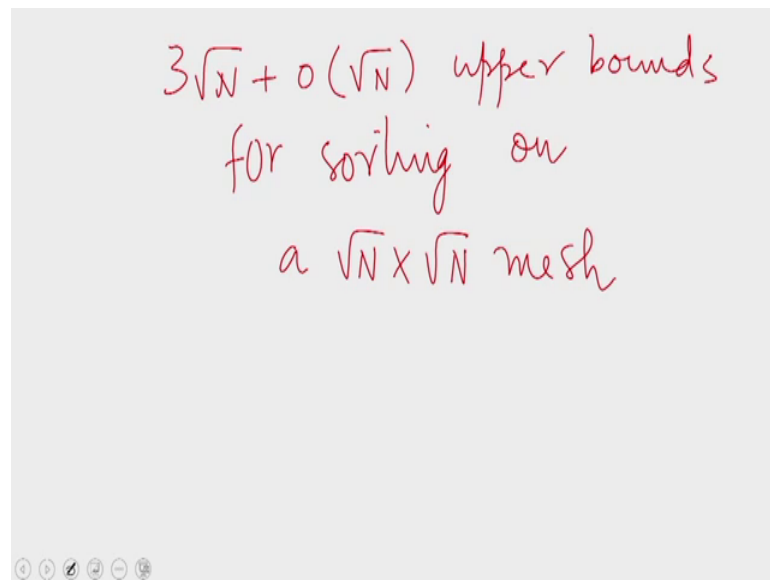


Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 25
Sorting on a 2D mesh

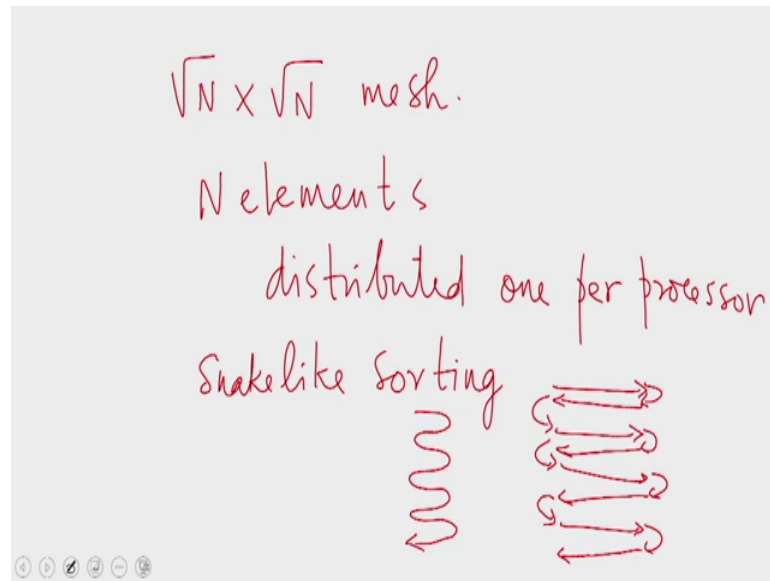
Welcome to the 25 lecture of the MOOC on Parallel Algorithms in the previous lecture. We saw a lower bound for sorting on two dimensional meshes the lower bound was $3\sqrt{N}$ minus small $o(\sqrt{N})$ open today, we shall see an algorithm which nearly matches the lower boundary.

(Refer Slide Time: 00:50)



So, today we shall see an algorithm that transcends $3\sqrt{N}$ plus smaller $o(\sqrt{N})$ steps for sorting on a \sqrt{N} by \sqrt{N} mesh.

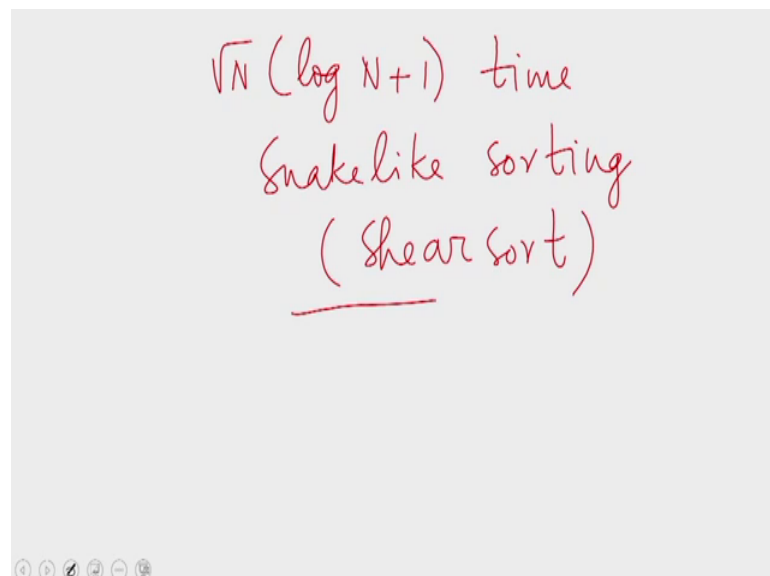
(Refer Slide Time: 01:22)



So, we assume that we have a root N by root N mesh, we have N elements distributed one per processor. We want a snake like sorting in snake like sorting the rows are sorted from left to right in odd rows. And the rows are sorted from the right to left in even rows and the elements of a row are all smaller than the elements of the next row.

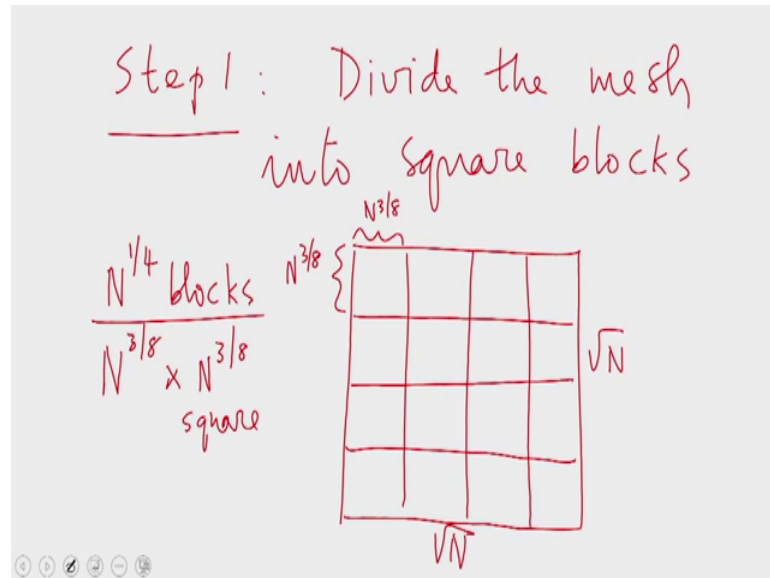
So, this is how the elements will be ordered. Therefore, we call it the snake like order. So, on the two dimensional mesh we will want to arrange the elements in this order. Finally, when the sorting is done the elements will have to be in the snake like order.

(Refer Slide Time: 02:47)



So, in the last class we all, we saw an algorithm that runs in the root N times log of N plus one time this algorithm was called Shear Sort. So, in our algorithm of today shear sort will be used as a subroutine; so this algorithm we saw in the last class.

