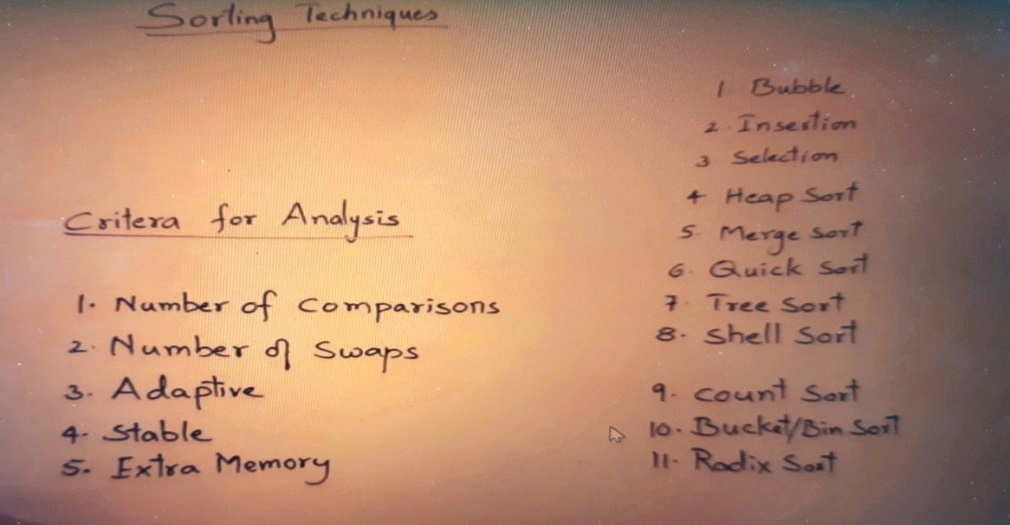
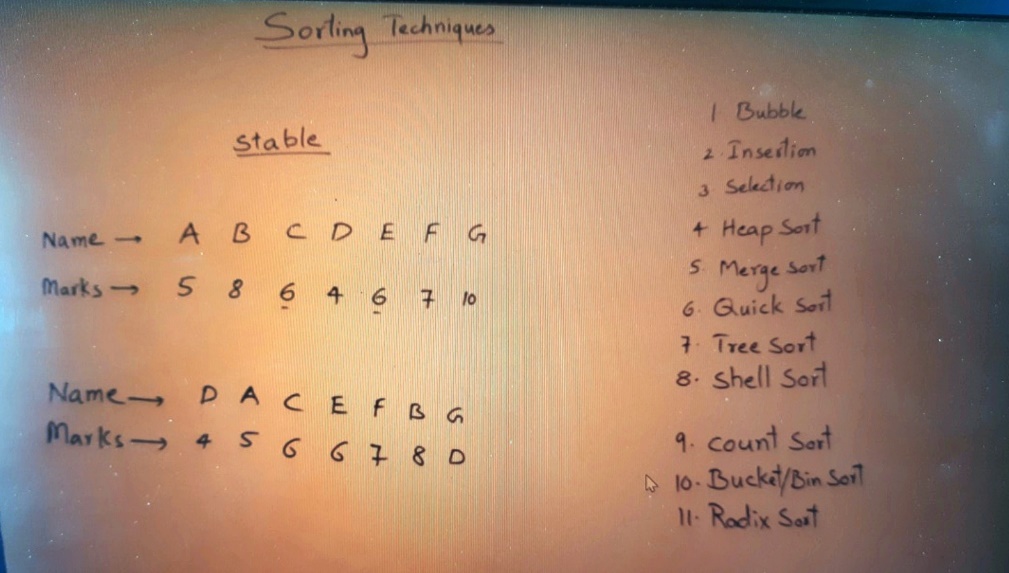
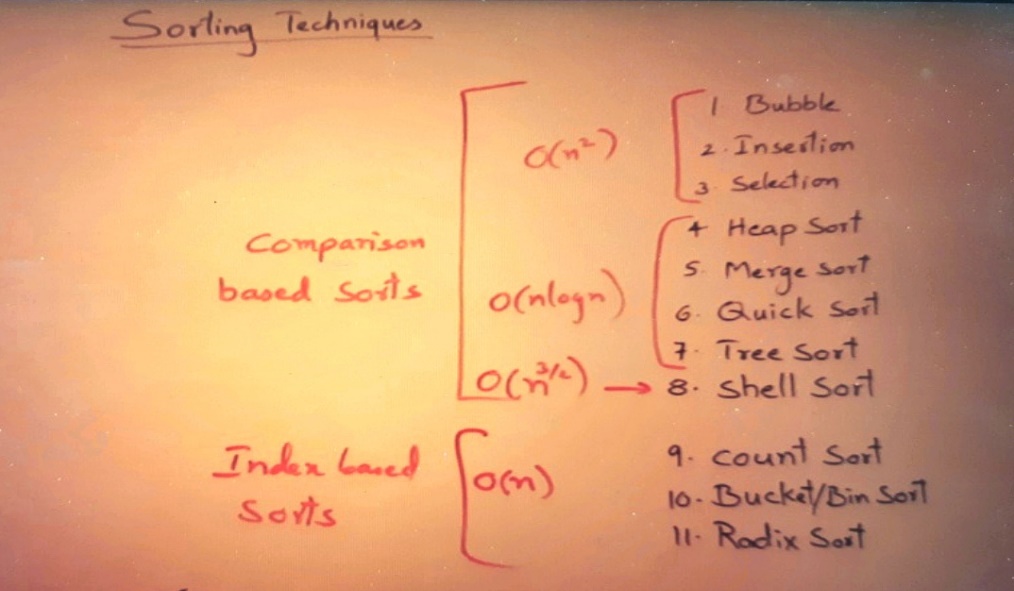
**Sorting**

