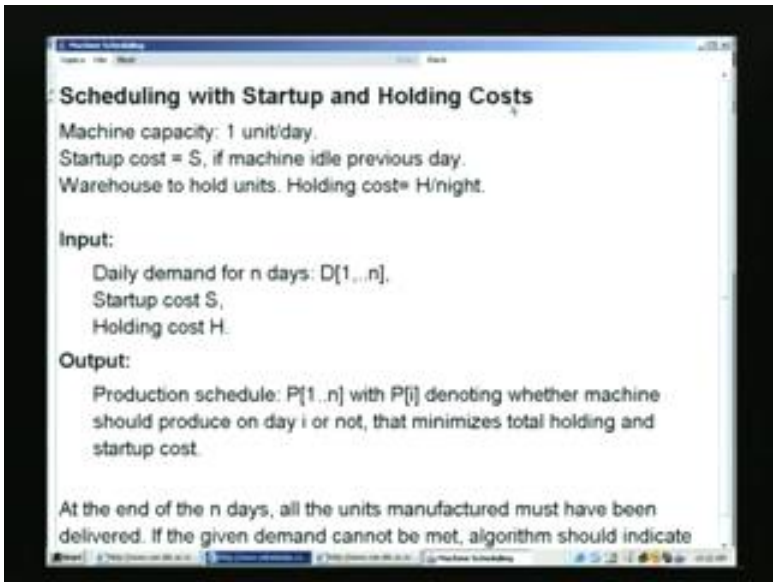
Design and Analysis of Algorithms
Prof. Abhiram Ranade
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 21
Scheduling with Startup and Holding Costs

Welcome to another lecture on design and analysis of algorithms. We will see 1 more problem which can be solved nicely using dynamic programming today. The problem we are going to solve is scheduling with startup and holding costs. Let me define this problem for you.

(Refer Slide Time: 01:10)

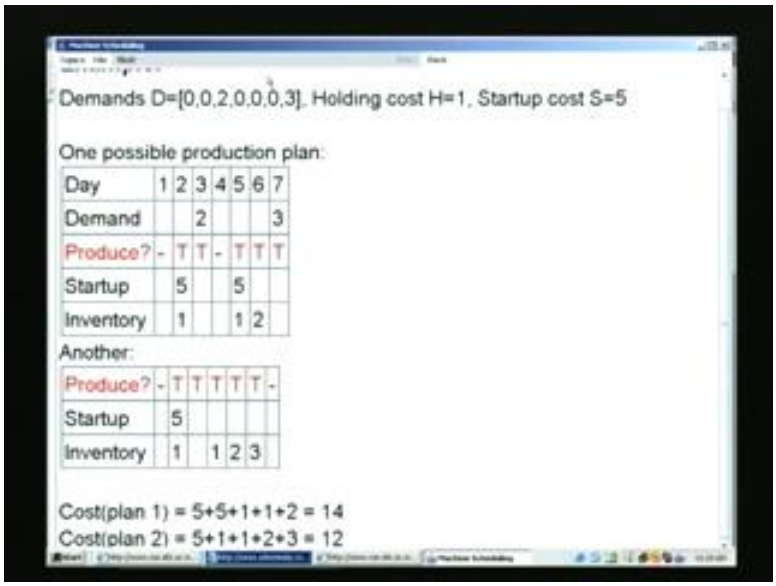


So, in this problem we are given a machine which has the capacity of producing 1 unit of something whatever it is per day. However, if I want to start a machine a machine on some day which is so, what I mean by that is that if the machine was not on today. And I want to get it working today then I have to pay a startup cost and that startup cost is some S units which is defined as the part of the input. Clarifying again if the machine was on yesterday then I do not pay a startup cost today if I want to keep the machine working. If the machine was off yesterday, but I want to start it today then I will have to pay a startup cost. We also have a warehouse to hold units which have been produced. So, if I produce something today and it is not to be delivered today itself then it goes to a warehouse. But of course, if I place something in the warehouse I have to pay some rent.

So, let us say that we will call that the holding cost and that holding cost will be some H rupees per night. The main input to the problem consists of a daily demand for n days. So, we are given the vector D of 1 through n in which D of I represents the demand for the ith day.

We are also given the startup cost S the value of S and the value of H the holding cost and what we are supposed to produce is a schedule. So, we are supposed to produce a vector P of 1 through n in which p of I denotes whether or not the machine is to be kept on the ith day or in other words whether the machine should be producing anything on the ith day. So, the final requirement which is which might be obvious perhaps, but let me state it nevertheless is at the end of n days all the units that have been manufactured must have been delivered. So, we cannot be left with any inventory at the end of n days and of course, it is possible that the demand that is being given to us is just not satisfiable. Because after all our machine can produce only 1 unit per day. So, in n days the machine can produce n units. So, certainly if the total demand is more than n we will not be able to meet. There will be other conditions under which the demand could not be met, but whatever these conditions are our algorithm must report if the given demand cannot be met.

(Refer Slide Time: 04:01)



So, let me now take an example to illustrate this problem better. So, in this example this is the demand vector that is given to us. So, what this means is on day 0, day 1 nothing

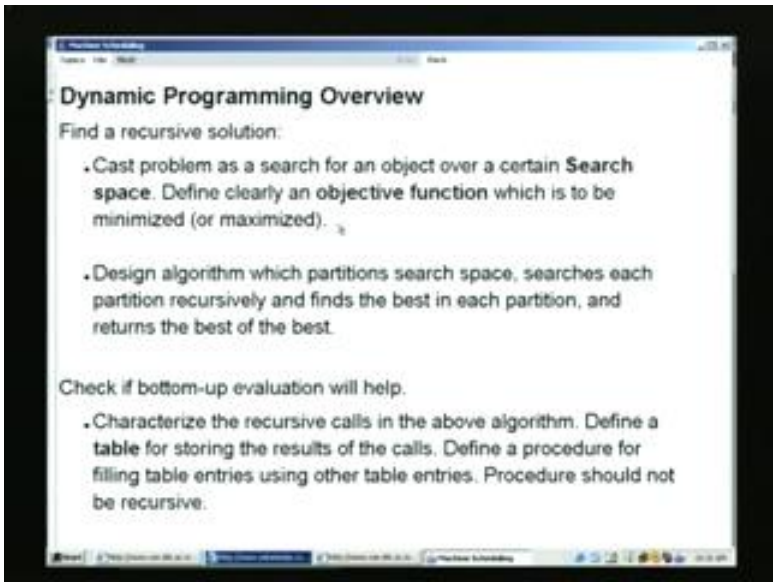
gets to be delivered nothing needs to be delivered, day 2 nothing needs to be delivered. On day 3 there is a order of 2 units which has to be delivered day 4, day 5, day 6 nothing has to be delivered and on day 7 there is an order of 3 units. The holding cost that is the cost of storing 1 unit in our warehouse is 1. So, it is 1 per unit and the startup cost is five. So, here is 1 possible schedule or 1 possible production plan which will meet this. So, I have shown this in a tabular form over here. So, we have days on the x axis. So, 1 through 7 since there are 7 days the demand is given as 7 days on the third day there is a demand of 2 as specified over here. And on the seventh day also there is demand and in this case the demand is 3. Here is a production plan which will meet this demand. So, notice that on this day 2 units have to be delivered and on any day we can deliver only 1 unit. So, which means we have to start producing earlier.