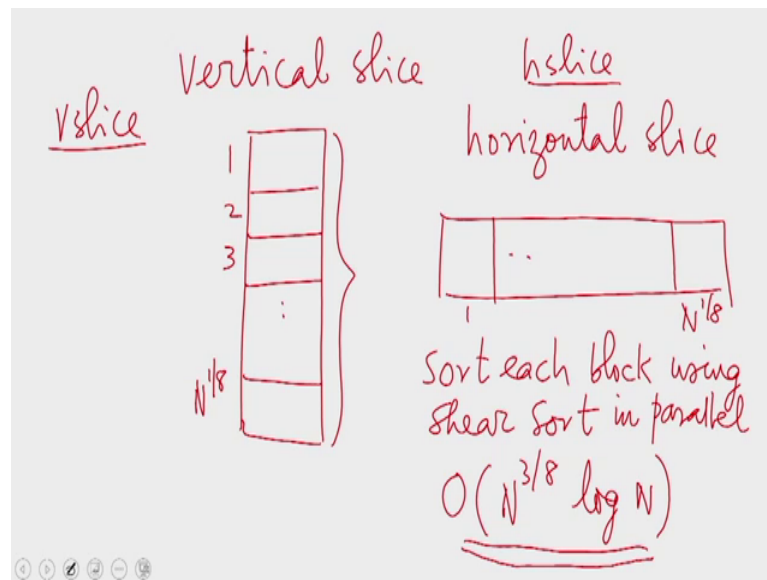
(Refer Slide Time: 03:21)



So, in the present algorithm what we do is this in step 1, we divide the mesh into square blocks that is the root N by root N mesh is divided into square blocks. So, that the size of each square is N power 3 by 8. So, we have N power 1 by 4 squares each blocks and N power 3 by 8 by N power 3 by 8 square.

So, the mesh is divided into square blocks of the sort of course, dividing this does not require any cost, because all that a processor has to do is some local computation to decide in which block it belongs.

(Refer Slide Time: 04:50)



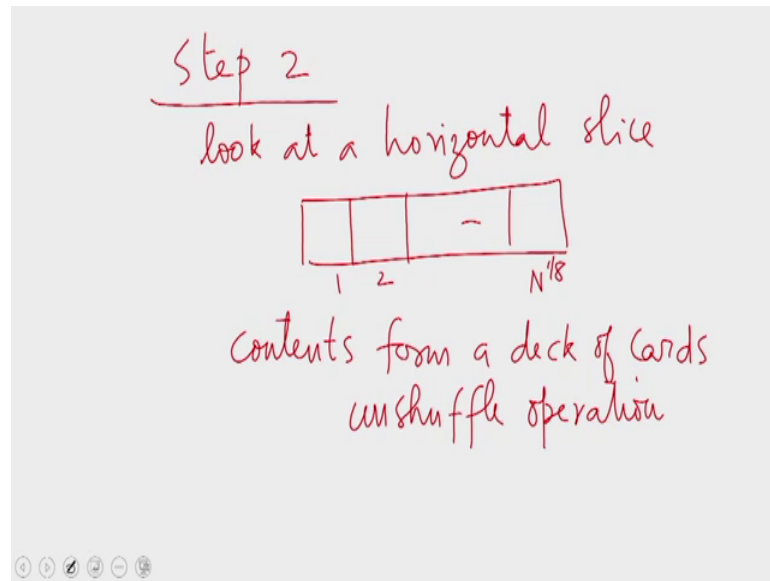
Now, let me define a couple of terms I define what is called a vertical slice. Vertical slice is a set of N power 1 by 8 blocks that appear one above the other. A vertical slice is a set of blocks that appear like this.

Similarly, a horizontal slice is a set of blocks that appear consecutively horizontally. So, we will use the short form h slice for a horizontal slice and v slice for a vertical slice. So, in the first step, we have divided the array into square blocks of the sort. And then in the second step hold on and then we also sort each block using shear sort.

We know that this takes order of N power 3 by 8 times $\log N$ that is from the analysis of shear sort that we saw in the last class. In the last class, we saw an algorithm called shear sort which sorts N elements distributed over a root N by root N mesh in root N times $\log N$ plus 1 steps, but here instead of a root N by root N mesh. We are considering an N power 3 by 8 by N power 3 by 8 mesh which is 1 square block.

So, all the square blocks are sorted in parallel. So, these are done in parallel the time required. Therefore, is order of N power 3 by 8 times $\log N$, N power 3 by 8 is the dimension of 1 square. And of course, it should be \log of N power 3 by 4, but which is the same as order of $\log N$. Therefore, the overall time complexity of this step is order of N power 3 by 8 \log .

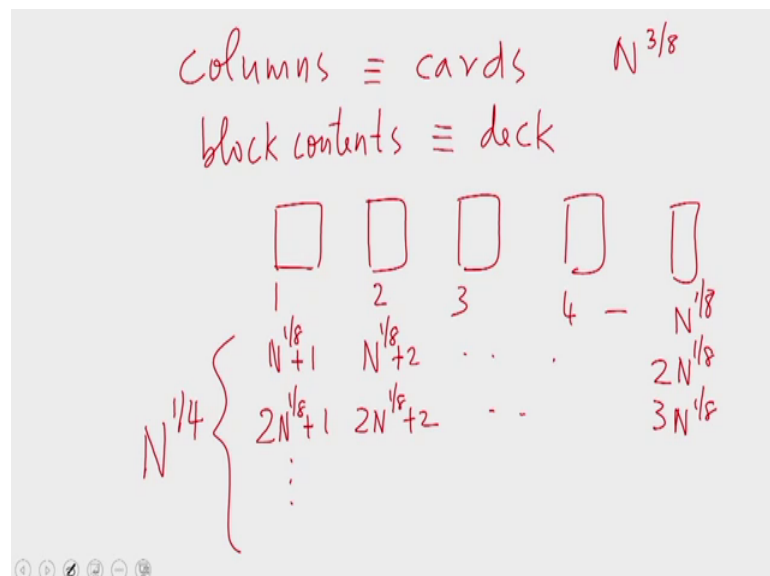
(Refer Slide Time: 07:23)



Now, we come to the second step, in the second step what we do is this we look at one horizontal slice, this horizontal slice consists of N power 1 by 8 blocks. Let us separate the contents of the blocks from the blocks themselves. So, let us separate the contents of the blocks from the blocks themselves imagine that the contents of a block, a deck of cards our idea is to redistribute this deck of cards using an unshuffle operation.

In particular, let us consider the first block. We take this first block and redistribute this the columns of this, let us say form the cards of the deck.

(Refer Slide Time: 08:53)

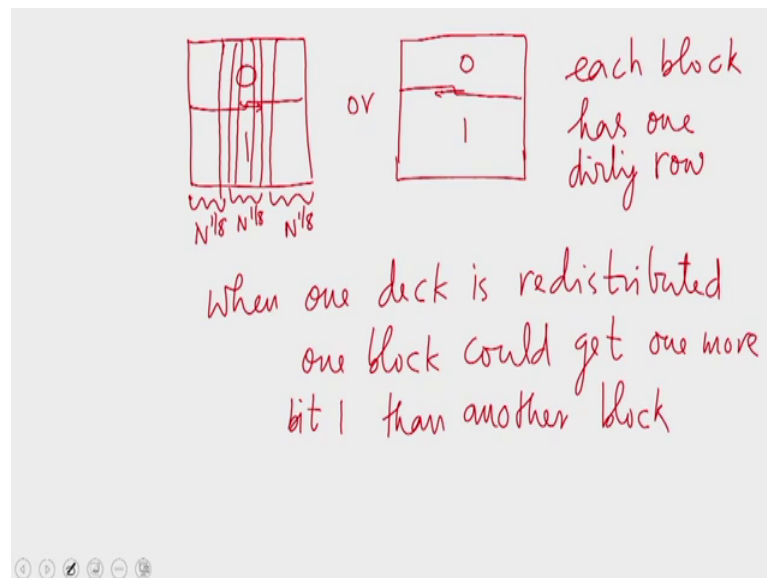


So, the columns are the cards and the block contents that is all the N power 3 by 8 columns put together. We will form a deck, we want to redistribute the columns in those fashion we take the first block and then we distribute the columns to the blocks themselves.

