Design and Analysis of Algorithms
Prof. Madhavan Mukund
Chennai Mathematical Institute

Module – 09
Lecture - 17
Sorting: Concluding Remarks

Let us look back on all the different sorting algorithms that we have considered.

(Refer Slide Time: 00:07)

So, one of the important criteria that we should not forget is that, sorting often happens in phases. So, we might have a list of names of people say Aswin, Bharathi, Chander and Deepa. So, we have sorted this in order and they might have got some marks in an exam. So, they may be got 60, 43, 60 and 95, now we might want to sort this by marks. So, when we sort this by marks of course, we need to exchange the position of Bharathi and Ashwin, but we do not want to exchange the position of Aswini and Chander.

So, in other words we do not want to disturb the sorting of alphabetical orders. So, the final answer should be Bharathi has 43, Ashwin has 60, Chander has 60 and Deepa has 95. But, it would be a wrong if you said Chander and then Ashwin are wrong, but undesirable. So, ==if we have a list of students in an alphabetical order and then we sort by marks, we expect that those with equal marks are still sorted in alphabetical order.==

So, in a spreadsheet where we have a kind of a table with columns, so supposing we sort by this column, and then we sort by this column, the second sorting should not disturb the order of the first sorting. So, this is called stable sorting that is there was, so very

often when you are sorting a sorting on one attribute, but there may be other attributes. So, data item is not typically a unique values, it is not just a number, but it is a... So, you are taking names, marks and various other addresses may be phone number and you might sort on difference.

So, think of a spreadsheet where we might have different columns. So, each successive sort should not disturb the previous sort, so this is called stable sorting. So, this is obviously desirable, because we do it all the time, we will be really would be very unhappy if the next round of sorting disturb the first round of sorting. So, whichever algorithm is stable.

(Refer Slide Time: 02:10)

So, quick sort as shown is not a stable operation, so remember that we took that at a bit and in the at least in our implementation what we did was we constructed this lower set and then the upper set to the right of it, so it is right, this is the picture we had and then finally, and this is the crucial point, we do this swap.

So, imagine that we have a situation where here we had for example, this point Ashwin with the 60 marks and Chander with also with 60 marks. So, at the point of constructing the lower set, they were in the correct order as in the original input. But, now after swapping what happens is that the pivot is come here and Chander with 60 marks has gone here. So, this swapping operation has basically reversed the order of A and C with this strike to dot it was in the input and henceforth therefore, this order will get jumped.

So, this swap operation during partitioning disturbs original order and this also happens
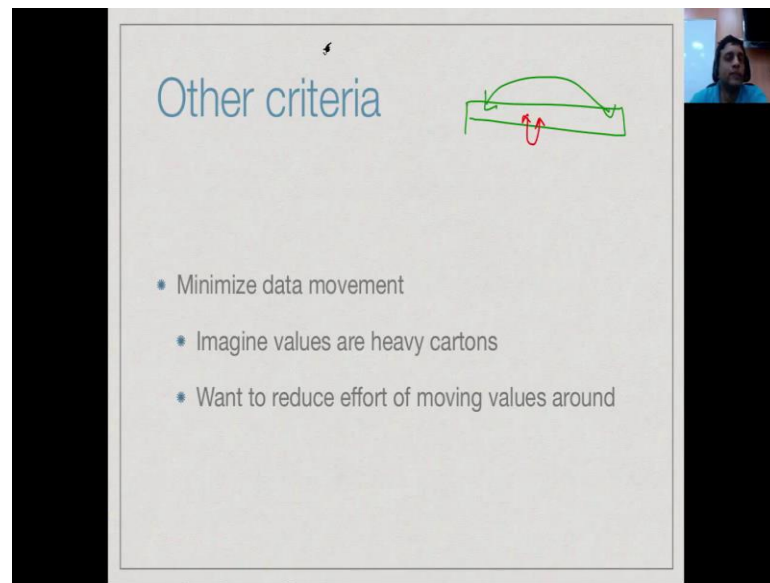
in things like selection sort. Because, in selection sort wherever we do this long distance things, so remember in selection sort we pick the minimum and then we exchange it to the beginning. Now, the if the element in the beginning have the same alphabetical order as, I mean it is smaller with, in the second one now it moves to later point in the array and the order get disturbed.

So, any kind of long distance swapping is very dangerous for stability. What about merge sort? Well, merge sort would normally we stable provided… all we want that something to the right should not go to the left of something in the original array. So, we do not want something… If you have two elements here and here which should remain in this order, we do not want them to be exchanged. Now, why they would be exchange? They would be exchanged only if they were equal, that is an only case where stability gets destroyed, where equal elements are disturbed, (Refer time: 03:57) to the previous sorting.

So, we have to make sure that no element in the right comes to an element to the left of something of from the earlier array and when you are merging, basically that is why we said that in the case, A i is less than or equal to B j, we pick from A and only… So, if you have made the mistake of writing less than and then we use greater than equal to, select from B, then when they are equal, we first pick from B and not from A and that would create instability.

So, crucial in our merge operation is that when A i is less than or equal B j we pick the element from A in preference to B, so this preserves the left right order of equal elements with respect to the original order and sense merge sort has be have done in stable. You can also check that similarly, if you do insertion sort carefully, then insertion sort is also stable and this is not to say that quick sort cannot be made stable, it is just at the way we have implemented it quick sort is not stable.

