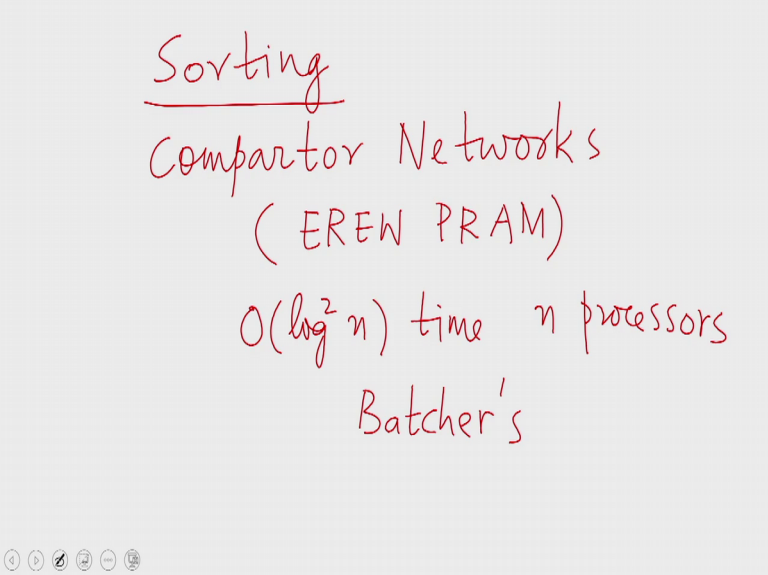
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 18
High level Description

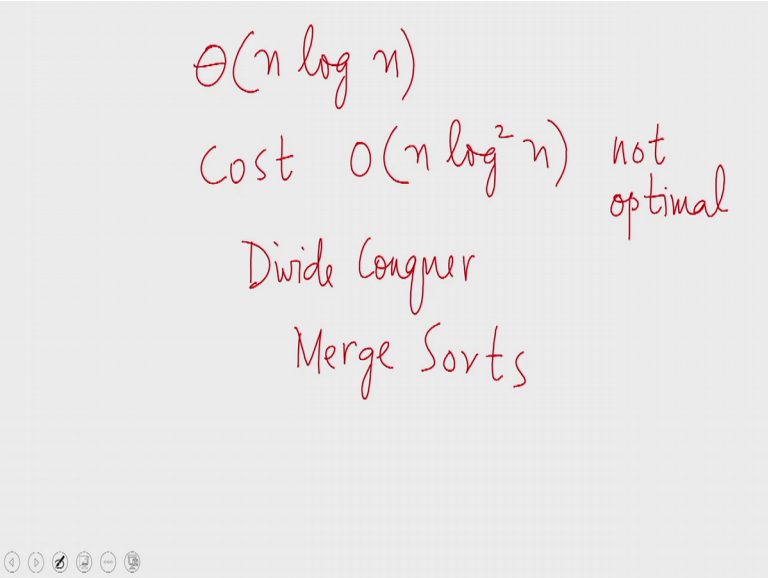
Welcome to the 18th lecture of the MOOC on Parallel Algorithms. In this lecture we are going to revisit a problem, they have seen before namely sorting. We have seen a couple of algorithms for sorting before.

(Refer Slide Time: 00:45)



In particular we consider algorithms on comparator networks. We saw that algorithms that we designed for comparator networks can be simulated on EREW PRAMS for the same time complexity and process of complexity. We designed two algorithms that transcend order of log square n time using n processors that is the complexity of the algorithm was order of n log square n these algorithms are due to batchers.

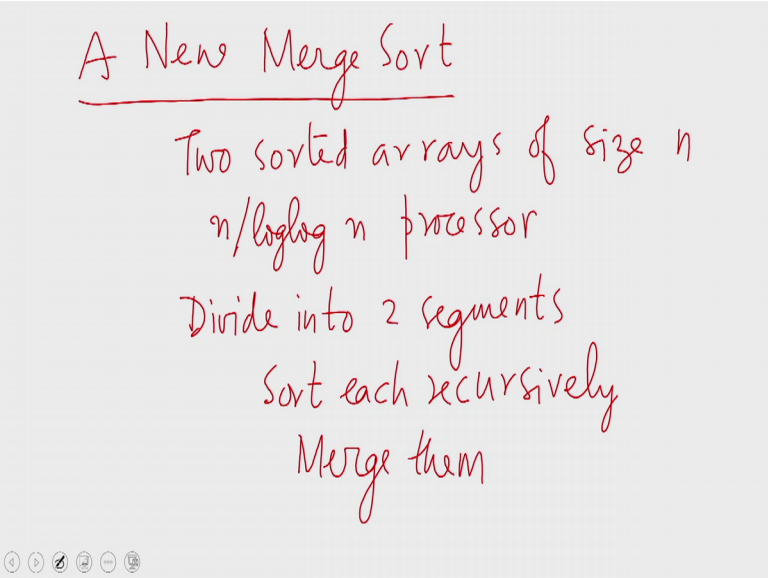
(Refer Slide Time: 01:49)