The first card goes here, the second card goes here, the third card goes here and so on, but then the number of cards then is N power 3 by 8, but there are only N power 1 by 8 blocks. Therefore, you will soon come to the end then you will start again. So, cards 1 to N power 1 by 8 are distributed to the blocks and then card number N power 1 by 8 plus 1 will again go to the first block card number N power 1 by 8 plus 2 will go to the second block and so on.

And likewise, we have N power 1 by 4 rounds of cards distribution. We take the first block and distribute the cards in this fashion the contents of the block will form a deck of cards we imagine that they form a deck of cards where each column is a cross and then we distribute the columns between the; among the blocks in this fashion.

(Refer Slide Time: 10:53)



Now, let us see how these deck of cards actually look like mind you, these are blocks that are already sorted a sorted block. Let us assume that the elements that we have to sort are all zeros and ones. If we can prove that our algorithm works correctly on all binary sequences then using the 0, 1 principle we can argue that it will work correctly for any sequence drawn from a linearly ordered set.

Therefore, through the proof, let me assume that we are dealing with binary sequences. Now think back to the first step, in the first step what we did was to do a snake like sorting of every single block. So, every block is now in snake like sorted order. And since, we are dealing with binary sequences what we find is that in every block the 0's appear before the 1's.

So, in snake like order, if the 0's appear before the 1's; you will have 1 either this situation or this situation. The 0s will always appear before the 1's depending on whether you have the 0 to 1 transition appears on an odd row or an even row, we have the situation if it happens on an odd row then we have a situation like this. If it happens on an even row, we have a situation like this. In any case, we can say that there is exactly one dirty row in each block.

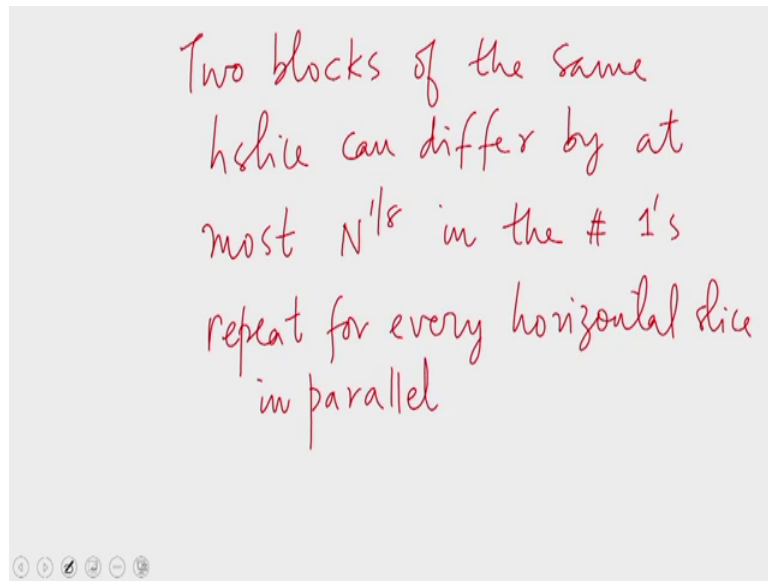
It is one such sorted block that we are treating like a deck of cards and distributing over the blocks of the horizontal slice. So, if you consider the various cards here, imagine the cards that are dealt in the first round. In the first round, we deal N power 1 by 8 cards. So, in this figure, we find that this is entirely divide of the transition, the transition happens here it does not happen within this round. Therefore, every single card dealt in this round is identical.

In other words the number of ones that are given to the different blocks through this dealing will be entirely identical, every block is treated exactly the same way. If you consider round which falls within this range, here again the situation is identical the number of ones in every pair of cards are identical, but if you consider round which spans the 0 to 1 transition. There we find that two rows could two columns could differ, two cards could differ.

So, in short what we can see is that when one deck is redistributed, one block could get one more bit one. Then another block that is redistributing a block a , a , a deck in this fashion can introduce an asymmetry of 1 between 2 recipient blocks.

And then we are doing this for every single block of the horizontal slice, there is in particular we separate the contents from the blocks. We take the contents of the first block redistributed over the blocks then we take the contents of the second block again redistributed over the blocks and so on. So, when we do this we can introduce a total difference of N power 1 by 8 between the recipient blocks.

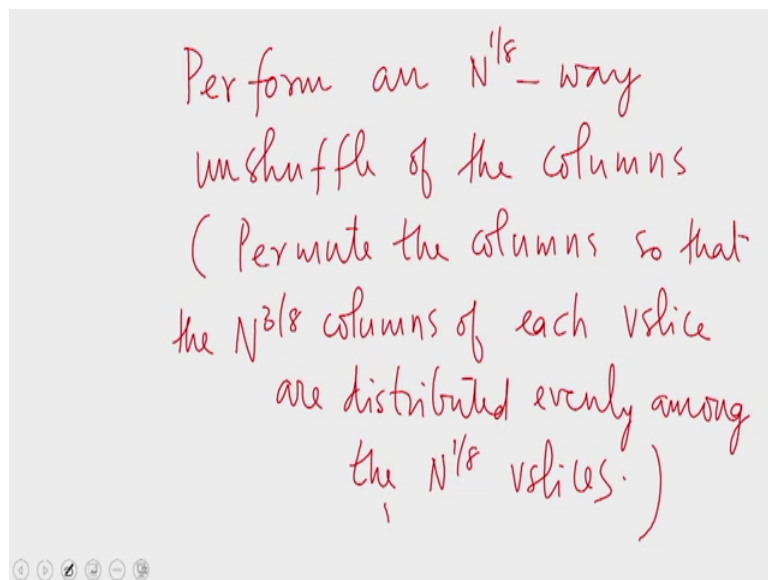
(Refer Slide Time: 15:14)



Two blocks of the same
hslice can differ by at
most $N^{1/8}$ in the # 1's
repeat for every horizontal slice
in parallel

Therefore, what we can conclude is that two blocks of the same horizontal slice can differ by at most N power 1 by 8 in the number of ones. Now this is what we are doing for one horizontal slice, we could repeat this for every single horizontal slice in parallel.

(Refer Slide Time: 16:14)

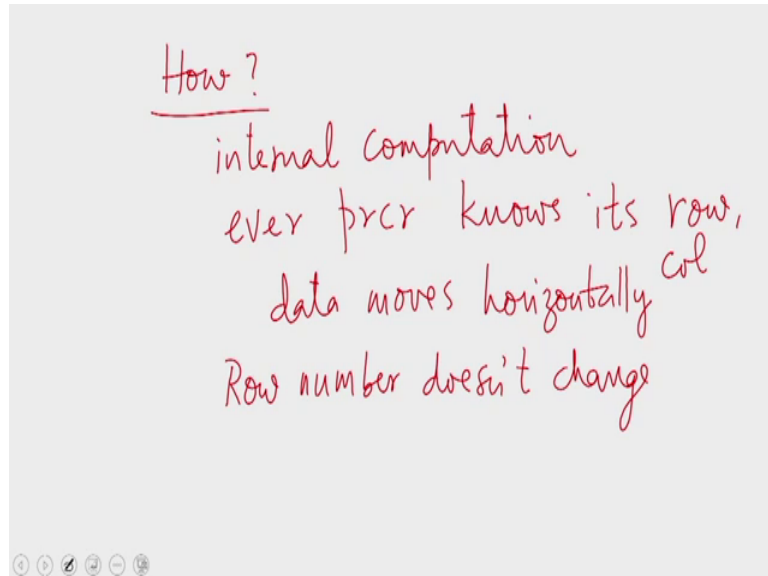


Perform an $N^{1/8}$ -way
unshuffle of the columns
(Permute the columns so that
the $N^{3/8}$ columns of each vslice
are distributed evenly among
the $N^{1/8}$ vslices.)

