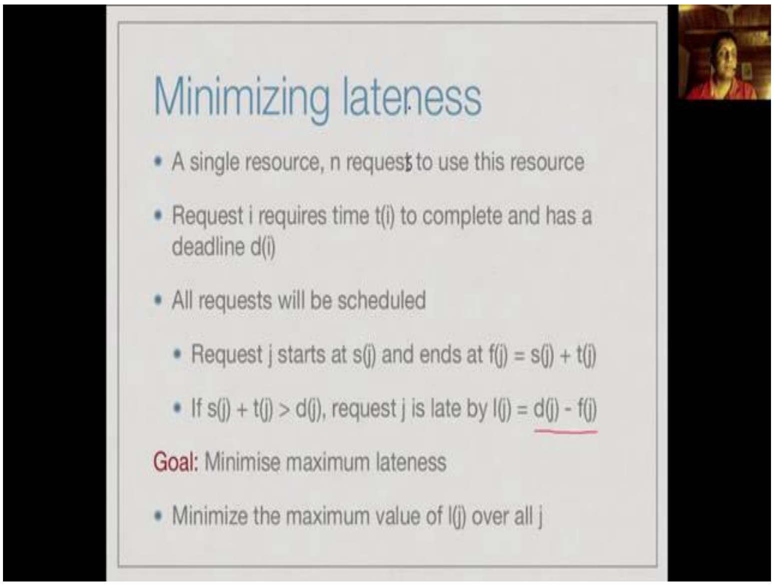
Design and Analysis of Algorithms
Prof. Madhavan Mukund
Chennai Mathematical Institute

Week - 06
Module - 04
Lecture - 42
Greedy Algorithms: Minimizing Lateness

We now look at a different Greedy Algorithm with a slightly more complicated proof of correctness. So, the problem we are looking at is called Minimizing Lateness.

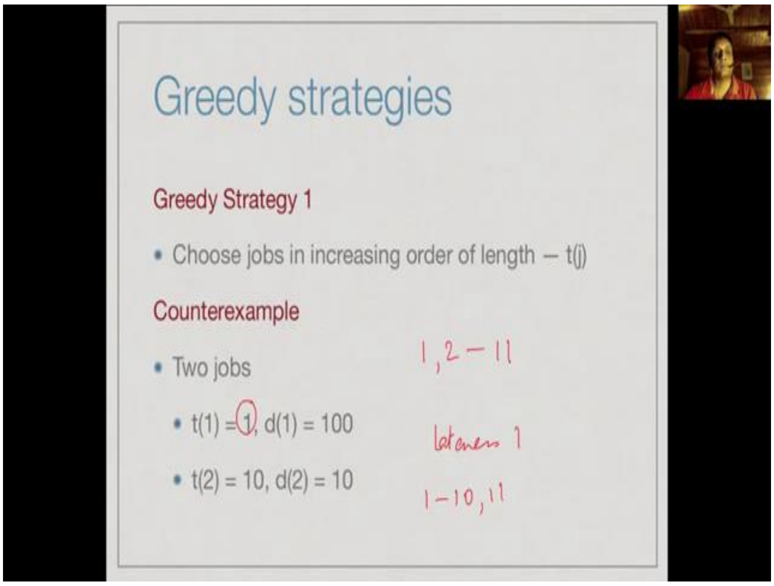
(Refer Slide Time: 00:12)



So, like our interval scheduling problem in the last example, we have a single resource and there are n request to use this resource. So, now unlike the earlier situation where we had a start time and a finish time and the resource had to be scheduled within this time. Here we just know that each request i requires time t of i to complete and each request i comes with a deadline d of i, here we are going to schedule every request.

So, each request j will started at time start of j, it is called s j and it will time take t j, so it will end at f of j, the finish time of j which will be the start time plus the time it takes to process request j. Now, if this finish time is bigger than the deadline, then it is late, so the amount that it is late is given by the difference between the delay and the finish time and the goal is to find a schedule which minimizes the maximum lateness. So, we want to minimize the maximum value of this l j over all the jobs j.

