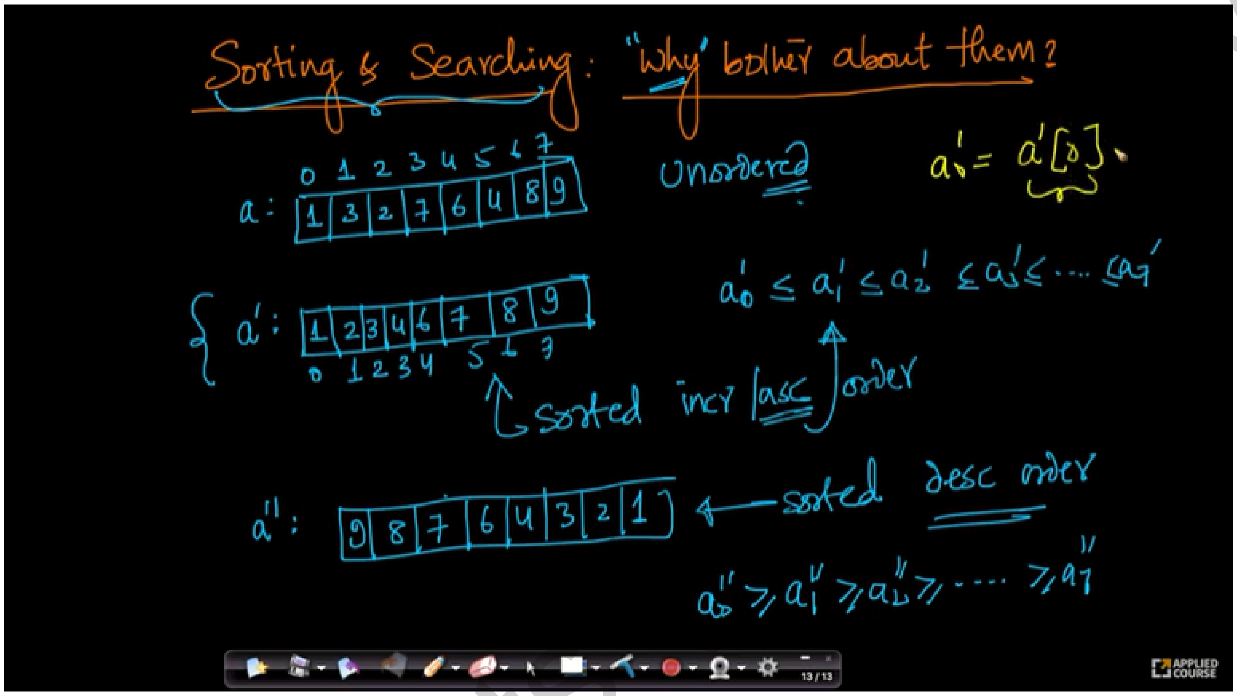
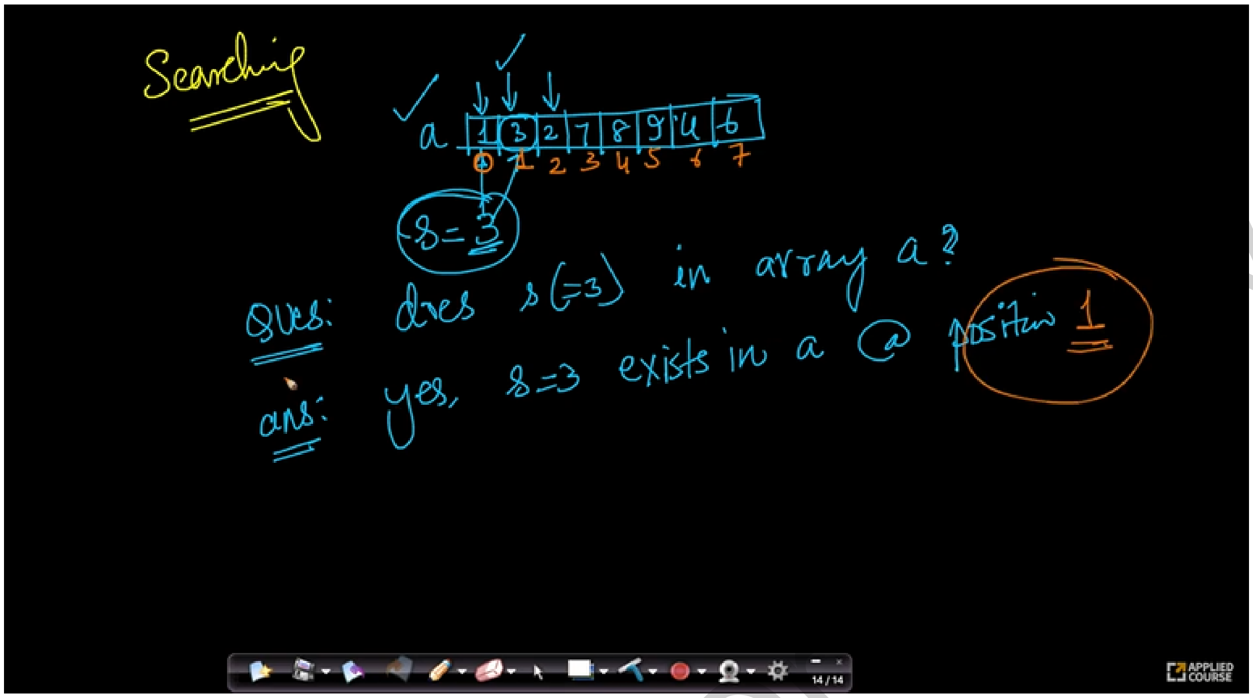
8.1 Sorting & Searching : why bother with these simple tasks?


