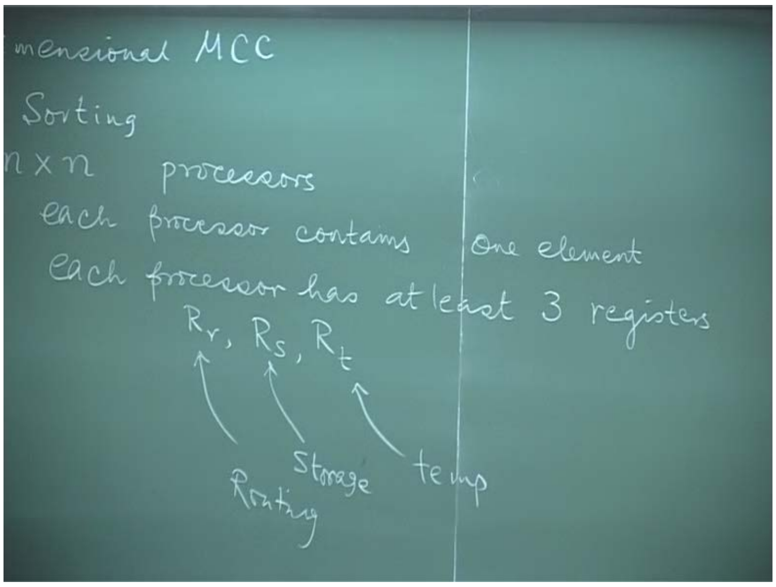
Parallel Algorithms

Prof. Phalguni Gupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 9

(Refer Slide Time: 00:16)


imensional MCC

Sorting

$n \times n$ processors

each processor contains one element

each processor has at least 3 registers

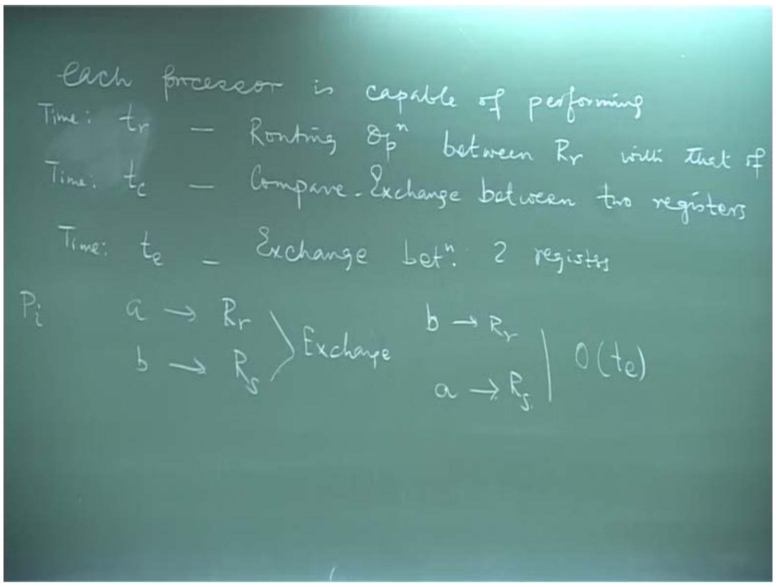
R_r, R_s, R_t

Routing
Storage
temp

Two-dimensional mesh connected computer, right and this algorithm is based on Bitonic sorting technique. I hope you have not yet forgotten what is Bitonic sorting. So, the algorithm is a Bitonic sorting technique and that has been implemented on two dimensional MCC. Just like this. Like the Bitonic sort, what it does? You remember in the Bitonic merge, what will you do? That i is compared with a n plus i.

The smaller one you keep it one side and larger one goes to the another side and then, go both the sequence is Bitonic sequence and they can recursively solve them. That is the idea. Now here, we show that a mesh is having n cross n, crosses, that is n processors and each process contains one element. Each process has at least three registers, R r, R s and R t. So, these are the registers. I mean, memory locations (( )) it needs. This is storage register; this is a say temp and this is called routing. So, in order to transfer one data from one processor to another processor, the data must be in this register, right.

(Refer Slide Time: 03:14)


Each processor is capable of performing

Time: t_r — Routing Op^n between R_r with that of

Time: t_c — Compare-Exchange between two registers

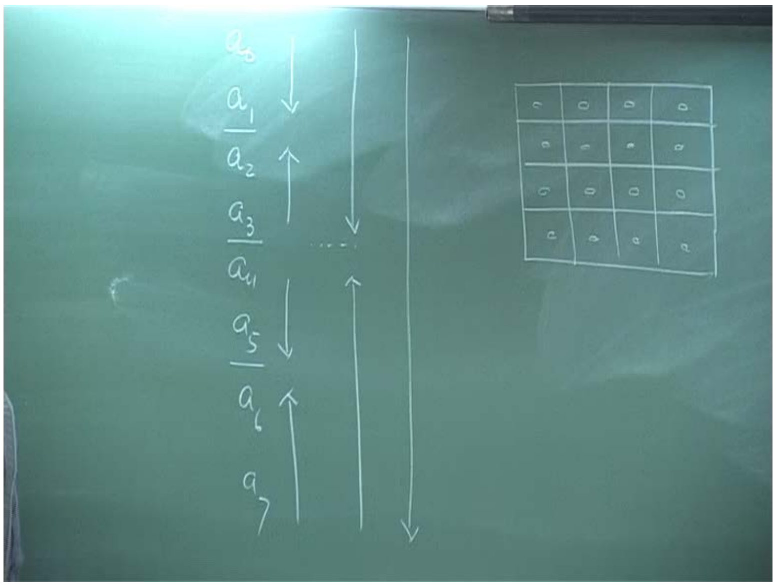
Time: t_e — Exchange bet^n 2 registers

$$P_i \quad \left.\begin{array}{l} a \rightarrow R_r \\ b \rightarrow R_s \end{array}\right\rangle \text{Exchange} \quad \begin{array}{l} b \rightarrow R_r \\ a \rightarrow R_s \end{array} \Bigg| \; O(t_e)$$

Now, each processor is capable of performing three types of operation. One is routing operations between R r with that of its neighbour. Compare exchange between two registers on same processor. Compare exchange means that content of R r and content of R s will be compared and if it is, the winner will be kept in R s and loser will be kept in R r. Who is the winner? Suppose, in the case of increasing order, minimum is the winner and in the case of decreasing order, maximum will be the winner.

Then, another operation is only exchange operations between two registers. So, these are the three operations the processors can perform. The time needs, I have already told about R r. So, let us put some other symbol. Let us put t r, t c and t e. So, the time need for one routing takes t r time and t c is the time need for compare exchange and t e is the only exchange. This is the time. So, up to this the definition is clear? So, in case you need, it is clear?

(Refer Slide Time: 07:38)

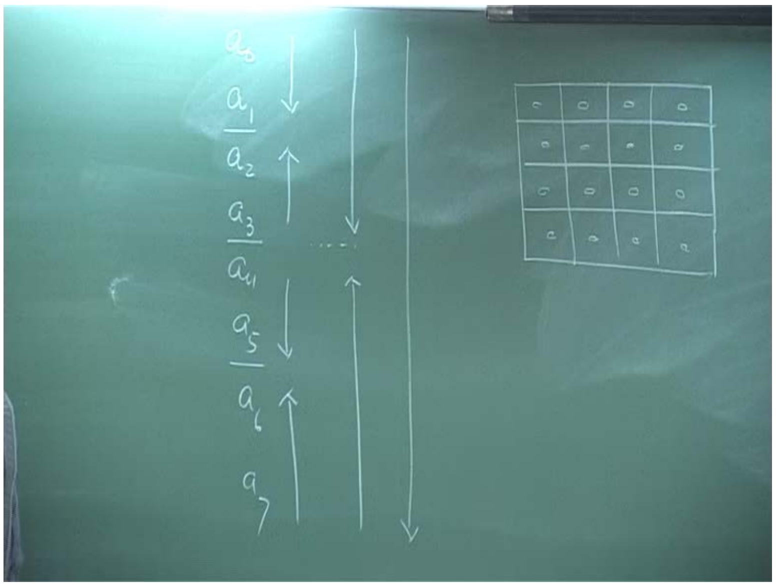


Time means, say p i has data a in R r register and b is in R s register. So, if you perform exchange operation, what will happen? That b will be here, a will be here, b is in R r register and a is in R s register and to do this, you need t e. Is it ok? Now, before we start about this algorithm, how will we do it, let us first recollect the Bitonic sorting technique. What you did a 1, a 2, a 3, a 4, a 5, a 6, a 7, let us take a 8 here. These are the 8 elements you have and every time you have to arrange in such a way that it should becomes a Bitonic sequence for the next step and also it is known to you that 2 sequence with 2 elements is always a Bitonic sequence.

So, this is one Bitonic sequence and this is another Bitonic sequence. There are 4 Bitonic sequences. So, you have to arrange in such a way that next time you get a Bitonic sequence. So, you have to have some idea that how to create it. So, if I do this is increasing order; this is decreasing order, then this whole sequence becomes Bitonic sequence. Similarly, is the case. Now, you get 2 Bitonic sequences. This is one Bitonic sequence and this is the another one. So, you arrange it in the second phase and then, finally you get this one, right. This is the strategy. Now, in case of two dimensional mesh, suppose we have 16 elements. These elements are here and this processor is connected with these 4 neighbours, right, at least, at most. So, it consist of, you remember that it consists of 6 steps and there it consists of 4 parallel phase. First phase, you will be making a Bitonic sequence of size, you have Bitonic sequence of size 2. If you consider, this is one size 2, 1, 1, 1, 1, 1, 1, like that 8 such Bitonic sizes you have.