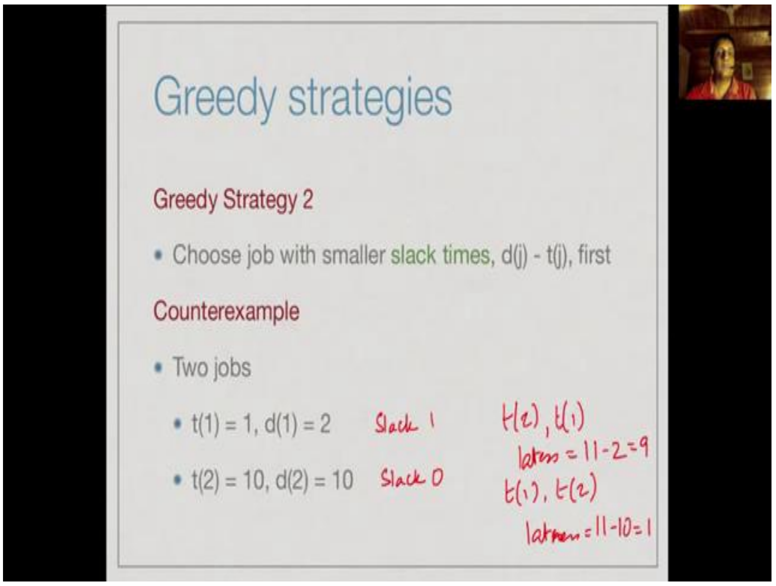
(Refer Slide Time: 01:28)



So, since we know we are looking for greedy strategies, let us try and suggest some greedy strategies for this problem. So, suppose we want to finish jobs as quickly as possible, so we choose a shortest job first, so we choose jobs in increasing order of length. So, this could be a greedy strategy, but unfortunately there is a fairly simple counter example. So, suppose we have 2 jobs job 1 takes 1 time unit and job 2 takes 10 time units, but the deadlines are 110 and 10 respectively.

In other words, the first job has a very long gap within which it can be scheduled without any penalty, whereas this second job has to finish more or less assuming it starts finishing. So, now if you pick this shortest job then we are going to incur a lateness of 1, because we are going to go from 1 and then we are going to go from 2 to the 11. So, the second job is going to finish 1 unit of time late, on the other hand if we do 1 to 10 then we do 11, then we get no lateness, we get lateness 0. So, here picking the shortest job first does not give us the best answer.

(Refer Slide Time: 02:44)

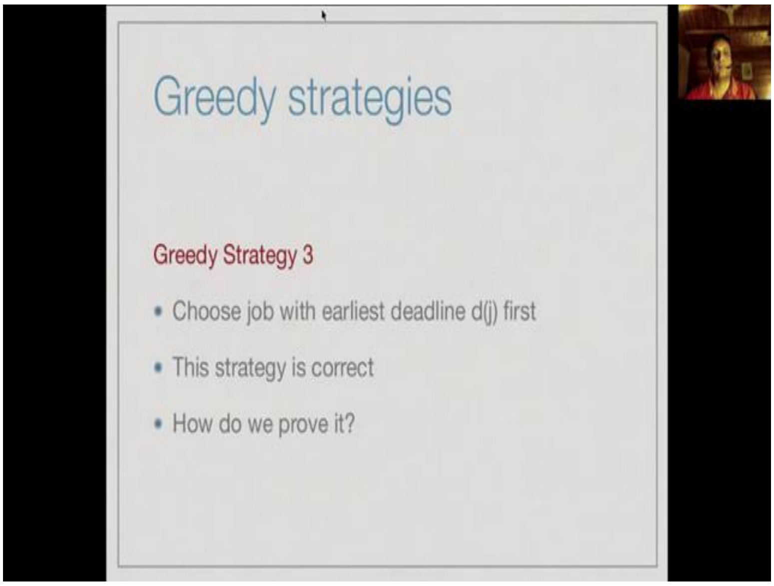


So, the second strategy might be to pick those jobs, so earlier we saw that we had a job which had 10 time units and it will also need it had a deadline of 10. So, we need to pick those jobs perhaps whose time is closes to the deadline. So, we look at the slack how much time we can effort to delay start in my job, d j minus t j and pick those which have the smallest slack. So, here we have a very similar example to the first one, except that the deadline of the first job which now to...

