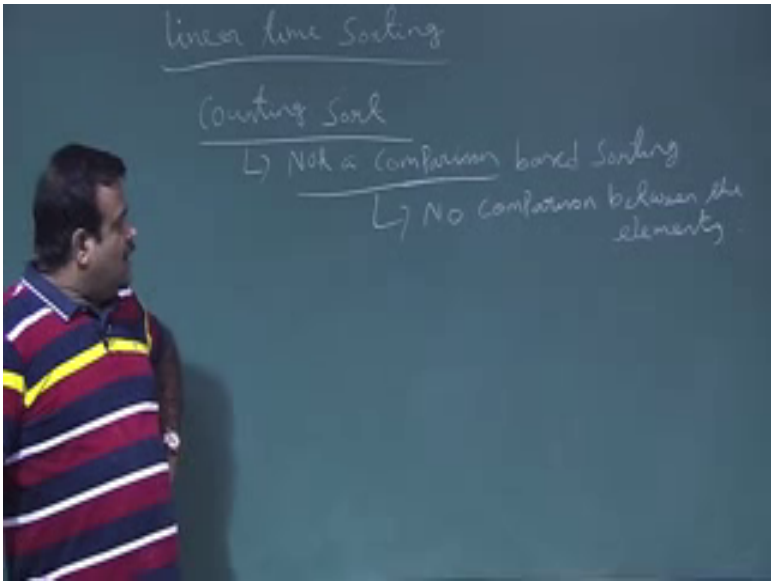
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 16
Linear Time Sorting

So, so far we have seen the comparison based sorting algorithm. And we have seen that, we have proved that by the help of decision tree that if you are using comparison based sorting algorithm we cannot go, we cannot go faster than the n log n. So, now, we talk about linear time sorting algorithm where we will not do comparison between the elements, but we will see the value of the element based on that we will do the we will do the sorting so this is.

So, we will discuss 3 such sorting algorithm one is counting sort or addict sort bucket sort. So, let us start with the counting sort, this is.

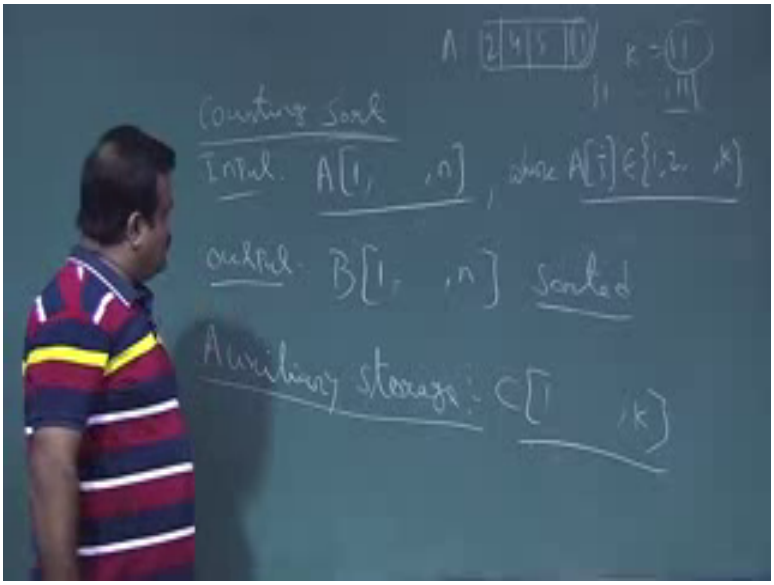
(Refer Slide Time: 01:04)



So, this is on example of linear time sorting algorithm. So, but this is not a not a comparison based sort, comparison based sorting so; that means, that means, no comparison between the element, no comparison between the element. So, we are not doing any comparison between the elements.

So, that is the basically, so that is how you could reduce it to the linear time. So, this is, so for this we need to have some assumption like, so suppose this is our number.