So, say in this case we have to start producing on day 2. So, we will produce on these 2 days and then for to meet for this demand we will produce on these 3 days machine will be idle on this day and not this day. Since the machine is going to start having the idle on the previous day there will be a startup cost on day 2. So, this will be a cost of 5 similarly on this day the machine will have to start again been idle on this day. So, there will be another startup cost on this day, day 2 whatever has been produced is not going to be delivered. It is only going to be delivered the next day. So, it will have to be held in the warehouse and. So, there will be a inventory cost of 1. Similarly, on this day whatever has been produced will have to be held in the warehouse whatever is been produced in the warehouse on this day will also be held. So, at this point there will be 2 units in the warehouse. So, holding cost of 2 will be incurred here this is of course, not the only plan here is another.

So, in this case what we have done is instead of keeping a gap on day 4. We have begun the production earlier, but notice if we begin the production earlier then we have our holding cost here as well. So, infact, the 4 dates we have to hold in our warehouse whatever unit has been produced. On the fifth day whatever has been produced has also to be held in addition to whatever have produced yesterday the day before that. And finally, whatever is produced in the sixth day there have to be held and all these 3 units will have to be delivered on the seventh day which of these is better. Well, let us calculate the time taken and the cost of each the cost of the first plan that is simply the startup cost which is 5 plus 5 here plus 1 plus 2. So, this cost for the first plan is 14 the

cost for this plan is 5 startup 5 for the startups and then 1 plus 1 plus 2 plus 3 which is written over here and this adds up to 12. So, eventually this plan is better our question in general is going to be to consider all such possible plans and figure out which is the best 1.

(Refer Slide Time: 07:56)



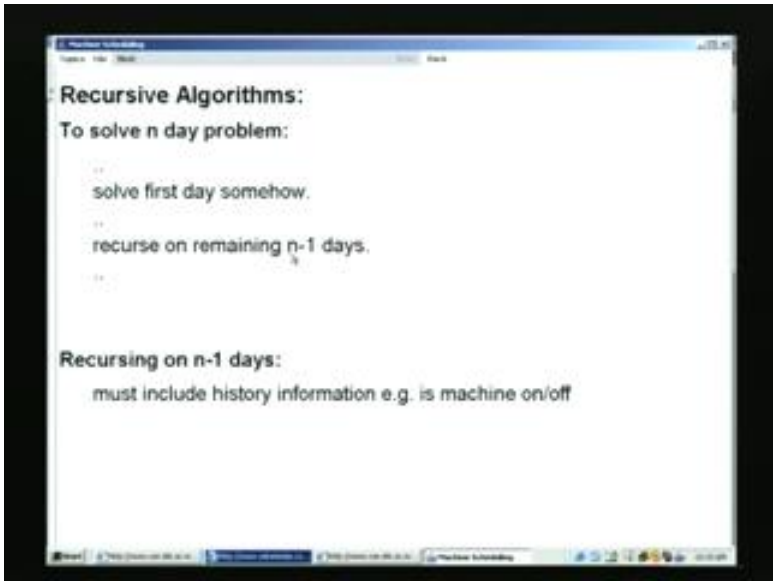
So, this naturally suggests a brute force algorithm. So, we will generate all possible plan for each will evaluate the costs and then will pick the best. Unfortunately the number of such schedules or such plans is going to be exponentially n where n is the number of days. So, this is going to be just slow and we would like to have a faster algorithm this is where the dynamic programming comes. So, here is a quick view of the dynamic programming. So, the first idea is to find some kind of a repulsive solution. How do we do that? Well, we cast our problem as a search for some object over certain search space. So, it is useful to define the search space quite clearly as well as it is important to define clearly an objective function which is to be minimized or sometimes it has to be maximized whatever it is it has to be defined very clearly. Then we design a algorithm which searches the space typically the algorithm is going to partition the search space.

So, it is going to say let us divide the search space into spaces and each space or each part of the space each subspace is going to be searched separately. So, I search the first subspace and I get the best solution. And I get the optimal solution in it what I mean by that is the solution which minimizes the objective function. Or if we want maximization

the solution which maximizes that objective function. So, we calculate the best optimal in each search space and then return the best of the best. So, that is the general idea of getting a recursive solution dynamic programming; however, does not stop at this point. It proceeds further in the following sense the idea is that will characterize what the recursive calls are in this part. So, essentially we are going to ask the question what is that we are what are the problems that we are solving? So, we essentially make a table of all those problems and in each cell of the table we would like to store the results of those.

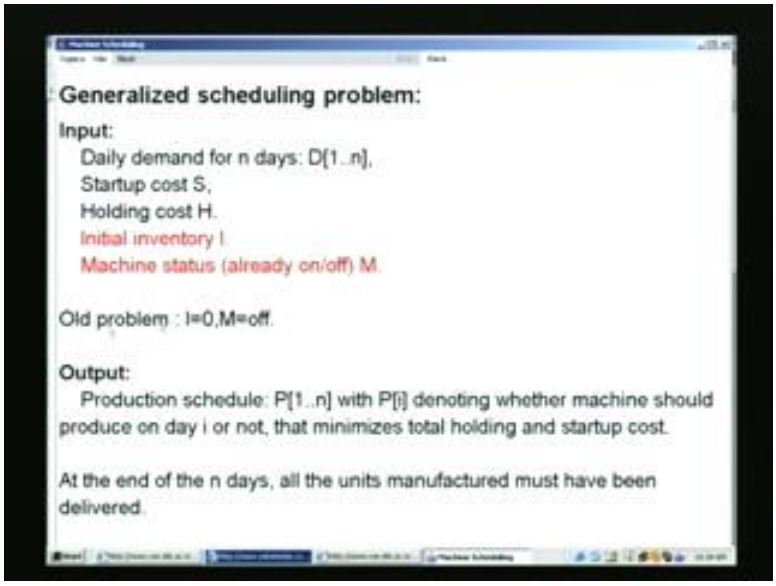
So, we will define such a table then will define a procedure for filling table entries and this procedure will be a direct procedure it would not very particular and it will assume that if I want to fill a certain table entry I can use entries which have been filled earlier. So, this will fill entries in the table given that other entries have been already filled. So, in this recursive procedure the recursive procedure might be slower, because it might calculate the same quantity several times. Whereas, in this case when we make the table we will carefully think about what it is that needs to be calculated and will calculate that exactly once. So, that is going to give us our efficiency. So, what we would like to do now is apply this whole idea to our given problem. The first step in this is to find the recursive solution. So, we want to cast our problem as a problem in the recursive fashion our algorithm in a recursive fashion.

(Refer Slide Time: 11:19)



So, here is how you might think of a recursive algorithm for our problem. So, we would like to solve a n day problem. So, typical steps in that might be that we somehow solve the first day and then we recurse on the remaining n minus 1 days. Now, unfortunately recursing on n minus 1 days will depend on what happens on the first day. So, say for example, if the machine has been switched on on the first day then when I recurse this information is very useful to me. So, if the machine is on then on the first day of recursion I do not have to pay the startup cost. So, which means our recursion must somehow include additional history information the most natural way of doing this is to generalize our problem.

