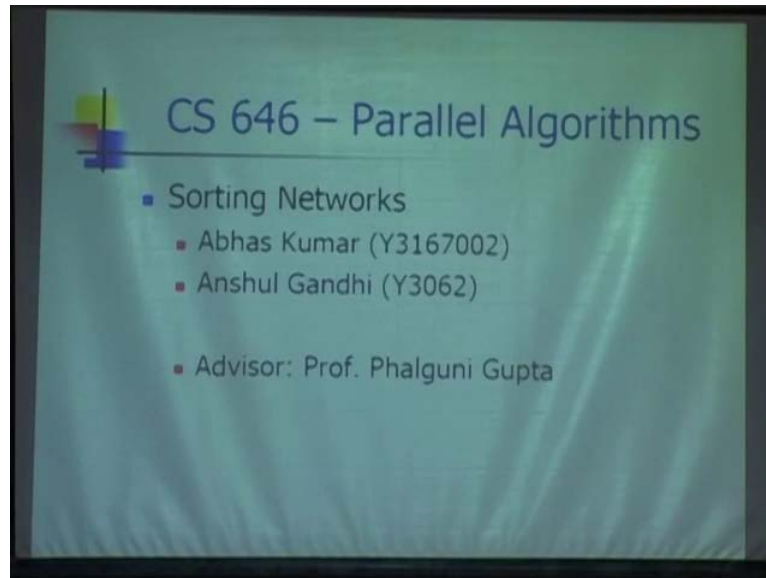


**Parallel Algorithms**  
**Prof. Phalguni Gupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kanpur**

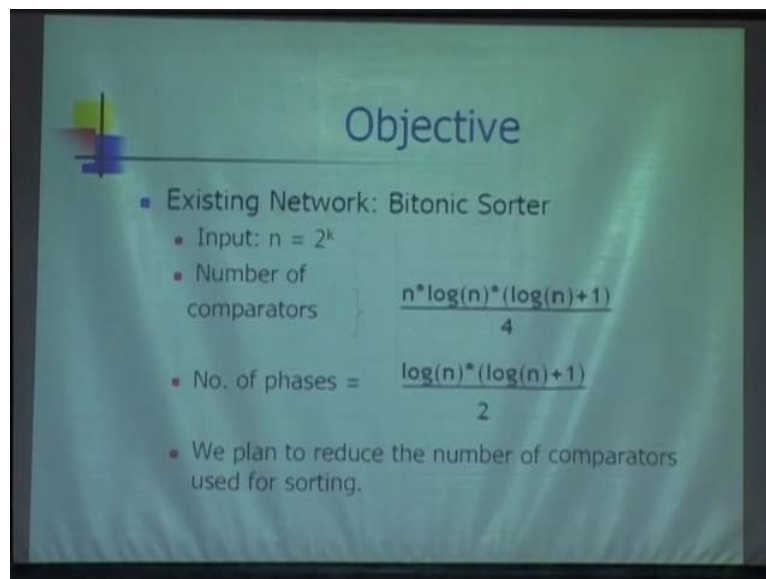
**Lecture - 23**

(Refer Slide Time: 00:17)



Hello everyone. We will be talking about a project, which was on sorting networks.

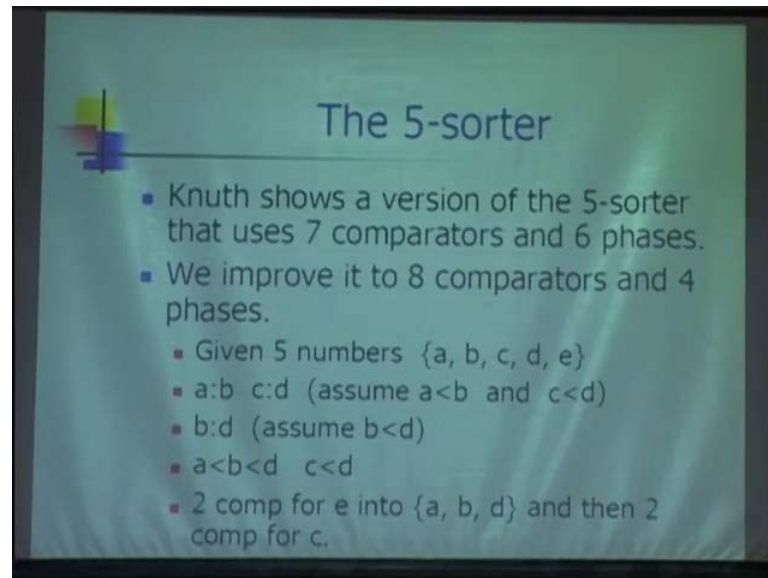
(Refer Slide Time: 00:20)



The problem is given set of input numbers. We wish to sort them. The already existing parallel network is the bitonic sorter. It uses the input of the form 2 power k. It has these

many numbers of comparators. Now, our problem is to design something better than this for at least one given value of  $n$ . Our target is only to reduce the number of comparators that are being used in total.

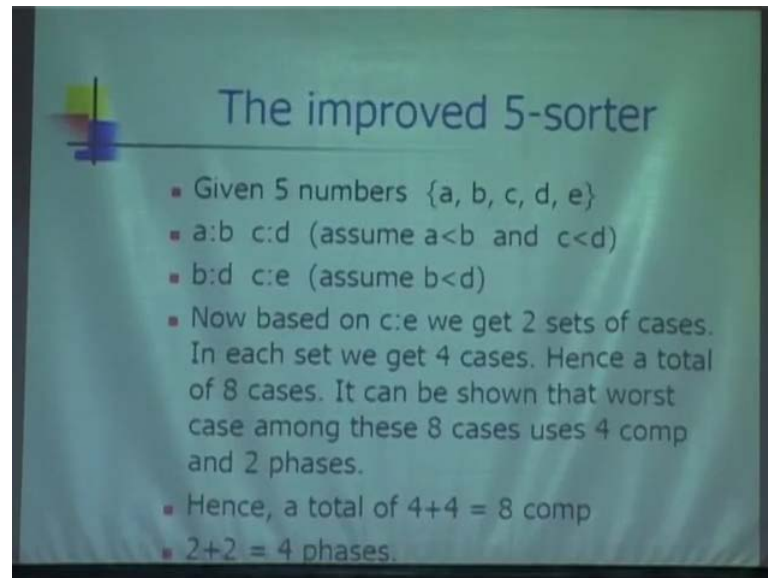
(Refer Slide Time: 00:47)



I will start by looking at the 5 sorter. If you look at Knuth, it shows a version of the 5 sorter. It uses 7 comparators. So, this is the problem given. We are given 5 numbers say  $a, b, c, d, e$ . You have to put them in a sorted order. So, what Knuth does is we first start with any 2 pairs. So, you take  $a, b$  and  $c, d$  and you compare them.

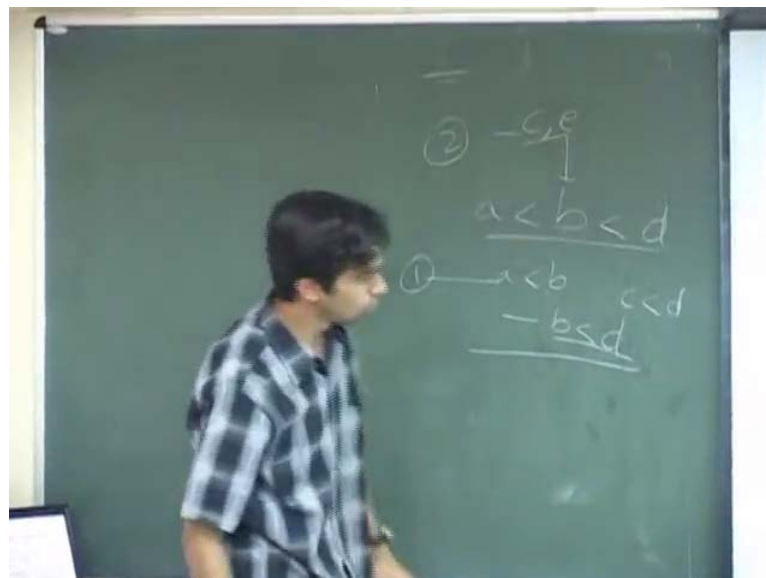
So, without loss of generality, we can assume that say  $a$  is less than  $b$  and  $c$  is less than  $d$ . Then, as a next step, we look at the greater numbers in this; are  $b$  and  $d$ . We compare them. Now, let us assume  $b$  is less than  $d$ . So, at this point with 3 comparisons, we have this particular order. Now, we wish to insert  $e$  into this part. We can do this by using a binary search. We used here. We should here. We will use 2 more comparisons and first see again, we will use 2 more comparisons. So, the total becomes 7.

(Refer Slide Time: 01:44)



But, the problem in this 1 is that it uses 6 phases. The 1 will carry to sequential algorithm. So, what we are going to do is we will improve that design. First, we will reduce it to 4 phases, but the cost is here. We will be using 1 more comparator from that 1. So, what we will do for that is the starting part we need.

(Refer Slide Time: 02:10)

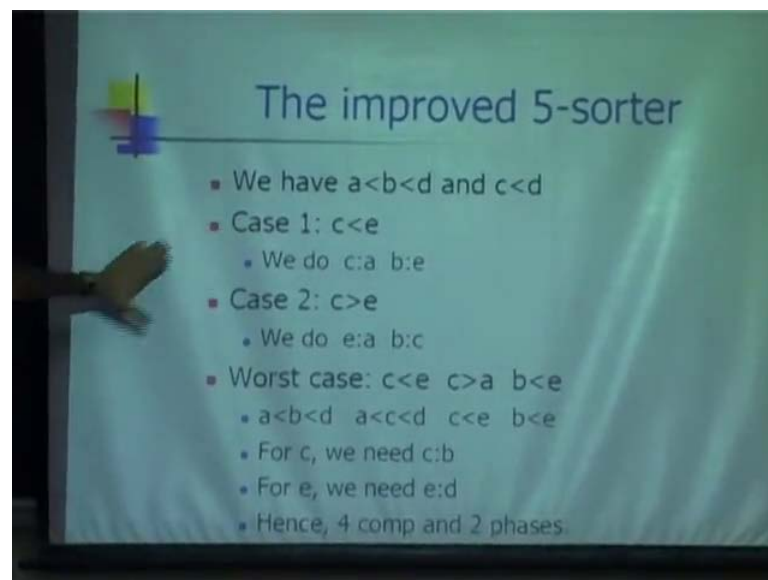


So, for now, the the first phase still remains the same. We should compare  $a$  and  $b$ ,  $c$  and  $d$ . Then, finding the greater of these 2, so again now our task is to insert  $c$  and  $e$  into this

at this point. What we do is we compare  $c$  and  $e$ . This will be done in parallel with  $b$  and  $d$ . So, because there is nothing common, we can compare these 2.

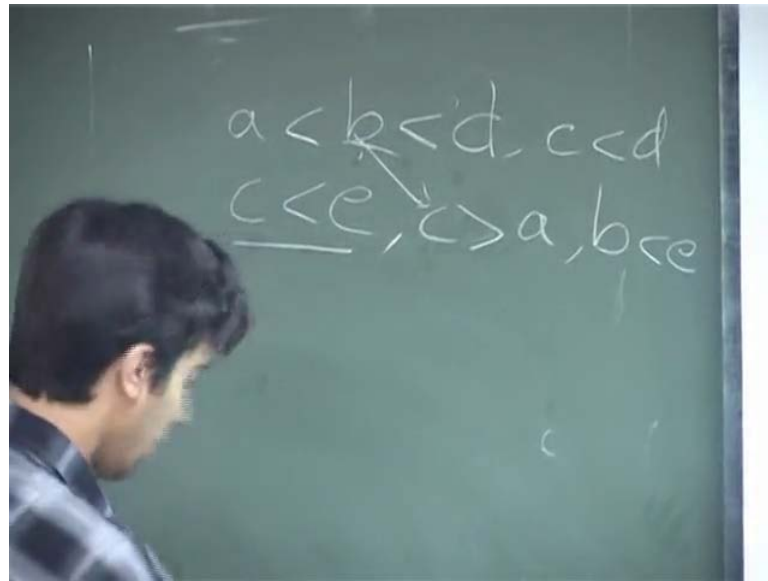
So, now based on the result of  $c$  and  $e$ , we get 4 cases in each of them. We will get a total of 8 cases. I just show that in the worst case, it takes 4 comparisons and 2 phases extra after this point. So, this point, we have 4 comparisons. Both happen in 1 phase; this and this happen in another phase. So, we will use the total of 4 phases and 8 comparisons for this.