(Refer Slide Time: 02:11)



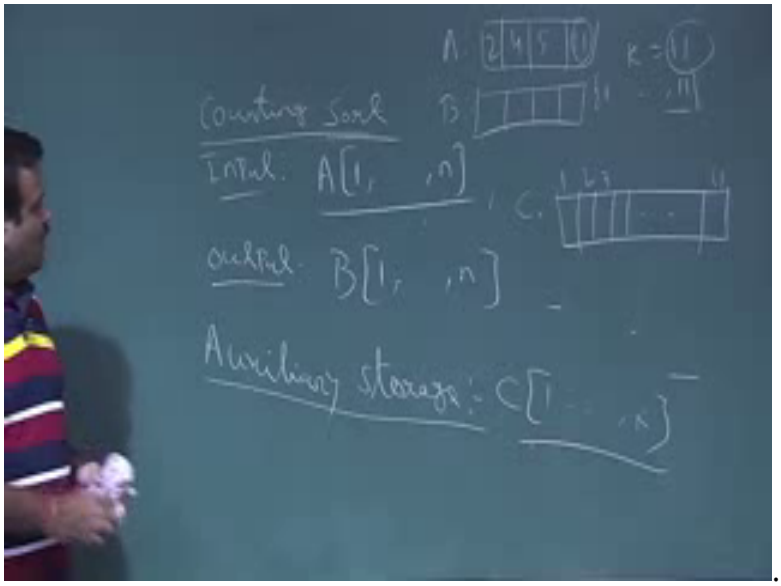
So, this is the input, what are the input? Input as say A array of numbers. And here we are assuming the value. So, we have to concern about the value of this number. So, we have to bound this numbers are coming from this range.

So, we have to fix the range of this number. So, that is also part of the input. So, that is basically where A i is coming from k. So, k is the maximum value we are allowing this number to take. So, this is also part of the input so; that means, we are fixing the range of the input. So, that is one of the, one of the criteria of this sorting algorithm. So, we have to, we have to mention the range of this numbers, otherwise we cannot sort this. Because basically based on that we need to take the auxiliary array.

So, the output with, so this is the input. Output will be in a B array which is basically sorted. What we need to take a, extra storage or extra or auxiliary storage here. So, this is the, these are basically you can say bucket auxiliary storage. So, this is denoted by C and this will be based on this range k. So, based on the value k we need to take this extra memory. So, this is these are not; obviously, in place sort. So, these auxiliary So, this storage also is a part of the input.

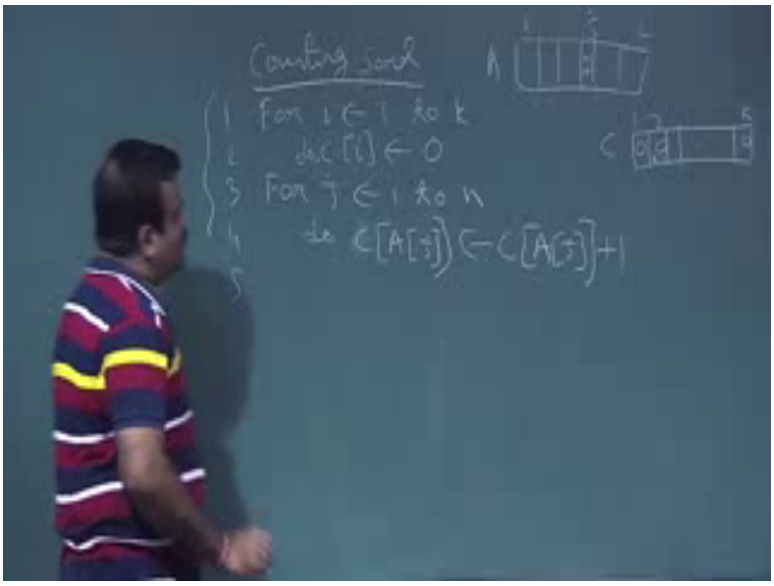
So, we need to take extra storage. So, so this is the range has to be fixed at the time of the input. So, once we fix the range then we can allocate the memory or the storage for that. So, this is the input output and the auxiliary storage. Now we have to write the pseudo code for this counting sort. So, so basically the basically, we are going to sort this number into the B array which is also of size n, but with the help of a extra another array which is the auxiliary array C array whose range is from one to k, and k is the maximum value of the input we are allowing the size, not the size the value input can take. Say for example, if we have a array, A array say 2, 4, 5 and then say 11. So, if this is our input. So, now, our k is basically 11. So, basically our numbers are coming from 1 to 11. So, k is the maximum value we are allowing as the input. So, we are having a range of the input. So, we are bounding the input to have for to, to come from this range.