Sorting & Searching: "why" bother about them?
a: 1 3 2 7 6 4 8 9
unordered
$a'_0 = a'[0]$
a': 1 2 3 4 6 7 8 9
$a'_0 \leq a'_1 \leq a'_2 \leq a'_3 \leq \dots \leq a'_7$
sorted incr / asc order
a'': 9 8 7 6 4 3 2 1
sorted desc order
$a''_0 \geq a''_1 \geq a''_2 \geq \dots \geq a''_7$

Timestamp: 3:57

Sorting: Rearranging the elements of an array in either ascending order or descending order.


Searching
a
1 3 2 7 8 9 4 6
0 1 2 3 4 5 6 7
s = 3
Ques: does s(=3) in array a?
ans: yes, s=3 exists in a @ position 1
14 / 14
APPLIED COURSE

Timestamp: 7:21

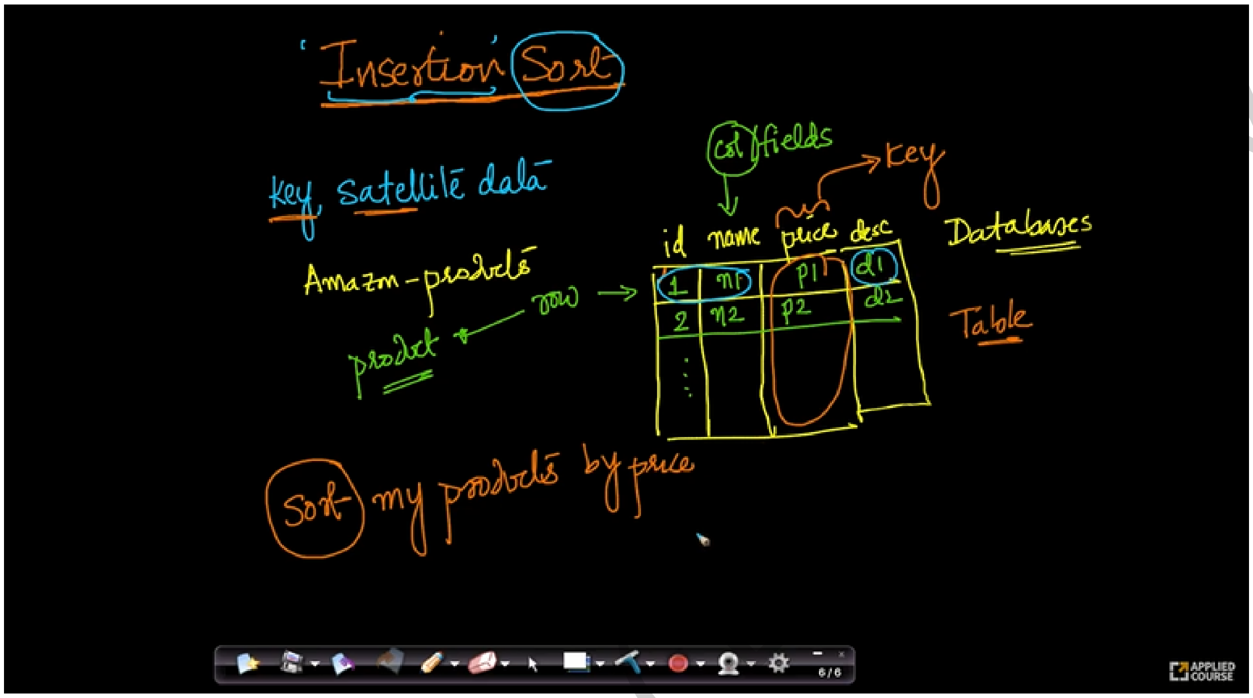
Searching: It is the process of finding whether an element occurs in an array or not.

Why bother about Sorting and Searching:

1) Searching and sorting are widely used in a lot of real world scenarios for example while Search for a product on amazon, searching for a friend on facebook, sorting the products based on price in amazon search results, ranking the web pages in the google search results, etc.

2) These are the most basic operations based on which more complex things can be built.