(Refer Slide Time: 03:03)



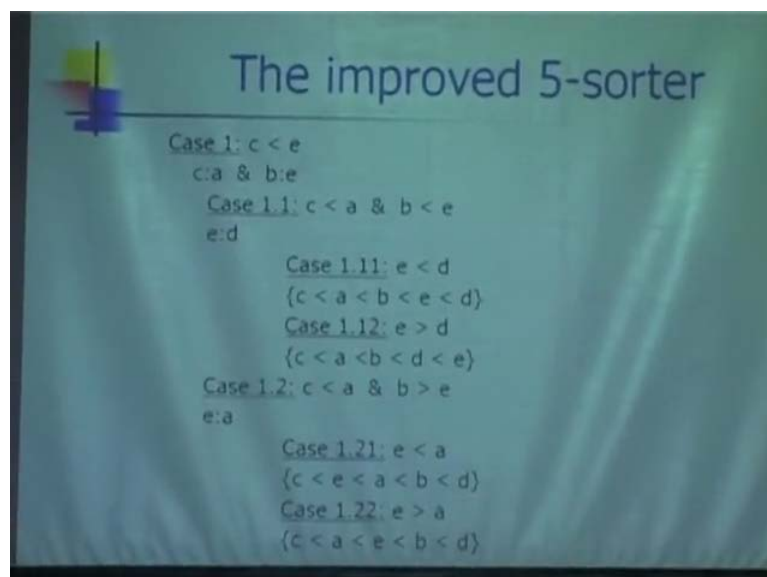
These are the cases. Here, I will talk about the worst case that exists. In the worst case, we have this.

(Refer Slide Time: 03:10)



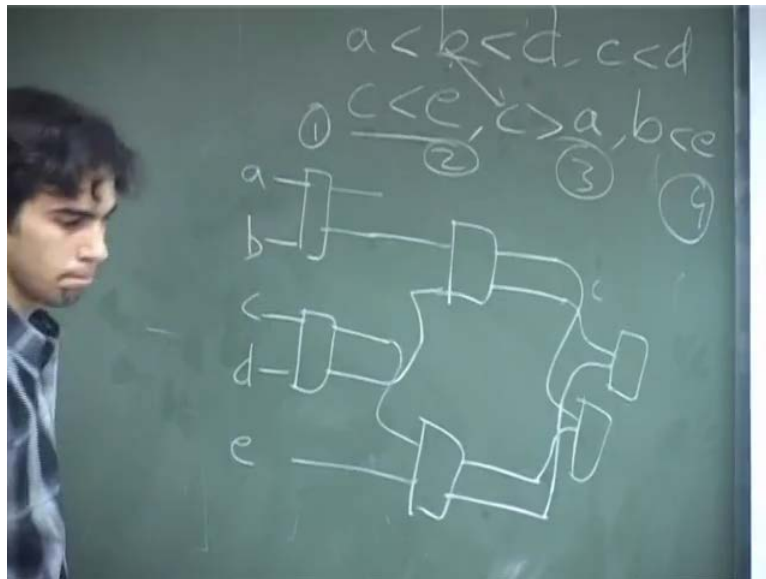
We have  $c$  is less than  $e$ ,  $c$  greater than  $a$ ,  $b$  less than  $e$ . So, we can see that since  $c$  is greater than  $a$  and less than  $d$ , we have even  $c$  less than  $d$ . To put  $c$  into its right position, we need to compare  $c$  and  $b$ . So, that is the first comparison that we will do and for  $e$ , we see that  $e$  is greater than  $c$ . So, in that case, we will compare  $e$  with  $d$ . For  $c$ , we need  $c$  and  $b$  and for  $e$ , we need  $e$  and  $d$ . So, with these 2 comparisons, we will be able to put  $c$  and  $e$  in to the right positions here. We already know what happened between  $c$  and  $e$ . So, we can put the entire file sorted together. So, this is where we have 4 comparisons and 2 phases. This is a case 5 analysis.

(Refer Slide Time: 04:03)



You just show me the network also. This is the first phase sir.

(Refer Slide Time: 04:19)



b is compared.

This is the second phase sir

b is compared with a, b.

a, b is in second first 1, second is b, d and c, e. So, b and d are larger ones. Then, we take c and e. So, this is the second phase. Now, we do b, d and c, e. Based on this result, it will be here. This will be the third phase and the fourth phase is only in here.

So, we have b, d, c, e sir.

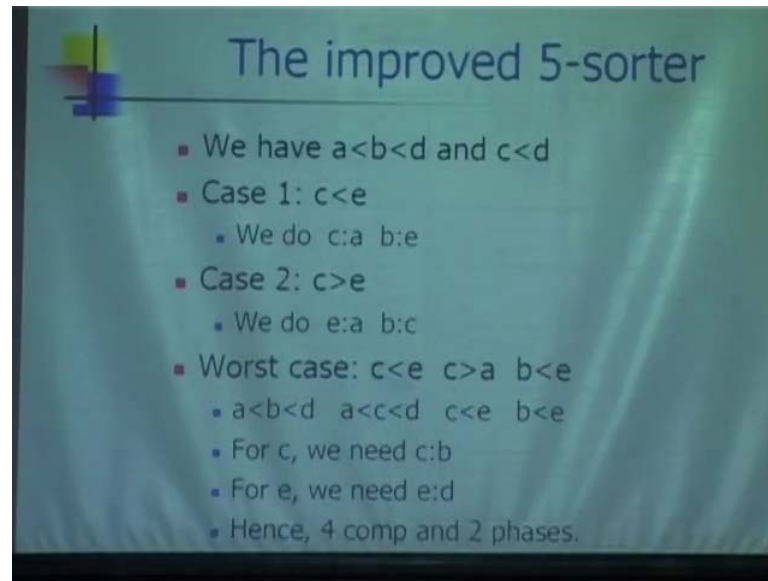
b, d, c, e.

So, this 1 sir, this is the second phase sir; b, d, c, e are in the second phase sir. This is the third phase sir. In case of this, we will go for this. We will go for this.

No, but you inter changed it, right? So, you always get right.

Yes, we get smaller, smaller one with a, which a way smaller sir, whichever will be smaller sir.

(Refer Slide Time: 06:01)



No, no, whichever will be smaller, you draw the diagram. Then, only I will be able to understand. Now, you you get...now, we take a.

Right sir.

a is smaller.

A is here.

This is minimum.

Right and this is going with the minimum of c and d sir. We compare c and d.

So, here this diagram, you have to give me otherwise with a, all those cases we have, this is the case I need.

I did according to case 5.

Case is ok.

Right.

Because, case is used to prove.

Ok.

That your d is correct. Now, ultimately according to the design, you have to go. This is your base based...

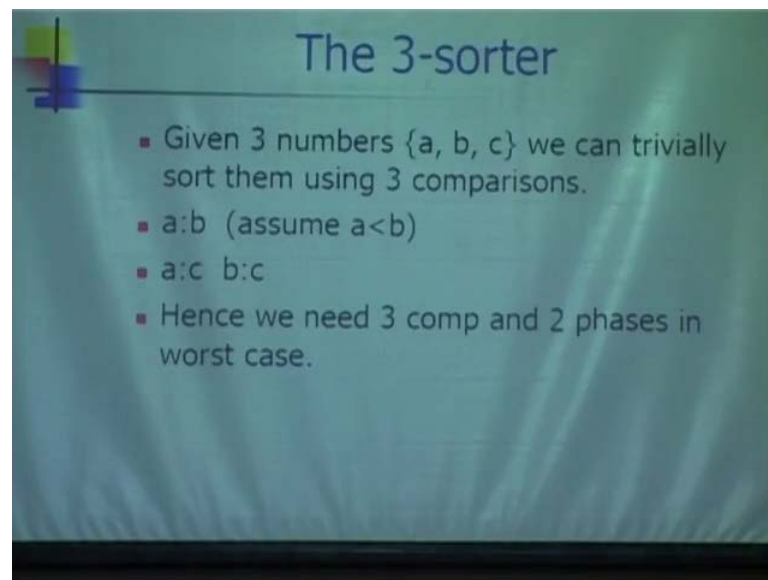
Based unit.

Unit that you will be use it.

Yes.

So, you have these structures again variable, but where you have to show this structure really get to, so for that, you have to give this.

(Refer Slide Time: 07:01)



Once the 5 sorters is done.

Let us assume that this is...

The 5 sorter, we will use another unit with it.

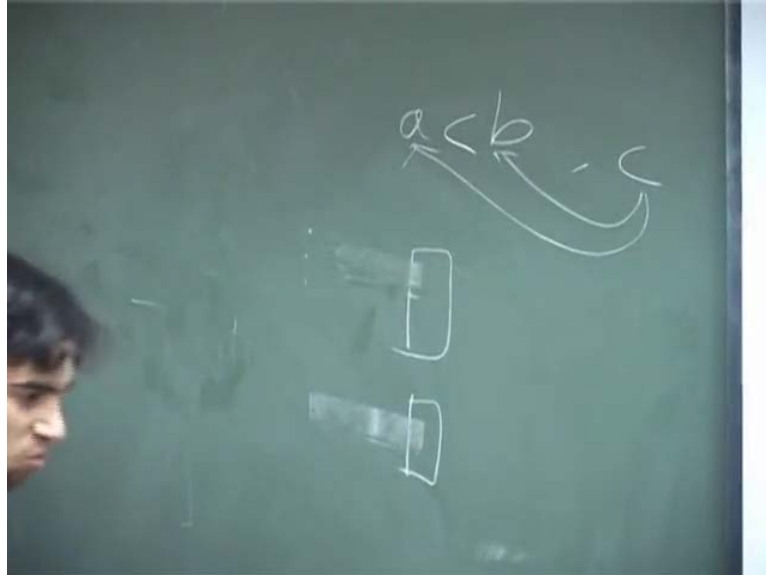
Because, it is other everything is gone.

No, sir. All all the cases are there report the cases we insert. Now, we look at the 3 sorter. This is a varied design, where we will design. Given 3 numbers, we just want to sort them and we are saying that we can do them in 3 comparisons and 2 phases this term. So,



we just take any 2 numbers initially and the third number, we compare with both with that.

(Refer Slide Time: 07:37)



But, we could even there, if you do not mind, what we are saying is we will take the 5 sorter as 1 unit, 3 sort as another unit and even this sorting will do it in parallel like as.

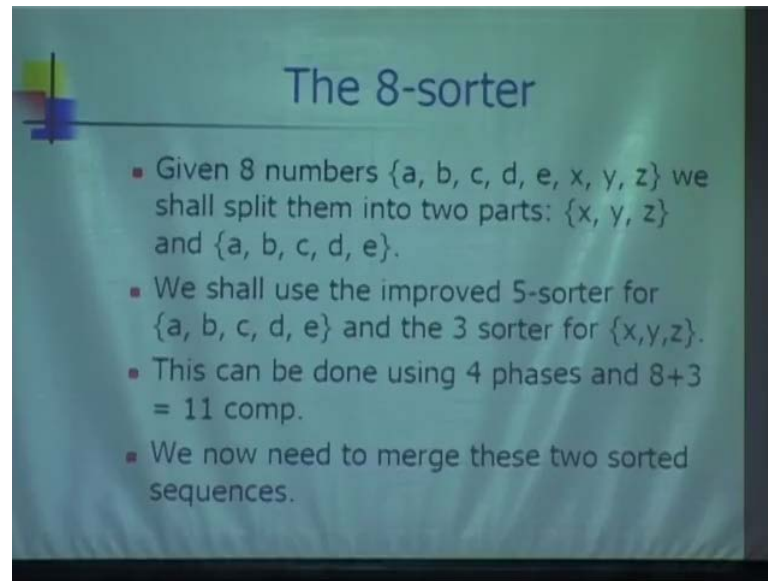
So, you get 8 sorter.

Yes sir, so 8 sorter is given as 8 numbers. The first 5 go here and the next 3 go here. So, beyond that...

So, up to this up to this, you have explained 4 phases right?

Yes sir.

(Refer Slide Time: 08:16)



So, from now onwards...

8 sorter, we can use that input 5 sorter and the 3 sorter and and this can be done in 4 phases and with 7 more comparators. We already we are using 11 comparator here for 5 sorter.

3 sorter, we are going to 3 phases.

Yeah and 4 phases or 5 comma 5 sorter, so they add in parallel; so all over there is 4 phases.

4 phases.

No, I have 1 question. Why you are making 5 entries? Why you have 5 entries?