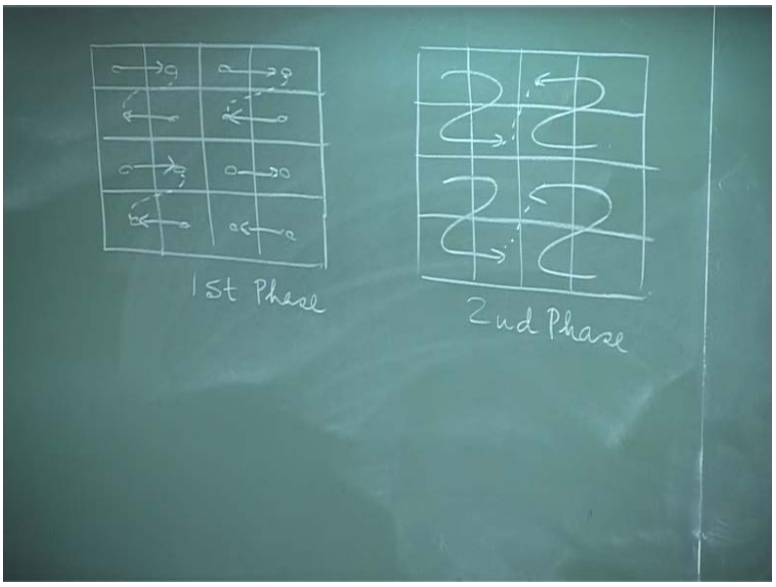
How do you know that there is a class going on? There was a telephone or message? Guessing? After 40 minutes, you cannot guess. Do not worry. Just random? It is difficult. For those of you who joined late, let me tell you, today we are discussing on sorting and the model is two dimensional mesh. Here, we have assumed that there are n cross n processors and each processor contains 1 element and each processor has at least 3 registers or 3 memory locations. One is R r and another is R s and R t. R r is the routing register, R s is the storage register and R t is a temporary storage register. The result, the final result would be is R s register. Now, each processor is capable of doing 3 types of operations, routing operations between two orders; order of processor i and order of processor j. But processor j must be a neighbour and time needs for that is t r. Compare exchange operation between two registers of a processor and it checks order, time order t c and similarly, exchange operation takes order t e time. This is the example what I gave by exchange operations.

(Refer Slide Time: 07:38)

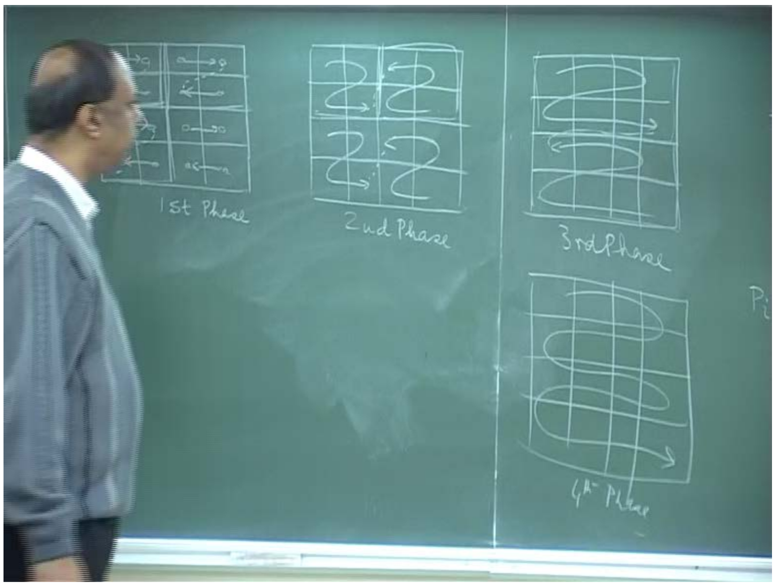


Then, we discussed about the strategy followed in the Bitonic sorting technique for 8 elements. Finally, we are discussing about that how we are going to use it on two dimensional mesh, where you have 4 cross 4 mesh and 16 elements. Now, if you observe this 16 elements, you can think that it can form 8 Bitonic sequences and each of size 2. Now, you have to go to the next level, which will give you 4 Bitonic sequences of size 4, right. So, you have to arrange in such a way that one is in increasing order and other is in decreasing order, right. One possible way could be that, you put it in increasing order and you put in decreasing order, so you get a Bitonic sequence of size 4, this side, this side, and this side and so on.

(Refer Slide Time: 13:09)



(Refer Slide Time: 14:41)

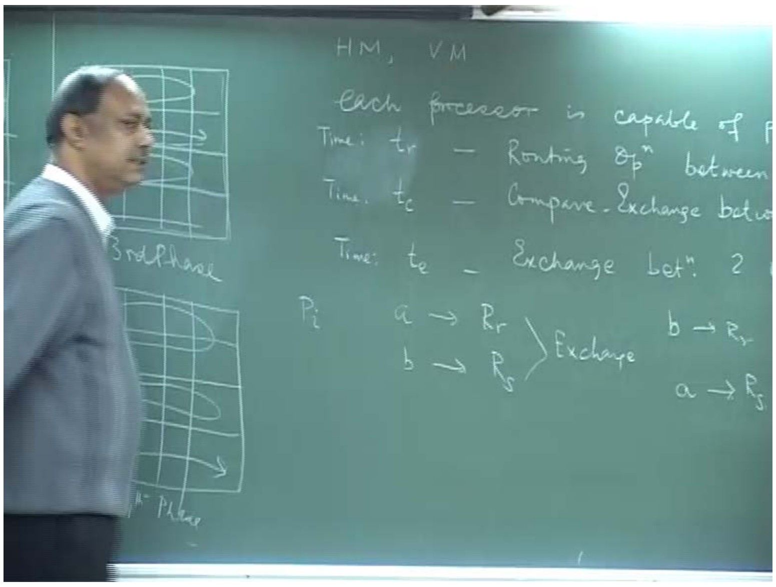


In that process, you find little bit of difficulty. What we do is, we arrange this in increasing order and this in decreasing order and this forms a Bitonic sequence. So, in the first phase, you arrange the data in such a way that you get 4 Bitonic sequences each of size 4. In the second phase, now you have to arrange the data in such a way that you get 2 Bitonic sequences of size 4, each of size 8. So, you arrange the data in increasing order and this becomes the decreasing order. So, you observe that this gives a Bitonic sequence of size 8. How you will be arranging? That part we will discuss later. Second phase.

Now, third phase is, you have arranged the data. You have Bitonic sequence here and another Bitonic sequence here, right. So, you arrange the data in such a way that you get 2 Bitonic sequences, each of size 8. Basically, you will be getting this one and then, this one. So, you get Bitonic sequence of size 61, right. This is increasing order and this would be in decreasing order. So, if I see from this point of view, then it will become a Bitonic sequence in the third phase. So, in the fourth phase, you have a Bitonic sequence. So, you can easily get a sorted sequence. So, basically at the end of the fourth phase, you get the sorted sequence.

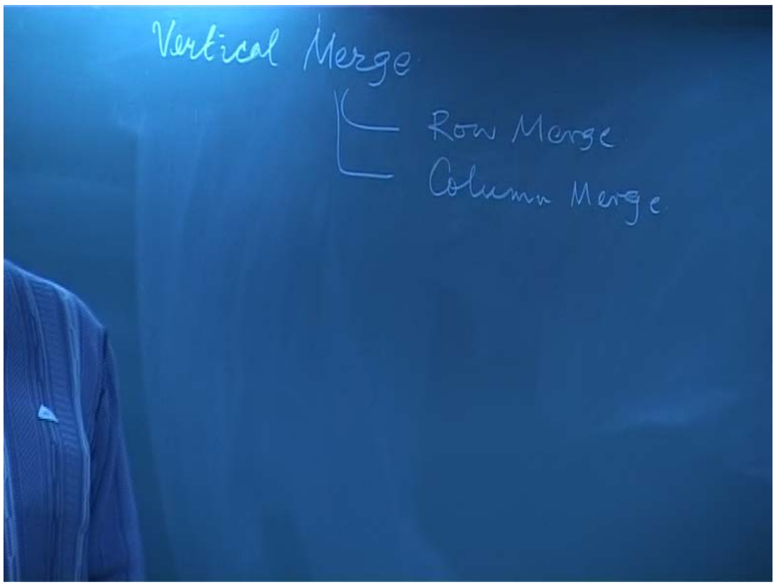
At the end of the third phase, you get sorted sequence. So, the strategy is such that, it should be such that, at the end of every phase, you get some number of Bitonic sequences. This is first one and the second one, it should collapse in such a way that, after logging time, you will be able to get the sorted sequence, right. So, what you observe? First thing is that, you do the merging with respect to the row, right. That is horizontal you merge it. Then, you get Bitonic sequence consisting of two halves, which you need to merge vertically. Here, you get this one and this sub block and you need to merge them horizontally, right. Now here, again you have one sorted sequence and another sorted sequence, this forms a Bitonic sequence and you have to merge it vertically and so on.

(Refer Slide Time: 17:39)


HM, VM
Each processor is capable of
Time: t_r — Routing op^n between
Time: t_c — Compare-Exchange betw
Time: t_e — Exchange bet^n 2
P_i $a \to R_r$, $b \to R_s$ } Exchange
$b \to R_r$
$a \to R_s$
3rd Phase
4th Phase

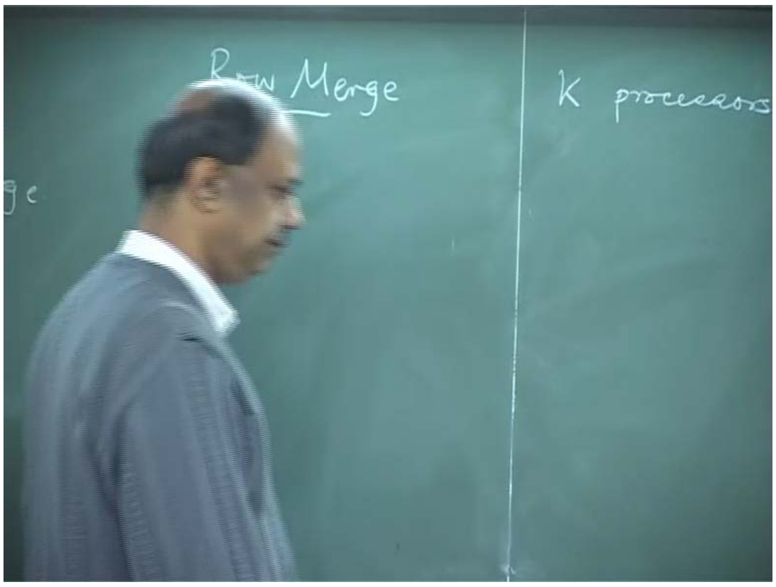
So, once you do the horizontal one and next you should do the vertical one. Horizontal, vertical like that to get the, ultimately find out after the logging session, you get the sorted sequence. Is it ok? The strategy is clear? So, first let us discuss about, so basically we need to know one is horizontal merge and another one is vertical merge, right. If I know this one, then we do iteratively. One after the other one to get this merge.

