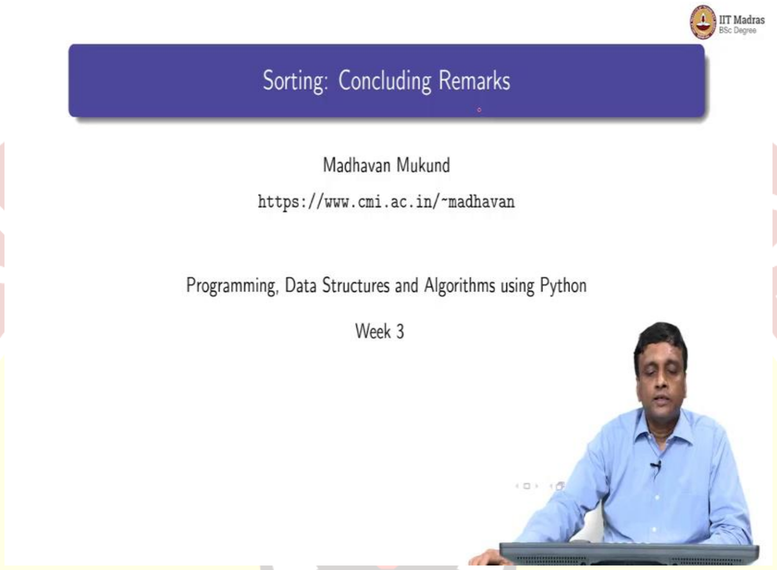
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
सिद्धिर्भवति कर्मजा

IIT Madras

ONLINE DEGREE

Programming, Data Structures and Algorithms using Python
Professor Madhavan Mukund
Concluding remarks on sorting algorithms

(Refer Slide Time: 00:09)



Sorting: Concluding Remarks

Madhavan Mukund

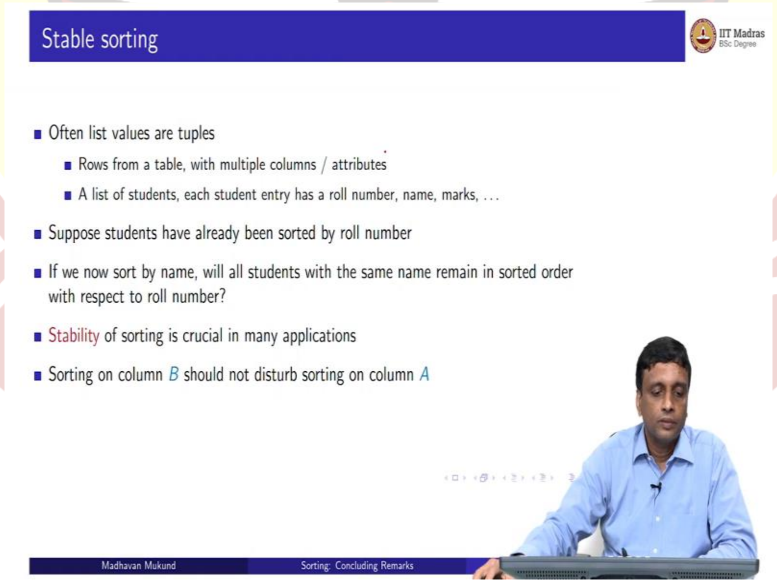
https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 3

So, let us conclude our discussion on sorting by looking at some other issues.

(Refer Slide Time: 00:13)



Stable sorting

- Often list values are tuples
  - Rows from a table, with multiple columns / attributes
  - A list of students, each student entry has a roll number, name, marks, . . .
- Suppose students have already been sorted by roll number
- If we now sort by name, will all students with the same name remain in sorted order with respect to roll number?
- Stability of sorting is crucial in many applications
- Sorting on column B should not disturb sorting on column A

Madhavan Mukund   Sorting: Concluding Remarks

So, one of the issues is something called stability. So, very often we are not sorting single values, but we are sorting compound values. So, imagine that we have a table and we are sorting the rows in the table. So, each element that we want to sort corresponds to one row. So, one row usually corresponds to values from many columns. So, it is a tuple of values. It could be, for example, a list of students, and each student has along the columns, you have roll number, you have the name, you have the marks, and so on.

