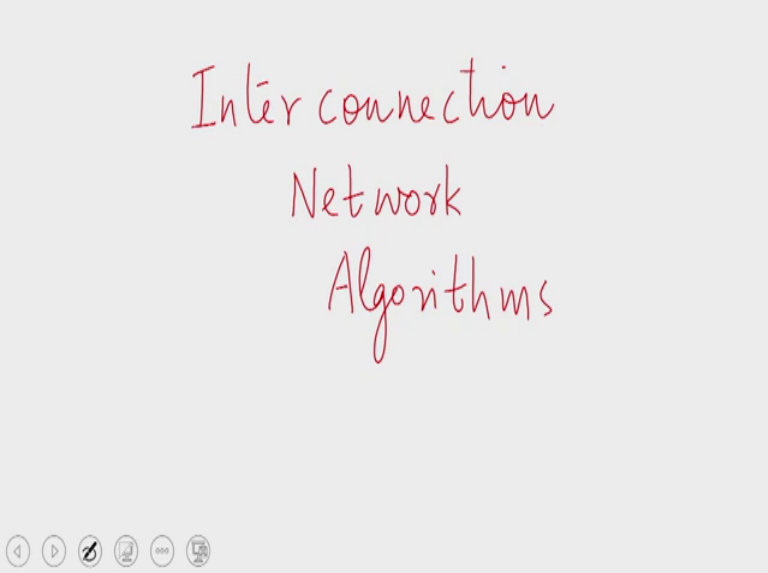
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 24
Sorting on a 2D mesh

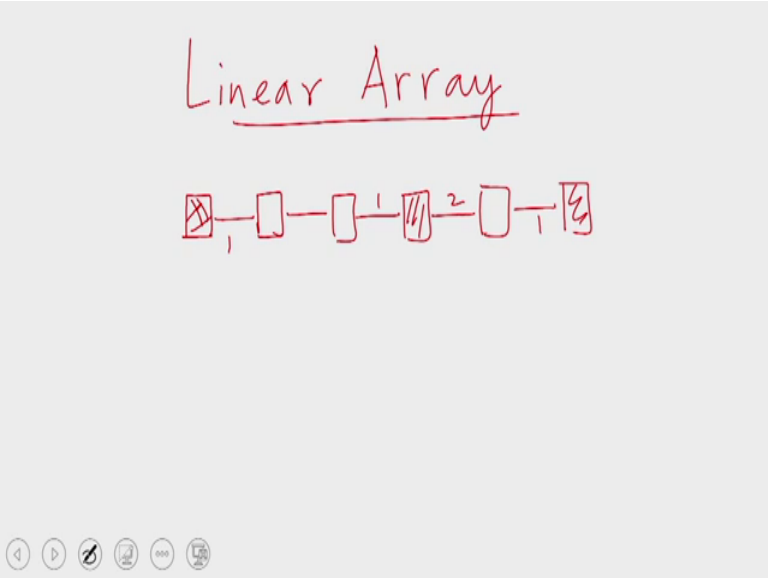
Welcome to the 24th lecture of the MOOC on Parallel Algorithms.

(Refer Slide Time: 00:39)



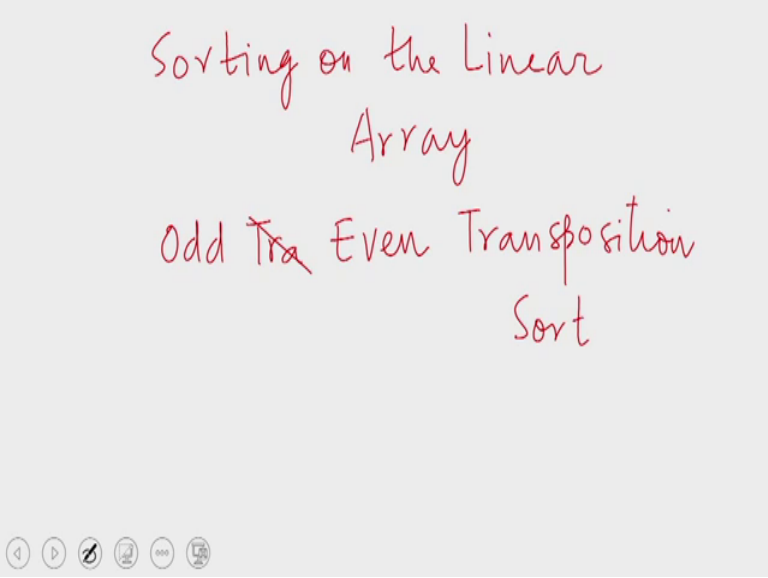
So, today's lecture, we will start the study of the interconnection network algorithms. We have already seen an introduction to the interconnection networks and we have seen some algorithms on them. Today we begin a detailed study of this series.

(Refer Slide Time: 01:06)



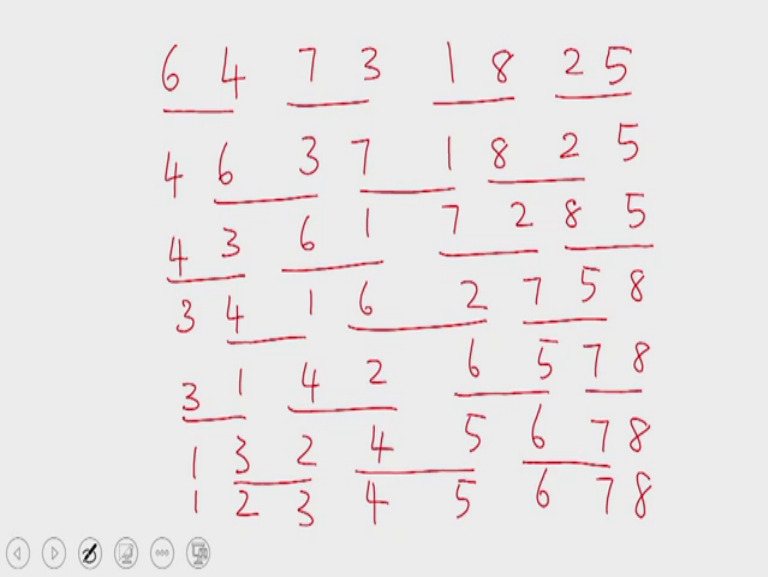
So, we will begin with linear arrays. You would have called that in a linear array, we have a number of processors connected up in this manner. Every intermediate processors had processor has got two neighbors; one on either side, but the processors at the extremes have got only one neighbor each. The extreme processors have a degree of 1 every other processor has a degree of 2. So, such an arrangement of the processors is called a linear array.

(Refer Slide Time: 01:51)



We saw an algorithm for sorting on the linear array. Let us recap and read this algorithm and do an analysis of this which we had not done earlier; this was called the odd even transposition sort.

(Refer Slide Time: 02:33)



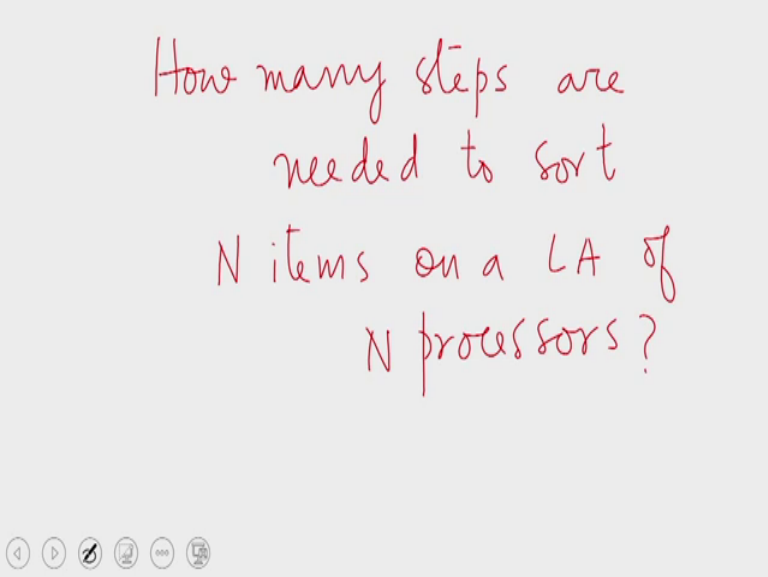
So, let us take an example. So, let us say we have given we are given a linear array which holds these elements which are to be sorted in place. What this means is that the processes already hold these elements. These elements are not streaming into the linear array. The linear array is already filled with these elements and what we need to do is to sort them in place.

So, the algorithm groups them from the left, an odd element is paired with the even element coming next to it. So, 6 is paired with 4 7 is paired with 3 and so on that is because the first position and the second position form pair, third position and the fourth position form a pair and so on. And then for each pair we perform a compare and exchange.

When 6 and 4 are compared we find that 4 is smaller than 6. So, we exchange them here we have 3 smaller than 7 we exchange them one is smaller than 8, there is nothing to do 2 is smaller than 5 there is nothing to do again. Then in the next step, we pair every even element with the subsequent odd element which means 6 is paired with 3, 7 is paired with 1, 8 is paired with 2. We compare and exchange 4 and remains where it is. Here we have 6 3 rectified to 3 6 7 1 becomes one 7, we have 2 8 and 5

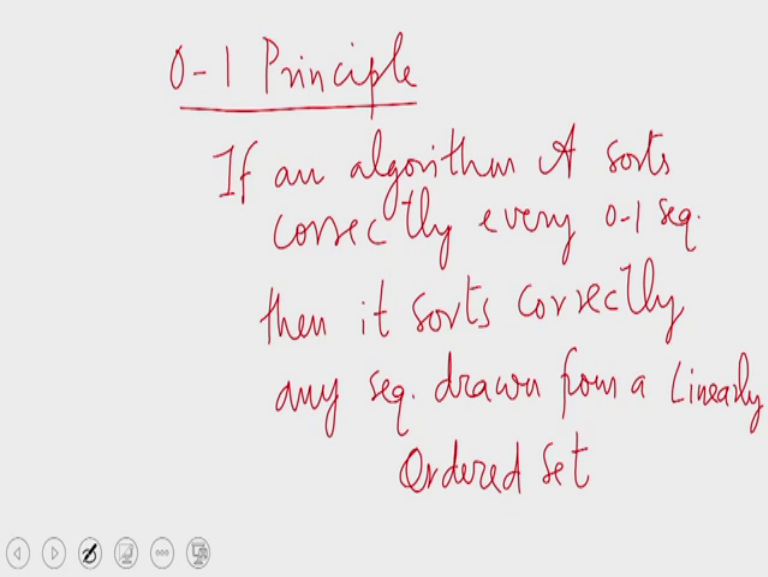
Then we go back to the original pairing. The odd even pairing here, we have 3 and 4 1 and 6 2 and 7 5 and 8. And then once again we have we have an even odd pairing. Again we have an odd even pairing and the next even odd pairing will fix the array.

