

Programming, Data Structures and Algorithms
Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 55
Greedy Algorithms-Job Scheduling

So, today's lecture is about greedy algorithms, and specifically we will be studying one algorithm for a problem called job scheduling. The initial part of the presentation will present you the motivation for greedy algorithms by a very simple algorithm, very simple example. The example is the change making example. So, let us consider the following problem where for an example let us consider we have 832 rupees, and using as few notes and coins as possible, we should keep 832 rupees let us say in our purse. In general we know there are 9 denominations of rupees in the Indian currency. So, let us call these denominations d_1 to d_9 , indeed they take the values from 1000, 500, 100, 50, 20, 10, 5, 2 and 1.

(Refer Slide Time: 01:18)

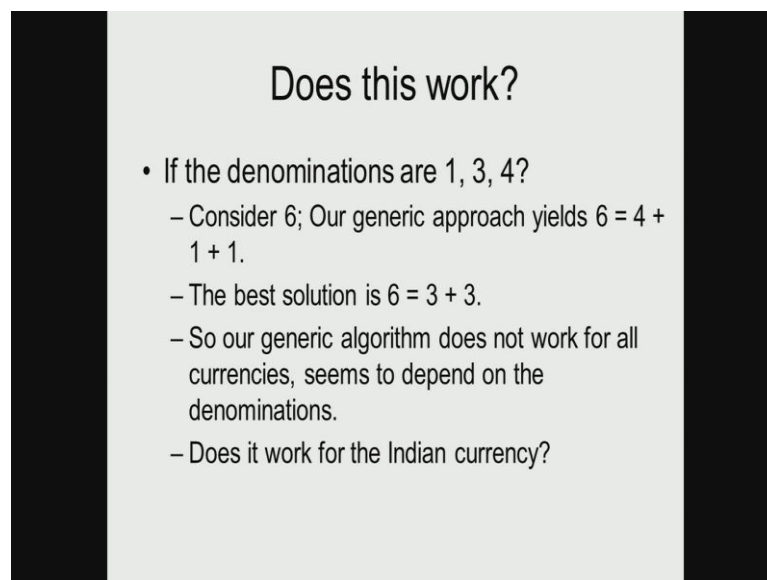
Change Problem

- Given a certain amount M Rupees, identify $x_1, x_2, x_3, \dots, x_9$
 - M should be the sum of $x_1d_1, x_2d_2, \dots, x_9d_9$
 - For example, $832 = 500 + 3*100 + 1*20 + 1*10 + s_2$
- What is a natural approach?
 - Iteratively, Select the large denomination smaller than the current amount

Naturally we wish to carry as few notes and coins as possible and the generic question is given M rupees identify 9 variables; these are integer valued variables, positive integer valued variables. Such that M rupees can be written as a sum of x_1 times, the first

denomination d_1 , x_2 times the second denomination d_2 so on up to x_9 times the ninth denomination d_9 . For example, 832 can be written as one 500 rupees plus three 100 rupees one 20 rupees, one 10 rupee, and one 2 rupee coin. In general do you have a natural approach here to solve this problem. The natural approach seems to be that iteratively we select the large largest denomination, which is available which is smaller than the current amount. Let us observe this by us the example itself 832 is the current amount in the largest denomination which is available to us is a 500 rupee note. So, we pick up 500 rupees and the balance that is to be expressed is 332. Now 332 the smallest the largest denominations smaller than 332 is a 100 rupee note, naturally we pick three times 100 th, we pick 100 once then the balance is 232, the smallest denomination large, the largest denominations smaller than this amount is another 100, and so on. So, we pick three times 300 rupees. Then we pick at 20 rupee note, because a balance that is to be express is 32 rupees and so on. So, the generic idea seems to be that we select the largest denomination that is available, such that it is smaller than the current amount that we need to express.

(Refer Slide Time: 03:10)



Does this work?