So, here we have slack 0 for the second job and slack 1 first job, so the second one has the deadline equal to it is time, the first one has the deadline which is one node that is at time. So, then by this strategy we would pick t 2 first and if you pick t 2 followed by t 1, then what happens is that this lateness is going to be 11 minus 2, because we first to t 2. So, we start t 1 a job 1 only a time 11, so it finishes the time 11, but it should have finish 2, so the lateness is 9.

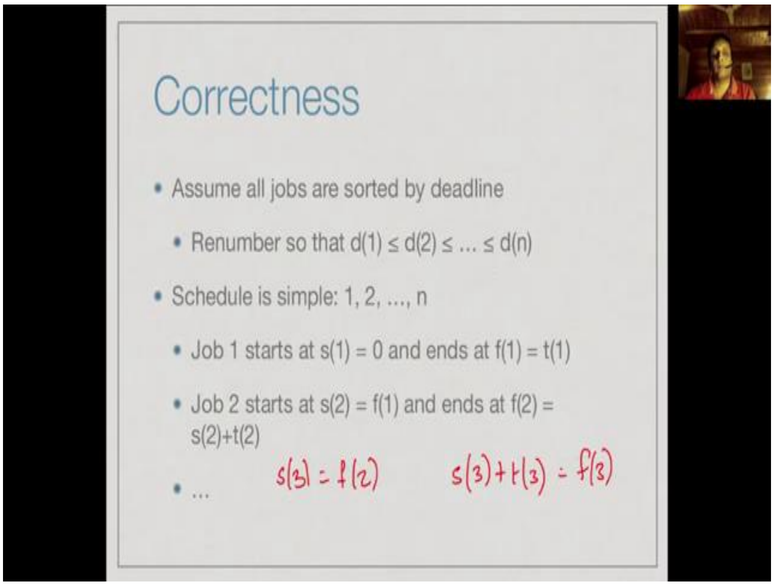
On the other hand, if we do t 1 followed by t 2, then we have that the lateness is just 1, because of the second job should have finished at 10 instead it finishes at 11. So, 11 ((Refer Time: 04:15)) is 1, so now here although our intuition told us to pick this smallest slack time actually that is not the good one.

(Refer Slide Time: 04:25)



So, turns out that a greedy strategy that does work is to choose the job with earliest deadline d of j first, the challenge is to proof that this strategy is in fact correct.

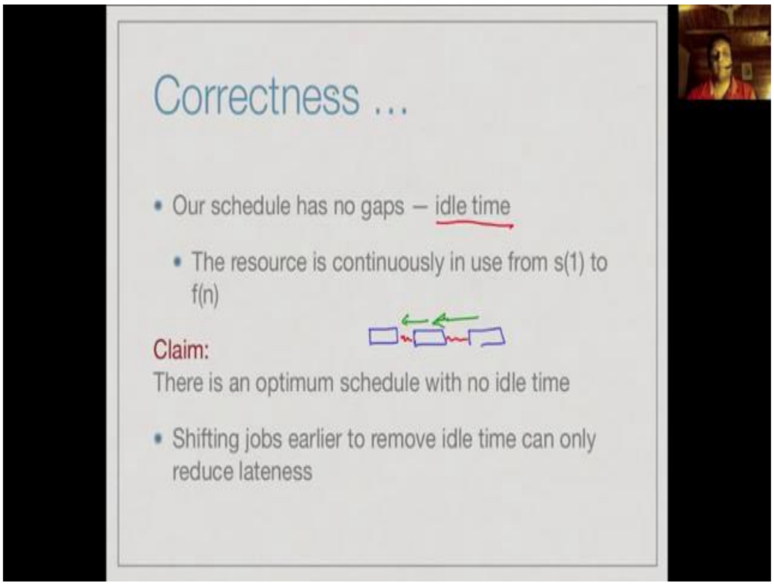
(Refer Slide Time: 04:40)