So, step 2, in fact, could be specified like this. What we do is to perform an N power 1 by 8 way unshuffle of the columns that is the same as saying that we permute the columns. So, that the N power 3 by 8 columns of each vertical slice are distributed evenly among

the N power 1 by 8 for different slices. So, this is what we want to accomplish algorithmically, how exactly will we do this that is a next question.

(Refer Slide Time: 17:36)

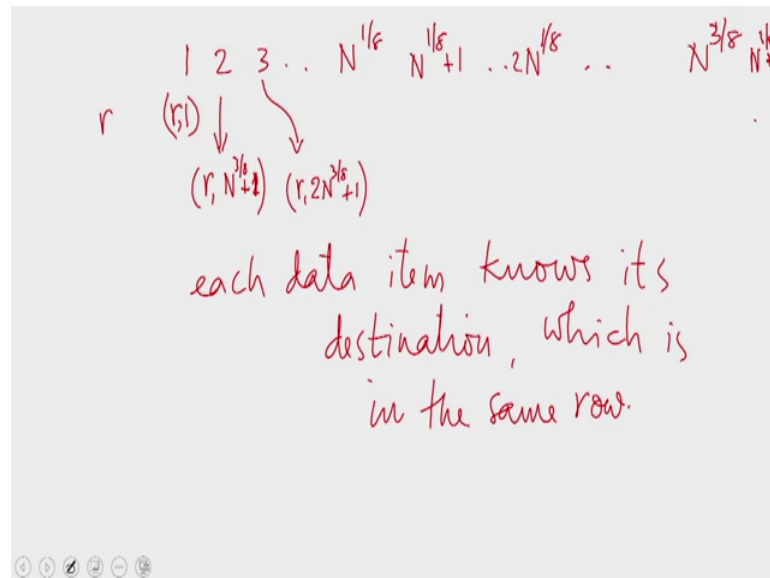


All we require is some internal computation. We assume that every processor in the mesh knows its index. In other words, it knows its row as well as its column in the mesh. We assume that every processor knows its row and its column in the mesh. Now in this step data is moving horizontally that is because we treat each column as a card that is to take part in the unshuffle operation, but the data is not moving within the column the entire column is treated as one single card and it is the cards that we shuffle.

So, there will be no data movement vertically data will move only horizontally. So, if a processor knows its row number and the column number it can easily calculate its destination row number. And the column number the row number will not change, because the data will not be moving vertically only the column number will change. So, that is an invariant for the step, the row number does not change.

Therefore to find the coordinates of the destination point a processor has only to figure out the new column number.

(Refer Slide Time: 19:22)



For example, let us consider the first deck of cards. In the first deck of cards, I have cards numbered from 1 to $N^{3/8}$ by 8 these are the cards of the first vertical slice cards are the same as the columns. So, these are the columns of the first vertical slice.

Now, to which columns will these elements go the first element will go to the same vertical slice, what it means is that the element that the first position. So, the element that row r . Let us consider the elements of row r . So, an element which is at the first position in the r th row will have to go to the same place, what it essentially means is that this element is not going anywhere this element should be exactly where it is.

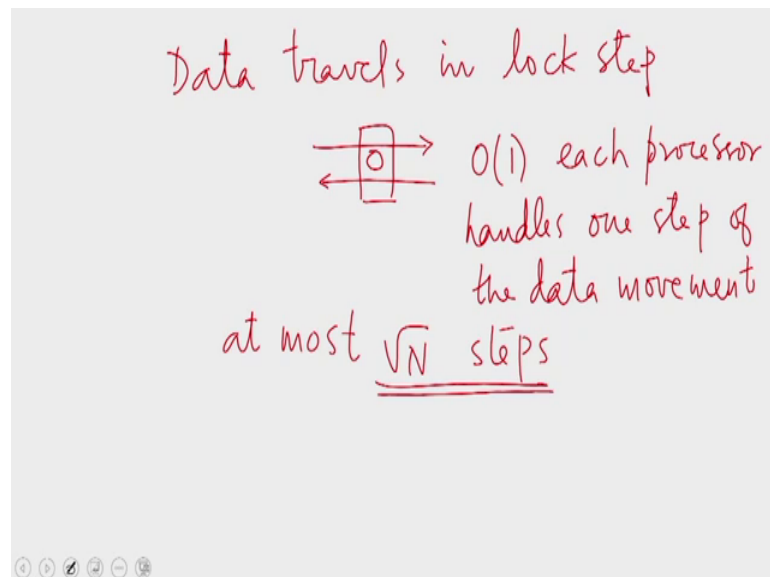
Now, the second element should be going to the second block which means that will be the first element of the second block, the second block, the second vertical slice begins with $N^{3/8} + 1$. So, here the coordinate is going to be $r, N^{3/8} + 1$ and this will be going to the third vertical slice which will be $r, 2N^{3/8} + 1$ and so on. So, likewise every single element in the mesh can find destination coordinate. Now the destination coordinate is going to be within the same row.

So, every element has to move within the same row, some elements will have to move right, some elements will have to move left. In particular, if I consider the element here the element which is at position $N^{3/8} + 1$, this will have to go to the second column, because it is going to this element will have to go to the first vertical slice. So, this will be taken up when the second vertical slices being unshuffled.

So, this element will go to the first vertical slice at an appropriate place. So, every element can statically calculate the destination of the data that it contains. So, that is what I said before you have to separate the contents of the block. From the block itself the block is recipient of data, the present data is being sent around in the mesh and this data will reach the destination and then settle down there.

So, after calculating addresses in this manner each data item knows its destination which is in the same row. Then all the data has to do is to just up and go the data will start walking in lockstep.

(Refer Slide Time: 22:44)



Imagine that the data travels in lockstep some of the data will be traveling from the left to the right. The rest will be traveling from the right to the left of course, some elements are not going anywhere like the element of the first column those elements are not going anywhere, the rest of the data will be traveling either to the left or to the right.

Now, since the data is traveling in lockstep to achieve this all we have to do is to establish certain q protocol set every single processor. So, at every single processor, we have two streams of data, data that is coming from the left and going to the right and data that is coming from the right and going to the left. And then a data item will occupy the contents of the processor.

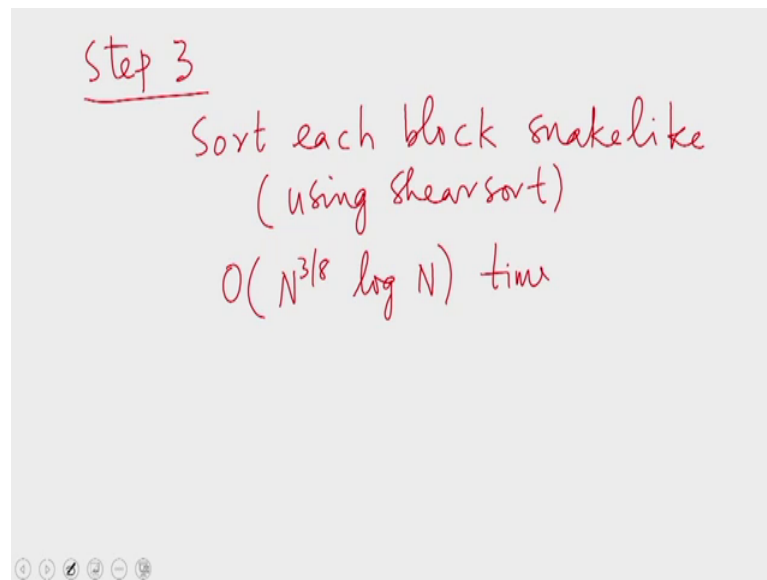
So, there is an empty seat as well. So, what the data what the processor has to do is this in every single step, it will receive one data item from the left which is a data item belonging to the upper q . If this data item is not designed to the same processor, it will be sent out onto the right. Similarly, it will also receive a data item from the right side this belongs to the lower q , if this data item is not designed to this processor it should be sent further on to the left side.

