NPTEL
NPTEL ONLINE CERTIFICATION COURSE

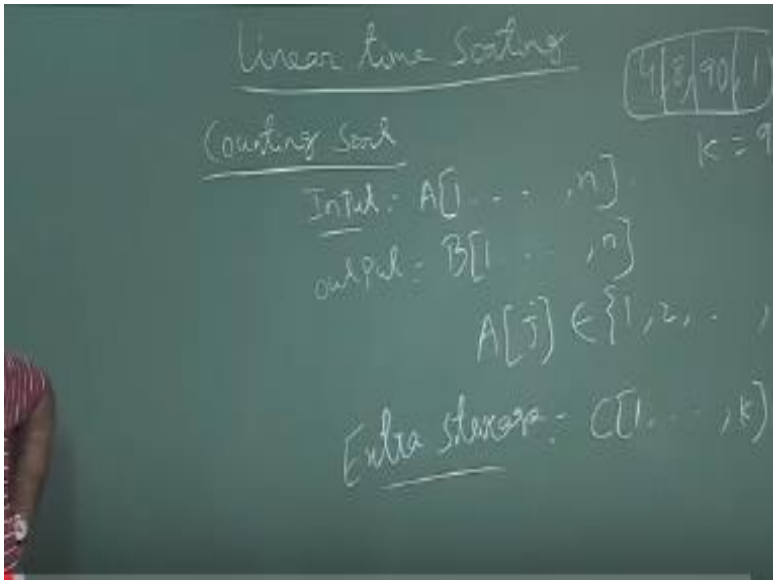
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 07: Linear Time Sorting

Okay, so now we will discuss linear time sorting algorithm which is basically bucket sort and it is linear, so it should not be comparison the sort, so we will not compare between the elements, we will see the value of the element and we will put into the bucket. So here we discuss two sorting or two bucket sort algorithm, one is counting sort and then second one is radic sort.

(Refer Slide Time: 00:43)



Let us start with the counting sort. Okay, so in the counting sort the input is, we have a array of size n which we need to sort and here, and the output will be another array of size A which will

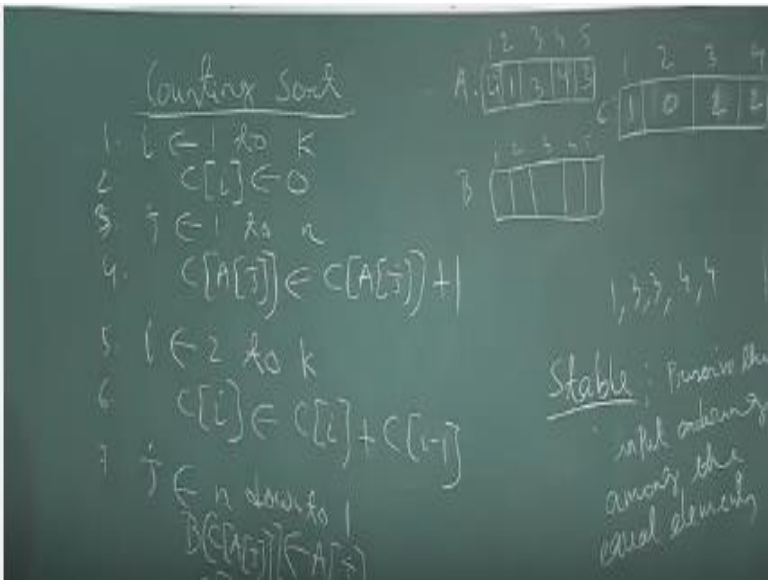
be, we have to sort in this array. And here we have another input which is K, K is the maximum hallow we are allowing for the element.

So that means we are assuming all the elements A[j] they are coming from 1-K. So K is the maximum allowed value for this element. So suppose we have a array like this 4, 8, 90, 1 so this is our array of four element which is our A, now K is 90. So that means every element are coming from 1 to K, so that is K.

So we need a extra storage for this algorithm which is the C array which is basically bucket, so the extra storage. So this is the auxiliary array we need to take, so this is C array we call C is from 1 –K this is now basically buckets we can say. There are K mean buckets depending on the maximum hallow of the element okay.

So now this is over, this is the input and we need to sort this A array into B array, but we need to take help of this extra memory or extra storage auxiliary storage okay. So let us write the code for counting sort.

(Refer Slide Time: 03:11)



So we have basically A array size n, this is the input and we want to sort it into B array which is also size n. And we have to have auxiliary array which is basically a bucket of size K okay. So this is, let us write the silver coat for counting sort okay. So we will first fill the bucket, so for that we need to just put the count of the in the bucket is 0 is 0-K, C[i] is basically 0, and then we need to fill the bucket.