8.2 Satellite Data and Keys

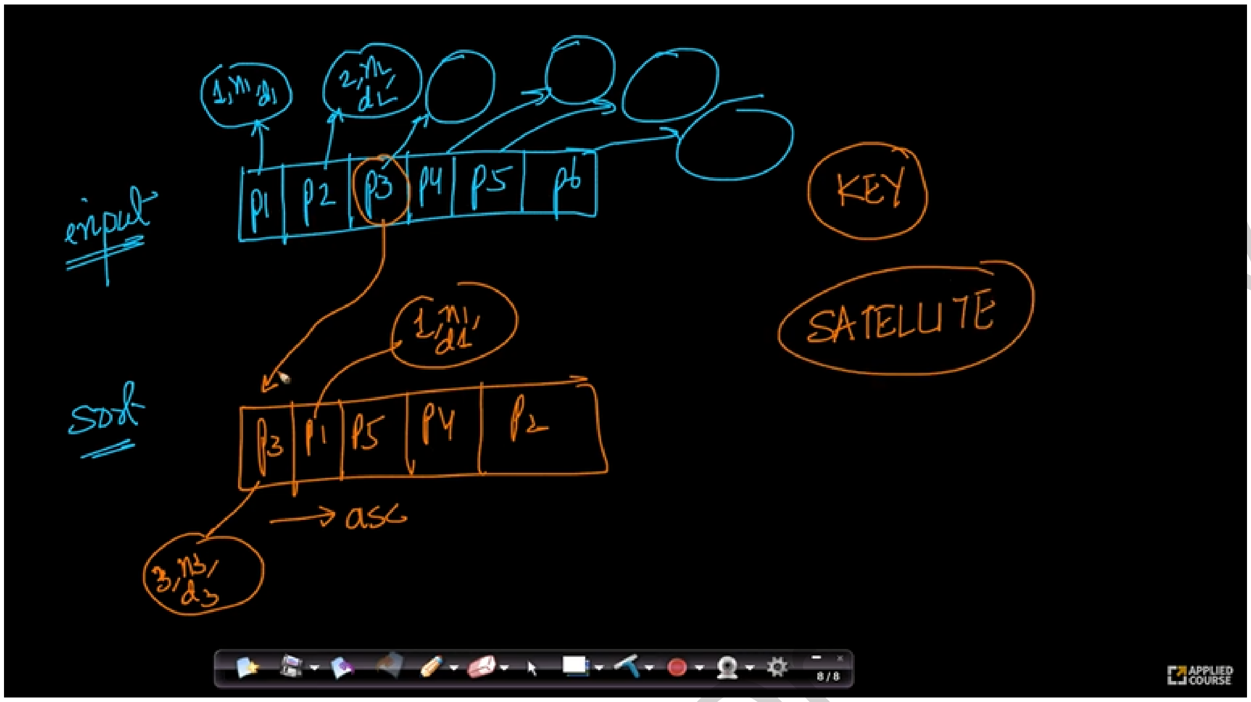


Timestamp: 3:51

Let us say we have amazon products data with id, name, price and description for each product. Suppose we want to sort the amazon products data by price, then price is the key and the rest of the data other than price is called the satellite data.

Key: The key, information or the aspect using which you want to sort the whole table is called the Key.

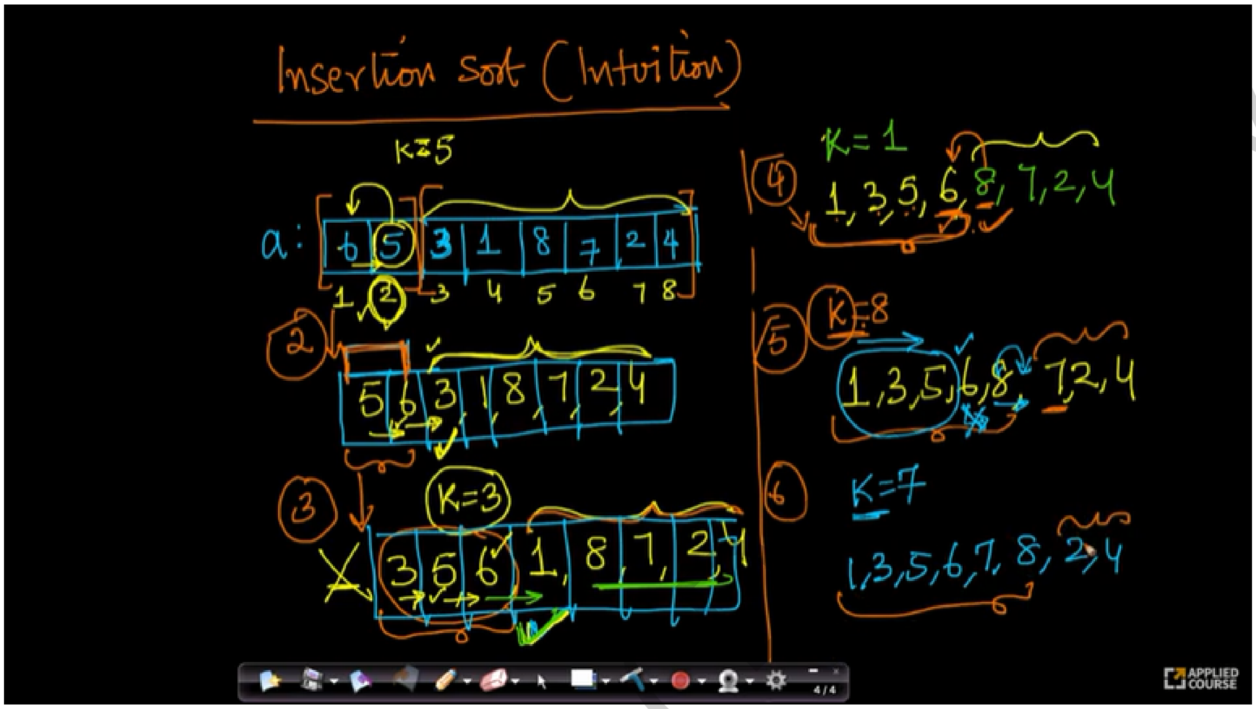
Satellite Data: The rest of the data that is not part of the key is called the satellite data.