So, they were the odd even merge sort and the bitonic sort merge sort. But then we know that the sequential time complexity of sorting us order of n log n theta of n log n to be precise. So, the algorithms that we discussed they were not optimal, they had a cost of order of n log squared n so, they were not optimal. They were divide and conquer algorithms and essentially they use a merge network as the basic building block.

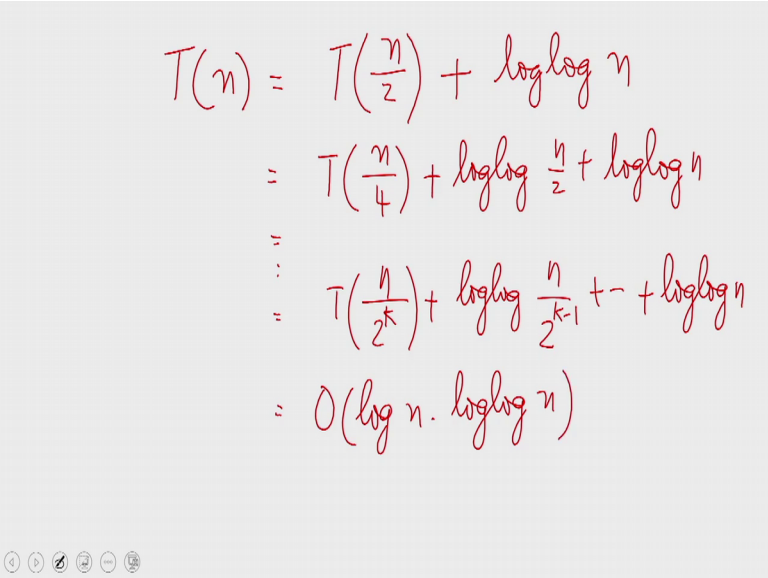
So, they have merge sorts. Now in the last lecture we saw an algorithm for merging two arrays of size n each using n by double log n processors in order of double log n time. This is an optimal merge algorithm for the CREW PRAM. Now once you have an optimal merge algorithm you can use it to construct a merge sort. So, let us see how we can use the merge algorithm that we saw in the last class to construct a new merge sort.

(Refer Slide Time: 03:03)



So, let us say we are given two sorted arrays of size n each, and we have let us say n by double log n processors. So, what we do is the classic divide and conquer, divide the array into 2 segments, sort each segment recursively with half the processors and then merge them using the algorithm of the last class.

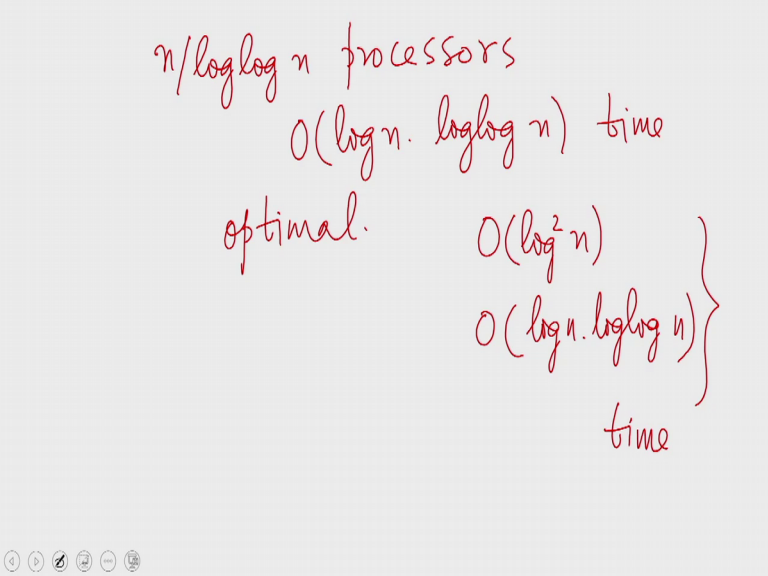
(Refer Slide Time: 04:19)



So, that is what the algorithm is. So, to merge we would require merging of an array of half the size plus double log n time. So, this is what the time complexity is.

