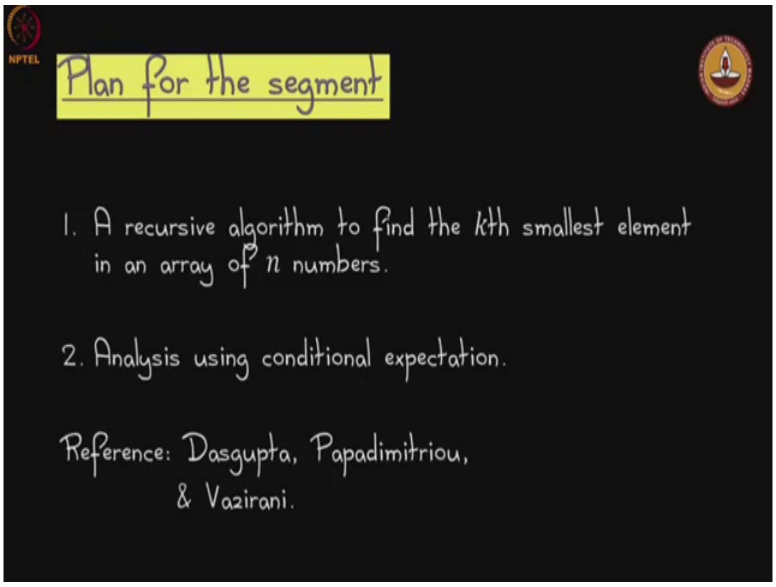
Probability & Computing
Prof. John Augustine
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 02
Discrete Random Variables
Lecture - 14
Randomized Selection

So, now we are in segment 6 of module 2, and we are going to talk about the problem of randomized selection, a special case of it is the problem finding median in an array of numbers.

(Refer Slide Time: 00:30)



Plan for the segment

1. A recursive algorithm to find the kth smallest element in an array of n numbers.

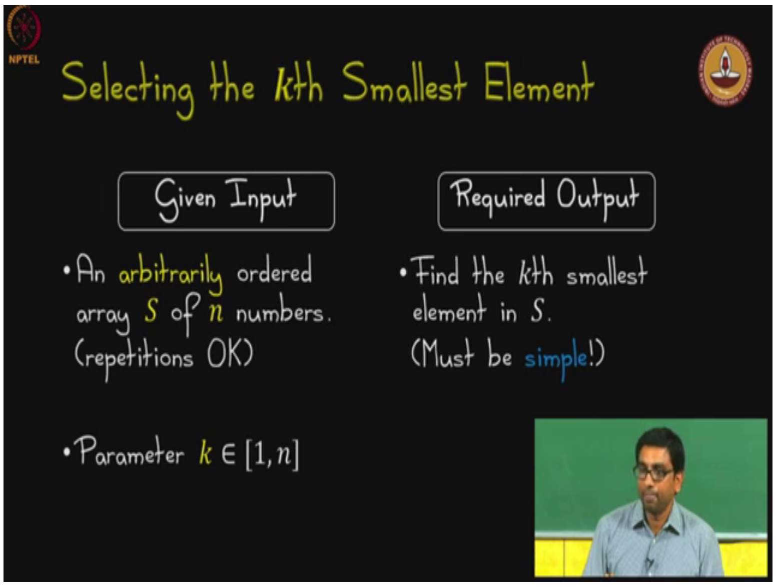
2. Analysis using conditional expectation.

Reference: Dasgupta, Papadimitriou, & Vazirani.

So, what we will do is we present a recursive algorithm to find the kth smallest element in an array of n numbers and this is going to be of course, randaomized algorithm and quite chances are that you have seen a deterministic algorithm for this problem. And if you recall that the deterministic algorithm is probably not a very trivial algorithm it is somewhat complicated, but the algorithm that we are going to present today is very simple and elegant ok.

And we will exercise our understanding of conditional expectation to analyze this algorithm the reference for this algorithm is Dasgupta, Papadimitriou Vazirani, it is the basic book on algorithms, but they do not have a complete analysis. So, you have to pay attention to the analysis here.