So, all this can be done in out of one time each processor has two messages to handle one from the left side and one from the right side. So, synchronously it will receive these two messages one from the left side and one from the right side. And then it will decide whether any of these messages should be saved within this processor, if it is saved then it is not to be sent further on, but if it is not to be saved then it must be sent along the appropriate direction. So, after checking this; the data will be sent out along the appropriate edge in the second part of the step.

So, in order of one time each processor can handle the data moment handles one step of the data movement. And the data may have to go from one to the first vertical slice to the last vertical slice or vice versa. So, it may have to travel a distance of $\theta \sqrt{N}$ in any case. So, the time taken by this step is at most \sqrt{N} assuming that one step constitutes all this receiving two messages from the neighbors and sending two messages to the neighbors that anyway we assume on all interconnection networks that is we assume that every processor is capable of receiving a message from all its neighbors and sending out a message to all its neighbors in one single step.

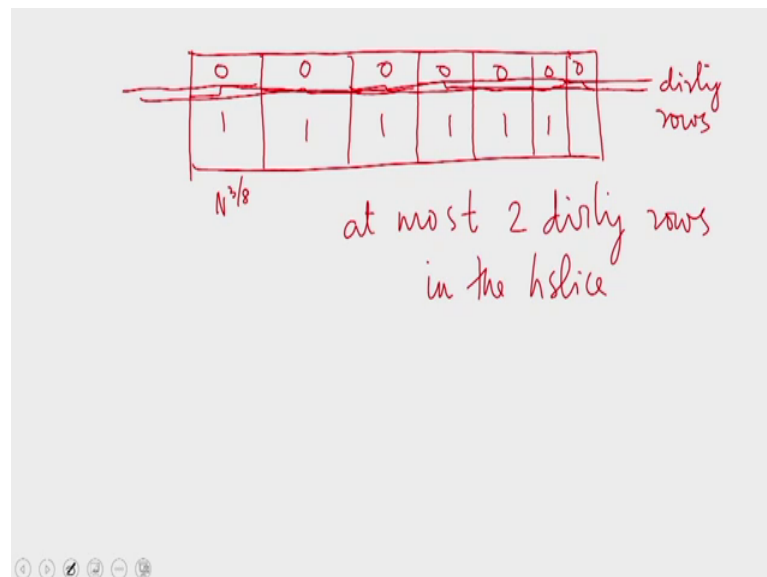
So, with this assumption, we can see that at most \sqrt{N} steps are necessary to ensure that the columns are unshuffled in this fashion. So, the second step here takes \sqrt{N} time the second step of the algorithm will take \sqrt{N} time

(Refer Slide Time: 26:13)



Now we come to the third step, the third step of the algorithm, in the third step of the algorithm we once again sort each block snakelike using shear sort that we saw in the last class. So, this is similar to the first step and therefore, we will take order of N power 3 by 8 times log in time.

(Refer Slide Time: 26:53)



So, now, let us see what the horizontal slices, we will look like take a horizontal slice. What we know is that any two columns of the horizontal slice differ by at most N power

1 by 8 and the number of ones in them that is what we have just estimated, but then the width of a block is N^3 by 8.

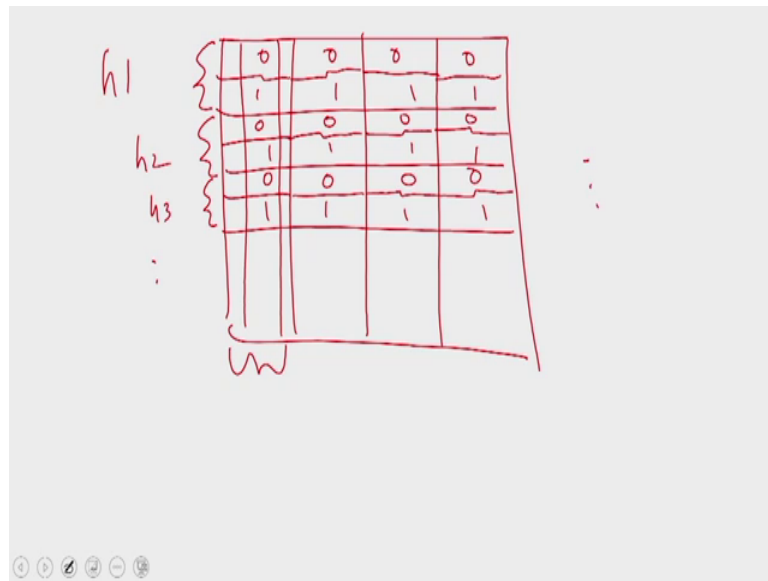
Therefore, when you look at the entire horizontal slice there can be at most two dirty rows that is because let us say, we are considering two blocks one in which the number of 1's is the least and one in which the number of 1's is the most if the number of 1's is the most then the kink in this block appears early the kink is where the 0 1 transition happens. So, in the whole of this horizontal slice, this is the block where the kink happens at the earliest.

Then let me consider the block in which the kink happens, the latest this is the block in which the number of 1's is the least these two have to be approximately at the same place that is because the number of 1's between any two blocks in this horizontal slice can differ by at most N^1 by 8 and N^1 by 8 is far less than N^3 by 8.

Therefore, the two kinks can be separate by at most N^3 by 8. Therefore, in the worst case there should appear on consecutive rows. So, this will be the case for these two blocks which have the largest number of ones and the smallest number of 1's then any other block that we consider will be in the mediate or in other words all of them look almost alike except that there are at most two dirty rows.

So, every horizontal slice now looks about the same the kinks in them differ by at most N^1 by 8 between any two blocks which means there can be at most two dirty rows, there is this window of N^1 by 8 is much smaller than the width of the block N^3 by 8. Therefore, in the most probable cases that this window will be entirely within one row which means the block is going to have only the horizontal slice is likely to have only one dirty row, but it is also possible that it spills over two adjacent rows in which case there could be two dirty rows overall.