So, if you continue like this, which if you continue will come to after k levels of unrolling we will have. So, if you put k equal to log n you find that this is order of log n double log n.

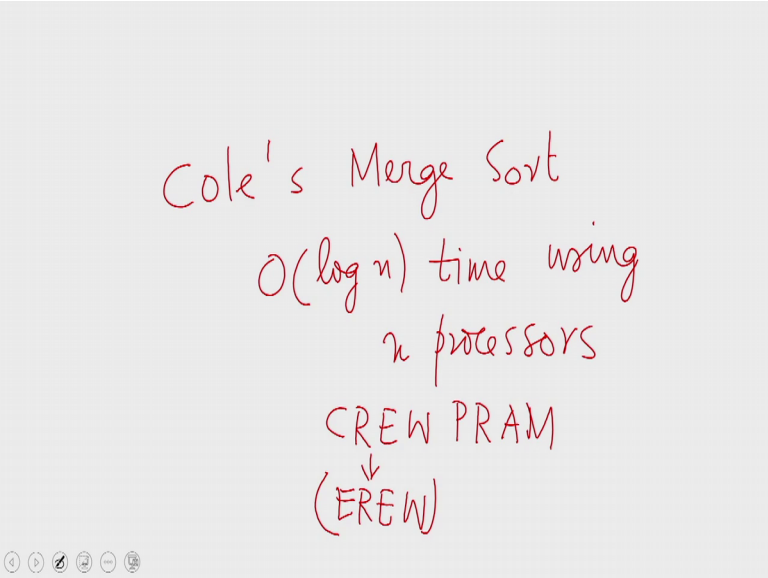
(Refer Slide Time: 05:37)



So, with n by double log n processors, we managed to sort an order of log n double log n time, therefore this algorithm is optimal and is an improvement over batches algorithms for the CREW PRAM because batches algorithms run in order of log square n time whereas, here this algorithm runs in order of log n double log n time.

So, using the algorithm of the previous class we can immediately obtain a sorting algorithm which is faster than batches sorting algorithm, but the algorithm that we are going to do study today hereafter actually does better.

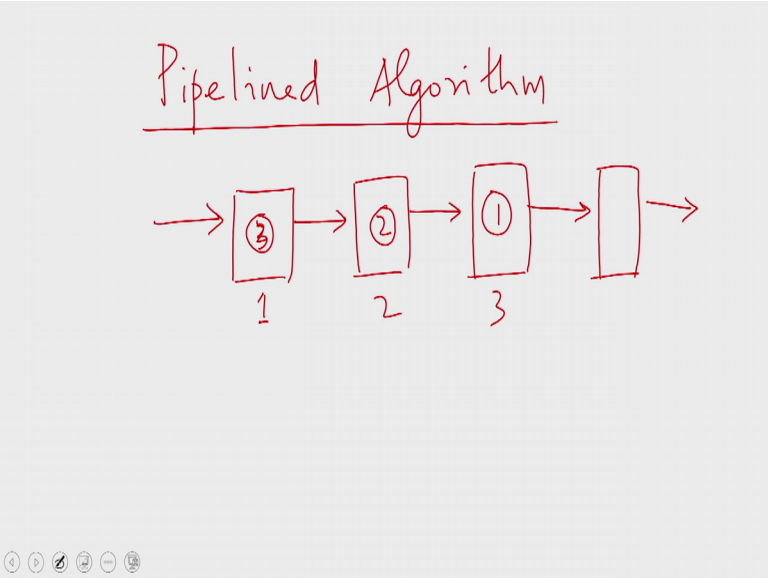
(Refer Slide Time: 06:47)



This is the algorithm that is famous as Cole's merge sort after it is inventor Richard Cole. So, in Cole's merge sort we managed to sort n elements in order of log n time, using n processors.

We shall study the version of the algorithm that runs on a CREW PRAM. In fact, you can improve the model to EREW PRAM, for the same time in processor complexity, but this version we will not study in this course because the algorithm is slightly more complex when we come to the EREW PRAM. So, we will study the CREW PRAM version of Cole's merge sort.