(Refer Slide Time: 04:53)



So, we find that the solution emerges after a number of steps, but what is the number of steps? Let us find out how many steps are needed to sort N items on a linear array of N processors.

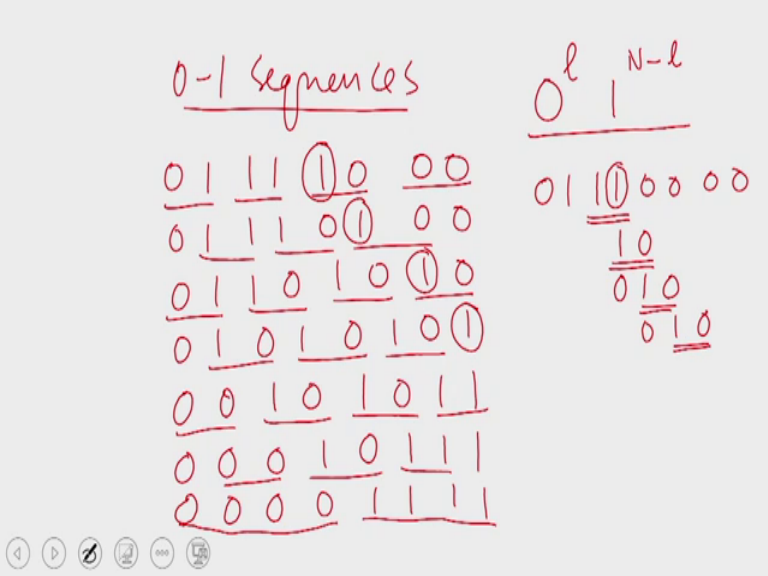
(Refer Slide Time: 05:33)



To prove this we shall fall back on a principle that we have already seen which is called the 0-1 principle. You will recall from our algorithm on comparator networks that 0-1 principle can be used to sort the; to prove the correctness of sorting algorithms.

What 0-1 principle says is that if an algorithm a sorts correctly every binary sequence, then it sorts correctly any sequence drawn from a linearly ordered set. So, to prove that our algorithm works correctly it is enough to show that it works correctly on all binary sequences

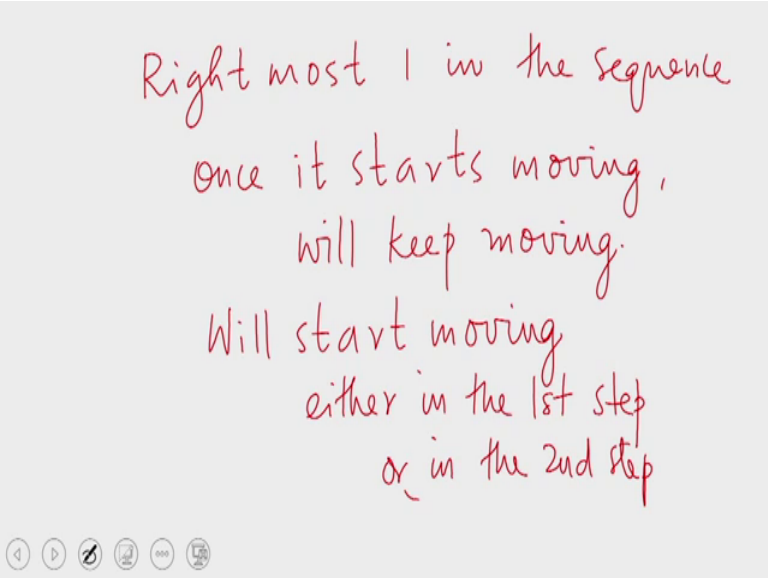
(Refer Slide Time: 06:51)



Then by the 0-1 principle we can argue that it will work correctly on any single input. So, let us see what happens to 0-1 sequence when it is subjected to the odd even transposition sort.

Let us take a binary sequence. We paired them off in this fashion odd even, odd even, odd even, odd even and then we sort them. 1 0 becomes 0 1 and then we pair them even odd. Again we pair them odd even; you find the algorithm now begins to converge. At the next iteration we have all the 0s at the left end and all the 1s at the right end. That is when a binary sequence is sorted; we will have an arrangement of this form. If the sequence has l 0's and n minus l 1s, then at the end of sorting all the 0s will be at the left extreme and all the ones will be at the right extreme. So, let us now try to argue that when odd even transposition sort is applied on a binary sequence, it will terminate within some number of steps

(Refer Slide Time: 08:56)



In particular let us consider the rightmost 1 in the sequence. Let us look at the example here. Here this happens to be the rightmost 1. Here the rightmost 1 is compared with the 0 on its right and therefore, gets exchanged for the 0 and therefore, it moves second position in the second step. And then it further moves on to the next position in the third step and in the fourth step it has reached its final destination.

So, what we find is that once the rightmost 1 starts moving, it will keep moving. This is because the rightmost 1 is going to encounter all 0s to its right. So, whenever it pairs with an element on to its right, it will get exchange with the element. So, it is moving on to the next position. In the next step, we will have a different kind of pairing this 1 will be paired with the 0 further to its right.

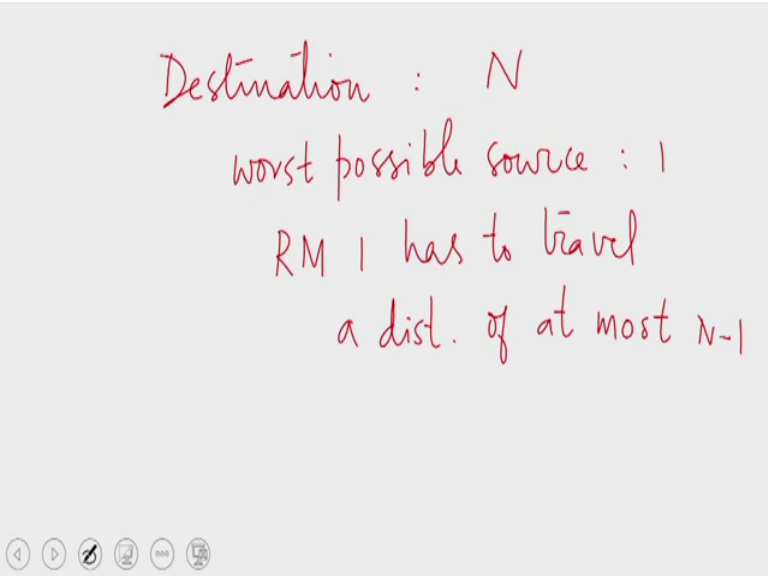
Therefore, it will move once again and once there it will the pairing changes again and therefore, it will be paired with the 0 further to its right and so on. Therefore, what we find is that once the rightmost 1 starts moving, it will keep moving. But then when will it start moving? In our example, we find that the rightmost 1 is at an odd position and therefore, it is paired with an even element to its right which happens to be 0.

Therefore 1 will start the 1 will start moving in the first step itself, but this need not be the case always. When we take a sequence of this form here, the rightmost 1 is this 1 the circled 1 and that is paired with a 1 on the left side. Therefore, in the first step, it will be compared with the 1 on the left side and this will ensure that the element does not move.

So, it is possible that this element does not move in the first step. But if the element does not move in the first step, it will certainly move in the second step because in the second step it is paired with the subsequent 0; 0 which is to its right and then there will be an exchange. This becomes 0 1 and then this 1 will be paired with the next 0 in the next step and it will keep moving to the right.

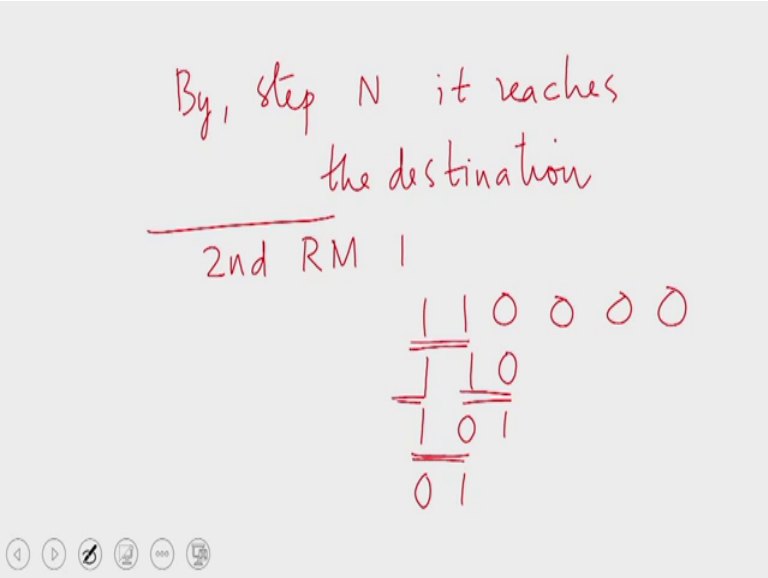
So, what we find is that the rightmost 1 will start moving either in the first step or in the second step. That is if the rightmost 1 fails to move in the first step we know that it will certainly move in the second step and once it starts moving it will keep moving.