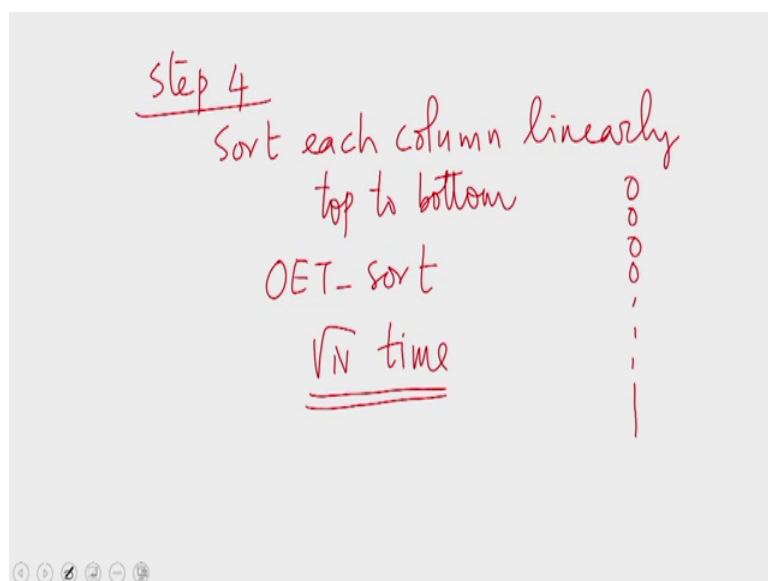
(Refer Slide Time: 30:00)



So, when you consider two horizontal slices they have at most two dirty rows. Therefore, in the entire mesh now, a situation like this prevails each horizontal slice.

So, the entire mesh now looks like this each horizontal slice has some number of clean rows at the top and some number of dirty rows at the clean rows at the bottom the top, clean rows as 0 rows. And the bottom clean row as one rows and then in between there could be at most two dirty rows.

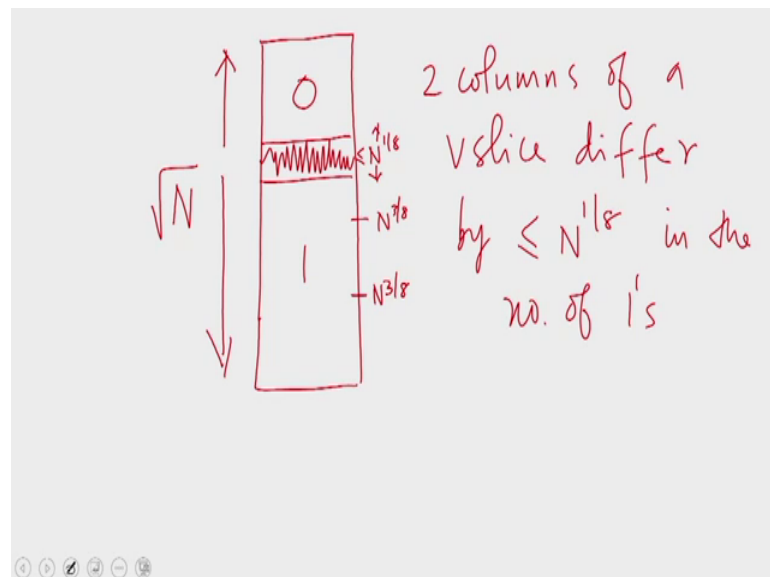
(Refer Slide Time: 31:27)



So, this is how each h slice now looks like. Then in step 4, we sort each column linearly top to bottom. Since, we are dealing with bits this is tantamount to letting all 1's fall down to the bottom. So, now we will have all the ones settling at the bottom of the column, all the 0s will be at the top. How much time will this take, if we use or even transposition sort for this, this will take \sqrt{N} steps.

So, step 4 takes \sqrt{N} steps, if you recall step two also took \sqrt{N} steps the unshuffling operation took \sqrt{N} steps, but steps 1 and 3 did not take \sqrt{N} steps, it they took smaller of \sqrt{N} steps. So, far we have spent $2\sqrt{N}$ plus smaller \sqrt{N} steps.

(Refer Slide Time: 32:40)



Now, at this point when you look at the vertical slice; look at any one vertical slice what we find is that in any column, the 0's appear before all the ones, but then when you consider 2 rows belonging to the same vertical slice, 2 columns belong into the same vertical slice. We find that they have approximately the same number of 1's. Why is that a vertical slices made up of blocks of the sort?

So, this is one vertical slice. So, if you consider 2 columns belonging to this vertical slice. We find that they have approximately the same number of ones that is the contribution towards the number of 1's. In these 2 columns is approximately the same from each block, there could be a difference of at most 1. Therefore, if you consider 2 columns, the difference in the number of 1's cumulatively could be at most N power 1 by

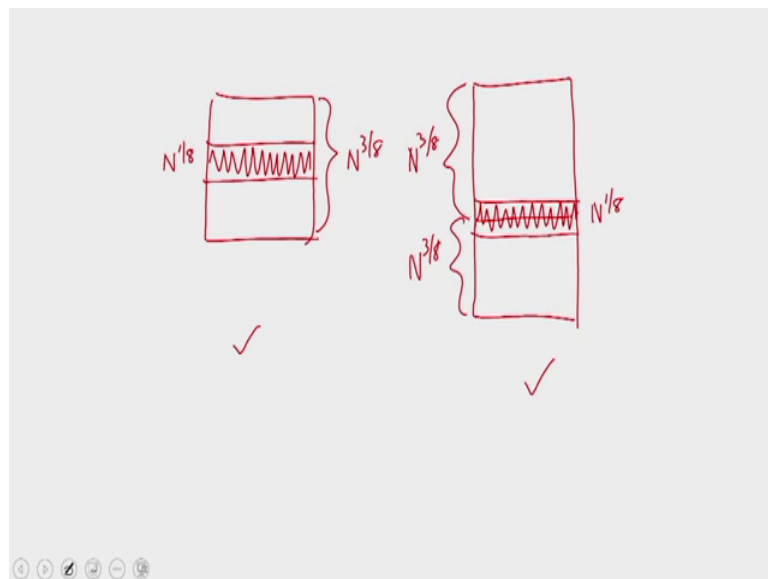
8. Therefore, what we know at this point is that 2 columns of a vertical slice differ by at most $N^{1/8}$ in the number of ones.

Therefore, if you take two consecutive columns from this vertical slice; from the same vertical slice, we find that the number of 1's between them differs by at most $N^{1/8}$ whereas, the height of the vertical slices is \sqrt{N} which is much larger than $N^{1/8}$. Therefore, the number of dirty rows that is the band of dirty rows has a height of at most $N^{1/8}$ in any vertical slice. So, the dirty band is now pretty narrow indeed in comparison to the total height of the vertical slice.

So, above this dirty band, we have a clean band of all 0's and below this we have a clean band of all 1's. So, this is the situation in vertical slice after vertical slice every vertical slice looks almost sorted there is a small dirty band of size at most $N^{1/8}$ in the middle of the vertical slice.

Now, this vertical slice is divided into several blocks each block has a width of $N^{1/8}$ and height of $N^{3/8}$. So, if you measure the vertical slice in units of $N^{3/8}$, what we find is that the dirty band either is entirely within one block or it spans at most two adjacent blocks.

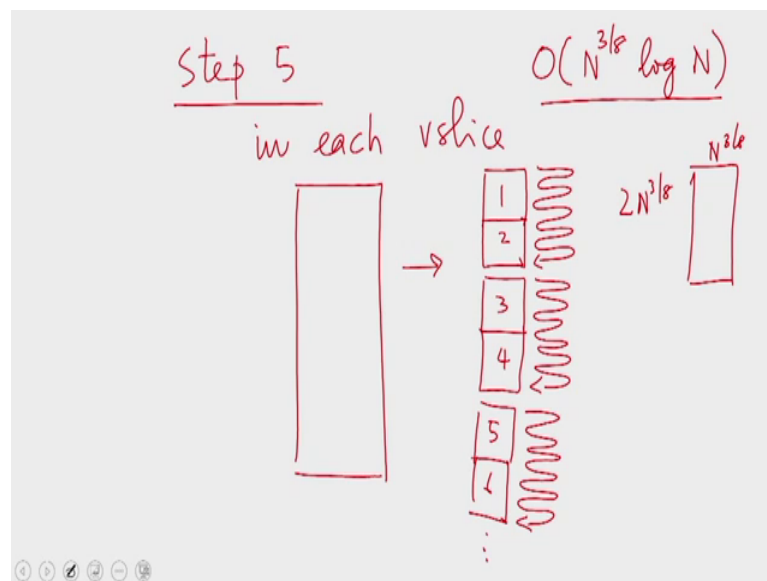
(Refer Slide Time: 35:39)



The dirty band could either be within one single block that is, because the height of the dirty band is at most $N^{1/8}$ whereas, the block height is $N^{3/8}$.