So, to proof that is correct we will first assume that we have actually numbered all our jobs in order of deadline. So, we number our jobs 1, 2, 3 up to n, so let that the deadline of 1 is less then or equal to deadline of 2 and so on. Now, having done this our schedule is very straight forward, we just schedule job 1 first, then job 2, then job 3 and so on. So, we do not have do anything, once we have shorted the jobs by deadline, we just schedule them in that order to the job 1 starts that time 0 which will call as set 1, it ends at f of 1 which is t of 1 0 plus t of 1.

Now, s 2 the starting time for job 2 is as soon as job 1 s, so at f 1 we start job 2 and it will end at s 2 plus t 2. So, likewise now s 3 will be f 2, we will start job 3 at time t 2 and we will go on to s 3 plus t 3 and call this f 3. So, we will just schedule each job as soon as the previous one ends in this deadline order.

(Refer Slide Time: 05:46)



So, since we have scheduling jobs one after the other without waiting, it is very clear that this schedule has no gaps, it has no idle time. The resource that we are trying to allocate is continuously in use, until all n requests are finished. So, now the claim is that there is an optimum schedule which has no idle time, because suppose you had an optimum schedule in which you have blocks like this where the resources is being used and there were these gapes in between which were idle.

Whereas, very clear that I can shift these things forward, look at this, there is no constraint on when I can skip to this, I only have a constraint on when thing should finish. So, when move things earlier I can only reduce the lateness, so if the blue schedule with gaps was optimal, I can move it, so that it does not have gaps and certainly my new schedule will have no more lateness in the blue one. Therefore, we can always assume that optimum schedule has no idle time.

(Refer Slide Time: 06:51)

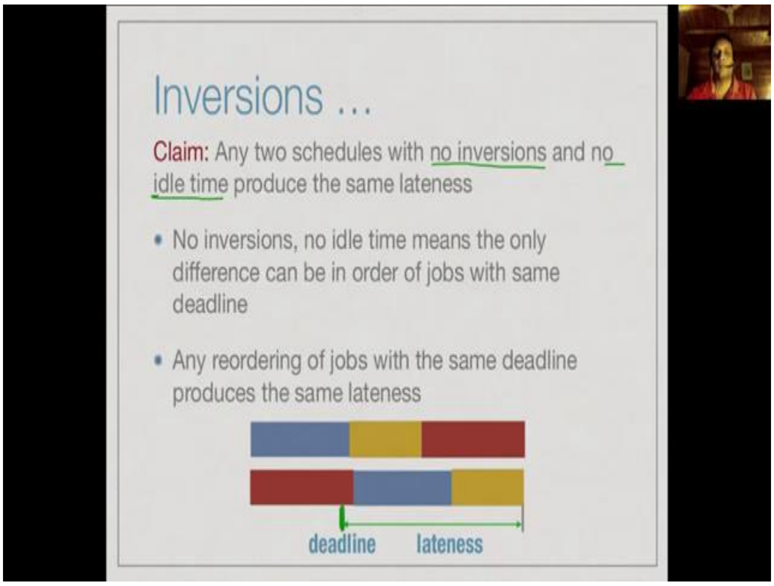
So, now our goal is to actually argue that this schedule that we have produced by sorting in terms of deadlines and then using that order blindly, this as could as any optimum schedule. So, here in the previous interval scheduling problem, we said that we would not be able to guarantee that schedule that we found is equal to a given optimum schedule, but we will just show that there are of same size. Now, here what we do slightly different, we will take an optimum schedule which is produced by some other strategy and we will step by step transform it into one that is the same as one that we have produced.