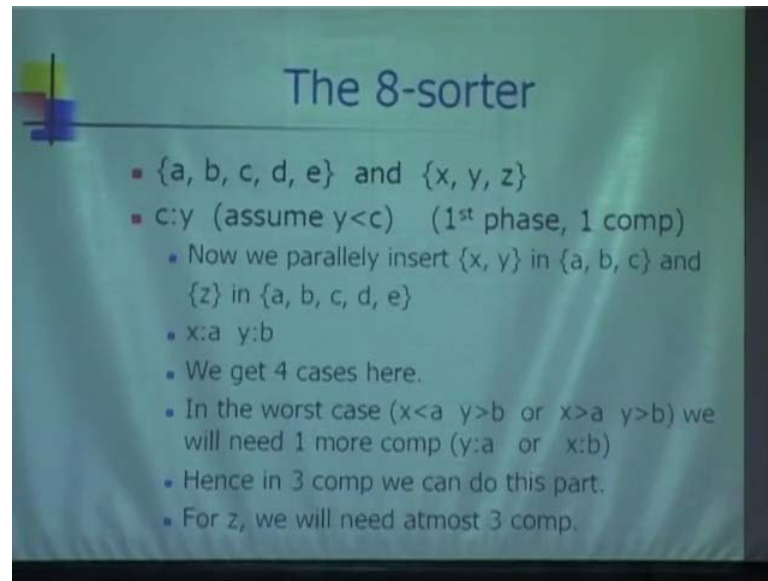
Sir, next class it can easier for us.

Ok, I will come to that point later on.

So, 8 plus 3 comparators for in this phase and for now, there will be 7 comparators.

Merge, we have to merge.

(Refer Slide Time: 09:06)



We have to merge both phases. To merge, we have certain cases.

If we have  $a b c d e$  and  $x y z$ , these are the 2 sorted sequences. To merge each, first we have to compare this with  $c$ . According to the result of this, we can in the next step, either we have to compare this with this and this with this or if it is greater than  $c$ , we have to compare this with this and this with this and this with this. So, without loss of energy, we can say that  $y$  is smaller than  $c$  in that case.

Now, in that case...

See the way you are telling, I do not know how you are going to show with the form of network. You understood?

Right. This is our basic unit sir.

So, basic unit...

It is merging after merging this.

Merging, also you have to give me the...

Yes sir.

So, problem...

Yeah, merging we will do.

(Refer Slide Time: 10:27)



Then, we have to insert in a b c x y and in a b c d e c d e z. This is the 2<sup>nd</sup> case. After this case, that y less than c, we have to compare y with y with b and x with a and and z can be inserted in 3 groups. First compare with this. Then, if it is greater than this, we have to compare with d. Then, compare with e. If it is smaller than c, we have to compare with b and then with a. So, in 3 steps, z will be inserted. For this, there will be 4 cases.

If x is less than a, if x is less than a, then it will be directly inserted here and y will be less than b, then we have to compare one more with a, so that we can know that the exact position of all. Then, we have this will be the x y a b c d e will be the sequence. If y is greater than this, then this will be less in case. In the second case, if x is less than a, then it will be here and y is y is greater than b, and then y will be in this position. There will be no comparison in this case. Then, there is a third case when x is greater than a. When x is greater than a, then, we we need 1 more comparison for x with b to insert either x here or here and similarly, is the 4<sup>th</sup> case.

So, here we need 1, 1 for the first comparison y with c. Then, 3 comparison for this to insert z and 3 comparison to insert x y; so total 7 comparison and they are 4 phases 1 phase for y to compare with c and then we may need 3 for this and 3 for this. Those are in parallel. Overall, they have 8 phases in our 8 sorter and 8 comparisons. We have 24

comparison bitonic and in all event, 20 comparisons. We can use this 8 sorter array of this.

In number of phases in bitonic...

Which 7...

7.

On it is sorter...

Or even 8 phases...

7 phases.

So, what you are doing with the minimum of number of comparators to be less?

We take less and more, more sir, 1 more sir.

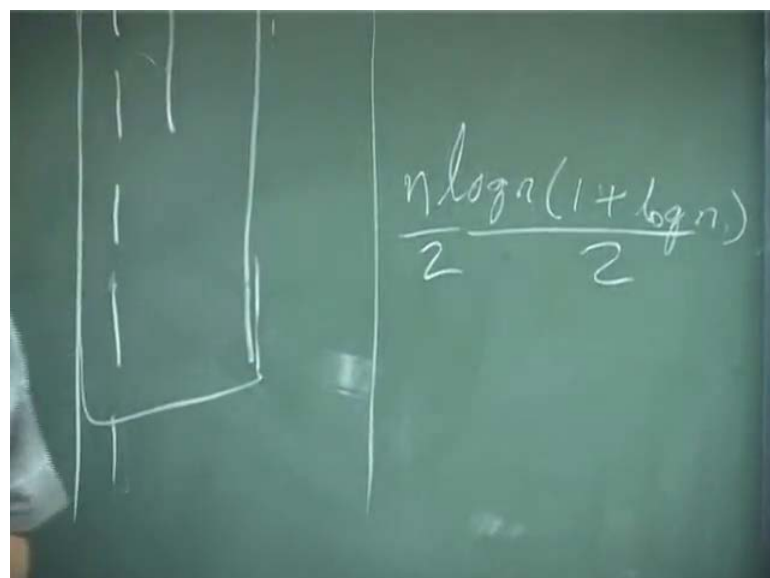
So many...

Network yes sir.

Your diagram is not I will not...

Yes sir. So, we can use that 8 sorter. We have basically here. 1 minute sir.

(Refer Slide Time: 13:16)



Bitonic sorter, in bitonic sorter, if it is  $n$  elements we have we use 2, 2, 2, then 4, then 8 and then 16.

So, for till 8, we can use our 8 sorter. In this way, we can we can reduce the number of comparators by 3  $n$  by 4 and 1 of the example is...

3  $n$  by 4 comparator to study...

Yes sir.

That is one fourth of the total number you are using, are you sure?

Yes sir.

Go back, go back, go back, go back.

Yes sir.

Go back in  $j$   $k$  comparator, in each 8 minute...

Yeah, 6, 6 and in each 8 minute in bitonic and...

In each...

8 minute in bitonic, we are using  $\log \log n$  plus 1 plus  $\log n$  by 2 into  $n$  by 2, these many comparators in bitonic.

You try to understand.

Yes sir.

You are saying it is only base 8 of 81.

Yes sir. Yes sir. Yes sir.

Right.

Yes sir.

Here, you are using 18 comparators, again 24.

Yes sir.

Waste that is also again 24.

Yes sir.

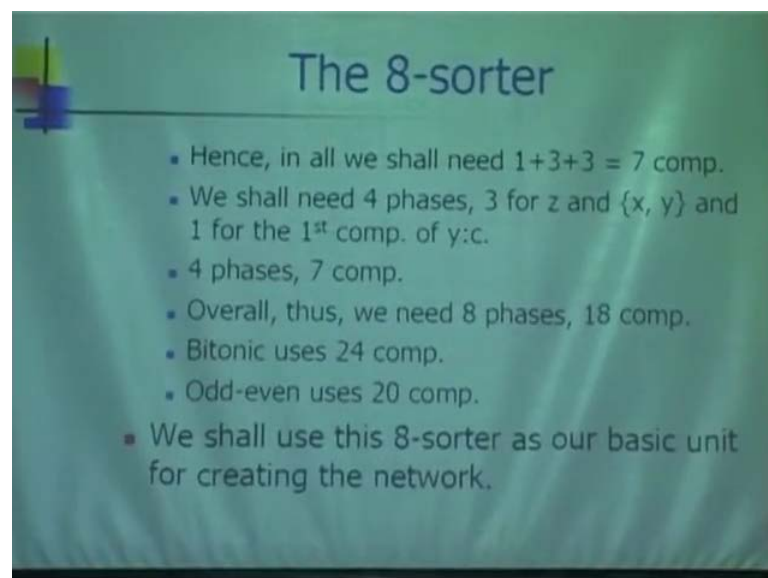
Waste is the same.

Then, the waste is the same thing.

There is the change.

Yes sir.

(Refer Slide Time: 14:41)

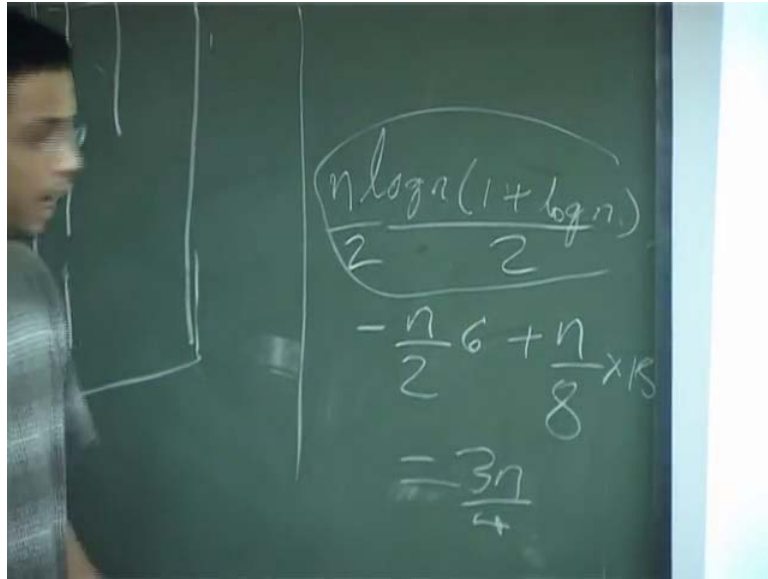


Only this cluster, that clusters. What is the total number of comparative you are using?

There is change.

This is the total number of comparators in bitonic. We are reducing the first 6 phases till the 8.

(Refer Slide Time: 14:57)


$$\frac{n \log n (1 + \log n)}{2}$$
$$- \frac{n}{2} \cdot 6 + \frac{n}{8} \cdot 15$$
$$= \frac{3n}{4}$$

It is that we are reducing  $n$  by 2 into 6 and we are adding the number of comparators. We are using from our base unit that is  $n$  by 8 into 18. So, this will be this will remain same and this gives 3  $n$  by 4 this 3  $n$  by 4 after subtraction.

So, then we have  $n$  over here say  $n$  by 12.

We are dividing this into 8 units  $n$  by 8 unit sir. In each unit, we will replace whatever bitonic in doing by 8 sorter. So, we have  $n$  by 8 into 6 comparators.

That is you are telling 3  $n$  by 4 divided by  $n \log^2 n$  gives the actual percentage of...

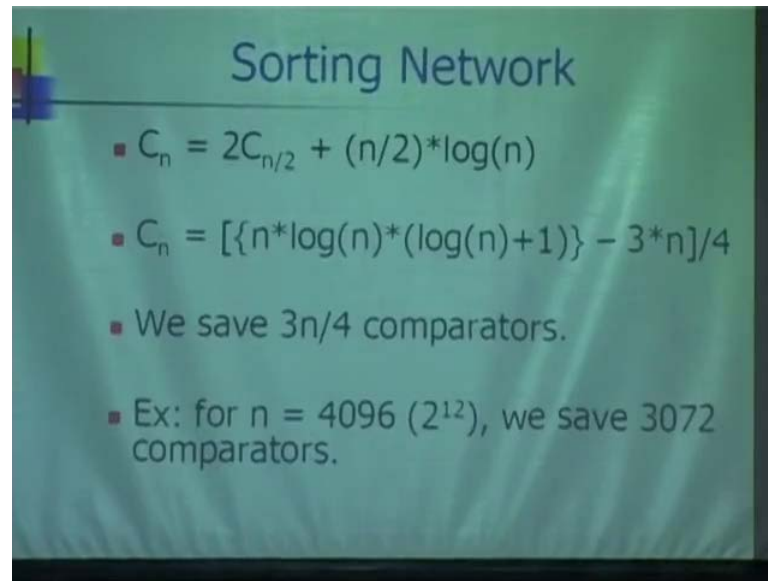
Yes sir.

So, that what you have to write...

What is the...



(Refer Slide Time: 15:57)



### Sorting Network

- $C_n = 2C_{n/2} + (n/2)*\log(n)$
- $C_n = [\{n*\log(n)*(\log(n)+1)\} - 3*n]/4$
- We save  $3n/4$  comparators.
- Ex: for  $n = 4096 (2^{12})$ , we save 3072 comparators.

So, you you did not. I am telling you one is that you make the diagram, network diagram. Second thing is that you have to prove.

Yes sir.

You need theorem diagram for that. It is like decision diagram. You know decision tree? You check; this is the 3 sorter or the 5 sorter, then the decision field...

Yes sir.