(Refer Slide Time: 17:50)



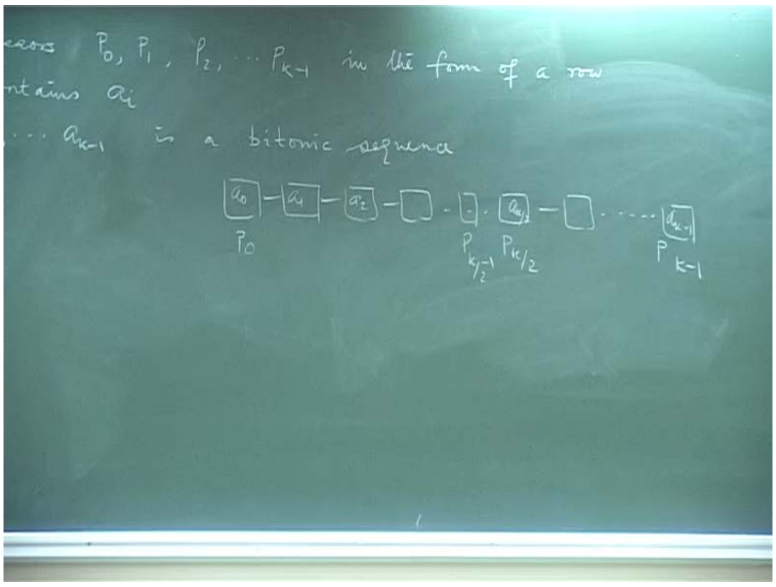
Now, vertical merge of Bitonic sequence consists of two algorithms. One is known as row merge and another one is column merge.

(Refer Slide Time: 18:30)



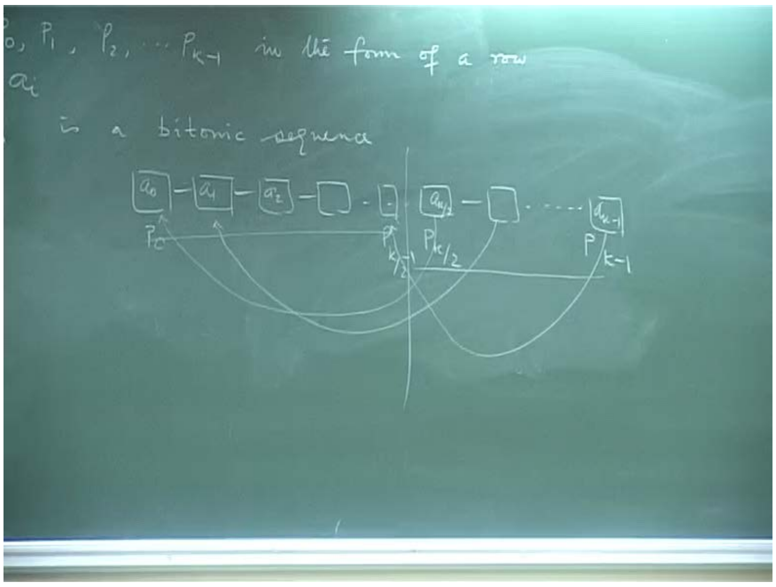
So, first let us discuss about the row merge.

(Refer Slide Time: 18:52)


P0, P1, P2, ... Pk-1 in the form of a row
contains ai
... ak-1 is a bitonic sequence
a0 a1 a2 ak/2 ak-1
P0
Pk/2-1 Pk/2
Pk-1

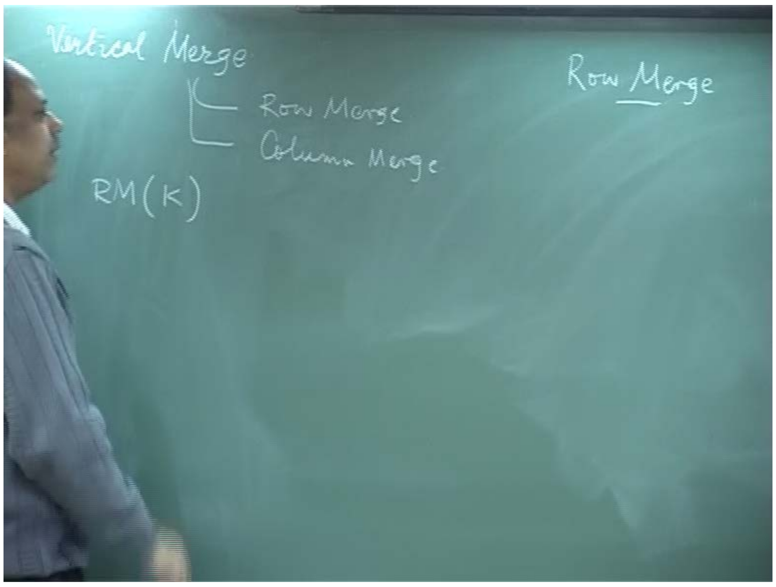
Let us assume that there are k processors, P 0, P 1, P 2, P k minus 1 in a form of row. P i contain a i, a 0, a 1, a k minus 1is a Bitonic sequence. So, what do you have basically is, you have the processors like that and the data is like this. So, what I am going to do in the Bitonic sequence case is, p i should be computed with p i plus k by 2. Agreed?

(Refer Slide Time: 21:10)



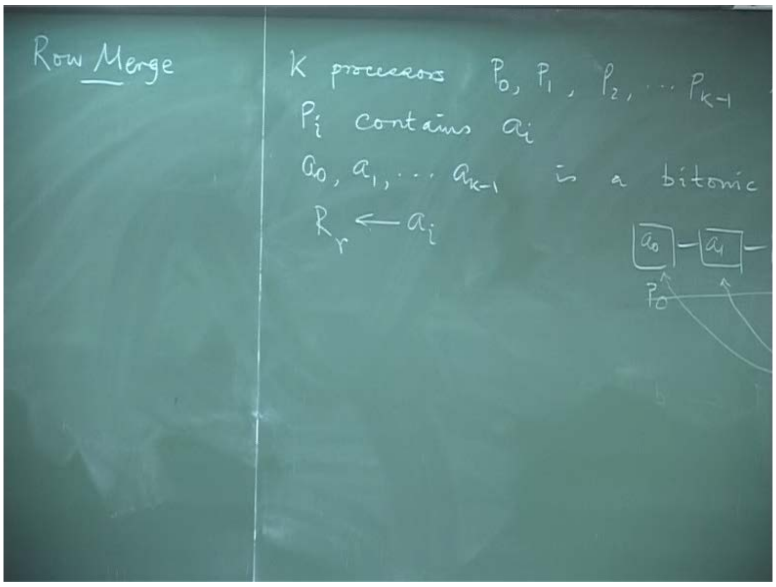
So, basically these data has to be moved here. These data has to be moved here. These data has to be moved here and then compare, right. There will be one accepted element and another is rejected element. The rejected element you send back this side again, right. So, what you will get? This is the Bitonic sequence and this becomes another Bitonic sequence. Agreed? Iteratively, again you call row merge to this one and row merge to this one and so on. You will be getting ultimately the sorted sequence.

(Refer Slide Time: 22:09)



So, if it is the case, so I can write row merge k and we assume data is initially stored in R r register.

(Refer Slide Time: 22:25)


Row Merge

K processors $P_0, P_1, P_2, \dots P_{K-1}$

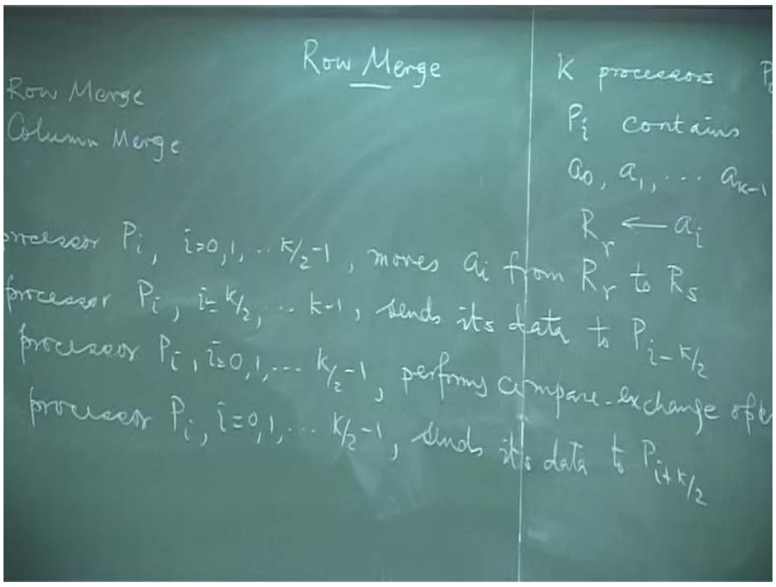
P_i contains a_i

$a_0, a_1, \dots a_{K-1}$ is a bitonic

$R_r \leftarrow a_i$

Assume data is stored initially in R r register. R r contains a i. Assume data contains with R r register. So, once you want to bring this data from here to here, this data has to be saved first. Agreed or not? Because, this data is also in R r register and routing has to be done through R r only. So, you move this data first to R s area.

