

Design and Analysis of Algorithms, Chennai
Mathematical Institute
Prof. Madhavan Mukund
Department of Computer Science and
Engineering,

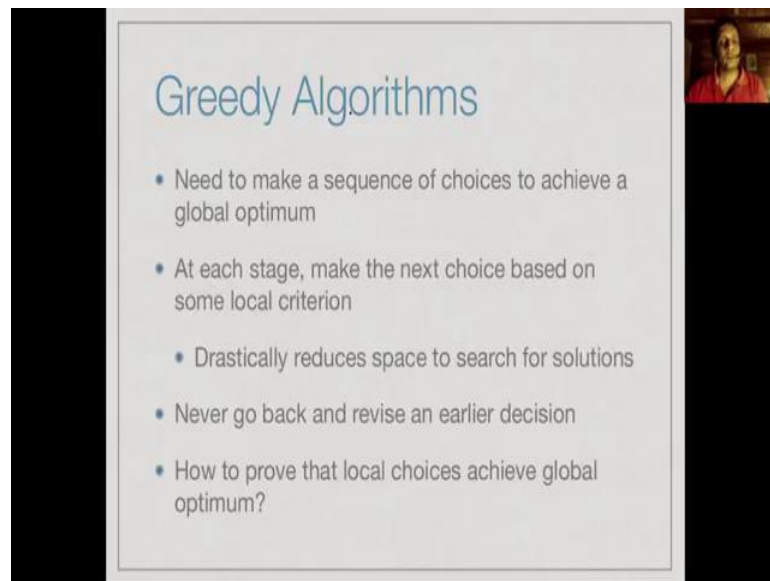
Module – 03

Lecture - 41

Greedy algorithms: Interval scheduling

Let us take another look at greedy algorithms.

(Refer Slide Time: 00:05)



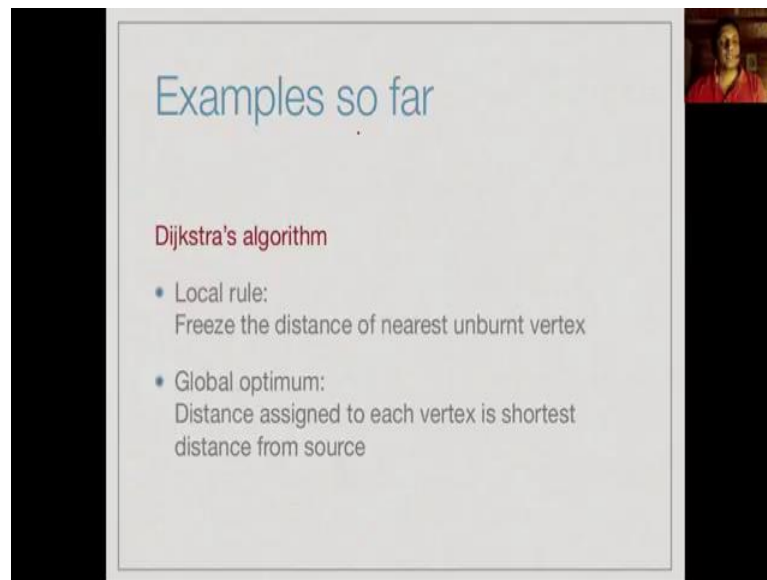
Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
 - Drastically reduces space to search for solutions
- Never go back and revise an earlier decision
- How to prove that local choices achieve global optimum?

So, we are looking at algorithms where we need to achieve a global optimum by making a sequence of choices. So, in a greedy strategy what we do is we make the next choice based on some local criteria. So, there maybe a number of choices we could make, but we just pick one of them based on something which looks good at the moment and now we never go back and revise an earlier decision.

So, we deterministically search through this space of solutions by picking of good choice at each step and this drastically reduces the space in which we have to search. So, the trickiest thing is that, this strategy very often does not work. So, if you have a greedy strategy in mind, we need to go back and prove that the way we made our local choices actually achieves the global optimum.

(Refer Slide Time: 00:53)



The slide is titled "Examples so far" in a large, light blue font. Below the title, the text "Dijkstra's algorithm" is written in a smaller, red font. There are two bullet points, each preceded by a small blue dot. The first bullet point is "Local rule:" followed by the text "Freeze the distance of nearest unburnt vertex". The second bullet point is "Global optimum:" followed by the text "Distance assigned to each vertex is shortest distance from source". In the top right corner of the slide, there is a small, square video feed showing a person with dark hair wearing a red shirt.

