NPTEL
NPTEL ONLINE CERTIFICATION COURSE

Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 06: Decision Tree

Okay, so we talk about decision tree the problem is, the question is how fast we can sort.

(Refer Slide Time: 00:28)

So this is the question. So, so far we have seen some comparison to sort all the sorting algorithm we have seen are comparison list, comparison list. So that means we are comparing two elements and then we are taking the information by comparing the elements and we are taking some decision.

So those are comparison by sort. So we have so far the sorting algorithm we have discussed are basically.

(Refer Slide Time: 01:14)

Insertion sort, then we talk about merge sort, quick sort, and we already we have discussed comparison of the quick sort which is randomized quick sort and then we talked about heap sort. But all are basically comparison by sort.

That means we are comparing the elements and we are taking some decision, we will discuss the decision tree. So now what is the time complexity based case, average case and the washed case. So for insertion sort what is the based case? When the array is already sorted, so then we are just comparing the previous one.

So it is basically order of (a), and the average case we discuss that on an average maybe we have to come from half way, so it is also giving us order of (a^2) and the washed case when the array is that we got sorted it is order of (n^2). And merge sort, merge sort is all the cases it is (nlogn), because we do not really care whether this is sorted or you got sorted in the merge sort, what we

are doing, we are going into the middle and we are sorting this part and we are sorting this part recursively.

So go here at the conquer field, this is a divide and conquer method. So we do not really care how whether they are equal, whether they are sorted, we got sorted, we are going to the middle, we are sorting this, we are sorting this recursively, that is it. So that is why it is the recurrence is that, so we got this solution for the merge sort. And what is the quick sort, quick sort the based case is when the partition is pivot is the route pivot, so it is partitioning half, half n/2, n/2.

So then we got nlogn, so average case for quick sort or this will come under the randomized quick sort version and the washed case is, when it is, because our partition is first element we are choosing as a partition, so if you take the input as a sorted or river sorted then always partition will take the pivot as a minimum or maximum, every time in the subsequent step.

Even though if we choose the partition as other element say, middle element of the array n/2 element, so we can always construct the input where we put in the middle element the minimum one. So we put the, this is the array so for this array this is n/2, we put the minimum log here for this whole array, again we come to this sub array we put the minimum one among this here, so one can easily construct a input where every time the pivot will be the minimum one.

So then our this partition will be a bad partition and it will partition is do is to n/1. So that is the washed case, so that solution we know is n^2 and now to work on that problem that we should not know which position we are choosing as a pivot. So that one should not come with some input where our algorithms are coming badly.

So to overcome that problem we choose the randomized version of the quick sort, that means we choose the pivot randomly. So we will not do, we will not say that we are going to choose this element as a pivot, so that will be decided at the run time, so that will be depend on the random number we are getting.

So for that the based case is again nlogn and the expected it is also nlogn, because we, the

expected runtime of this and the washed case is always n^2 even for the randomized portion of the quick, because we may be it happen that every time our random number is so bad that it is choosing the minimum or maximum of that corresponding sub array.

I mean, so it will always give this, and the heap sort it is basically nlogn. Okay, so but we know that we are interested in the washed case analysis, because that is the most essential analysis one can think about. So this washed case we have nlogn the based one. So now question is whether we can do something better than n log n or not, so in the worst case the question is.

(Refer Slide Time: 06:24)

In the worst case can we do better than n log n in the worst case, so that is the question so to answer this question we have talked about the decision tree, the answer is no, we cannot do better than n log n while we are using the compression best sort, that means we are comparing two element and we are taking the information by comparing two element and we are taking some decision.

So for comparison best sort answer is no, no if we are using any comparison based sort, comparison is sorting algorithm, okay so, so how we can justify this, we will take help of a

decision tree, so what is a decision tree, okay, decision tree we will take an example suppose we need to sort three element a_1, a_2, a_3, okay so to sort three element what we will do we compare fist a_1 and a_2, here we are assume the elements are distant, so we compare a_1 and a_2.

