Parallel Computing
Prof. Subodh Kumar
Department of Mechanical Engineering
Indian Institute of Technology – Delhi

Module No # 06
Lecture No # 32
Algorithms, Merging & Sorting

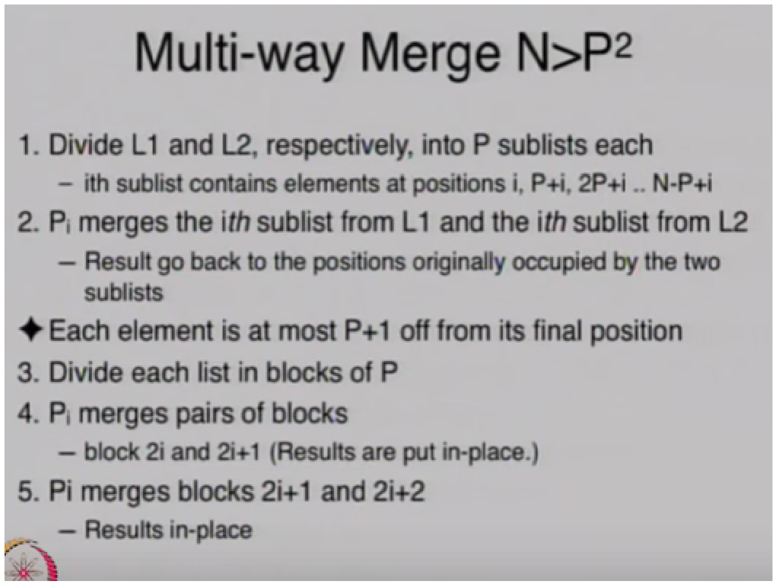
Okay let us get back to that analysis that we were just discussing. Now if you think back to more generic case where number is greater than T square, you can also look at the case when number is less than T square. But maybe not as interesting as when the number is greater than T square. How to change the algorithm? If you have to take this, see that it has the ISS strength is that it does lots of things in a sequential way, which is if you look back to the earlier slides.

We were talking about the design. You try to keep things as sequential as possible try to remove communication remove synchronization. Now instead of N T square elements, we have many times T square elements. You could do simulation like before. Professor student conversation starts - in a two scenarios initially it may start with N of B number of elements which will be greater than P then or we can start with P number of elements.

But more than N by P number of chunks and then the merging elements will be more. But you would ideally like if you want to keep thing local then which of those two would you choose left one right. You want to divide into P work everybody does sequentially, that is what we are going to do Professor Student conversation ends. You are going to say divide L one again every P PF elements goes to a given processor.

So 0 P- 1, 2P - 1 extra we are going to processor 0, but now processor is getting much more than P, fine it still is going to get N over P it is going to N over P from here, N over P from here it is going to merge it sequentially and then we are going to do something about the rest the

(Refer Slide Time: 03:06)



Multi-way Merge N>P^2

1. Divide L1 and L2, respectively, into P sublists each
   - ith sublist contains elements at positions i, P+i, 2P+i .. N-P+i
2. P_i merges the *ith* sublist from L1 and the *ith* sublist from L2
   - Result go back to the positions originally occupied by the two sublists

✦ Each element is at most P+1 off from its final position

3. Divide each list in blocks of P
4. P_i merges pairs of blocks
   - block 2i and 2i+1 (Results are put in-place.)
5. Pi merges blocks 2i+1 and 2i+2
   - Results in-place

The element gets to again and this is again P + 1 is in pair within P positions and this requires a little bit of proving which I am not going to do. But essentially the same logic you start with that proof and you see that this is still true. Now if we have said that after this process there are chunks of P that we have found because every element is between P of each other within P of its final rank position and each of these P is sorted.

You still have to do the same thing as before. You take blocks of P, merge it blocks of P, which is excellent because you keep blocks of P that you are taking, can be done in one place where somebody else is doing another P and merging it. Now there are many more P block store that is, What is different, that the difference are in point. Number 2 or step number two in that, although it is written exactly the same the text is exactly the same.

But now you are getting the size of each list, sub list to become bigger here. The two differences are here, so once you get done with that pair 0 and one you go and do P square. P square with elements there are P blocks or rather P pairs that you did like 0 and 1, 2 and 3. P - 2, P -2 and 2 P - 1 and then somebody is going to do the next 2. So everybody was busy doing these P pairs then they are going to do the next P pairs.

Still remain sequential, it is just that if that thing is happening in a loop merge pairs of P things in a loop and again for the next, we do the same thing. How does the time change? The changes are at those three places. Professor student conversation starts I am saying that our N power P has increased size of supplies size of increased and then we are saying that first P pairs has sorted and then counter shift to P square element.

So we are moving like this so ultimately we have first P processor has added up on the first P things. So it is related to the first P something like we are looking now P see exactly some remain the same each time exactly. So this size that was P ultimately and it was doing sequential work your saying, I am asking for one hydration for sub list size will be the same P. Sir after hydration are over then might whole sub list size make P.

Because I think it useful to look what is happening after first step. Suppose you have many more processor, does not matter how you merge that right. The point is if you have blocks of P then if you do every P, F you signed them to one process, sign them to hundred processor. You send them to whatever number of processor after they merge this, every PF of their frame of their sub list then elements are not to be within P of each other.

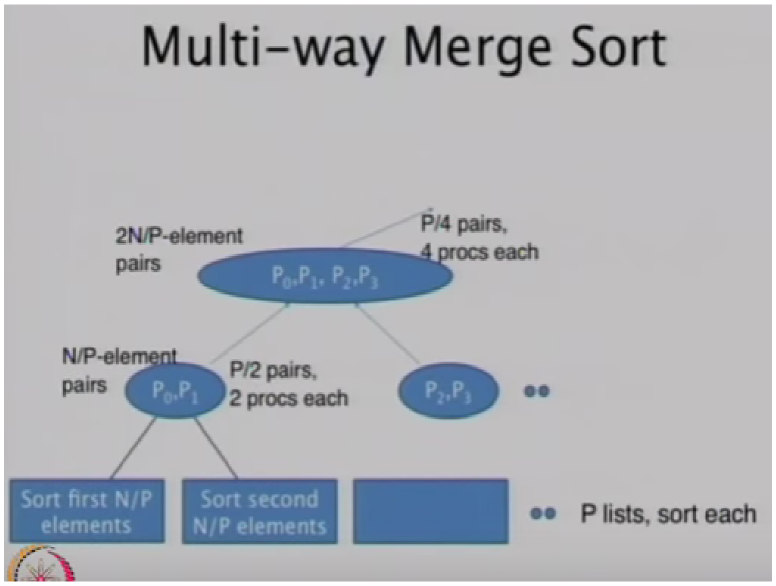
Basically the same argument as before which means, now if you form blocks of P and take another pair with another P elements next to it and had to merge it. After if you do it with right and the left once then the entire thing will be solved .Now if you look at, so that is what needs to be done. Now who does what accounting needs is to be done to figure out the time. So you are doing one pair at one processor at a time. How many pairs are there total pairs?