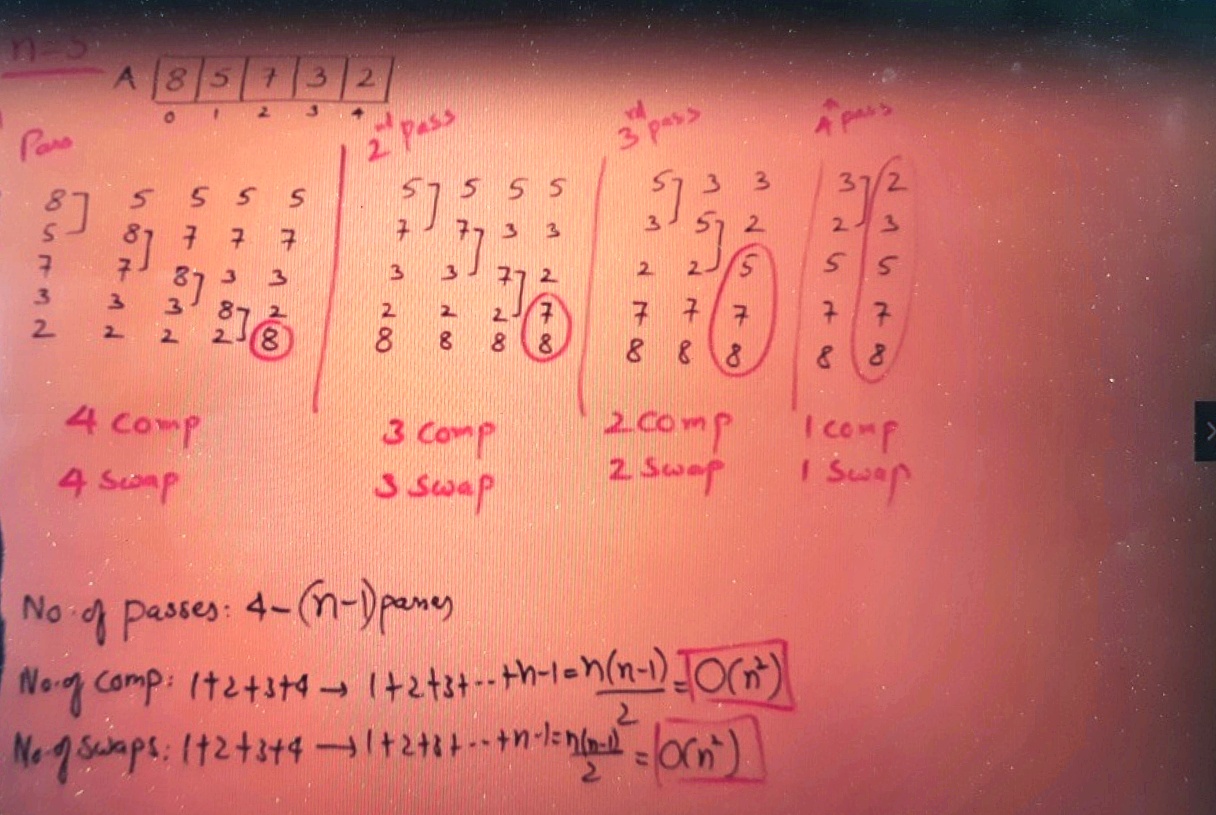






**Bubble Sort**

1. **How many comparisons, how many swaps .**
2. **The time complexity of any sorting algorithm is given based on a number of comparisons. So, the time taken by this algorithm is O(n^2) now of five criteria that we have seen in our first video**
3. **now out of 5 criteria that we have seen in our first video of the section, I have completed 2 criteria comparisons and swaps remaining 3 are there whether it is adaptive or not, whether it is a stable or not, whether it is taking extra memory, these things we have to discuss.**



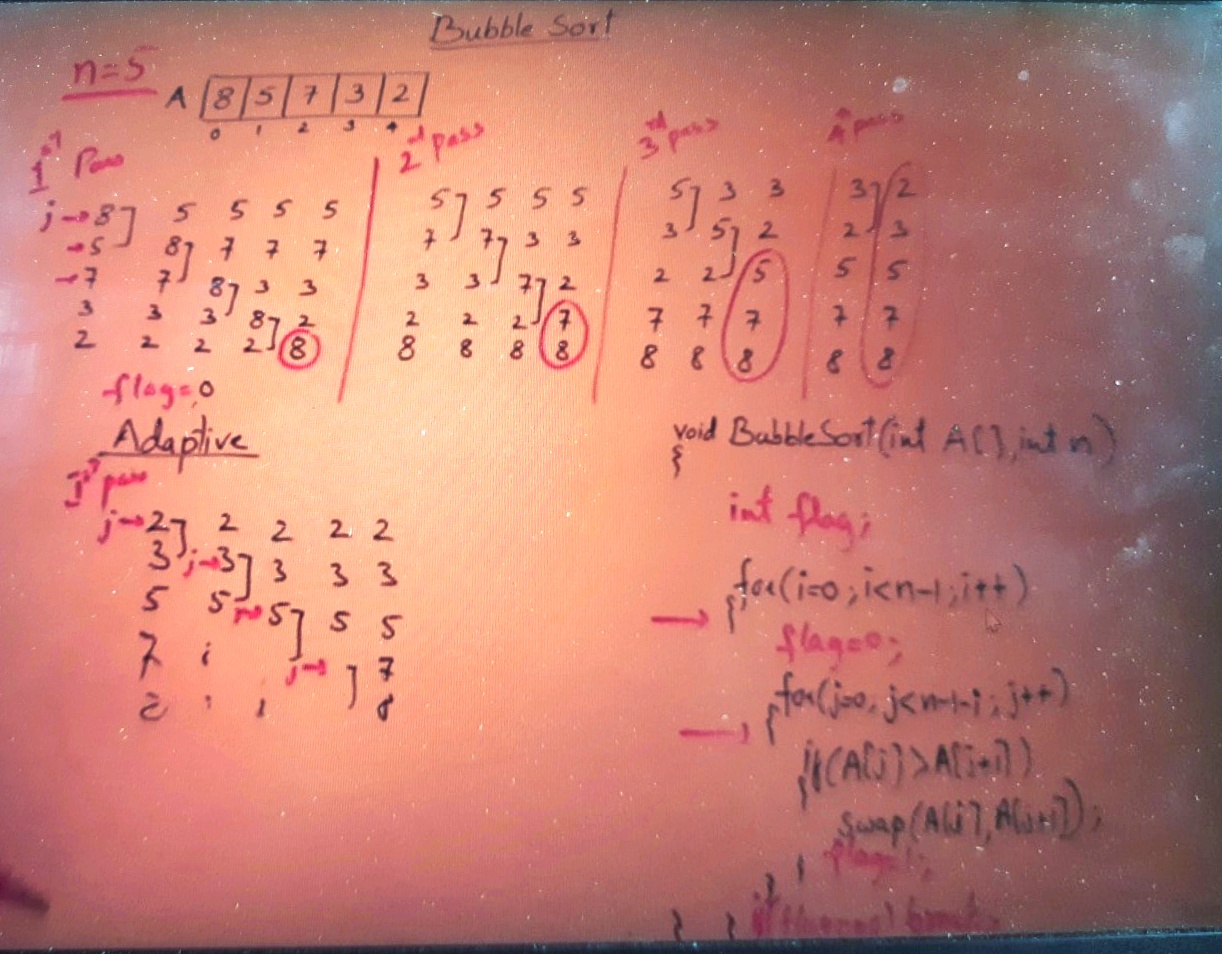
why it is called the bubbles sort, this was the list of elements. This was demonstrating this one. 8 is heaviest element. So, if you drop it from here, it is going down and settling at the bottom. And the other elements like 2,3,7,5 they are lighter elements. So, they are slowly moving up. So, they are moving up.