So we look at the elements if this is four we will put, we will throw into the four that means we will increase the counter of the bucket number four to indicate that there is a four. So what we do here, we will just, we have a j loop 1-n so A[j], so C[A[j]] is basically C[A[j]]+1 so this is the, you are counting the, we are just increasing the counter of the bucket.

So suppose we can take an example, that will be easier to, suppose this is our A array we have five element, so 4, 1, 3, 4, 3 so A[1], A[2], A[3], A[4], A[5] and we have to store this into the B array which is also 1, 2, 3, 4, 5 and we have this bucket so we have maximum size is 4 so we have a bucket of size 4 1, 2, 3, 4 so this is our C array now C array is initialized by 0 now in this bucket filling what we are doing we are seeing it 4 we throwing into 4 so this is 1 now again 1 so this is 1 this is remain so this 3 , is 1 now we see another foot this is 1+ 1 this is 2 and we see a 3 this is 2 that is it.

So this is the C array after this okay so now what we are doing so now we can stop if we just a simply bucket sort because what we can do if we have this bucket so what we can do we reward the bucket we see there is all the 1, 1, we can print 1 there is no 2 no need to print 2 there is 3, 3 there is two 4 so we can 4,4 sort it so why we need to have more code in this also because we need to have another property which is called stability.

Stable sorting algorithm we need to because if among the equal and stability means the it should pacer the input ordering among the equal element so here there are two four this 4 and this 4 so here we have no way to judge which 4 is coming fast, so stability means we want this 4 should come first than this 4 so if that is that property is there then we call what sort algorithm is stable sorting algorithm this we required for settled data for satellite data so this 4 and this 4 are looks are same but there may be some little difference.

So we can put tack over here to indicate that difference so now after sorting we want this 4 which is coming in the before this 4 in the input order that should come first in the output but there is I if you just execute this there is no way to see whether this is the 4 over here so, so for that stability purpose we need to have little more coding here so that let us do so this is for i=2 to k this is cumulating frequency we can say, so we will just fill up this C[i] is basically C[i] + C[i-1] I will explain and then we will fill the B array.

So for j is n down to 1 what we do we do B of so we want to fill A[j], A[j] in B array so what we fill A[j] B[C[A[j]]] this A[j] we will put here B[c[A[j]]] and then we will decrease this C[A[J]]/1 counter C[A[j]]-1okay so this is pseudo code so we can just execute this cord so now this is our C up to this now we want to have a cumuli to see so this is our basically so we want to execute this, so this will be remain 1 and this will be just this + this so 1 and this will be this+ this three and this will be this + this 5 so now this is the condition of C array 1, 2, 3, 4 this is the condition of C array now we need to fill up this A into the B array so this is our B array sorry okay we can have here 1, 2,3,4,5 this is our B array now we will execute this j down from her so this is our A[j], A[j] is 3 now we want to put 3 into this B array this is our B array.

So for that what we will do we look at into C[3] , C[3] is basically 3 so we go to B3 and we will put t here and we will decease these by 1, 2 this is by these steps, so what is the meaning of that? Meaning of that is if we again look at if we get a c that we are going to put in2 that is the indication in order to get the stability because we are putting in the down way. So next we have a 4 this is the loop we are executing.

Next we have a 4 so we will go to c4, c4 is 5 so we will go to 5 and we will put it here this 4 and will decrease this by 1, 4. So what is the meaning of this? Meaning of this is if we again see a 4 then that we are going to put it here, so that will increase of the stability so again we see as 3 so you go to this now it is 2 so we will put it here, now we see a 1 so we will go to here so it is put it here.

Now we see a 4 so we will go to here now it is telling us to put it here 4, so stable. So for that we need to have this code to get the stability because this 4 is coming before this 4, okay. So this is the stable sorting algorithm, okay. Now what is the time complexity for this? Time complexity is so there are basically few loops so this loop is order of k and this loop is order of n and this loop is again order of k and this loop is basically order of n.

So total time complexity is O (n + k) okay. Now we want this to be linear in n so this can be only linear if the k is O(n) of n then the time is basically O(n) if k is O (n) suppose we are dealing with maximum size is say 100n, 200n some d x n then it is O(n), okay. So next so this is linear time sorting algorithm but provided this size is O(n) otherwise k will be dominating, so next we will discuss another bucker sort algorithm which is called radix sort which is also linear time sorting algorithm.