Timestamp: 9:07

Whenever we sort an array using the key, the satellite data moves along with it, hence giving its name. (Just as how the moon, a natural satellite of earth, moves along with earth).

8.3 How it works: Card - sorting



Timestamp: 12:04

Note: Unless explicitly stated, sorting an array means sorting in ascending order.

1) Insertion Sort works by maintaining two subarrays, one subarray is the sorted part of the array while the other subarray is the non-sorted array.

2) It works by taking the first element of the unsorted array in every iteration then inserting it in the appropriate position in the sorted subarray.

3) When each element from the unsorted subarray is being added at the appropriate place in the sorted subarray, the size of the sorted subarray increases and the size of the unsorted subarray decreases until the sorted subarray contains all the elements of the array and the unsorted subarray has no elements.

4) When we move each element in the unsorted subarray to the sorted subarray, we start by first comparing the first element of the unsorted subarray (let's say K) to the rightmost element in the sorted subarray and then we move left along the sorted subarray and keep comparing them with K.

5) When moving from right to left along the sorted subarray and comparing the elements in the sorted subarray to K, we may come across two situations.

    a) K is less than the element that we are comparing.
    b) K is equal or greater than the element that we are comparing.

a) If K is less than the element we are comparing, then we move K to the left and move that element to the right.
b) If K is greater than or equal to the element we are comparing, then the sorted subarray along with the element K is perfectly sorted. Hence we can move on with the next element in the unsorted subarray.

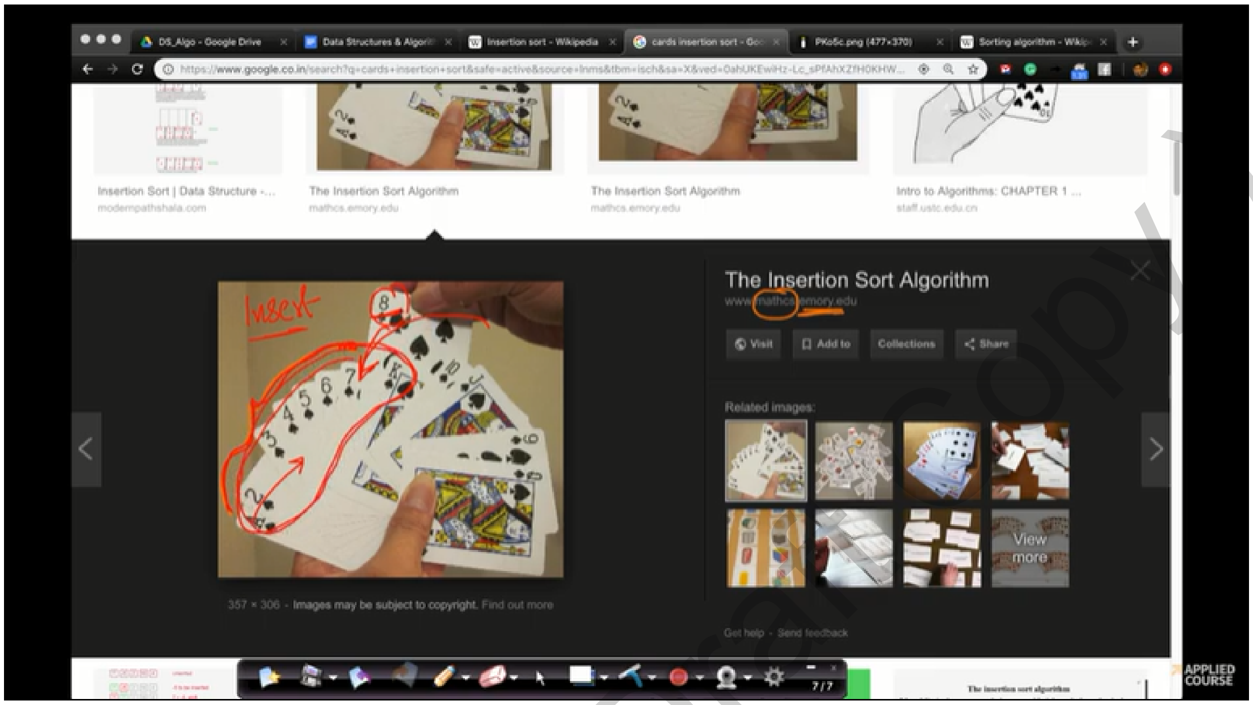
Let us look at both these condition through the below example:

ex) Let us say at some point in our insertion sort we have the array as follows
Where array = [**1,3,5,6,8,**7,2,4] where [**1,3,5,6,8**] as the sorted subarray and [7,2,4] as the unsorted subarray

Now since 7,the first element of unsorted subarray, K, is less than 8. We come across condition a.