**So, the effect is just like if you throw a stone in water, then stone is heavier, so it will go and settled down than the bubbles which are lighter that will come up.**

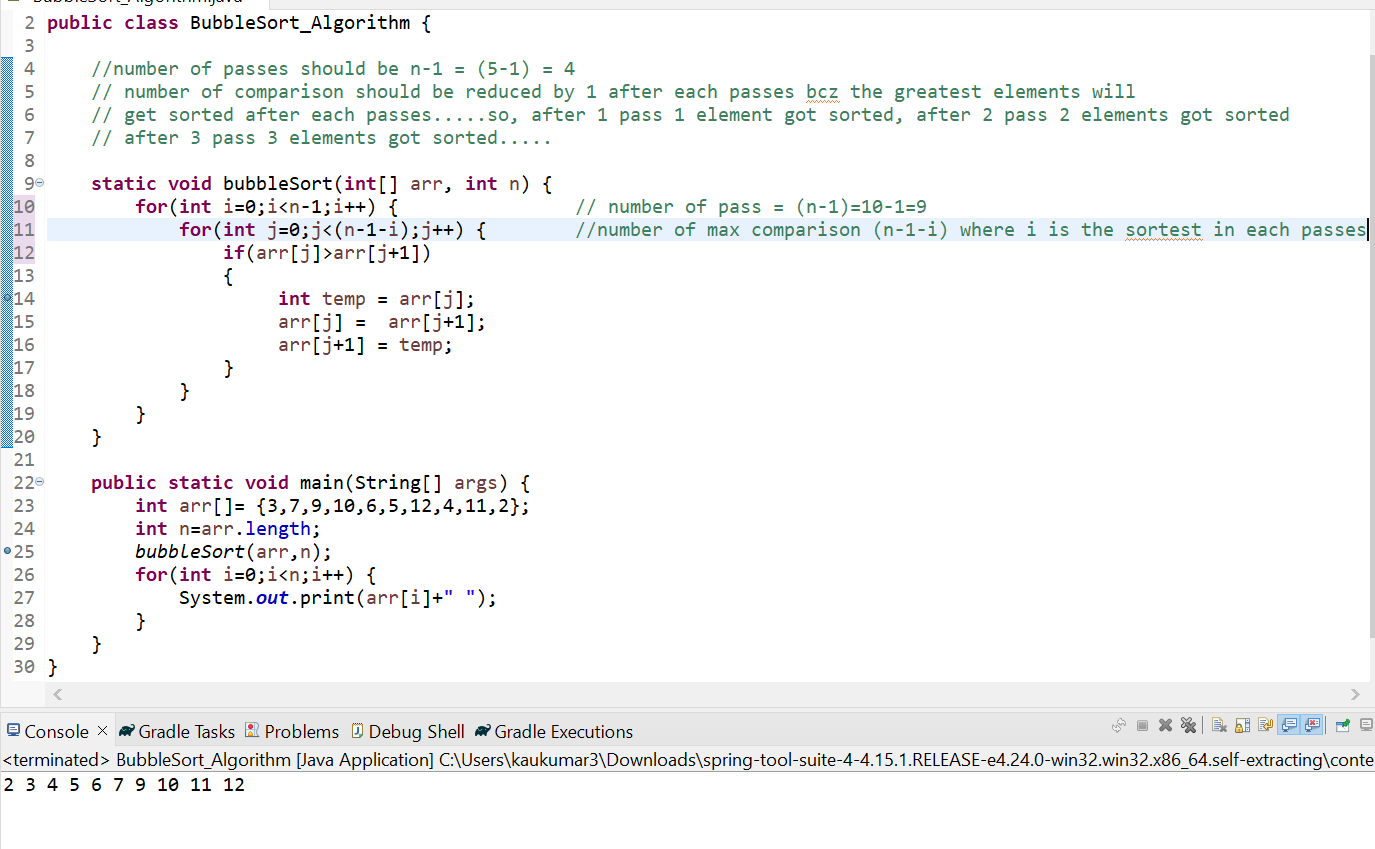
So that's why the name of this sort bubble sort, so the effect is just like bubbles in water when this heavy thing is thrown in water. I don't know who observed this and given the name, but this is what the observation based on which the name bubbles sort just given.

**one more interesting thing in 1 pass, I got the largest element, then in 2nd pass I got 2 largest elements. 3rd pass three largest elements. for some n elements. If we perform K passes, then what? K Largest elements we get.**

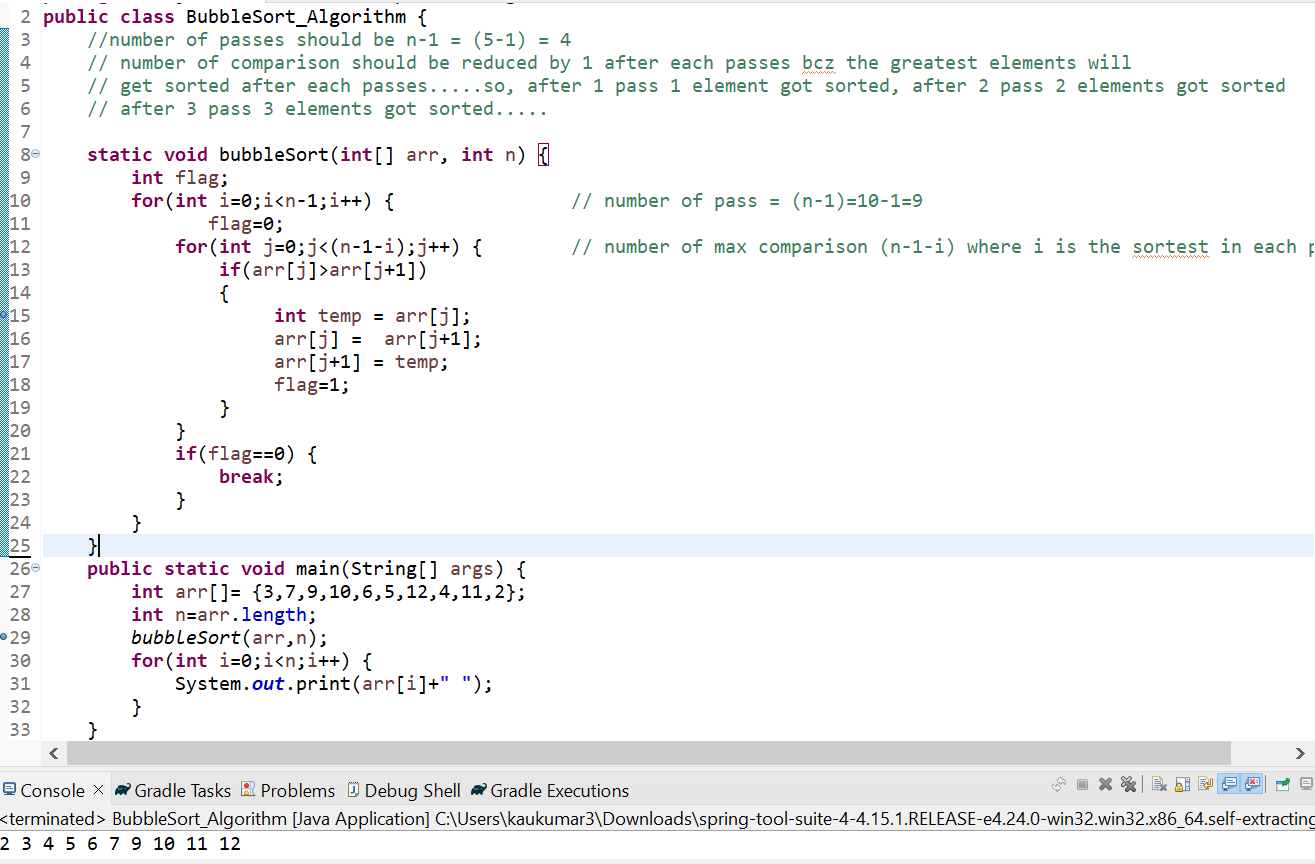
**Suppose you have a list, and you don't want to sort everything, just you want which is the greatest then perform 1 pass. I want first 3 greatest elements then perform 3 passes and stop. So, you can take the help of this procedure this algorithm for performing selected number of passes to get the desired result or required result.**