So, this is what is called an exchange argument, we start with some schedule and then we keep moving things around in that schedule preserving optimality, until eventually we transform the given schedule O into our schedule A, which would get among greedy strategy.

(Refer Slide Time: 07:52)

So, our strategy processes and schedules jobs in order of deadlines, so we can say that this schedule O, your optimum schedule has an inversion, if it actually has two jobs which appear out of order within deadline. So, there is a job i which appears before jobs j and O, but the deadline of j is strictly before the deadline of i. So, notice that our solution, because the greedy solution processes change in deadline order, there cannot be any inversions in our schedule, but the optimum schedule the arbitrary optimum schedule that we have presented with may well have inversions.

(Refer Slide Time: 08:37)



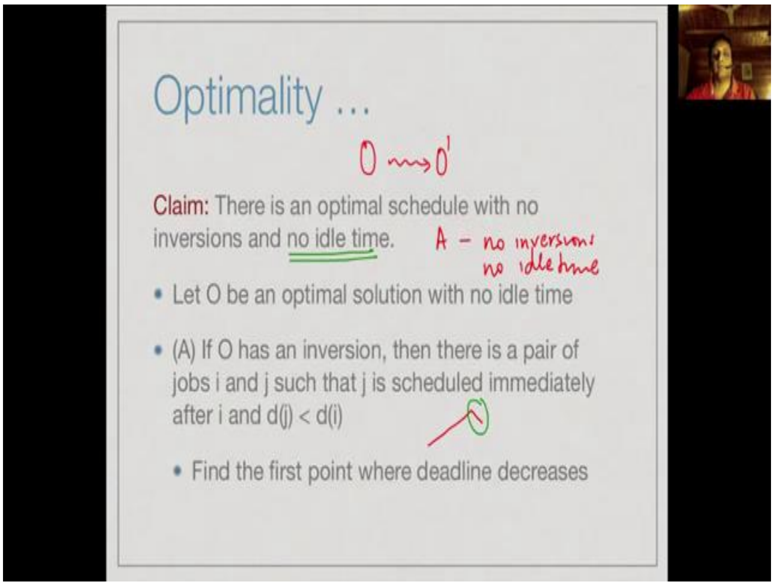
So, now the first point is that if you have no inversions and no idle time, then the lateness must be the same. So, first of all if you have no inversions and no idle time then the only flexibility we have is to reorder, because we not allowed to put things with later deadlines ahead of things with earlier deadlines. The only flexibility we have is to reorder the things with the same deadline, we may have multiple jobs within the same deadline and we could pick different sequential iterations or different reordering of these same deadlines, they would not validate inversions, because they are equal.

But, inversions happens only when we have something strictly smaller coming after something that is strictly become. So, the claim is that in such a situation, we cannot have a different answer, because of even if you allow our self to shuffle jobs in same deadline. So, here is a picture, so suppose these three jobs, the blue job, in the yellow job, the red job all have the same dead line. So, here is one sequence where we do blue first, then yellow, then red here is some another sequence, so we do red first, then blue then yellow.

Now, all have the same deadline, so the same deadline is at this point. So, deadline is here, now the last of these jobs regardless of how we shuffle them will end the same point. Because, we have the total the sum of the times and that will be the end and the last job will have among these, the maximum delay with respect to this deadline.

So, since we have counting the maximum lateness, the maximum lateness cannot change regardless of how I shuffle these jobs, which ever jobs ends will end at the same time, because all of these are of the same length or I mean the sum of the these are same length regardless of how I shuffle them. And therefore, the lateness does not change, so in somebody if you have two schedules which have more inversions and no idle time, then the answer in terms of the lateness we produce is this.