So, you can think of what you are sorting is actually a spreadsheet. So, your list is a list of rows. But each element in your list consists of some sub-parts. So, now when we sort, we typically sort on one or more of these sub-parts. The sub-parts are attributes or columns. So, you might want to sort by name or you might want to sort by roll number or you might want to sort by marks.

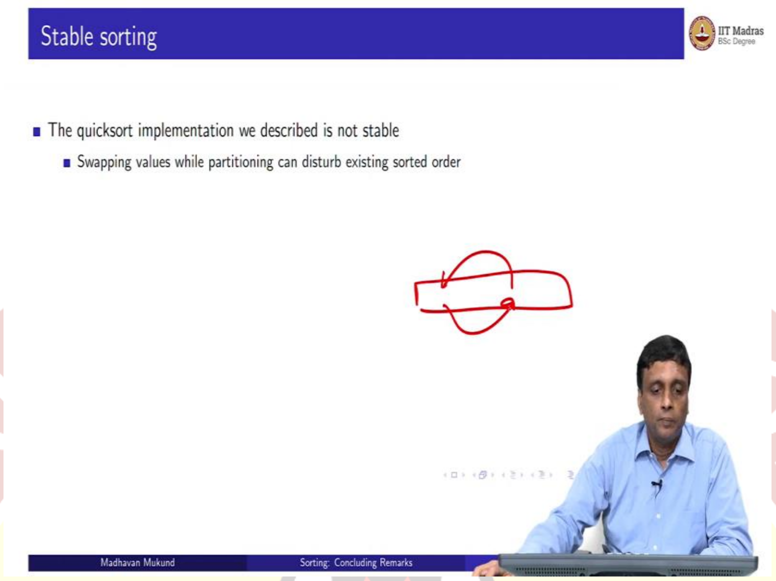
And quite often you might want to sort something which has already been sorted. So, for instance, it could be that your list is provided to you by roll number. And now, for whatever reason you want to group the students in alphabetical order of name. So, you sort them by alphabetical order of name.

Now, the question is, when I sort by alphabetical order of name, I might get two students who have the same name. They would have had a roll number. In the original sorting one of them would have a smaller roll number than the other. In the new sorting by name, are the roll number still in the same order or not or have they got shuffled.

So, this is something that we want to know. So, for example, supposing you have sorted in alphabetical order of name and now you want to sort by marks, now when you get the same students with the same marks, are they sorted in alphabetical order of name or do they get shuffled? So, this is called stable sorting.

So, stability says that when I sort in another column, it should not disturb the sorting on a previous column. So, later sorting should not disturb an earlier thing. And this is something which is quite natural if you actually work with sorted things in spreadsheets and all that. If you sort on something, you do not want to go back and figure out that something else has got sorted badly that you had already sorted in a previous column.

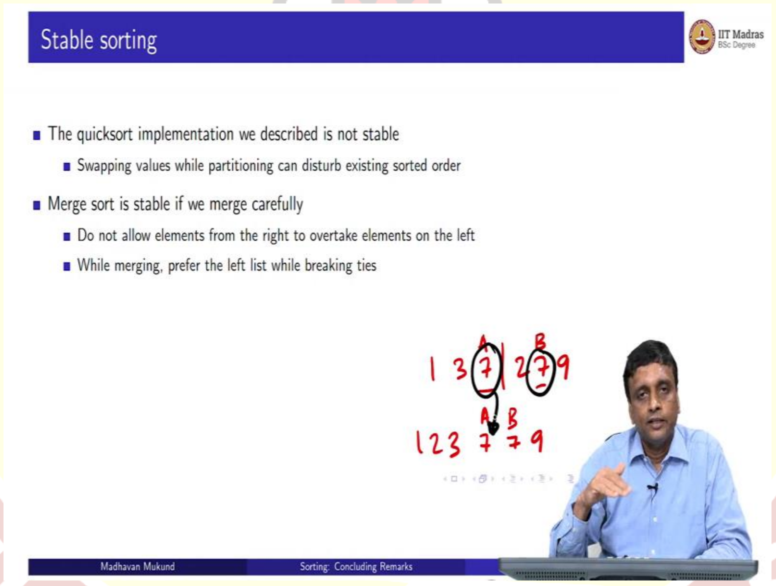
(Refer Slide Time: 02:24)