(Refer Slide Time: 07:51)

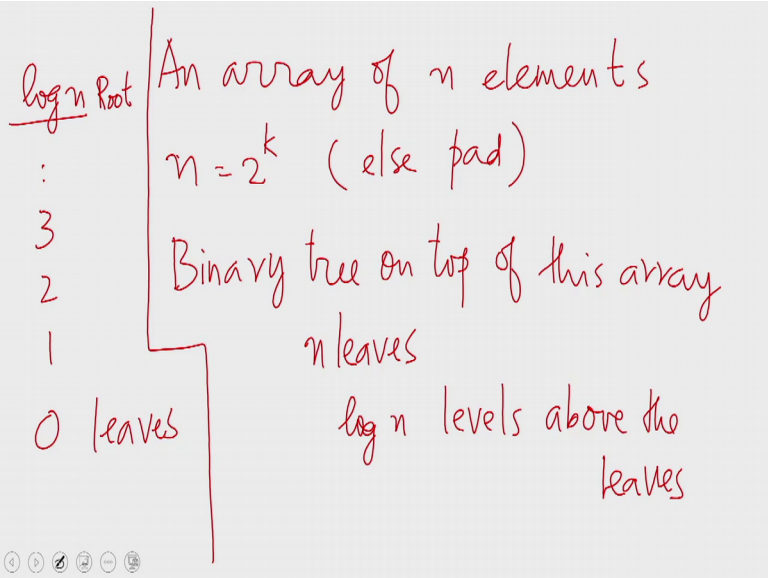


Now, Cole's merge sort is a pipelined algorithm; a pipelined algorithm is where you divide the algorithm into several components, and the data flows through the components in this fashion.

So, all the data is not sent through the algorithm simultaneously, a part of the data is 1st given to block 1, after block 1 has processed the 1st part of the data, it sends it to the 2nd component. So, when the 2nd component is processing the 1st part of the data, the 1st component will take the 2nd part of the 2nd component of the data and process it, after that all the data will be send one step forward.

The 1st part of the data when it reaches the 3rd component, the 2nd part of the data would have reached the 2nd component and the 3rd part of the data would have reached the 1st block, 1st component to the algorithm. So, the data pulses through the components of the algorithm in this fashion. Therefore, we get a speed up, such an algorithm is called a pipeline algorithm this is very similar to the pipelined architecture, you would have studied in advanced architecture courses.

(Refer Slide Time: 09:33)

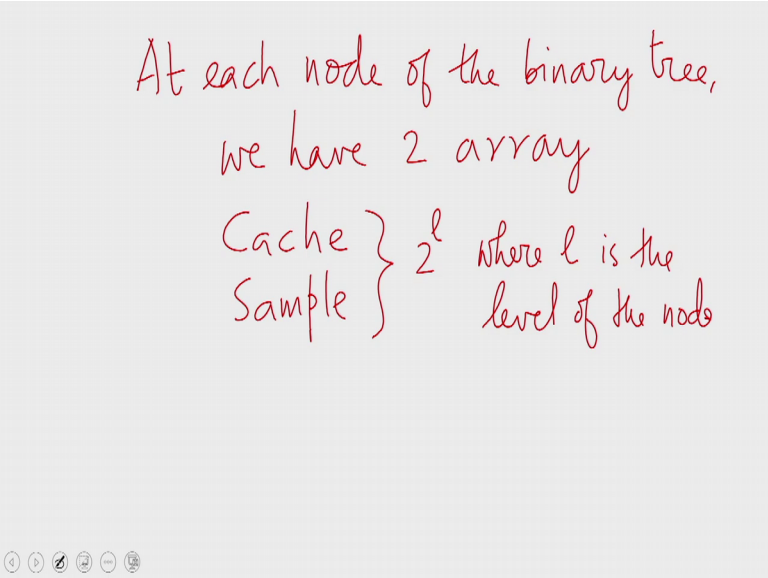


The same idea is being applied here. So, in Cole's merge sort let us say we are given an array of a n elements to sort. Let me assume that n is an exact power of 2 otherwise we pad the array with dummy elements that are larger than all the given elements. So, those dummy elements after sorting will be at the right end and can be removed. So, let me assume that n is an exact power of 2 and then what we do is this, we construct a binary tree on top of this array.

So, this is the binary tree with n leaves, so, it will have log n levels above the leaves. So, let us say the leaves are labeled 0, that is the leaves are at the 0th level the parents of the leaves are at level 1 then we have level 2 and so on, and the highest level is labeled log n this is where the root is. So, we construct a binary tree on top of the given array of n elements.