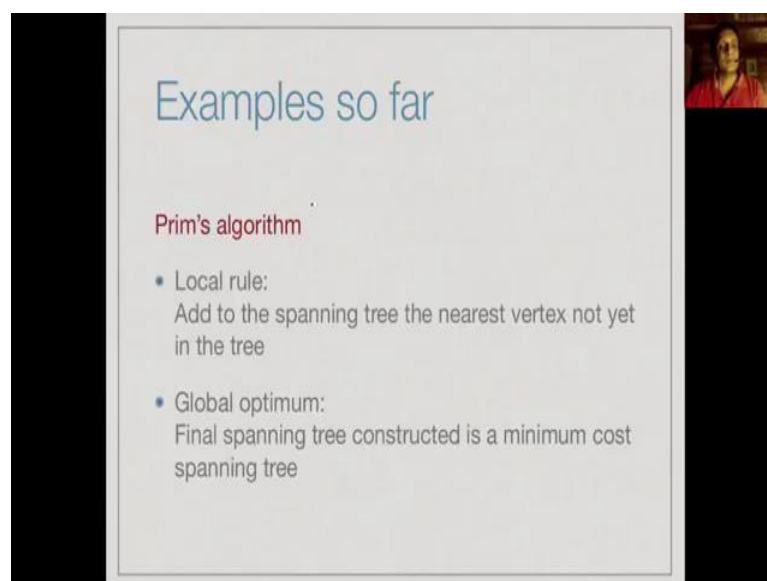
Examples so far

Dijkstra's algorithm

- Local rule:
Freeze the distance of nearest unburnt vertex
- Global optimum:
Distance assigned to each vertex is shortest distance from source

So, we have seen three algorithms so far which follow this 3D paradigm. The first was Dijkstra's algorithm for the single source shortest path problem. So, recall that in this algorithm we kept burning vertices and at each stage we froze the distance to the nearest unburnt vertex and claim that this would in fact be the shortest distance to that vertex from the source. So, globally the optimum we achieved in this algorithm is that the distance assigned by this greedy strategy happens to be the shortest distance from the source.

(Refer Slide Time: 01:30)



The slide is titled "Examples so far" in a large, light blue font. Below the title, the text "Prim's algorithm" is written in a smaller, red font. There are two bullet points, each preceded by a small blue dot. The first bullet point is "Local rule:" followed by the text "Add to the spanning tree the nearest vertex not yet in the tree". The second bullet point is "Global optimum:" followed by the text "Final spanning tree constructed is a minimum cost spanning tree". In the top right corner of the slide, there is a small, square video feed showing a person with dark hair wearing a red shirt.

Examples so far

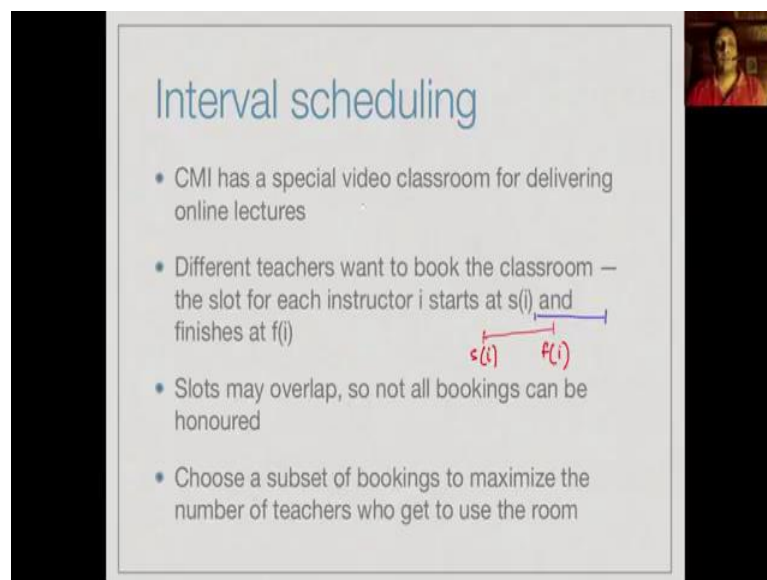
Prim's algorithm

- Local rule:
Add to the spanning tree the nearest vertex not yet in the tree
- Global optimum:
Final spanning tree constructed is a minimum cost spanning tree

A closely related algorithm is Prim's algorithm for the minimum cost spanning tree. So, here we incrementally build up a tree and at each stage we add to this spanning tree, the nearest vertex that is not yet in the tree. And here the global optimum that we achieved is that we construct the spanning tree that is minimum cost. Another algorithm for a minimum cost spanning tree is Kruskal's algorithm.

Here, we do not build up a tree directly, but rather we keep collecting edges and form a connected component overall which becomes a tree. So, here we keep adding to the current set of edges in our set, the next smallest edge that does not form a cycle with those at we have already choose and now the global optimum is that the edges that we collect in this way form a minimum cost spanning tree.