**So, there will be 2 things 1 is the number of passes and 2nd thing is the number of comparisons.**



**To check either it’s an Adaptive or not we will use a flag and we will make flag=0 after pass and if any swap has been done we will make flag = true, which means given number was not in sorted order bcz at least 1 swapping has been done.**



**We can improve bubble sort we can make it adaptive by introducing a flag.**

**Time Complexity – O(n^2) bcz there are 2 loops 1 for no. of pass and another for comparison.**

**Insertion Sort**

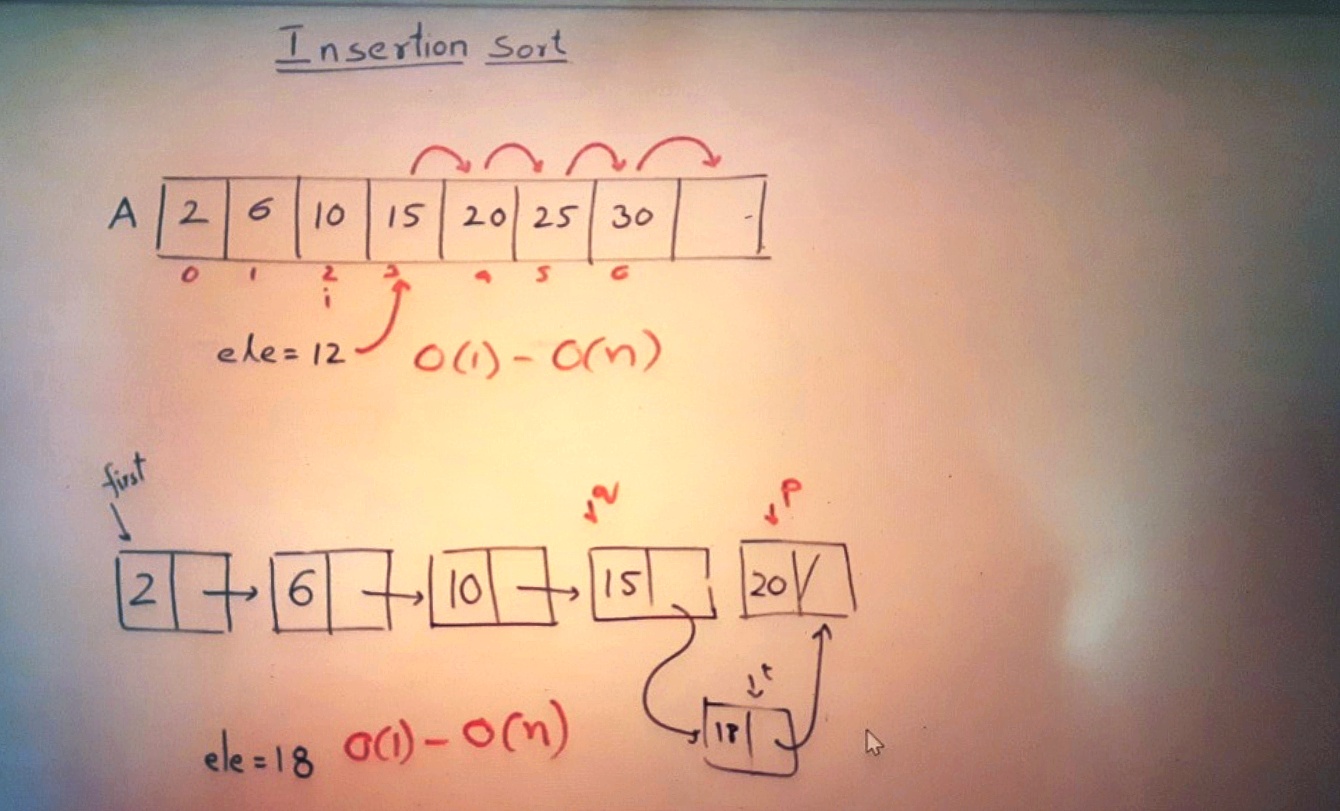
First thing, if I perform just one pass, what is the result, see, after the first pass, I got result 5 and 8. Is it useful anyway? Can I say that the first element will be the smallest element? No, we cannot say that that is not the smallest element. If suppose I have one more element of 9 here, can I say 8 is the largest element? No. So intermediate result that is just 1 pass or 2 pass will not give you any useful result in insertions sort.

Like in bubble sort, if you perform 1 pass, you get 1 largest element for 2 pass u will get 2 larger elements. But in insertion sort, you will not get anything useful.

We have seen insertion sort in an array as well as insertion in a LinkedList so in an array we have to shift the elements but in linked list we don’t have to shift anything. So, the benefit of insertion sort upon LinkedList is that you don't have to shift the elements without shifting you can insert in LinkedList.

So, insertion sort is more useful or more compatible with the linked list than Array. Or, I should say that Insertion sorts are designed for LinkedList.

So, always remember for sorting linked list insertion sort is better.



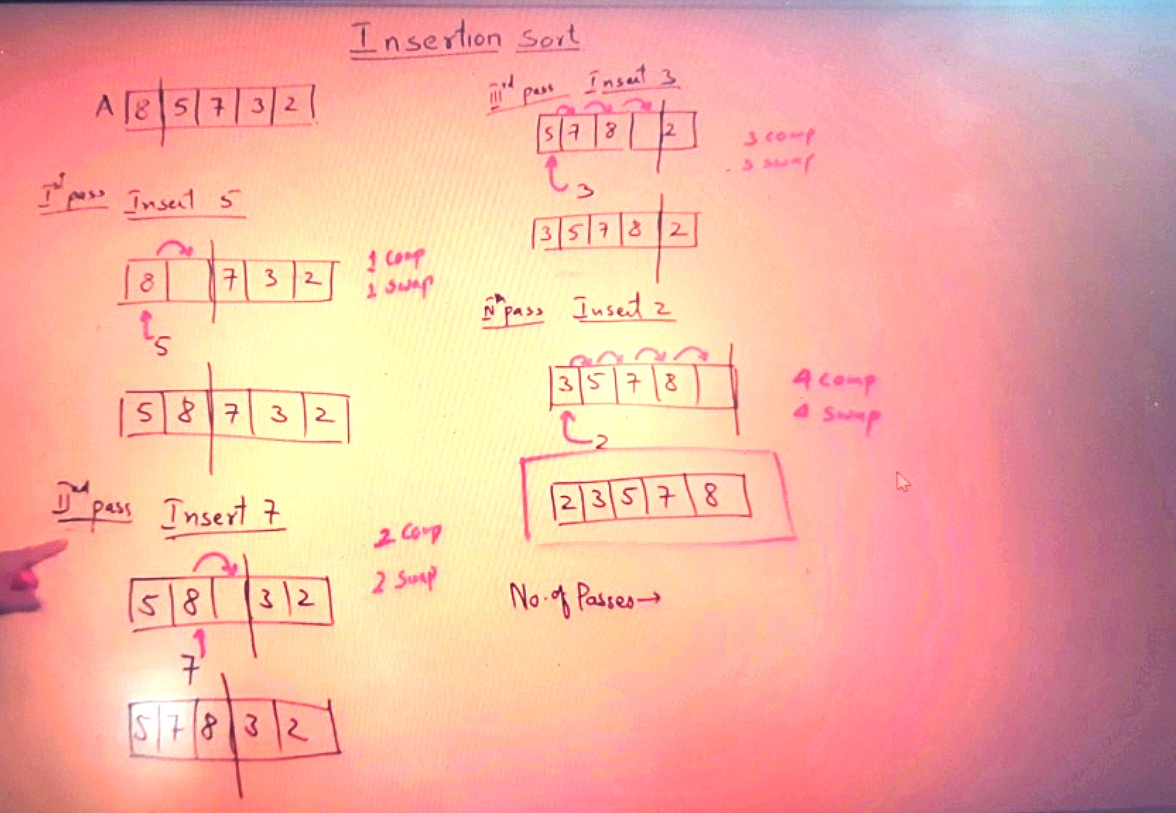
The very first thing is I have 5 elements here. So, we assume that the 1st element is already sorted.