(Refer Slide Time: 01:25)



Selecting the kth Smallest Element

Given Input

- An arbitrarily ordered array S of n numbers. (repetitions OK)
- Parameter $k \in [1, n]$

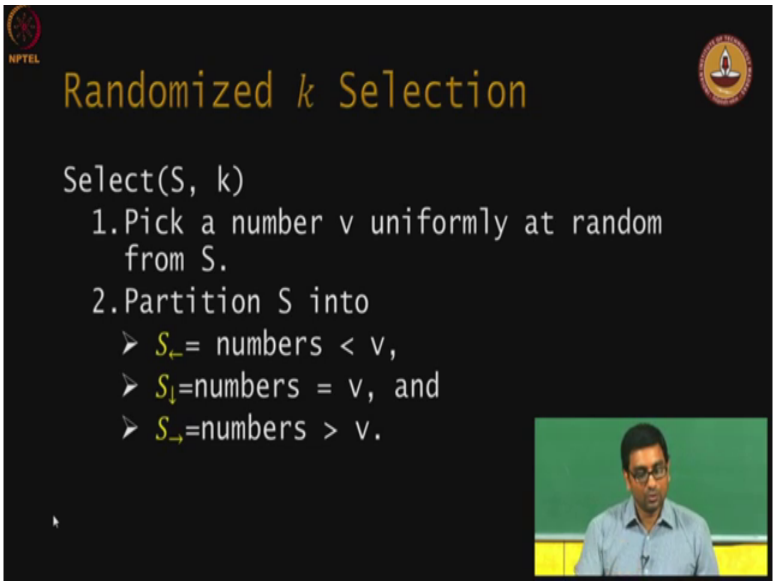
Required Output

- Find the kth smallest element in S. (Must be simple!)

So, let us just formally define the problem. Your input is an arbitrarily ordered array of n numbers repetitions are ok, it is arbitrarily or ordered of course, if it is sorted then it will be easy to solve this problem. So, it is an arbitrary ordered array of numbers. And you are given an input parameter k, and what are you asked to find out you are required to find the kth smallest element in the array S of course, when k is n over 2 you get the media and importantly we want this algorithm to be simple because we already know complicated algorithm for it ok.

So, got further do as just make sure is a problem create everybody ok.

(Refer Slide Time: 02:16)



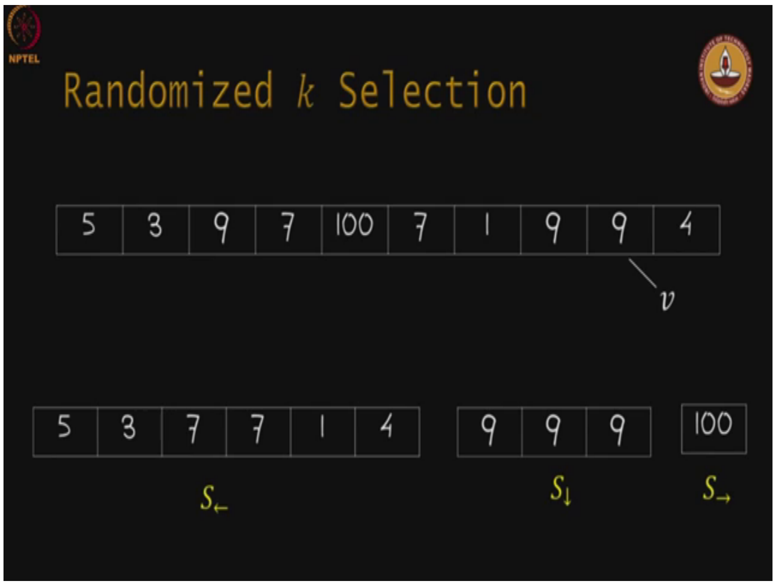
Randomized k Selection

```
Select(S, k)
  1. Pick a number v uniformly at random
     from S.
  2. Partition S into
     ➢ $S_{\leftarrow}$ = numbers < v,
     ➢ $S_{\downarrow}$ =numbers = v, and
     ➢ $S_{\rightarrow}$ =numbers > v.
```

So, let us proceed to understanding the algorithm it is basically recursive. So, we are going to call this high level function calls select and the inputs will be the array S and the parameter k ok. So, here is where the randomness comes up. So, we start by picking a random a number uniformly at random from the set S.

