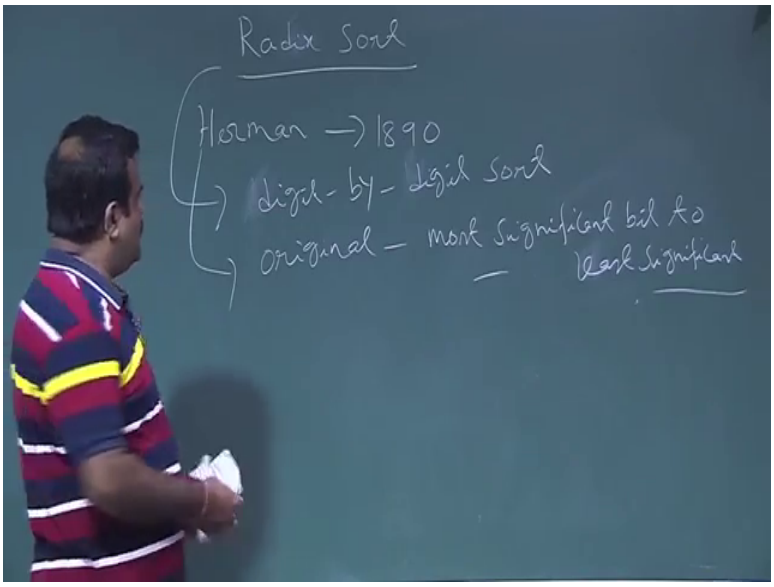
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 17
Radix Sort

So we talk about another linear time sorting algorithm which is radix sort. It is the idea of radix sort was invented by Herman in 1990, very old sorting algorithm.

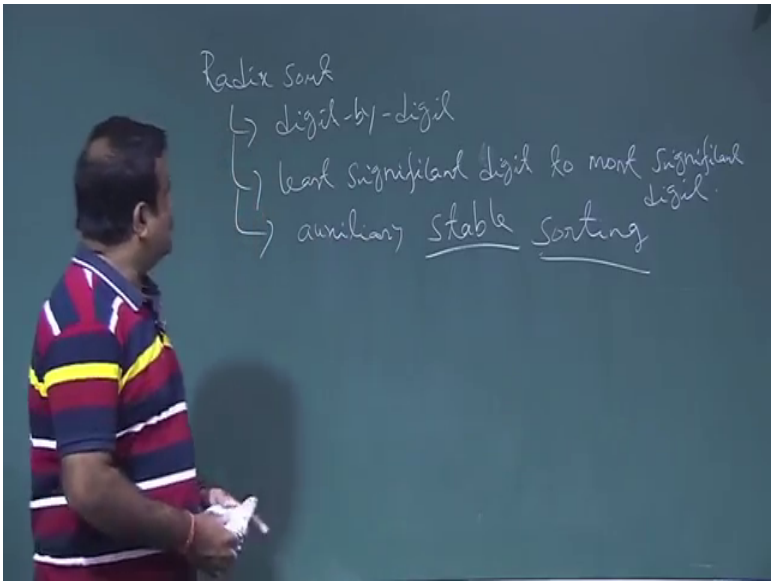
(Refer Slide Time: 00:32)



And it was used for US sensor. So, they use a cord for this sorting algorithm yes, sensor and the idea of this radix sort is basically digit by digit sort digit by digit sort. And the original version of the Herman was from most significant digit to the least significant digit, but that idea was having some drawback.

So, the original version of the radix sort is most significant bit to least significant bit. But that is having some drawback that idea is not good. So, then the then it is modified and the current radix sort is basically, least significant to the most significant.

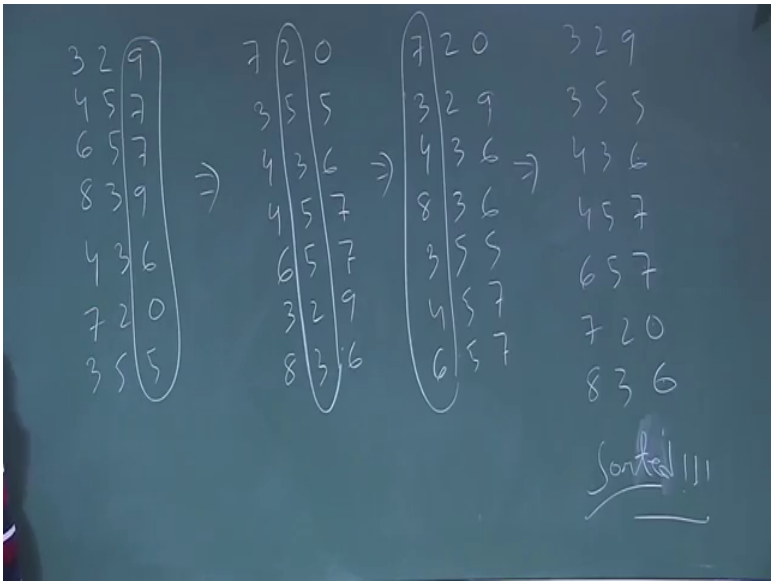
(Refer Slide Time: 01:59)