If there is only one element, then obviously it is sorted. So, we assume that just the first elementis sorted , then the remaining elements are not sorted. So remaining elements will take out from an array and insert them on this side in sorted list. So, take out an element and insert this side So take out an element and insert this side.

So right now, in an array, 1 element is sorted. 4 elements are not sorted. One element that is 8 is sorted, so from an unsorted list will take out an element and insert this side, so take out an element and insert this side so this is the process is.

Let’s start process-----

1. I will insert 5. so far inserting 5 see first element 8 is as it is, I'm inserting 5. So, the remaining elements as it is now, the list will be extending to 2 elements for inserting 5 check with this element 8 is greater than 5. So, shifted. Now there is nothing, there's nothing to check. So, we have reached the 1st place. So, insert 5 at this place after inserting 5, The result is 5 than 8 still here, the list is remaining 7, 3, 2 they are not yet sorted. Let’s call this as a 1st pass.



2. Then second pass in second pass let us insert 7. In the list already, 5 and 8 are sorted, we are trying to insert 7, so list will extend till here and the remaining elements are these these are not yet sorted. So, 7, we are going to insert, so 7, take it outside, now for inserting 7 starts shifting the elements which are larger than 7 from right hand side 8 It is greater, 5 it's not greater, so 7 should be inserted here. So, the result after inserting 7 is. 5, 7, 8 and till 3 elements list is sorted. We will also analyze this one. See how many elements were compared, just one comparison, how many swaps means how many shifting. Only 1 element was shifted. So let let's say one swap. Now in 2nd pass how many elements were compared in our example, only one element was compared, but at most how many? Both 8 and 5 can move possible. If the number is a smaller than 5, then both can move. So let us write on maximum comparisons and the maximum swaps for maximum comparisons are two comparisons and the maximum swaps are 2 swaps.

Now we have to insert two more elements and sort two more elements. Let me continue with the rest of the elements. The resultant array I will take it here.

**So, number of passes = (n-1) passes**

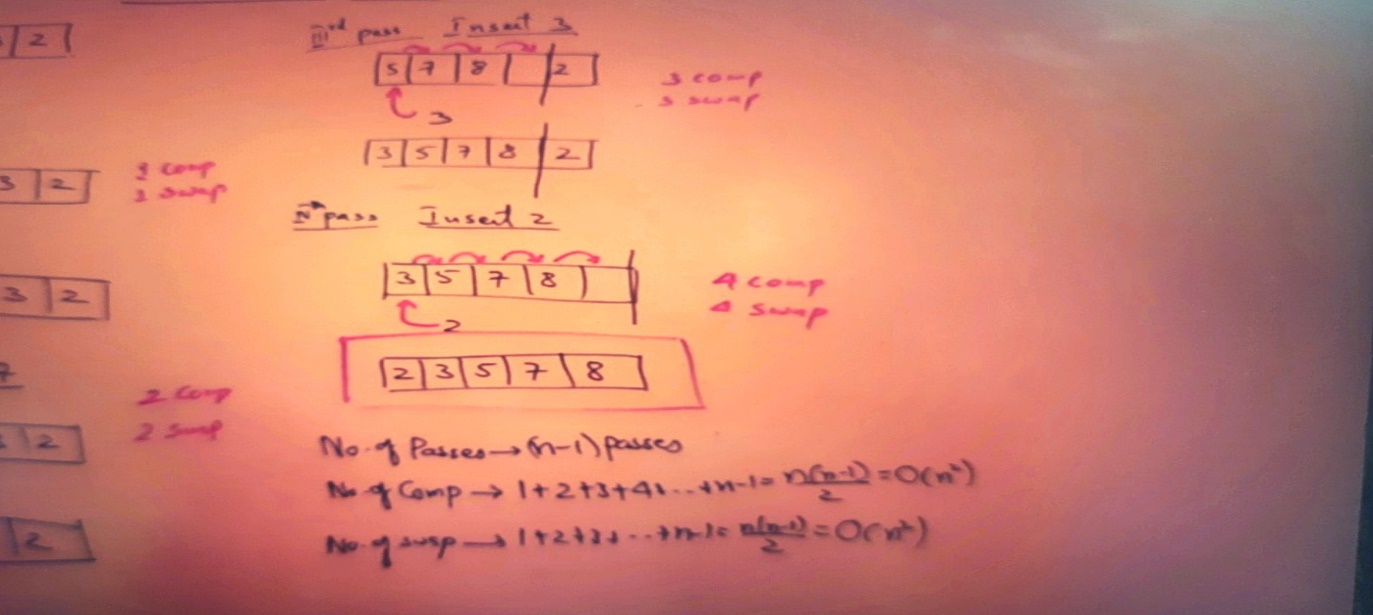
**No. of comparisons = 1+2+3+…..+n-1=n(n-1)/2= O(n^2)**

Like in bubble sort, if you perform 1 pass, you get 1 largest element for 2 pass u will get 2 larger elements. But in insertion sort, you will not get anything useful.

We have seen insertion sort in an array as well as insertion in a LinkedList so in an array we have to shift the elements but in linked list we don’t have to shift anything. So, the benefit of insertion sort upon LinkedList is that you don't have to shift the elements without shifting you can insert in LinkedList.

So, insertion sort is more useful or more compatible with the linked list than Array. Or, I should say that Insertion sorts are designed for LinkedList.

So, always remember for sorting linked list insertion sort is better.



Let’s code---

So here is a function for insertion sort which is taking array and number of elements. In our example size n=5. But indices are from 0 to 4 only.

Now 1st is for loop for number of passes, how many passes for n elements (n-1) passes.

But if you see the passes the first pass which element they are trying to insert, that is second element, and that index is one.