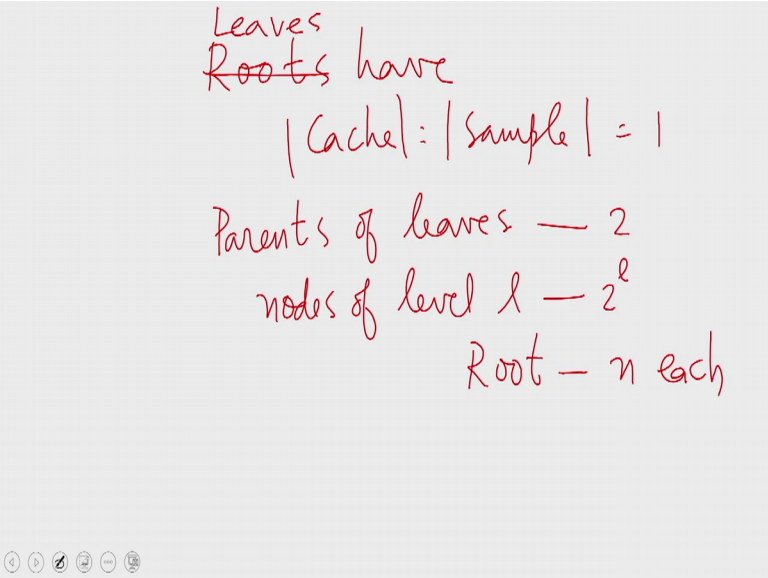
So, this binary tree has log n plus 1 levels on the whole, the leaves are at level 0 and the root is at level log n.

(Refer Slide Time: 11:29)



And then in this binary tree we add some extra data structures, at each node of the binary tree. We have 2 arrays; 1 array is called a cache and the other array is called a sample. Both of these have a size of 2 power l where l is the level number, which means leaves have caches in samples of size one each. And the root has cache and sample of size n each.

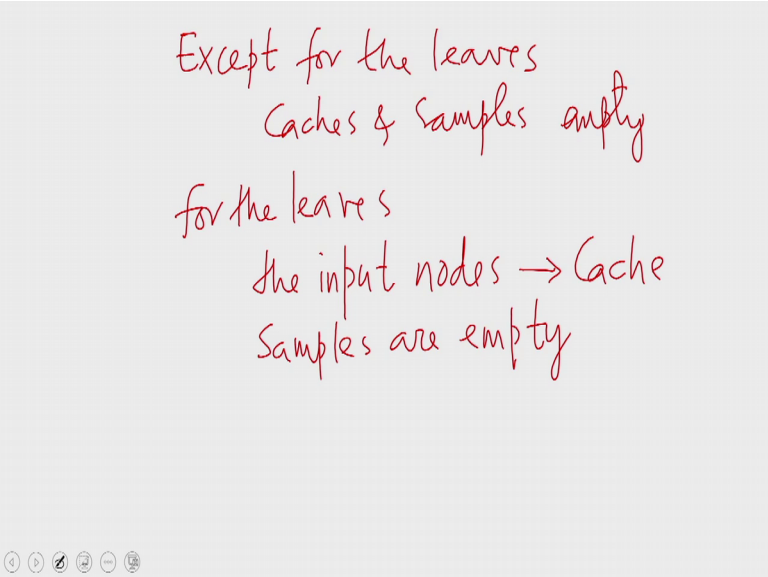
(Refer Slide Time: 12:27)



The roots have the cache size equal to the sample size equal to 1. The sorry the leaves the leaves have cache and sample size of 1 each. Parents of leaves have a size of 2 each. In

general nodes of level k nodes of level l have a size of 2 power l the root has arrays of size n each because the roots at level log n.

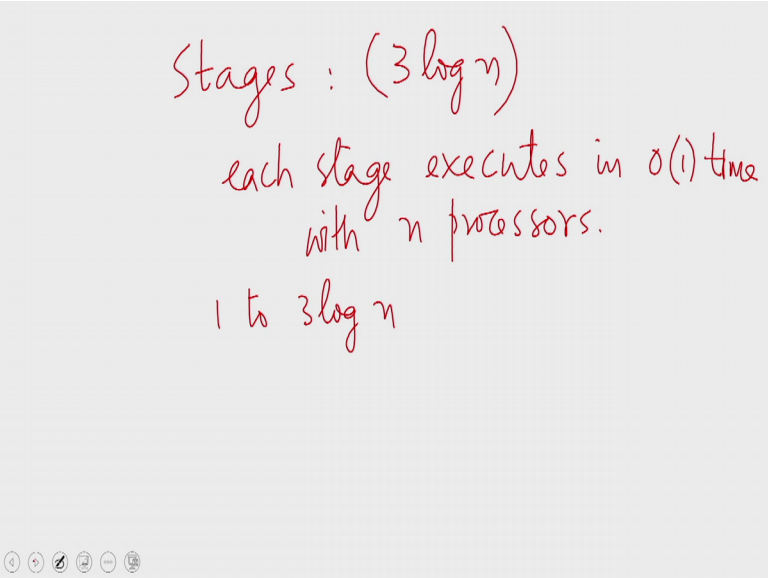
(Refer Slide Time: 13:23)



So, after setting up arrays in this fashion except for the leaves, the arrays are initially empty. The caches and samples of all nodes other than the leaves are initially empty. For the leaves the input nodes occupy the caches, remember the leaves have caches of size one each. So, the cache can hold only one element, one input element is held by each node. The sample arrays are empty for the leaves as well.