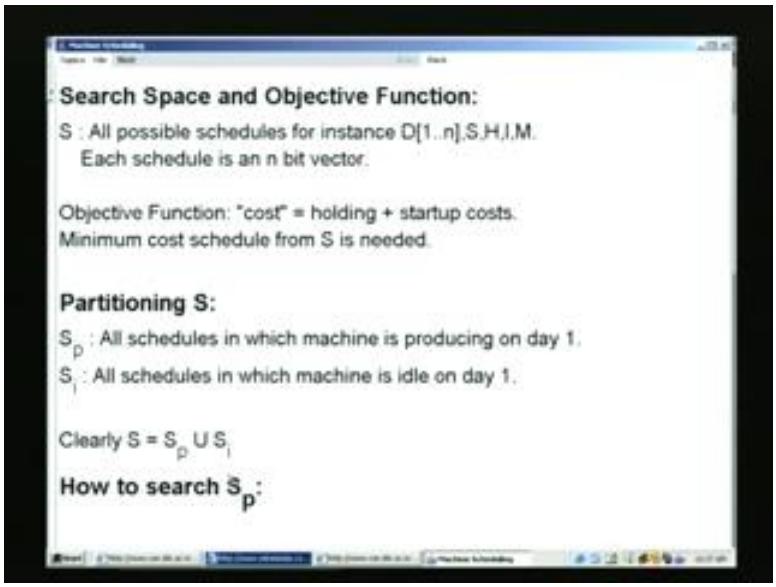
(Refer Slide Time: 12:18)



So, let us see that. So, our generalized scheduling problem looks something like this it has the same inputs as before except there are some more. So, it has inputs the demand as inputs the demands for n days the startup cost the holding cost. And then there are 2 additional inputs and input I which gives the initial inventory. What I mean by inventory is that how many units have already in stock on day 1? So, when I begin the whole thing it is possible that I already have some units in stock? So, that number is specified as a part of the input and that is this number I. Then on the day which I begin the machine might be already on or off and that is specified by this variable m. So, M can take values on or off I can take values any integer the rest is similar well I should point out that our old problem which we had defined earlier corresponds to having I is equal to 0 that is no inventory when we start and M equal to off. So, this is the generalization in the sense that

now we allow I and M to take on a wider range of values our output is as before. We are supposed to produce a production schedule and the requirements are same that at the end of n days everything that has produced must be consumed.

(Refer Slide Time: 13:54)

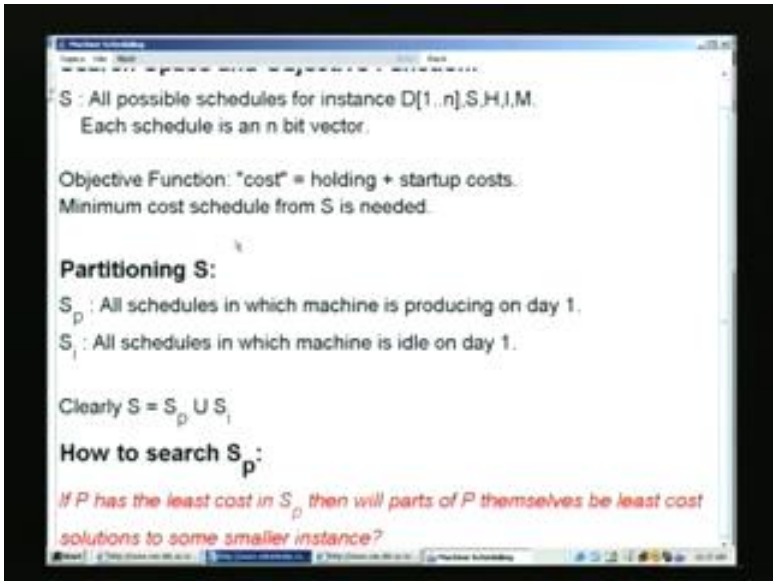


So, let us now try to define a recursive algorithm for this problem. So, the first step was to design or think about a search space for this problem and the objective function. So, let me call S the search space. So, what is the search space for the input problem as given. Well the search space will contain all possible schedules for this instance under instances characterized by these inputs. So, a instance is defined as D S H I M where D is a vector of n elements if it helps us to think about will think of each schedule as being as n bit vector. So, each bit specifies whether the machine is to be on or off on the corresponding day. The objective function is the cost and it just consists of the sum of the holding and the startup costs.

And our goal is to get the minimum cost schedule from this space S. Next comes how we are going to partition this here is a natural way in which the space can be partitioned? So, we will ask what does what happens on the first day. So, in sub space S sub p we will put all schedules in which machine is producing on day 1. In space sub space S sub I we will put all schedules in which machine is idle on day 1. Now, this union this is this in the sense that in every schedule in every element of s. Either the machine is producing on the

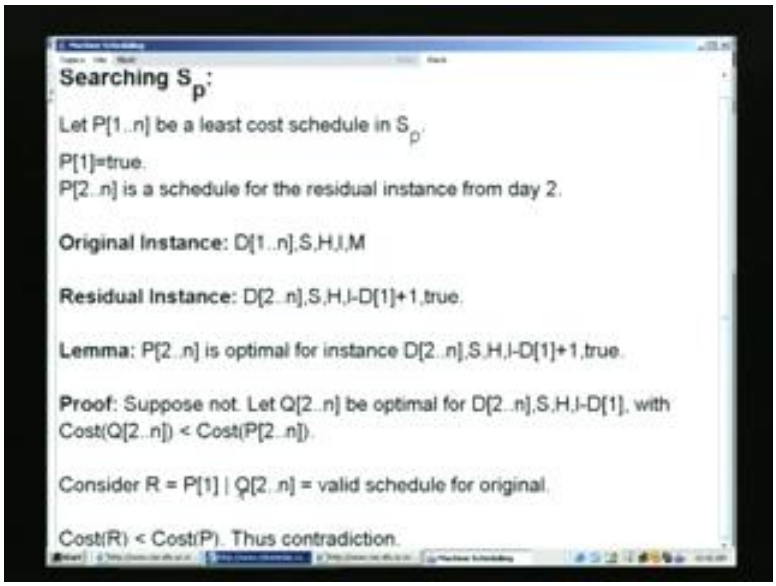
first day or it is idle on the first day and therefore, this is equal to this union this. So, we have partition S and so the next question is how do you search each of these sub spaces.

(Refer Slide Time: 16:13)



So, let us look at S sub p first and we will try to think of how to search S sub p. Here is the key idea in devising recursive algorithms for optimization problems. So, if p is the least cost element of S sub p then the question we should be asking is will parts of p themselves be least cost solutions some smaller instance. If they are then we can use recursion. So, this is called the so called optimal sub structure idea. So, if p is an optimal solution in this subspace S sub p. The question is will parts of p will also be optimal for a smaller sub problem and in fact; it turns out in our case that that is true.

(Refer Slide Time: 17:00)



So, let P 1 through n be a least cost schedule in S sub p will show the parts of it will have to be optimal for a sub problem for a smaller instance. So, if p is the least cost schedule what do we know about it well we know that P of 1 is equal to true, why because p is the least cost schedule in S sub p and S sub p is a set of schedules in which the machine is on in the first day. So, P of 1 must be true P of 2 n is the rest of the schedule, but now I can think of P of 2 as a schedule for the residual instance from D 2.

So, let me clarify this. So, our original instance was this D S H I M when n integer vector. The residual instance is just a part of this which begins in the second day. So, in the residual instance we have only demands for days 2 through n then S is the same the startup cost does not change. The holding cost does not change; however, the inventory on day 1 is going to be different on day 2 is going to be different on day 1 the inventory is 1. And on the first day we produced of the inventory that was present on day 1. We are going to deliver D of 1 which was what the problem require has to deliver and then it will add whatever we produced on day 1.