(Refer Slide Time: 12:56)

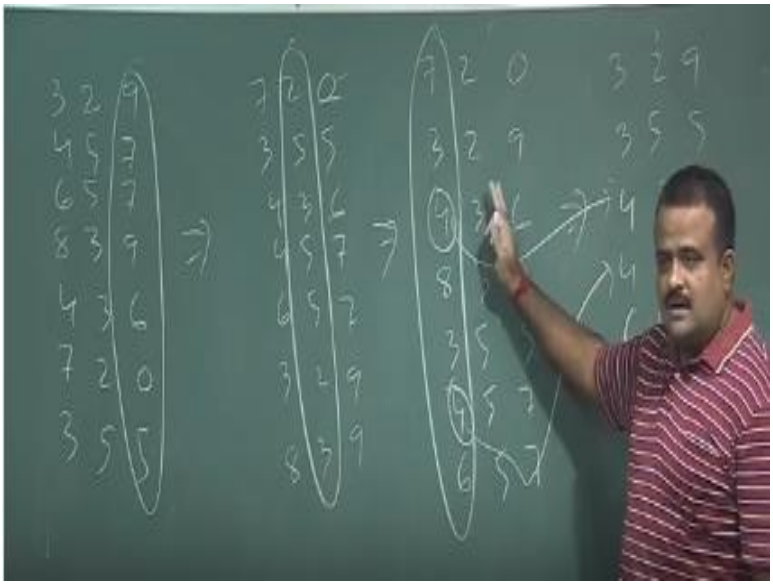
So radix sort so it is basically a digit by digit sort and we start with least significant digit to most significant digit from least significant digit to most significant digit and while we are sorting the digit they seem into most if while we are sorting each of these digit will use a oxalic sorting

algorithm which is a stable sorting algorithm we use a stable sorting algorithm then we can use just now we have seen counting sort.

May be we can use counting sort while we are applying on the digit, okay. So we will take an example and we will see how we can sort is so let us take an example, suppose we have a number like this 3, 2, 9, 4, 5, 7, 6, 5, 7, 8, 3, 9, 4, 3, 6, 7, 2, 0, 3, 5, 5, suppose this is our input we have some numbers n numbers each number is 3 digit basically these are in decimal, so now we this is the least significant digit.

This the most significant this is the second least significant this is the most significant so this is in a digit sort we will sort by digit by digit so we sort this numbers based on digit, and here we will use a stable sorting algorithm we can sue the counting sort.

(Refer Slide Time: 15:17)



So if you do that then, we, we will use, we will sort this number based on this digit and we will sue stable sort 0 is the minimum so 7, 2, 0 will come first and then 5 is there so 3, 5, 5 and we have 6 here so 4, 3, 6 and we have 2,7, here but since we are using the stable sorting algorithm,

so this 7 will come first 4, 5,7, 6, 5, 7 and we have two 9 here but this 9 is coming first 3, 2, 9 and we have 8, 3, 9.

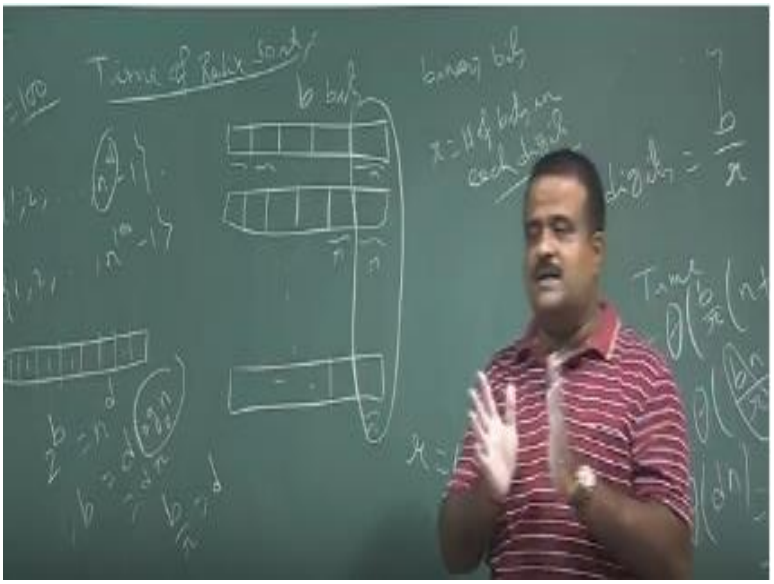
So this after sorting in based on the let see now in sorted in based on the next significant digit this way so 2 is the minimum so but there are two 2 so sort table sorting this will come first then this 9 then there is only one 3, so 4, 3, 6 and there is how many 5 three 5's so among this, this is the first so 3, 5, 5 4, 5, 7, 6, 5, 7 okay. So how many numbers 1, 2, 3, 4, 5, 6, 7 1, 2, 3, 4, 5, 6, one here missing what is that, we have 3 two 3 oh we have two 3, so we need one three okay, this 3 will come then 8, 3, 9 and then we have three 5, so 3, 5, 5 and 4, 5, 7 and 6, 5, 7 so 1, 2, 3, 4,5, 6, 7, okay.