(Refer Slide Time: 23:03)


Row Merge

Column Merge

Row Merge

K processors

P_i contains

$a_0, a_1, \dots a_{K-1}$

$R_r \leftarrow a_i$

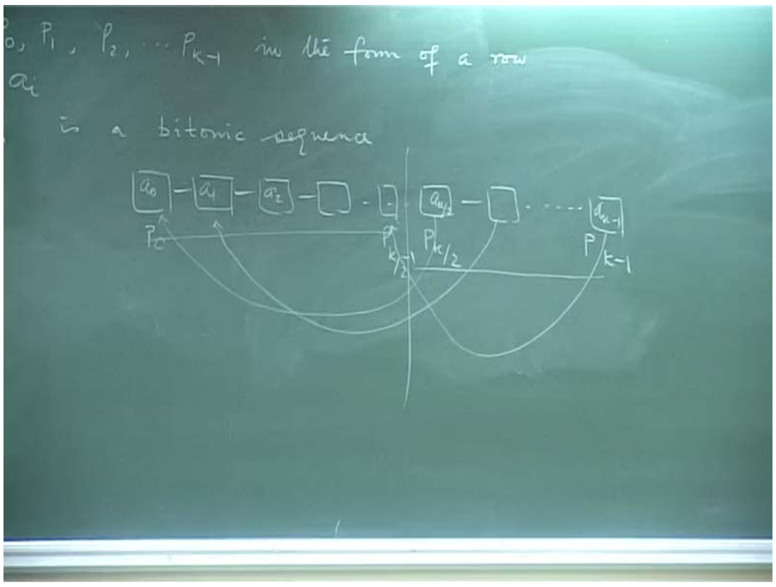
processor P_i, $i=0,1,\dots K/2-1$, moves a_i from R_r to R_s

processor P_i, $i=K/2, \dots K-1$, sends its data to $P_{i-K/2}$

processor P_i, $i=0,1,\dots K/2-1$, performs compare-exchange of

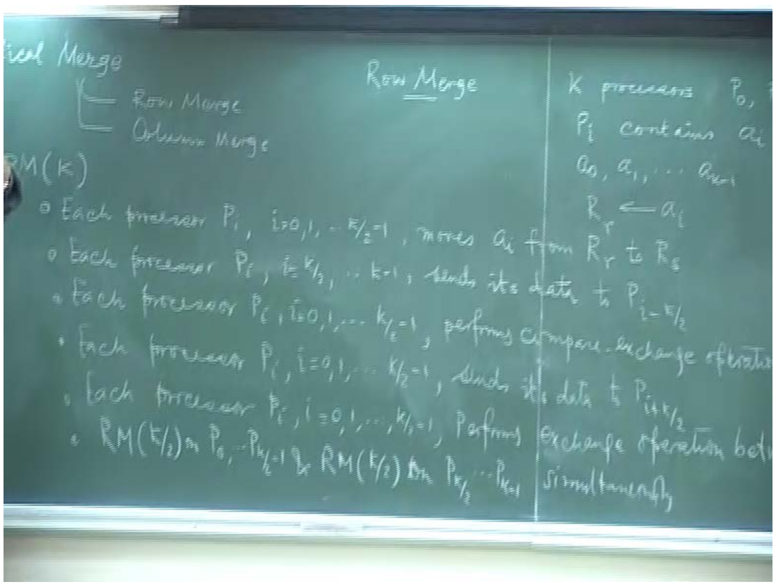
processor P_i, $i=0,1,\dots K/2-1$, sends its data to $P_{i+K/2}$

(Refer Slide Time: 21:10)



So, each processor p i, i is 0 1 to k by 2 minus 1 moves a i from R r to R s and then, next one is, each processor p i, i is equal k by 2 k minus 1 sends its data, once I write that it sends its data, it must be in R r register to p i minus k by 2. Next is, each processors p i i is 0 1 k by 2 minus 1. What it does? It compares, right and performs compare exchange operations between R r and R s. Now, once I perform the R r compare exchange operation between R r and R s, the accepted element is in R s and rejected element is in R r. That is true always. Accepted element will be in R s area and rejected element will be in R r area. Now, if it is the case, now this rejected element has to be sent back to the other part of the area. So, what I do is, this process p i i is 0 1 k by 2 minus 1 sends this beta to p i plus 1 k by 2, right. So, the rejected element has come here and that is in R r area.

(Refer Slide Time: 26:26)



ical Merge

- Row Merge
- Column Merge

Row Merge

RM(k)

- Each processor P_i, $i = 0, 1, \ldots k/2 - 1$, moves a_i from R_r to R_s
- Each processor P_i, $i = k/2, \ldots k-1$, sends its data to $P_{i-k/2}$
- Each processor P_i, $i = 0, 1, \ldots k/2 - 1$, performs compare-exchange operatio
- Each processor P_i, $i = 0, 1, \ldots k/2 - 1$, sends its data to $P_{i+k/2}$
- Each processor P_i, $i = 0, 1, \ldots, k/2 - 1$, performs exchange operation betw
- $RM(k/2)$ on $P_0, \ldots P_{k/2-1}$ & $RM(k/2)$ on $P_{k/2} \ldots P_{k-1}$ simultaneously

k processors P_0,

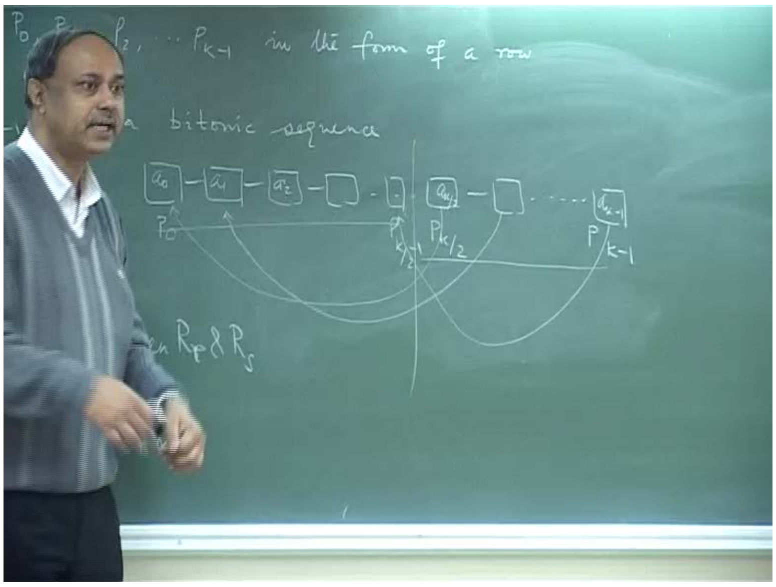
P_i contains a_i

$a_0, a_1, \ldots a_{k-1}$

$R_r \leftarrow a_i$

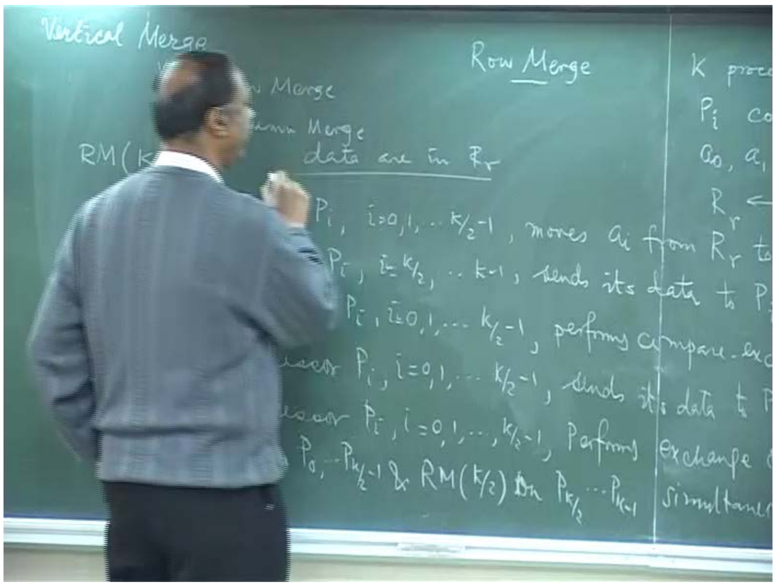
Now, R s contains first k by 2 processors, means it contains all the accepted elements. So, each process, so that you can start again, p i i is 0 1 k by 2 minus 1 performs exchange operation between R r and R s. Now, all the details are in R r registers. So, we can call row merge k by 2 on p 0 p k by 2 minus 1 and R M k by 2 on p k by 2 to p k minus 1 simultaneously. Any (( )) please tell me.

(Refer Slide Time: 21:10)



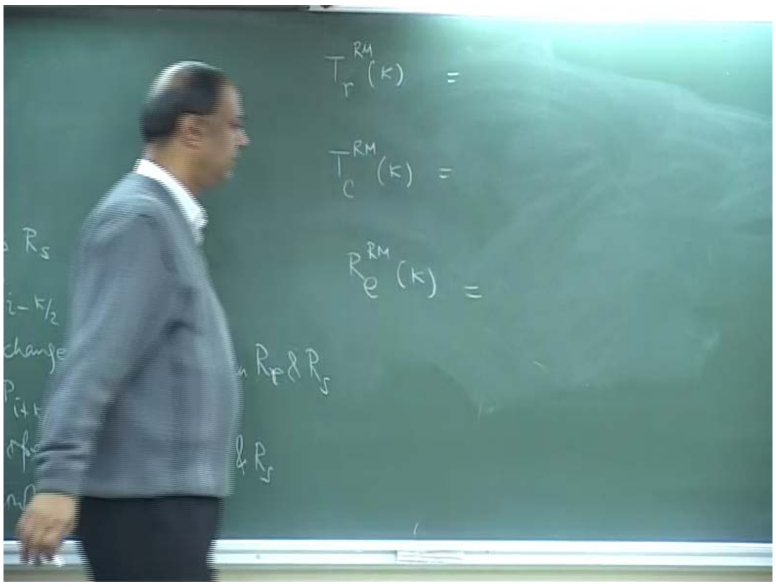
This one. Otherwise, see here, compare and exchange is doing, now R s contains all accepted elements and R r is all the rejected elements. Now, this rejected element has to come here (( )).