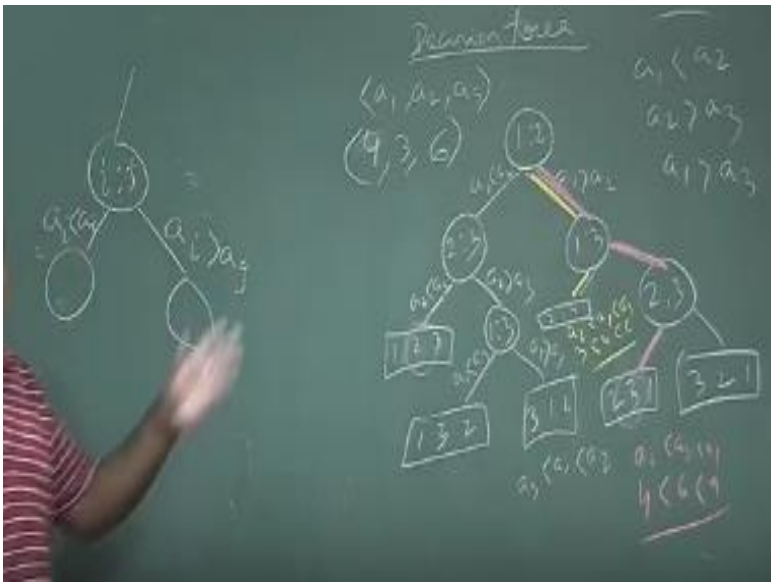
So if we compare two element there are two possibilities because we are taking the distant elements so there are two possibilities either a_1 is less than a_2 or a_1 is greater than a_2, if a_1 is lesser than a_2 then we will take this path and we will do the subsequence comparison before we reach to a decision, and if a_1 is greater than a_2 we will take this part and we will do the sub sequence comparison before we take the decision.

So here in this path again I compare a_2 and a_3, if I compare a_2 and a_3 again we have two way to go so these way if specifically a_2 is less than a_3 and this way a_2 is greater than a_3, now if we take this path then we can reach to decision and that decision is 1, 2, 3, that means a_1 is less than a_2 less than a_3 because this a_1 is less than a_2, here a_1, a_2 is less than a_3 so by transfer property of less than we can say 1 is less than a_2 less than a_3.

Now if we reach to this path then we need to do another comparison between a_1 and a_3, then again we have two path a_1 is less than a_3 and a_1 is greater than a_3, so if we reach to this path then we have a decision 1, 3, 2, that means a_1 is less than a_3 less than a_2, and if we reach to this path we have a decision 3, 1, 2 that means a_3 is less than a_1 less than a_2 because this path is basically telling us a_1 is less than a_2 and a_2 is less greater than a_3 and a_1 is greater than a_3 so a_3 must be the minimum than a_2, because from here yeah so that because a_1 is a_2, a_1, so a_3 a_1 a_2 like this. So here we will do a_3 and a_1 and a_3 so if we reach with here we can take a decision 2, 1, 3 and here we will do the subsequent comparison, so here we reach a decision, this is basically 2, 3, 1 and here 3, 2, 1.

So this is the decision tree prior each level is denoted by.

(Refer Slide Time: 10:57)



So each level is denoted by i:j and this is the path where if i is less than j this is the path for the subsequent comparison but this is the path we will follow if i is less than j and this is the path we follow if i is greater than j, and then we will do the subsequent comparison here, subsequent comparison here and we will reach to here, this is also each node is like this i:j, that means we are comparing A[i] and A[j] and finally we are reaching to a decision, these are all loop nodes.