(Refer Slide Time: 02:31)



The slide is titled "Interval scheduling" in blue text. It contains four bullet points:

- CMI has a special video classroom for delivering online lectures
- Different teachers want to book the classroom — the slot for each instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

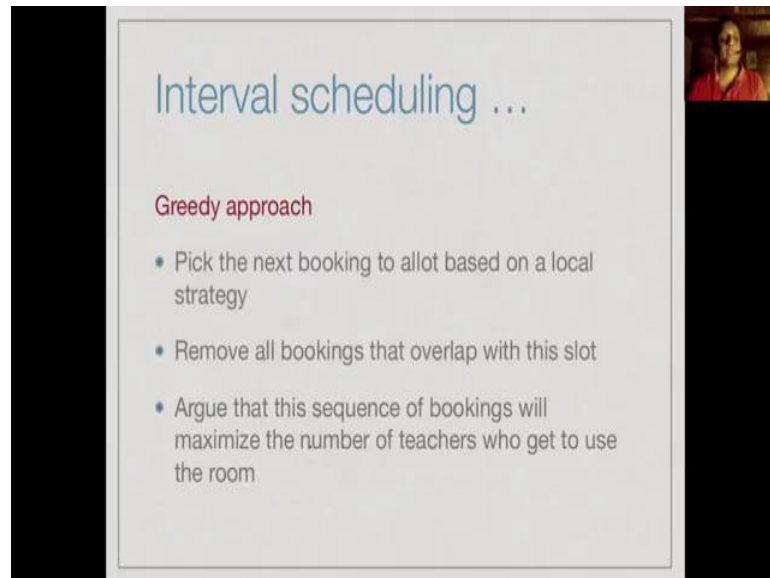
 To the right of the second bullet point, there is a diagram showing two horizontal intervals. The first interval is red and labeled $s(i)$ at its start and $f(i)$ at its end. The second interval is blue and starts before the red interval ends and ends after it, illustrating an overlap.

So, now let us look at a completely different problem, a problem called interval scheduling. So, suppose we have a special video class room, where we can deliver online lectures. Now, different teachers want to book the class room to deliver classes and each instructor has a slot that he would like to deliver this lecture in. So, instructor i has a slot, let us starts at a time s_i and finishes it at f_i . So, you have a slot which starts at s_i and finishes at f_i , now two instructors may have over lapping slot.

So, the maybe somebody who wants the slot like this, so the blue slot starts before the red slot ends, so obviously both these slots cannot be in given bookings, because there were interfere with each other. So, our task is to look at the set of bookings and chooses

subset which is feasible that is no two bookings that we choose interfere with each other. So, there we maximize a number of teachers who get to use the room.

(Refer Slide Time: 03:40)



Interval scheduling ...

Greedy approach

- Pick the next booking to allot based on a local strategy
- Remove all bookings that overlap with this slot
- Argue that this sequence of bookings will maximize the number of teachers who get to use the room

So, broadly if we follow a greedy approach, this is what we would do. Among all the bookings that are not yet allocated and which are still available to allocate. We will pick one based on some local strategy, then we would remove all conflicting bookings, bookings that overlapped with this booking that with the slot that we just allocated and somehow we have to argue that this sequence of bookings that we are choosing maximizes the number of teachers who get to use the room.

(Refer Slide Time: 04:12)

Interval scheduling ...

Greedy strategy 1

- Choose the booking whose start time is earliest
- Counterexample

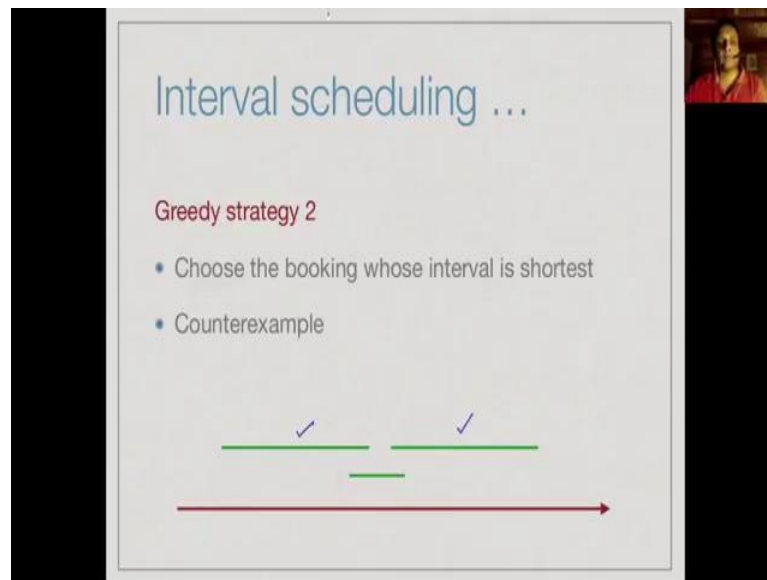
bookings

time

So, let us look at some typical greedy strategies that one might want. So, one strategy might be to choose the booking whose start time is earliest, but it is not difficult to come up with the counter example. So, if you look at this picture, there is one long green booking it starts earliest and in fact ends after all the other bookings are made.

So, if we use this greedy strategy we would allocate this very long booking and the entire period it will be allocated to just one teacher, whereas if we choose the booking that starts a little later, then we could actually satisfy six teachers' bookings and since our goal is to maximize the number of teachers who can use the room, that could be a better strategy. So, this greedy strategy is clearly flopped.

(Refer Slide Time: 05:00)



Interval scheduling ...

