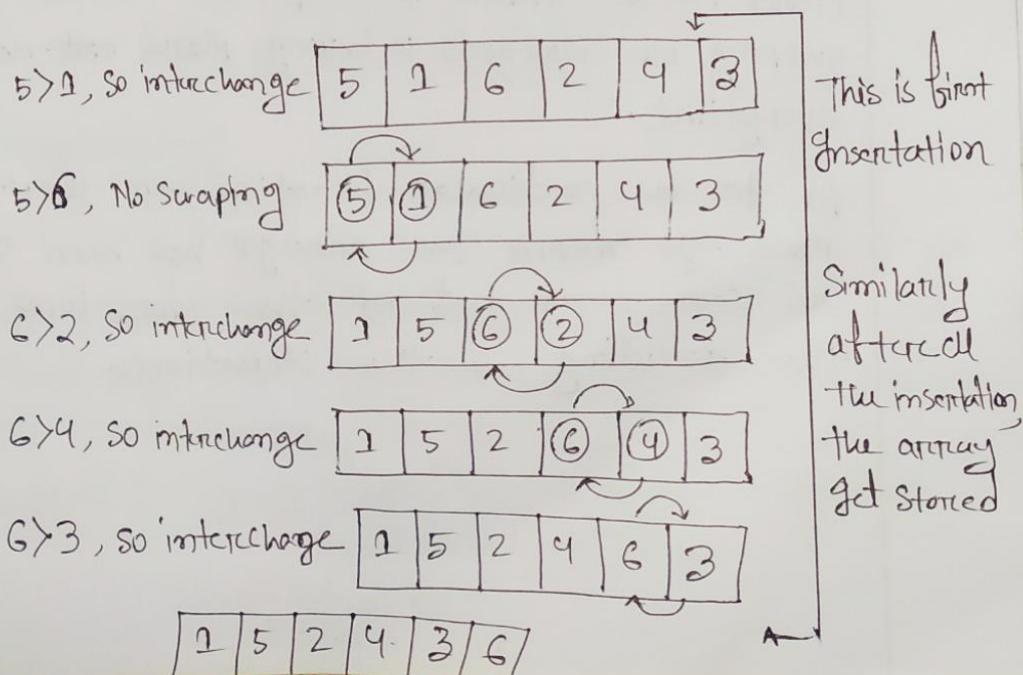
Ahmed Zubayer Sunny
101-15-12560

Algorithm - 2020
0-24

Implementing Bubble Sort Algorithm

- ① Starting with the first element (Index=0), compare the current element with the next element of the array.
- ② If the current element is greater than the next element of the array, swap them.
- ③ If the current element is less than the next element, Repeat Step 1.

Lets consider array with Value $\{5, 1, 6, 2, 4, 3\}$



So as we can see representation above, after the first iteration, 6 is placed at the last index, which is the correct position of it. Similarly after the second iteration, 5 will be at the second last index. and so on.

Optimized The Bubble Sort Algorithm:

We can indicate flag to monitor whether elements are getting swapped inside the inner for loop.

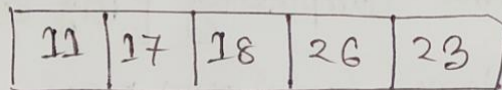
Hence, in the inner for loop, we check whether swapping of elements is taking place or not, everytime..

If for a particular iteration, no swapping took place. It means the array has been sorted and we can jump out of the for loop, instead of executing all the iterations.

Lets consider any array with values

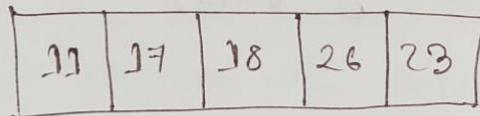
$\{11, 17, 18, 26, 23\}$

$11 > 17$
(No Swapping)



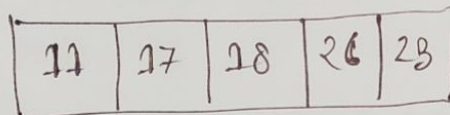
flag = 0
(flag remains 0)

$17 > 18$
(No Swapping)



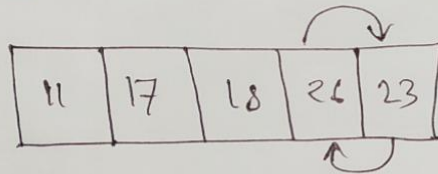
flag = 0

$18 > 26$
(No)



flag = 0

$26 < 23$
(Swapping)



flag = 1

Complexity:

In bubble sort, $n-1$ comparisons will be done in the 1st pass, $n-2$ in 2nd pass, $n-3$ in 3rd pass and so on. So the total number of comparisons will be,

output: $(n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1$

$$\text{Sum} = n(n-1)/2$$

$$\text{i.e. } O(n^2)$$

Hence Time Complexity of bubble sort is $O(n^2)$

→ Worst time complexity $[Big-O] : O(n^2)$

→ Best time complexity $[Big-O] : O(n)$

→ Average time complexity $[Big-Theta] : O(n^2)$

→ Space complexity : $O(1)$

Linear Search

Worst Case:

In the linear search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() function compares it with all the elements of $a[]$ one by one.

So the worst case time complexity of linear search would be $O(n)$.

Average Case:

For the linear search problem, assume that the all cases are uniformly distributed. So we sum all the cases and divided the sum by $(n+1)$.

$$\begin{aligned}\text{Average Case time} &= \frac{\sum_{i=1}^{n+1} O(i)}{(n+1)} \\ &= \frac{O((n+1) \times (n+2)/2)}{(n+1)} \\ &= O(n)\end{aligned}$$

Best Case : Merge Sort does $O(n \log n)$ operations in all cases.