So, at the beginning of the algorithm the sample arrays are empty all over. The caches are the cache arrays are empty everywhere accepted the leaves, at the leaves each cache will hold one element. The caches of size one and it holds only one element.

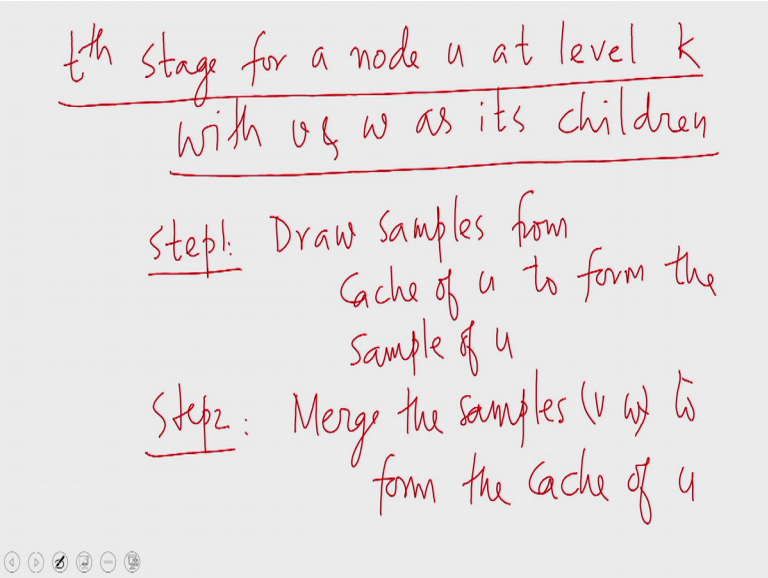
(Refer Slide Time: 14:41)



And then our algorithm executes in a number of stages: the algorithm is made up of several stages. As we shall see there will be 3 log n stages, in the algorithm there will be 3 log n stages, when we have n items to sort.

We shall end up showing that, each stage executes in order 1 time. Therefore, the overall running time of the algorithm will be order log n. So, our stage will execute an order 1 time with n processors. So, with n processors the algorithm will run in order of log n time. So, let us the let us stages be numbered from 1 to 3 log n and the algorithm procedure in this fashion.

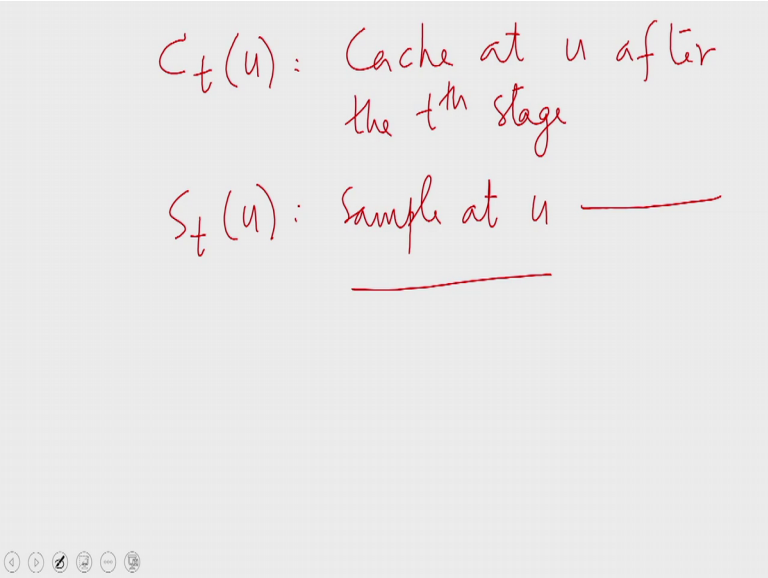
(Refer Slide Time: 15:43)



Let me specify a stage, for a node u at level k let us say with v and w as it is children. So, let me specify in particular the t th stage. So, I am going to specify the t th stage for a node u at level k with v and w as it is children. So, in the stage we have 2 steps; in the 1st step, we draw samples from the caches from the cache of u to form the sample of u. In step 2 we merge the samples of the children, in this case v and w to form the new cache of u.