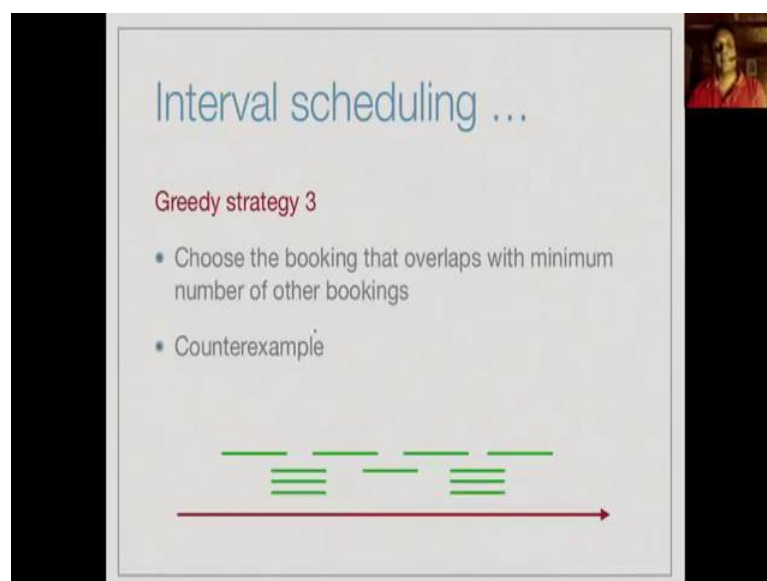
Greedy strategy 2

- Choose the booking whose interval is shortest
- Counterexample

The diagram illustrates a counterexample for Greedy strategy 2. It shows a horizontal timeline with three green intervals. The middle interval is the shortest. Two longer intervals are positioned on either side of the middle one, such that they overlap with it. A red arrow at the bottom indicates the timeline. The middle interval is marked with a checkmark, suggesting it was chosen by the greedy strategy, which leads to a suboptimal solution as the two longer intervals cannot be scheduled simultaneously.

And other greedy strategy we might think of is to choose a booking whose interval is shortest. Once again here is a counter example, the interval in the middle is the shortest one, but if we choose this it is in conflict with both the other bookings, so we have to rule both of them optimum. So, if we choose a shortest interval then we can only allocate one teacher to the room, whereas if we know the strategy and if we choose the too longer intervals, then we can actually use the room for two teachers and get a better optimum for the problem that we have chosen.

(Refer Slide Time: 05:40)



Interval scheduling ...

Greedy strategy 3

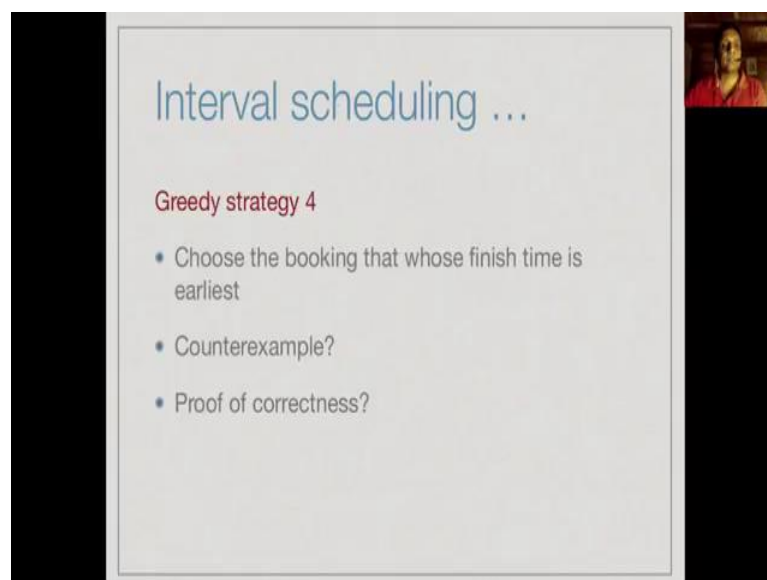
- Choose the booking that overlaps with minimum number of other bookings
- Counterexample

The diagram illustrates a counterexample for Greedy strategy 3. It shows a horizontal timeline with several green intervals. There are two long intervals at the ends and several shorter intervals in the middle. The shorter intervals are arranged such that they overlap with both long intervals. A red arrow at the bottom indicates the timeline. The diagram suggests that choosing a booking that overlaps with the minimum number of other bookings (likely one of the shorter intervals) might not lead to an optimal solution.

So, the previous example suggest that there is something to do conflicts, so maybe we might choose bookings in terms of how many other bookings they ruled up. So, one strategy now we might think of is to choose the booking that overlaps with the minimum number of other bookings. In other words, by choosing this booking we rule out as few other bookings is possible. So, let us look at this example, here the center booking overlaps with only two, this one and this one, every other booking overlaps with at least three.

So, if we choose this booking, then we rule out the bookings on either side of it and that means, there we also, we can do either this one or one of these. So, if we take this center booking we can do at most three bookings overall, we cannot do the two in either side of it. So, we can either do the two extreme ones or we can do anyone of these and anyone of these. So, we can do a total of three, we can allocate a total three bookings among these. However, if we do not do this, if we choose a better strategy, a better strategy would be to clearly take the four on the top. So, we can allocate four teachers in this room, if we do not use this strategy, then we must pick the one with the minimum number of conflicts. So, this greedy strategy also fixed.