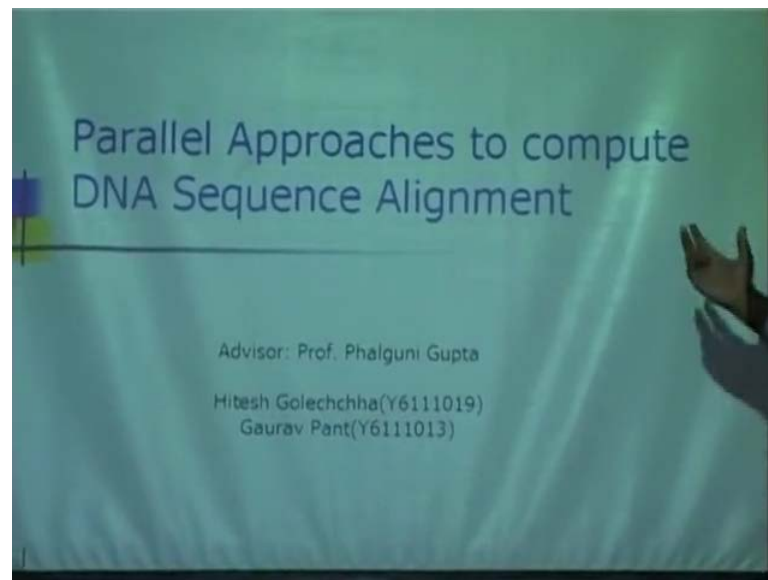
So, similar type of case, we have to do whatever you begin case wise that is also in terms of tree less than b.

No, no, overall we are using 2 phases extra, nearby 4 comparators, but retaining 2 phase comparator in 8 sorter part, we are using 2 modes and that is all nowhere else.

So, you find out also complexity, cost and other things.

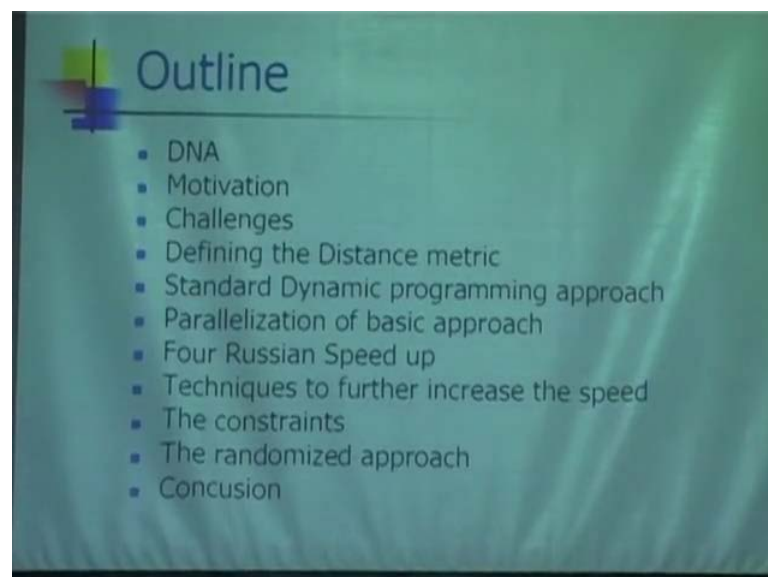
Yes sir.

(Refer Slide Time: 17:18)



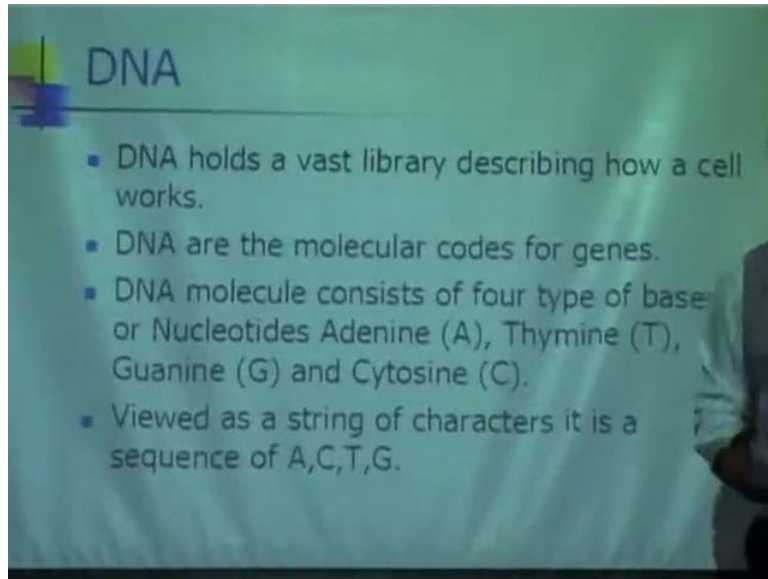
Good morning to all of you. Our problem is parallel approaches to compute DNA sequence alignment.

(Refer Slide Time: 17:26)



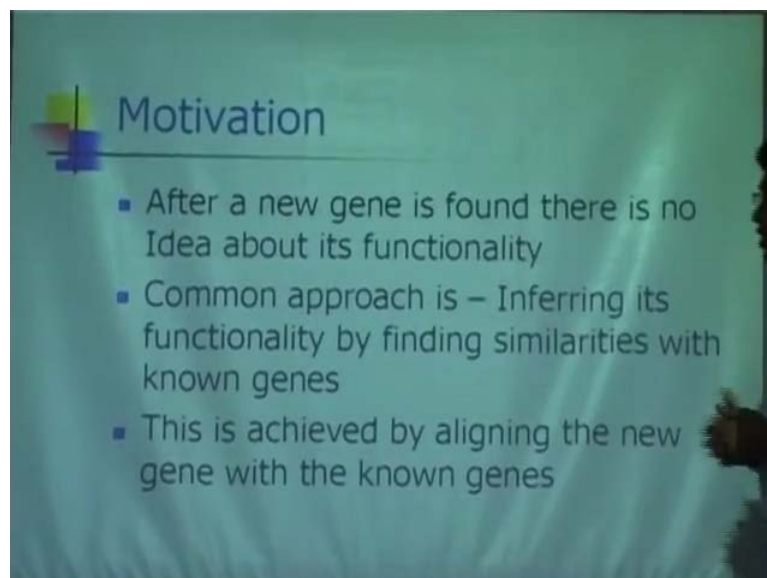
This is the outline of our work.

(Refer Slide Time: 17:29)



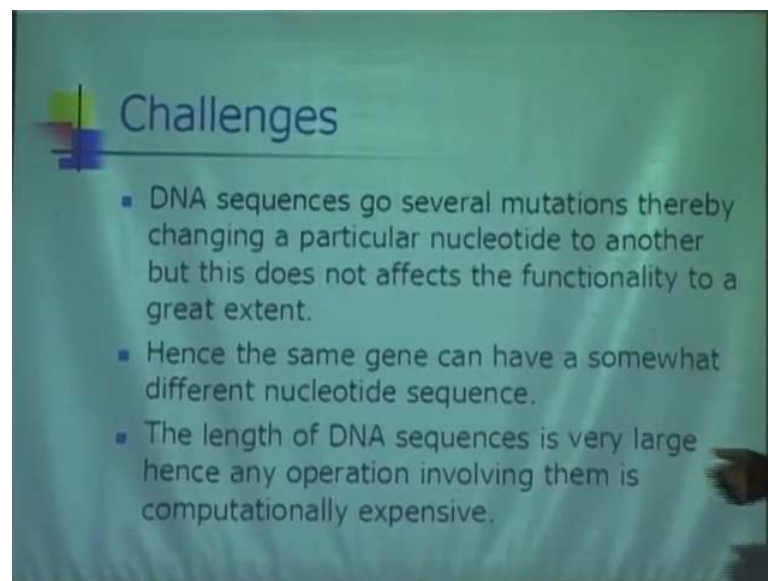
First of all, the the problem is DNA sequence dimension. We need to understand basically what the DNA is and how it is important to align the DNA sequence. So, first of all, DNA holds a vast library describing how a cell works. Basically, DNA is the molecular codes for the genes. So, DNA consist of 4 types of nucleotides A, T, G, C. So, from our problem point of view, we need to understand that basically DNA is nothing but the sequence of these 4, 4 types of nucleotides.

(Refer Slide Time: 18:09)



So, this is the motivation for this problem. Basically, whenever a new gene is found, there is no idea about the functionality of that particular gene. So, what we did? What we do? We try to infer the functionality of that particular gene with the help of the existing genes, which function and whose functionality is already known. So, basically what we try? We try to align that particular gene where DNA sequence with the existing sequence and whatever the matching in between that particular in this DNA sequence, on the basis, directly we try to define the functionality of that particular gene.

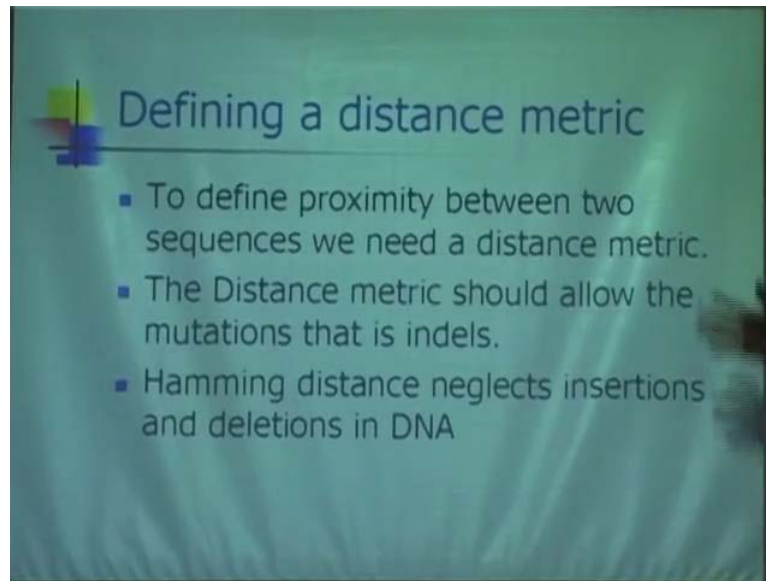
(Refer Slide Time: 18:52)



Now, what are the challenges in the DNA sequence alignment? Basically, in DNA sequence, sequences, there is a thing, which is called mutation. Basically, what happens is some nucleotides keep keep around changing to other. So, basically this mutation does not affect the functionality functionality of the DNA molecules.

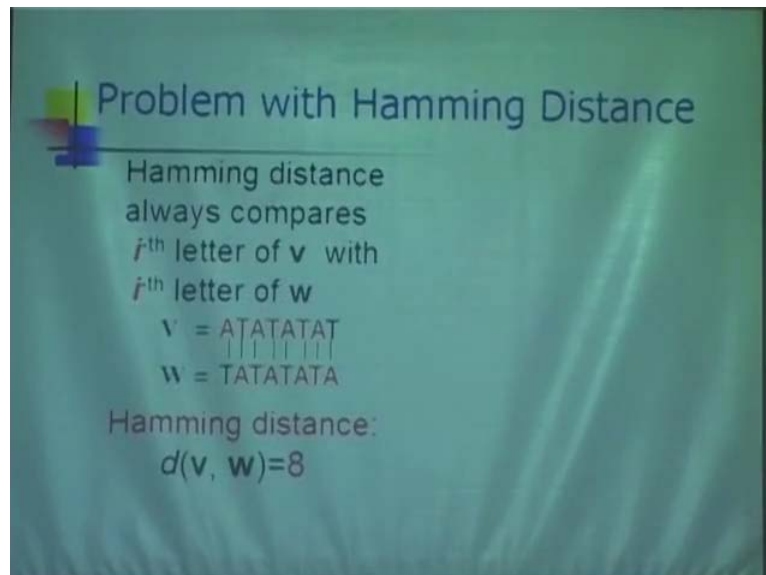
Suppose that there are some nucleotides, which change into some other nucleotides. So, basically the functionality remains the same. So, so, many times, we we will do not meet the exact excrement and those are different. This DNA sequence may have the same type of functionality.

(Refer Slide Time: 19:36)



So, first of all from technical point of view, we are going to define a distance matrix. So, to define proximity between 2 sequences, we need a distance metric. Basically, we are going to assign 2 sequences. So, we need some some kind of measurement to measure how close those 2 sinks each other. So, we are going to consider the hamming distance.