(Refer Slide Time: 10:41)



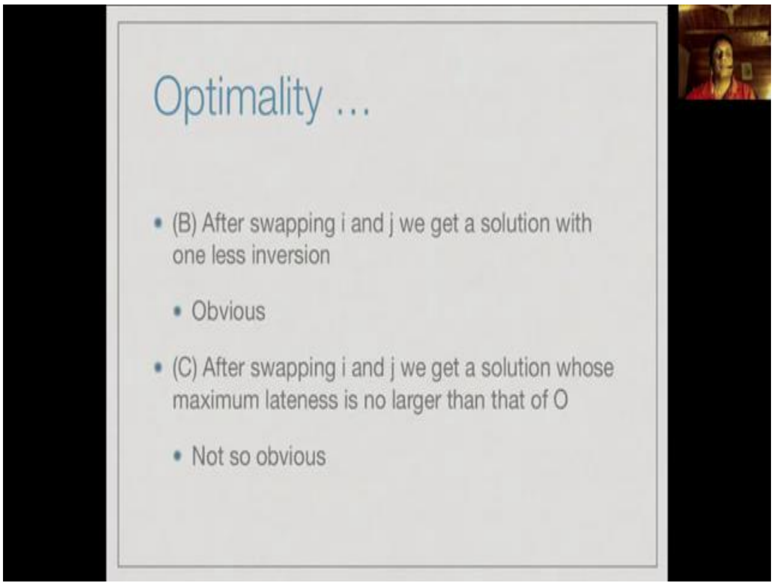
So, now if you can claim that there is an optimum schedule with no inversions or no idle time. Now, recall that our schedule A has this property; A has no inversions by construction and no idle time. And now I am going to claim that there is an optimum schedule, we going to start with O and we are going to produce from this some O prime which has no inversions, no idle time. And by the previous remark, since O prime and A both have no inversions no idle time, they must actually produce the same lateness.

So, how do we do this? So, first of all we know that we can assume that the optimum schedule has no idle time. Because, we already said that idle time is useless we can always shift anything left, compress out the gaps, so that there is no idle time. So, the first observation is that now we have no idle time, so one part of this requirement is assumed, so the only thing that we have to worry about is inversions.

So, the first claim is that if O has an inversion, then in fact we have an inversion among two consecutive elements, there is a pair of jobs i and j such that j is immediately after i, but the deadline of j is smaller than the deadline of i. So, we have something with a smaller deadline which comes later than something to the bigger deadline and this is very clear, because if there is an inversion, then the deadlines normally will keep increasing and then somewhere this is an inversion, so it comes down.

So, at the point where it comes down, we must have two adjacent things, where the bigger one comes before the smaller one. So, whenever we have an inversion anywhere in the sequence, we can find some point where two consecutive items have an inversion.

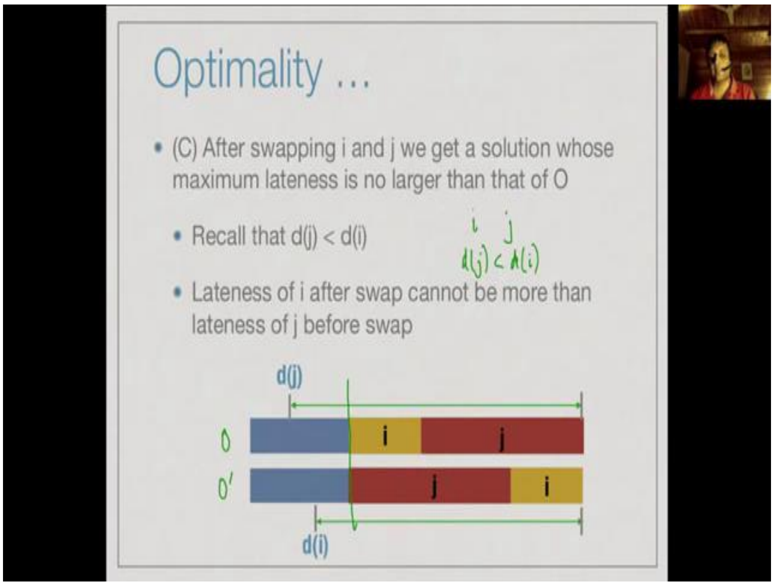
(Refer Slide Time: 12:37)