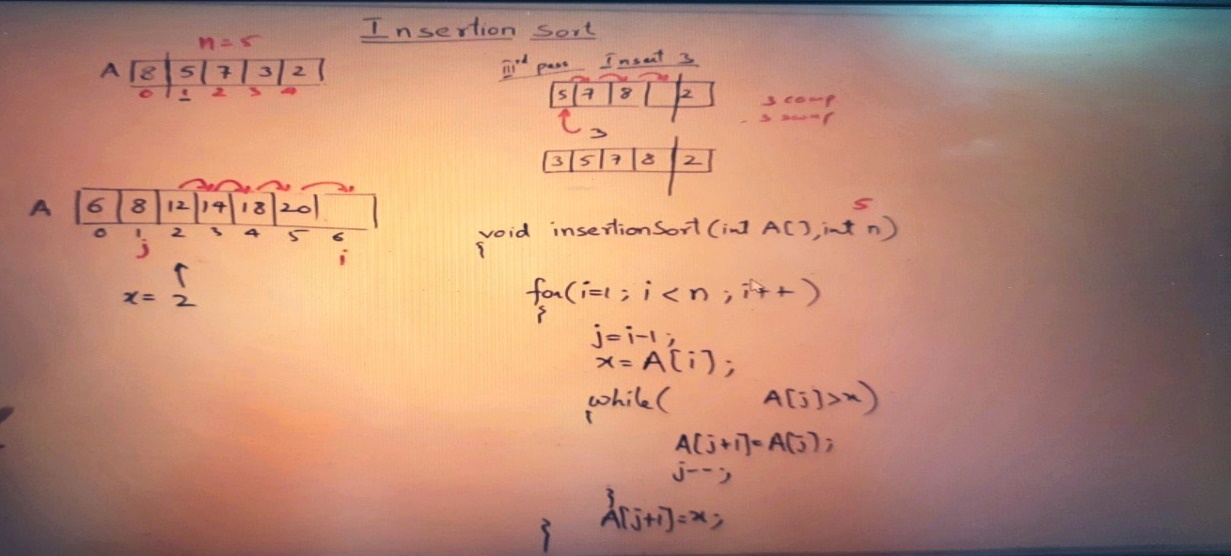
So, we don’t have to start from 0 we have to start from 1. So, insert the elements from index 1 then 2 then 3 then 4. So, it will start from 1 and it will go till 4 till less than n. So, from one to four, because at zeroth index we have the first element and that we are assuming it's already inserted and sorted. So this loop for repeating passes…

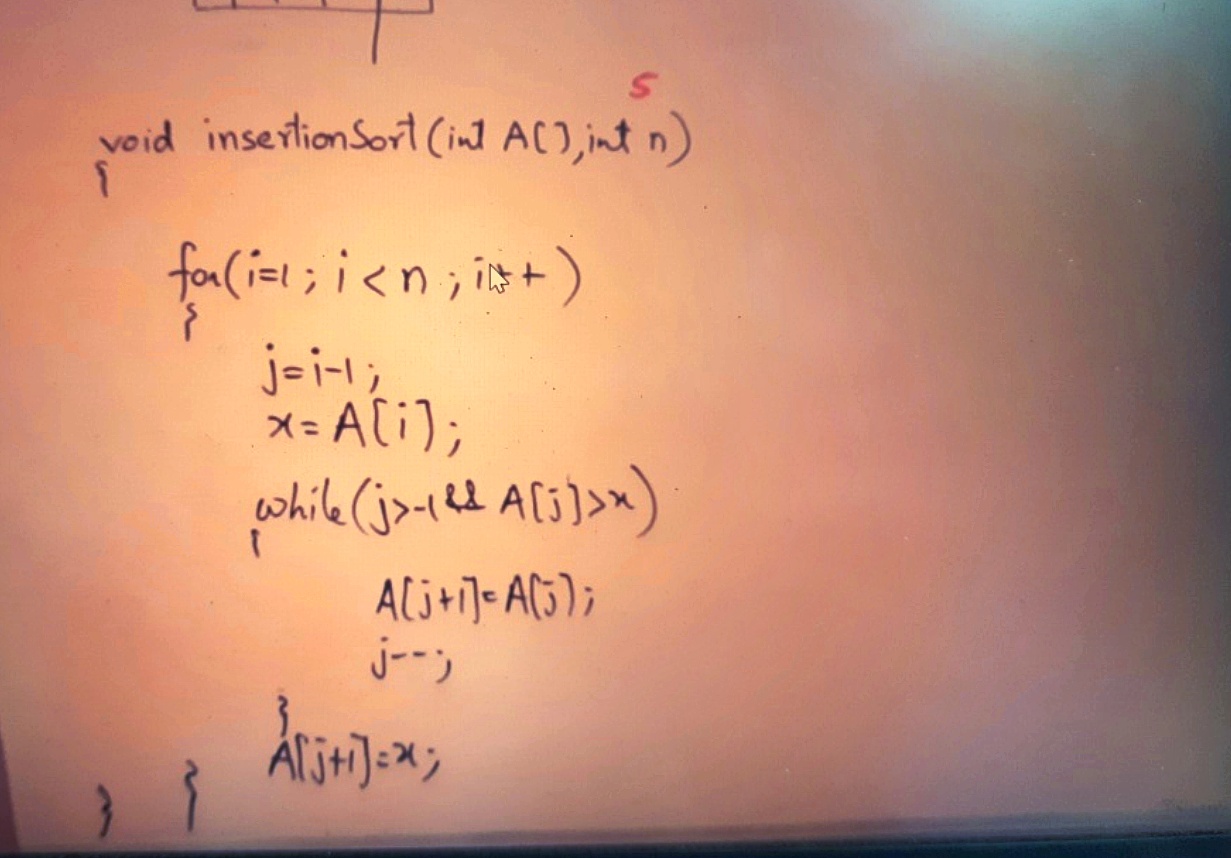
Then in each pass what we are supposed to do ….

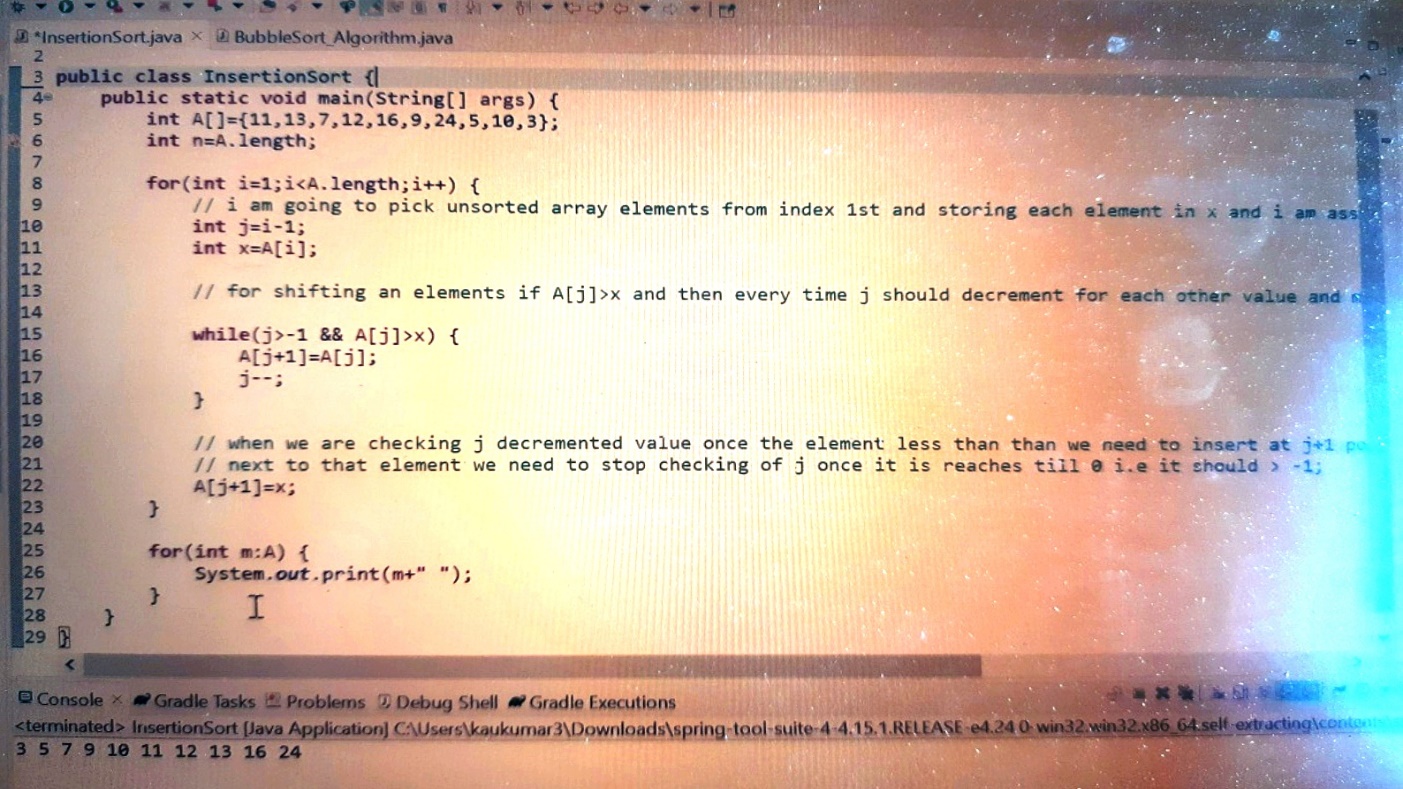
So, first take first element of index 1 in some variable x which we need to compare with the sorted element which is less than the 1st index value. so, x=A[i]…