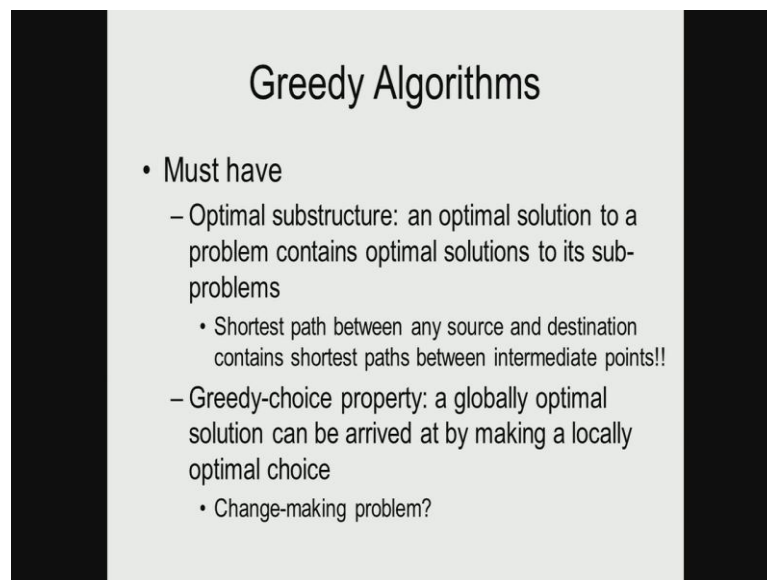
- If the denominations are 1, 3, 4?
 - Consider 6; Our generic approach yields $6 = 4 + 1 + 1$.
 - The best solution is $6 = 3 + 3$.
 - So our generic algorithm does not work for all currencies, seems to depend on the denominations.
 - Does it work for the Indian currency?

Does this always work independent of the denominations, this is an interesting question. Observe that we did our exercise with the Indian currency with a fix set of denominations, let us imagine whether this approach will work for any arbitrary set of

denominations in some currency. Let us assume that there is a country with currency right whose denominations are 1, 3 and 4 respectively, and there is no other denomination. Would our natural generic approach yield the optimum solution, let us see. Let us consider the simplest case of 6. So, if you apply our generic approach we will pick the largest denominations smaller than the amount which is 6, therefore we will pick 4 after which we are left with 2, which can be expressed as 1 plus 1.

So, our solution would be 4 plus 1 plus 1, on the other hand it is clear in this very simple example that the best denomination possible, the best representation of 6 possible in this currency is 3 plus 3. Therefore, it is clear that this generic algorithm which seems to be a very natural approach, it does not work for all currencies, it depends only the seems to depend on the denominations. At definite exercise which we will not address in this class is does this work for the Indian currency in if so why does it? We will not addresses, but this is a left as an exercise for the interested student.

(Refer Slide Time: 04:46)



Greedy Algorithms

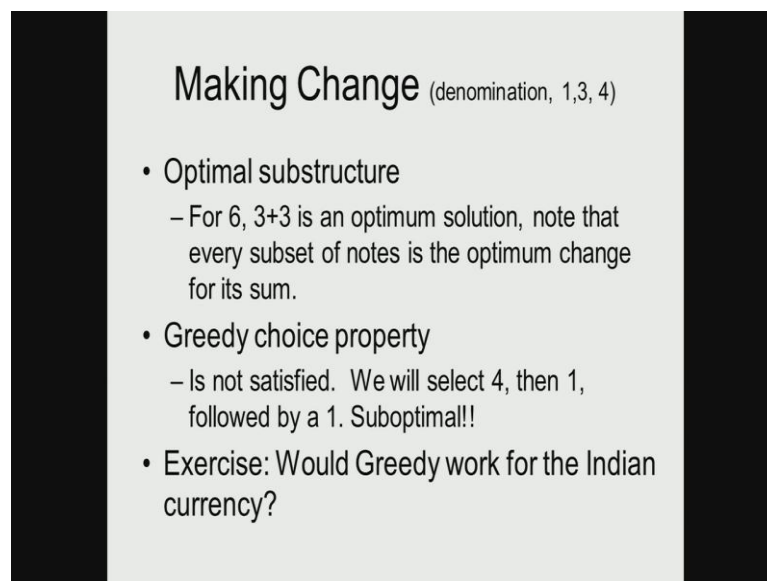
- Must have
 - Optimal substructure: an optimal solution to a problem contains optimal solutions to its sub-problems
 - Shortest path between any source and destination contains shortest paths between intermediate points!!
 - Greedy-choice property: a globally optimal solution can be arrived at by making a locally optimal choice
 - Change-making problem?