So, we sort this number digit by digit, by the least significant bit to the radix sort. So, digit by digit sort, least significant digit to least significant digit or bit to digit to. So, least significant digits first then the we come most significant, least significant digit to most significant digit ok.

And whether we are sorting this each digit will use a help of auxiliary sorting algorithm these are stable sorting algorithm. So, we So, when we sort this each digit I mean number based on that particular digit. So, we will take help of a auxiliary stable sorting algorithm. Maybe we can use the counting sort, stable sorting algorithm. So, let us take an example then it will be more clear. So, suppose we have this number 3 2 9 4 5 7 5 6 5 7.

(Refer Slide Time: 03:32)



These are the input 8 2 9, these are the 3 3 digit number and these are decimal digit each of this is the decimal number, 3 8 3 9 4 3 6 7 2 0 3 5 5. So, how many numbers 1 2 3 4 5 6 7, 1 2 3 4 5 6 7. So, there are these are the input we need to sort this numbers. So, this is basically 3 3 digit 3 bits. And each digit are digit each digit are a decimal digit. Now we this is a digit by digit, digit sorting algorithm and we start with the least significant digit.

So, this is the least significant digit. So, we sort this number based on this value, and we will use a stable sorting algorithm, say we are using a we are taking help of a counting sort, which is a stable sorting algorithm we know. Which is we are taking a linear time stable sorting algorithm. So, we cannot use the comparison based sort, it is not a linear. So, we take help of a stable sorting algorithm, and we sort this number based on this digit. So, 0 will come first. So, 7 2 0, then what is the next 5, 3 5 5, then we have 6, 4 3 6 and then we have 7. But we have 2 7 and this is a stable sorting algorithm.

So, this 7 is appearing first in the input. So, this should come first. So, basically we should have 4 5 7 and then 6 5 7 because of stability. We are using a stable sorting algorithm. And then the next number is 9 again we are using the stable stability. So, 3 2 9 will come first then 8 3 6. So, this is the sorting after first 7 2 0, 3 5 5 4 3 6, 4 5 7 6. So, this is after sorting based on the least significant digit. Now this is the digit by digit sort here. So, this is the next digit.

And we sort again this number based on this digit. And again we are using a stable sorting algorithm where we can use the counting sort. So, if you do that then just look at the volume that is it. So, the 2 is the minimum. So, there are 2 2s. So, it should preserve the stability. So, these 2 should come first. So, 7 2 0 then we have these 2, 3 2 9 and then we have 3, but they are 2 3s. So, 4 3 6 should come first then the 3 6 8, then we have no 4 we have 5 we have 3 5s, 1 2 3 and among this is first came first stability.

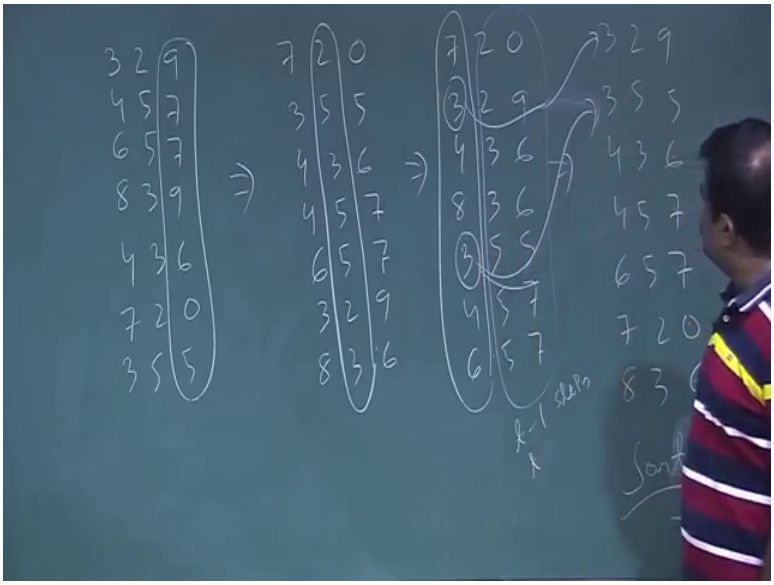
So, then 4 5 7 and 6 5 7. 1 2 3 4 5 6 7. So, now, finally, most significant digit and again here we are using the help of a counting sort. Or any stable sorting algorithm. So, if you do then this is the minimum 3 and here also you need to preserve the ordering. So, this 3 should come first 3 2 9, then this 3 3 5 5 and then 4 there are 2 fours this 4 should come 4 3 6 and 4 5 7. And there are 6 6 5 7, and there are 7 7 2 0 and 8 3 6, 1 2 3 4 5 6 7 sorted, sorted. So, this is basically the radix sort.