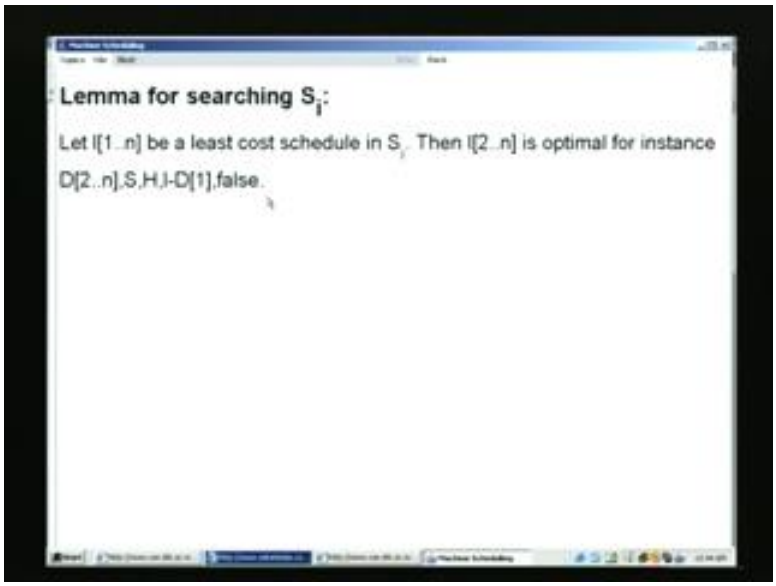
So, this is the inventory for this instance beginning on the second day the final component is the machine status. So, over here the machine status could have been whatever now we know since we did produce on the first day the machine status had better be true. So, this I the residual instance and clearly p 2 n must be a schedule which satisfies this residual influence. Here is the key lemma this lemma says that not only

does it satisfy the residual instance, but in fact it has to be a optimal schedule for this instance. Or in other words it has to be a least cost schedule for this instance the argument for this is fairly straight forward. And it is of the typical argument in dynamic programming arguments. So, well do the argument by contradiction. So, we will assume that p 2 n is not an optimal schedule which means there are better be a schedule Q 2 n which is optimal for this instance and it is cost be better smaller than the cost of p 2 n.

So, cost of Q 2 n must be smaller than cost of p 2 n, but now let us consider this schedule R where R follows p on the first day. Or in other words R is true on the first day and on the remaining days it follows Q clearly this is a valid solution for the original instance right. Because well on the first day we produced and then we were left with this instance and that is what Q took care of. Now, what is the cost of R? Clearly cost of R must be smaller than the cost of p why because cost of R is nothing but the cost for remaining days plus the cost of p.

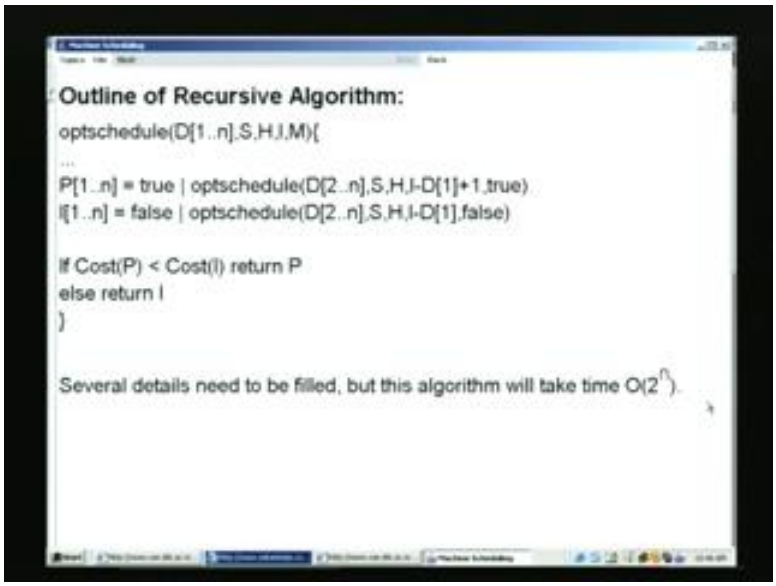
The cost of the first day this is the cost of R cost of p is cost with the first day plus cost of the remaining days which is not Q 2 through n, but p 2 through n. But in p we are adding this whereas; in R we are adding this; this is smaller than this. And therefore, the cost of R must be smaller than cost of p, but remember we assume that p is a least cost schedule in S sub p. But here we are showing there exists a schedule R which has even less cost than p and also the first element of R is true, because it is P 1 itself. So, R also belongs to S sub p and therefore, we have a contradiction. And therefore, our basic assumption must have been false or in other words our lemma must have been true then this is also optimal for instance the residual instance. So, this tells us how to search the space S sub p and a similar idea works for the S sub r.

(Refer Slide Time: 22:00)



So, let us say I 1 through n be a least cost schedule in S sub i. Then I 2 through n is a least cost schedule or is optimal for the instance D 2 n S H I minus D 1 false or this is the residual instance. So, see that the last component the machine status is false, because S I consists of schedules in which the machine is off on day 1. Therefore this is false and the original elementary was i but on day 1 we delivered something without producing. So, the new entry is I minus D 1. So, this is the residual instance and this remark claims that I 2 through n must be optimal for this residual instance. So, we have accomplished the goals that we set. So, we have shown how to search S sub p and how to search S sub I and this has been done by using recursion.

(Refer Slide Time: 23:14)

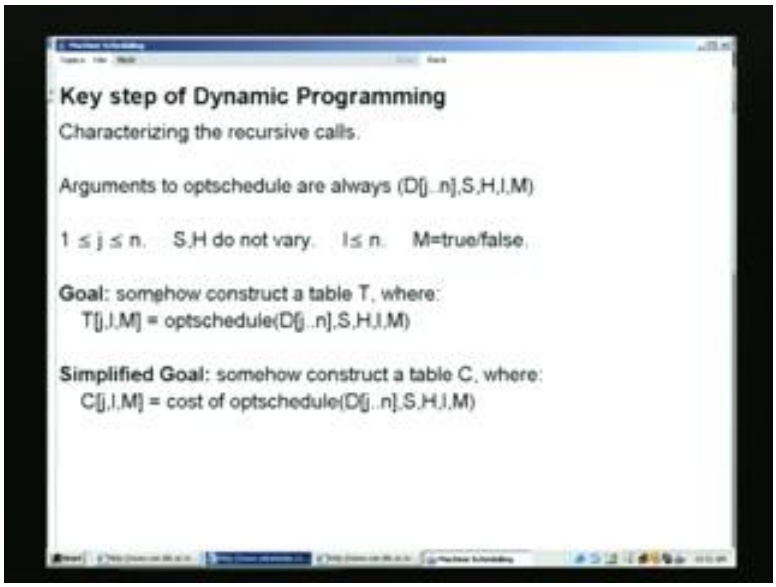


So, we are essentially ready to build a recursive algorithm for sorting this problem. Here is a outline. So, we will call our algorithm opts schedule arguments the entire demands and when I write D 1 through n over here I simply mean that D is going to be a vector with n elements. It will take additional argument which is the startup cost the holding cost the current inventory at the beginning and the machine status at the beginning. There will have to some best case that will have to take care of, but this is sort of the main core of the recursive algorithm. So, this part in this part we are going to search the space S sub p. So, we are going to find the optimal schedule for the recursive residual problem. Assuming that on the first day we do produce and that will be our schedule P 1 through n I 1 through n will likewise be the optimized schedule for S sub I and in this case. On the first day we are not going to produce anything and this is just the residual problem given that we do not produce anything on the first day.