(Refer Slide Time: 12:37)



Now, what is the final destination of the rightmost 1? It is supposed to go and occupy the Nth processor. So, its destination is N. And where does it begin? The journey the rightmost 1 could be the first element. If there is only 1 1 in the sequence and that happens to be that the leftmost position. So, the worst possible source for the rightmost 1 is 1 which means the rightmost 1 has to travel a distance of at most n minus 1. In the worst case it starts from position 1 and has to reach position N. Therefore, it has to travel a distance of N minus 1 and again in the worst case, it might not start moving in the first step.

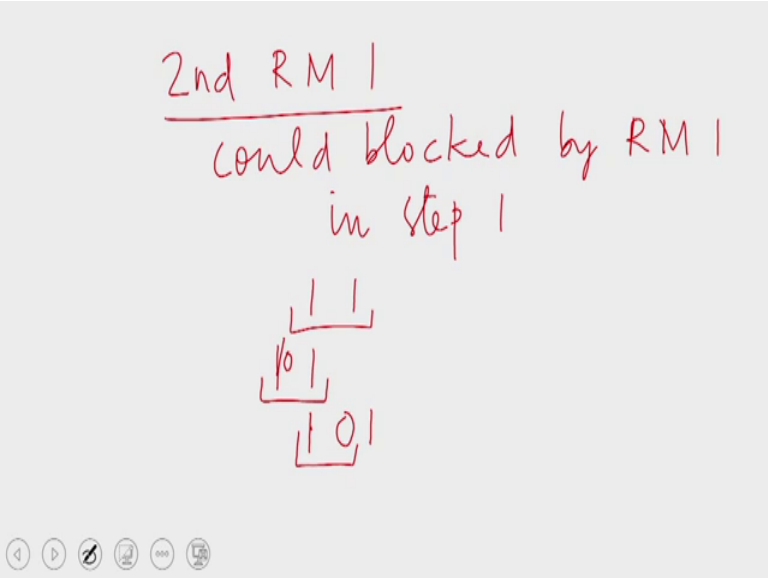
(Refer Slide Time: 13:43)



So, in any case we can say that by step N, it reaches the destination. Of course, the analysis could be tighter, but for our purpose we can say that the element will reach the destination by the N step. Now let us come to the second rightmost 1. It could be that the second rightmost 1 is paired with the rightmost 1 in the first step and therefore, it does not move nor does the rightmost 1 move. In this next step this is paired with an element to its left and therefore, does not change and the rightmost 1 will now change. So, the rightmost 1 has now moved after the second step. Now the second rightmost 1 will be paired with the 0 which appears to its right and therefore, it will start moving.

So, in this case we find that the second rightmost 1 starts moving in the second step, but then it is possible that it does not move in the first step in the second step e even.

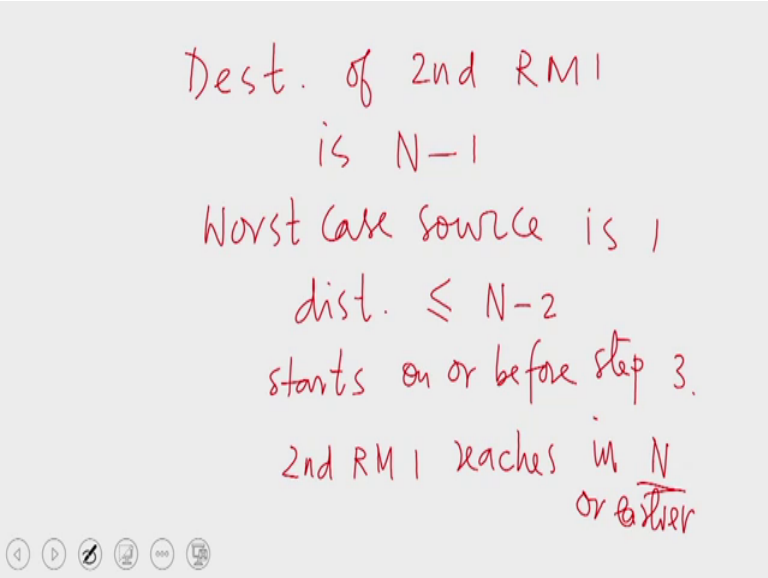
(Refer Slide Time: 15:15)



Yeah the second rightmost 1 could be blocked by the rightmost 1 in step 1 that is the second rightmost 1 and the rightmost 1 are paired in this first step. So, they fail to move neither of the moves. Then let us say in the next step it is paired to the left and therefore, it does not move again this could be the 1 or 0. So, it does not matter the second rightmost 1 fails to move in the second step also. But in the third step now the rightmost 1 has moved away and a 0 has appeared in its place. Therefore, the second rightmost 1 will get paired with that.

Therefore this will now start moving; now in the third step, the second rightmost 1 will start moving. By this time the rightmost 1 has moved away and therefore, there are 0s between the second rightmost 1 and the rightmost 1. Therefore, what we establishes that the second rightmost 1 will start moving by the third step and will keep moving afterwards.

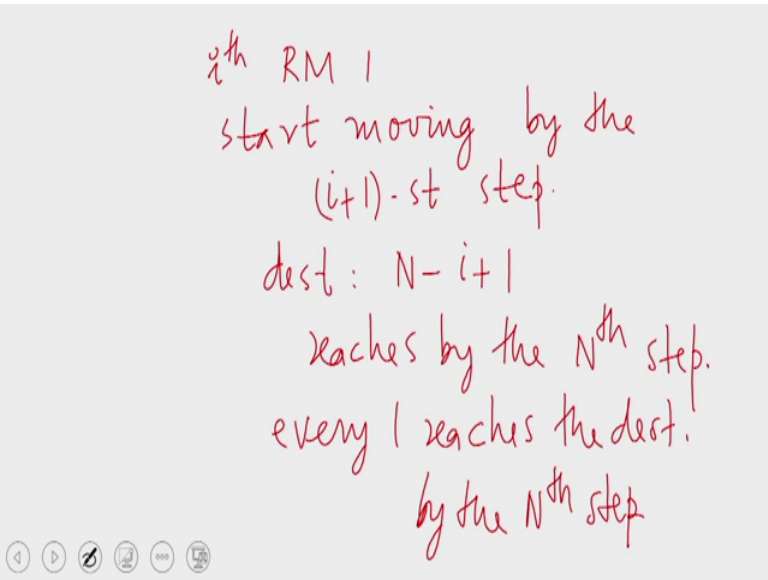
(Refer Slide Time: 16:26)



Now, the destination of the second rightmost 1 is processor number N minus 1 and its worst piece source is 1. Therefore, it has to cover a distance of at most N minus 2.

Now, what we have established is that it will start moving from the third step that is latest by the third step. Therefore, what we know is that even the second rightmost 1 reaches its destination in step N or earlier.

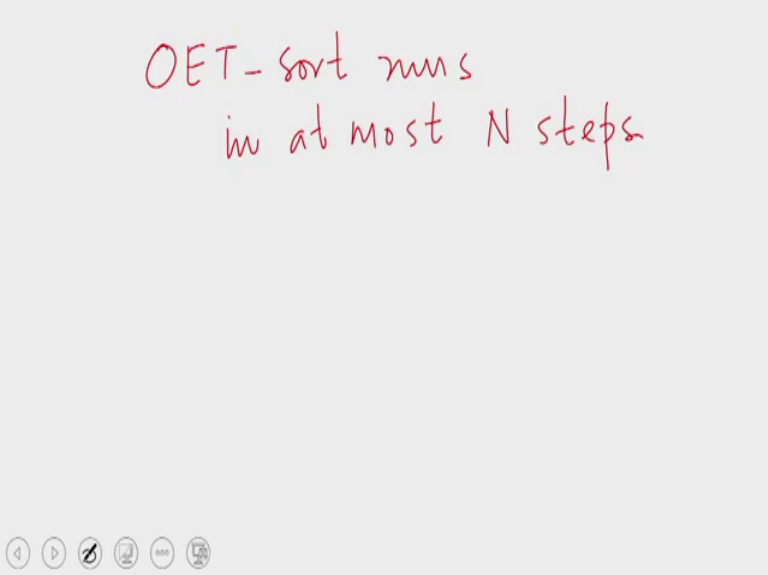
(Refer Slide Time: 17:39)



So, you can repeat this argument for the other positions in general. You can consider the ith rightmost 1 you can show that this will start moving by the; i plus first step and the

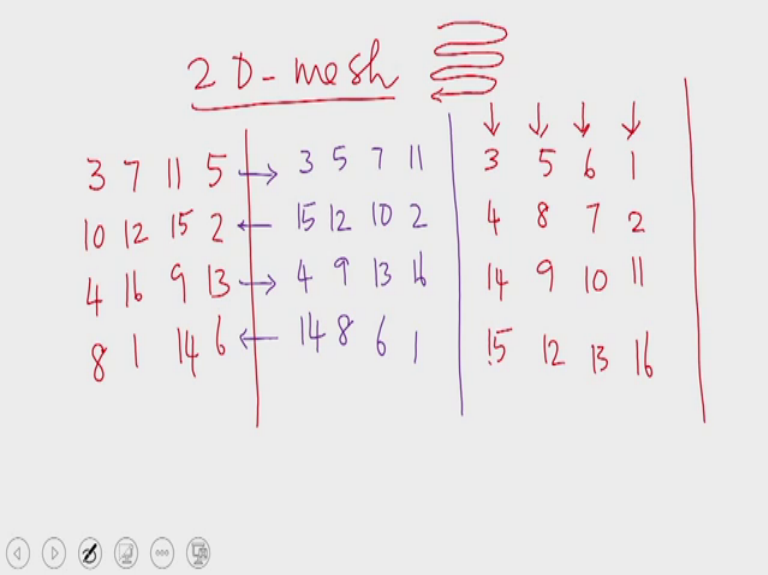
destination for this is N minus i plus 1. Therefore, the I'th right most 1 reaches the destination by the N step. So, what we have establishes that every 1 reaches the destination by the N step. Then by that point in time all the 0s will be to the left of all the 1s; therefore, the input will be sorted.