So, it is a digit by it is a digit by digit sort, and it start from that least significant digit to the most significant digit. So, here digit are by decimal digit 0 to 9 this is the least significant digit. So, we sort this number based on this and then we sort this number based on the next digit and then we sort this number based on the next which is the most significant digit. So, this is the shortest, now what is the correctness of this? Why this is working? To know the correctness we have to, we have to think of in terms of the induction method varying induction way.

Now suppose We are assuming this algorithm is correct up to t th digit, or t minus 1 digit. So, suppose t is say 2. So, suppose this algorithm is correct up to this. Now we need to prove that this algorithm is correct after this t f digit to sort. Now we are assuming this algorithm is correct up to this. Now if there are 2 numbers different like here say there are 3 and 4 if they are different then 3 will come first and 4 there is no issue, only issue will come if there are common number this 3 and this 3.

So, among this 2 3 we know the algo is correct up to this, this is the assumption we made this is the method of induction. We are proving we know the algo is correct up to t minus 1 steps.

(Refer Slide Time: 10:02)

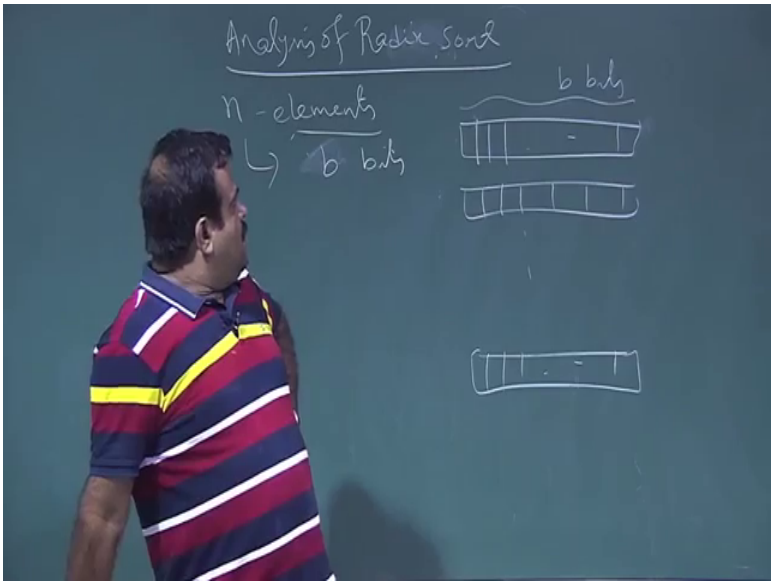


This is our assumption and we are going to show this is this correct after t th step. Now if the algo is correct up to t minus 1 steps. Then the this will appear before this because this value is less than this. Now once this will appear before this th en this is a stable sorting algorithm we are using. So, this 3 must appear first than this 3 sorry, I missed 3 yeah I sorry yeah this 3 must appear Before than this 3. So, this is the correctness.

So, we assume the our algo is correct up to t th step up to, say after second step which is now we are going to show it is correct after up to t minus 1 step now we are going to show it is correct after t th step. So, to do that So, we take. So, between the different element we do not have no issue, only problem is between the equal elements. So, these 3 2 are equal. So, now, we know this is correct up to this. So that means, this will come before than this ok.

So, since it is coming before than this, and we are using a stable sorting algorithm at the t th step. So, this must come before this, that is what is happening. So, this is correct. So, this is the correctness of this radix sort now we talk about the time, we have to analyze this radix sort.