(Refer Slide Time: 28:35)


Vertical Merge
Row Merge
Merge
Merge
RM(K
data are in R_r
P_i, $i=0,1,\ldots K/2-1$, moves a_i from R_r
P_i, $i=K/2,\ldots K-1$, sends its data to
P_i, $i=0,1,\ldots K/2-1$, performs compare-ex
P_i, $i=0,1,\ldots K/2-1$, sends its data to
P_i, $i=0,1,\ldots,K/2-1$, Performs
$P_0,\ldots P_{K/2-1}$ & $RM(K/2)$ on $P_{K/2}\ldots P_{K-1}$
K proc
P_i
a_0, a_1
R_r
exchange
Simultane

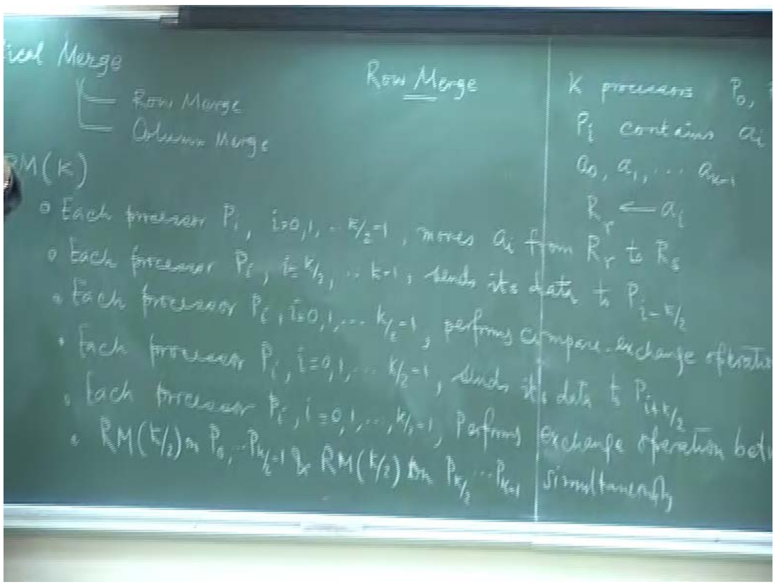
Now, once you have sent the rejected elements here, now this R r contains all junk things and I have to get back this R s thing here, so that, recursive (( )) I can do. Because, in the initial thing, where I decided that, all the data is are in R r register, right. Initially, think that data are in R r. So, if data is in R r, so to call this one, I must have the data in R r registers. So, to do that, just I am performing this operation, right.

(Refer Slide Time: 28:58)



Now, in order to compute the time complexity for this t r row merge k, compare row merge k exchange row merge k.

(Refer Slide Time: 26:26).


Merge
Row Merge
Column Merge

Row Merge

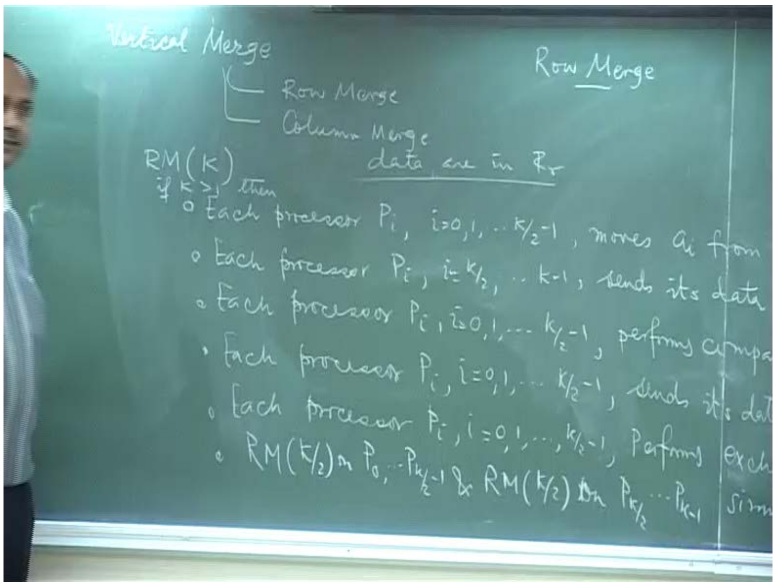
RM(k)

- Each processor P_i, $i=0,1,\dots k/2-1$, moves a_i from R_r to R_s
- Each processor P_i, $i=k/2,\dots k-1$, sends its data to $P_{i-k/2}$
- Each processor P_i, $i=0,1,\dots k/2-1$, performs compare-exchange operation
- Each processor P_i, $i=0,1,\dots k/2-1$, sends its data to $P_{i+k/2}$
- Each processor P_i, $i=0,1,\dots,k/2-1$, Performs exchange operation bet
- $RM(k/2)$ on $P_0,\dots P_{k/2-1}$ & $RM(k/2)$ on $P_{k/2}\dots P_{k-1}$ simultaneously

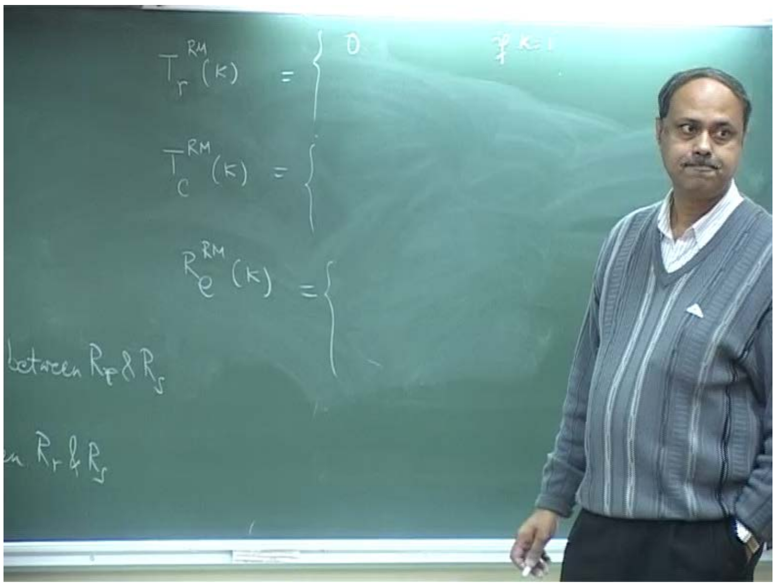
K processors P_0,
P_i contains a_i
$a_0, a_1, \dots a_{k-1}$
$R_r \leftarrow a_i$

Now, the thing is that, we have to go there. I have mentioned about boundary the conditions. Because, once it is recursive, I could tell about the boundary solutions, right. So, what should be the boundary condition?

(Refer Slide Time: 30:04)


Vertical Merge
Row Merge
Column Merge
Row Merge
data are in R_r
RM(K)
if $K > 1$ then
∘ Each processor P_i, $i = 0, 1, \ldots K/2 - 1$, moves a_i from
∘ Each processor P_i, $i = K/2, \ldots K-1$, sends its data
∘ Each processor P_i, $i = 0, 1, \ldots K/2 - 1$, performs compa
• Each processor P_i, $i = 0, 1, \ldots K/2 - 1$, sends its dat
∘ Each processor P_i, $i = 0, 1, \ldots, K/2 - 1$, Performs exch
∘ RM(K/2) on $P_0, \ldots P_{K/2-1}$ & RM(K/2) on $P_{K/2} \ldots P_{K-1}$ sim

(Refer Slide Time: 30:22)

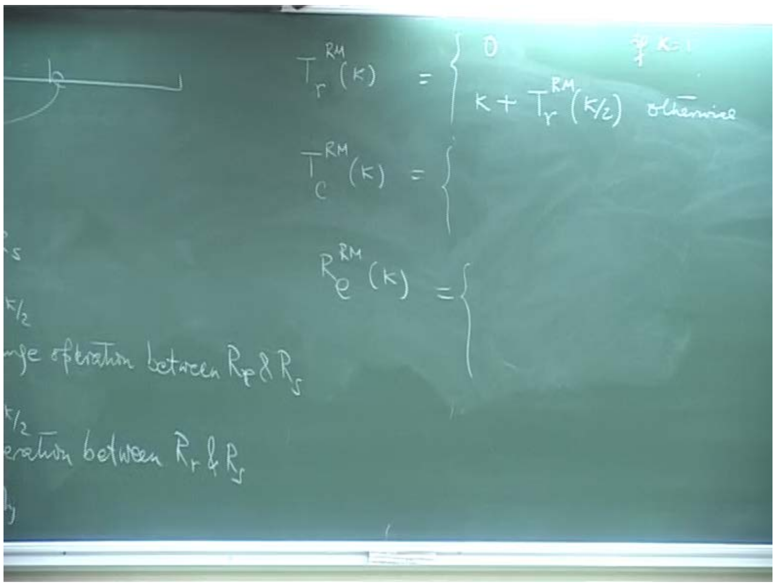

$T_r^{RM}(K) = \{ 0$ if $K = 1$
$T_c^{RM}(K) = \{$
$R_e^{RM}(K) = \{$
between R_p & R_s
n R_r & R_s

Whenever k is more than 1, you do it. So, if k is greater than 1, agreed, you do this. If it is the same, So, if k is 1, How many routings you need? 0. Otherwise? How did you do get that? You are also supporting that.

(Refer Slide Time: 31:23)

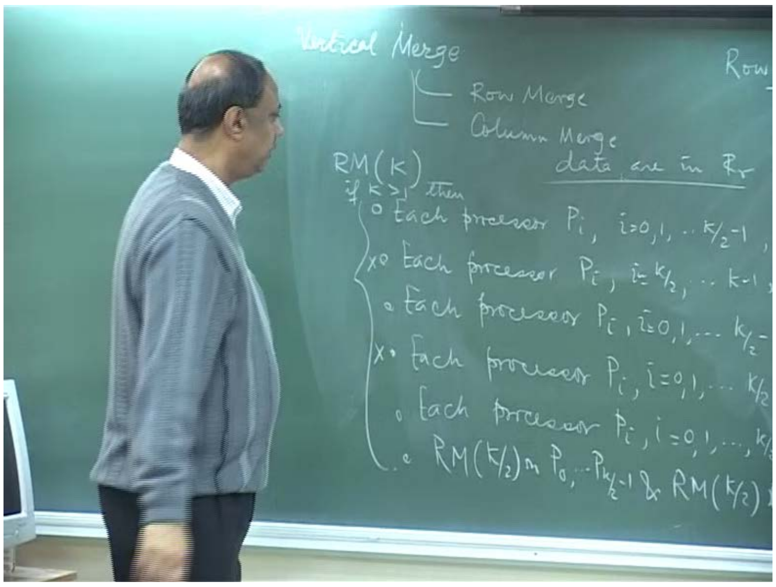
You have, now you have to move this data from here to here. How much time you need? How many routings you need? k by 2 routings you need. Again, you have to pump the data. So, another k by 2.