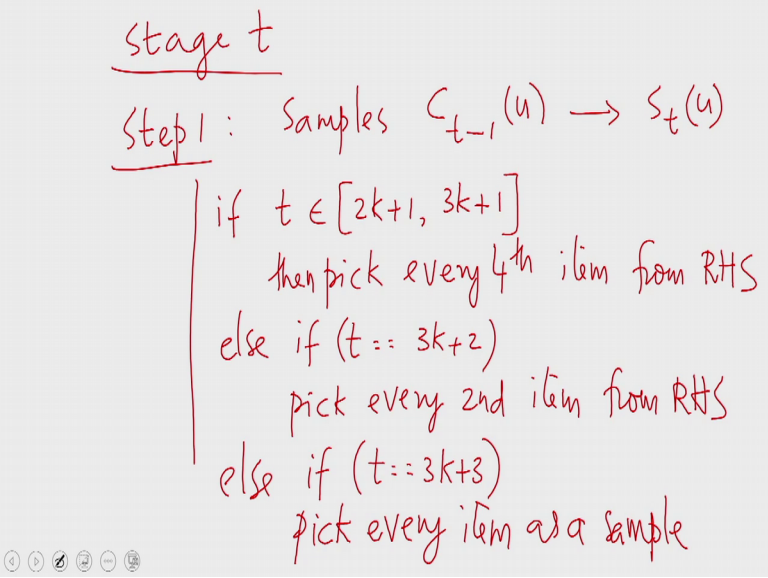
So, a stage of the algorithm consists of these 2 steps; in the 1st stage we draw samples from the cache of u to form the new sample of form the sample of u. And then we in step 2 we merge the samples of the children v and w to form the new cache of u. So, here some definitions are necessary.

(Refer Slide Time: 18:03)



So, let me define C t of u as the cache at u after the tth stage. Similarly by S t of u I will denote the sample at u, after the tth stage likewise. Now with these let me give more specification for the steps of a stage.

(Refer Slide Time: 18:43)



In stage t a node u at level k with v and w as it is children, thus this in step 1, it draws samples from the old cache at u to form the new sample at u. So, this is how S t of u is formed once it has formed in step one it does not change in this stage. So, in stage t S t of u will be formed by drawing samples from C t minus 1 of u. So, the drawing of the

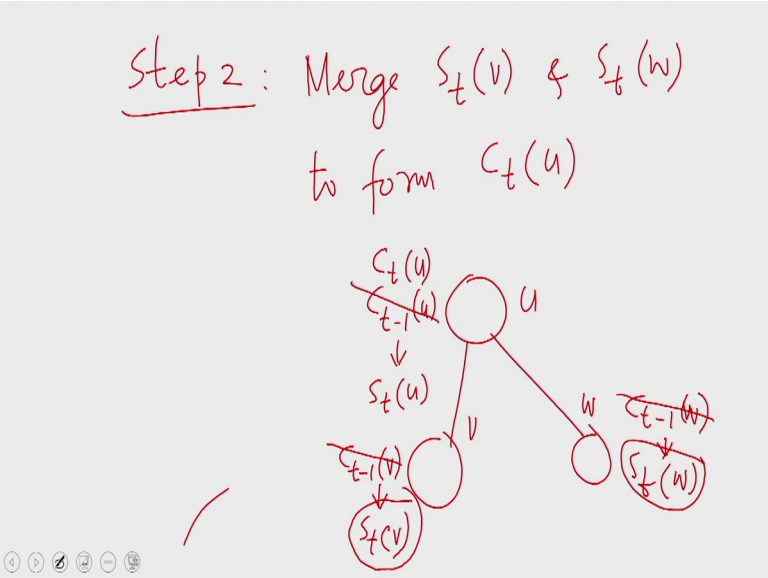
samples is done like this, if the stage number t belongs to this range that is 2 k plus 1 to 3 k plus 1 remember k is the level number of node u for which we are writing the stage.

So, if the stage number t belongs to 2 k plus 1 to 3 k plus 1 this range. Then, we pick every 4th item, from the right hand side as a sample. The sampling is done like this, if t belongs to the range 2 k plus 1 to 3 k plus 1 then we pick every 4th item from the right hand side as a sample, else if t is 3 k plus 2 we pick every 2nd item from the right hand side as a sample. And finally, if t equal to 3 k plus 3 we pick every item as a sample.

So, once again the samples are picked from C t minus 1 of u which is the cache r at u at the end of stage t minus 1. In this fashion to form the sample array of u, that must be left behind at the end of stage t. So, the sample array is formed in this fashion. If the stage number t happens to be between the range 2 k plus 1 to 3 k plus 1 it is within this range. Where k is the level number of node u, Then we pick every 4th item from the right hand side of C t 1 of u as a sample. Else if p is 3 k plus 2 we pick every 2nd item from the right hand side of C t minus 1 of u as a sample, else if p equal to 3 k plus 3 we pick every item as a sample that is every item of C t minus 1 of u.