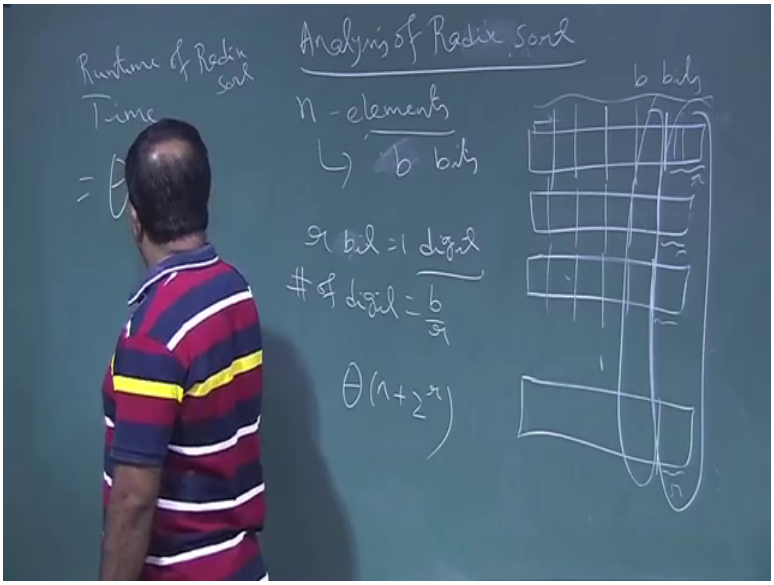
(Refer Slide Time: 11:58)



So, let us have a quick analysis of radix sort. So, analysis of radix sort and we will see that it is a linear time sorting algorithm. Sorry, radix sort. So, to analyze the radix sort suppose there are n, element suppose there are n inputs. And say each inputs are say b bits. So, each inputs are b bits. So, there are n numbers each of this are binary bits.

So, if we are given input we convert into binary, 0 on this. So, there are n numbers. So, there are n numbers. So, each number is of total b bit. So, we have given a number any number we convert in to binary, and the maximum size is b bits. Now we have to fix our digit now suppose we take our digit as say r bit. So, these are the numbers which are b bits, b bits number n numbers.

(Refer Slide Time: 13:15)

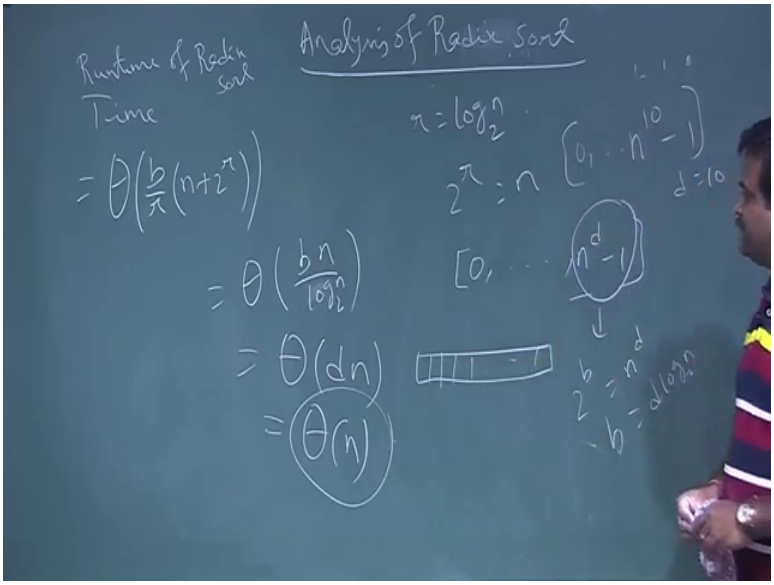


And now we fix our digit by r bit, r could be any value say 3. So, there are 3 bits we are using as a r. So, r is the r is the r bits 1 digit. So, this is our we are defining this. We are taking r bit as 1 digit, and we divide this into digits.

So, r bit r bit, r bit provided this b is multiple of r. So, these are our digit dot, dot, dot, dot like this. So, this is the r bit, this is the r bit this is least significant digit r bit. So, how many digit are there. So, number of So, each digits r bit. So, number of digit will be b by r, this is the number of digit. Now we are going to sort this digit by digit. So, this is the least significant digit then this is next significant digit like this. Now we are going to sort digit by digit. So, we sort this first this we sort this number based on this value, this is the least significant value of this. So, this is r bit.