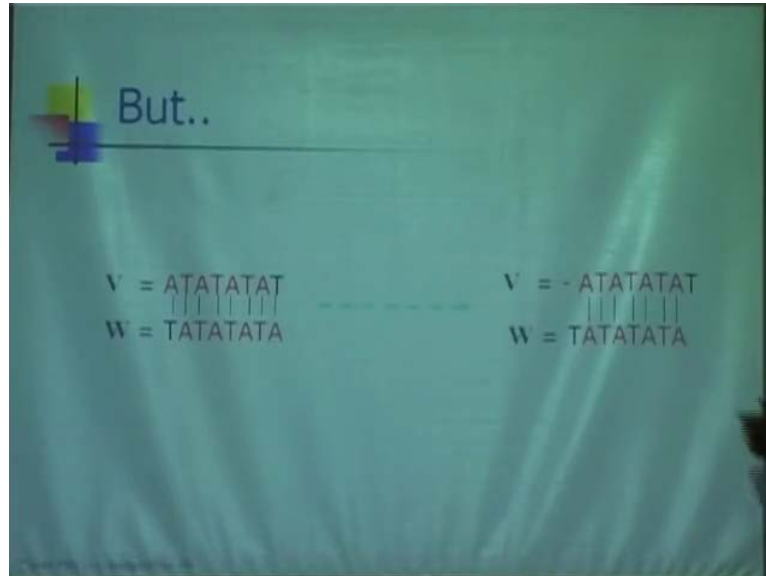
(Refer Slide Time: 19:58)



So, the problem with hamming distance is that the  $i^{\text{th}}$  letter of first string is compared with the  $i^{\text{th}}$  letter of the second string. So, you can just see. If we are going to compare

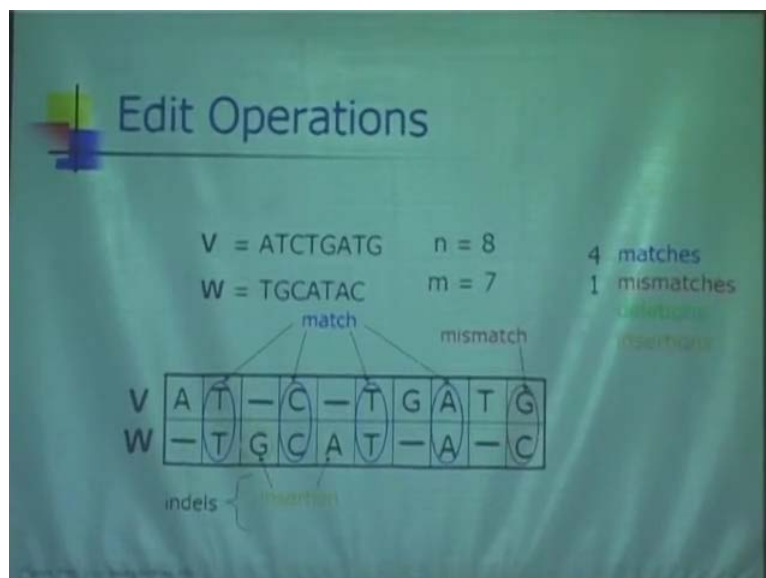
these 2 strings, the hamming distance is not the very good idea to compare these 2 strings.

(Refer Slide Time: 20:16)



But, if we simply make 1 shift to 1 string, you can see these 2 strings are almost the similar.

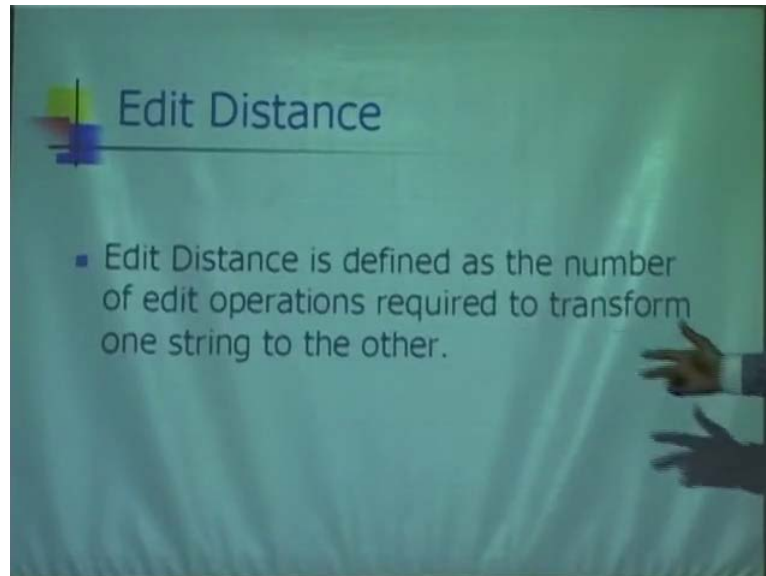
(Refer Slide Time: 20:25)



So, here here is the concept of edit operations. What we are trying? We are trying to calculate the edit distance other than the hamming distance. So, basically there are 4 types of things in the edit edit the edit operations. First is match mismatch deletion and

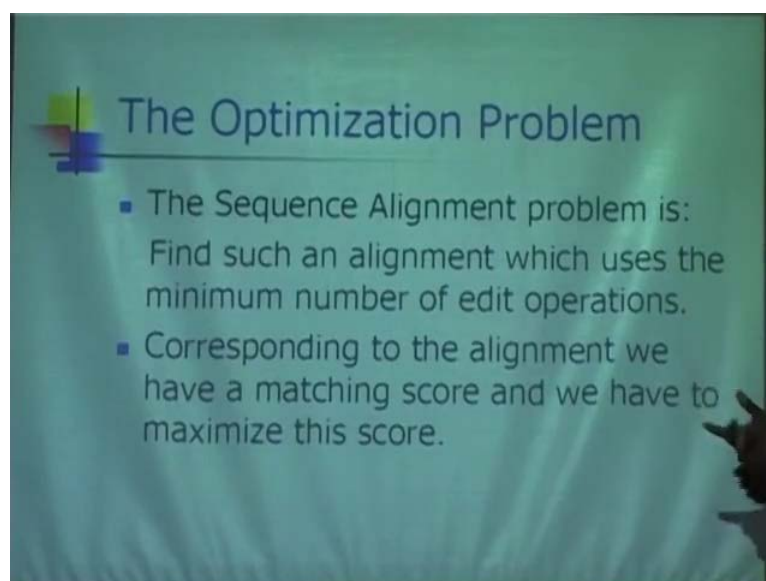
insertion, which you can see whenever the 2 characters in the strings are going to match this kind of match.

(Refer Slide Time: 20:46)



Here, you can see. This is the mismatch. This operation is going to represent the deletion. This is the insertion. These are edit operations. Basically edit edit distance is nothing but it is the number of edit operations required required transforming 1 string into another string.

(Refer Slide Time: 21:08)

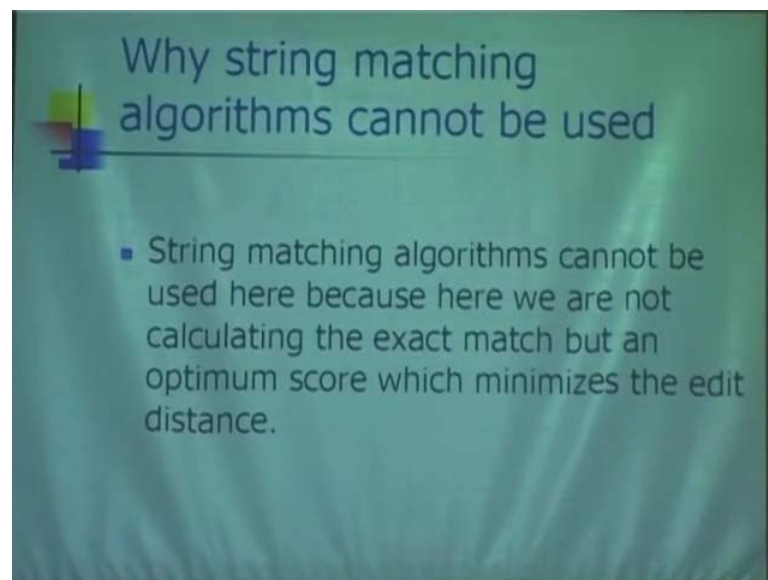




Now, basically what the problem is? It is basically the optimization problem. The sequence alignment problem is find such an alignment, which uses the minimum number of edit operations. You want to convert 1, 1 sequence into another sequence. You want to optimize the number of edit operations; optimize in the sense we want minimum number of edit operations, the same.

The other half flavour of the same problem is that you whenever you match with you align it to a sequences and whenever match occurs, you are you are you have assigned some score to that particular match and penalty for the mismatch. So, corresponding to the alignment, we have to we have a matching score and we have to maximize this score.

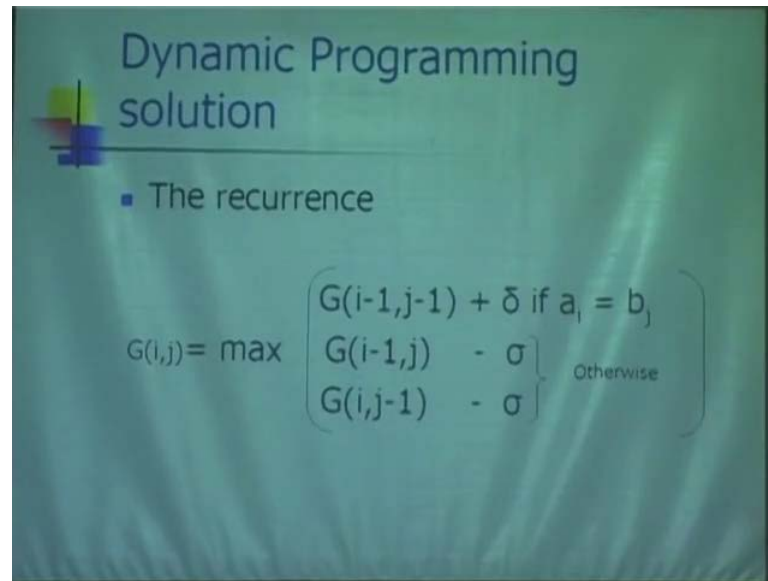
(Refer Slide Time: 21:54)



Now, why string matching algorithms cannot be used? Actually, string matching algorithms take up a pattern and try to search for that pattern in the given text. It finds, it tries to find the exact match, but here what we want is that we want a score, which corresponds to the matches and insertions and deletions. So, we cannot use the string matching algorithms, the standard string matching algorithms.

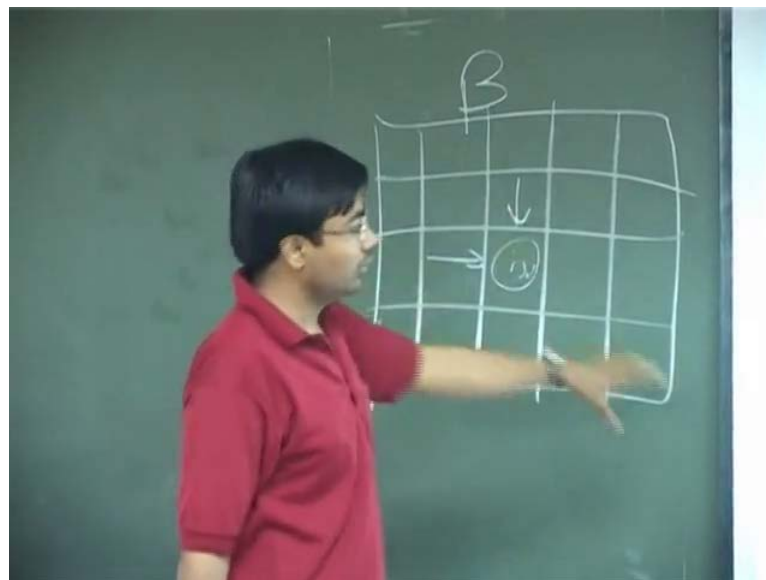


(Refer Slide Time: 22:16)



So, here what we have is a dynamic programming solution entry. We form an alignment matrix.

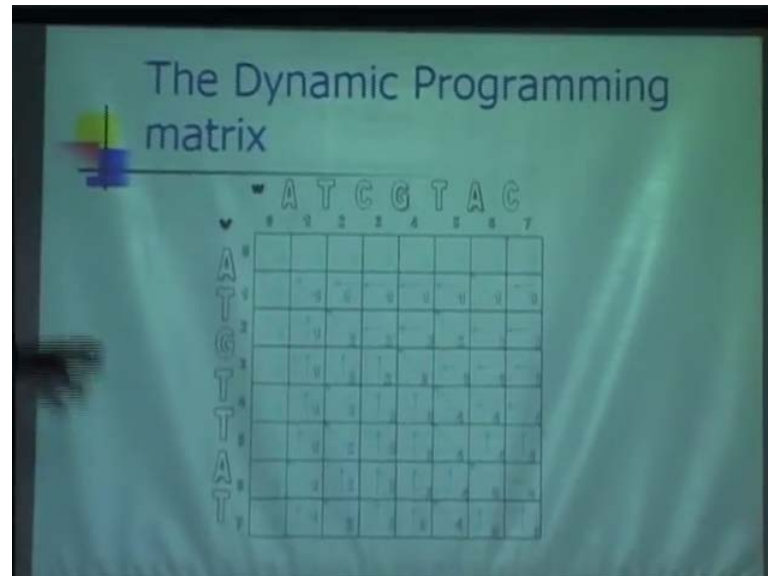
(Refer Slide Time: 22:27)



Here, is the first sequence and the second sequence any entry  $i, j$ ,  $G(i, j)$  is the optimal alignment from beginning from  $0, 0$  to up till  $i, j$  that is an optimal path from this starting starting cell to this cell. The the recurrence is defined as either the max of  $G(i-1, j-1) + \delta$  if there is a match in  $a_i$  or  $b_j$ , otherwise what we do is that

either we come from this place or we come from this place that these 2 positions correspond to insertions and deletion. For them, we impose a penalty that is sigma.