So this is after, now we sort based on this digit, if the last step now here we have three 3 so this 3 come first so 3, 2, 9 then this 3, 3, 5, 5 and anymore 3 no, so we have already one 4, so 4, 3, 6 and then we have no we have two 4 sorry, so 4, 5, 7 and here one 6, 6, 5, 7 and we have 7, 6, 4, 4, 2, 3, 6 yeah, so we have 6 then 7, 7, 2, 0 and we have 8, 3, 9, 1, 2, 3, 4, 5, 6, 7 sorted. So this is the execution of the radix sort so we will do this is the sorted array, if you just look at this array we will look at this number this as sorted so what is the correctness of the code.

The correctness is we can prove the correctness 5 induction that means suppose we are assuming that up to this everything is okay, up to i^{th} level I mean as one level. Now we are going to short this and we have to ensure that this is correct. Now if there are two different digit, if the all the digit are distinct then there is no issue this is going to sorted. Now if the two digit are same like we have two 4s, this 4 and this 4 now we want to ensure that this 4, this 4 is coming here and this 4 is coming here.

Because this is this part is correct up to this, so this, this is less than this, so if it is less than this so this 4 will come before this and here also we are using sorting, stable sorting algorithm, so this part will come first then this. So this is the correctness we can see this by method of induction, okay. so now we are going to analysis this code the time complexity of this code.

(Refer Slide Time: 19:40)



Time for radix sort, okay so for that so we want to see whether this is a linear time sorting algorithm or not, so for that analysis we assume we are dealing with the binary numbers so we have a numbers, elements we are converting into binary.

So each elements here are l bit binary number so this is the first element second element we have a array of n element so each element are in bit binary so these are all binary bits so you have given number these are the number so you convert into binary and we take the maximum size as l bit okay so now we defined our digit by r bit.

So we choose r is the number of digit number of bits in is digit so that is our r so we choose r as the number of bit is digit so what we are doing we are diving this into the digit this is these are all r bits similarly for each of this so these are r bits so each of this is a digit so this is r bit, r bit like this so this is the less significant digit of the numbers of the elements now we are sorting these first using the counting sort and then your sorting again this is a counting like this we are going okay.

So now how many digit at their number of digit so what is the number of digit number of digit is basically b/r yeah if this is L we can take this is other b, b is the number of bits so this is basically b/ r so we have to tell b bits we divide this bits into r digit so it is b/r is the number of digit and you are sorting is digit using counting sorts so how much time we are spending there for counting sort.

We know it is Θ(n + k) so k is the maximum we are align for this digit now each digit is r bit what is the maximum halo this is r bit when each or one so this is basically put the r so the total time complexity for addict sort is b/ r x n + 2^r this is the time complexity for addition when the question is how it is coming to be linear order of in okay so for that we can choose r to be logn if we choose r logn then this is a order of n so this will become.

Bn/r where r is order of logn okay but now you want these to be linear how we can get this linear now we choose this we choose our number form this side 12 up to n^{d-1} where d is the constant they could be say 100 0r T(n) something like that so our numbers are basically coming from 1 to n^{100-1} something like that so these are the this is our key the numbers we are allowing the each element are coming from this now this is the maximum size now our elements are.

b bits element by if it is b bits each so what is the maximum size it is 2^b basically but we are allowing into the per d so 2b – 1 so n^d = 2d okay so from that can you get a release on safe so if we take the log it is basically b is basically d logn, logn is basically r so dr so basically d/ r is = d so this b/r is basically d so this is basically order of dn now d is just a constant d is so obtain 10 100 or something so this is basically.

Linear time so this is linear time sorting algorithm so this is basically digit by digit sort it is starting from less significant digit to the most significant digit and now the original read sort as from most significant police significant so we have a array so we can just sort most if the all numbers ay distinct then that will your perfectly because if the all the most significant digit distinct fine so that will decide the order of the element.

Most significant digit but the problem is if some digits are same then we have a problem we cannot just because then among that again we have to look at the second level where this most significant bits are same so again we have to care of this part to sort them so again among that if the they are also same the next level so but we need to have some storage for that so that is a have so to overcome that this we will sorted for least to most and correctness we are same patterns we can do by just we can see by the induction we assume everything is okay up to jet level j – 1 level in the jet level we have seen it is direct, okay, thank you.