So, what is the maximum value it can sort how much time it will take. So, if we are using counting sort. We know counting sort will take theta of n plus k, what is the k? K is the maximum range of that number. So, if it is r bit the range is basically 2 to the power r. So that means, if it is r bit each can take one value. So, it is order of 2 to the power. So, basically, it is basically theta of n plus 2 to the power r, this is the time complexity to sort each of this digit. We sort this number based on this digit and this is the time. So, total number of digits this.

(Refer Slide Time: 15:49)



So that means, time complexity for radix sort is run time of radix sort is basically theta of, how many digit b by r digit? And each digit will take this much time n plus 2 to the power r. So, this is the time complexity of radix sort. Where b is the total number of bits in binary and n is the number of element to be sorted, and r is the size of our digit we are defining that we are fixing some size for our digit. So, now, how it is coming to be linear. So, let us just talk about that. Now suppose we have to choose r to be log n. Suppose we fix our device size to be log n base 2. Then 2 to the power r is basically n. So, then this will be basically theta of d n by r, r is again log n base 2 ok.

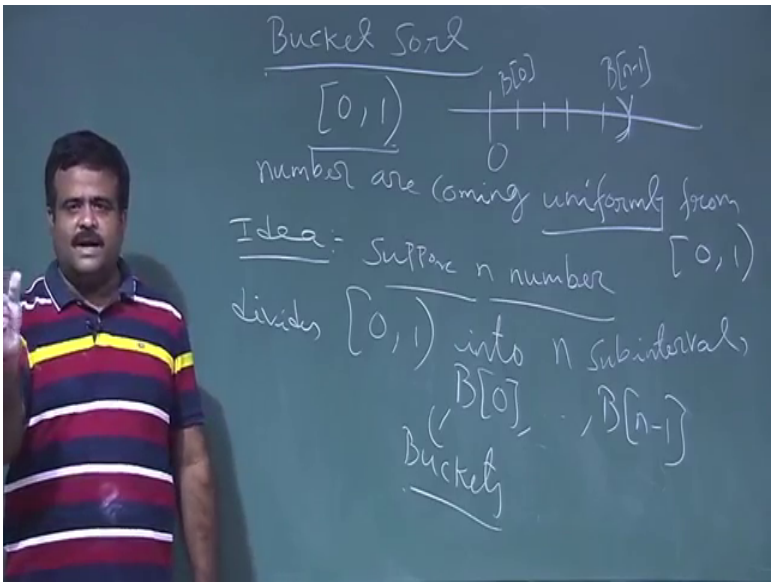
So now suppose, suppose we know. So, our we know our numbers are coming from, our numbers are coming from this range, 0 to n to the power d minus 1. Suppose our numbers are coming from this range. So, this is the maximum value. So, this is the maximum value our number can take. So, we have n such numbers each number can take this. Now numbers are basically b bits each number is b bits binary bits. So, what is the maximum value yeah it can take. So, 2 to the power b minus 1. So, basically 2 to the power b is basically n to the power d. So, this means what if you take the log, b is basically d log n base 2.

So, b by d, b by log n is basically d. So, this is basically theta of d of n, theta of d of n. What is d? D is the we have n numbers and we are assuming the numbers are coming from this range. 0 to n to the power d minus 1. So, that case it is basically theta of d n.

Now if d is constant just like if the numbers are coming say n to the power 100 minus 1, 0 to n to the power minus 1, then d is d is sorry n to the power d is 10. So, this is a constant. So, then this will be theta of n if d is a constant. So, this is a linear time algorithm provided, d is a constant. So, bucket sort is a linear time sorting algorithm, provided I mean if the number we choose are from this range ok.

So, next we will talk about another sorting algorithm, which is basically this is radix sort, which is now we talk about bucket sort. So, for bucket sort we are having some, we have we have some numbers, and here we need to assume the numbers are coming uniformly from the interval 0 1 bucket sort.