(Refer Slide Time: 23:06)



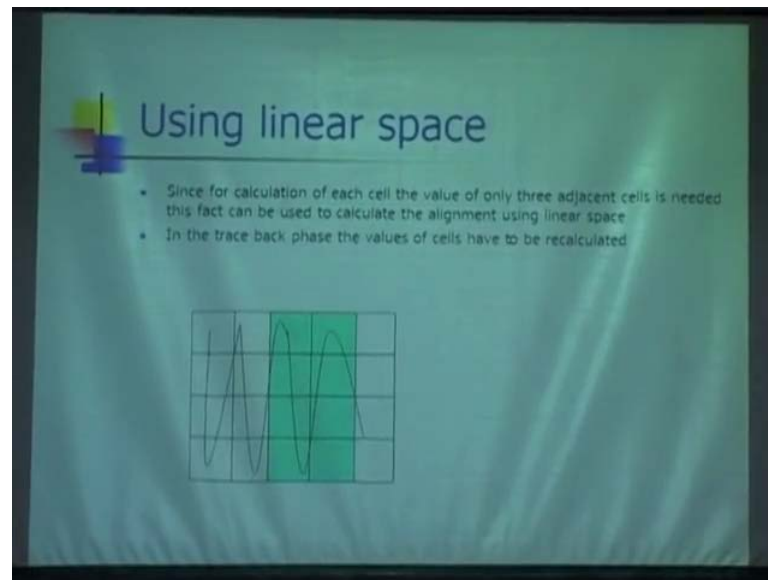
So, what we have is this sort of dynamic programming matrix. This is this is the case of longest common subsequence, which is the sub, which is the sub problem of special case; you can say of sequence language. So, we just form this matrix and fill up these values. Then, from this last cell, we have to trace back to find the optimal alignment.

(Refer Slide Time: 23:31)

- 
- The slide is titled "The computational expenses" and lists the following points:
- Time required =  $O(n^2)$
  - Space Required =  $O(n^2)$
  - Since DNA sequences are very large it is not feasible to align using  $O(n^2)$  space
  - Hence we align using linear space on the expense of increased time.

The computational expenses here are the time required is order  $n^2$  because we have to fill up this  $n^2$  matrix, where the size is  $n \times n$ . The space required is also  $n^2$ , but, the problem is that the DNA sequences are so large that it is not feasible to align using the  $n^2$  space. So, we use the linear space alignment algorithm.

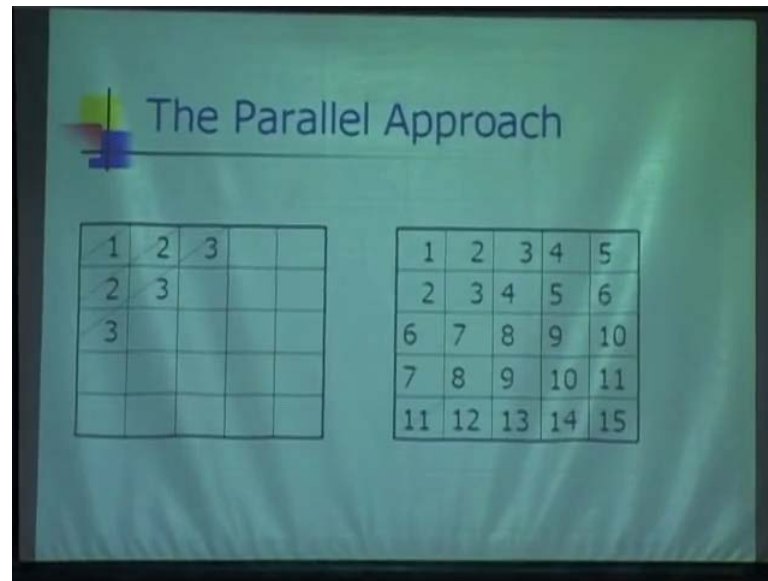
(Refer Slide Time: 23:49)



Here, what we do is that we observe that there is a dependency for calculating this cell. We need the values of only these 3 cells. We do not need any other values. So, what we do is that we start calculating our values in this fashion. That means we calculate first this column. Then, we calculate all the values of this column. For example, if we have to calculate this value, we have to just use 3 values.

Now, when we are done with these 2 columns, we we begin this column. We do not need the values of this column because all these values are dependent on these values. So, we can just free this space and calculate this column. So, progressing in this fashion, we calculate the score up till here. Then, what we have to do is that now since we have used only this is this 2 columns linear space, so to find the alignment, actual alignment, we have to again in the trace back phase, we have to again fill up these cells as we move from this column to the first column. So, this is the linear space approach.

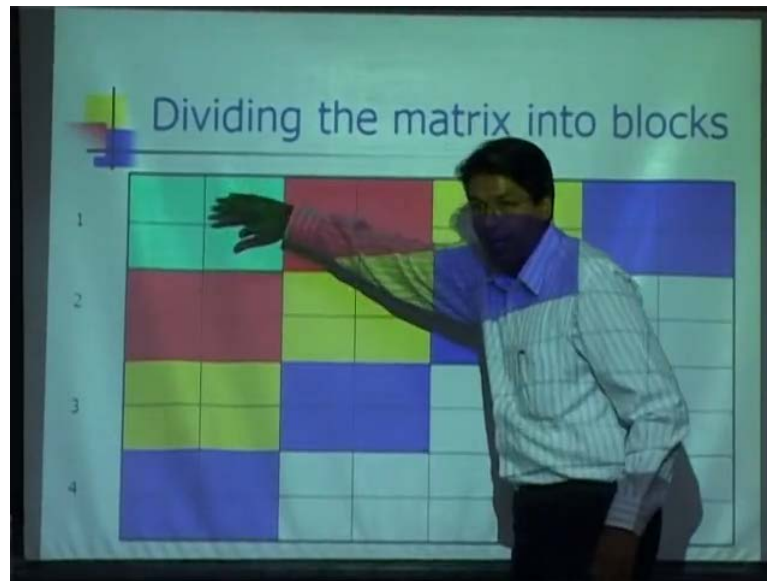
(Refer Slide Time: 24:49)



Now, the parallel approach basically, as you can see a particular value is particular value is dependent on the 3 levels. It is 3 levels. So, there can be a wave front parallelism in this particular approach. So, basically what we try?

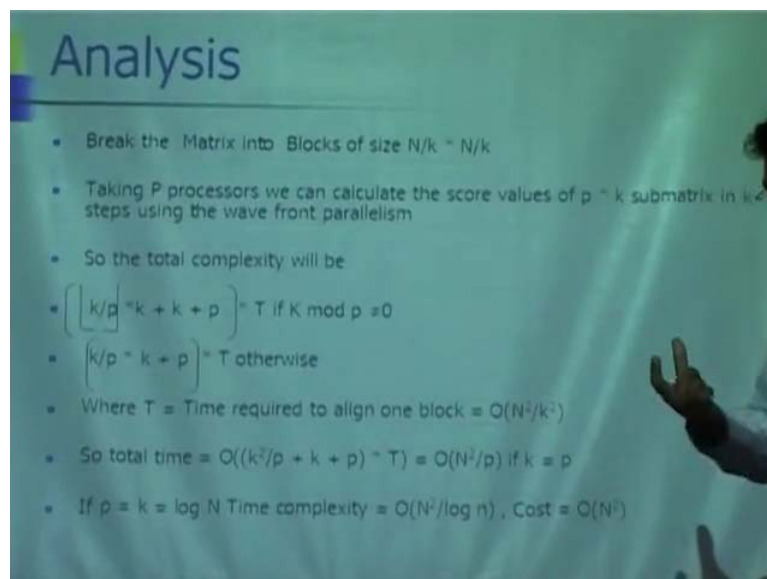
The basic approach is that suppose we have  $p$  processors, so you can assign the diagonal in this fashion like  $p$  rows diagonally,  $p$  rows. You can calculate all these values when suppose in this particular iteration, this processor is now free. So, you can assign this processor to this particular row. So, this value and this value both get calculated in this next step. Hence, if we are in the next step, so you can calculate in this particular fashion.

(Refer Slide Time: 25:33)



So, basically this is the basic funda of that particular approach. Now, what here we are doing that we are dividing the whole matrix into number of blocks. Here, a particular block contains a number of sub sequences of that particular string. Now, we are calculating a particular block in diagonal fashion, in the similar fashion we just shown, we are calculating with the help of wave point wave front parallelism.

(Refer Slide Time: 26:05)



So, here is the analysis of that particular approach. Suppose we are breaking the matrix into the blocks of  $N$  by  $k$  cross  $N$  by  $k$ . Now, suppose we have  $p$  processors. So, we can

calculate the score value of  $p$  cross  $k$  sub matrix in the  $k$  steps using the wave front parallelism. Actually, what we are doing is that here these are actually, the blocks not the individual cells.

So, if we have  $p$  processors, what we can do is that using this type of parallelism, first we calculate this sub matrix of  $p$  cross  $k$  the first in the first step the first only. What we use only 1 processor in this. Here, we are assuming there are 3 processors. In the second step, we can use 2 processors. In the third step, this is the third step, we can use 3 processors. We can go on in this fashion up till we reach this point.

Here, when we are using only 2 processors, what we can do? We can assign the third processor to this 1. So, in 1 phase, what we can assume is that we are calculating  $p$  cross  $k$  sub matrix.

So, when we are going to compute the  $p$  cross  $k$  sub matrix, so the total time complexity will be  $k$  by  $p$ . This is because we will have  $N$  by  $k$  steps,  $k$  steps for this. Each sub matrix will take  $k$  steps, so into  $k$  plus. If  $K \bmod p$  is not equal to 0, then we need some more time, and more some extra  $k$  steps.

Finally, we need  $p$  steps when here for the for the for the last particular case. Suppose that  $T$  is the time taken to calculate the values of 1 particular block. So, we simply multiply with the  $T$ . So, this is the total time complexity. Now, if we are taking, consider because here, we are using the dynamic programming; usual dynamic programming matrix.

So, aligning  $N$  cross  $k$  into  $N$  cross  $k$  takes  $N$  square by  $k$  square times, so multiplying with  $T$ , we get this. So, here is the total time complexity. If we are taking  $\log N$  processors, the time complexity is order of  $N$  square by  $\log N$ . If we are going to calculate the cost, it is order of  $N$  square. So, it is cost optimal cost.

(Refer Slide Time: 28:12)

### 4-Russian Speedup (For LCS)

- Considering the case of LCS where the scores of the consecutive cells horizontally or vertically differ only by one.

```
5 5 6 7 8 8
6 - - - - -
7 - - - - -
7 - - - - -
8 - - - - -
```

- This can be represented as a binary sequence
- Given the value of the top left corner of a block all the values along the border can be represented as a bit string. The above sequence can be represented as

```
0 1 1 1 0
1 - - - - -
1 - - - - -
0 - - - - -
1 - - - - -
```

Now, what we in this? What we see is that we cannot do anything regarding this part because there is a dependency that for calculating this block, we need the values of these 3 blocks. So, the 4 Russian approach, what it tries to do is that it tries to somehow minimize this value of  $p$  that is the calculation for 1 single block. Now, how this is done?

(Refer Slide Time: 28:33)

### 4-Russian Speedup (For LCS)

- Considering the case of LCS where the scores of the consecutive cells horizontally or vertically differ only by one.

```
5 5 6 7 8 8
6 - - - - -
7 - - - - -
7 - - - - -
8 - - - - -
```

- This can be represented as a binary sequence
- Given the value of the top left corner of a block all the values along the border can be represented as a bit string. The above sequence can be represented as

```
0 1 1 1 0
1 - - - - -
1 - - - - -
0 - - - - -
1 - - - - -
```

Actually, this is just for a special case of LCS, which is special case of sequence alignment. Now, in LCS, what we observe is that whatever penalty is you are imposing, this is the difference. This delta is 1 and this sigma is 0. So, what happens is that if we

take, if we consider this as a block, so for calculating the values of a block, what you need is that inside this, we have small cells. This is a block.

So, what you need is that these values and these values for calculating this block. Now, what happens is that since the penalty, the score awarded is 1 and the penalty is 0, so these values only differ by 1 or 0. So, what we can do is that if we are giving the corner most value, we can represent this as a bit stream 0 1 1 0.