So, it is basically some element in this array ok. And we look at that value v and we use that to partition S into 3 sub arrays this is a very intuitive algorithm you think about it. So, whenever, you can just you basically run a for loop through the entire list of elements in S and for each element that is less than v you include it in this the first sub array S left if you will, and whenever the number you encounter is greater than v you include it in S right arrow of (Refer Time: 03:23) ok, and whenever and you because of duplications you could have numbers that equal v which you just put them into S down arrow ok. So, you get 3 sub lists.

(Refer Slide Time: 03:37)



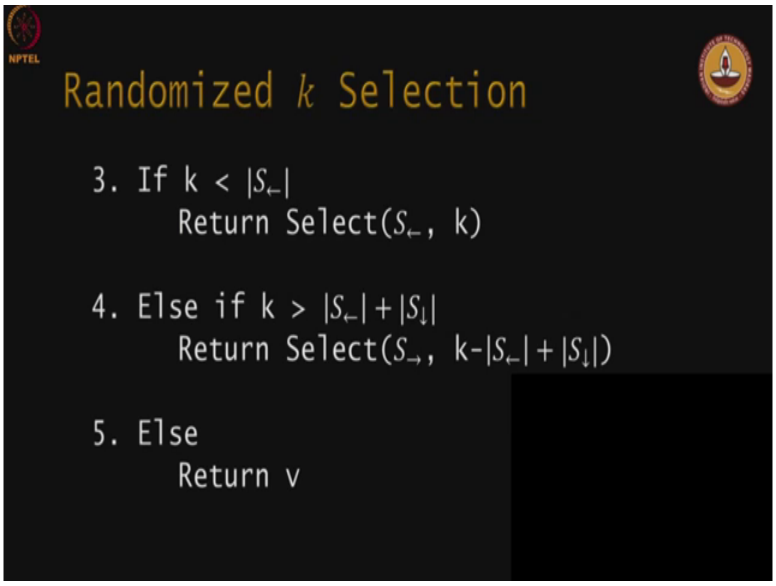
Randomized k Selection

5	3	9	7	100	7	1	9	9	4

So, here is an example quick example. So, let us say this is the array that you have you pick a random element within the array. And what do you do? All those numbers that are less than 9 you put them in the left array you have 3 nines. So, you put them in the S down array and then there is one element greater than 9. So, you put that in the S right array.

Let us actually look at this. Suppose you want to find the third element in this array which sub list will you look into clearly you look into S left ok. If you on the other hand where to look at, if you are looking for the 8th element in the array there are ten elements in this if I guess. So, 8th you intuitively know that you only have to look at the S down and so on. So, the moment you break it up into these sort of sub lists you know which way to recurs into ok. So, that is exactly what we doing here.

(Refer Slide Time: 04:38)



Randomized k Selection

```
3. If k < $|S_{\leftarrow}|$
       Return Select($S_{\leftarrow}$, k)

4. Else if k > $|S_{\leftarrow}| + |S_{\downarrow}|$
       Return Select($S_{\rightarrow}$, k-$|S_{\leftarrow}| + |S_{\downarrow}|$)

5. Else
       Return v
```

If the input parameter k is less than the cardinality of the less left sub array then we know that in it is its going to be somewhere in this sub array.

So, we basically just invoke select S left and use the parameter k, otherwise we check if the k that we are looking for is greater than the cardinality of this element this array plus the cardinality of this array ok. If it is greater than these 2 cardinalities then it we are we must be looking at. So, they are not going to the case the case smallest element is not going to fall in the first two sub arrays, it is going to be in the third sub array ok. So, that is satisfy we will recurs in to the third sub array. But we need to be a bit careful here. So, what will the parameter be over here?