(Refer Slide Time: 19:10)



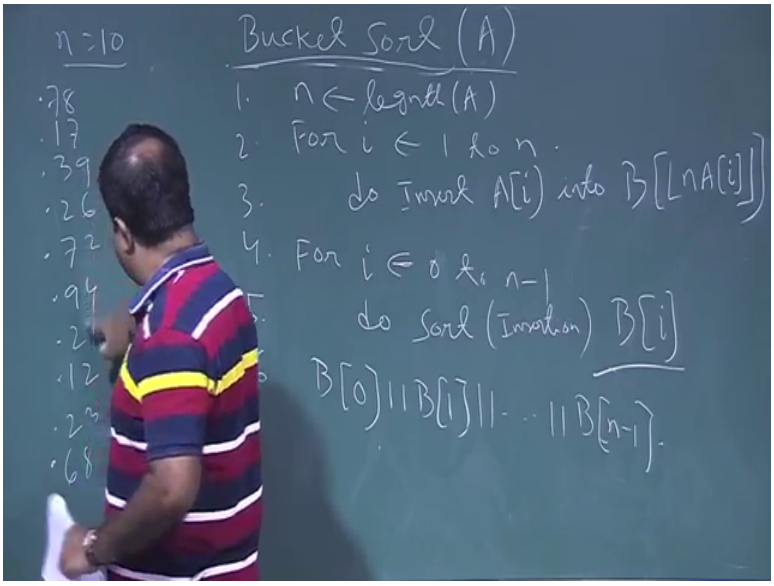
So, here we are assuming that the numbers are coming from the interval 0 and 1, this is open interval this is close interval. The elements are coming the numbers are coming uniformly from this uniformly means, we need to have this assumption in order to get the time complexity for this. So, what we do?

So, we basically the idea is to suppose there are n numbers, suppose n numbers. So, idea is to partition this into n sub intervals. We divide this into divide the interval into n sub intervals, intervals. So, numbers are n numbers are coming from this. Now we divide this into sub intervals, and each are called bucket. So, this is a b 0 b 1 dot, dot, dot b n minus 1. This is starting from 0. So, these are called buckets. So, basically we have we have n

buckets. B 0 to b n sub intervals they are denoting by p 0, b 1, b n minus 1 and these are basically bucket ok.

And then we have this number we put into the buckets, depending on the value of this, and once we fill the bucket, and we sort individual bucket and we report it, that is it. So, let us write the code. So, let us write the pseudo code for bucket sort.

(Refer Slide Time: 21:43)

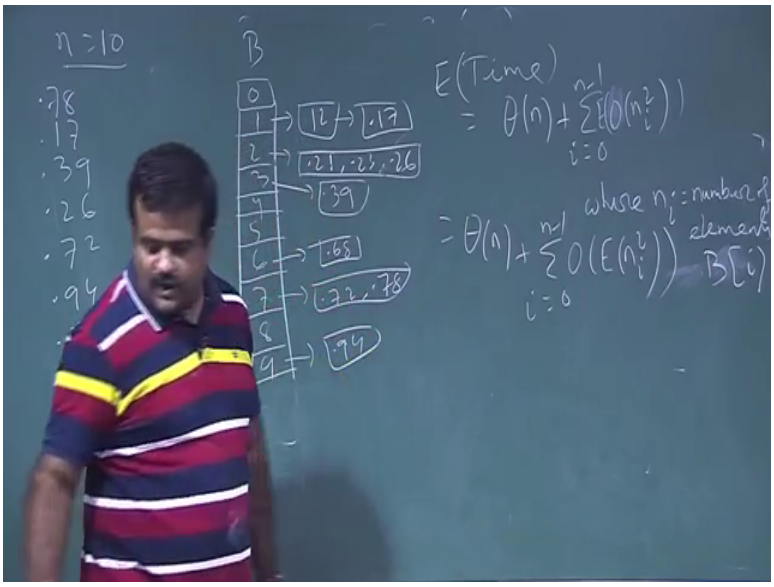


So, there are n numbers. So, which are giving in a array. So, n is the length of this array. So, what we do for i is equal to 1 to n, we fill the bucket, do insert. We insert a i into the corresponding bucket, b of n of a i are lower ceiling ok.

So, this is the bucket filling we will take an example, and then after that once we have the buckets of numbers, then we sort this. Each bucket for there are n buckets for i is equal to 1 to sorry, this will be 0 to n minus 1 yes. 0 to n minus 1 because bucket is starting from b 0 to b n. Do, we sort sorting means we use the insertion sort basically, insertion sorting algorithm we sort this b i, we sort each bucket. Each bucket contain some numbers we sort it. And then we concatenate just the bucket after sorting, we concatenate this b 0 then b 1 concatenated this b n minus 1, and this will give us a sorted one ok.

So, let us take an example, quick example suppose our n is 10 and the numbers are basically say suppose n is 10. And suppose numbers are like this 0.78, 0.17, 0.39, 0.26, 0.72.