So these loop nodes are basically the decision, okay. Now we can take an example, suppose we have 9, 4, 6, so a_1 is 9 a_2 is 4 and a_3 is 6 so we compare $a_1$and a_2 so which path we should follow? If we compare a_1 and a_2 then a1>a2 so we follow this path 9>4 and now here we come, so we will, here we compare a1 and a3 so 9>6 so we will take this part and here we compare a2 and a3, so a2 is 4 a3 is 6 so a2 is less than, we follow this part, so we reach to a decision that a2 < a3 < a1 so a2 is what? A2 is 4 which is less than a3 which is 6 which is < 9, so this is the sorted one.

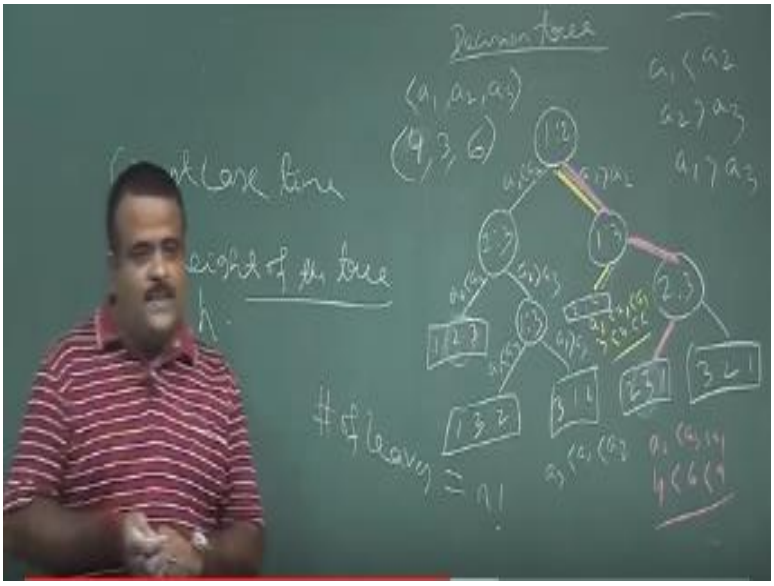
So we reach to a decision by following this path, so leap, all the leaps are the decision so for any input, if we have a input we are following this path, okay. So now if we have say if we have, the example is suppose we want to have this, this part so if we have say 9, not 9 if we have 4 then 3

and then 6 if we have this is our input then which path we should follow? So this is our a1, a1>a2 so you must follow this path and then a1 < a3.

So you must follow this path, so here the decision is a2 < a1 < a3 so a2 is what? A2 is 3 < 4 < 5 so this is the decision, so this we are receiving very fast so depending on the input, so if we take this path it is faster so what is the time for this? If we have a input what is the time, time is the basically the path we are following, the length of that path, okay. This is shorter length but this is bigger length.

So we are talking about vast case so what will be the vast case time? Vast case time will be the maximum length of the path, so maximum level of this tree and that is called height of a tree, maximum depth is called height.

(Refer Slide Time: 14:35)



So vast case, worst case time is basically height of the tree, okay. Now if we look at any comparison in the sort quick sort, insertion sort, merge sort anything and if we execute that that is basically following a path in a decision tree.

Because in any comparison based sort we are comparing two element and then we are taking some decision so we are following any of this path for the further comparison, and then finally we are going to a decision finally we are going to a leap level, so any comparison sort, base sort, execution on a input is basically following a path in the decision tree because there also we are comparing two element and we are taking some decision.

And then after that we are further comparing so that is the, the path of a decision tree so the worst case of that comparison based sort is basically height of this tree, so if you denote this height by h then how we can calculate 'h', now what is the number of leaps? Number of leaps is basically this nodes, this nodes are basically permutation of this nodes. So that is basically factorial n.

Because there are n nodes so this is factorial n, so if it is factorial n then is there any relationship we can draw from with the n and this number of leaps, so this is a binary tree so if it is a complete binary tree then what will be the number of leaps? Number of leaps should be 2^h, okay. So $2^h > n$ because this is not a complete binary tree because this is ending here, okay it is complete means it should go at the end.