Student: k minus (Refer Time: 05:39).

k minus S. So, you will have to do k minus the cardinality of S sub exact it.

Student: (Refer Time: 05:57).

Yeah, yeah ok. So, that is going to be our new parameter and otherwise what is the option that we have if it is not going to be recursing into the left array or the right array you are left with the middle array, but all the elements in the middle array are just the element v. So, we simply return v it is a very intuitive simple algorithm. So, any questions?

Student: (Refer Time: 06:38).

Oh, where.

Student: (Refer Time: 06:42) k minus (Refer Time: 06:48) the cardinalities should be (Refer Time: 06:54), it should be k minus of (Refer Time: 06:57).

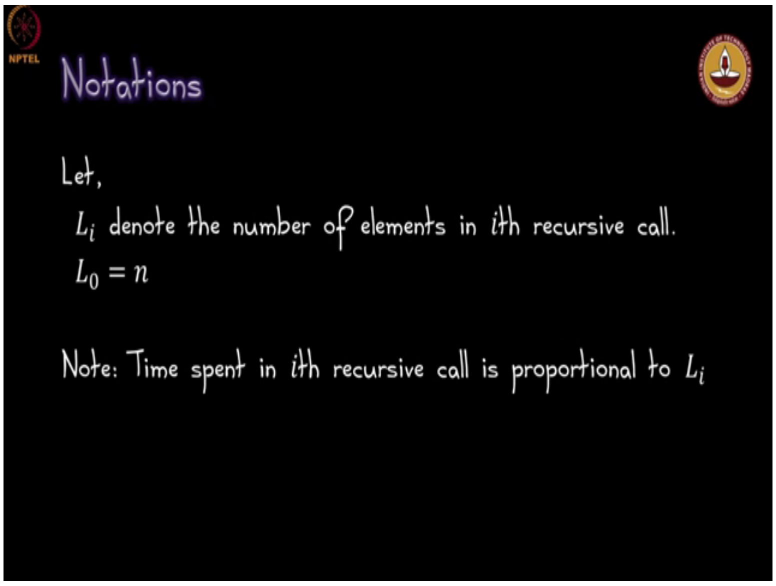
Ha, ok, yeah sorry.

Student: (Refer Time: 07:04).

Yeah, thank you yeah. So, other than that is the algorithm clear to everybody.

Let us now analyze this algorithm. We are going to see that the running time of this algorithm actually is o of n on expectation. Let us try to understand why that is the case.

(Refer Slide Time: 07:27)



Notations

Let,

L_i denote the number of elements in ith recursive call.

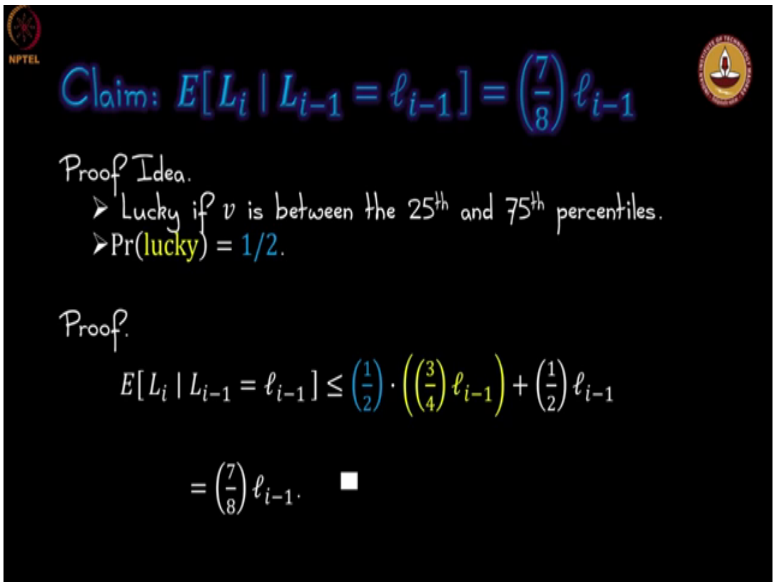
$L_0 = n$

Note: Time spent in ith recursive call is proportional to L_i

Let us use L i to denote the number of elements in the ith recursive call we start with n elements. So, let us start with L 0 and L 0 is n.