(Refer Slide Time: 23:39)



So, we are assuming this numbers are coming from uniformly from this, uniformly from this interval 0 1, 0.94, 0.21, 0.21, 0.23 and then 0.68. How many numbers 1 2 3 4 5 6 7 8 9 10. So, we want to sort this into the buckets. So, here n is 10. So, we have buckets like 0 to 10.

So, these are the bucket basically. So, you have 0 bucket. So, 1 2 up to nine. So, 1 2 3 4 5 6 7 8 then 9. We divide into equal. So, now, we put into the bucket. So, one bucket means the 0.1. So, 0.1 is basically we have 0.21 and then we have 0.17, 0.21 and we have 0.17. And then we have 0.2 bucket is basically, we are putting into this like this. So but it is a basically a linear is 0.21 and then 0.23 and 0.26. And in number 3 we have only one element 0.39, these are the bucket and 6.

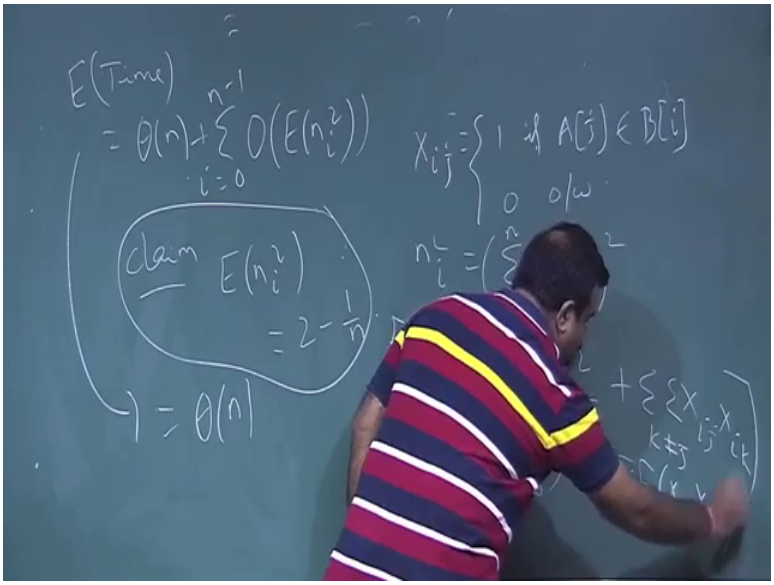
We have only one, 0.68 and 7 we have 2 elements 0.72, 0.78. And then we have 9 we have 0.94. So, this is the bucket filling then after that we sort this using the insertion sort individual buckets. So, these are the number falling in the bucket number 2, b 2 then we sort this numbers using the insertion sort. And then we just print the buckets like concatenate the bucket. So, this is basically bucket sort. Now what is the time complexity

for this bucket sort? Time is basically So, we are reading all the elements, and we are filling into the bucket So that will take the theta in time.

And then we are after bucket filling we are sorting the individual buckets using insertion sort. So, that will take the time depending on the number of element in the i th bucket. So, if the number of element in the i th bucket is n then this is basically summation of sorry, n i theta of n i square i is equal to 0 to n minus 1. Where n i is the number of elements fall in the in b i i th bucket. Because each bucket is sort by using the insertion sort. So, this is the time, now if we take the expected value of this, then this is basically expectation of this. So, this is basically theta of n plus summation of we can take big o here. So, big o of. So, summation of big o of expected value of n i square.

So, i is from 0 to n minus 1. Now this expected value of n i square. We can prove that this is to be So ok.

(Refer Slide Time: 27:31)