(Refer Slide Time: 18:59)



So, what we have establishes that the odd even transposition sort runs in N steps. So, it is possible to sort N elements on a linear array in N steps.

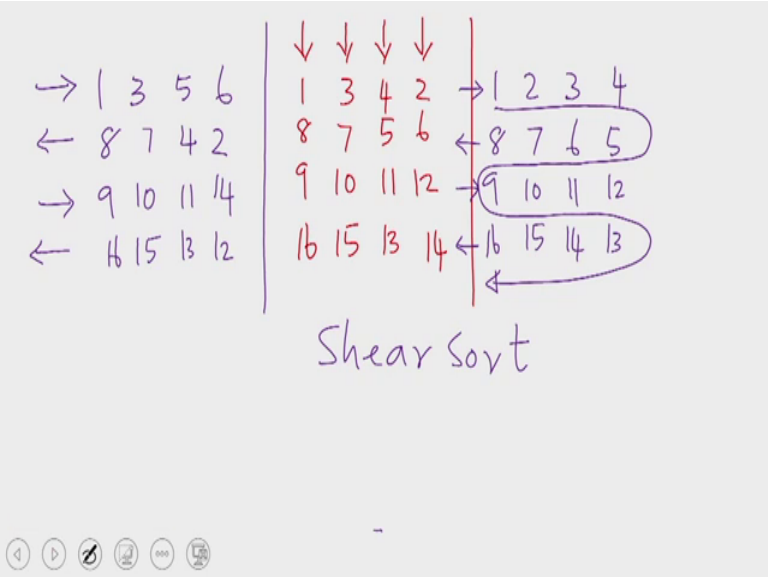
(Refer Slide Time: 19:24)



So, now let us escalate to the higher model, the next model that we consider is the 2 dimensional mesh. Let us say on a 2 dimensional mesh we want to sort the elements in snake like order; just this order; we have seen an algorithm for this already. Let us once again do a recap relation of this algorithm. Let us say we are given this mesh. So, we are given this array 2 dimensional array. What we do is to sort the rows in opposite orders; the odd rows will be sorted from the left to right and the even rows will be sorted from the right to the left.

So, here in the first row we have 3 5 7 11, in the second row we have 2 10 twelve and 15, in the third row 4 9 13 16. This is from the left to the right and then we have 1 6 8 and fourteen from the right to the left. And then in the second step we sort the columns from top to bottom. So, from the first column we get 3 4 14 15 in the second column we get 5 8 9 and 12, in the third column we have 6 7 10 and 13, in the fourth column we have 1 2 11 and 16.

(Refer Slide Time: 21:35)

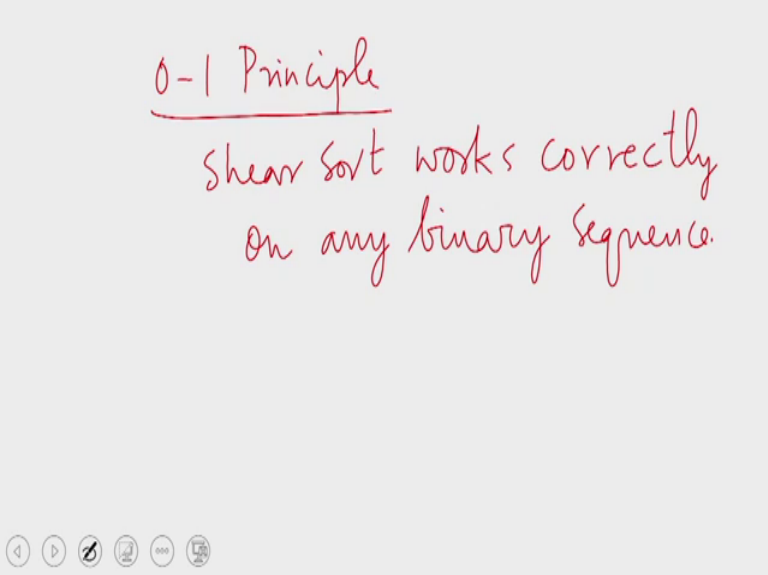


And then once again we sort the rows. When the rows are sorted we have 1 3 5 6 in the first row, 2 4 7 8 in the second row. The second row is sorted from the right to the left. This is how the mesh looks like after the horizontal sorting of the second step. Then in the vertical sorting part of the second step, the columns are sorted. The first column is already sorted and so, is the second column. The third column reads fixing and so, does

the fourth column. And then we come to the third step. In the third step, we have sort the rows again

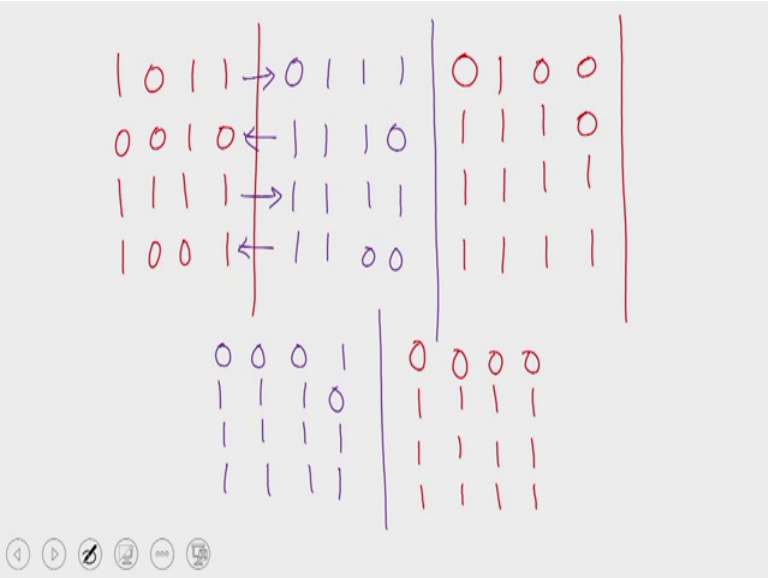
So, the first row gives us 1 2 3 4 in sorted order and then the second row sorted from the right to the left will give us 5 6 7 8 from the right to the left then 9 10 11 twelve and 13 14 15 and 16. Now we find that the elements are in sorted order. This algorithm is called the shear sort, but then given the specification of the algorithm it is not clear why it should terminate and it is not even clear how for how long the algorithm has to be run. So, let us now prove the correctness as well as the time complexity of this algorithm.

(Refer Slide Time: 23:43)



So, once again we will invoke the 0-1 principle. Once we invoke the 0-1 principle all we have to show is that shear sort works correctly on any binary sequence and even the analysis we need to do only on binary sequences. Because if the algorithm works correctly on binary sequences of with the time complexity of t of N, then it will work correctly on any linearly ordered set in the same time complexity.

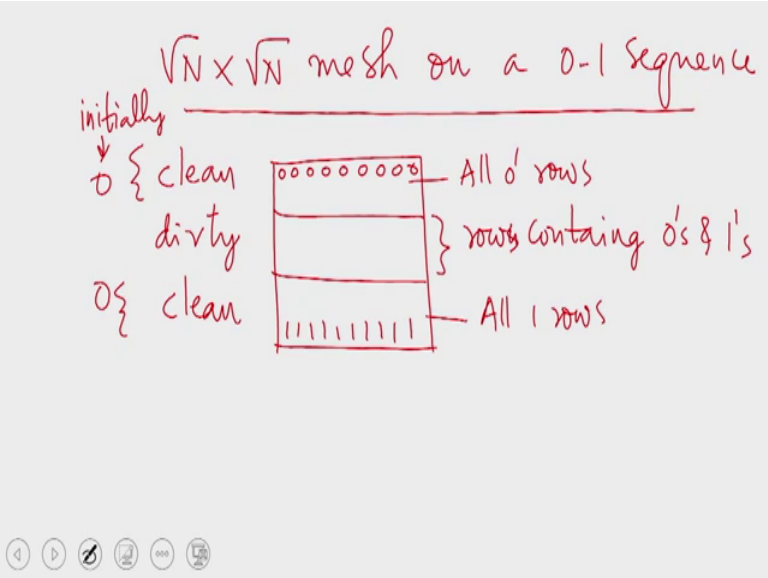
(Refer Slide Time: 24:33)



So, let us see how this will work on a binary input. When we sort the rows we get 0 followed by three 1s, this is being sorted from the left to the right. Then in the second row we have again 0 followed by three 1s in the third row it is all ones anyway. So, it is already sorted, then we have two 0s followed by two 1s and then when we sort the columns we have the columns are all sorted from the top to the bottom.

So, now in each column we find that all the 0s are at the top and all the ones are at the bottom and then once again we sort the rows. The second row is already sorted so, are the third row and the fourth row. Then once again we sort the columns only the last column needs fixing. We find that the sequence is already sorted. So, let us see how the algorithm behaves on a general sequence a general binary sequence.

(Refer Slide Time: 26:25)

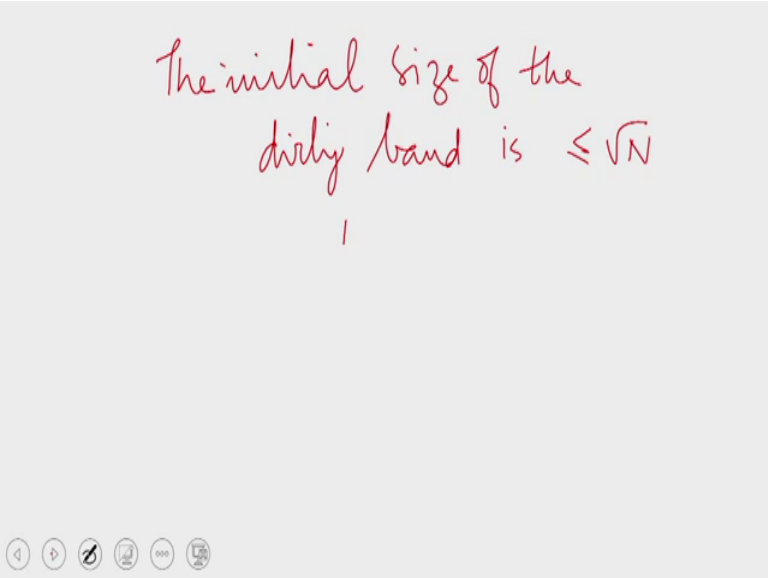


So, we will take a root N by root N mesh minus 0 1 sequence. We want to show that shear sort works correctly on root N by root N mesh on which we are given a 0 1 sequence of length N. So, the elements are distributed arbitrarily over the processes of the mesh.