Stable sorting

- The quicksort implementation we described is not stable
  - Swapping values while partitioning can disturb existing sorted order

Madhavan Mukund   Sorting: Concluding Remarks



Stable sorting

- The quicksort implementation we described is not stable
  - Swapping values while partitioning can disturb existing sorted order
- Merge sort is stable if we merge carefully
  - Do not allow elements from the right to overtake elements on the left
  - While merging, prefer the left list while breaking ties

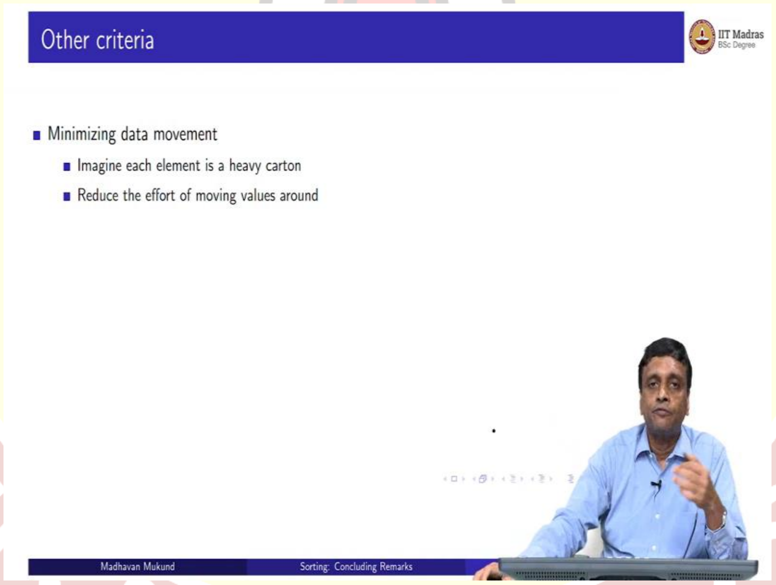
Madhavan Mukund   Sorting: Concluding Remarks

So, unfortunately, the quicksort implementation that we looked at is not going to be stable, because it does all this, if you remember, you suddenly do these kinds of exchanges. So, if these values originally were in some order with respect to some value you are not using right now for sorting and if, while partitioning your sort of swapping things around, then you might very well be exchanging things which are equal in some other key and this is not good. So, the partitioning of quicksort has to be very carefully done to make sure that it is stable. It is possible to do it, but not the partitioning that we saw.

Merge sort is easier to do. Basically, what you want to make sure is that when I have something like 1, 3, 7 and merge it with, say, 2, 7, 9. So, maybe this had an A and this had a B with it. So, I want to make sure that in the final list the A7 comes before the B7 this is what stability means. It means that in the final list I want 1, 2, 3, 7, 7, 9, but the order of the 7s should be in the same order in which I was presented to me in the beginning. I should not have 7B followed by 7A, even though A and B are not part of my current sorting procedure. So, I am not looking at the A and B, but I want to guarantee that this does not happen.

So, in merge, what you do is, basically, if I see the same value on both sides, then I will move the one from the left first. So, the value from the right should not overtake the value on the left. If you guarantee that that your merge has this kind of preference for the left list, when you compare equal values, then you will get a stable merge.