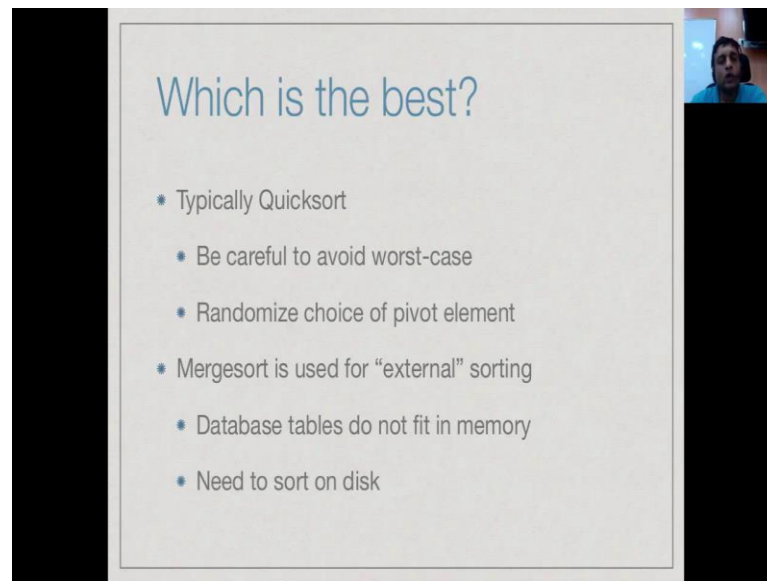(Refer Slide Time: 04:52)

And another criteria, which is related to what we just saw is that we have only concentrated on the number of steps required in order to compare and exchange two elements. But, you might want to do other things. For instance, you might want to penalize movements which take you across last segments of the array. So, you might want to say that it is not so good to exchange things for away, it is better to exchange things near each other.

So, something like bubble sort which we did not look at, which only exchange adjacent elements would be better than something like selection sort which exchange across large in to array. Now, this could happen if data has a cost and supposing you are sorting an array, but this array is across different, it is so big that it is not stored on a single server. When each time you want to swap from one server to another, you could pay an extra cost which is different.

So, in other words there might be different criteria which are behind the scenes which we do not see. So, imagine when you sorting something by hand, supposing you are sorting a bunch of heavy cartons, then you can imagine that moving of heavy carton a long distance is more expensive than moving it a short distance. So, there could be other criteria which govern your cost of sorting which we have not considered at all in this and people have looked at these things in the literature.
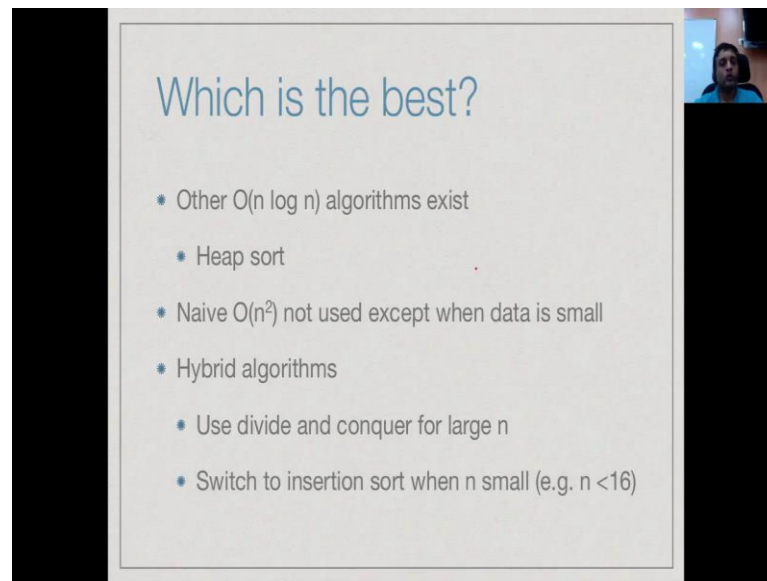
(Refer Slide Time: 06:05)

So, very often a question is asked which sort is best. So, unfortunately it turns out that no single sorting algorithm always guarantee to be the best, it really depends at, we said like sorting depth, the movement and other criteria. Some sorting algorithm may work well in some context, some in other contexts. In most, memory based context for simple arrays, quick sort is the best. This is why as we said, quick sort is usually the default implementation of choice for built in sorting algorithms. Provided we do something intelligent about choosing the pivot and various other elements.

On the other hand when we have to move a lot of data, sometimes we cannot store the entire thing if you are sorting a database. We cannot sort the entire thing in the memory, so actually we use a variation in merge sort which is called external merge sort. So, it really depends on the contexts, sometimes we will use one, sometimes we will use the other.

If they were a best algorithm, there would be no need to study the other algorithms. So, the very fact that many of these algorithms where studied, shows that in different context you might need to use one or the other idea in order to make things work better.

(Refer Slide Time: 07:10)

We have seen merge sort as an order n log n algorithm, there are other n log n algorithm we will see one later on this course for heap sort. The main thing to remember is that if you have a naive n squared algorithm which is not like quick sort provably n log n in the average case, then this will only work for small data. However, sometimes those small data, the simplicity of a naive algorithm beats the complexity of a complicated value.

So, you could even have hybrid algorithms, you might use divide and conquer where n is large and then the n becomes small then you switch over to may be insertion sort. So, there are many different strategies that people use and very often experimentally the array with some combination of strategies. So, it is not enough to say that one algorithm is the best, you need to have a good idea about what works and what does not works, so that you can tailor your sorting procedure to suit your requirements.