(Refer Slide Time: 07:17)



Interval scheduling ...

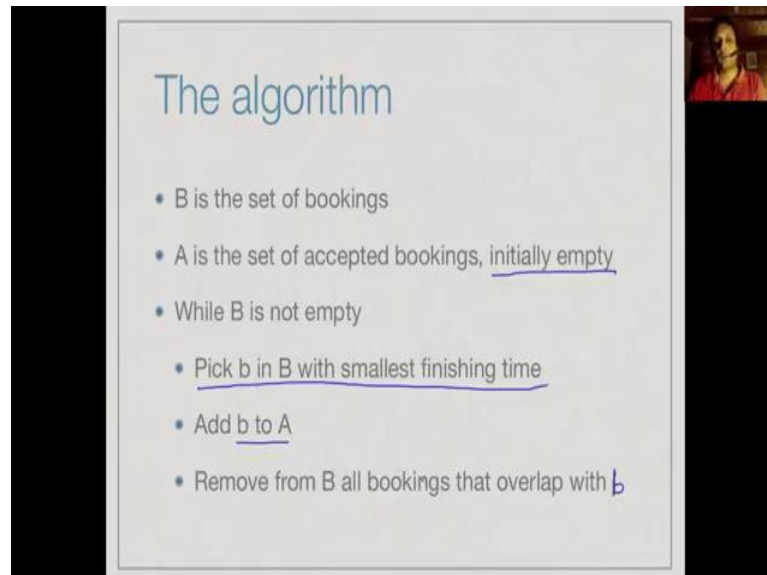
Greedy strategy 4

- Choose the booking that whose finish time is earliest
- Counterexample?
- Proof of correctness?

So, here is a fourth strategy, instead of choosing the one like we begin with whose start time is earliest, let us choose the one whose finished time is earliest. So, can we come up

with the counter example or should we instead try to prove this is correct. So, in fact this strategy does work and let us see how we can prove it.

(Refer Slide Time: 07:41)



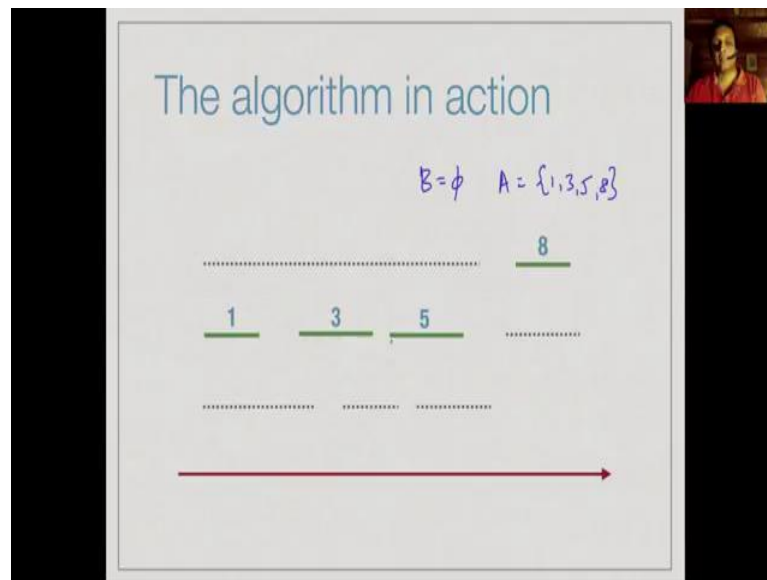
The algorithm

- B is the set of bookings
- A is the set of accepted bookings, initially empty
- While B is not empty
 - Pick b in B with smallest finishing time
 - Add b to A
 - Remove from B all bookings that overlap with b

Before we prove it, let us formally write down the algorithm a little more clearly. So, we start with the set of bookings B and we want to construct from this set, a subset A of accepted bookings. So, initially we have no accepted bookings, because we just starting to build this set and now we do the following. So, as long as we have pending bookings which are still feasible, we pick that booking which has the smallest finishing time among the set which is ((Refer Time: 08:14)) and we add that to b, that to A and now having added that to A, we cannot schedule any more bookings which overlapped with this b.

So, we remove from our set capital B, all the bookings which overlapped with the booking b that we just choose. So, each time we pick up the next booking which is still available with the smallest finishing time and we remove everything which is in conflict to it.