So, at the beginning of any intermediate iteration, we can divide the rows of the mesh into three groups. At the top of the mesh we have rows that are filled with all 0 rows and all 0 row is 1 in which every element is a 0. At the bottom, we have all 1 rows and in between we have rows containing both 0 s and 1s. We will call the top band clean and similarly the bottom band is also clean, but the middle band we say is dirty. So, a clean row is 1 in which every element is the same either or are 0 or all are 1. So, we assume we have a clean band at the bottom as well as at the top and in the middle we have a dirty band a row is dirty if it has both 0s and 1s.

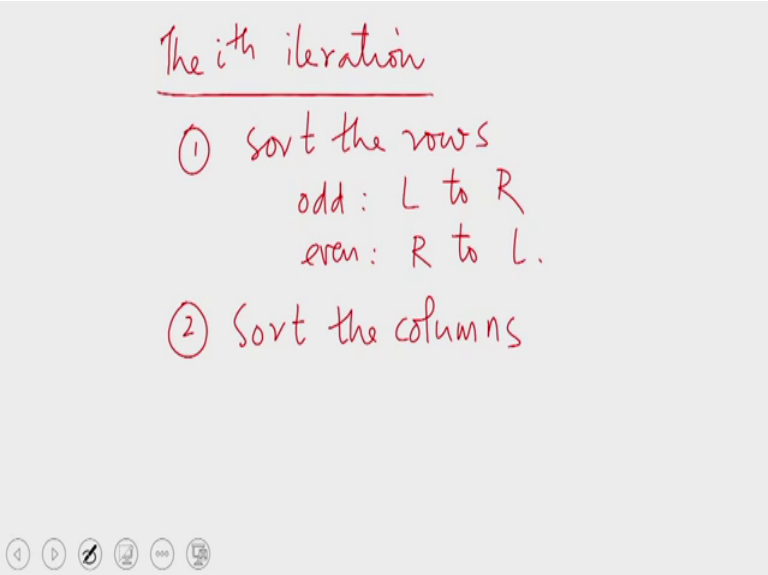
So, of course, any the in the beginning the clean bands could be of size 0. Initially there could be no clean row at all. It is possible that there is not a single clean row to begin with. Therefore, the size of the clean band and the clean band at the top as well as at the bottom could be 0.

(Refer Slide Time: 28:46)



What it means is that the initial size of the dirty band is less than or equal to root N initially that is we do not make any assumption that we have clean rows at the beginning; it is possible that every single row is dirty. So, the size of the dirty band could be root N in the worst case to begin with.

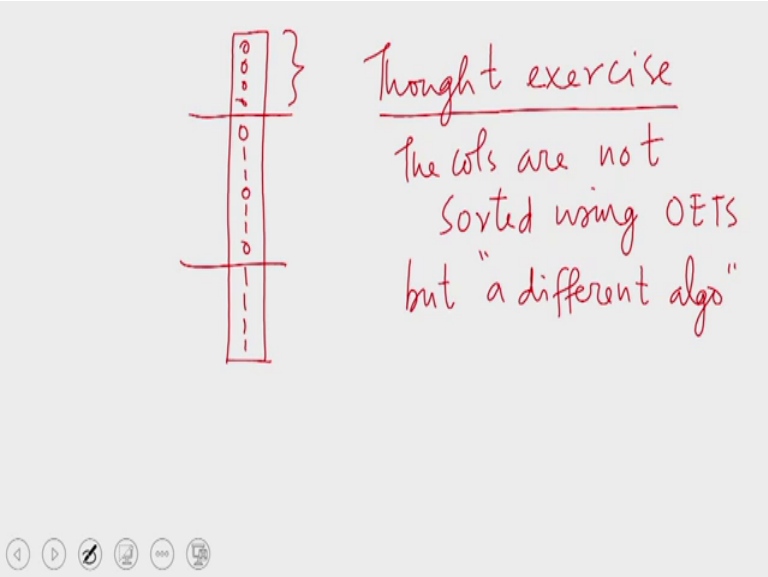
(Refer Slide Time: 29:25)



Then let us see what the algorithm does. In the first iteration or in general in any ith iteration in the ith iteration in the first step of the iteration, we sort the rows. The odd

rows are sorted from the left to the right; the even rows are sorted from the right to the left. Then when we come to the second step, we sort the columns.

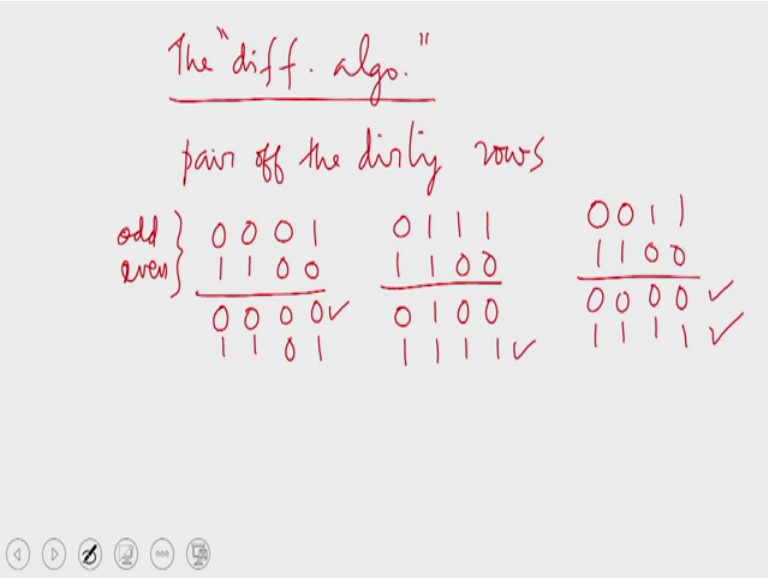
(Refer Slide Time: 30:12)



Now let us consider a column here in the column, we have a clean portion at the top which comes from the clean band at the top; all its elements are 0 that is because in the top clean band every single element is 0. Similarly at the bottom we have a clean band every element of which is 1. Therefore, when the column is being sorted; there will be no change to these elements. In any case the 0s are supposed to be above the 1s. Therefore, when a column is sorted the elements which are which are in its clean positions, we assume do not change their positions.

So, only the dirty band elements will get sorted. Now let us assume that the sorting is done using a different algorithm. As a thought exercise let us assume that the columns are not sorted using the odd even transposition sort, but at different algorithms which I will describe in a moment.

(Refer Slide Time: 31:56)



So, if the columns were to be sorted using a different algorithm and is this different algorithm happens to be the following; mind you this is only a thought exercise. So, we are now invoking a different algorithm for sorting the columns what this does is the following its pairs of the dirty rows. So, let us see what sort of scenarios can happen when we pair of dirty rows

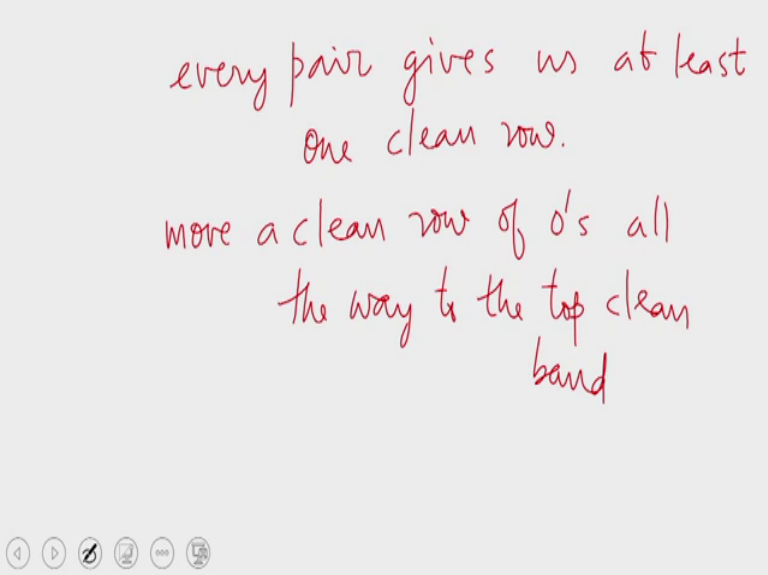
Let us consider two dirty rows of size four each. Now these dirty rows are already sorted and they are sorted in different directions for odd rows and even rows. So, when we pair them off, we pair adjacent rows together. Therefore, when we consider the next row the pair of this row, it would be of the sort. So, let us perform a column wise compare and exchange between these 2 rows. So, this is a pair of dirty rows that are adjacent to each other.

They are sorted from the left to the right for the odd rows and the right to the left for the even rows. So, I have done an odd even pairing. So, this is an odd row and this is an even row. It could we will have in the other way; when we look at the dirty band, we take the first row and the second row and pair them off together. If the first row happens to be an odd row of the matrix, then we will get an arrangement like this if the first row happens to be an even row of the matrix; then we will have a different arrangement. It would be just a mirror image of this that is an upside down image of this.

Now when we sort these two rows individually that is we sort them in isolation then for each column when we perform a compare exchange what we get is this. We perform a compare exchange within each column, but restricting ourselves to within the pairs. So, within these pairs we have such compare and exchanges. What we find is that one of the rows turns completely clean. So, in this case we have more 0 s than 1s. Therefore, we get one clean row of 0s. On the other hand if we had an arrangement of this sort when we compare and exchange within the columns, we will get a clean row of 1s and one dirty row.