This takes us to this whole idea of greedy algorithms. So, observe that our algorithmic approach is a greedy approach, we choose the locally best possible choice. So, given a certain amount, we choose the largest possible denomination without exceeding the current amount. So, this is in some sense a greedy choice hoping that this will

eventually lead to an optimal solution. As we have seen it does not work for all currencies. So, therefore, greedy algorithms should have an optimal substructure. What is an optimal substructure? We should be able to guarantee the existence of an optimal solution to the problem, such that this optimal solution also contains optimal solutions to the subproblems. This is a very important concept, you have a problem and you also have subproblems of the problem. This is best illustrated by the shortest path problem in any network. Let us assume that the network is undirected, and let us solve the shortest path problem between a source and destination.

Observe that if you pick any shortest path between the source and destination, and if you pick any pair of vertices on the shortest path not necessarily, so source and destination, it is clear that the shortest path contains a shortest path between the intermediate points. So, very natural property of shortest path. So, this is the optimal substructure. Observe that no algorithmic choices here, it is just a property of an optimum solution. The greedy choice property is an algorithmic choice, it says that you can obtain a globally optimum solution by making locally optimal choices. We have seen this example of the change making problem, and we wonder whether it has a greedy choice property.

(Refer Slide Time: 06:47)



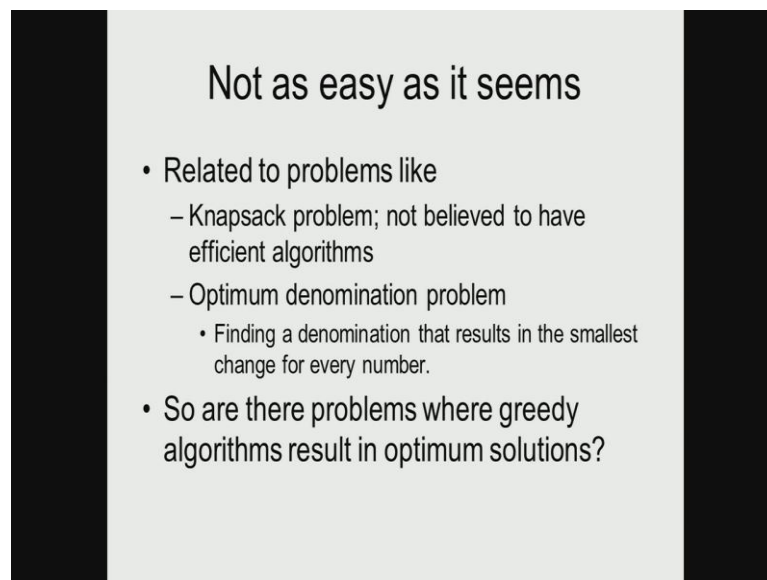
Making Change (denomination, 1,3, 4)

- Optimal substructure
 - For 6, 3+3 is an optimum solution, note that every subset of notes is the optimum change for its sum.
- Greedy choice property
 - Is not satisfied. We will select 4, then 1, followed by a 1. Suboptimal!!
- Exercise: Would Greedy work for the Indian currency?

We have already observed that if the denomination is denominations are 1, 3 and 4

respectively, if you take an optimum solution. Let us take the optimum solution for 6 in this example, which is 3 plus 3, note that every sub set of notes is the optimum change for it some. So, in this case every sub set is just the note 3 itself, which is optimum change for itself. So, this is easy. The greedy choice on the other hand is not satisfied as we have already seen we will select 4, then we selected one followed by a one in this turned out to be suboptimal. So, we repeat this exercise would the greedy choice work for the Indian currency. Indeed in this slide we have observed that the optimal substructures seems to work for any currency.

(Refer Slide Time: 07:51)



Not as easy as it seems

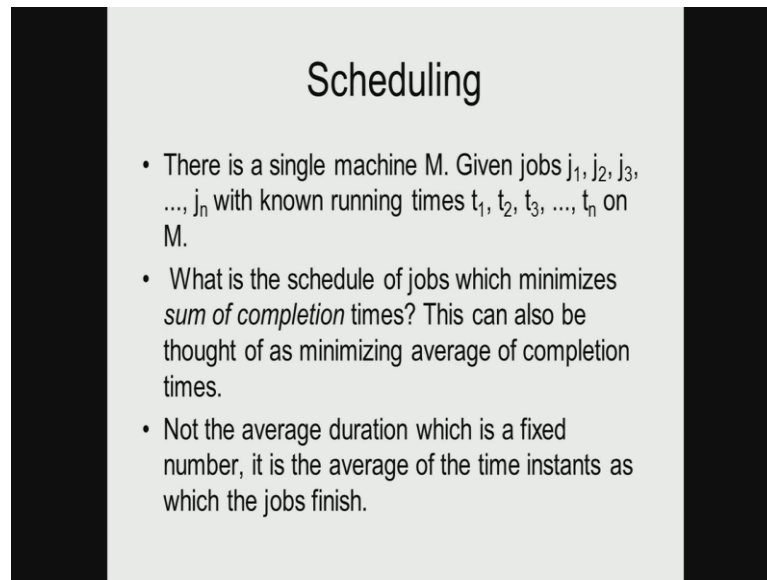
- Related to problems like
 - Knapsack problem; not believed to have efficient algorithms
 - Optimum denomination problem
 - Finding a denomination that results in the smallest change for every number.
- So are there problems where greedy algorithms result in optimum solutions?