(Refer Slide Time: 05:56)



So, that is one of the restriction, or one of the criteria for this so; that means, if k is 11 then. So, this we are going to sort in B array. So, B is also this 4 B array and, but we need to take a C array of size 11. So, so C array is of size 11, C array will be 1, 2, 3, 4, dot, dot, dot, 11 - 1, 2, 3, 4, 11. So, this is the auxiliary array we are going to, we need to take for this algorithm. So, now, let us write the pseudo code for this counting sort. So, this k is the range of the input. So, counting sort ok.

(Refer Slide Time: 06:48)

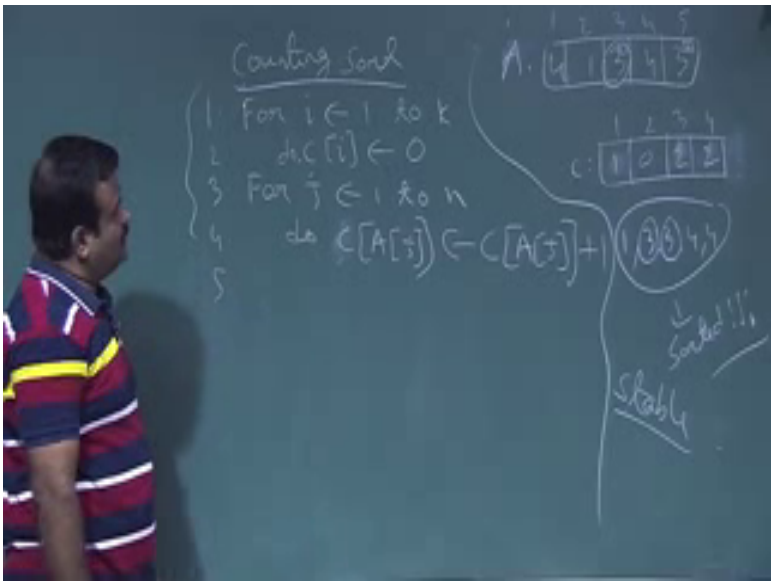


So, first of all we have to fill the C array. We have to initial initialize the C array by 0, C is just a frequency. Or we can say C are the bucket and we can fill the bucket like this. Anyway, so we will come to that. So, for i is equal to 1 to k. So, size of the C array is k. So, we fill it we initialize, we put the count as 0. So, this is the initialization and then, then we read the array. So, we have given the A array of size n. So, this is our A array and this is our C array of size 1 to k. So, initialize this all by 0 and then we read this A array and we accordingly if that we put the frequency in C array.

So, for j is equal to 1 to n, this is the A array. So, we read A j. So, this is particular j. So, we read A j. So, depending on the value of A j. So, if this is say 2 then we go to this and we put plus 1 like this. So, this is sort of frequency we are counting frequency of the that particular number to So, what we do? Now we do, we do this A, of A C of A j. C of A j we increase by 1; C of A j plus 1. So, this is just a frequency counting. So, the number of depending on the value of the element, this is the frequency count of that element.

So, after this what we are doing? So, up to this if we just count the frequency. Say suppose we have say input say like, this suppose we have a input. Say suppose, Our A array is 4 1, 3, 4, 3, ok.

(Refer Slide Time: 09:18)



Suppose this is our A array. So, this is 4 is the maximum size. So, C array is 1, 2, 3, 4. This is our C array. Now what we are doing this step we are initializing by 0, and we are just counting the frequency. So, we first read this is 1, 2, 3, 4, 5 there are 5 elements. So, we first 4.