Hence the array becomes:array= [**1,3,5,6,7,8,**2,4]

Now since 7, is greater than 6. We come across condition b. Notice that 7 has found its perfect place in the sorted array. Hence we move on with the next element in the unsorted array and make K=2.

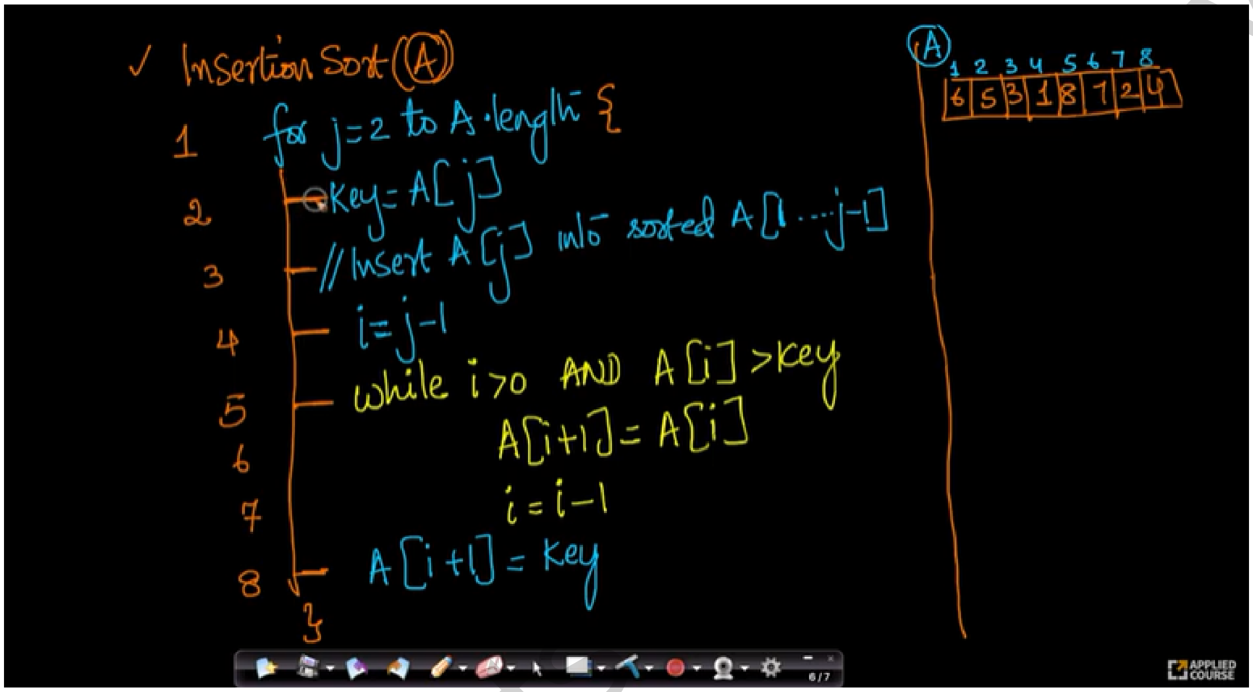


Timestamp: 17:56

When we sort a pack of cards, we sort them from left to right. At any time while sorting, let us say we have card A,2,3,4,5,6,7 already sorted, now we to **insert** the current card 8 in its right position in the sorted array, thereby giving the name of this sorting algorithm **Insertion sort.**

Please refer to the gif in the link https://en.wikipedia.org/wiki/Insertion_sort#Algorithm

8.4 Pseudo Code



```
✓ Insertion Sort(A)
    for j = 2 to A.length {
        key = A[j]
        // Insert A[j] into sorted A[1 ... j-1]
        i = j-1
        while i > 0 AND A[i] > key
            A[i+1] = A[i]
            i = i-1
        A[i+1] = key
    }
```

A

1	2	3	4	5	6	7	8
6	5	3	1	8	7	2	4

Timestamp: 8:04

Please try the above pseudo code for the given example on the right and check whether the array is sorted at the end of the pseudo code.