So this is the relationship we have, okay. So now how we can get something from here, so h is basically greater than equal to log (n!). Now factorial n can be approximate by $(n/e)^n$ this is the Stirling's formula, okay. So if we use these approximation then what we have? We have basically $h \geq \log(n/e)^n$ is this, so this we can take $n \log n - n \log_2 e$, so this is basically greater than equal to n log n.

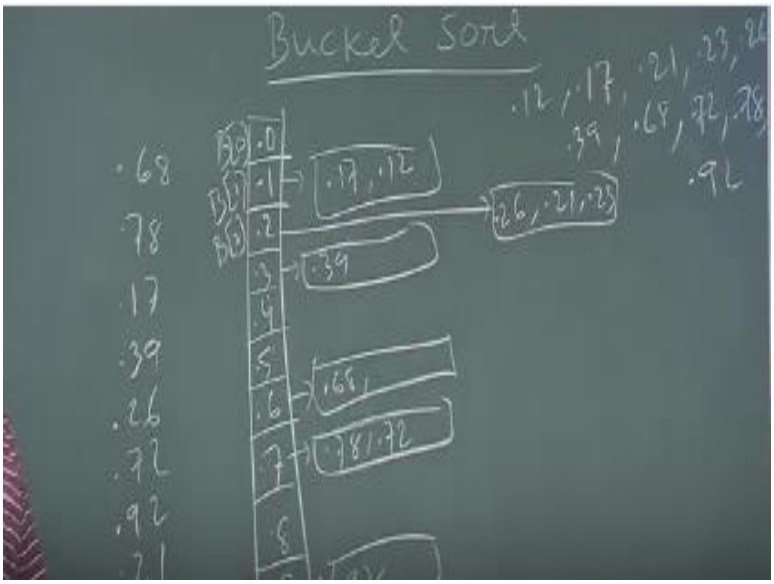
So height is basically Ω (log n) so height is nothing but the worst case runtime for any comparison based sort because that is the execution of the decision tree from the root to the leaf, if you take any input then for any comparison based sort you have to take root to the leaf, this is the height.

(Refer Slide Time: 18:19)

So the any, so, so the worst case, worst case run time for any comparison based sort is equal to height of the tree, decision tree which is basically Ω(n logn) sorry Ω(n logn) this is n. So this we have proved that we cannot do better than n logn in the worst case if we are using comparison based sort.

So then what is the solution if we want to have a linear time sorting algorithm, so we cannot use comparison in the sort, so there we will be using what is called bucket sort, so there we will not compare between two elements, what we will do, we will see the value of the element and we will put into the bucket, so no comparison between the elements, if we have to compare between the element we cannot do better than n logn in the worst case. But we want to do a linear time sorting algorithm so that is not a comparison in the sort, so those are called bucket sort, we will see the value of the element, and we will throw into the bucket.

(Refer Slide Time: 19:55)



In a bucket sort what we are doing so this is not a comparison the sort, so we have some, we are taking some bucket, say suppose the numbers are elements are coming from the interval 0 and 1 and we have some equal sub intervals suppose there are A in sub intervals, B n-1, so what we are doing in bucket sort, we are just taking a number and we are putting into that bucket and then after storing in the bucket we are now sorting individual bucket by the insertion sort. So we will take an example, so suppose all numbers are like this say .78, .17, .39, .26, .72, .92, .21, .12, .23, then .68, okay so all the numbers are between 0 and 1.