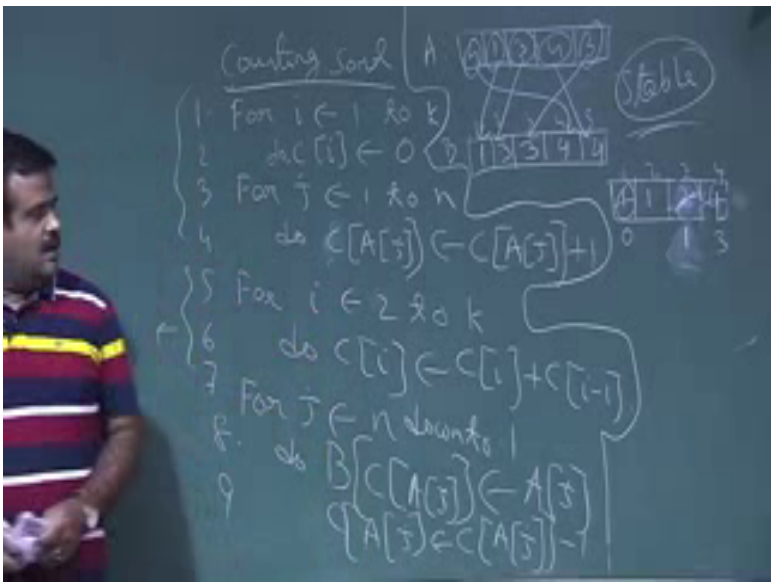
So, we put it plus 1. So, this is 1 then 1, 1, this is 3. 1 this is 4 again. So, this is plus 1 2 this is 3 this is 2. So, this is the execution after this. So now, now we know that there is only one 1. So, we can just pin the 1. So, if you put a bracket. We can just pin the 1, then we know there is no 2. So, we will just ignore that and we know there is 2, 3. So, we can pin this to 3 and we know there is 2, 4. So, we can pin this 2, 4 sort. So, sort it sort it. So, should we stop here, because we just, if we just read the array, read the C, C i mean frequency wise, that is it.

So, we can just get that sorted one, but should we stop that, stop here? No because, we want a extra property in this algorithm. What is that? That is call stability. Stable sorting algorithm or stability or stable sorting algorithm, we want the this sorting algorithm is stable. Stability means it should preserve the between the equal element it should preserve the ordering of the equal element like, like if we print just this. So, this 3 and this 3, we do not know which 3 come first in the input. We know this 3 came first than this 3.

So, we want in the output this 3 should come first than this 3. So, suppose there is a small tag over here. So, these are very important for satellite data. So, if we see the Google map some if we just capture the images. So, 2 image looks like same, but they are not exactly same. So, for those say suppose this 3 contents is 3 point we put a tag over here, 1 and we put a tag over here, 2 to indicate that this 3 come, this 3 came first than this 3. So, in the output also we want this 3 should appear first than this 3, ok.

So, that is called the stability. So, stability means, the input ordering should preserve in the output ordering, the between the equal elements. So, that is the that is called stability. So, we want that stability. That is why we have to execute few more, we have to write we have to do few more step for this counting sort in order to get this stability. Let us just complete this pseudo code. So, so for this what we do? We first have the cumulating frequency. Let us just erase this for the timing. So, we come back to this example again. So, let us just do the for i is equal to k we will just take the cumulating frequency 2 to k.

(Refer Slide Time: 13:21)



So, we do C i basically C i plus C of i minus 1. And then we fill the B bucket like this. For j n down to 1, for j n down to 1 what we do? We just fill this B of, B of So, basically A j we are going to fill in this B of C of A j. So, this A j we are going to fill in B of C of A j. And we decrease C of A j by 1 and C of A j is decreased by C of A j minus 1. So, C of A j is decreased by C of A j. So, this is 8, this is 9. C of A j is decreased by C of A j minus

1. So, this is the bucket filling. Let us take an example. So, we take the same earlier example.

Suppose we have this input A array there are 5 element 4, 1, 3, 4, 3. So, this is A array. So, this is A array now. So, we take a so, we are going to fill this in to the B array. So, B is also has to be 5 element 1, 2, 3, 4, 5. Now we have a C array. So, this maximum k is 4. So, C is 1, 2, 3, 4. So, this is B array and this is C array. So, now, we just execute this code. Now we fill this is the initialized step. So, we initialize this C by 0 and then we put the frequency.

So, after this there is only 1, 1, no 0 no 2, 2, 2. So, this is the, after this frequency of C. Now after that we have, we have to compute this C prime or cumulating frequency. So, we can say this is C prime. So, this is basically our new C. So, cumulating frequency means we just. So, it is starting from 2, up to 2, this plus this 1. And then, then we have this plus, this plus this, 2 plus 3, this plus this, so 5. So, this is the after this cumulating frequency we have this position of this array. So, this is the C array after the execution of that loop, 4, 5. Now 1, 1, 3, 5.

So, this is our C array after this execution of this loop. So, this is meaning of this is, up to this how many elements are there? One element. Up to this, how many elements are there? 3 element. Up to this how many elements are here? 5 elements because, that total 5 elements are there this is the sense. So, now, this is our C array. Now we go for the exact bucket filling by this loop. So, we start with n. So, we just look at this and we have to put this into some of the B array. So, how we can put this? So, put it by this way. If we go to the C of A j. So, A j is 4 A j sorry, A j is 3.