Because there is only P in each block, N by P pairs. So P pair are going to do the first of the P pairs are being done by P processor first, then another P pairs and until all are covered. So how long does this take N by P besides, because it is sequential as before. You do among the work done is proportion to the number of elements, you handling. So N by P into number of hydration that is exactly what I was just explaining.

Because each case not N by P each p. So every processor, another way to think to argue is every processor is handling one of them. One pair and every element is being handled by one processor and you do only amount of work that is supported with the elements given to you. So of course total work is done by proportional to the number of elements is handled by only one. Professor student conversation ends.

So the time remains N by P it turns out that it is little hard to keep the time saying for this algorithm if you make the number of processor bigger then square root of N. So this algorithm is inherently less parallel which should not come as a surprise, it is trying to keep things sequential. Then how will you use this for sorting?

(**Refer Slide Time: 10:05)**



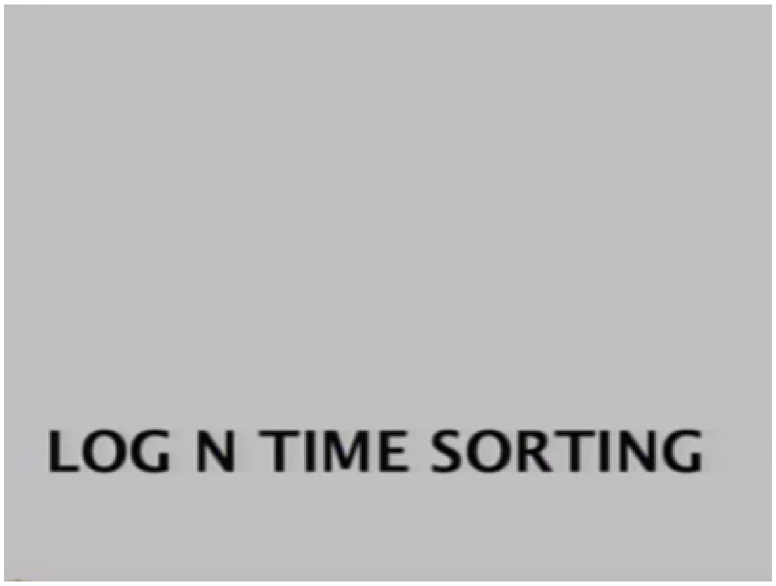
Multi-way Merge Sort

And how long will it take again? Let us keep the same framework where we are talking about P processes and demand of time taken with P processor. If the modes keep so fast for time being again, we take the same argument that there are E square elements to sort. Actually this case we do not need to worry about it. Professor student conversation starts N P Log N it is N / P Log N, but how of the stage sir P stages and NP stages takes N development?

Why does this take N / P Log N, because if the first stage, if I have N / P time Y the first how many (()) (10: 56) the two processes and 2 N / P chunks. So this actually the list size doubles and number of processes handling that list size also doubles. So it is going to take the same amount of time each one is going to take N of P which is demonstrated here, Professor, student conversation ends. And the total time will be N / P log.

But of course original initial sorting was also needed. The local sorting because we can merge only after the individual list are sorted and so that will take N / P Log N / P which is Log N. Now we are going to come back to the mode of work time scheduling and see what it takes again. The algorithm itself is not my main focus it is not something you would necessarily limit it is less than simple. However the technique that it uses is an important technique it uses the pipelining in a very efficient way and so just you learn that technique I am going to talk about.

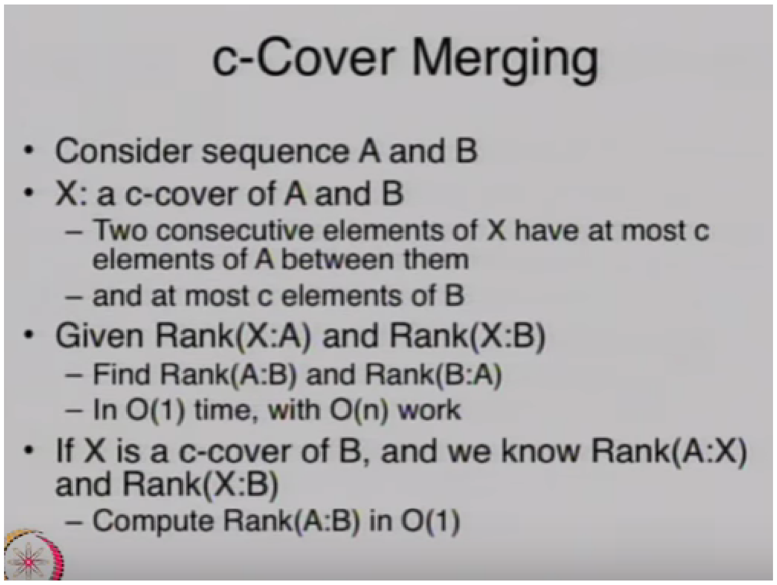
(Refer Slide Time: 12:34)



LOG N TIME SORTING

So this is Log N time sorting and it is based on the idea of C cover merging which means that

(Refer Slide Time: 12: 45)



c-Cover Merging

- Consider sequence A and B
- X: a c-cover of A and B
  - Two consecutive elements of X have at most c elements of A between them
  - and at most c elements of B
- Given Rank(X:A) and Rank(X:B)
  - Find Rank(A:B) and Rank(B:A)
  - In O(1) time, with O(n) work
- If X is a c-cover of B, and we know Rank(A:X) and Rank(X:B)
  - Compute Rank(A:B) in O(1)

I am going to merge two lists given some prior information is given in the form of a color, a third list. This third list is called as C cover. If the third list, the very idea of cover is that this list is the cover itself has fewer elements than the original list and the list, the cover distributed evenly to the original list. For example I can say if I take every fourth element of the list of a given list that forms a four cover.
Professor student conversation starts four which is sorted version, cover is for a sorted list Professor Student conversation ends. We should end that so you can define C cover. Next is C cover of A between any two elements of X. There are most C elements similarly it is C cover of P between any two elements of X. There is C elements so C cover is essentially giving you some (()) (14:15) of doing of your original and it tells you something about your original.