(Refer Slide Time: 04:01)


Other criteria

- Minimizing data movement
  - Imagine each element is a heavy carton
  - Reduce the effort of moving values around

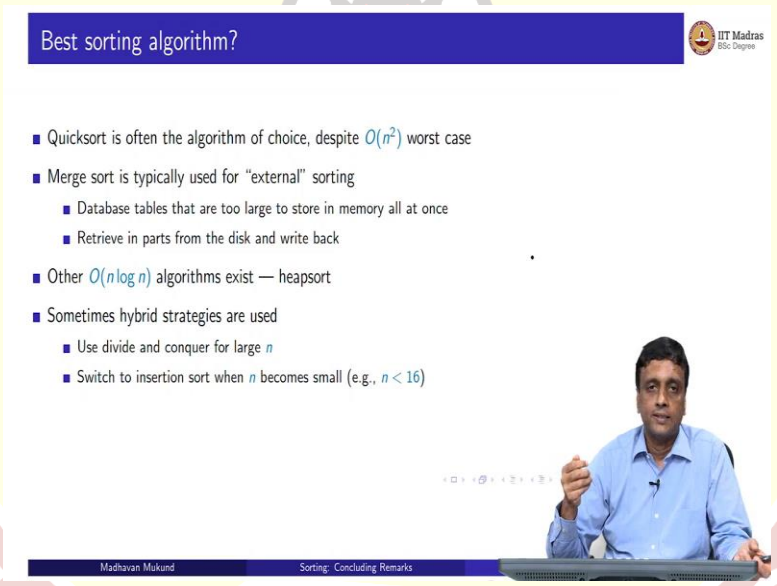
Now, we have looked at basically criteria which involve the number of comparisons and swaps that we make. But there are situations where actually the data has to physically be moved and it is not an easy thing to move data from one place to another. So, you can imagine that the data sitting in some heavy boxes.

So, supposing you are doing a physical sorting or something, then one of the things that you might want to minimize is not just the number of boxes you examine and exchange, you might be willing to look at more boxes to examine and exchange if you have to move your boxes overall.

So, data movement is an orthogonal thing compared to anything that we have discussed so far. But this is also an important criterion when looking at certain categories of sorting. So, sorting is a fairly vast. As we said, sorting is used as a first step for a large number of things and there are many different dimensions to sorting which people have looked at.

So, you should not think that if you know merge sort and quicksort you have exhausted everything to do with sorting. There is a lot of sorting which is still there and many different ways of sorting which emphasize different aspects of the problem.

(Refer Slide Time: 05:08)



Best sorting algorithm?

- Quicksort is often the algorithm of choice, despite $O(n^2)$ worst case
- Merge sort is typically used for "external" sorting
  - Database tables that are too large to store in memory all at once
  - Retrieve in parts from the disk and write back
- Other $O(n \log n)$ algorithms exist — heapsort
- Sometimes hybrid strategies are used
  - Use divide and conquer for large n
  - Switch to insertion sort when n becomes small (e.g., $n < 16$)

So, is there a best sorting algorithm? Well, as you can see from our discussion, the hints are quite clear that there is no best sorting algorithm overall. Quicksort is, as we said, very often the algorithm of choice when we are using built in functions and all that people use quicksort, despite its worst case.

But there are situations, like we were talking about those boxes, where you do not have the luxury of sorting everything as a function. So, sometimes you need to sort, example in a database, and in a database you need to put parts of the data into memory and part out. So, this is called external sorting.

So, usually, merge sort is used for external sorting. And merge sort is not the only n log n algorithm. There is also an algorithm which we will see later called heapsort, which also does n log n. So, there are other n log n algorithms. And very often, you might actually use a hybrid strategy. So, you might use a divide and conquer strategy when n is large. But then when you get down to a small list, say like 16 or 32, you might switch over to insertion sort.

So, a lot of different ways are there of combining algorithms or using the properties of, sorting algorithm for different situations. So, you, it is useful to know that all these exist. Although in many cases, you will typically be using a built-in sorting routine. So, you do not really need to know the sorting algorithm as long as a built-in sort function works. But there are situations where you will need to sort for yourself and then it is useful to know that there are these different options and they all have their pluses and their minuses.