Now I will take another pointer j which is less than I will start from 1st index , we will compare 1st element with 0th index if 0th index i.e sorted array element is greater than x or 1st index value then we need to shift right if not then we need to insert …

So, the process is simple Go on and shifting the elements as long as A[j] is greater than X. so, j always be a less than i-1…

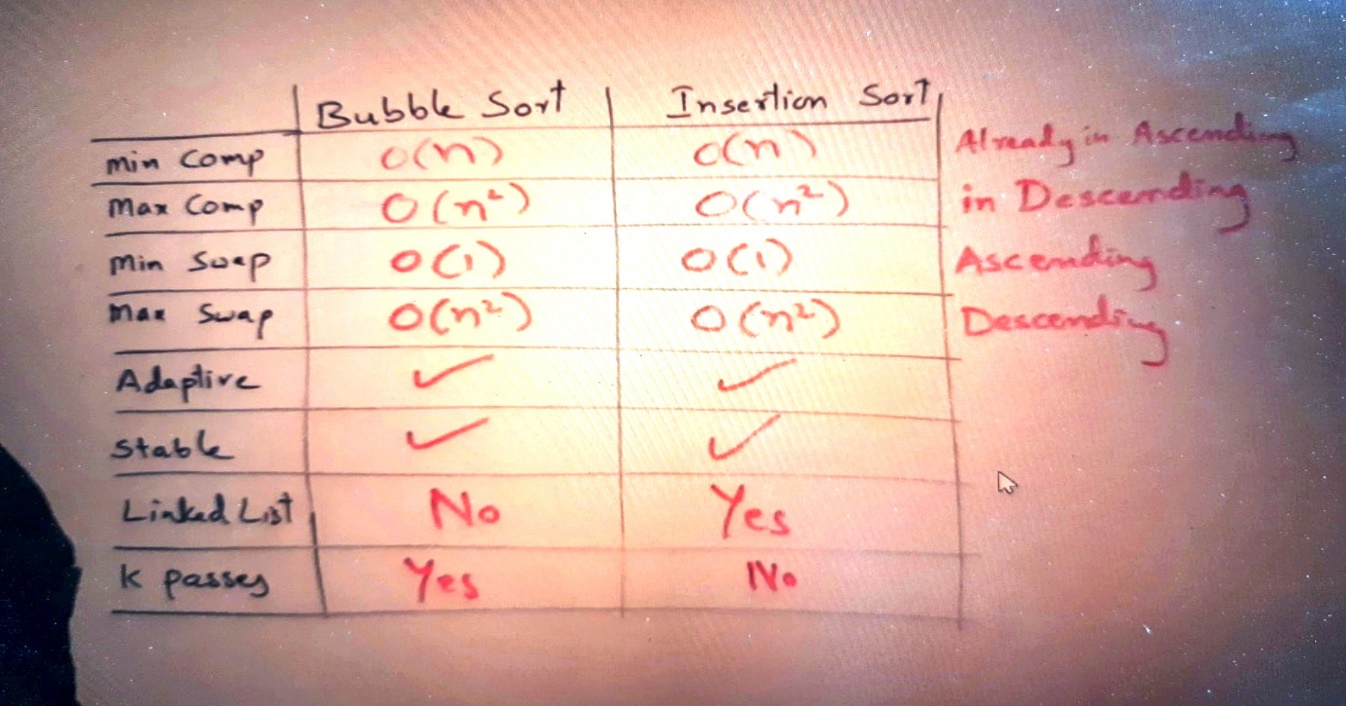






**Comparison between Bubble sort and Insertion Sort**

**===========================================**



**Selection Sort**

This algorithm also sorts the elements in passes. In each pass one element will be sorted.

List = **8,6,3,2,5,4**

In 1st pass we will select first position to find out an element for that position.

So, let’s take i here for a position so i is pointing on a position that is index 0 now, now who should come at this place, so the smallest element in this list should come at that place. So, find out minimum element so find out whichever element is minimum .

So, for finding out minimum element we will take 2 pointers k as well as j to both of them will be on same position wherever i is pointing.

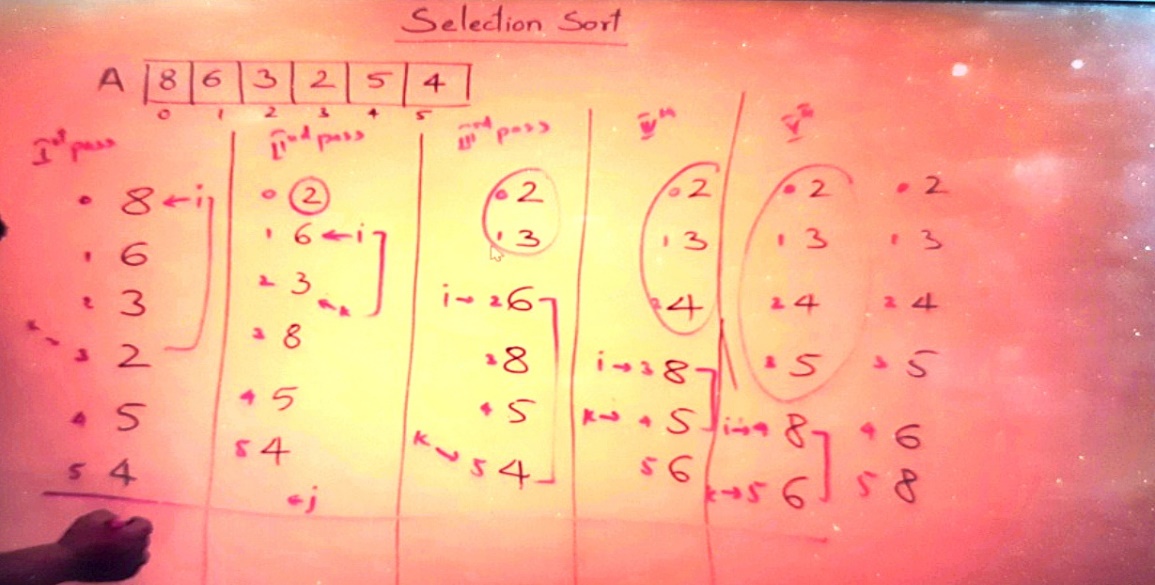
**Now let’s us see how to find minimum ----->**