So, these are the problems which we exactly looked at in the 2 lemmas that we just saw. And then we are going to look at the cost of the first schedule the cost of the second schedule and we are just going to return the best of these. So, this is the best of best idea. So, this is the best solution in the first sub space this is the best solution in the second sub space what we are returning is the best of the bests and number of details have to be filled up over here. However, you should be able to argue that this algorithm will take time O of 2 to the n see just try to recurrence and this will just come over. So, what we are going to do now is not to fill up the details the remaining details and there are a few

important details which are missing over here. But what we have done is we have essentially identified what kind of recursion we are going to have. So, we will just proceed to the next step of our agenda.

(Refer Slide Time: 25:34)



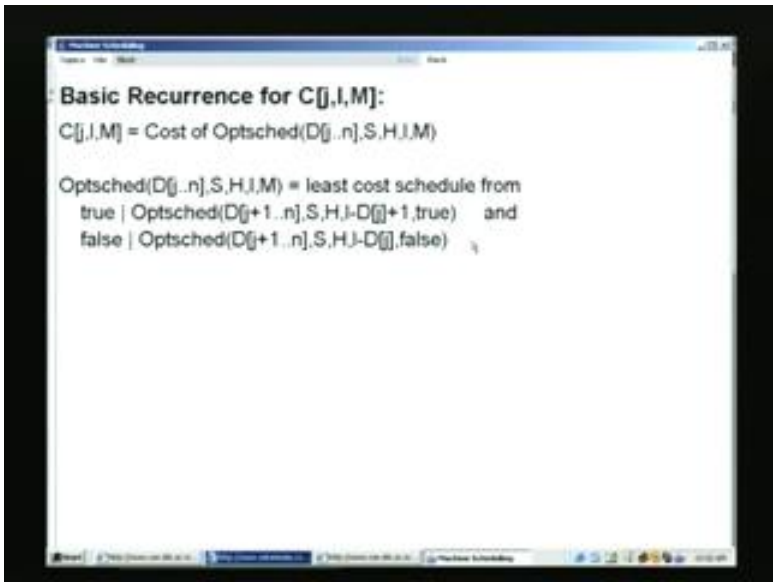
So, the next step is the dynamic programming idea. So, this step is the key step of dynamic programming which is characterizing the recursive calls. So, the first observation is that opts schedule is always called using arguments D J through n and S H I M where J can be any number. So, let me just observe let me just point out why this is. So, initially opts schedule is going to be called with the entire input. But later on it is going to be called with the residual problem in which we are only going to pass the sub array beginning with the second index. Now, what happens in this call itself in this call we are again going to throw out the first element of the demand array and then we are going to call it with the rest of it. So, when we are doing the recursion on it we will be calling it with D 3 n and then when we recurse on that we will be calling D 4 n and so on. So, in other words the argument is going to be of this form that we will be that the first argument that is going to be passed will be some sub range.

Well, it will always be a terminal sub range of our input D. So, suffix of that D array and then these could be pretty much anything. So, J has to be some number between 1 and n. So, the largest index is n. So, J could be anything until n S and H get passed without any change what so ever the inventory can be at most l why is that? Well, we have n days

during which the machine produces 1 unit per day and therefore, the inventory cannot ever build up to more than n. So, this argument will be an integer at most l and the last argument could either be true or false. So, we have a reasonably good characterization of all the recursive calls that could get made in our recursive algorithm. So, the characterization says that these are the arguments that are going to be passed and each argument can vary in this manner or not vary at all. The next step of dynamic programming says well now that you have identified what the recursive calls are going to be make a table in which we are going to store the table. So, we will construct a table T in which T will have 3 indices J I and M. So, T of J I M will store the result of optschedule of D J through n S H I M. The point of making the table really is that we just want to focus we just want to make a list of what are the sub problems that we are ever going to solve.

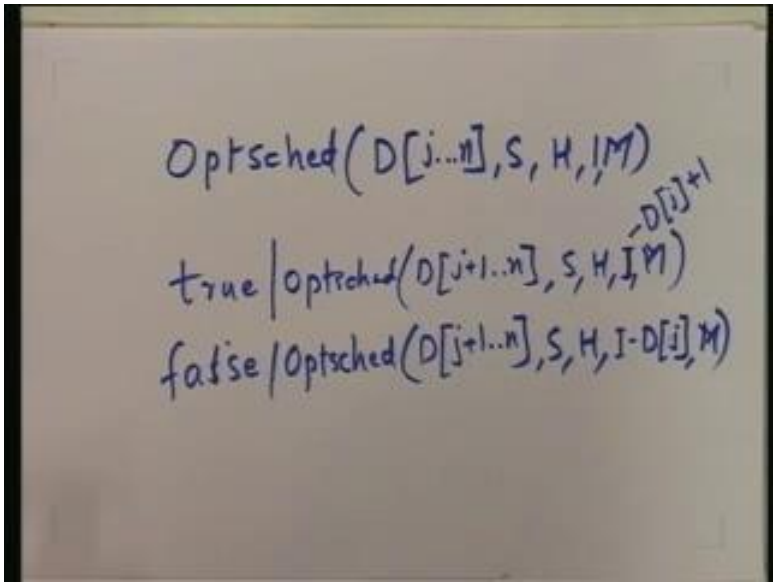
So, this is what we will do. Now, we can work with this table T; however, dealing with schedules is a little bit cumbersome because we are going to simplify the problem just for a few minutes. So, instead of working with entire schedules we will just work with the cost of optschedule. In the table T we were planning to store in each cell the entire optimal schedule instead of that we will build a different table let us call it C for cost. It will have same very similar entries similar number of entries with similar indices and in this we will just store the cost of the optimal schedule. So, now, we are now going to work with this table and what dynamic programming requires to do is to figure out how entries in this table depend upon each other. So, in another words if the entry C J I M has to be filled assuming the remaining entries are full what is the exact computation that needs to be done.

(Refer Slide Time: 30:13)



So, we need to derive some kind of recurrence for C J I M let me just remind you that C J I M is the cost of this optimal schedule for the sub problem or for the residual problem actually D J n S H I M. Let me just remind you that you did not have S H in the table, because they are fixed throughout the execution anywhere. Here is what the optimal schedule for D J n is going to look like. So, I claim that the optimal schedule for D J n S H I M is going to be the schedule with the lower cost from this schedule and this schedule. This really comes from the procedure that we wrote a minute ago. So, let us just see that.