However, like I said earlier we will not address it in this presentation, because the problem was not as easy as it seems, it is very easy to state it, but it is very closely related to problems, which after many years of research have not yielded to give efficient yielded to efficient algorithms for example, there is a problem called the knapsack problem. It does not have efficient algorithms to date it is believed not to have efficient algorithms, there is also the optimum denomination problem which is actually a problem face by currency designers. Where the question is what is the best denomination to ensure that for every number you get the smallest possible change by our approach.

So, therefore, let us ask as slightly different question, other problems for which greedy

algorithms result in optimum solutions, this is the focus. The change making example was a natural example to visualize the greedy algorithm scenario.

(Refer Slide Time: 08:42)



Scheduling

- There is a single machine M . Given jobs $j_1, j_2, j_3, \dots, j_n$ with known running times $t_1, t_2, t_3, \dots, t_n$ on M .
- What is the schedule of jobs which minimizes *sum of completion times*? This can also be thought of as minimizing average of completion times.
- Not the average duration which is a fixed number, it is the average of the time instants as which the jobs finish.

Let us look at the problem, which is very well in call, very well known as a scheduling problem. There is a single machine, and there are n jobs; each of these jobs can run only on this single machine, and should be run on the single machine, and the jobs are already known to have running times t_1, t_2, \dots, t_n respectively. In other words the job j_i takes t_i units of time on the machine M . Their aim is the following, the aim is identify a schedule of jobs, that is the order in which the jobs will execute in the machine. So, that we minimize the sum of the completion times.

Now minimizing the sum of completion times can also be thought of as minimizing the average of the all the completion times, it is very important for the student note that this average duration, this is not the average duration which is a fixed number, it is the average time that a machine spends waiting for the average time a jobs spends waiting for a machine, and then the time that it spends on the machine itself, one once to minimize this quantity. Therefore, the goal of this exercise is to find an ordering of jobs to execute on a machine. So, that we minimizes the sum of completion times.

(Refer Slide Time: 10:03)

Scheduling

- Consider this example
- 4 jobs and average duration is $36/4 = 9$ units of time

Job	Time
j_1	15
j_2	8
j_3	3
j_4	10

Let us do the small example; there are 4 jobs; job one takes 15 units of time, job 2 takes 8 units of time, job 3 takes 3 units of time, job 4 takes 10 units of time. The total duration that the machine will spend executing these jobs, clearly is 15 plus 8 23 units plus 3 26 plus 10 36 unit of time. Therefore, the average duration on the machine seems to be 9 units of time is 9 units of time.

(Refer Slide Time: 10:38)

Scheduling

Average completion time = $(15+23+26+36)/4 = 25$

Average completion time = $(3+11+21+36)/4 = 17.75$

However, the completion time is a completely different entity as shown by this graphic. Consider the first schedule in this graphic, where job 1, job 2, job 3, and job 4 are scheduled; job 1 finishes after 15 units of time on the machine, job 2 then finishes after 8 units of time, therefore the completion time of job 2 is 23, job 3 which actually spends a least amount of time on the machine, which is 3 units of time completes at time instant 26 and job 4 completes a 36 as a time ((Refer Time: 11:10)). Therefore, now if you average these completion times, we see that the average is 25, that is 100 units of time is spent, there is sum of the completion times is 100. On the other hand, let us consider the second schedule, where interestingly the shortest job is schedule first, then the second shortest job, then the third shortest job, then the fourth shortest job which is the order, job 3, job 2, job 4, and job 1. Now observe that the average completion time is 17.75 units of time, which is smaller significantly than the schedule where the jobs where scheduled according to the order in which they were presented in the input. So, now do you have an algorithm here to minimize a sum of completion times. To do this let us write down the formula for the sum of the completion times.