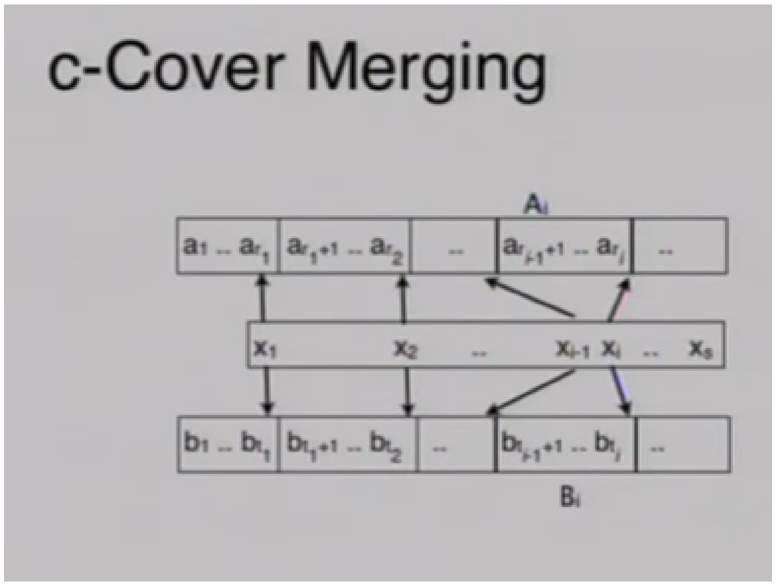
For example if I gave you the its given elements location in the C cover then you can find it in the original list by looking at most C cover. Because if I know there is between these two elements in the cover list and between these two there are only four boards then this element has to be between among those four boards. So the idea of cover is simple enough to finish off it is clear yes it is important to the rest of the other.

Professor student conversation starts sir, what are the scenarios that such a cover would readily be there? We will build this cover not assumed like Log N, you first exactly built the cover then we will use the cover. This is not under resumption of somebody giving you some. If you had C cover let us say, X because C cover for both A and B which is what we are starting to merge the input is listed and the list B and I happen to get a cover X which is C cover for both A and B, which means that for example every element of X, no I get the C cover, not just C cover.

I also get some further piece of information which is rank of the elements of the cover in the original list. So I know where to find the third element of my cover in A and in B and the fourth element and the fifth element given that looks like you given a lot of information, you can find rank of A and B. Similarly rank of B in A in instant. How do you think a cover one, two, three four and cover is

Professor student conversation starts one, two, three, four is a Professor student conversation ends. If you are talking about you have to choose A C if I say two cover then it can be one three, five, ten it does not have to be elements of A it just needs to have the property that we are doing any two elements of the cover. It is fixed there is only two more elements if it is a true cover.

(Refer Slide Time: 17: 06)



c-Cover Merging

We have the cover X, the middle vector and elements A. The vector A and B we are trying to find the rank of A and B and the rank of B and A. What we have given is the rank of X in A and B which means those lines that you are seeing going across are known. So for example if I say X I and X are X I minus one and X I between them there are C more X in A as well as B and I know the rank of X I.

So X I lies between A R I and it element to its height the greatest element less than equal to X I is A R I similarly on the B it is B T I. Professor student conversation starts one and R two, there was gap on the element between the element two. So here it this is a C cover does not necessarily mean that every two pair will have exactly four. So it is some at most four Professor student conversation ends. So these are just some indexes variables R one can be it has to be less than C.

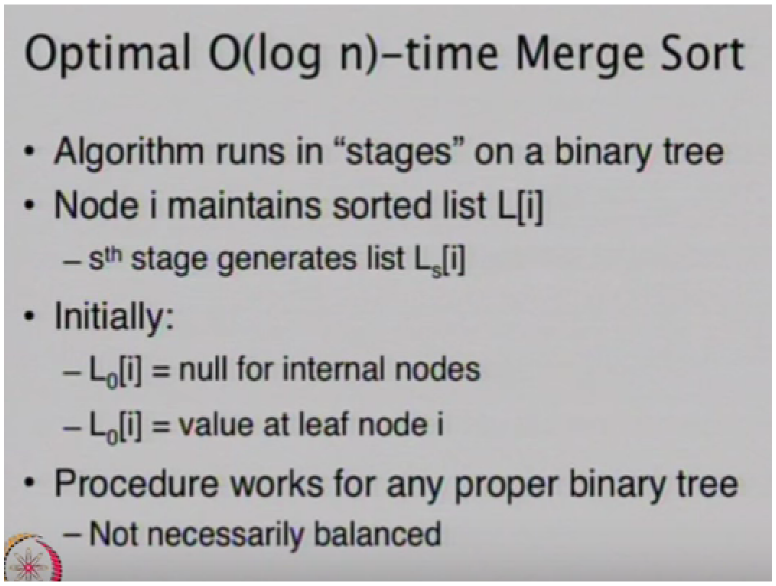
So now if I give you an element in X in a small, I have space here. I can make this diagram bigger than this and we want to find it is rank B. How can you find it order one, Professor students conversation starts how does this step how X I - 1 can give you where its lesser count which but A. So X I - 1, we know its position it was in A and B and it is shown by that two arrows which means anything to the left of the arrowheads are less than X I - 1.

