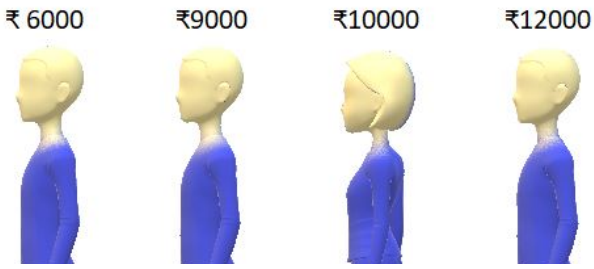
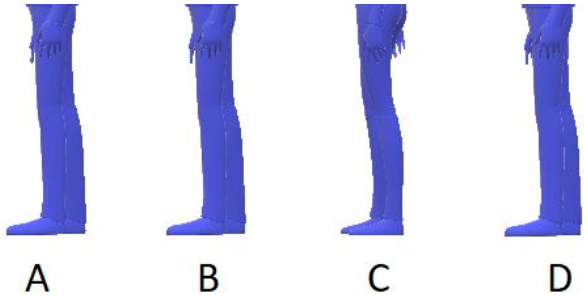
Show Course Contents

Overview Q&A Downloads Announcements

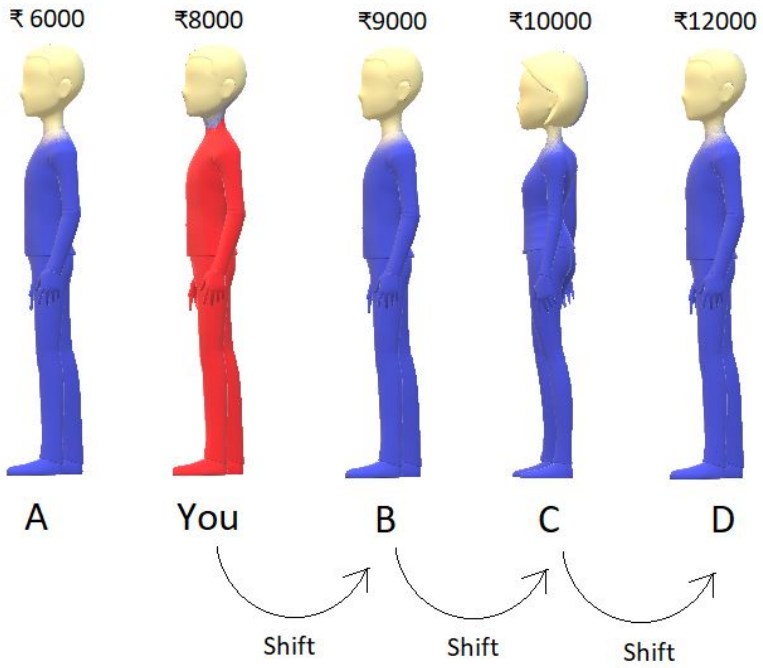
Insertion Sort Algorithm in Hindi

In the last lecture, we finished learning the Bubble Sort Algorithm. We learned how to write a program for the same in C language. We saw its characteristics as well. Today, we will learn about a new sorting method, called the Insertion Sort Algorithm. I will make it very intuitive for you to understand. I will use some real-life applications to make the process easy and will steadily dive into the technical part. Suppose you were to stand in a queue where people are already sorted on the basis of the amount of money they have. Person with the least amount is standing in the front and the person with the largest sum in his pocket stands last. The below illustration describes the given situation.





Problem arises when you suppose you have ₹8000 in your pocket, and you want to be a part of this queue. You don't know where to stand. So, now you start from the last and keep asking the person standing there whether he has more money than you or less money than you. If you find someone with more money, you simply ask him/her to shift backward. And the moment you find a person having less money than you, you stand just behind him/her. So, after doing all this, you find a position in the 2nd place in the queue. The final situation is:



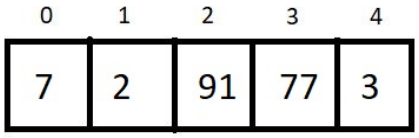
So, this was one of the examples I had in mind. Now, suppose these were not the people but the numbers in an array. It would have been as simple as it is right now. We would keep comparing two numbers, and if we find a number greater than the number we want to insert, we shift it backward. And the moment we find a number smaller, we insert the element at the vacant space just behind the smaller number.

And basically, what did we learn? We learned to insert an element in a sorted array. Although it felt very intuitive to just put yourself in the second position, what would you do if the queue had a thousand people? Not easy, right? And this is where we need a proper algorithm.

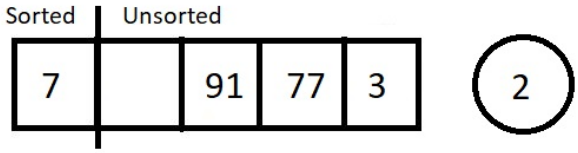
Insert Sort Algorithm:

Let's just take an array, and use the insertion sort algorithm to sort its elements in increasing order.