(Refer Slide Time: 08:44)

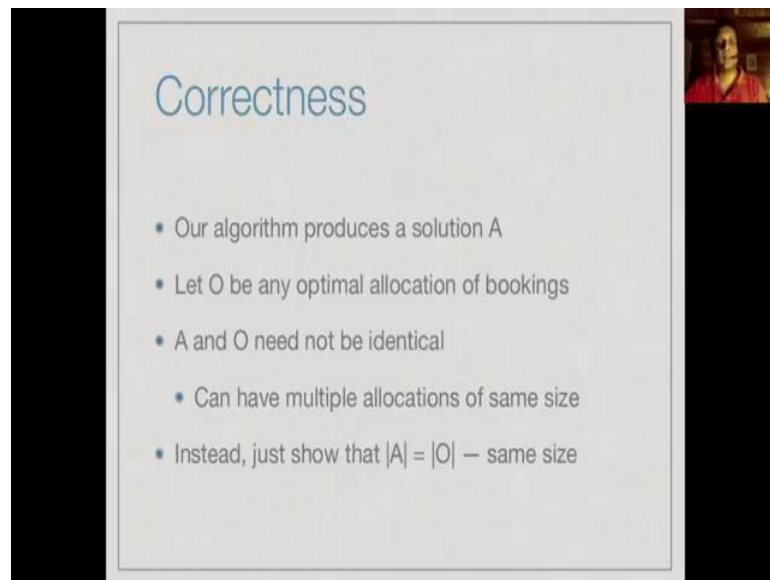


So, here is an example of power algorithm that work. So, here we have nine bookings, the blue lines indicate the bookings and the numbers of the bookings are given above it. So, in this, the one with the... So, initially our set B has all these nine bookings and our set A is initially empty. So, now what we look at, it is a smallest finishing time among nine bookings and that happens to be 1. So, we select 1 and then having selected 1, we find all the bookings which overlapped with it.

So, 2 overlaps with 1 and stored at 6, so we will move 2 from our set and we will remove 6 from our set, so now B has been thrown to 3, 4, 5, 6, 7, 8, 9 and A has the booking number 1. So, now among this feasible set 3, 4, 5, 6, 7, 8, 9 we pick the one that ends earliest which is 3 and then since 4 is in conflict with 3, we remove 4. So, continuing in this way we now pick 5, because 5 is earliest one to finish and then because 7 is in conflict with 5, we remove 7.

And now we have two left, 8 and 9, but 8 finishes before 9, we could actually pick either one, but our algorithm will pick 8, because 8 has the shortest finishing time. So, we pick 8 and then we will say that 9 is not feasible, so we do not pick it and now we have the B is empty and A is 1, 3, 5, 8 and since B is empty, we have no more jobs to schedule, no more bookings to honor, so the algorithm ends. So, we have found a feasible set of four bookings that can be accommodated with in this list.

(Refer Slide Time: 10:46)

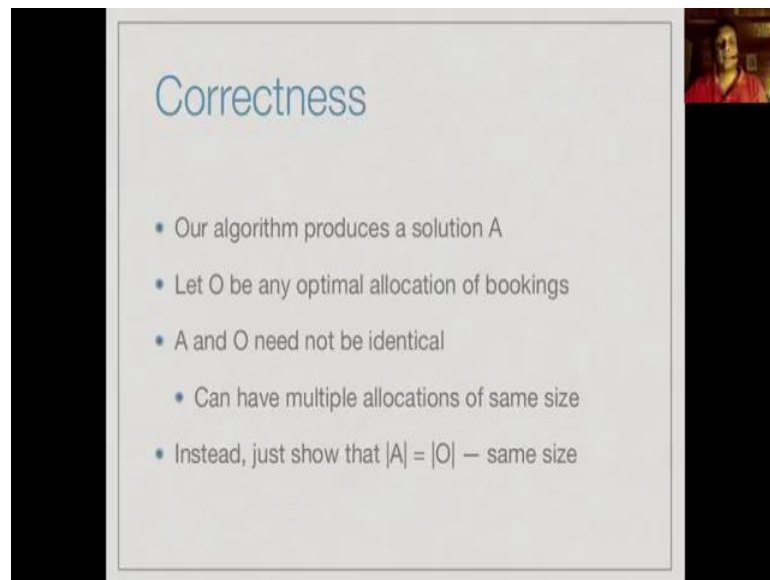


Correctness

- Our algorithm produces a solution A
- Let O be any optimal allocation of bookings
- A and O need not be identical
 - Can have multiple allocations of same size
- Instead, just show that $|A| = |O|$ — same size

So, our goal is to show that the algorithm, the solution A produce by our algorithm is actually correct. So, suppose there is an optimal set of bookings O, now we cannot in general assume that our solution A is identical to O, because there maybe multiple ways of producing solutions of the same size. Remember that all we want is a solution which allocates as many teachers as possible to rooms. So, there maybe two different ways to allocate the same number of teachers, so we cannot argue that A and O are identical, but it is surprises to show that A and O are of the same size. In other words, moment of what optimal booking is produced by some other strategy, our strategy our greedy strategy produces one which is of the same size.