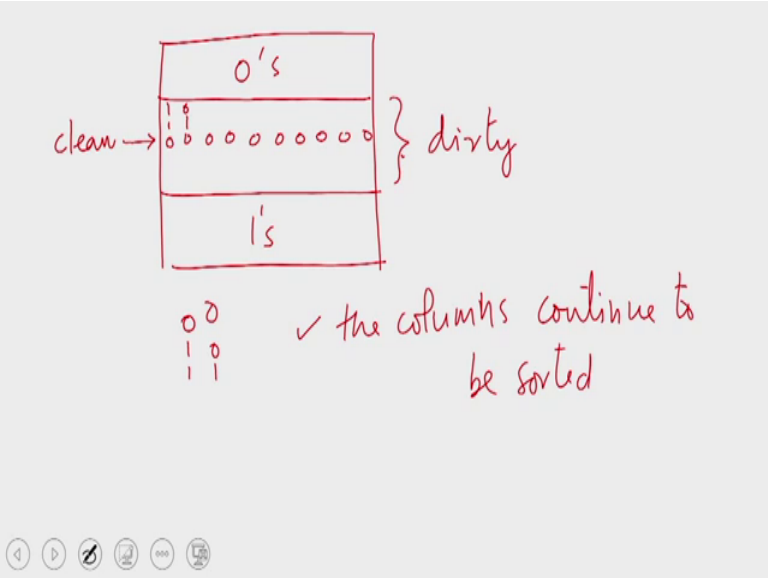
If the number of 0 s and number of ones were exactly the same; then after I perform a compare and exchange I would get to clean column clean rows. So, in the different algorithm for sorting vertically column wise what we do is this we pair of the dirty rows adjacent rows are paired into one group and within each group will perform a compare exchange within the columns as a result we will get at least one clean row out of every pair.

(Refer Slide Time: 35:40)



So, what we know is that every pair gives us at least one clean row and then let us move a clean row of 0 s all the way to the top clean band

(Refer Slide Time: 36:29)



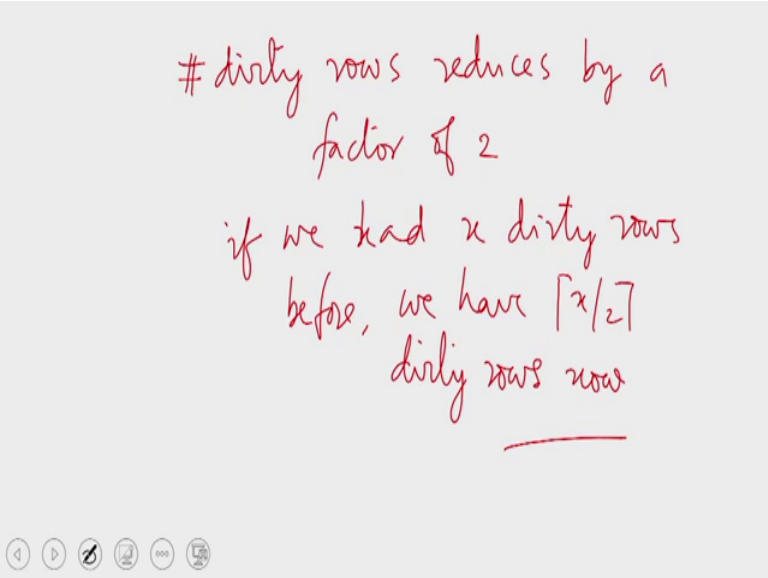
That is in the matrix we have a clean band at the top which is all 0 s, a clean band in the bottom which is all 1s and in the middle we have the dirty band. In the dirty band, now we have got a row which contains all 0s. So, this is a clean row a clean row within the dirty band.

Let us say we move this clean row to the top by pulling all the elements above these 0 s by one position down. So, if we had two 1s above we will now come to this arrangement 0 1 1. Those two 1s are pulled below the 0. Iif we had a 0 1 above this, we will pull them down and the 0 will go up. So, in this fashion if we push the clean row all the way to the top, we find that every single column still continues to be sorted.

All that has happened is that these 0 s has bubbled to the top one single 0 in every column has bubbled to the top. So, the columns have compressed by one single position. So, the columns continue to be sorted this we did for one single clean row, but when we paired off the dirty rows and compared and exchanged within each pair we got one cleaned row from every single pair.

Now, let us do the same action for every single clean row every single clean 0 rows will be moved to the top and every single clean one row will be moved to the bottom. Then what remains in the middle will be the dirty rows by the same argument, we show that the columns will continue to be sorted when we do this

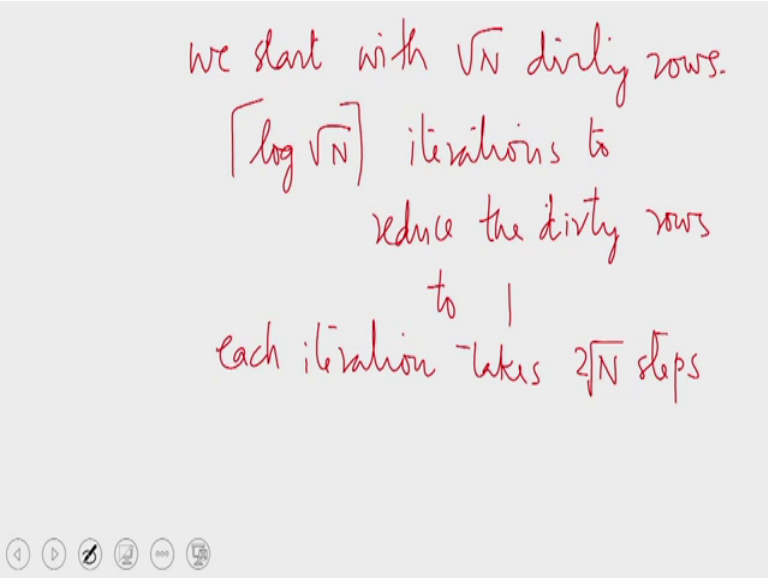
(Refer Slide Time: 38:27)



But then this establishes that the number of dirty rows reduces by a factor of 2. If we have x dirty rows before, we have ceiling of x by 2 dirty rows now, but then we have used not the odd even transposition sort as shear sort is supposed to use, but we have used a different sorting algorithm. But does it matter this was the result of applying some sorting algorithm, but every correct sorting algorithm should produce the same output.

Therefore in the second step of our iteration instead of using this different sorting algorithm; if we had used shear sort we would get exactly the same result that is even shear sort would ensure that the number of dirty rows reduces by a factor of 2 at the end of the iteration.

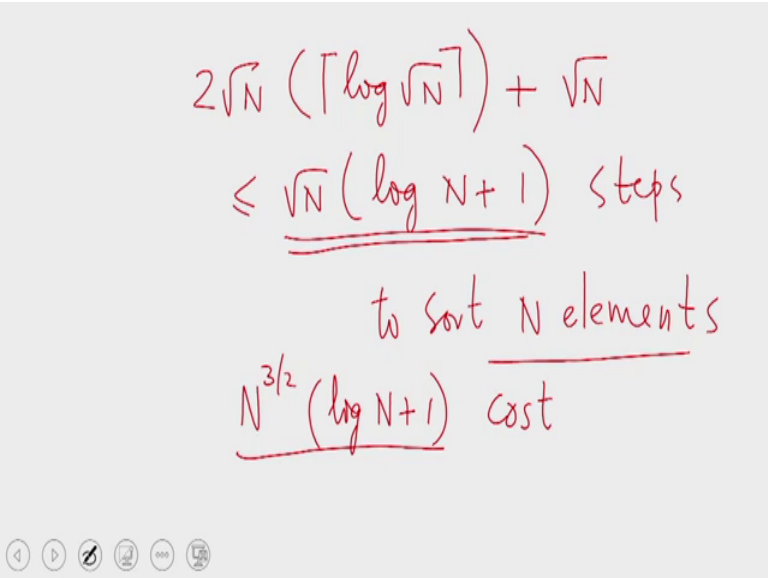
(Refer Slide Time: 39:55)



So, each iteration reduces the number of dirty rows by a factor of 2 we start with the root and dirty rows. So, the algorithm would require log root N iterations to reduce the dirty rows to 1. The number of literals to be reduced to 1 will require log row 10 iterations and each iteration takes 2 N steps. If we use odd even transposition sort that is we use odd even transposition sort horizontally in the first step of the iteration.

In the second step of the iteration, we use odd even transpositions sort vertically. So, the total time taken would be 2 root 2 N 2 root N steps.

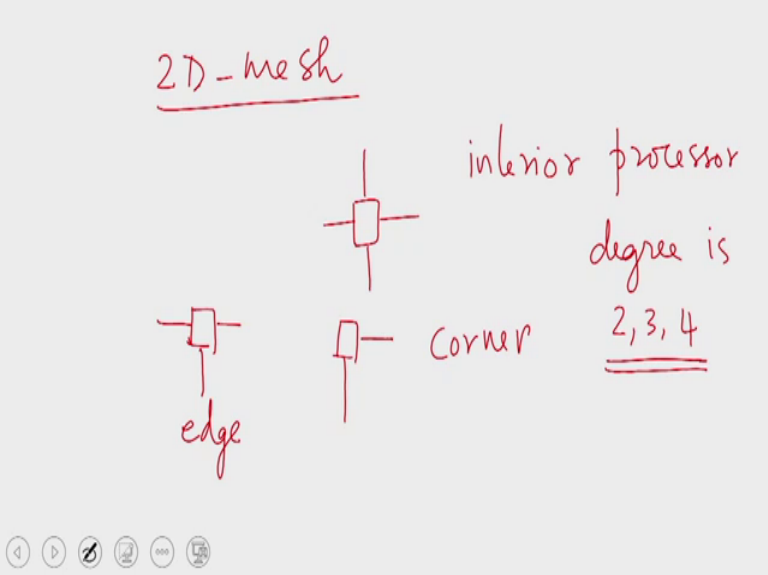
(Refer Slide Time: 41:20)



Therefore, the overall over all time complexity of the algorithm would be 2 root N times ceiling of log root N plus we will require 1 round of horizontal odd even transposition sort. So, this we can show is less than or equal to root N times log N plus 1. Therefore, this is the time complexity of the shear sort algorithm; what we have established is that when we begin with root and dirty rows in the begin with there is the worst possible in any case if we begin with root N dirty rows. Then after so many iterations the number of dirty rows will be 1. Therefore 1 extra odd even transposition sort done over the rows will ensure that the mesh is completely sorted.