So, if this is the case we have exactly one dirty block in the vertical slice or it could be that this dirty band spans 2 adjacent blocks the height of the dirty band is at most $N^{1/8}$ the height of a block, here to here is $N^{3/8}$. And here to here is also $N^{3/8}$. So, now, the dirty band will span at most two consecutive blocks within every vertical slice, this is the worst that can happen. The second case is the worst that can happen the first case is the best that can happen.

(Refer Slide Time: 36:53)



Now, in the next step which is step 5 of our algorithm, what we do is this in each vertical slice, we pair of the blocks in this fashion, the first block and the second block together will form 1 pair. Then the third block and the fourth block will form another pair then the 5th and the 6th will form another pair and so on.

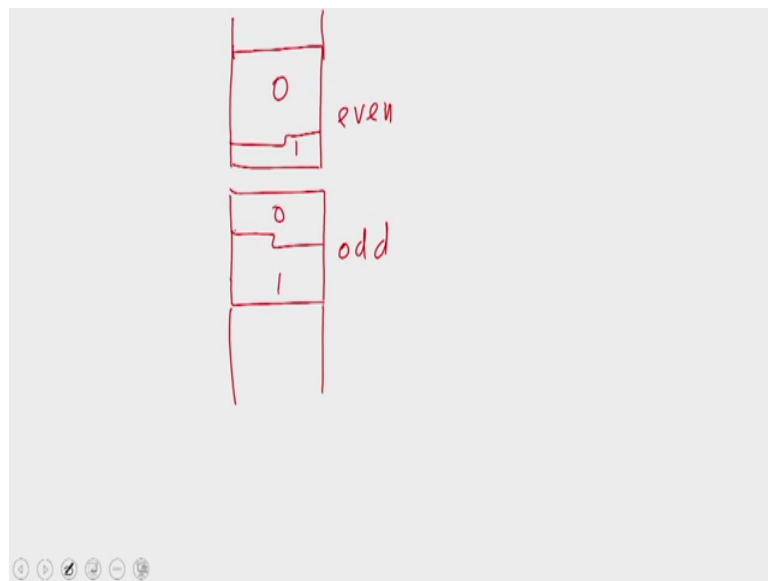
We visualize every vertical slice as consist consisting of pairs of blocks in this fashion and then we will sort each pair in snakelike order. In the shear sort algorithm that we studied, in the last class we saw how to do snakelike sorting within a squarish mesh, but here we are considering a rectangular mesh where the height is twice the width, but then exactly the same algorithm can be used. You can verify that the same algorithm will work here. So, using that algorithm, we will sort each pair in snakelike order each pairs taken together this sorted and snakelike order.

So, this is nothing, but an invocation of the shear sort algorithms. So, we saw in the last class it will take order of $N^{3/8}$ times $\log N$. Now the height is twice $N^{3/8}$

by 8, the width is N^3 by 8. For each instance of shear sort still the algorithm will run in order of N^3 by 8 times $\log N$ times. So, that you can verify, you can verify that shear sort algorithm will work correctly on rectangular basis as well.

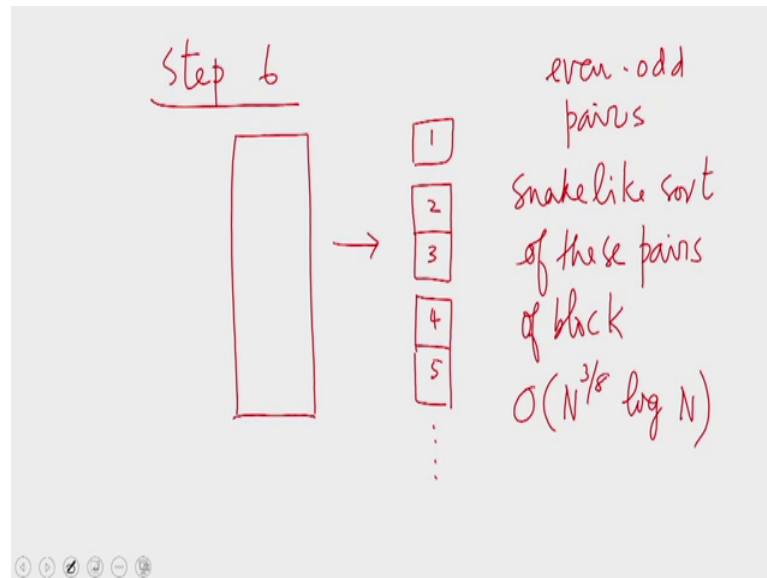
Now, if the situation was the first one that is if the dirty band was entirely within one block. Now that dirty band has been cleaned up, it has been replaced with one single dirty row. And therefore, the vertical slice will have exactly one dirty row, but had the situation been the second one that is if the dirty band spanned two consecutive blocks, we do not know whether they were an odd even pair. If it was an odd even pair the snake like ordering of this odd even pair would have cleaned up this dirty band, but had it not been that is had the dirty band been spanning an even odd pair then this would not have happened.

(Refer Slide Time: 39:36)



Then we would have the upper 1 cleaned up and the lower 1 also cleaned up the upper even block and the lower odd block are both separately cleaned up, but then there would be some number of ones at the bottom here. And there would be some number of 0's at the top here. So, if you look at this we would find an arrangement like this, but then this can be cleaned up in the next step if you do an even odd pairing.

(Refer Slide Time: 40:15)

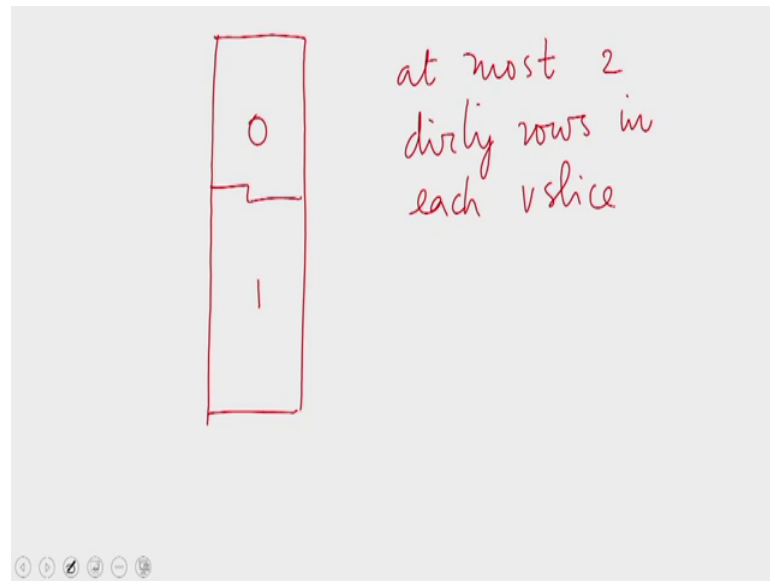


So, step 6 is identical to step 5 except in that within every vertical slice we form pairs in this fashion. The first block does not pair with anybody the second block and the third block will form a pair then the 4th block and the 5th block will form a pair and so on.