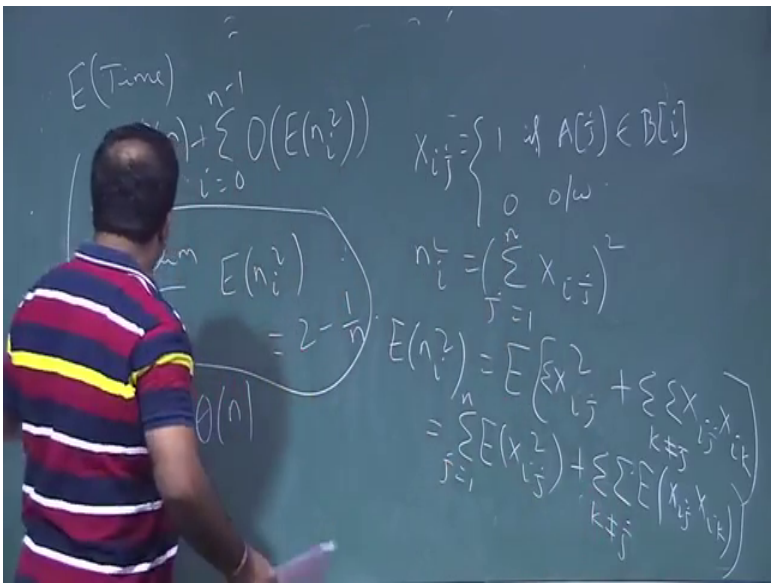


So, the time complexity is basically time accepted time complexity is basically, run time is theta of n plus summation of big o of expectation of n i square and this I is from 0 to n minus 1. Now we claim that this expectation of n i square we have to prove this, is basically, 1 2 minus 1 by n. If we can prove this then this is basically suppose, this is this yet to prove. Suppose we assume this is for all i then this will become what? Then this will become expected this will become linear.

So, how to prove this? Expected value of n i square is 2 minus this. So, to prove this we need to take help of indicator random variable. So, this we have to prove. So, what is that? Now we define x ij is 1 if, if a j falls in j i th bucket if a j falls in i th bucket, 0 otherwise. X ij is 1 if a j is false in because this ball this elements are chosen randomly from 0 1. And we have n buckets if the b 0 to b n minus 1. So, we define this. So, what is n i? N I is basically summation of x I. So, n i is the number of element in i th bucket. So, n i is Basically summation of x i and this i is j is varying from 1 to n. So, we take the expectation of n i th square. So, n i th square is basically this square and we take the expectation of this. So, this will basically the expected value of expectation is a linear function, we will take that x i square x ij square plus summation of double summation of x i k x k j x i ij I k, where k not equal to j. This is the form of x square now expectation is a linear function. We take the expectation inside. So, this is basically summation, this is summation.

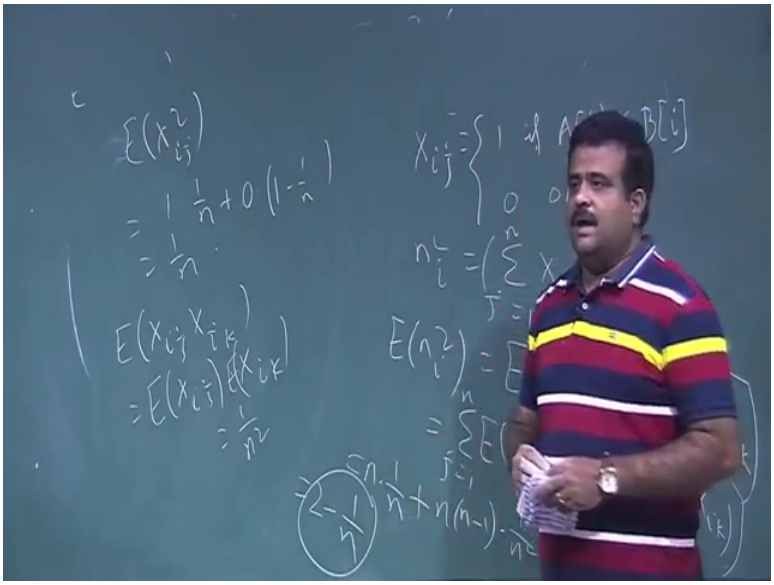
So, we take the expectation inside. So, expectation of x i x j square, and this j is from one to n plus this summation of expectation of this 2, and this is a independent.

(Refer Slide Time: 30:33)



So, this anyway let us write this, and this is k not equal to j. So, so if we write this, then this is basically now this term is basically expectation of x i square, is how much?

(Refer Slide Time: 30:51)



X i x j square is basically 1 into 1 by n plus 0 is to 1 minus 1 by n. So, this is basically 1 by n. And now this will give us expectation of x ij x i k, if we take the independence of this 2 into expectation of x i k. So, this will give us Basically 1 by n square.

So, if we plug this value over here, we are getting basically n into 1 by n plus, n into n minus 1 into 1 by n square. So, this will give us basically 2 minus 1 by n. That is what we are looking for 2 minus 1 by n. So, that is basically expectation of n i square. So, if we put this expectation there we are getting basically bucket sort is linear.

Thank you.