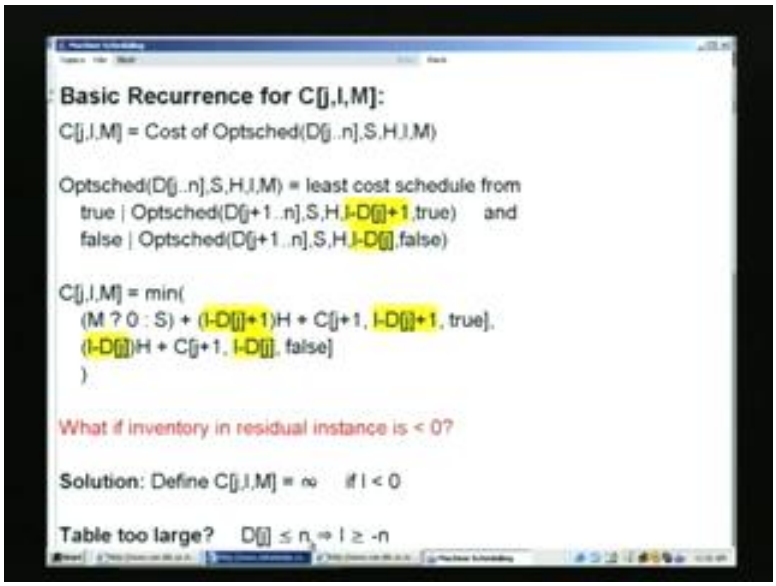
(Refer Slide Time: 31:16)



So, let me just write this term. So, my claim is or what I want to examine is the optimal schedule for D of J through n S H I M this is what I want to understand and I want to figure out how what will the optimal schedule to this be? So, let us go back to the code that we wrote or the recursive algorithm that we wrote. And we said that the optimal schedule for D 1 through n SH I M is going to be one of these 2 now, one of these 2, when D is passed as the entire array is this, but only when a small smaller range is passed. So, the first argument is 1 what will this be? So, the first schedule will simply be true concatenated with optschedule and this time instead of passing 2 through n. I should really pass J plus 1 through n, because this was just dropping the first element. So, if I drop the first element from this this will become D of J plus 1 through n S H I M what about this?

So, this will become false and optsched D of J plus through n S H and in the inventory I should really remove what was present on day 1. So, or not on day 1, but on day J and I should have M this is what I should have. Here instead of inventory being I i should really have I minus D of J plus 1, because since the machine was on the first day I had to have a plus 1 over here. So, you can see that this is a this is exactly what I have written over here. So, true concatenated with optimal schedule for the demands D J plus 1 through n S H I minus D J plus 1 and true and the other schedule which is optimal schedule false concatenated with optimal schedule of D J plus 1 through S H I minus D J and false. So, now we understand what optimal schedule is. So, we just now have to figure out what the cost relationship amongst the cost is going to be?

(Refer Slide Time: 34:20)



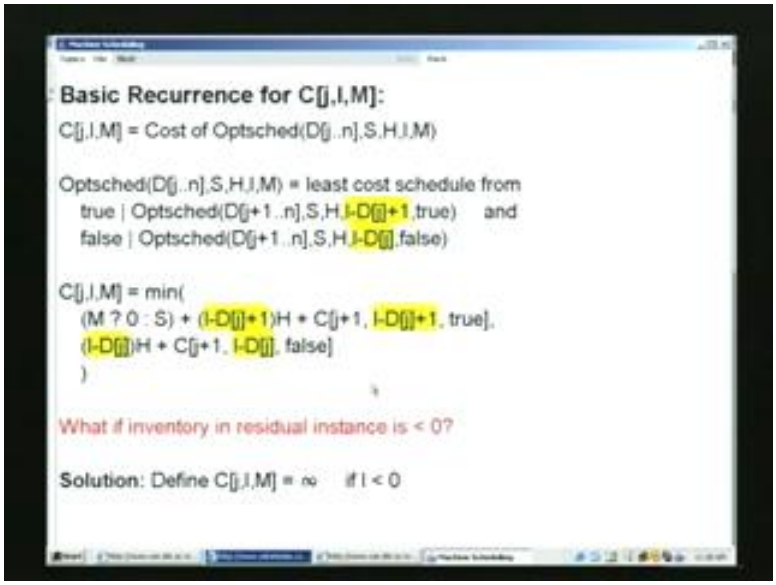
So, C J I M is simply the cost of this. So, what is that going to be well this is the least cost of this? So, C J I M better be the minimum cost of the cost of these 2 schedules. So, what is the cost of these 2 schedules? Well, the first schedule starts with keeping the machine on day 1. So, here there is some potential for incurring a startup cost whether the startup cost is incurred depends upon whether this n was on or off or true or false to begin with. So, if M is true then there is no startup cost or the startup cost is 0. If n is false then the startup cost is S. So, this expression which I have written down over here is to be understood as the C expression that is the C expression mark true value colon false value. So, if M is true then this bracket evaluates to 0 otherwise it evaluates to false in other words this bracket over here represents the startup cost it is either 0 or it is depending upon M. Then on day 1 there is going to be some inventory cost. So, this inventory cost is given over here. So, the inventory for day 1 is going to be this whatever the inventory subtract whatever was delivered on D J add whatever was produced. So, times H will be the inventory cost and then there is a residual cost. So, it is just the cost of this part of the schedule. But this part of the schedule the cost of this is simply C J plus 1 I minus D J plus 1 true.

So, we have related the J I Mth entry to the J plus 1 I minus D J plus 1 2 entry. So, we have to take the minimum of this quantity and this quantity and in this case the computation of the cost is actually simple. Because there is since the machine is off on the first day this is false there is no startup cost, but there is some holding cost. So, what

is the holding cost on day J something gets deleted the inventory gets reduced by I minus D j. So, the inventory is just this and the inventory cost is just this and this is the cost of the residual problem. So, now, we have a recurrence connecting C J I M to other entries in the table. So, this is the one other entry and this is 1 other entry there are some problems though. So, when we wrote down these numbers, we did not we have not so far considered the possibility that I minus D J plus 1 or I minus D J might become negative that does not make sense. So, I minus D J plus 1 or I minus D J is the inventory. So, it does not make sense for the inventory to become negative.

So, we somehow have to take care of that essentially what is going to happen is what should happen is that we should check that if we want to consider the schedule. And if we are requiring that I minus D J plus 1 be negative then we should not really be considering it at all and we should just use this. Here is a very nice fix to this a simple fix. So, we are going to define C J C of J I M to be infinity if I is less than 0. So, let us see what this does? So, in this example in this alternative for example, if in this table entry this is some negative number. So, we are asking for C of J plus 1 some negative number true now this entry would be defined as infinity which would mean that this number would be infinite. But if this number is infinite then this entire cost would be infinite. But since we are taking the min over here, that would force us to look at this cost. So, essentially we would be ignoring this cost.

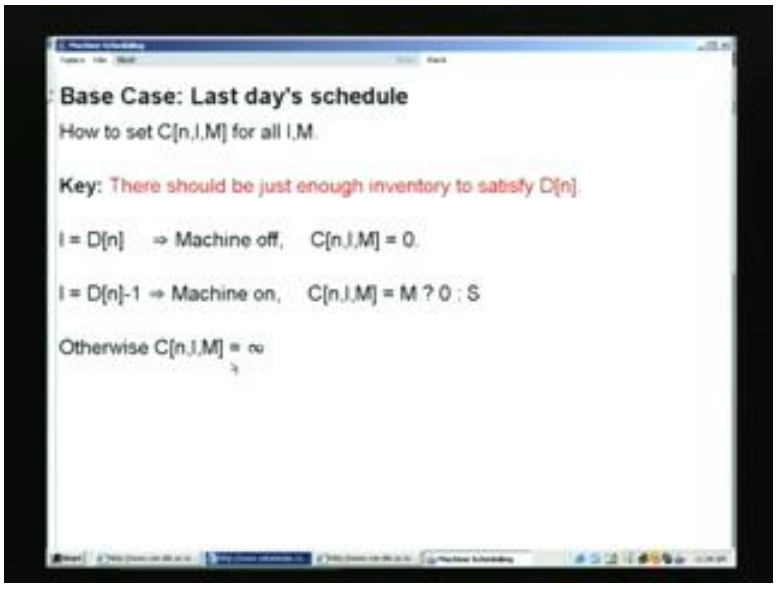
(Refer Slide Time 39:00)



So, by defining C J I M equal to infinity we are saving ourselves the trouble of checking the indices are less than 0 in this expression and automatically getting the same effect. Now, of course, you might ask if we want to do this for all I less than 0, do we need to consider a infinite all the infinite negative numbers for I? Well, we do not and there is a simple reason for it, remember that the demand on any day cannot be bigger than l. Why if the demand is bigger than n then we cannot fulfill that instance anyway. And so, we can check at the beginning whether the demand on any day is bigger than n. In which case we just reject we just say that this problem is unsolvable.