Now, the next observation is that we can remove this inversion by swapping these two jobs. So, we have i and j which has an inversion, then if we exchange i and j that is we put j before i, then now d of j is less than d of i and this inversion is gone. So, it is obvious that why we remove the inversion. But, what is now the obvious is that this operation of removing this inversion by swapping these consecutive jobs which are out of order will actually not affect the quality of this notion.

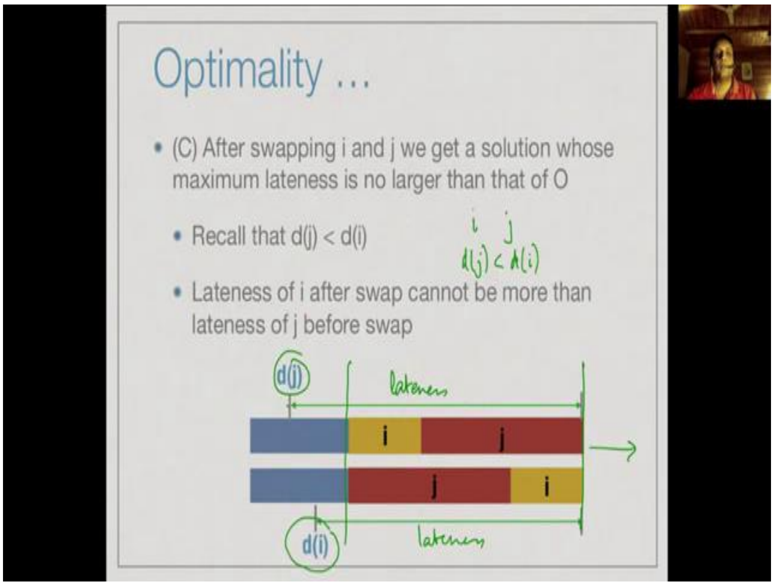
So, what we need to ask is whether after swapping i and j we get a solution whose maximum lateness is no larger than that of O. So, we have an optimum solution, we have an inversion and an adjacent consecutive inversion, we want to undo this conversion by swapping those two consecutive elements, but we do not want to change the optimality.

(Refer Slide Time: 13:38)



So, this can again we seen by a diagram, so remember that this inversion said that i came before j, but d of j was strictly less then d of i which is why it was an description. So, we have this kind of situation, so we had some history this blue history and then at this point we had i and then j. So, this is my original and now I am going to go from go to O prime by exchanging ((Refer Time: 14:08)).

(Refer Slide Time: 14:10)