Notice that the time spent in the ith recursive call is proportional to L i. So, we might do other things. But essentially what we do in the ith iteration is or the ith recursive call is we sweep through the list of elements that we are focusing on, and try to understand which ones are less than the pivot element and which ones are greater than the pivot element. So, it requires that one sweep and therefore, the time spent in the ith recursive call is proportional to the size of the sub array that is focused on ok.

(Refer Slide Time: 08:25)



Claim: $E[L_i \mid L_{i-1} = \ell_{i-1}] = \left(\frac{7}{8}\right)\ell_{i-1}$

Proof Idea.

- Lucky if v is between the 25th and 75th percentiles.
- Pr(lucky) = 1/2.

Proof.

$$E[L_i \mid L_{i-1} = \ell_{i-1}] \leq \left(\frac{1}{2}\right) \cdot \left(\left(\frac{3}{4}\right)\ell_{i-1}\right) + \left(\frac{1}{2}\right)\ell_{i-1}$$

$$= \left(\frac{7}{8}\right)\ell_{i-1}. \quad \blacksquare$$

So, now let us make a claim. What we claim is that the expected size of the array at iteration ith that is L i given that in the size of the array in the previous iteration was some fixed l sub i minus 1. Now, given the that is that was the size of the array in the previous recursive call what is the expected size of the array in the current recursive call we claimed that is going to be a fraction of the previous size of fraction 7 over eight ok.

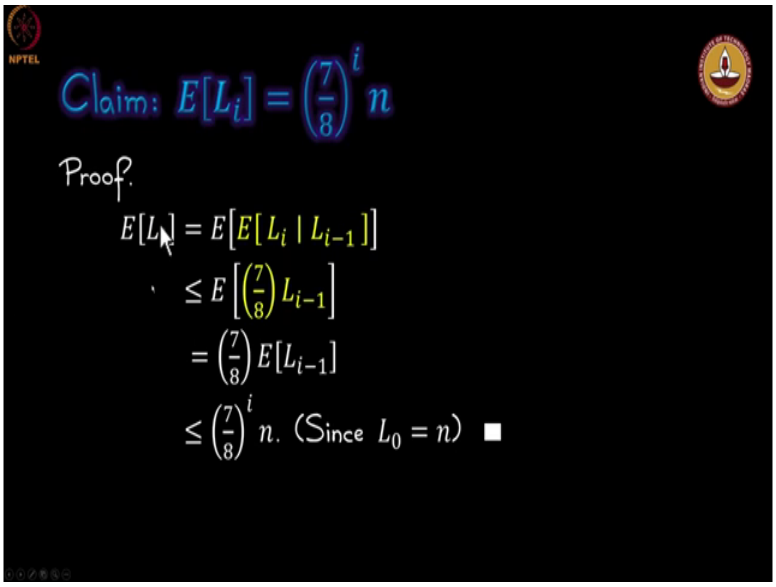
Let us try to understand why that is the case. Look at let us look at the pivot that is that we chose we chose that is pivot v if we are lucky that pivot is going to fall between the 25th and the 75th percentiles and clearly the probability of being lucky is one-half. So, how do these ideas play out? So, let us try to be a little bit more careful about this what we care to bound is this expectation on L i given this condition. And let us look at how that can be worked out.

Well, as we noted if we are lucky the size of the array is going to shrink otherwise it could be unlucky and it is hard to figure out exactly what does what it is going to be is, but since we are just interested in an upper bound we can just use we can just assume that there is no shrink in the size of the array at all ok.