So, this algorithm or whatever algorithm we are going to design will only be called if D of J is less than or equal to n. But if D of J is less than or equal to n then we are only subtracting n from whatever has to be a positive number earlier. So, in which case it suffices I is bigger than or equal to minus n. So, we only need to consider I only a few negative values of n that is values of I that is values going from minus l to minus n. So, in other words our table is not going to become too large the final problem that we need to consider is the base keys. So, we relate entries with the table with other entries, but this cannot go on forever. So, some entries we have to set ourselves.

(Refer Slide Time: 40:39)

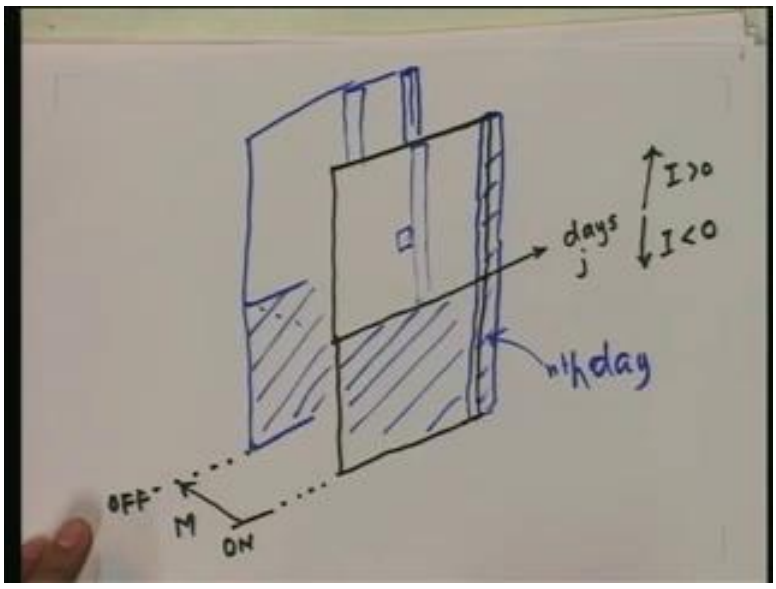


So, these are the entries for the last day schedule. So, remember that C J I M were the indices for the table entries. If you look at the last day then the first index over here should be n. So, in this case we are asking for a single day schedule. So, we should be

able to set this without much doubt. So, here is the idea if we want to figure out what C I and M are to be and we need to figure whether there is a legal schedule. So, the main condition over here is that whatever inventory we come in with should not be too large or should not be too small. If it is too large then even after satisfying the demand we will be left with inventory and that is illegal that will not constitute a valid schedule. If it is too little we may not be able to satisfy D of n at all. So, the inventory should be just enough to satisfy the last day's demand.

So, very simply if I is the last day's demand if I takes the value last demand then we should run with the machine we should not produce anymore units and. So, in that the cost for the last day is simply 0. So, C of n I M must be 0 if I is equal to D of n if on the other hand the inventory is just 1 less as we come into the nth day. Then what the demand is then we would better produce to make up the inventory that we want to deliver whatever is needed to be delivered. So, we will have the machine be on and now for the last day we need to run the machine. So, our cost is going to be 0 if the machine was already off was already on and S if the machine was off. So, this is again the C style expression which evaluates to either 0 or S for all other values of I other than D n and D n minus 1. This means that we have too much inventory or too little inventory in which case we should set the cost to be infinite. So, that takes care of the best case as well.

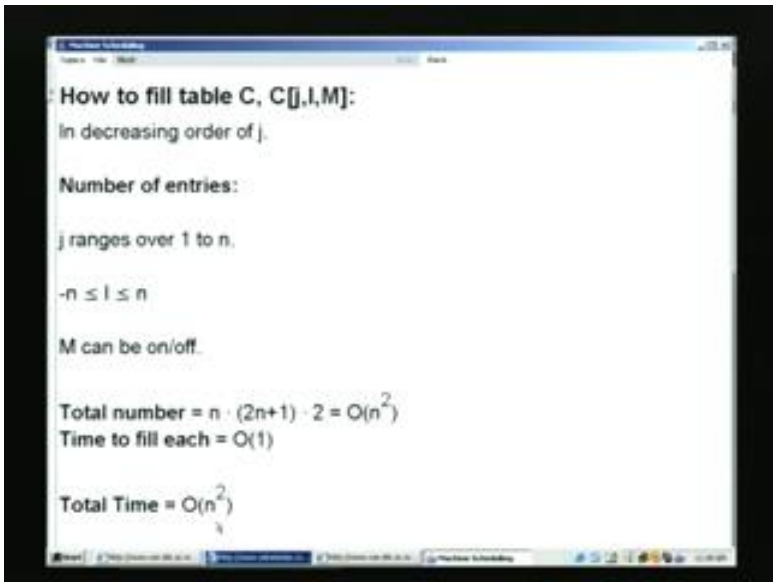
(Refer Slide Time: 43:23)



So, let me pictorially show what has happened here or let me pictorially show whatever table is going to look like. So, our table is going to look like something like this. So, on this axis I am going to have I am going to have the days on this axis or this is where how J is going to vary on this axis I am going to have inventory . So, this is inventory greater than 0 on this side I have I less than 0 and on this axis into the page if you will I will have M . So, this corresponds to say on and this corresponds to off. So, I will draw the off in a different color. So, it will look something like this. So, there will sort of 2 sheets. So, even the second sheet. So, our table is going to look something like this now how did we fill the table? Well, in this case we looked at the last column on the days side. So, this is what we filled out all right. So, this part we filled out. So, this is nth day and this in the second part in the off side also we filled it out. So, these values are known then, because of this we divided C J I M equal to infinity where ever I was 0. We have also filled this out this part.