So, we form even odd pairs and do a snake like sort of these pairs of blocks using shear sort exactly as we did in step 5. So, this would again take order of N power 3 by 8 times $\log N$ time. So, this is again not one of the dominant steps, but at this point we can be assured that every vertical slices in sorted order. The vertical slice looked like this before there was a small dirty band in the middle this dirty band could have spanned two consecutive blocks, but we do not know whether those consecutive blocks are odd form an odd even pair or an even odd pair.

Therefore, we consider both the cases, first we form odd even pairs and sort the odd even pairs in snake like order then we form even odd pairs and sort the even odd pairs and snake like order. Once, we have finished doing this we could, we would have ensured that the V slices completely sorted. So, at this point the vertical slice is completely sorted.

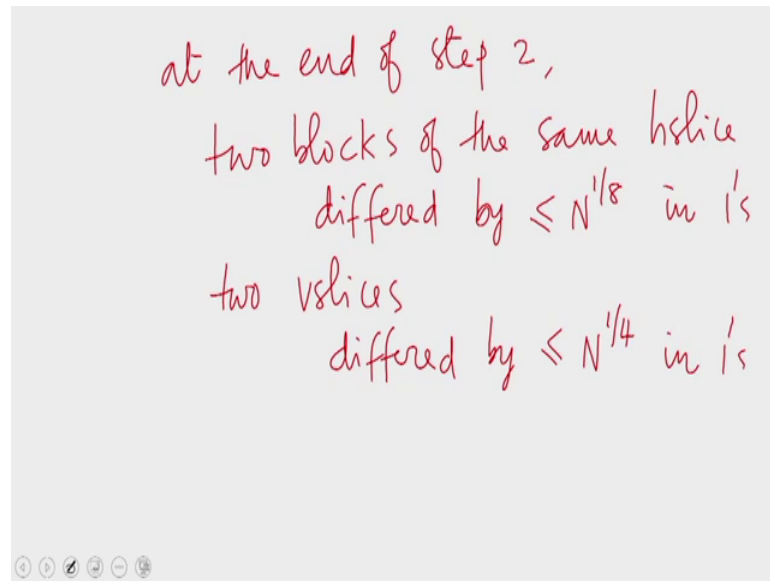
(Refer Slide Time: 42:10)



We have vertical slice that looks like this with 1 kink. So, we know that there are at most 2 dirty rows in each vertical slice. After step 2, we figured that when you look at two blocks of the same horizontal slice the number of 1's between them differed by at most N power 1 by 8 that was the case after step 2.

Now, after step 2, we have not moved data around horizontally all the data moments have been vertically. For example, in step 3 we sorted the columns in step, in step 3 we sorted the blocks. Therefore, the data does not spill out of the blocks. In step 4, we sort the columns the data moves only vertically not horizontally. And in step 5 again data moves only vertically not horizontally. In step 6 again data moves only vertically not horizontally.

(Refer Slide Time: 43:30)

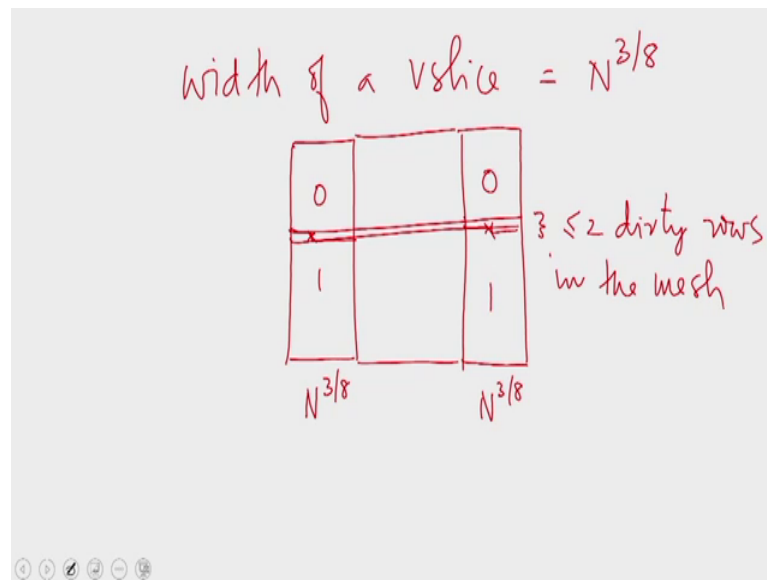


at the end of step 2,
two blocks of the same hslice
differed by $\leq N^{1/8}$ in 1's
two vslices
differed by $\leq N^{1/4}$ in 1's

Therefore, what our argument now is that at the end of step 2, two blocks of the same horizontal slice differed by at most $N^{1/8}$ in the number of 1's.

Therefore, if you aggregate over all horizontal slices, you can say that two vertical slices differed by at most $N^{1/4}$, there are $N^{1/8}$ vertical slices. In the number of ones from step 2 to step 6, we have moved data only vertically as I said just now. Therefore, the same invariant holds, even now that is if you consider two vertical slices they differ by at most $N^{1/4}$ in their number of 1's. But then the width of a block which is the same as the width of a V slice is $N^{3/8}$ that is a 0 to 1 transition within a vertical slice.

(Refer Slide Time: 44:42)

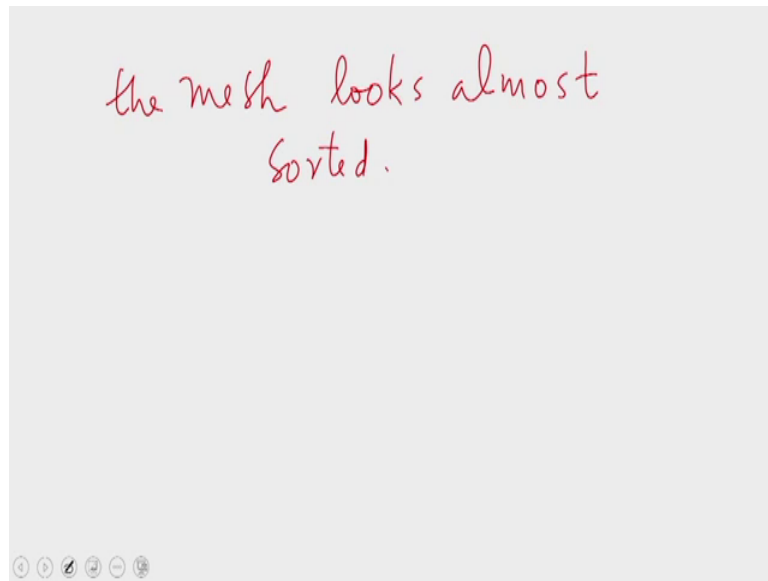


In particular, if I consider a vertical slice with the least number of 1's and the most number of 1's. The kinks in them, the kinks in them differ by at most $N^{1/4}$ but the width of the vertical slice is $N^{3/8}$ much larger than $N^{1/4}$.

Therefore, we can safely say that the kink in one vertical slice is either in the same row as the kink, in another vertical slice or it is in an adjacent row. If you take the vertical slice with the least number of the ones and most number of 1's, we find that their kinks differ by at most $N^{1/4}$.

Therefore, in the entire mesh now we find that we have at most two dirty rows. Now the graph looks almost sorted the mesh looks almost sorted.

(Refer Slide Time: 46:20)



How do we ensure that the remaining two rows are also cleaned up in the desired bounds will decide our result that we will be able to sort the elements in $3\sqrt{N}$ plus smaller \sqrt{N} steps, but then the details of the last 2 steps. We shall see in the next lecture. So, that is it from this lecture hope to see you in the next.

Thank you.