But now here the being lucky and un unlucky happened with probabilities half and half. Now, you may ask why do we claim that the size of the array shrinks to three-fourths are the previous size when we are lucky. Well, to understand this you should realize that the two extremes are within the lucky region is the 25th percentile and the 75th percentile. So, let us look at these two extremes they happen to be symmetric. So, let us just focus on what happens if the element we choose the pivot element is the 25th element.

Now, that is going to divide up the array into a quarter of the array and the remaining three-fourth of the array and the median we will be able to figure out is in the larger of those two which is three-fourth of the array. And therefore, the shrink in the size of the array is factor of three-fourths at least provided we get lucky, and that is exactly why we have this three-fourths over here. And when you evaluate this expression you get 7 over 8 times L i minus 1.

(Refer Slide Time: 11:31)



Claim: $E[L_i] = \left(\frac{7}{8}\right)^i n$

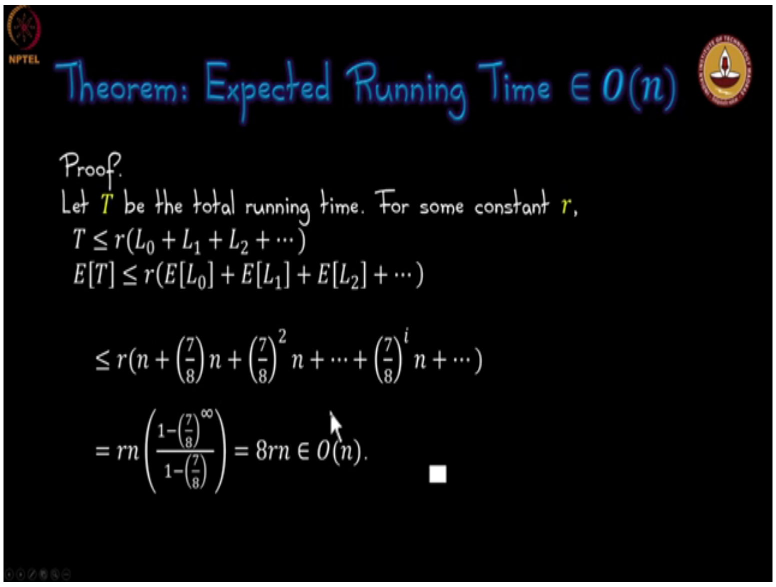
Proof.

$$\begin{aligned} E[L_i] &= E[E[L_i \mid L_{i-1}]] \\ &\le E\left[\left(\frac{7}{8}\right) L_{i-1}\right] \\ &= \left(\frac{7}{8}\right) E[L_{i-1}] \\ &\le \left(\frac{7}{8}\right)^i n. \text{ (Since } L_0 = n) \quad \blacksquare \end{aligned}$$

Now, we want to bound this expectation of L i. And we know that the expectation of this L i is nothing, but the expectation of the expected expectation of L i conditioned on L i minus 1 ok, and keep in mind that we already have a handle on this conditional expectation inside and that is going to be 7 over 8 times L i minus 1. We worked it out for a specific value, but for any random variable also it is going to be 7 over 8 times L i minus 1 and by linearity of expectation you get 7 over 8 outside and so expectation of L i is 7 over eight times expectation of L i minus 1.

And you can continue this recursively and as a result we would get expectation of L i to be 7 over 8 raised to the power i times n because when the recursion ends at L 0 you will just have n over there ok.

(Refer Slide Time: 12:41)



Theorem: Expected Running Time $\in O(n)$

Proof.