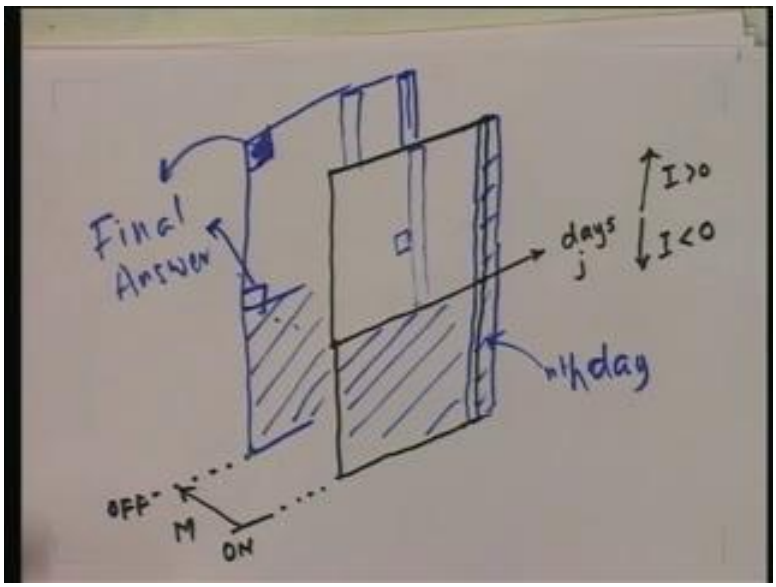
So, these parts are filled out to begin with by very simple ideas. So, the recursion part comes only for this part and this part over here. So, how is that filled? Well that is filled using these expressions. So, essentially if I want to fill this entry what am I going to use. Well, I am going to use some entry in the J plus 1 th column some value from this column and also the corresponding column of the off side on the, of the machine being off, because I am getting C J plus 1 true and C J plus 1 and false as well. So, this entry depends upon the next column. So, now our calculation is quite straight forward. So, this part is already been filled up easily by the last 2 days calculation and by this calculation. And then to fill each entry we will just use this expression and this entry can be filled in constant time as I have seen over here as I have shown over here. So, basically the idea is that we fill the entire table in decreasing order of J. So, we start from here and go backwards and we fill in the table. So, the number of entries here well let us calculate that carefully J ranges from 1 to n. So, here it is 1 and here it is n I goes from minus n and it goes all the way to plus n.

(Refer Slide Time: 47:20)



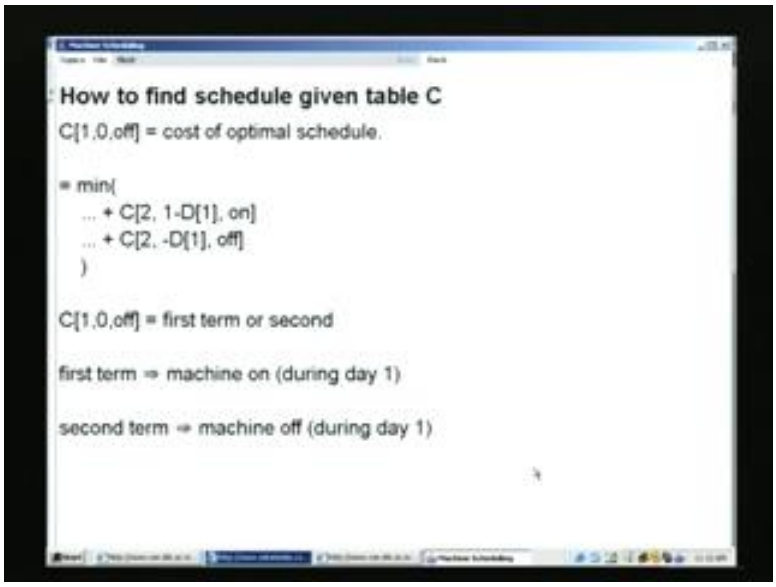
So, there are 2 n plus values for I and M can be on or off. So, there are only 2 values of M. So, that total number of entries in the table is just all possible choices for J all possible choices for I and all possible choices for n. So, n times 2 n plus 1 times 2 or O of n square time to fill each entry our recurrence can be evaluated in constant time or each single step can be evaluated in constant time. So, that is O of 1 and as a result the total time is O of n square. So, we will fill the entire table in O of n square time we should ask well we have filled this table, but which part is the 1 that we want.

(Refer Slide Time: 48:19)



So, coming back to this; this is the entry which we want. So, at the end this is our final answer or I am sorry not this it is this, this is the final answer or the optimal cost is found in entry 1 0 falls of the table. So, 1 is the J value 0 is the, I value. So, it is this entry on this axis itself and it is in the off or the false side. So, this just says what is the cost of generating a schedule for the entire demand? Given that there is no inventory to begin with and that the machine is off.

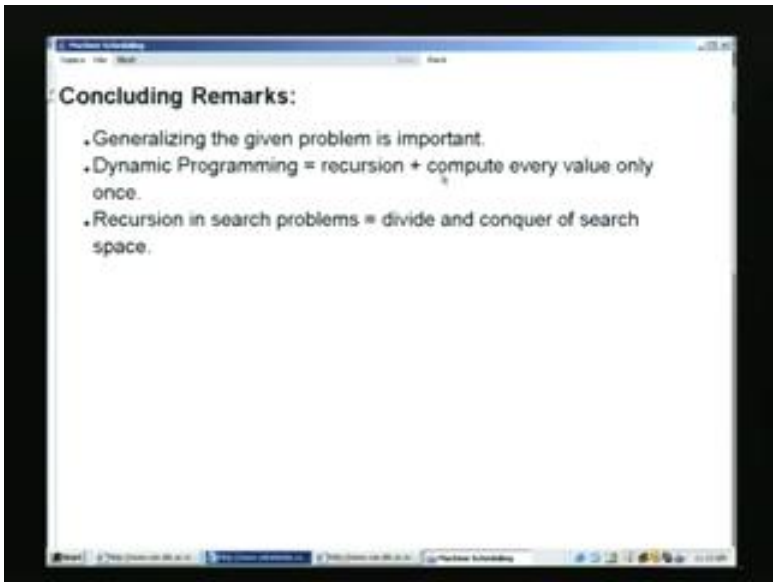
(Refer Slide Time: 49:12)



The final topic is how do we find out, how do I find the schedule given the table? So, remember that C 1 off was the cost of the schedule. Now, this cost is going to be a minimum of some costs which costs well, we note we noted that if I want to calculate this cost it is the minimum of some cost in this and some cost in the corresponding column over here. So, C 1 off would be the minimum of some expression involving some column some element in the next column on the off side as well as in the next column on the on side. So, we simply check whether C 1 is equal to the first term or the second term it is the minimum. So, it should at least be equal to at least one of these. If C 1 off is equal to the first term what do we know? We know that this must be generated by keeping the by using the first term or by keeping the machine on during day 1. If this term is equal to the second term over here then the machine should be off on that first day. So, in this way by knowing the optimal cost we were able to and knowing the table we were able to figure out whether the machine should be on or off on the first day.

And we are also able to figure out what the corresponding entry for the optimal schedule for the residual plan is is it this or is it this So, then we can apply this argument again and again and will get machine status for every day and will does generate the entire schedule. So, let me now conclude. So, in this problem in today's lecture we saw another problem for which dynamic programming could be used before using the dynamic programming. There was a important step that we took which is quietly an important and interesting step very often we are given problems for which to device recursive algorithm a we need to generalize the problem formation itself. So, here is 1 such example you must have seen similar example in another problem which was the problem of medium finding. If we want to device a recursive algorithm for medium finding well, we cannot do that very easily or very simply. So, what we do instead is we generalize the problem and ask for recursive algorithm for finding the arc smallest. So, some similar issue was applicable here as well.

(Refer Slide Time: 51:59)



I would like to make a comment on dynamic programming and the comment is simply the dynamic programming can be thought of as recursion a basic idea. The basic idea the first basic idea, basic idea is recursion and the next idea is make sure that you compute every value only once. And our table essentially made as focus on what values we were computing and we were and we thereby we could only calculate we can make sure that we have calculated every value only once. Finally I would like you to I would like to draw your attention to recursion itself and point out that when we use recursion in search

problem in any search problems. It can be thought as a divide and conquer of the search space. So, with that I will conclude this lecture.