(Refer Slide Time: 11:36)



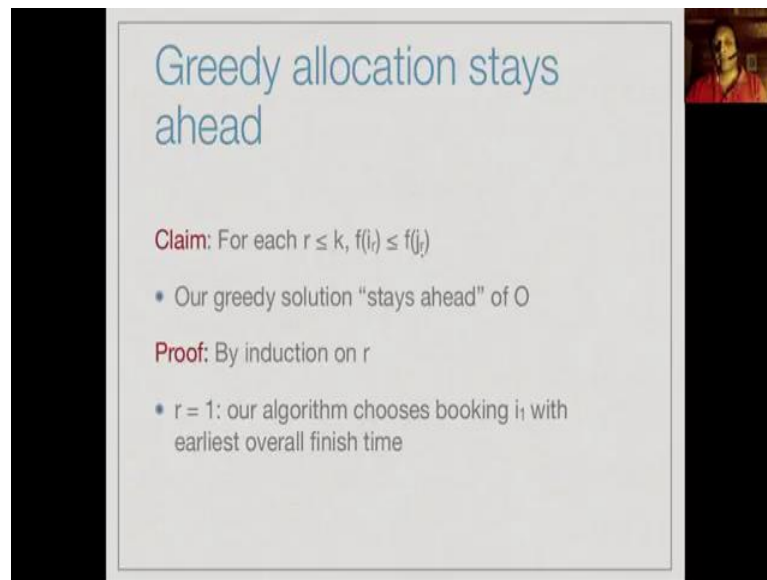
Correctness

- Our algorithm produces a solution A
- Let O be any optimal allocation of bookings
- A and O need not be identical
 - Can have multiple allocations of same size
- Instead, just show that $|A| = |O|$ — same size

So, let A be the set of bookings that our strategy chooses and this could be the order and which we chooses, so i_1 is chosen first and then i_2 and so on, so when i_1 is chosen and i_2 is still feasible and since i_1 was the earliest finishing time overall, we have that the finishing of i_1 is before the starting time of i_2 , the finishing time of i_2 is before the starting time of i_3 and so on.

So, these bookings in A are in sorted order, now let us assume that we had an optimum solution with m bookings j_1 to j_m again in sorted order. So, j_1 ends before j_2 starts, j_2 ends before j_3 starts and so on. So, our goal is to show that k in fact is the same as it, in other words the optimum solution is of the same size as the solution that the greedy strategy produces.

(Refer Slide Time: 12:30)



Greedy allocation stays ahead

Claim: For each $r \leq k$, $f(i_r) \leq f(j_r)$

- Our greedy solution "stays ahead" of O

Proof: By induction on r

- $r = 1$: our algorithm chooses booking i_1 with earliest overall finish time

So, we will actually show that for each job in the sequence i and j , the corresponding job in the A sequence finishes no later than the corresponding job in the O sequence. So, for every r up to k , f of i_r is earlier than or equal to f of j_r . So, in this sense we are trying to argue that the greedy solution stays ahead of any optimum solution, we may produce by any other method.

So, the proof of this claim is by induction on r . So, when we look at the first job i_1 , we know that i_1 is overall the earliest finish time among all the jobs, all the bookings in our list, since i_1 has the earliest finish time over all the bookings, it must definitely be less than or equal to f of j_1 , because j_1 cannot be smaller than the overall finish time.

(Refer Slide Time: 13:30)

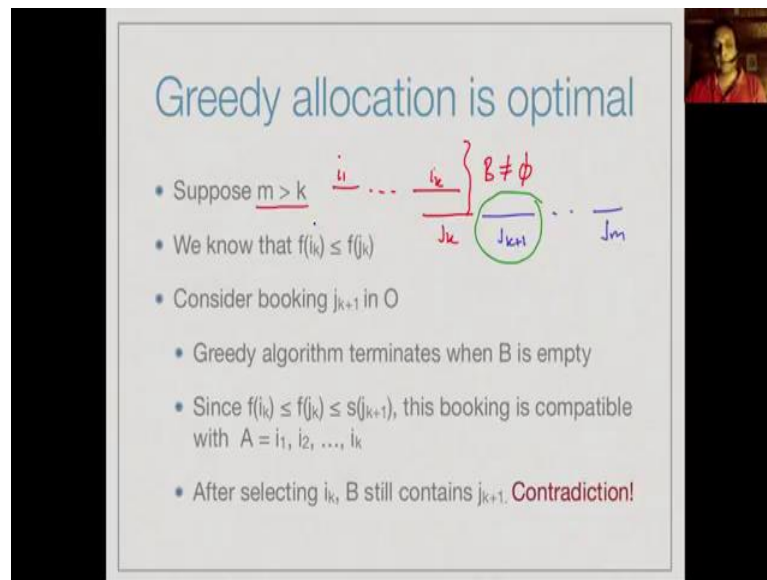
Greedy allocation stays ahead

- $r > 1$: Assume, by induction that $f(i_{r-1}) \leq f(j_{r-1})$
- Then, it must be the case that $f(i_r) \leq f(j_r)$
- If not, algorithm would choose j_r rather than i_r

Now, let us assume that we have established by induction that up to r minus 1, the booking i minus i of r minus 1 has a finish time which is earlier than booking j of r minus 1. Then, we claim it must be the case that i_r finishes before j_r , because if we did not have this then we would have the picture as below. So, we have that i_r minus 1 finishes before j_r minus 1.