X I - 1is the location the arrow is where it is found. So this one is all the elements less ten will need constant where C is constant Professor Student conversation ends. So is definitely all the elements somebody give me formal argument. So why all the elements before that arrowhead coming from A are less than a Professor Student conversation start a is greater than x I minus one and all those things are less than X I Professor Student conversation ends.

Similarly it cannot be outside of this block why similar reason exactly, so it has to be somewhere in this block and you know that there are only C elements in the block. So in C most steps we can find the rank of it. So officially the rank of A and B is rank of X I minus one in B plus rank of A and B I and if we had N processor for every element of A.

Then in order N work you would have find all ranks one at one time because they are all independent of each other at one time or N work. We can find the entire rank alright. Now so that just preliminaries. Now we understand that if somebody give us an cover with the ranks are done not much more, but where are these covers coming from.

(Refer Slide Time: 21:15)



Optimal O(log n)–time Merge Sort

- Algorithm runs in "stages" on a binary tree
- Node i maintains sorted list L[i]
  - s^{th} stage generates list $L_s[i]$
- Initially:
  - $L_0[i]$ = null for internal nodes
  - $L_0[i]$ = value at leaf node i
- Procedure works for any proper binary tree
  - Not necessarily balanced

Let us look at a few things just out of the loop it may seem and in the back of the mind there is this mystery right, when you do merge sort what you do you have one elements at the leaf. You merge pairs needed sizes of two and your merge pairs again get sizes of four keep merging until you get size of N. If you say N then at the end to it whatever we will keep the tree in mind. Because this is going to speed up the same tree, it is going to run in stages.

The stage is not necessarily the level of the tree in fact it is not the level of the tree. But it is related to the level of the tree every element of this note, no every node of this tree maintains a list. It merges children it generates the merge list and the root merges it two children's sub list gets the final list. So then the same principal still is going to apply except you will merge it many times it is not just that you merge.

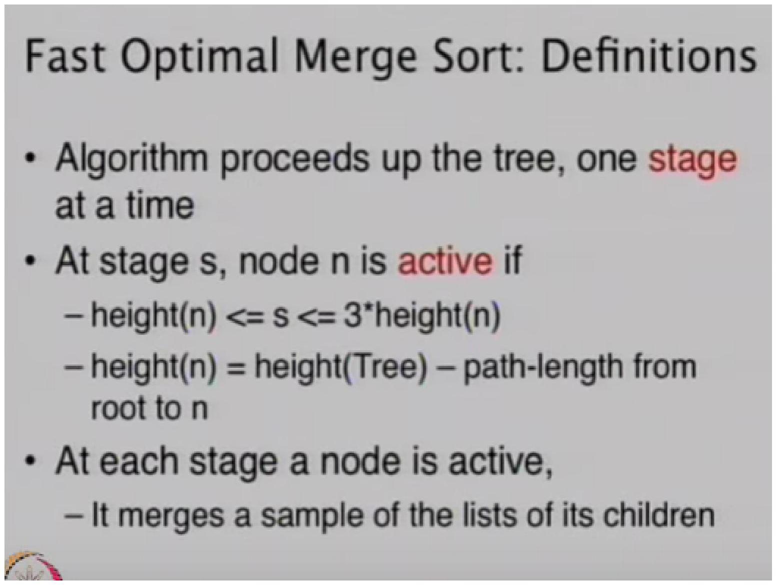
When you are chance comes when all of your children have or both of your children have established there list and there list then you merge them because that is where the time was being spent Log N levels were needed each merge. We say that we can do in Log N time, this level had to wait for that and if we cannot do merges in order one, then this entire process can never be done in order Log N.

So we want to do this entire process in Log N without the need for having mergers be done in order one and below bound is on mergers is Log N. So you could not only do it in order. So continuing with definitions, we will say that there are this stages and the list for node I has a subscript which says its value and given at a given stage. So when I say L zero of I is at the beginning at stage 0 can be thought as a steps, just steps is more generic.

We will talk about specific stages so using a kind of A non-regular term. So that whenever I say stage it is very clear about what I mean so L 0 of I for every node, that is a leaf node is one of the elements just like in the regular merge stage and for everybody else 0 is nothing null list. But still going to go through the levels which mean there will be Log N involved. But we are not going to wait for entire merge of the children to happen to generate the merge list on a given node.

That is how after what time you can start process of the next Stage even without having full result the previous stage as the sign node. This procedure works for not just a complete binary any proper tree works it is in other context where you have proper binary tree, your algorithm same process you can still get.

(Refer Slide Time: 25: 32)



Fast Optimal Merge Sort: Definitions

- Algorithm proceeds up the tree, one stage at a time
- At stage s, node n is active if
  - height(n) <= s <= 3*height(n)
  - height(n) = height(Tree) – path-length from root to n
- At each stage a node is active,
  - It merges a sample of the lists of its children

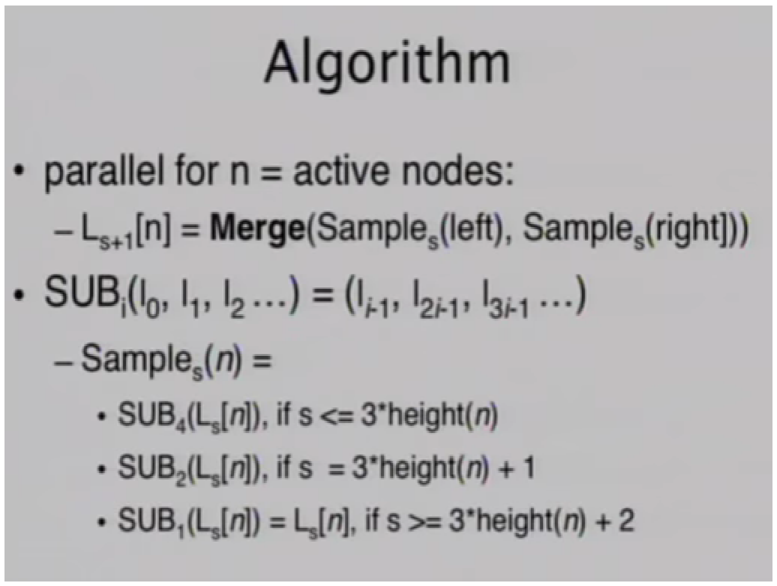
Okay some more definitions; these are less of definitions now but some properties. So the levels of the tree are activated by stage. So at zero stage the leaf node becomes active first stage then next level. So next stage, next level so after Log N stages the root becomes active just like it happens in the merge stage except they remain active, they continue to work and a node at height of used variable heights on is saying that if the nodes height from the leaf level which leaf height be 0 is N.