(Refer Slide Time: 31:44)

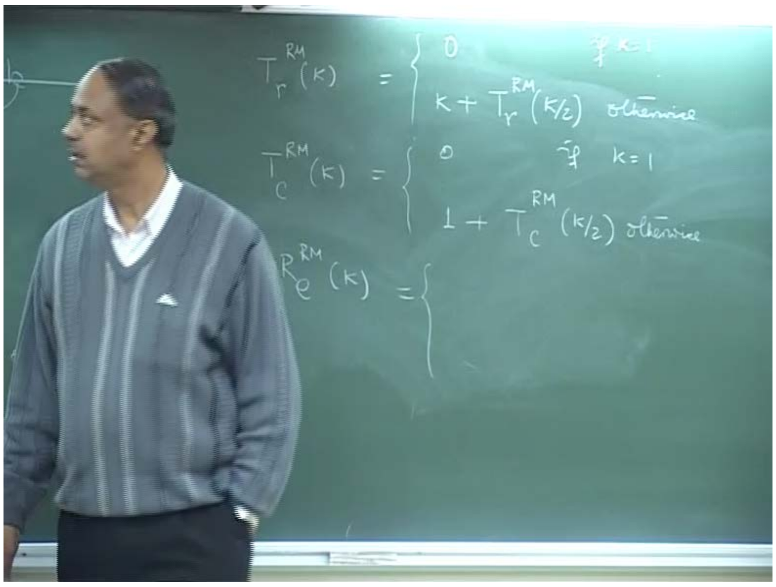

$T_r^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ k + T_r^{RM}(k/2) & \text{otherwise} \end{cases}$
$T_c^{RM}(k) =$
$R_e^{RM}(k) =$
exchange operation between R_p & R_s
operation between R_r & R_s

So, you have k plus time you need, on row k by two (( )). Agreed?

(Refer Slide Time: 32:14)

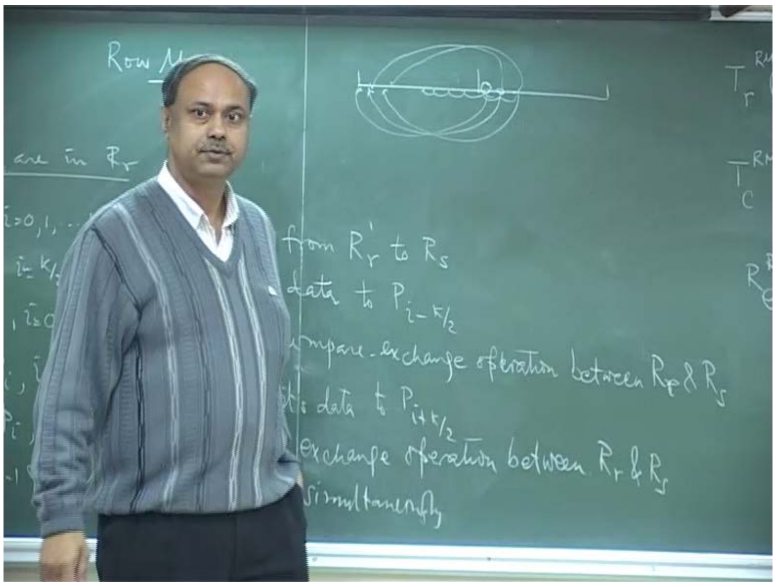

Vertical Merge
Row Merge
Column Merge
data are in R_r
RM(K)
if K > 1 then
Each processor P_i, i=0,1,.. K/2-1
Each processor P_i, i=K/2,.. K-1
Each processor P_i, i=0,1,... K/2-
Each processor P_i, i=0,1,... K/2
Each processor P_i, i=0,1,..., K/
RM(K/2) on P_0,..$P_{K/2-1}$ & RM(K/2)

(Refer Slide Time: 32:24)


$T_r^{RM}(K) = \begin{cases} 0 & \text{if } K=1 \\ K + T_r^{RM}(K/2) & \text{otherwise} \end{cases}$
$T_c^{RM}(K) = \begin{cases} 0 & \text{if } K=1 \\ 1 + T_c^{RM}(K/2) & \text{otherwise} \end{cases}$
$R_e^{RM}(K) = \{$

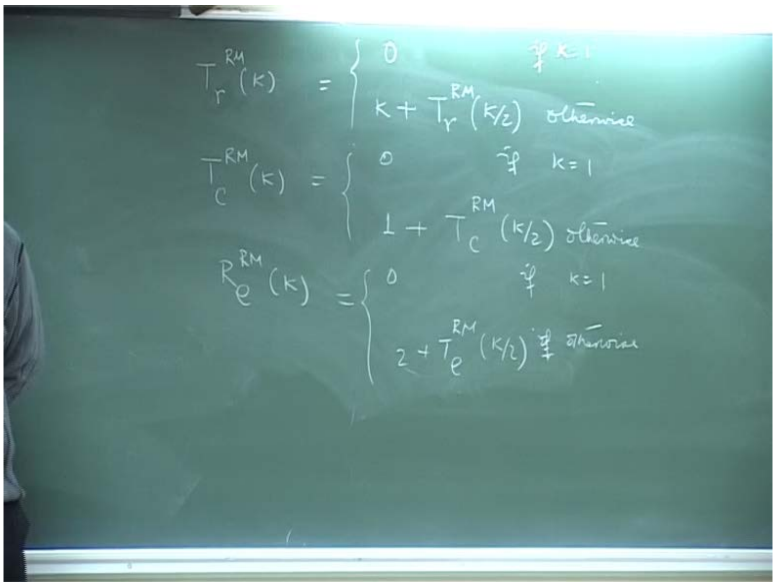
Now, so routing part is over. So, what is the routing? This one and this one. Now, can you tell me compare exchange? How many compare exchange? This is all can be (( )) simultaneously, plus t compare to, agreed? So, compare exchange is also over.

(Refer Slide Time: 33:23)


Row M
from R_r' to R_s
data to $P_{i-k/2}$
compare-exchange operation between R_r & R_s
its data to $P_{i+k/2}$
exchange operation between R_r & R_s
simultaneously

Then, exchange operations? It is parallel. Why I am sending the data from here to here? This comes from here to here. This comes here to here, but, how you are going to bring this element here? This will come here, here, here, here. While this element is here, this element is here, right. So, it is, see, this is the pipeline technique, right. So, you need by distance to cover. For him, this distance to be covered. For him, this distance to be covered, which is k by 2. Agree?

(Refer Slide Time: 33:59)



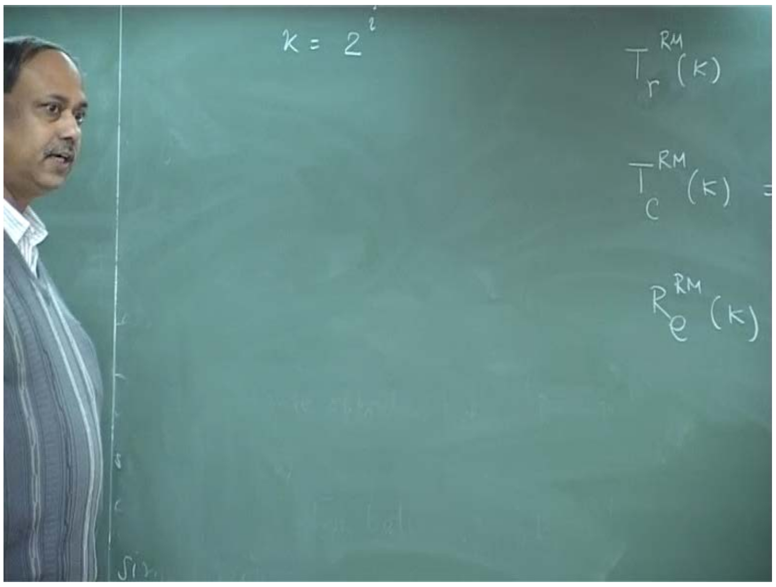
$$T_r^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ k + T_r^{RM}(k/2) & \text{otherwise} \end{cases}$$

$$T_c^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ 1 + T_c^{RM}(k/2) & \text{otherwise} \end{cases}$$

$$R_e^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ 2 + T_e^{RM}(k/2) & \text{if otherwise} \end{cases}$$

Now, 0, if t equals to 1 and 2 plus t e R M k by 2, then two on the (( )). Now, first let us find out the solution of this one? What is the solution of this?

(Refer Slide Time: 35:05)



$k = 2^i$

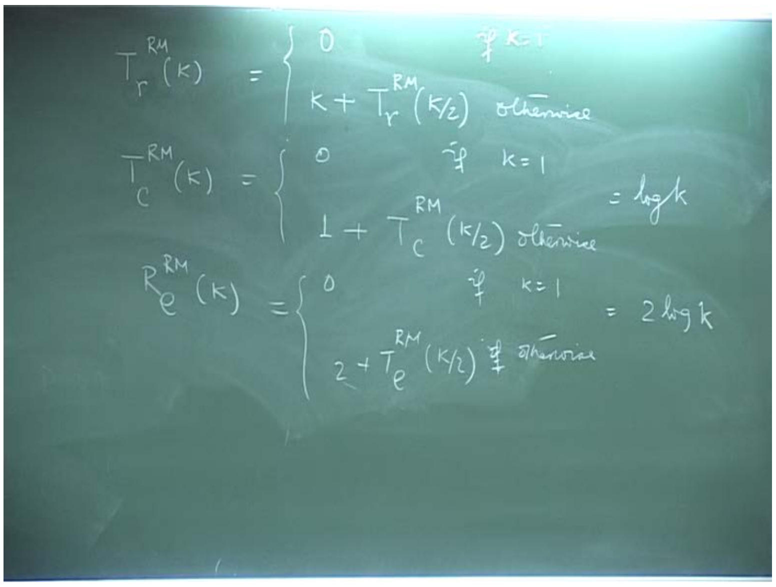
$T_r^{RM}(k)$

$T_c^{RM}(k) =$

$R_e^{RM}(k)$

Suppose k equals two the power 1 Are you sure i or i minus 1? I do not know. i or i minus 1? i? 100 percent right. I do not know. i? Fine. Then, it is log k. That is it.