(Refer Slide Time: 12:14)

Scheduling

- Greedy-choice property: A schedule cannot be worsened by exchanging the position of the shortest job with the first
 - In our example, the jobs before j_3 will complete 3 time units faster, but j_3 will be postponed by time to complete all jobs before it.
- Optimal substructure: if shortest job is removed from an optimal solution, remaining solution for $n-1$ jobs is optimal

Before that let us look at the schedule. Let us look at properties of the schedule before we do the calculation for the sum of the completion times, let us look at a property of a schedule. One observation that we can make is that if we taken arbitrary schedule and

exchange the position of a shortest job in particular let us say the first shortest job in the schedule with the first job in the schedule. So, let us look at our example in the first job, if we exchange the jobs j_3 with a first job observe that we will get a better schedule, then the one that was given first. So, if we assume that there is always an optimal schedule where the first job is a shortest job, then it is clear that there is a very interesting optimal substructure that if you remove the shortest job, the remaining schedule for the remaining jobs is indeed optimal. In other words, if you take a optimal schedule in which the shortest job is schedule first by removing that shortest job. The remaining schedule is indeed an optimum schedule for the remaining N minus 1. Of course, we do not know if there is an optimal schedule which contains the shortest job first, that is what we are going to study now by writing down a formula for this sum of the completion times.

(Refer Slide Time: 13:42)

Sum of completion times

- Total cost of a schedule $\sigma = \sigma(1), \dots, \sigma(n)$ is

$$\sum_{k=1}^N (N-k+1)t_{\sigma(k)} =$$

$$t_{\sigma(1)} + (t_{\sigma(1)} + t_{\sigma(2)}) + (t_{\sigma(1)} + t_{\sigma(2)} + t_{\sigma(3)}) \dots$$

$$(t_{\sigma(1)} + t_{\sigma(2)} + \dots + t_{\sigma(n)})$$

$$= (N+1) \sum_{k=1}^N t_{\sigma(k)} - \sum_{k=1}^N k t_{\sigma(k)}$$

- First term independent of ordering, as second term increases, total cost becomes smaller

So, do this let us assume that there is a schedule which we refer to using the Greek letter sigma, and let us assume that sigma 1 denote the first job and sigma N denote the Nth job in the sequence. Now, let us look at the sum of the completion times, the sum of the completion times is return as a first formula where there are N jobs, and let us observe that the time taken by the first job, that is the job sigma of 1 that will be counted it will delay every sub sequent job, apart from using $t_{\sigma(1)}$ units of time on the machine. In the first job that we schedule which is sigma of 1 takes $t_{\sigma(1)}$ units of time on

the machine, not only there it also delays remaining $n - 1$ jobs by t_{σ}^1 units of time, same for t_{σ}^2 it is a second job - its completion time is the time there it spent waiting for the machine which is t_{σ}^1 plus a time that it spends processing on the machine which is t_{σ}^2 . In general if you look at the whole expression for the sum of completion times, the formula is given there, it is N terms in the summation and there are $n - k + 1$ copies of t_{σ}^k .

And that is explained in the equality there which says that the completion time is t_{σ}^1 plus the completion time of the second job which is $t_{\sigma}^1 + t_{\sigma}^2$ plus a completion time of the third job which is $t_{\sigma}^1 + t_{\sigma}^2 + t_{\sigma}^3$ and so on up to the completion time of the n th job which is $t_{\sigma}^1 + t_{\sigma}^2 + \dots + t_{\sigma}^N$. If I rewrite the summation by adding and subtracting a few terms, we get the third term in the whole equality sequence which is viewed as 2 summations. Observe that the first summation which is $N + 1$ multiplied by summation of the processing times of the N jobs, that is the first term minus summation k times t_{σ}^k that is the k th job the processing time of the k th job. Observe that the first term is independent of the schedule and the second term really is very dependent of the schedule. Observe that this sum of completion times is indeed valid for every schedule σ , this is a very important thing.

So, what we have done in this slide is to write down the close form expression for the sum of completion times, for an arbitrary schedule which we have called σ . And then we have observed that this sum of completion times can be viewed as 2 summations; one that is independent of the schedule itself and the second one which is dependent of the schedule. And there is a subtraction term there, therefore as a second term increases the total cost becomes smaller. So, let us see what makes a second term to decrease? Let us see what makes the second term to actually increase, and therefore reduces sum of completion terms completion times.