As I said if height is then at after H height stages that node is going to become active. It is going to remain active up to three height stage. So it is not that all the nodes remain active all the time after becoming active which means the root is going to become inactive after three Log N stages which means that if we are able to manage each stage root of the merge tree.

Professor student conversation starts it will be active only once, no that is exactly what this slide is saying that you remain active for stage if you become active at stage equal to your height there is a global counter for that stage and you become active at stage equal to your height and you remain active until your stage is equal to three times your height. But number of stages total number of stages is Log N more than N in fact it is three Log N.

Because after root becomes inactive there is nothing much to do Professor Students conversation ends. And the basic idea is that at every stage instead of taking your left and the right child and merging their list you simply take some of your left Childs list, some of your right child list and merge only that. So you sample your left and the right child just too many elements to handle. Professor student conversation starts height is depth height, depth from the leaf height of the node. So height is always on the down, Professor Student conversation ends.

(Refer Slide Time: 28: 37)



Algorithm

- parallel for n = active nodes:
  - $L_{s+1}[n]$ = **Merge**(Sample_s(left), Sample_s(right]))
- $\text{SUB}_i(l_0, l_1, l_2 \ldots) = (l_{i-1}, l_{2i-1}, l_{3i-1} \ldots)$
  - $\text{Sample}_s(n)$ =
    - $\text{SUB}_4(L_s[n])$, if s <= 3*height(n)
    - $\text{SUB}_2(L_s[n])$, if s = 3*height(n) + 1
    - $\text{SUB}_1(L_s[n]) = L_s[n]$, if s >= 3*height($n$) + 2

And the definition of sample is also out of the loop. If yours active three times your height, until that time you are sample is every fourth element. After that the next step your sample will be second element, now your sample means your parent is taking your sample if you have become inactive at that point your pay. But your parents will remain for how many more steps, three more steps it is N + 1 H +1.

So 3H+ 3 so for those three steps it has to figure out samples and so after you have become inactive your sample becomes every other element and this next time your sample becomes every element. Now you have become full list that you expect from your children then you and the entire list is being taken by the parents. Parent is going to generate a full list at three N + first stage 3 N + 3 height, three your height plus three that is why the parent will have generated all of the list with that belongs to that subject.

We have to stop here but you can probably see now the under applying logic of this merge sorting. Professor student conversation starts, it is unsorted data original data on the leaf is unsorted Professor student conversation ends. So what is happening here you Professor student conversation starts you said that some of the limit and the sum of the elements is going to become the cover that will be critical one that would be critical.

We have a lot to talk about right now we are going to Professor student conversation ends. But basically the intuition should now start to develop in that in one step you should be able to take the sample and merge the sample using the sample. You took the last sample, you took the last time was some indication of what your samples are, now the sample. You just last time, you are going to cover for the next set of samples.

So using that cover you are going able to generate the new merge list which is not the full merge list, just the merge list of the sample in constant time. But you will be active between certain number of stages, each time more samples come from your children. You keep merging it building your merge list to bigger and bigger. Basically from the previous cover and you after when become inactive at the last step.

Before that you have generated the sorted list of all the elements in your sub list and this become every stage out of one and there are three Log N stages will be done in algorithm. Professor student conversation starts background is not also comes from Log N side very different is this you will see. Let us discuss just needed for example every logarithm, now if you for example but that is what you are doing here.

You are kind of merging you will be ultimate role is to merge left and the right one. So when you already have full list here, you did not have the full list you are building the full list as your children are doing related to the number of times, number of samples. What I am saying, height is active part by then all the element from not just for three stages after children become inactive like it is height to three times height.

So there is the root is active from Log N to three Log N root is active for very long. Professor student conversation ends. So we are at the same page within turn of quizzes. We have got six quizzes. So far in addition to quiz that was the minor. So the minor that was the quiz.

Okay so let me just quickly go through some reminder of some. Professor student conversation starts, Sir which was more powerful? They are equally powerful they are equally powerful in terms of computable whatever you can compute in one, you can compute in another. Because basically we were talking about generating a Log N time question. Yes first stage is you sort the odd in the states, the next then you start even first, second position well the lowest two elements are the two positions.

How do you end up because both of them are sorted are between an odd and even. No I will L one and L two are sorted you are only merging things. We are not sorting even or odd anything left is sorted, right is sorted. We merge the odds, we merge the evens. But we know that in the leftmost will be in the first here and the first here, if we merge the two odds then where is the minimum come to?

But you can take already, no it is merging. It is a step of merge sort for example, it is not a arbitrary. If we take a arbitrary sequence and simply compare the odd ones and sort them in proper place, guaranteed nothing at the end of it is a merge, which means merge of something if you take two arbitrary sequences you can merge it in order one time.

Assuming that suppose you create a arbitrary sequence if you are supposed to got it then you cannot do anything better than log n time because sorting takes Log N time Professor student conversation ends. So back to this specific time from pipeline merging, where you still looking at the same merge tree where at the leaves are on the element and you take pairs merge them pairs of pair, merge them until you reach the root and it is time you merge all of them.

You do this in set of stages like, so these are S stages or no Log of the number of nodes of stages. But a node activates in the same way that you are original algorithm activated. The original algorithm, the first stage leaf is active. The next stage, the next level is active and after Log N stage the root is active and then nobody is active here. They do activate themselves in the same order but the deactivation is happening immediately after a stage where it happens three times at height. Professor student conversation starts

Sir in the same processes leads to the less higher stage and the same process do the I plus one stage, show the process where it will reach arbitrary. So why do we, but we needed to failed that is why we are not talking about sync process. Because this is P the model, we are considering so it is already in sync. So what is the problem in the regular how long did it take. But the processor time, they were not taking any did not work.