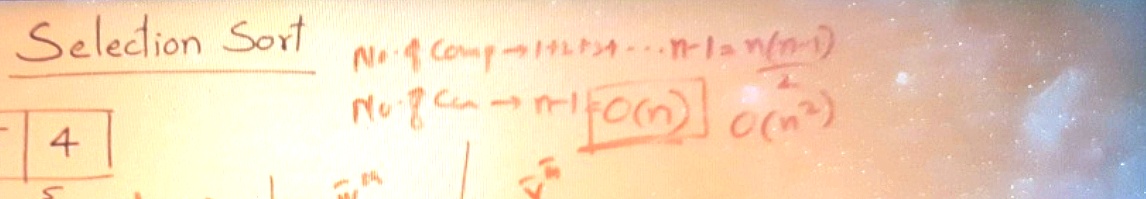
Move j to the next element and check whether this is smaller than where k is pointing if yes bring k at this place so j will find out minimum element and if any element is found it will bring k there continue j move to the next location now check is j is the small element where k is pointing then move k to j position now move j to the next position , is it smaller than the element where k is pointing yes so call k here this is the smaller element we found. Now move j to the next element is it smaller than the element where k is pointing yes so call k here. now j move to the next element is it the smaller element where k is pointing no, move to the next position Is it the smaller element where k is pointing no, so don’t move k. continue so move j to next but it is out of the list, so we have finished the entire list… and found out a smallest element where k is pointing…

So, just observe one thing, just we found out which is the minimum element, and we got this index. We have not swapped any element, we didn’t swap any element.

Then Interchange the element from i to k…

Then the new list will be **2,6,3,8,5,4**





**Important points**

* We were selecting a position and finding an element for that position…
* It is just like Imagine there are some students in a class, and this is the first seat in the class I have to find out who should seat here…so I selected the seat and finding out who should seat here.
* So, we selected a position 0 that is first position and find out which element should come there for that I am using 2 pointer j & k
* So, I found in entire list 2 was minimum so 2 was brought there…
* So, selecting a position and finding an element.

**Analysis:**

Let’s find number of comparison and swaps…

In 1st pass we compared almost all elements so total 6 elements so 5 comparisons done, and no. of swap is only 1.

Total elements = 6

In 1st pass 5 comparison and 1 swap.

In 2nd pass 4 comparison and 1 swap.

In 3rd pass 3 comparison and 1 swap.

In 4th pass 2 comparison and 1 swap.

In 5th pass 1 comparison and 1 swap.

**So, No. of comparisons = 1+2+3+4……...n-1 for n elements = n(n-1)/2 = O(n^2)**

**Time complexity = O(n^2)**

In each pass we are performing only 1 swap

**No. of swaps are = n passes = n-1 swap = O(n)**

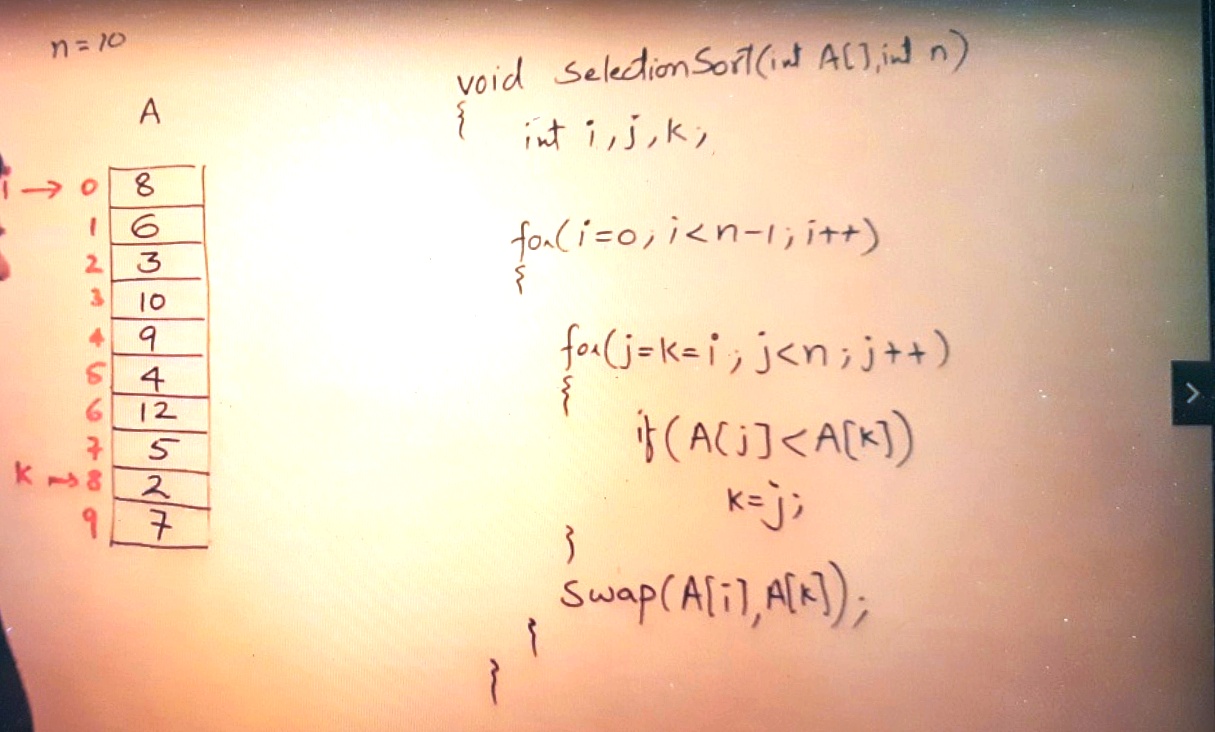
Imp->

**In selection sort for n elements, we do have only n-1 swaps. So, this is the only algorithm which sorts the elements with minimum number of swaps.**

To avoid swaps we have taken 2 pointers j as well as k we find the index once u found the index we interchanged will not be interchanging every time. So, we can avoid interchanging of elements with the help of extra variable k.

**So, selection sort is good for less number of swaps.**

If I perform only one pass, we got smallest element. If I perform 2 passes, I got 2 smallest elements which means if I perform if I pass k passes, I will get k number of smaller elements. So, just like bubble sort where we were getting k largest elements, in selection sort we will get k smallest element. So, intermediate result of selection sort is useful.



Not an Adaptive and not a stable.