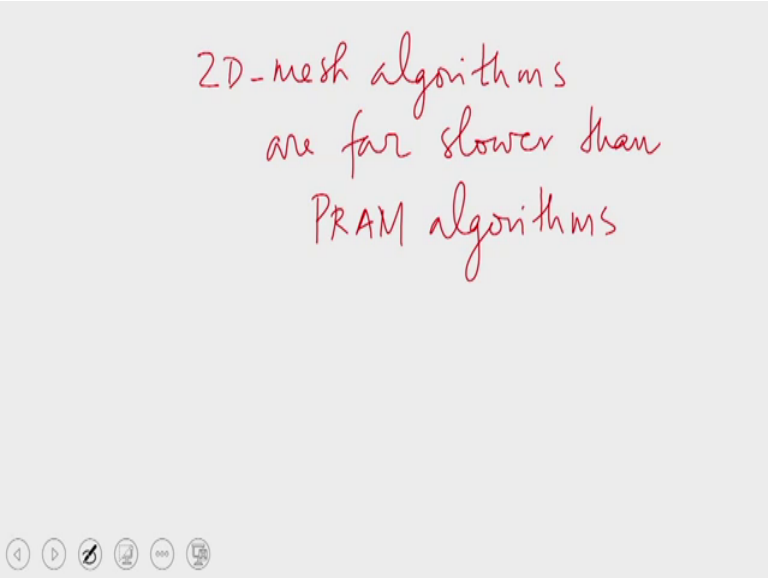
So, the algorithm runs in square root of N times log N plus 1 steps to sort N elements. What is the cost of the algorithm then? We have N processors. So, N power 3 by 2 times log N plus 1 is the cost of the algorithm which is surely suboptimal because the sequential time complexity of the problem is order of N log n

(Refer Slide Time: 43:00)



But then the model that we use is the 2 dimensional mesh. In this mesh each processor has at most 4 neighbors. This is the case with an interior processor. A processor along the edge will have 3 neighbors along, the top edge will have 3 neighbors and a corner processor will have only 2 neighbors. That is the degree of a vertexes 2 3 or 4. What this means is that a processor is capable of communicating with only a small number of peers in every single step. Therefore, we would expect that an algorithm that runs on a 2 dimensional mesh will be considerably slower than an algorithm that runs on PRAMs.

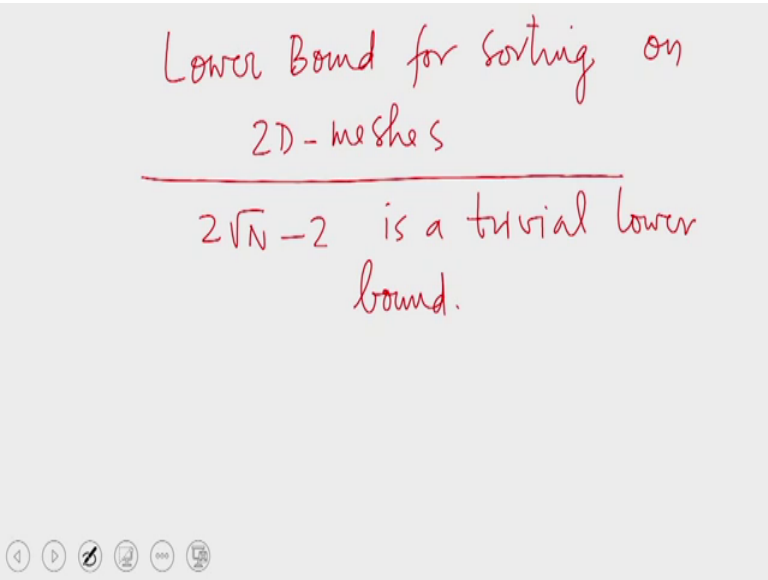
(Refer Slide Time: 44:13)



This is because in PRAMs the presence of a shared memory allows for a rich design of communications in every single step whereas, in the case of a 2 dimensional mesh a processor can communicate with at most 4 other processors in any single step.

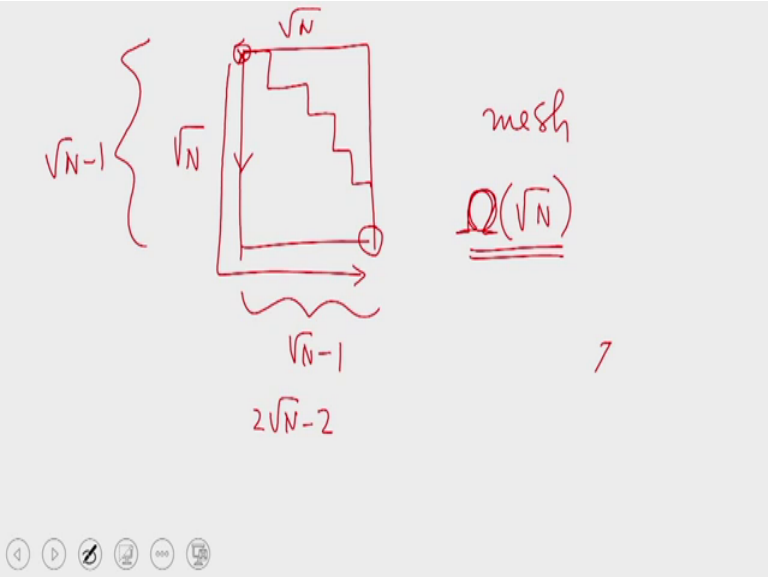
So, that is a tough constraint. Therefore, that makes designing algorithms on 2 D mesh a lot more challenging we certainly cannot achieve poly logarithmic time bounds on 2 dimensional meshes.

(Refer Slide Time: 45:13)



Now let us see a lower bound for sorting on 2 dimensional meshes a trivial lower bound is 2 route N minus 2.

(Refer Slide Time: 45:53)



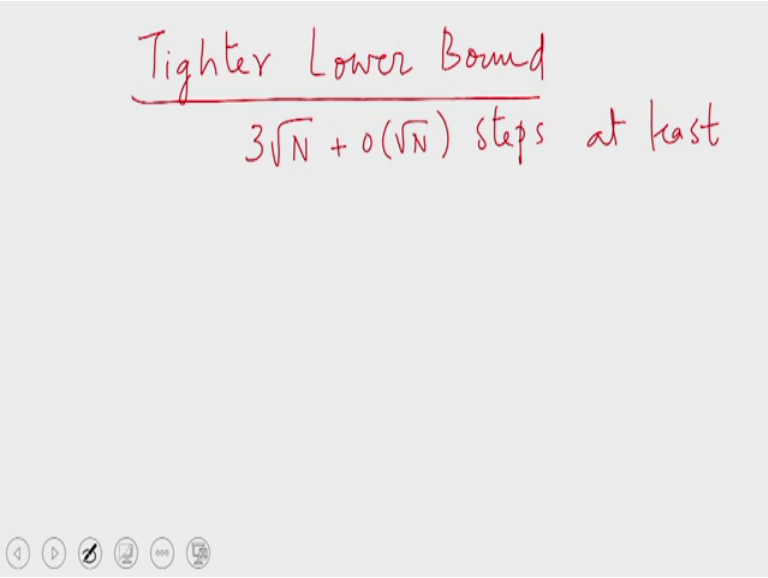
Why is this? Let us consider a root N by root N mesh and we want to sort N items which are pre loaded into the mesh 1 element per processor using the mesh. So, let us say the item that we place in the top left corner is the item which is destined to the bottom right corner. That is we place an element x here if we manage to sort the elements in snake like order as we did in the case of a shear sort, then let us say x should end up in the bottom right corner. Then during the algorithm this element will have to travel from the top left corner to the bottom right corner.

Now, what is the distance from the top left corner to the bottom right corner. There are multiple paths, but let us take 1 single path all paths have the same length. So, the length of this path you find that is you can decompose this into two paths. You start from the top left corner and come to the bottom left corner the distance for that is root N minus 1. Coming directly downwards there are root N nodes in the mesh along this edge.

So, starting from the first processor reaching the root nth processor will require traveling a distance of root N minus 1. So, the element x has to traverse this distance and then it has to traverse an additional distance of root N minus 1 traveling along the horizontal edge. So, the distance is 2 root N minus 2; of course, this need not be the only path that x can take.