So why see that that the thing is that they were not quite busy, not busy enough that the without our various flavors' of merging bottom line is that at the bottom level you have N processes doing order N things. But then you have to do smaller number of merging like these sizes have become the node size has become bigger, but smaller number of mergers. That is a multiple process are involved in the smaller number of and at the top level every processor is involved in the single merge.

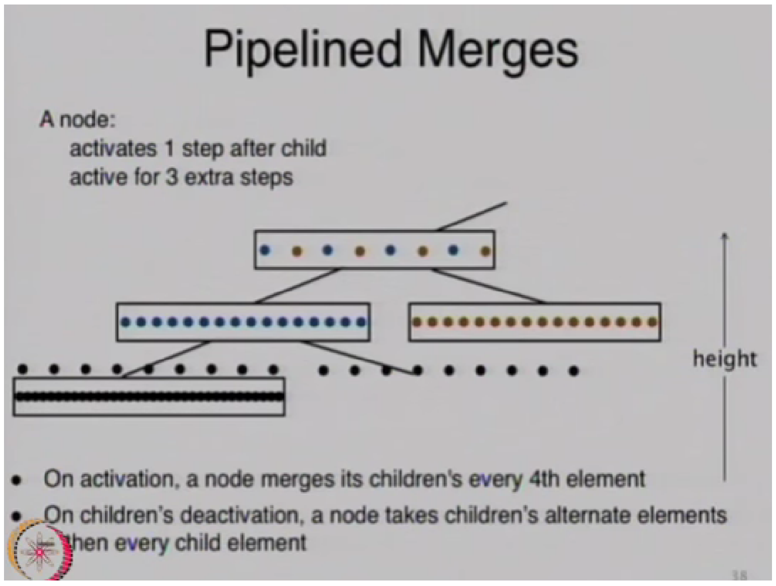
How you do that efficiently was not quite totally clear all though people look at couple of examples all of them end up taking optimal work. But the time span was too long here have to perform sort thing such that it can done in Log N it is more of a negative exercise. But it is more as I exile to understand how this algorithm proceeds. So you can apply it another not particularly learn how to do Log N sorting.

Professor student conversation ends. Now so that was one important aspect of this that you remain active keep doing some more work for little wide longer and the second was that you do not get as in the original algorithm it take your entire results produced by the children and merge this entire list to produce your output. Then you are done here, you do that in phases instead of saying, let me take all my children let us take just the sampling of my children and merge them and merge the remaining sample.

But not all the remaining sample A keep adding to the sample you have merged little bit at a time such that the previous merge helps you doing the next merge very quickly and we defined the sampling business where sample of given node in the merge tree at stage S is given by either every fourth element or every second element or every element it is every fourth element most of the time until you are at the full height write the full activity node remain node N remains active from height of N to height of N times three.

So until that period activity of N times three every force element of the node is going to get promoted up for merging one step later every second element gets promoted up two Steps later every elements get promoted at that point. If all of your elements, all of your sub tree, all the children, all the leaves in your sub tree have reached you then you are promoted on. So one step later all the leaves have reached your parent and if you are siblings on the siblings leaves have also reached the parent then the parents leaves have reached the parent.

(Refer Slide Time: 42:13)



Pipelined Merges

A node:
- activates 1 step after child
- active for 3 extra steps

- On activation, a node merges its children's every 4th element
- On children's deactivation, a node takes children's alternate elements then every child element

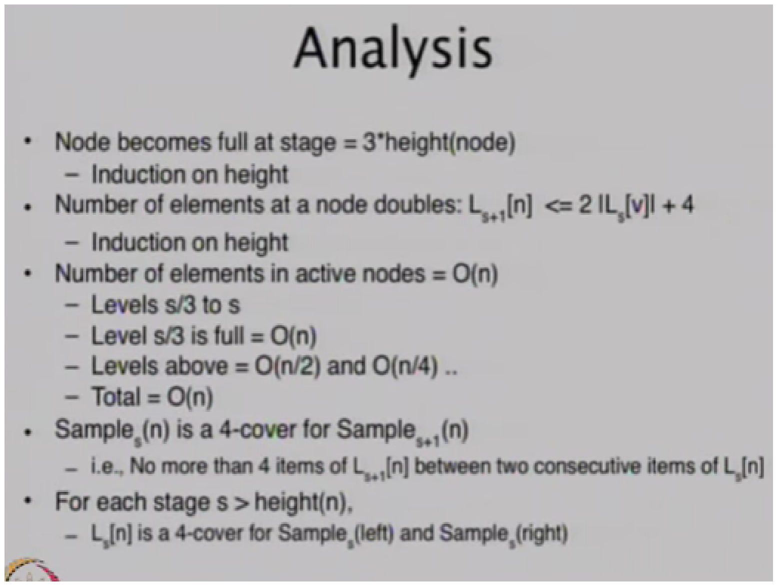
So here is an example we have got a node with two children with so far have generated some elements. We take the sample of the left and the right and we merge it the left and the right elements. We are not all the elements necessarily in the sub tree just like you. Now do not have all the elements in your sub tree. You have only one fourth of the element that left gave you and the one fourth that your right.

Or the one fourth the left had and the one fourth the right had the left itself may have had many fear, for example the left may have sample itself from its child which is much more dense. So as you reach up the tree as many levels are as active as they are given time, the bottom most level that is still active which is going to turn in active in the next step has everything still have to prove this things and will intensively thought by wide angles.

So at the bottom most level and every element has been received the next element and second has been received. Then the next and the before has been received and then onwards before fourth pipe. So it is a really spark sense at all, so on activation a node is going to merge sample of its children and the activation time is children sample every fourth by the role. We talked about earlier and which children deactivates at that time it is seem all of its it would have received all of its children and the next step every alternate will come in and the following step.