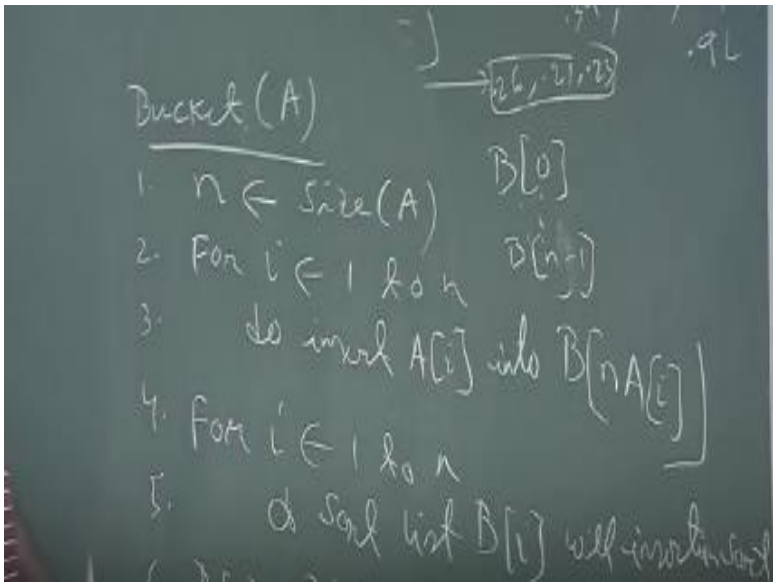
And suppose you have bucket say starting from 0 to .0, 0.1 like this, so we have say, B[0] this is basically .0 bucket like this I mean, so B[1] .1 like this, B[2] .2 like this, so point I mean 0 to 9 so B[9], okay. so this is .6 by so .3, .4, .5, .6, .7, .8 and last one is ninth bucket this is represented by .9. so this is, this will go to the sixth bucket, so we have this bucket so we will put here .06, and then point so this will go to the seventh .7 bucket, so .78 like this we are filling the bucket we are checking the element and throwing in the bucket.

So 1,7 .17 and then 39, .39, .26 so it will be here, .26 then again .72, .72, .92, .92, .21, .21, .12, .23 will be here, so this is the bucket filling. Now we have individual bucket and this number we

are going to sort by insertion sort. So suppose in higher bucket in is the number element putting to the higher bucket so we are applying the insertion sort one this and we get this point 12 then point 17 then again we apply the insertion sort on this we get so point 21.23.26 then there is only in the mean point 39 only one element point the you have two we will applying some sort on this 178 like this and point 9.2 sorted okay.

So this is typically bucket sort we will see the element and the throwing into the bucket and then we will sort this bucket using the insertion so now what is the time complexity of this so here in this case we are we have 0 to 9 bucket but in general suppose you have in buckets.

(Refer Slide Time: 24:04)



So let us write the code for that so bucket sort on so what we do n = size of the size of the array the number of element and then for this is the filling the bucket. What we are doing we are insert we are inserting sorry A(i) into the B (nA(i) okay so we have big row up to B(n) bucket so then so we have basically n buckets so this is the bucket filling then individual bucket into sort using the insertion sort for i= 1 to n there are n buckets so we sort, sort the least what are the element in the B(i) higher bucket with insertion sort insertion sort and then we just bring the bucket we just

con cutting it this bucket we just bring this bucket B(1) like this in – 1 if it is up to n- 1 we are studying from 0 so this is the typical bucket sort now what is the time complexity for this.

(Refer Slide Time: 25:56)

So time complexity will depend on suppose ni is the ni = number of element going to higher bucket going to higher bucket okay then the run time for this is T(n) = so order of n this is for filling the bucket and then individual bucket and then individual bucket we are sorting so there are ni, ni is the, so ni^2 this is the time taking by the insertion sort there are ni element in the higher bucket so time is ni^2 then there are n buckets.

So this is the time complexity for this bucket sort now if you take the expectation of it because ni we don't know how many number will be going to higher bucket so if you do the expecting analysis of this so this is basically $\Theta n + \sum$ of expectation of ni^2 and these can be proven prove that this is also linear by some taking some indicator random array we suppose we assume this is one if the ith element if I is going to it, then this one if Aj is going to higher bucket 0 otherwise then basically ni is basically $\sum$ of X_{ij} j = 0 to 1 to n.

Okay, so then now by taking the square of this turn I am taking the expectation we can saw this is basically linear so this is Θ away so this is the bucket sort time complexity it is a linear time algorithm.