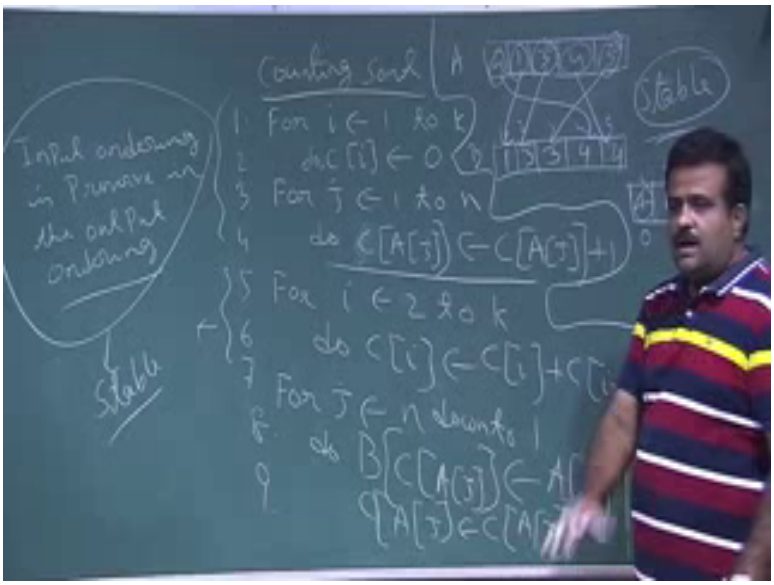
So, C of A j, so this is basically 1, 2, 3, 4. So, C of A j is basically 3. C i by A j is 3. So, we will put it into here. So, this 3 we put it into here and we decrease this by 2. I mean this C of A j is decrease by C of A j minus 1. So, this now this value is 2. So, what is the meaning of this? Meaning of this is if again we see a 3, and that will going to put in this position and that will assure the stability. Is this clear? If again we see a 3 we are going to put it here in the place of 2 in the here.

So, that will give us the stability, and that we are doing a n down to 1. Anyway let us complete this. So, we reduce this by now it is now this value is 2. This is the C array. Now this is our next element, this of this loop j loop n down to 1. So, now, it is 4. So, we

go to C 4, C 4 is 5. So, we put it here this 4, and we decrease by this one. So, what is the meaning of that meaning of that is if again we see a 4 we are going to put it in this place. So, that will give us the stability. So, now, this is basically 4, now we are here 3.

So, we go to here now it is telling us we put this 3 in here. And we reduce this by 1 so; that means, again if we see a 3 we are going to put it here. So, now, come here so C 1, C 1 is basically 1 and we put it here, and we reduce this by 0. So, there is no more 1 now come here. So, this is C 4. So, we go to the C 4 it is now 4. So, we are going to put it here this 4. And this is reduced right now 3. So, done sort it not only sorted this preserve the stability also. So, this is stable, stable in the sense that.

(Refer Slide Time: 20:56)



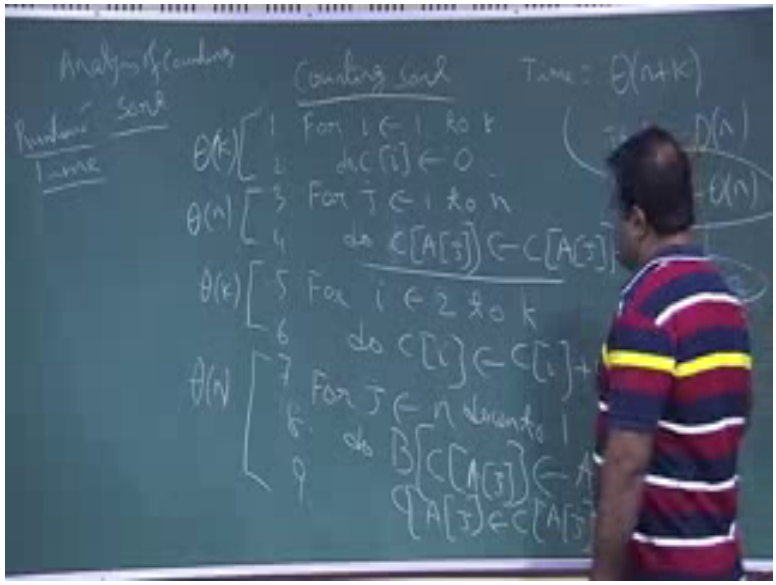
Stable means, we preserve the, it is preserved input ordering, is preserving in the output ordering. If this happen then it is called stable sorting algorithm. So, which is the comparison based sort is stable? Quick sort? Think about it. Now if you hear this stable because this 3 and this 3 are same. These are 2 equal element, but this 3 is coming first then this 3 if you put a little tag over here 0.1, 0.2 So, this 0.1 is coming before then 0.2. Even this 4 are equal, but this 4 is coming before this 4.