(Refer Slide Time: 35:31)



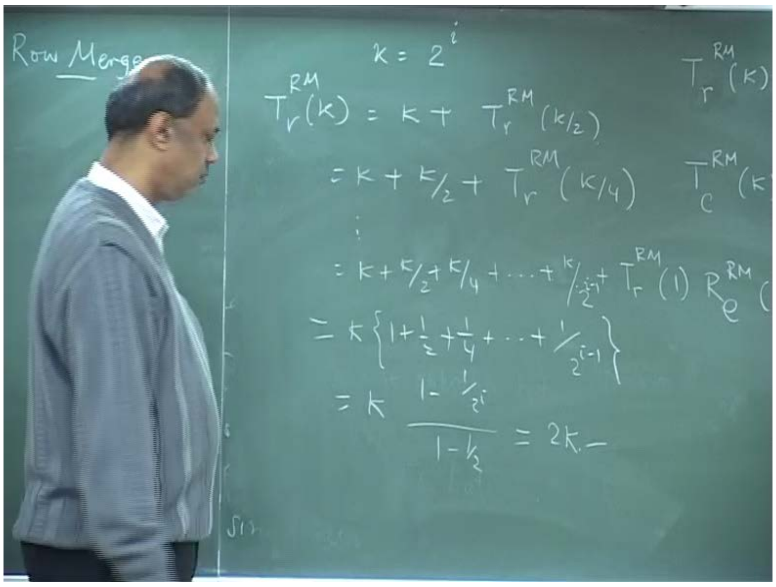
$$T_r^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ k + T_r^{RM}(k/2) & \text{otherwise} \end{cases}$$

$$T_c^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ 1 + T_c^{RM}(k/2) & \text{otherwise} \end{cases} = \log k$$

$$R_e^{RM}(k) = \begin{cases} 0 & \text{if } k=1 \\ 2 + T_e^{RM}(k/2) & \text{if otherwise} \end{cases} = 2 \log k$$

Then, you will be writing this is 2 log k. You please check whether it is i or i minus 1. Otherwise, it is log k. Agreed? So, because this is 2 times, constant times, so it is masters theorem. What about this part? k log k? How? Usually, k log k, then what for I am doing all these things? Because n log n is starting time and sequential merging time this n log n.

(Refer Slide Time: 36:22)



Row Merge

$k = 2^i$

$$T_r^{RM}(k) = k + T_r^{RM}(k/2)$$

$$= k + k/2 + T_r^{RM}(k/4)$$

$$\vdots$$

$$= k + k/2 + k/4 + \dots + k/2^{i-1} + T_r^{RM}(1)$$

$$= k\left\{1 + \frac{1}{2} + \frac{1}{4} + \dots + 1/2^{i-1}\right\}$$

$$= k\,\frac{1 - 1/2^i}{1 - 1/2} = 2k -$$

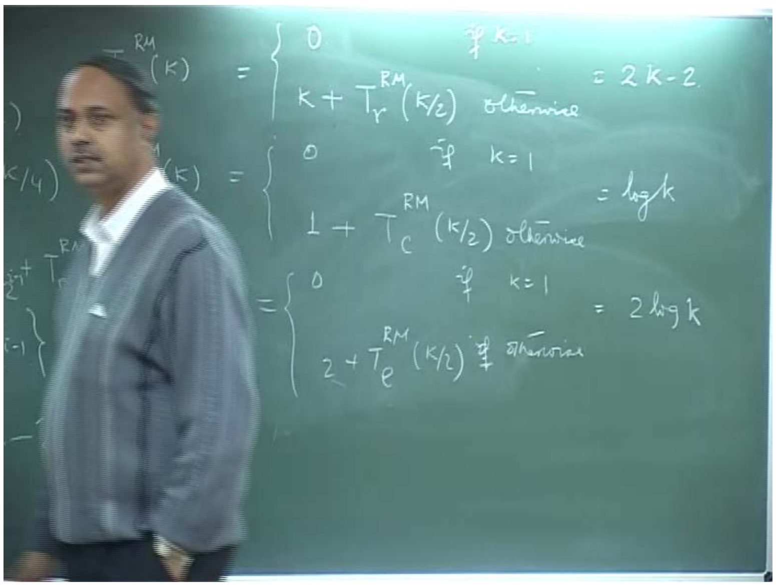
$T_r^{RM}(k)$

$T_c^{RM}(k$

$R_e^{RM}($

Let us see T r k R M is equals to k plus T r R M k by 2, which is k plus k by 2 plus T r R M k by 4 k plus k by 2 plus k by 4. When k, this is k, then I get 1 and k means 2 to the power i. So, when this is one, what I will write here? k by 2 to power n. No. Are you sure? When I am writing 4 here, I am writing 2 to the power 1. When I am writing 2 to the power i, it will 2 to the power i minus 1, right and this is 0. So, you get 1 plus half plus 4; 2 times k minus 2.

(Refer Slide Time: 38:37)



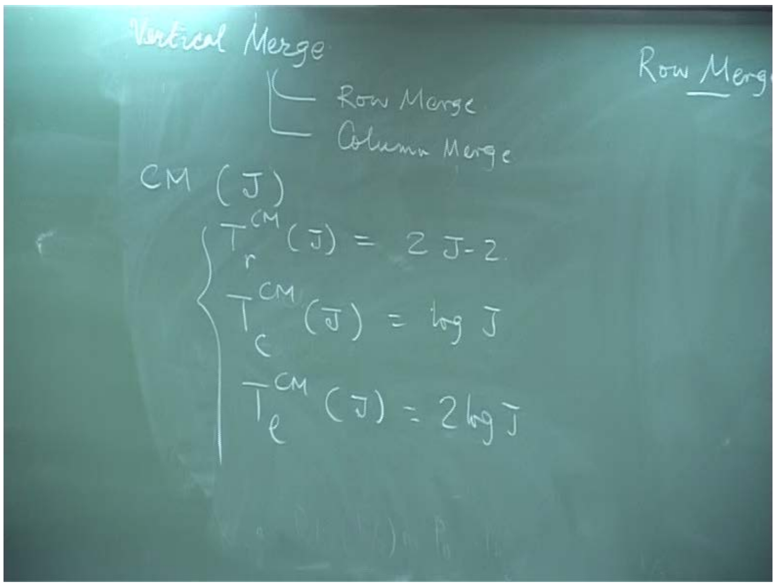
$$T_r^{RM}(K) = \begin{cases} 0 & \text{if } K=1 \\ K + T_r^{RM}(K/2) & \text{otherwise} \end{cases} = 2K-2$$

$$T_c^{RM}(K) = \begin{cases} 0 & \text{if } K=1 \\ 1 + T_c^{RM}(K/2) & \text{otherwise} \end{cases} = \log K$$

$$= \begin{cases} 0 & \text{if } K=1 \\ 2 + T_e^{RM}(K/2) & \text{if otherwise} \end{cases} = 2\log K$$

So, this solution is; is that ok? So, this is the number of routings you need. Now, if I tell that can you prove it that it gives the sorted sequence from a Bitonic sequence, right. Your input is in this row mode. Input, I told that the row contains a Bitonic sequence and after row merge call, I get a sorted sequence. Now, obviously this algorithm you have to prove that it gives you a sorted sequence.

(Refer Slide Time: 40:10)



Vertical Merge

- Row Merge
- Column Merge

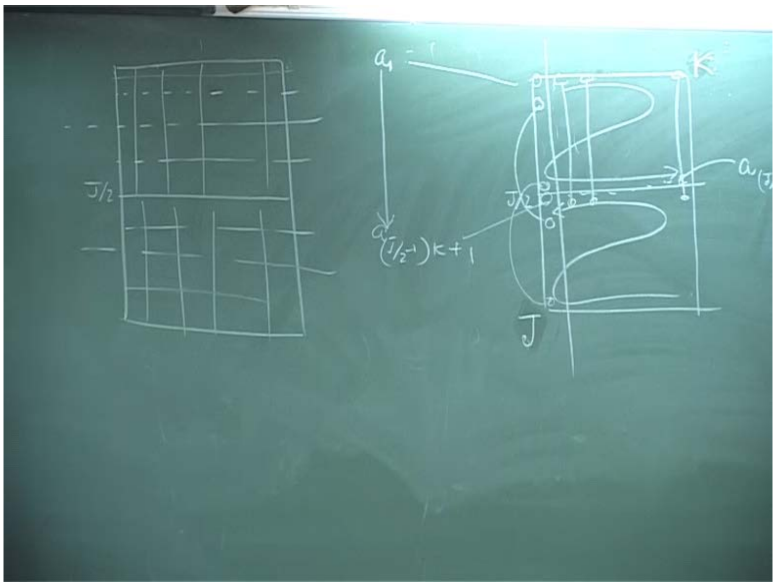
CM (J)

$$\begin{cases} T_r^{CM}(J) = 2J-2 \\ T_c^{CM}(J) = \log J \\ T_e^{CM}(J) = 2\log J \end{cases}$$

Any idea? Does it not obvious? Because, a i is compared to a i plus n and then, I am making it two rows, this row and this row and then, again we recursively calling. I am not changing any or I am not making any new strategy or new things I am introducing into that, right. Now, if it is case, so row merge gives you the sorted sequence and the second thing is that column merge. In the column merge, let us put the name C M and j is the number of elements in that column.

Same way, I can do it. Instead of assuming that it is in the form of row and I am assuming that it is in the form of column, right. So, column merge will also give you the sorted sequence form if the elements form a Bitonic sequence in that column. Agreed? So, the time complexity for this, I can write t r c m j is equals to 2 j minus 2 t c c m j, it is log j and t e c m j is your 2 times log j. Now, if you know the row merge and column merge, you really know what it does.