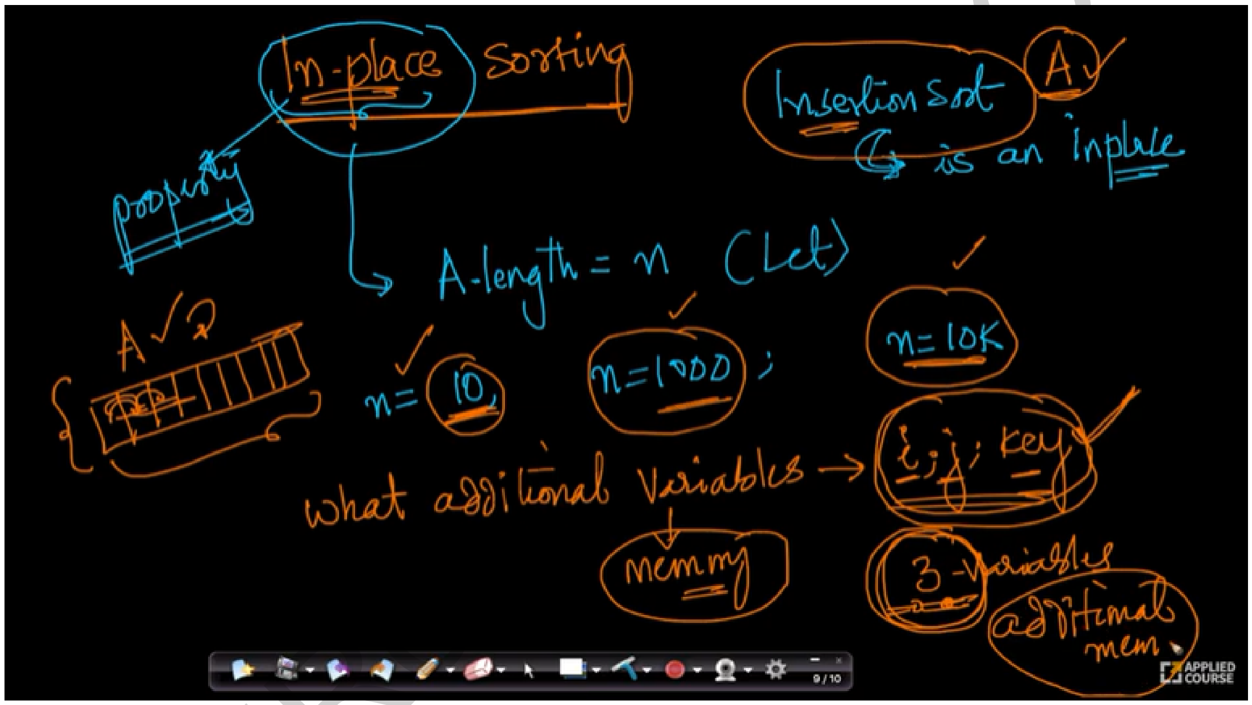
8.6 Correctness

Correctness: In the case of insertion sort, we want to prove that the insertion sort algorithm works perfectly to sort an unsorted array.

One argument to improve the correctness of insertion sort is as follows: At any time during the insertion sort, we have an already sorted subarray and we are taking the first element from the unsorted subarray and inserting it in the sorted subarray at the appropriate place where it belongs such that the already sorted subarray along with the new added element becomes a new sorted subarray.

As the insertion sort progresses, we are adding more and more elements into the sorted subarray and reducing the elements of the unsorted subarray until we are left with the sorted subarray that contains all the elements of the original array and the unsorted subarray doesn't contain any elements. Thereby proving the correctness of insertion sort.

8.7 Inplace Sorting



In Insertion sort, apart from the original array that is to be sorted we are only using three additional variables (i,j, key in the pseudo code) irrespective of the size of the input array.

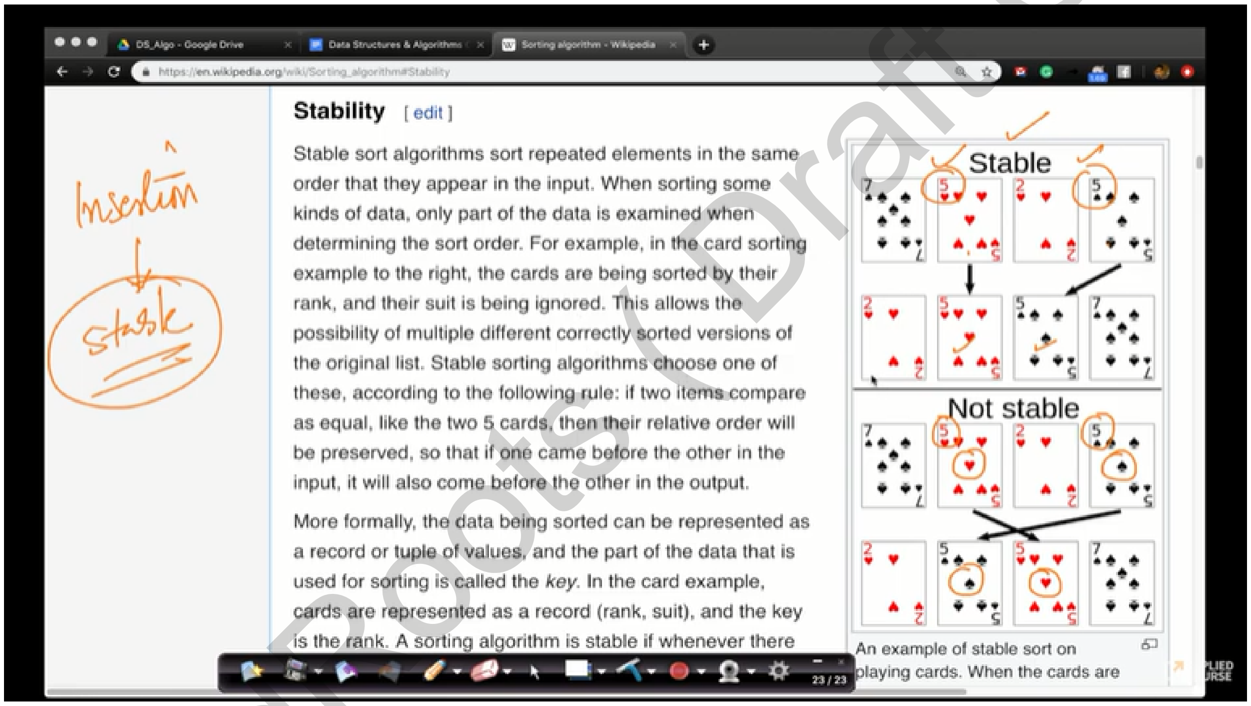
So by doing this, we are using only an additional memory for 3 variables apart from the memory used by the input array. This number(3) of variables is independent of what the size of the input array.