All the step, all the children will come in for merging at the parent and so at the end of it you would have merged everything that you would have merged in a regular merged way. You would have merged it in a several way it was merged just substance of it in a bit more, ultimately having received all of it. Just look at the analysis

(Refer Slide Time: 45:07)



Analysis

- Node becomes full at stage = 3*height(node)
  - Induction on height
- Number of elements at a node doubles: $L_{s+1}[n] \le 2|L_s[v]| + 4$
  - Induction on height
- Number of elements in active nodes = O(n)
  - Levels s/3 to s
  - Level s/3 is full = O(n)
  - Levels above = O(n/2) and O(n/4) ..
  - Total = O(n)
- $\text{Sample}_s(n)$ is a 4-cover for $\text{Sample}_{s+1}(n)$
  - i.e., No more than 4 items of $L_{s+1}[n]$ between two consecutive items of $L_s[n]$
- For each stage s > height(n),
  - $L_s[n]$ is a 4-cover for $\text{Sample}_s(\text{left})$ and $\text{Sample}_s(\text{right})$

And most of these statements have already made along the way. Every node becomes full meaning the part of its subject all the leaves it is something has reached there at the stage that it is thrice its height which is the last time it is going to be at. So just before the deactivation it makes sure that it is complete it has done what original merge tree would have done. How would you prove that suppose leaf level is complete.

Can you start receive leaf level has everything that every worker has. What about a general level? Assume that your child is complete three steps, later you are going to deactivate the member in three steps. Are you going to merge all your children? Yes because once at this till your child became inactive next step you will taking every alternate, the second step, you will be taking every one of them and you are active for three more stage.

So when you take every one of them the final stage where you do the merging. Third stage you will have merged all the elements that your children had at three steps earlier. So three steps later you are going to have all your children, because you are you are getting every element from the left and every element from the right and many of these things can be done on reduction the other is the number of elements have a node keeps doubling approximately.

This is like condition like so L is the list of elements at node N at stage S plus on sub S plus one like that and it is less than equal to twice of what its value towards its previous stage same node plus four. Again you can prove it; actually we just have to look on the three stages your sampling, every four, second or every one of them. Now in cases you will be doubling. Professor student conversation start, you sample doubling every fourth of production quite some time.

During that time child, your child is also sampling every fourth for quite a while. So your child would have been yes Professor Student conversation ends. The third statement, these are all basically level that you would be proving and not formally proving them. The number of elements in active nodes the O of N meaning that if you take up all the active nodes add up weather there is only that many elements to be worked on all the active nodes working together are going to be doing O of N work.

Although that many nodes active at the same time if you add up all the elements that they have it is all done. And this needs a little bit of more construction to prove it basically you would say that levels S by three to S . What is the number of nodes and then you say at S by three, you are going to become full because that is when your height stage by three is what your height is at stage equal to thrice your height.

You can inactive, so if you are height is at S by three you became inactive so you see that what happens at level S by three and then levels above. What happens again? This is just have a proof I do not intend to go to the detail of the proof. I will point you to paper that this comes to get more detail on the proof. I will post it online, but you can also go work it out its not a hard proof once you understand or you simply focus on the main thing.

Just keep adding the number of elements at each of these active stages or active level. Now here are the more important ones which is sample of A given node meaning sample of its list at stage S is a four cover or its sample at the next stage it is not too hard to see where you going to get at most four more things at a dense. In any of this stages and the other thing is that for every stage after your height meaning you become active your list is a four cover.

Your list parents list is four cover of the sample of left side and the sample of the right side again little bit of maths will get you to there it is just counting algorithm which means that I have a list and I am going to generate the sample of my left generate the sample of my right and merge it that is my next step. I already have an old list and I am creating a new list by taking sample of my left sample of my right and merging it is my old list itself is a cover for the two things.

I want to merge and I can merge in one time right that was the thing we talked about last class. If you have C cover available for two list and we know the rank which I have not told you yet about, if you know the rank of C cover of both A and B then we can find the rank of A and B and B and A which means, we have merge it in order one time here C is four them in four steps. So what will be the algorithm become at any stage I have the previous list which in the beginning is null.

I am going to get a sampling of my left sampling of my right and the previous list is going to help me merge two in order one. Because previous list is the cover for both of them. Because previous list actually came from the previous samples of the same thing. The new samples will just be among them more sample among the older one that we talk. So how Professor student conversation starts sir samples more we get if I take, let us say I have choosing my take the sample to current stage and I merge to get the early thing and previous thing.

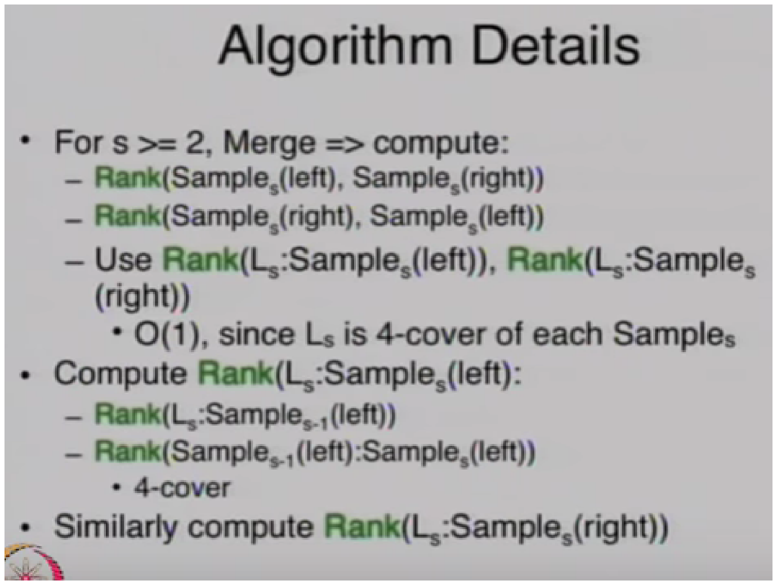
Now to the next stage I take new samples, so these new samples would not be, they will be no they have, if you go back to this picture and we look at this samples like suppose, the next time more things come in like meaning the those gaps that you see in the blue list get filled with let us see one more thing coming in or two more things coming in depending on the stage every fourth becomes different element not necessarily.

Suppose one more thing comes in then now these will remain. But this one element coming in between the old sample also old sample still remain the sample are still in the samples. They have to be there to order one exactly and the new samples are simply in between the old samples You had a question, Sir the new samples becomes the lower ones or the lower ones are having the C cover. Both of them, but the main thing you will want this.

Because your goal is to take the sample here and merge it which means and do it fast and do it fast means in order one which means if you give me C cover for the samples I am done and my previous list is a four cover of the samples. But C cover of the samples again be some samples from the previous ones. No the covers it is a sample from the previous stage. But it is my list I generated them previous stage.