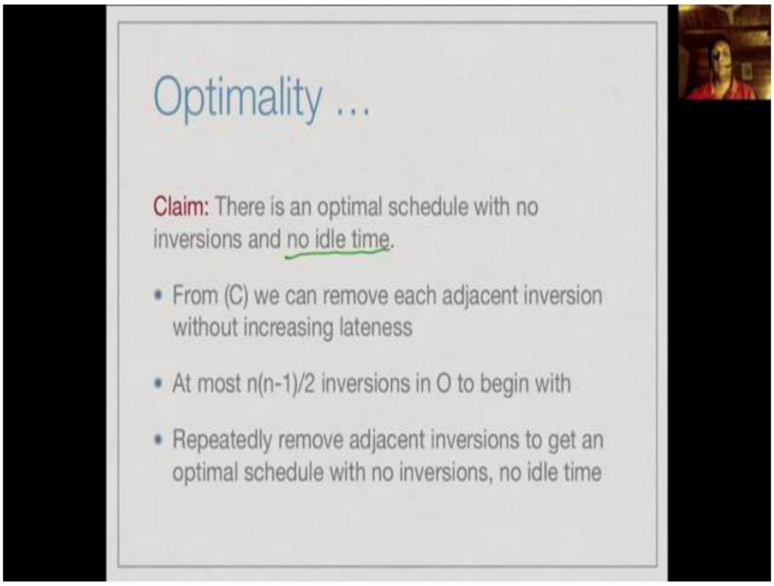


So, now observe that d of j is to the left of d of i by a assumption, so d of j strictly less then d of i. So, now let us look at the lateness of j, so it is from the ends i plus j take the same amount of time, whether I do i before j or j before i. So, if I look from the deadline of j up to where j n's then this length, this lateness must be more then the lateness below cannot be the less then. Because d of i is strictly to the right of d of j and the n point is the same.

So, if I look at the n point and subtract the deadline point, the deadline point for i is closure to the n, then the deadline point for j, because d j is before d 1. So, therefore by exchanging i and j not only have an a move on inversions have also guaranteed that because of this notice that late no other job, every other job up to this point answer to the same time when the every time which is the after this point also end the same point.

So, no other job has is lateness affected by the soft only to jobs who lateness changes or i and j by the change in such a way that the overall lateness then only reduce, it cannot increase. So, therefore this is the safe soft in terms of preserving optimality.

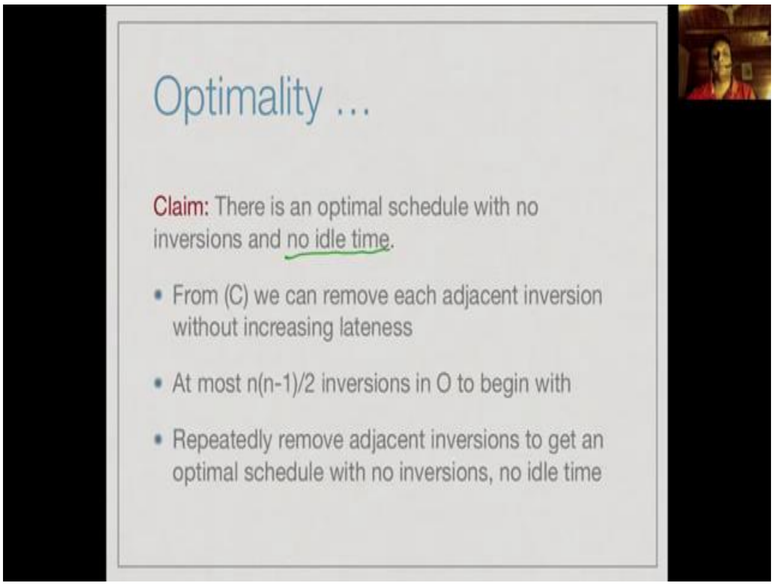
(Refer Slide Time: 15:34)



So, therefore now we come back to our claim that there is an optimum schedule with no inversions and no idle time, we know that we have an optimum schedule with no idle time, because that general principle. Now, from the previous argument we can remove every consecutive inversion without increasing lateness. Therefore, the optimality is preserved, now if you add n jobs even if a every pair of them is out of order, we have only n to n minus 2 1 by 2 inversions to begins with.

So, we can systematical inverts every one of them, without affecting optimality, until we get an optimum schedule with no inversions and no idle time. And we already saw that any two schedules is no inversions and no idle time must be equivalent terms of lateness. Therefore, our schedule A which has the property that it has no inversions and no idle time as the same lateness as this transformer version of O at since a transform version of O has the same lateness O itself and O is optimal our algorithm our solution A is also optimal.

(Refer Slide Time: 16:45)



So, the trick in this problem was to actually prove that the greedy strategies was correct, the implementation and the complexity are very easy to calculate, we just have to short the job is by deadline and then read out in this schedule in the same order. So, shorting the jobs takes n log in is usual and reading of this schedule just takes order n time, because we just we read out jobs 1 to n after they are shorted. So, overall we have and n log n algorithm.