Now, suppose we claim that j_r actually ends before i_r , then our algorithm would at this stage find the j_r is still feasible, because it does not overlap with i_r minus 1 and among the jobs which remain j_r has an earliest finishing time than i_r . So, our greedy strategy would pick j_r rather than i_r , so therefore the fact that we have picked i_r and not j_r means that we cannot have a picture like this. It cannot be that i_r ends strictly after j_r , it must end before or at the same time as j_r end.

(Refer Slide Time: 14:37)



Greedy allocation is optimal

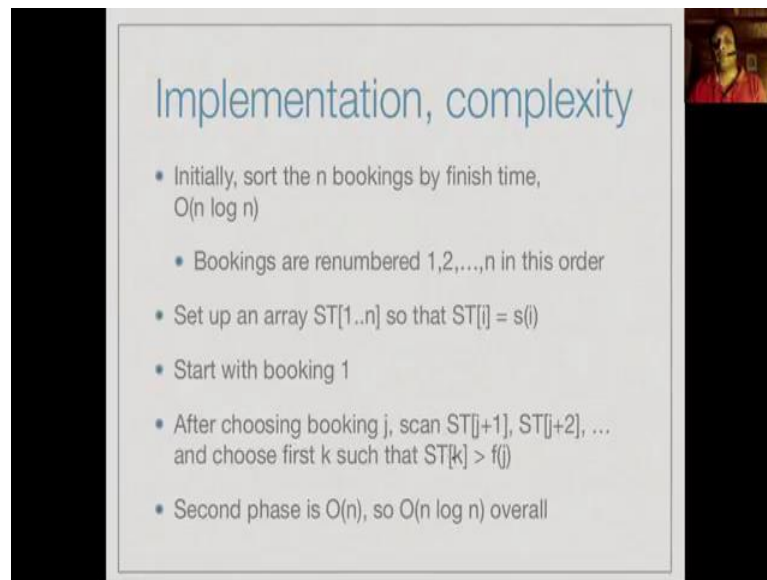
- Suppose $m > k$
- We know that $f(i_k) \leq f(j_k)$
- Consider booking j_{k+1} in O
 - Greedy algorithm terminates when B is empty
 - Since $f(i_k) \leq f(j_k) \leq s(j_{k+1})$, this booking is compatible with $A = i_1, i_2, \dots, i_k$
 - After selecting i_k , B still contains j_{k+1} . **Contradiction!**

So, now having shown that the greedy strategy always stays ahead, we will now claim that actually our solution must be optimum. So, suppose that m is actually strictly greater than k , then we know that when we reach i_k , it is before j_k . Now, because we have a solution which is longer than k , there is another job after this called j_{k+1} , assuming that m is strictly, because this goes up to job booking j_m . So, since this happens there must be a sequence of job, sequence of bookings after j_k , so let us look at the sequence.

Now, the claim is that this particular booking at this point is not ruled out by anything that is happened before, so if we look at i_1 up to i_k , none of these overlap with j_{k+1} , because j_{k+1} is after j_k . So, i_k finishes before j_k and j_k finishes before j_{k+1} starts. Therefore, i_k is compatible with j_{k+1} , this means at this stage B is not empty.

When we have to finish in our greedy algorithm processing i_1 to i_k these not empty, but we claim that we start with i_k and the only reason our greedy algorithm will stops is because B is empty. So, if there is a job or a booking j_{k+1} , then it cannot be that our algorithm stopped at this point, so there is a contradiction. So, therefore we cannot have any bookings in the optimum solution which go beyond k and therefore, m must be equal to k .

(Refer Slide Time: 16:24)



Implementation, complexity

- Initially, sort the n bookings by finish time, $O(n \log n)$
 - Bookings are renumbered $1, 2, \dots, n$ in this order
- Set up an array $ST[1..n]$ so that $ST[i] = s(i)$
- Start with booking 1
- After choosing booking j , scan $ST[j+1], ST[j+2], \dots$ and choose first k such that $ST[k] > f(j)$
- Second phase is $O(n)$, so $O(n \log n)$ overall

So, having shown that it is correct, let us just quickly look at how we would implement this and estimate the upper bound of the complexity. So, initially we sort the m bookings by finishing time, this takes time $n \log n$ for n bookings and now let us assume if the bookings are renumbered 1, 2 up to n in this sorted order. So, booking 1 has an earliest finishing time, booking 2 has a second earliest finishing time and so on.

Now, we set up in one order n scan, an array ST such that ST of i contains the starting time of booking array. Now, we start with booking 1 and each time we choose a booking j , we start from j plus 1 and keeps scanning the start time of bookings till we find the earliest k whose starting time is beyond f of j . In other words, we are looking... So, so we know that these bookings are in order of finishing time.

So, we know that after j the booking that ends next is j plus 1, but if it is starting time is not beyond the finishing time of j , it is overlapping, so it cannot be compatible. So, we just scan this array ST until we find the smallest k which actually starts after f j ends. So, in this way one order n scan we can go through all our bookings and pick up a greedy optimum set. So, this is an order n scan sorting takes order $n \log n$, so overall this greedy strategy is correct and it takes time $O(n \log n)$.