So it is with me I simply know that is going to be a C cover for the new set of samples. Because all I have to do the next stage is take the sample merge them using my previous list and replace the previous list and next step will do the same Professor student conversation ends.

(Refer Slide Time: 55:49)



Algorithm Details

- For s >= 2, Merge => compute:
  - Rank(Sample$_s$(left), Sample$_s$(right))
  - Rank(Sample$_s$(right), Sample$_s$(left))
  - Use Rank(L$_s$:Sample$_s$(left)), Rank(L$_s$:Sample$_s$ (right))
    - O(1), since L$_s$ is 4-cover of each Sample$_s$
- Compute Rank(L$_s$:Sample$_s$(left):
  - Rank(L$_s$:Sample$_{s-1}$(left))
  - Rank(Sample$_{s-1}$(left):Sample$_s$(left))
    - 4-cover
- Similarly compute Rank(L$_s$:Sample$_s$(right))

Okay so here to merge what we need? We need a cover, but we also need the covers rank in both. How do you get the covers rank? So we want to find the rank of the new sample in my left child in the other child samples, the rank of new samples in right child in the left child sample. Those are the two things I am going to need rank of sample S left and and the sample S right, left meaning left child, right meaning right child. Sample means sample under S.

How do I find the rank of each of these samples? Or the other way round rank of my previous list in each of these sample knowing that previous list was a merger of the previous samples and the new samples are just more things among the samples seems durable and this is where you are going to use the fact that the lists that we generate is a cover for the samples. That is something that one of the elements in the previous slide and because it is four cover for the samples.

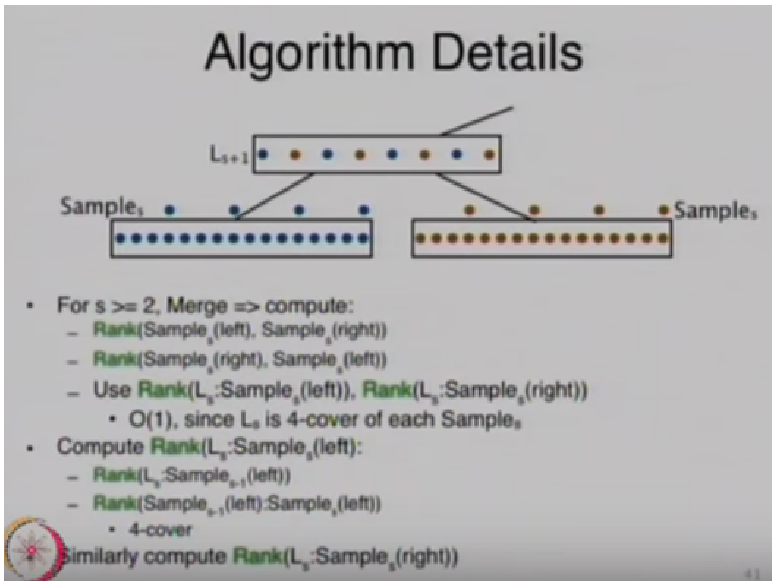
We know that the new samples, the whole lists can only have four new samples in between every consecutive value. So that can be done. Now how do you find the rank of the list of the new sample? So the previous statement says to find those ranks you are going to use. The rank LS of sample left and rank LS of sample right. But to compute LS of sample by going to the previous stage, we know it was a sample, it was a sample four cover of the previous stage.

So to find the rank of LS in the new sample about to find the rank of LS in the old sample and the rank of old sample in the New sample. Professor students conversation stats old sample starts when I merge that two that okay on a merge C if I have a blue sample in my, that is why it is only true, not necessarily. But just there may be two consecutive groups. They are just the merger of the two blue and red and in fact we are not very interested in what is how many reds are between blues samples.

We are interested in bringing out where this blue line is, just in the red in the merge list which we found out. Because when the previous stage, what we are trying to figure out, where do all of these blues go there and where about the reds go there? So we found that out in the previous when we merged, we figured out where the blue ones went, where the red ones went. But in the blue list, we figure out where are blue lies between which two reds and that is what we have to find out.

But you do it in kind of indirect way. We do it in two steps, we will say, we will know where the new Blues are among the old Blues and we know where the old blues among the merge list. So that is how we will figure out where the new blues are going to be in the merge list or vice versa same thing.

(Refer Slide Time: 1:00:42)



Algorithm Details

- For s >= 2, Merge => compute:
  - Rank(Sample_s(left), Sample_s(right))
  - Rank(Sample_s(right), Sample_s(left))
  - Use Rank(L_s:Sample_s(left)), Rank(L_s:Sample_s(right))
    - O(1), since L_s is 4-cover of each Sample_s
- Compute Rank(L_s:Sample_s(left):
  - Rank(L_s:Sample_{s-1}(left))
  - Rank(Sample_{s-1}(left):Sample_s(left))
    - 4-cover
- Similarly compute Rank(L_s:Sample_s(right))

Excuse me Sir can you say how we merge the first time co-ordinate? What is going to happen in the first one element will come there is nothing to do. So first three steps nothing happens in the beginning zeroth level. You say and not well there may be relative merging in the first two types, so let us see what happens in the beginning. The leaves have one element each what does the parents say give me every fourth and he gets nothing because we know every fourth then it is again next time itself give me every fourth.

Next time it is giving me every second, still does not get anything. Next time give everyone of them it gets one write it is going to generate one. But in the meantime when it gets two of them its parents give me every fourth and it does not happen. Ultimately it is going to give me one that is what it gets four elements as soon as your child gets. So it will be one to the parent it is merging one on one is constant time and then you start to build on top of it.

But the entire most thing, we are considering either starts with first two local, no this is full complete merge tree. This is using whatever number of processes you need, regular main algorithm. We are not talking about doing P separately, where we had slightly different type of analysis. This is back to the old time work framework.

So that is picture is showing we have to generate the ranks of the red samples and the blue samples. We are going, where the previous list has come from and you know the previous list came from the old samples. So you can already know where to locate them, look at this question as you merged it how long out of time also. But let me finish this, how long does this take probably it will take time. Let please stop.