(Refer Slide Time: 29:25)

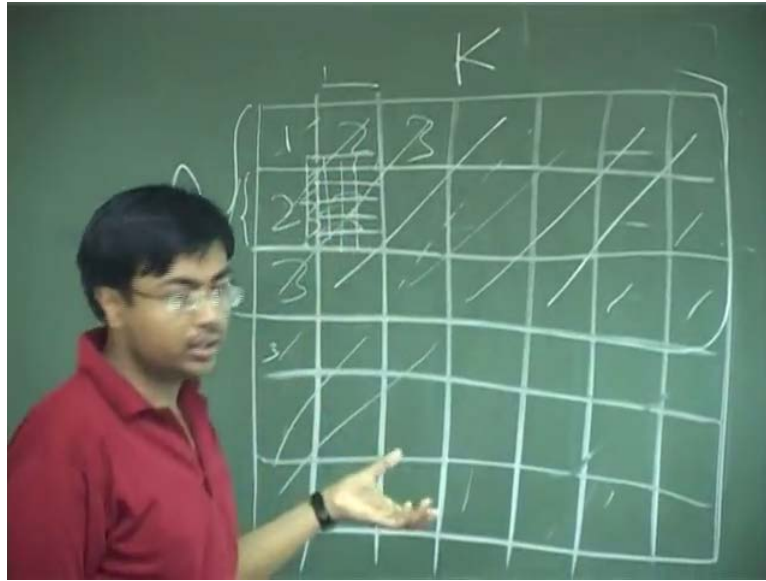
**4-Russian Speedup (Contd..)**

- For a block of dimensions  $k' \times k'$  the total possible combinations of binary values is  $2^{k'} + 2^{k'}$
- The possible combinations of the sequences is  $4^{k'} + 4^{k'}$
- So total possible combinations are
  - $2^{k'} + 2^{k'} + 4^{k'} + 4^{k'} = 2^{k'+1} + 4^{k'}$
- If  $k' = \log N/6$  then  $C_{total} = N$  (since  $k' = N/k$ ,  $k = O(N/\log N)$ )
- Now instead of computing the alignment score of a block we just look up in the table which takes  $\log N$  time. The size of table being  $2^{k'+1} = N$
- The time for Construction of this table sequentially =  $O(N^2/\log^2 N)$
- Using  $\log N$  processors the table can be constructed in  $O(N^2/\log N)$  time
- Time for Computing alignment =  $O(N^2/\log^2 N)$
- So total time =  $O(N^2/\log^2 N + N^2/\log N)$ , Cost =  $O(N^2/\log N)$

Now, for a block of dimension  $k$  dash cross  $k$  dash, the possible combinations of binary values is 2 to the power  $k$  dash plus 2 to the power  $k$  dash for all this bit streams and for all these, for whatever we are aligning this sequence and this sequence.



(Refer Slide Time: 29:39)



This is of length  $k$  dash and this is of length  $k$  dash. Since, the alphabet is 4 a c d g, so the maximum possible combinations are 4 to the power  $k$  dash into 4 to the power  $k$  dash. So, total combination possible combinations are 2 to power  $k$  dash into this gives us 2 to the power  $6k$  dash. So, for 1 block, how many possible configurations are possible is equal to 2 to the power  $6k$  dash.

So, if in the case when  $k$  dash is equal to  $\log N$  by  $6M$  then this total number becomes linear. So, what we do is that we recomputed all these possible blocks. We make a table of size  $N$  because this is linear. Now, instead of when we are actually calculating the values of this block, but now what we have to do is that we just have to look up in this table.

This is because we have precompiled this table. This look up takes only  $\log N$  time. So, instead in the previous case, it was taking  $N$  square by  $k$  square time. Now, we have to just look up into the table and that time for computing this table is  $N$  square by  $\log$  square,  $N, N$  into  $\log$  square  $N$  for sequential. We parallelize it. So, we got  $N$  cross  $\log N$  time using  $\log N$  processors. Now, the time for computing alignment becomes  $N$  square by  $\log$  square  $N$ . The total time becomes  $N$  square by  $\log$  square  $N$  plus  $N \log N$ . This is the time for precomputing the table. So, the cost is  $N$  square by  $\log N$ .

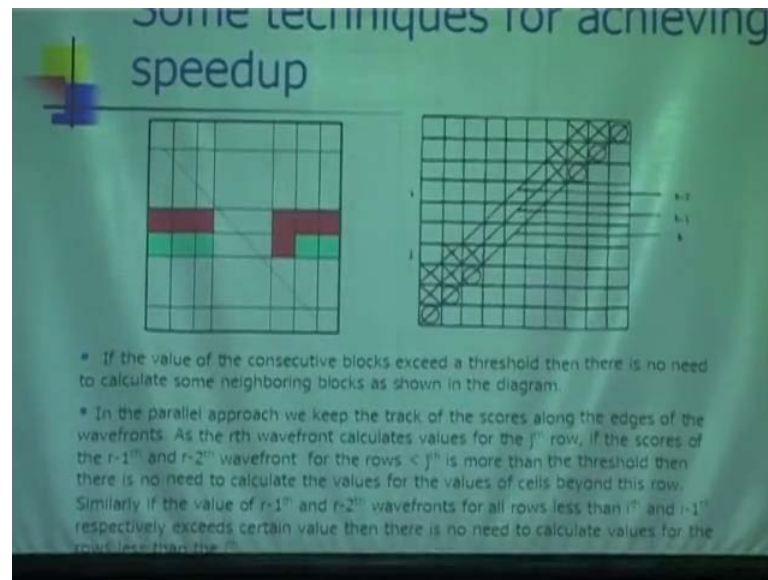
Now, my point is that just the fact that you told the square...

Sir this is the one. Both are different proximal.

Value them is for proximal we matches if the lower bound...

Sir, actually for for for the sequential alignment, till now, no lower bound has been told. So, it cannot be there. There is an example which cannot be done. So, this is the 4 Russian speed up. So, we are using the parallel approach. This is the best we got that is  $N^2$  square. If this factor is greater then so of order of  $N^2$  by  $\log^2 N$ , where in previously, we are getting  $N^2$  by  $\log N$ .

(Refer Slide Time: 31:50)



Now, due to this constraint, there are not many options for deterministically doing some new parallelism. So, what we do is that we use some techniques for achieving more speed up. Now, for that what we do is that we define a threshold threshold value that up till some some consecutive blocks. If we have exceeded, we have got our score more than actually we have to model the problem in opposite phase. This score is such that if the score is more that means, these 2 sequences are more far apart. It is just the positive and negative are changed in the penalties.

So, if this alignment up to here exceeds certain score, then what we say is that we cannot have a path passing through these blocks. This is because the theory behind is that if you reach up till this block and you have some alignment score, from this end, from this place what best you can have is that you align all these all the possible all the possible

part of sequence left. But, still some of this alignment is left. This score will give will give you a very bad value.

So, that means what we do is that we if we reach this this threshold value.

We do not have to calculate all these values below this because no path no path can come like this and go. This is because unless you have to align in this fashion. Similarly, if you see over here, if you if exceed the values in these blocks, then you do not have to calculate the values. These 2 paths can pass on like this.

So, when we are doing this parallelly, what we do is that if we are having the k th wave front, this is the k th wave front. We see, we observe the values of the edges of values of the k minus 1 and k minus 2 th wave front. So, if here the values exceed beyond, we have to find a row j, where the values of blocks exceed the threshold. So, if this exceeds the threshold, we do not have to calculate all these values. Now, our matrix reduces to this much part only.

Similarly, in this case, if this is k, then if these 2 blocks of k minus 1 and k minus 2 have more than this value, then we do not have to calculate all these values further because all these values, this area will be neglected. The path will pass on like this. So, this is 1 speed up, but the thing is that all these approaches are sort of improvisations. They do not give you aN improvement by a factor of by some order.

(Refer Slide Time: 34:28)

Using the forward and backward recurrence

| The forward recurrence  | The backward recurrence   |
|---|---|
| $g(i,j) = \max \begin{cases} g(i-1,j-1) + \delta \text{ if } a_i = b_j \\ g(i-1,j) - \sigma \\ g(i,j-1) - \sigma \end{cases} \text{ Otherwise}$ | $f(i,j) = \max \begin{cases} f(i+1,j+1) + \delta \text{ if } a_i = b_j \\ f(i+1,j) - \sigma \\ f(i,j+1) - \sigma \end{cases} \text{ Otherwise}$ |

This is a divide and conquer approach which reduces the time by a factor of  $1/2$  each time. Each time we find the value of the column  $j$  which maximizes the value  $g(n/2,j) + f(n/2,j)$  and then we divide the problem into two parts independent of each other.

Then, there is another approach of using forward and backward recurrence. Here, what we are doing is that it was  $g_{i,j}$  denoted the optimal alignment from starting from the 0, 0 cell to  $i, j$  cell. The backward recurrence calculates the value starting from the last cell and  $f_{i,j}$  means the optimal alignment from this from  $n, n$  to  $i, j$ .

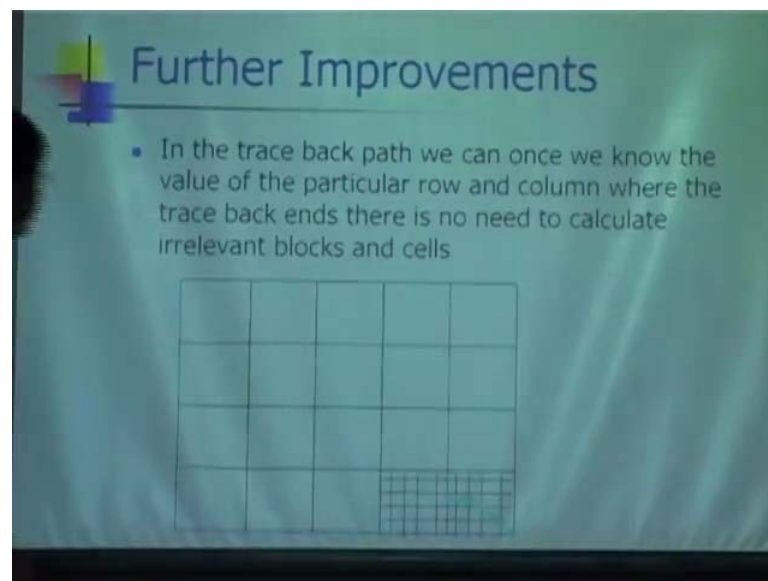
Now, what we do is that we divide a matrix into 2 parts. The first part is calculated using the forward recurrence. The second part is calculated using the backward recurrence. Both can be done using the linear space. Now, we take the middle column and calculate the sum of values of  $g_{i,j}$  and  $f_{i,j}$ . Now, the optimal path will always pass through that cell where  $f_{i,j}$  plus  $g_{i,j}$  is maximum. So, what we do is that we have found the cell in this column through which the path will pass on.

So, now, what we can do is that once we have found this, so we can divide our problem into 2 parts since this is the optimal line up to here and this is the optimal line up to here. Now, we have to we forget these blocks because the alignment always pass through this. Actually, we are doing it using linear space. So, we only have 4 up till here.

Now, we have to again trace back and again calculate these values. So, in each step, what we do is that we speed up by a factor of half.

But, this also does not give us an order improvement order. Further, there are some approaches.

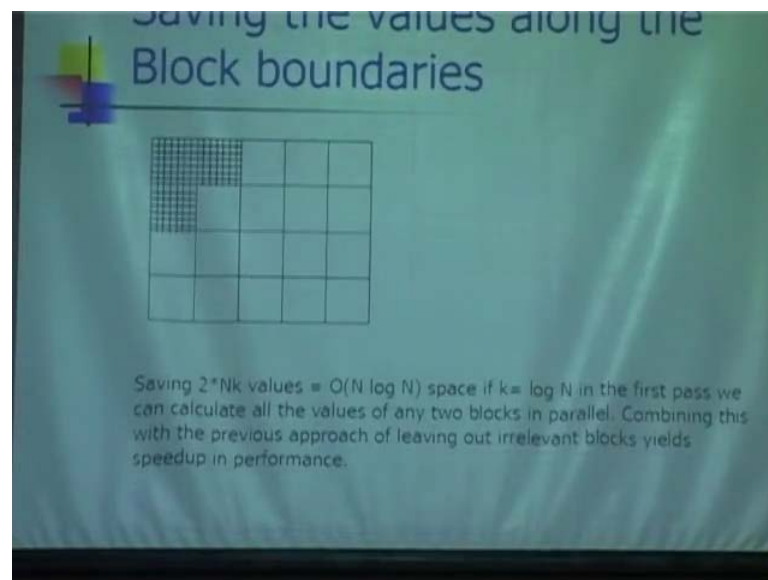
(Refer Slide Time: 36:09)



For example, you can see over here that the path in the trace back phase. We see that the path is passing through these values. Now, if we know that the path has reached up to this cell, obviously, we do not have to again calculate in parallel these blocks. This is because the path will never pass in these blocks.