So, the input ordering is preserved between the equal element. So, that is the property we want in our sorting algorithm. And to have this property we need to execute this extra step. Otherwise we could stop at this place. This is this up to the frequency this kind of bucketing I mean we have the buckets we are filling the buckets and then we have print

the frequency of that numbers, but we need this extra property to have achieve this extra property which is stability we have to execute this, and this is a stable sorting algorithm.

So, now we have we want to, we want to have the time complexity of this. So, let us talk about, we want to analysis this code. So, this is the code and now we want to have the run time of this code, analysis of this code.

(Refer Slide Time: 22:58)



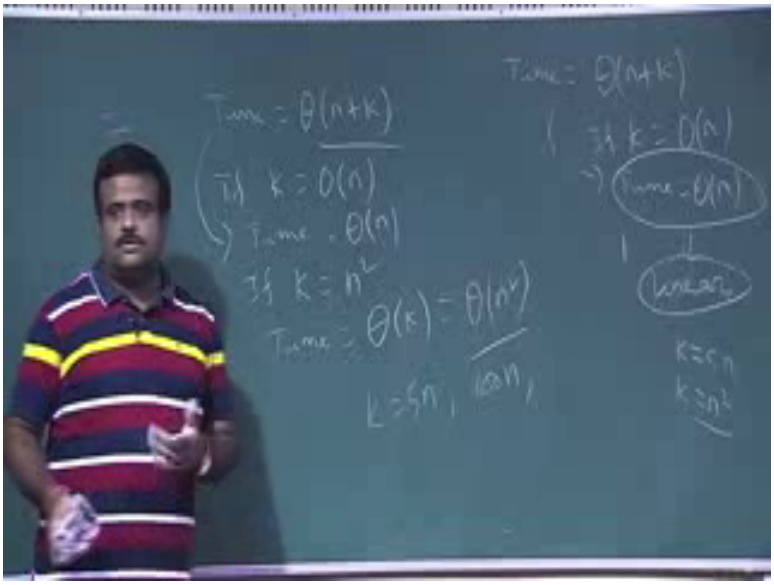
So, analysis of counting sort. Basically we do the run time analysis. What is the time complexity of this counting sort run time? So, let us look at the code. So, so here there are few for loops.

So, this is basically a for loop. So, this will take the order of k this the loop of size k. And here we are initializing this so, we are in asymptotic notation this initialization will take the constant effort. So, asymptotic sense So, that is why it is theta of k and this will give us theta of n the filling of the that frequency, and this will again give us the theta of k and this exact bucket filling will give us theta of n so; that means. So, what is the time complexity then? So, time is basically, adding if we add this to it is basically, theta of n plus k ok.

So, now what is k? K is the range, range of the elements. We are fixing the if we fix some range say if k is order of n then the time is basically theta of n, if k is this. So, it is

linear. So, it is linear provided the range is also bounded by n bounded by sorry, bounded by the number of inputs. So, if there are say n input.

(Refer Slide Time: 25:10)



And now we are bounding by say k is say, some c n or some constant n or something.

But if the k say n square then this is the dominating term if k is n square then the time complexity is basically what? So, so the time is basically order of n plus k. Now depending on which is dominating. If k is now the case one if k is big o of n, then the time is linear. Otherwise, if k itself is equality, k is say n square, or n log n. Then we have work then the time will be dominating by order of k it is basically, order of n square. Which is basically worse than the comparison based sort, ok.

So, that is the, that is the condition this to be a linear time sorting algorithm if the k is order of n. Other than that it is not possible. So, like a say 5 n 100 n something like that some constant into n then we can have this sorting algorithm to be linear. Otherwise it is a, depending on the value of k it will be either order of k or order of n.

So, next class we will talk about radix sort and the bucket sort, that is those 2, are also another sorting algorithm, another linear time sorting algorithm.

Thank you.