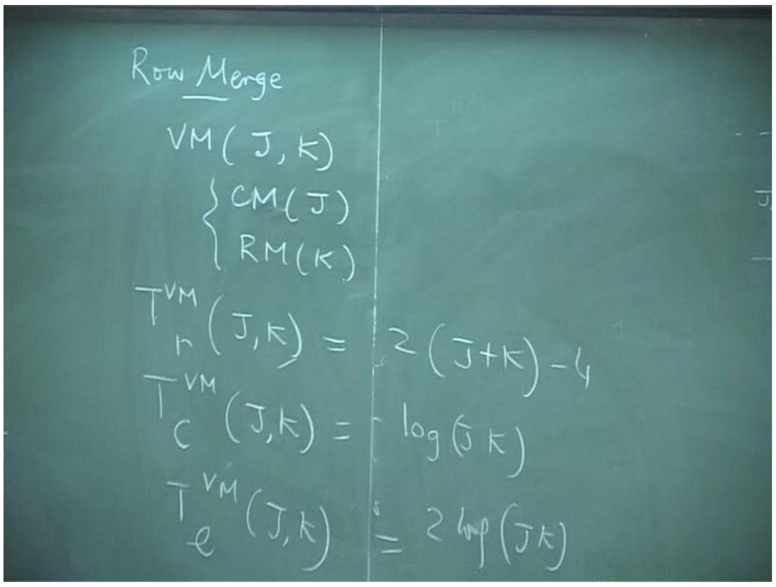
(Refer Slide Time: 41:27)



Suppose you have j cross j columns and j row and j columns, right. Suppose you have this and this forms a Bitonic sequence and this is in increasing order and this is in decreasing order and combining this, it forms a Bitonic sequence, right. Let us assume that this j by 2. So, first you, what you have to do? This element is compared with this element. a i is compared with a i plus n. So this, position of this is a of j by 2 minus 1 into k. This is the, well, elements position of this one and this elements position, if I expand it in 1 row, at a sequence, this position is a j by 2 minus 1 k plus 1, right and this is your a 1. So, this a 1 has to be compared with this elements. I did not or not?

Similarly, this element has to be compared with this element. Now, think about this element and this element has to be compared with this element. This element has to be compared with this element and so on. So, basically I can think that these are several columns you have. If I do the column merge, what will I get? If I do the column merge, I get a Bitonic sequence in this job and another Bitonic sequence in this job. Yes or no? So, but the size is j by 2. Again, you do this column merge again here. Columns merge here again and then, you get like this size. Again you do the columns merge and you get this size. Finally, you will get Bitonic sequence in each row. Agreed? This finds the property that these elements are smaller than these elements. These elements are smaller because that is the property of Bitonic. Now, you do the row merge. You do the row merge and then, you combine these things and you get the sorted sequence. Finally, you will get the sorted sequence.

(Refer Slide Time: 45:38)



Row Merge

$$VM(J,K) \begin{cases} CM(J) \\ RM(K) \end{cases}$$

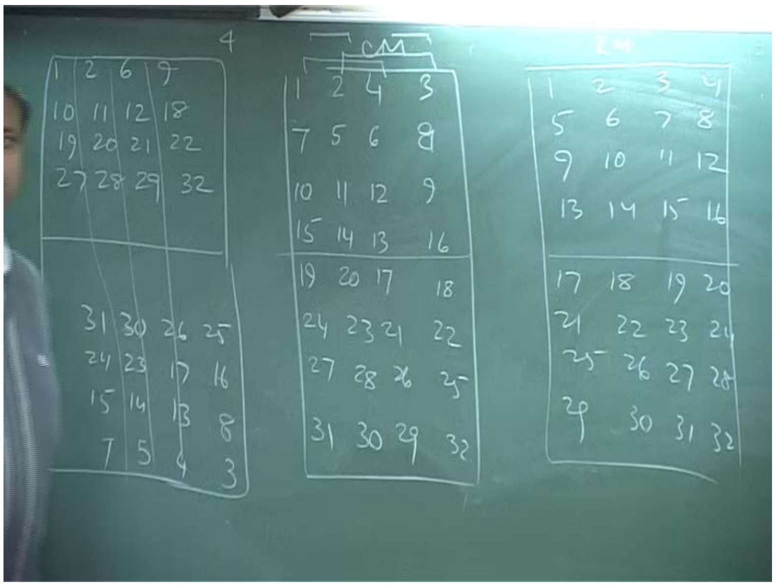
$$T_r^{VM}(J,K) = 2(J+K)-4$$

$$T_c^{VM}(J,K) = \log(JK)$$

$$T_e^{VM}(J,K) = 2\log(JK)$$

Tell me again? First row elements are smaller than the second row. But, within the row, that forms a Bitonic sequence. That is the, that is not sorted. The content of these elements is Bitonic sequence and these elements are smaller than these elements. These elements are smaller than these elements and so on. Now, you have to do the row merge here, right. So, vertical merge, if I write j comma k, first what you do? You do the column merge and then, you do the row merge. So, you do first column merge, then you do the row merge. This is your vertical merge algorithm. We will consider one example to see how it works. Now, the time complexity for that is t v m r routing j comma k is your 2 times j plus k minus 4, compare vertical merge j comma k is log j k, exchange vertical merge j comma k is 2 times log j k. Just I am combining these two algorithms.

(Refer Slide Time: 46:58)



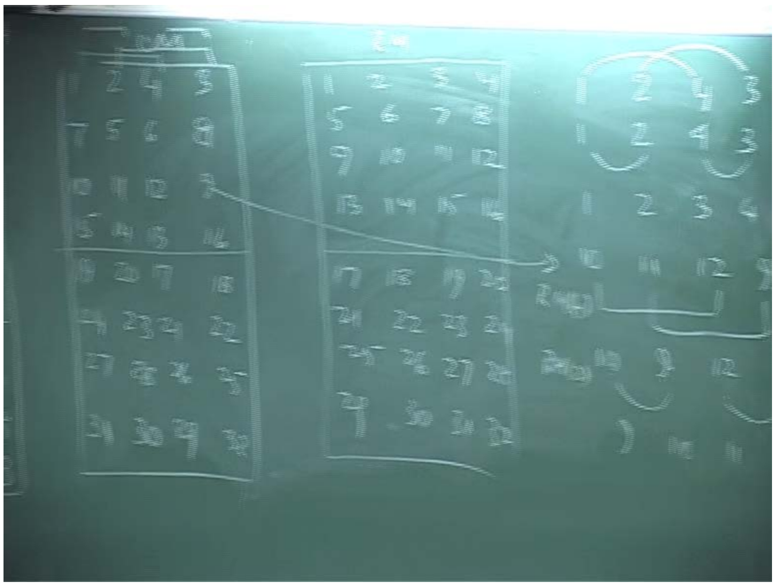
1	2	6	9
10	11	12	18
19	20	21	22
27	28	29	32
31	30	26	25
24	23	17	16
15	14	13	8
7	5	4	3

1	2	4	3
7	5	6	8
10	11	12	9
15	14	13	16
19	20	17	18
24	23	21	22
27	28	26	25
31	30	29	32

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32

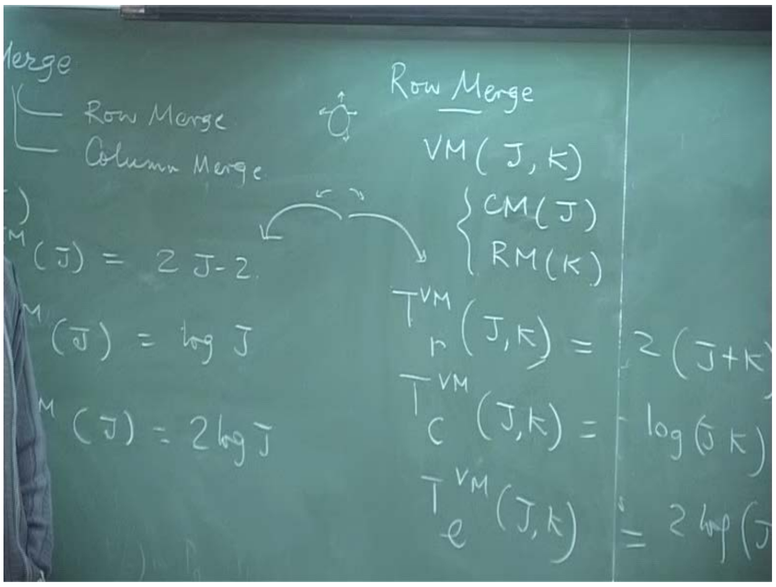
One example, you have, suppose 4 cross 8. But you have to give me 32 elements. Let us do here, 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32. So, you have, suppose this is the Bitonic sequence you have, right. This is in the increasing order and this is decreasing order. Now, if I do the column merge, assuming this is one column, this is another column, another column, so what I will get? What I will get? Tell me. If I do the column merge here? I will be getting 1 7 10 15 19 24 27 31. What about this one? 2 5 11 14 20 23 28 30 and then, you get 4 6 12 13 17 21 26 and 29, 3 8 9 16 18 22 25 32. So, column merge you will be getting that and you observe that every element has got its own row.

(Refer Slide Time: 51:59)



But within the row, it forms a Bitonic sequence, right. If you do the row merge, you will be getting, this is column merge. But while trying at home, you use all the steps of column merge. Within the column merge, you will be getting the column merge j by 3, then column merge j by 4 and so on. So you will be getting here row merge, 1 2 3 4 5 6 7 8, right. Because, you know, from here to here, it involves, first this type of thing, row merge 4 and then, row merge 2. Row merge 2 will be this one. Yes or no? Say, for this case, 1 2 4 3, so what you will be getting? Row merge these two. (( )) go back, so you will be getting 1 2 4 3. Then row merge size, so then only you will be getting 1 2 3 4, right.

(Refer Slide Time: 53:03)



For this case, you will be getting 10 11 12 9. First you do merging these. So, you will be getting, 12 is coming here and 9 will be coming here and accepted elements will be retained and rejected will be sent back. So, 10 will be here; 12 will be here; 11 will be there and then, row merge compares to 9 10 11 12. So, this is R M 4 and this R M 2. Which one? This one or this one?

(( ))

First one indicates yes and second one indicates no. See here, (( )) is tilted by this direction or this direction, that is here. Depending up on the south Indian or north Indian, all those things, but in both the case, if you get this and this; that is always known.