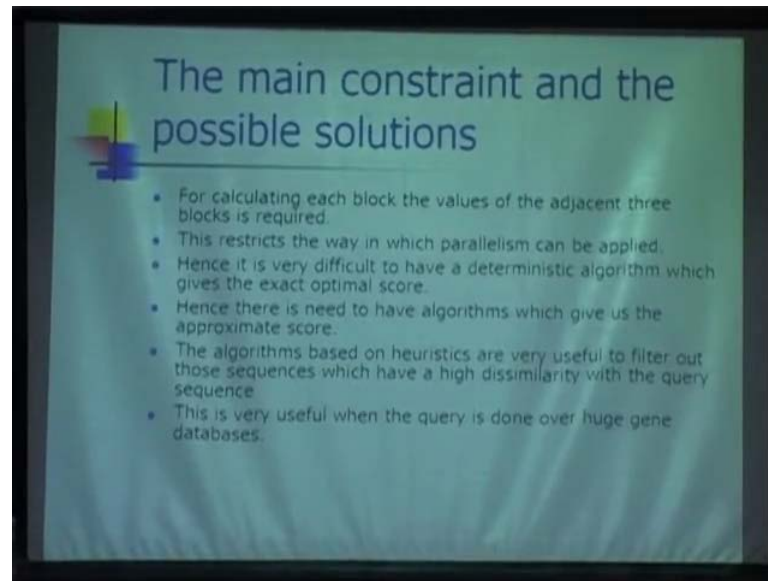
So, what we are doing? We do not have to calculate this path, this block. We can instead calculate in some other blocks by storing the values of this. I will show in next ppt. We also do not have to calculate the values for these cells because the path will never pass through this portion.

(Refer Slide Time: 36:39)



So, what we do is that we see. We save these values along the borders this and this of each block. So, these are  $2 N k$  values. If  $k$  is  $\log N$ , we are using  $N \log N$  space. Now, what we do is that in this case, instead of calculating value of this block, since we have all the border values, we can calculate whatever block we require for our optimal block.

(Refer Slide Time: 37:06)

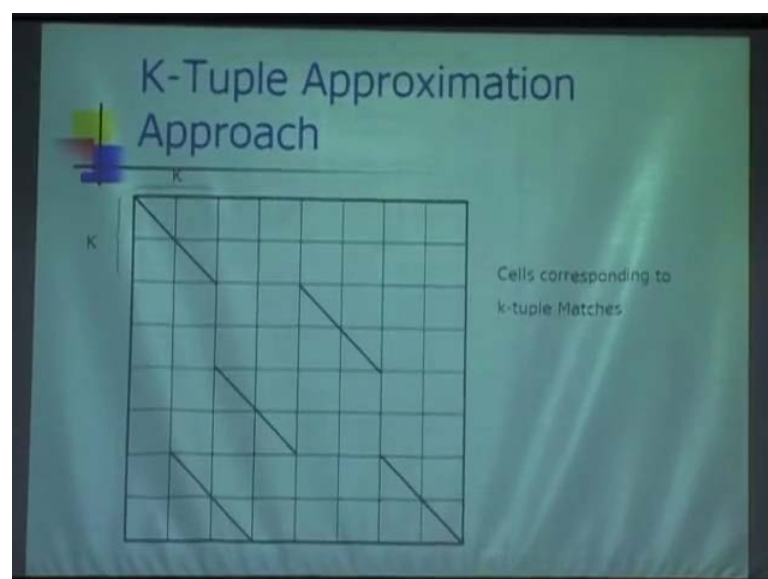


### The main constraint and the possible solutions

- For calculating each block the values of the adjacent three blocks is required.
- This restricts the way in which parallelism can be applied.
- Hence it is very difficult to have a deterministic algorithm which gives the exact optimal score.
- Hence there is need to have algorithms which give us the approximate score.
- The algorithms based on heuristics are very useful to filter out those sequences which have a high dissimilarity with the query sequence
- This is very useful when the query is done over huge gene databases.

So, now the main constraint what we observe was that for calculating each block, the values of adjacent 3 blocks is required. So, this is the main constraint in going through in calculating the value using a deterministic algorithm, which gives you a correct exact optimal score. So, the solution is that we have to have algorithms, which gives us the approximate score. These algorithms are based on heuristics. These algorithms what they do is that they when we are searching for a gene and huge database, we have to filter of those genes, which are highly dissimilar, when the queries query is a gene. So, these algorithms are used here.

(Refer Slide Time: 37:40)



### K-Tuple Approximation Approach

K

K

Cells corresponding to k-tuple Matches

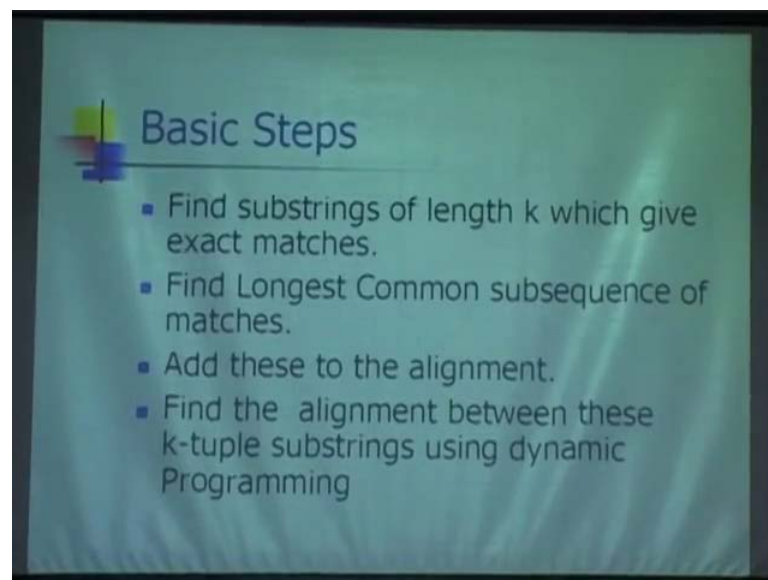
The diagram shows a 6x6 grid with diagonal lines representing k-tuple matches. The top-left cell is labeled 'K' and the top-right cell is labeled 'K'. The text 'Cells corresponding to k-tuple Matches' is located to the right of the grid.

First is the basic approach, which we consider. Then, we base on randomize algorithm on this approach.

What we do is that we take  $k$  size substrings, which gives exact, match in 2 sequences. For example,  $k$  exact match  $k$ ,  $k$  size exact match can occur here, here, here, all these place.

Now, what we say is that the optimal path will pass through some of this  $k$  exact matches.

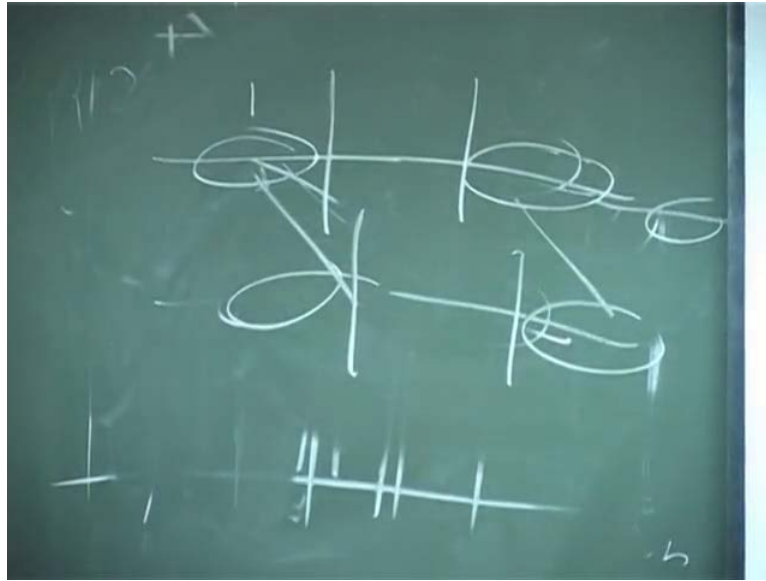
(Refer Slide Time: 38:12)



So, here what we do is that the steps are here. First, we have to find substrings of length  $k$ , which gives us exact matches. Then, we have to find the longest common subsequence of matches, if these are 2 sequences in the same  $k$  substring, it can be over here, here, here and here.



(Refer Slide Time: 38:23)



So, we can have something like cross matches. So, we do not, we have to avoid this cross matches. What we do is that save the positional information of this once. For example, if we name like number 1 and this is also corresponding 1 to 1 matches 2, 2. Now, we have to form longest common subsequence of matches. So, we will. This is done to just to avoid these cross matches. Now, once this is done, what we do is that we have something like this.

This is the whole sequence. In some parts, we have k exact matches like this. Now, these are exact matches and between these, we have gaps. We use the standard dynamic programming algorithm to calculate these gaps. Now, we will present the randomized approach, which is working about base.

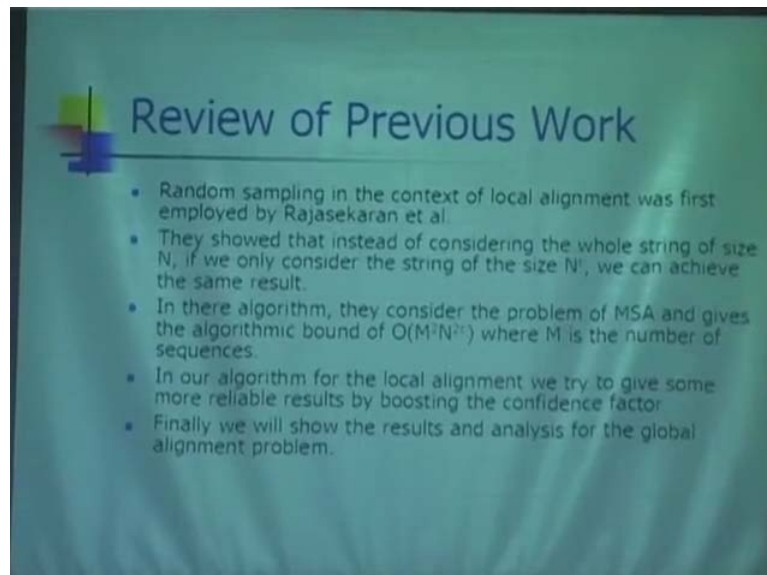


(Refer Slide Time: 39:12)



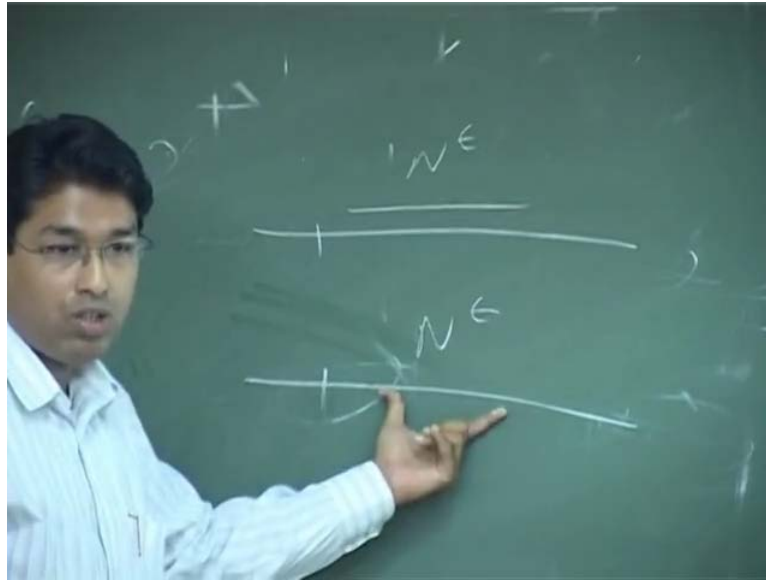
Basically, we see lots of algorithms, lots of improvements and through which, we can improve the running time of particular algorithm, but that improvement is not of some particular order. So, what in randomized approach, what we are trying to do? We are trying to do, trying to find some approximate this 1, but with some lower time bounds.

(Refer Slide Time: 39:35)



Basically the idea for this approach, we got from the paper presented by Rajasekaran in his paper what he showed.

(Refer Slide Time: 39:59)



What he did? He took a random number particular and he took a string of length  $N$  to the power epsilon. Once again, he took the same position in the second particular string and need to get  $N$  to the power epsilon length string from this substring from this string also. He tried to make he tried to align these 2 particular strings.