(Refer Slide Time: 17:21)

Optimal Schedule

Suppose there is a job ordering such that $x > y$
and $t_{\sigma(x)} < t_{\sigma(y)}$

*Swapping jobs (smaller duration first) increases
second term decreasing total cost*

We now show that $xt_{\sigma(x)} + yt_{\sigma(y)} < yt_{\sigma(x)} + xt_{\sigma(y)}$

$$\begin{aligned} xt_{\sigma(x)} + yt_{\sigma(y)} &= xt_{\sigma(x)} + yt_{\sigma(x)} + y(t_{\sigma(y)} - t_{\sigma(x)}) \\ &< xt_{\sigma(x)} + yt_{\sigma(x)} + x(t_{\sigma(y)} - t_{\sigma(x)}) \quad \text{Since } x > y \\ &= yt_{\sigma(x)} + xt_{\sigma(y)} \end{aligned}$$

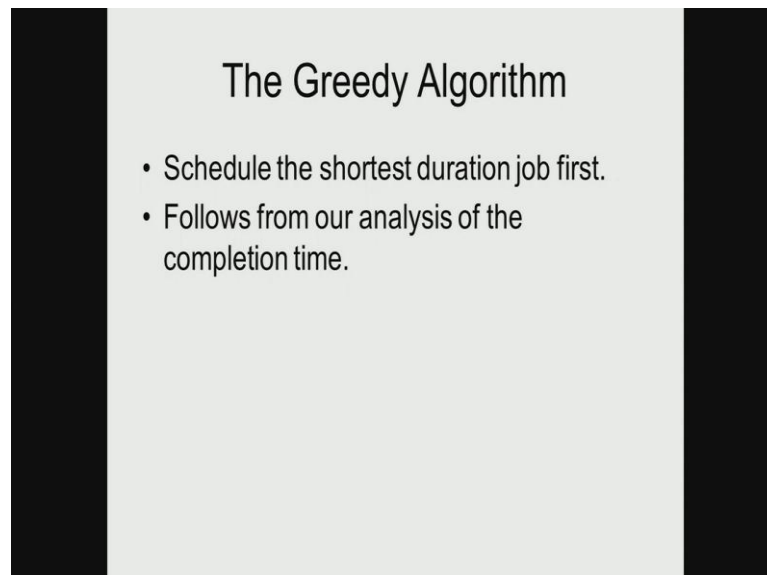
So, let us see a property of an optimal schedule. So, let us imagine a schedule sigma, and let us assume that there is an index x which is more than an index y, but the processing time for the job which is scheduled as sigma of x is smaller than the processing time of the job schedules sigma of y. In other words y is scheduled sigma of y is later than sigma of x, but the processing time of sigma of y is larger than the processing time of sigma of x. So, let us try the most natural thing, let us exchange the positions of the 2 jobs. So, in other words, what do we do we exchange the position of the job in position sigma of y with the position of the job sigma of x. And let us see how would changes the duration. So, let us see what happens, we should just go back to the formula and observe that the terms just change by modifying the multipliers appropriately. In other words, we have written down in inequality which says that x times t of sigma of x plus y times t of sigma of y, this is the contribution to the second term by these 2 jobs, and then we may exchange the position of these 2 jobs. The contribution just becomes y times t of sigma of x plus x times t of sigma of y.

We now show that x times t of sigma of x plus y times t of sigma of y is smaller than the summation after swapping it, that is y times t of sigma of x plus x times t of sigma of y. This is very easy to see by the following sequence of expressions by just rewriting x times t x times t subscript sigma of x plus y times t subscript sigma of y, this sequence of

expressions actually shows that the exchange actually increases the value of the second term.

As a consequence of this, it is clear that if we start off with an arbitrary ordering and if we identify a pair of jobs at sigma of x and sigma of y with the property that sigma of y is later than sigma of x, and the time taken by the job schedule that sigma of y is more than the time taken by the job scheduled at sigma of x. If we exchange these 2 jobs. The contribution to the second term in the sum of completion times reduces, and therefore we have a new schedule whose sum of completion times is strictly smaller.

(Refer Slide Time: 20:45)



The Greedy Algorithm

- Schedule the shortest duration job first.
- Follows from our analysis of the completion time.

Consequently the greedy algorithm is a very simple algorithm it says that from the given set of jobs schedule the shortest duration job first. That it is indeed an optimal algorithm follows from our analysis of the completion time that this schedule has a smaller completion time than any other schedule.