Let T be the total running time. For some constant r,

$$T \leq r(L_0 + L_1 + L_2 + \cdots)$$

$$E[T] \leq r(E[L_0] + E[L_1] + E[L_2] + \cdots)$$

$$\leq r\left(n + \left(\frac{7}{8}\right)n + \left(\frac{7}{8}\right)^2 n + \cdots + \left(\frac{7}{8}\right)^i n + \cdots\right)$$

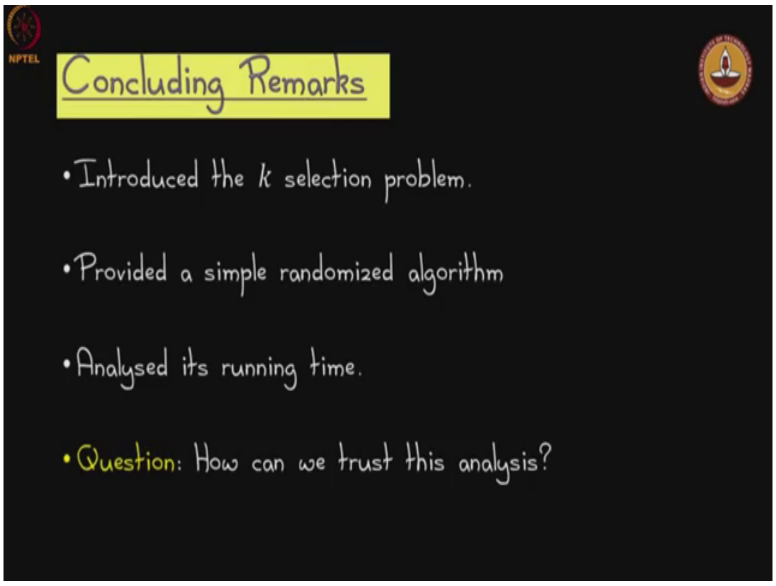
$$= rn\left(\frac{1-\left(\frac{7}{8}\right)^\infty}{1-\left(\frac{7}{8}\right)}\right) = 8rn \in O(n). \quad \blacksquare$$

So, how does this impact the running? Time we claimed that the expected running time is O of n and let us see why that is the case. Let us denote the running time as T.

Now, if you recall I said that the running time in each recursive calls proportional to the number of elements that we consider. So, let r capture that proportionality, so r some constant. So, the running time T is at most r times the number of elements that were there in the 0th recursive call the first recursive call and so on and so forth ok.

And if T is that value then the expectation of T by linearity of expectation and just taking the expectation into the expression you will get r times expectation of L 0 plus expectation of L 1 plus expectation of L 2 and so on ok. And we have already derived the value for the expectation L i. So, we can plug that value into this expression and what we will get is the sum of a geometric series which is not hard to evaluate and we will get a value of an upper bound of 8 r n for the expected running time which of course, because r and 8 are both constants this upper bound is O of n.

(Refer Slide Time: 14:18)



Concluding Remarks

- Introduced the k selection problem.
- Provided a simple randomized algorithm
- Analysed its running time.
- Question: How can we trust this analysis?

So, let us conclude this segment. We introduced the k selection problem which is a generalization of the median problem, provided a very simple randomized algorithm ok. We analyzed its running time and we show that it runs oh on expectation in O of n time. But there is one question that I want you to ask yourself. So, this is an expected time running time analysis, how can you trust this answer. So, can you be sure that will not take too much time in the worst case ok? What does that even mean ok?

So, these are some questions that we need to ask ourselves because when you when you claim something on expectation you are not guaranteed to end in O of n time, there could be situations where the actual running time could be much higher than n that is a possibility and if you are going to run a mission critical operation in which this shows up as a critical piece and you need some more better guarantees than this. So, that is something that should ranking you and hopefully we will be able to address that in a subsequent module ok.

So, with that we come to the end of this module 2, where we talked about random variables and the expected expectation of random variables and we looked at a couple of examples.

(Refer Slide Time: 15:55)

Next Module

Tail Bounds and Claims that hold with high probability

In the next module we will be talking about tail bounds and we will address exactly this question that is hopefully ranking you. We will try to go from expected time analysis not just go away from it expected time analysis is also important. But we will also try to make claims that hold with high probability. So, that we have this extra guarantee that with high probability and we can actually make it arbitrarily high, with high probability the running times would be good enough for us ok.

So, that is that will be the focus of the next modules. Actually not just one module might spill over into multiple modules ok. So, with that we will close the module 2.

Thank you.