Such algorithms which consume a **constant** amount of **additional** memory irrespective of the input size are said to be inplace(meaning: within itself) algorithms. Since Insertion sort agrees the above, it is an inplace algorithm.

8.8 Stable Sort

When the initial order of the element having the same key value is retained even after sorting using a sorting algorithm then such an algorithm is called a stable sorting algorithm and is said to have stability.

Insertion sort is a stable sorting algorithm.


DS_Algo - Google Drive
Data Structures & Algorithms
Sorting algorithm - Wikipedia
https://en.wikipedia.org/wiki/Sorting_algorithm#Stability
Insertion
stable
Stable
Not stable
23 / 23

Stability [edit]

Stable sort algorithms sort repeated elements in the same order that they appear in the input. When sorting some kinds of data, only part of the data is examined when determining the sort order. For example, in the card sorting example to the right, the cards are being sorted by their rank, and their suit is being ignored. This allows the possibility of multiple different correctly sorted versions of the original list. Stable sorting algorithms choose one of these, according to the following rule: if two items compare as equal, like the two 5 cards, then their relative order will be preserved, so that if one came before the other in the input, it will also come before the other in the output.

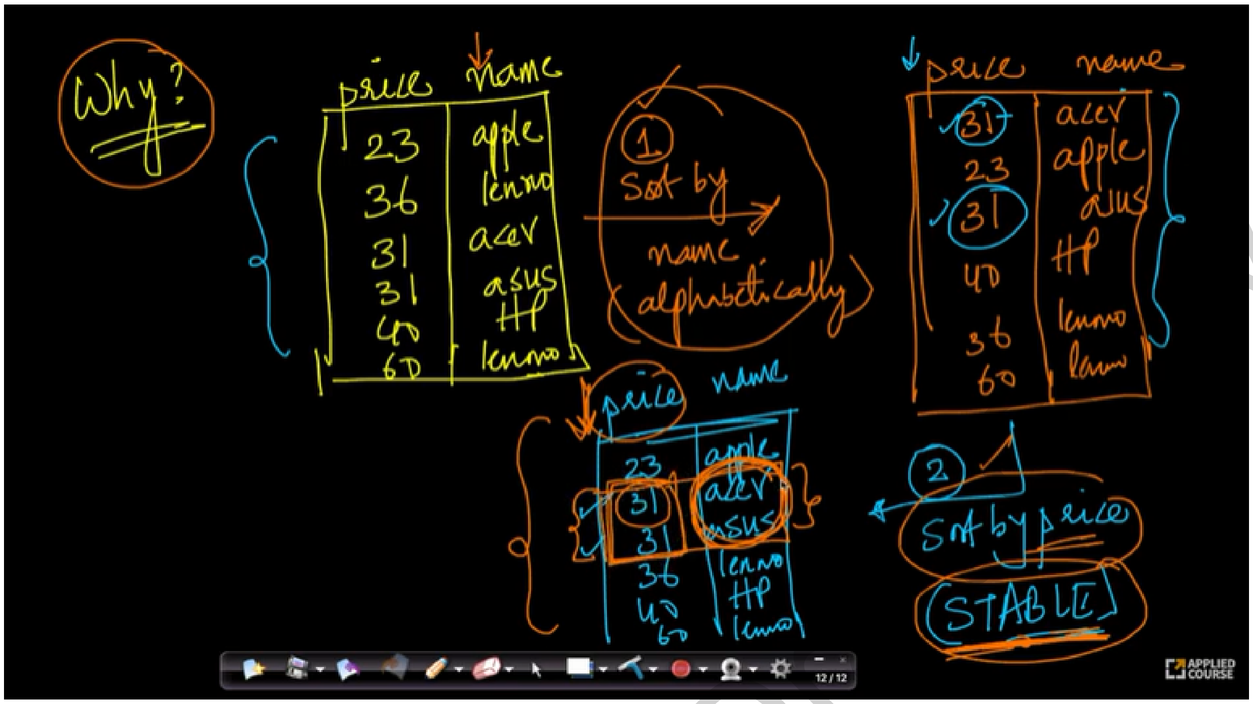
More formally, the data being sorted can be represented as a record or tuple of values, and the part of the data that is used for sorting is called the *key*. In the card example, cards are represented as a record (rank, suit), and the key is the rank. A sorting algorithm is stable if whenever there