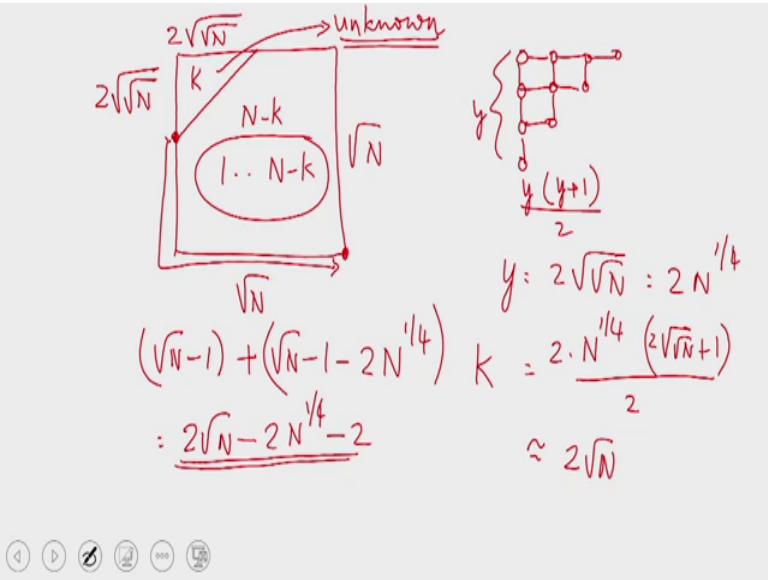
For example x can take part of this sort, but then we know that the total horizontal distance has to be root N minus 1 and the total vertical distance it travels must also be root N minus 1 because it is not possible for an element to move along the diagonal. Therefore, element x will necessarily have to travel a distance of 2 root N minus 2 and therefore, that is a lower bound. So, in any case you did not expect to sort in smaller root N time. Sorting is going to take omega root N time omega root N is a lower bound for sorting N elements on a root N by root N mesh, but let us try to get a tighter lower bound.

(Refer Slide Time: 48:34)



What we are going to show is that we require 3 root N plus smaller root N steps at least we cannot do better than this what we want to show.

(Refer Slide Time: 49:04)



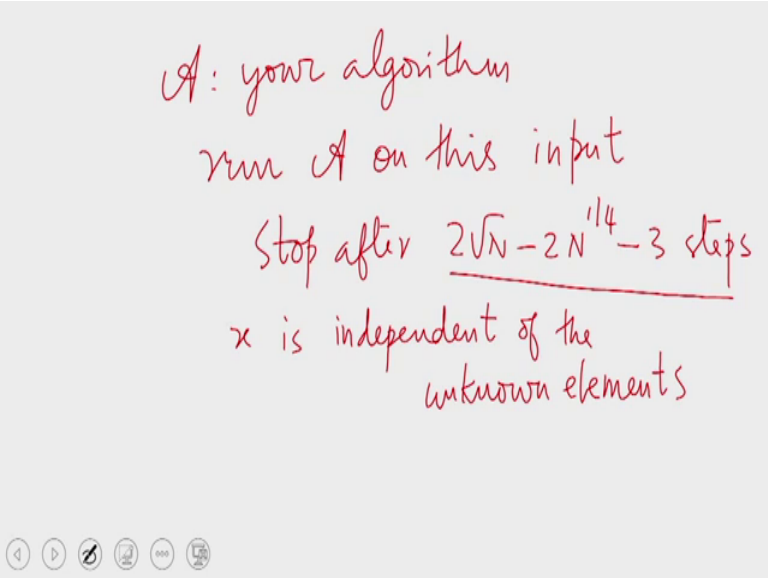
To prove this we consider the root N by root N mesh. In this root N by root N mesh, we isolate a corner of size 2 power foot of root N 2 times root of root N and 2 times root of root N that is we identify a corner of this sort. In particular in a 2 dimensional mesh when I identify a corner like this the number of processes in this would be if I have taken y elements here, then that will be y into y plus 1 divided 2.

So, if y equals twice square root of root N which is twice N power 1 by 4, then twice N power 1 by 4 into twice N power 1 by 4 plus 1 divided by 2. It is approximately 2 root N. Let us say this quantity is K; this is approximately 2 root N. So, there are K processes within this triangle and in the remaining we have N minus K processors. Now this is an adversarial argument, you have found a fast algorithm for sorting on a 2 dimensional mesh.

I want to make your algorithm do badly for the purpose of making your algorithm do badly I am going to cook up an input. I will cook up an input in this fashion, I will place the elements 1 to N minus K in this part. In this part, I will place the numbers 1 to N minus K anyhow it does not really matter how they are distributed and then the elements I am going to place in this top triangle which has K processors are all unknown at the moment.

I will fix them later. So, at the moment they are all unknown

(Refer Slide Time: 51:40)



Then what I do is this I take your algorithm A; A is your algorithm. I run A on this input and stop the algorithm after 2 root N minus 2 N power 1 by 4 minus 3 steps. I will stop the algorithm after these many steps. Now let us look at the mesh this is a root N by root N mesh. Here I have identified 2 power N by 2 into N power 1 by 4 processors in the first column. Let me consider the last of these processors the lowest of these processors the distance from this processor to the bottom right corner is going to be root N minus 1 horizontally plus root N minus 1 minus 2 N power and by 4 vertically.

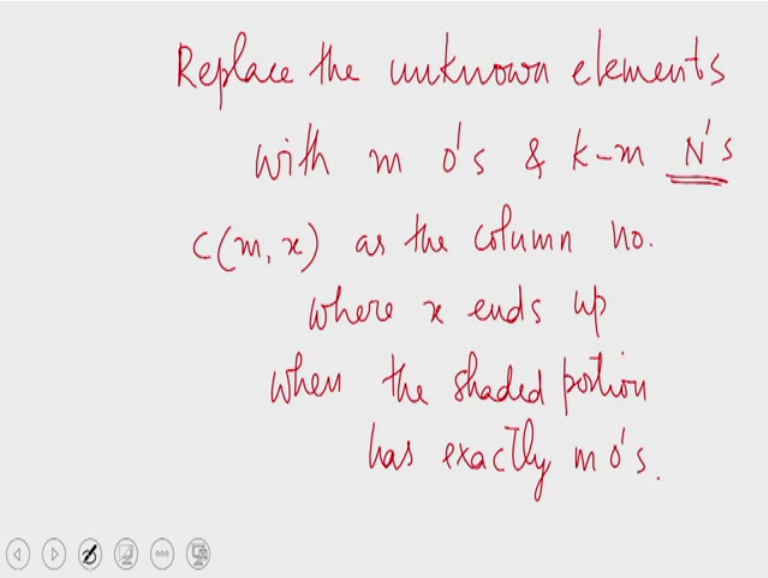
So, this is 2 root N minus 2 N power 1 by 4 minus 2. So, the step number that I have chosen for stopping your algorithm is 1 less than this. So, I have stopped your algorithm on this input after running it for 2 root N minus 2 N power and by 4 minus 3 steps. So, the algorithm is now stopped. Let us look at the bottom right corner. The element which appears here is let us say element x; let me call this element x.

Now based on this element x, I will choose the contents for the top left triangle. What we know is that this element x the identity of x is independent of the unknown elements that is because a message from the unknown elements could not have reached the bottom right corner within. So, many steps the distance from here to from the shaded portion this is the shaded portion let us say, the unknown elements. The distance from the unknown elements to the bottom right corner is 2 root N minus 2 N power and by 4 minus 2 and we have spent 1 less step than this 1 fewer step than this. Therefore it is not possible for

us to have taken any information from the unknown elements to the bottom right corner. What it means is that the identity of x is independent of the unknown elements.

So, even if we change the unknown elements and rerun the algorithm for exactly these many steps x would still end up exactly at this place that is the bottom right corner.

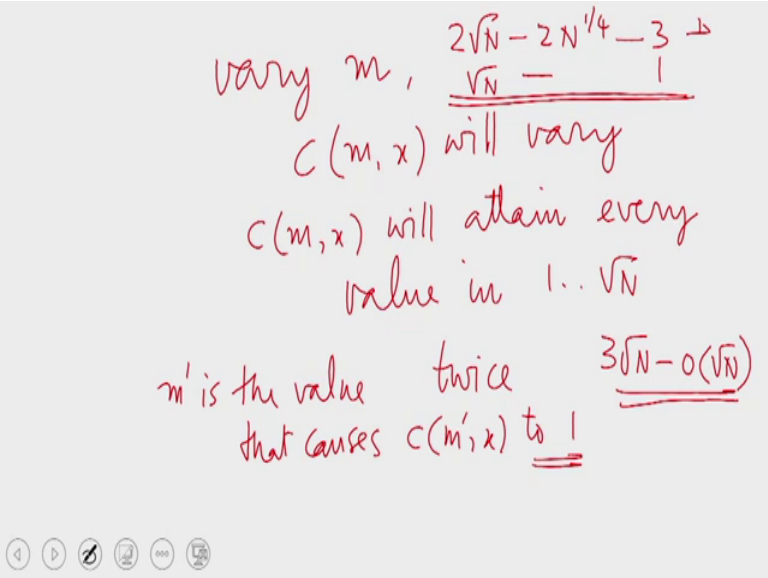
(Refer Slide Time: 55:06)



Now, what I am going to do is this replace the unknown elements with m 0s and K minus m N's. Then when you run the algorithm A on this input again; if we stop the algorithm after exactly these many steps, x would be exactly where it is now; it will be in the bottom right corner. But then if you continue executing the algorithm, x would move from this position that is because now the unknown elements are going to put influence this x.

So, let me define c m of x c m x c of m x as the column number where x ends up when the shaded portion which is the set of unknown elements now has exactly m 0 s with the rest being all ns. So, the remaining are all larger than the elements that we place in the unshaded portion. We have put m 0 s in the shaded portion along with these largest elements, then these m the value of m will decide where x will end up finally.

(Refer Slide Time: 56:58)



So, vary m c m x will vary in particular when I vary m by 1 c m x will vary by 1. So, if I continue doing this; so, if I continue varying m c m x will attain every value in 1 2 root N. In fact, twice because the total size of the shaded portion is greater than 2 root N by varying m, I can make sure that c m x will attain every value in the range 1 to root N. This will happen; in fact, more than twice at least twice. All we need is one particular value; all we need is one particular value.

Suppose m prime is 1 value; suppose m prime is the value that causes c m prime x to be 1. So, if I put m prime 0s in the top left corner, then the element x which appears at the bottom right corner at the end of these many steps will have to travel all the way to the first column. What it means is that the total number of steps taken is in other words we are going to take 3 root N minus smaller root N steps for sure in sorting the elements. So, this is a title low bound and the one we saw earlier. In the next class, we will see an algorithm which almost matches this lower bound. So, that is it from this lecture; hope to see you in the next lecture.

Thank you.