Consider the given array below:

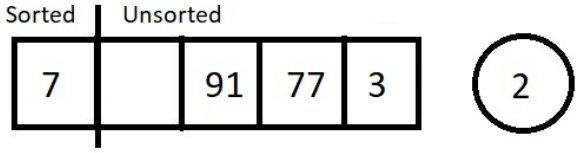


And what have we already learned? We have learned to put an arbitrary element inside a sorted array, using the insertion method we saw above. **And an array of a single element is always sorted.** So, what we have now is an array of length 5 with a subarray of length 1 already sorted.



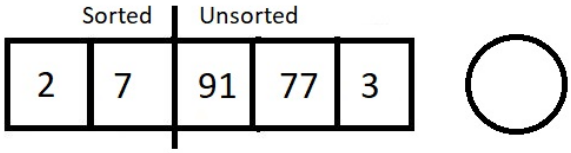
Moving from the left to the right, we will pluck the first element from the unsorted part, and insert it in the sorted subarray. This way at each insertion, our sorted subarray length would increase by 1 and unsorted subarray length decreases by 1. Let's call each of these insertions and the traversal of the sorted subarray to find the best position, a pass.

So, let's start with pass 1, which is to insert 2 in the sorted array of length 1.

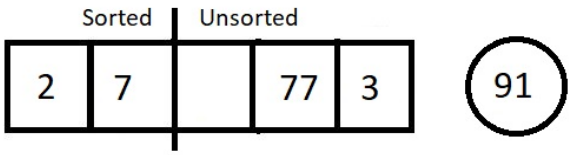


So, we plucked the first element from the unsorted part. Let's insert element 2 at its correct position, which is before 7. And this increases the size of our sorted

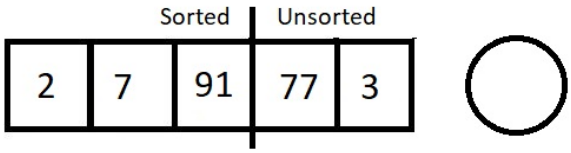
array.



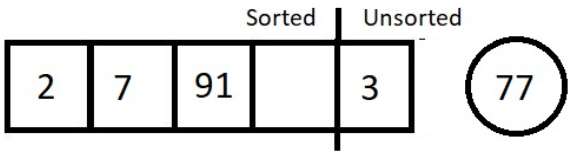
Let's proceed to the next pass.



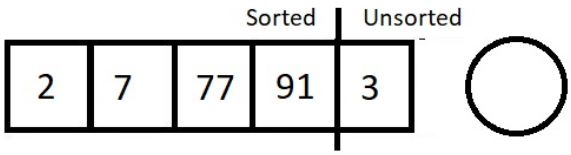
The next element we plucked out was 91. And its position in the sorted array is at the last. So that would cause zero shifting. And our array would look like this.



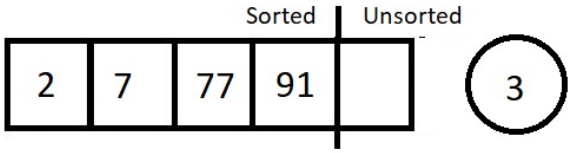
Our sorted subarray now has size 3, and unsorted subarray is now of length 2. Let's proceed to the next pass which would be to traverse in this sorted array of length 3 and insert element 77.



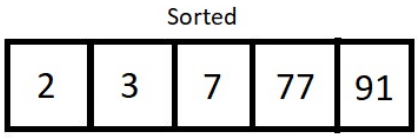
We started checking its best fit, and found the place next to element 7. So this time it would cause just a single shift of element 91.



As a result, we are left with a single element in the unsorted subarray. Let's pull that out too in our last pass.



Since our new element to insert is the element 3, we started checking for its position from the back. The position is, no doubt, just next to element 2. So, we shifted elements 7, 77, and 91. Those were the only three shifts. And the final sorted we received is illustrated below.



So, this was the main procedure behind the insertion sort algorithm.

Analysis:

Conclusively, we had to have 4 passes to sort an array of length 5. And in the first pass, we had to compare the *to-be* inserted element with just one single element 7. So, only one comparison, and one possible swap. Similarly, for *ith* pass, we would have *i* number of comparisons, and *i* possible swaps.

1. Time Complexity of Insertion Sort Algorithm:

Let's now calculate the time complexity of the algorithm. We made 4 passes for this array of length 5, and for ith pass, we made *i* number of comparisons. So, the total number of comparisons is 1+2+3+4. Similarly, for an array of length n, the total number of comparison/possible swaps would be 1+2+3+4+ . . . + (n-1) which is n(n-1)/2, which ultimately is **O(n2)**.

2. Insertion sort algorithm is a **stable algorithm**, since we start comparing from the back of the sorted subarray, and never cross an element equal to the *to be* inserted element.

3. Insertion sort algorithm is an **adaptive algorithm**. When our array is already sorted, we just make (n-1) passes, and don't make any actual comparison between the elements. Hence, we accomplish the job in **O(n)**.

Note: At each pass, we get a sorted subarray at the left, but this intermediate state of the array has no real significance, unlike the bubble sort algorithm where at

each pass, we get the largest element having its position fixed at the end.
And that was all about the insertion sort algorithm. I expect you all to take your own unsorted array, and use the insert sort algorithm this time to sort it. Rather, use both bubble sort and insertion sort, and see if it matches, and tell me which one you found convenient. If something seems unclear, go through the lectures again.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn how to code the insertion sort algorithm. Till then keep coding.

Previous Next