An example of stable sort on playing cards. When the cards are

Timestamp: 5:26

Notice in the above diagram, for a stable algorithm that uses the number on the card as the key for sorting, the initial order of the 5 of hearts and 5 of spades is maintained even after sorting.

Whereas a non-stable algorithm **doesn’t guarantee** that the initial order will be preserved post sorting as in the example of the above figure.



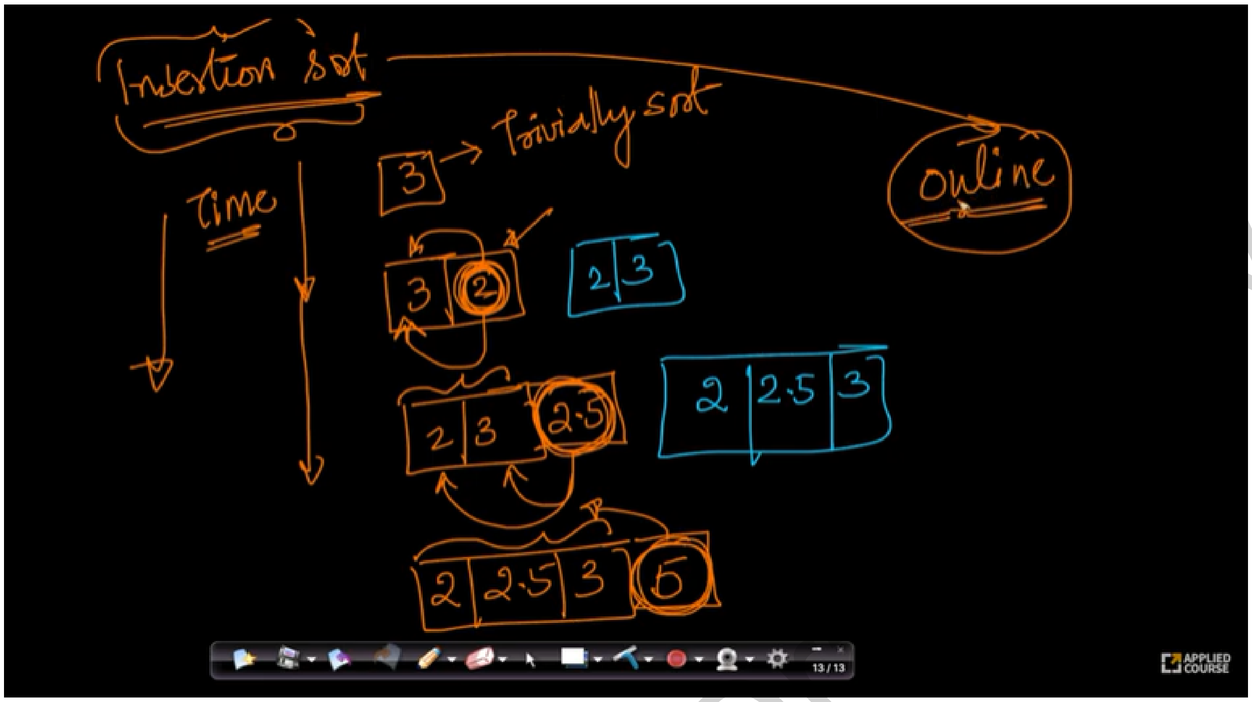
Timestamp: 14:15

Why use stable sorting algorithms: In real world there might be multiple scenarios like the one above, where having the data sorted by price, you also wanted your end results to have the data sorted by other column, in this case, the name of the product.

In these cases you need a stable sorting algorithm like insertion sort that maintains the initial order of the products (in this case the order of Acer and Asus) that have the same key value (here price of 31).even after sorting.

8.9 Online Sorting

Many at times, in real world cases such as the case of online cab services, the data that needs to be sorted may not be given to us at a time. Rather, the data is given to us over a time. If our sorting algorithm can sort data that is given to us over time then the sorting algorithm is said to be an online sorting algorithm.



Timestamp: 9:48

For example as shown in the above figure, let us say we were sent expected time for reaching us by different cab drivers and these expected times are sent to us over a period of time.
So at any time t, we wanted to sort whatever data we receive till then as shown in the above figure.

Insertion sort, in the way it works by inserting a new element into an already sorted subarray, it can be trivially converted to an online sorting algorithm. Insertion sort is an online learning algorithm that can sort the data as the data arrives over time.