So, these are the actions for step 1, remember if t happens to be greater than 3 k plus 3 or less than 2 k plus 1 there is no sampling. The sampling will happen at u precisely when t is within this range that is t is within the close range of 2 k plus 1 to 3 k plus 3 outside of this range there will be no stamp sampling at u. So, whether sampling will happen at u depends on both t and k. Then where t is the stage number and k is the level number of u.

(Refer Slide Time: 22:47)

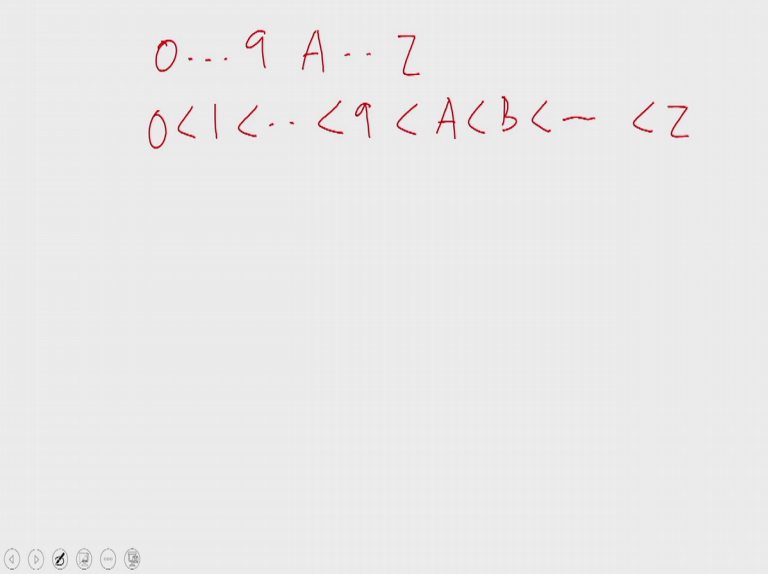


Then after having formed the samples, in the 2nd step of the t th stage what we do is this. We merge the samples that have just formed at the children S t of v and S t of w to form C t of u. So, at the end of the stage number t, C t of u is the cache at u. This will override C t minus 1 of u from which we had been picking the samples.

So, what happens is that node u, which had cache C t minus 1 of u will generate the sample S t of u. At the same time the children v and w of u will also be doing the same thing from C t minus 1 of v. We would be forming the sample S t of w here from C t minus 1 of w we will form S t of w. And here from C t minus 1 of v we will form S t of v.

And then the samples S t of v and S t of w will be merged to replace C t of u. The result of the merge of S t of v and S t of w will be replacing the cache at u. And this replaced cache will form C t of u. The cache at u at the end of the t th stage. Simultaneously the caches at v and caches at w will be replaced by the samples of their children merged appropriately.0

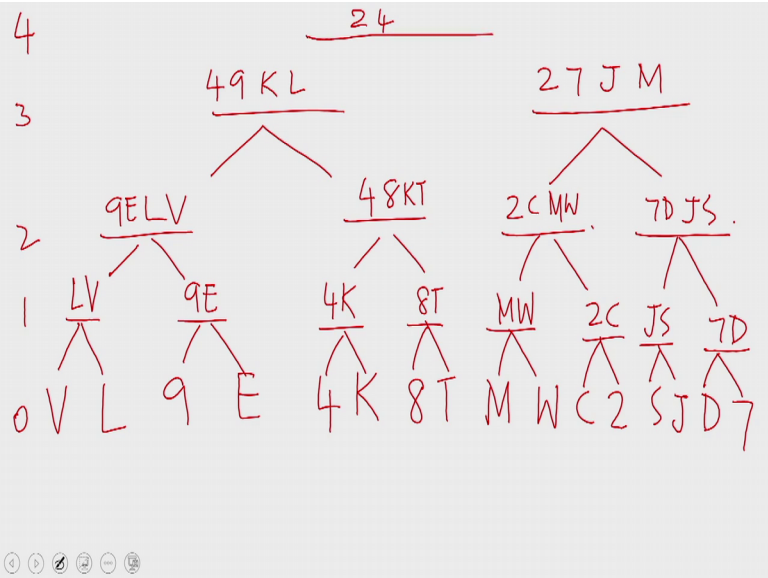
(Refer Slide Time: 25:03)



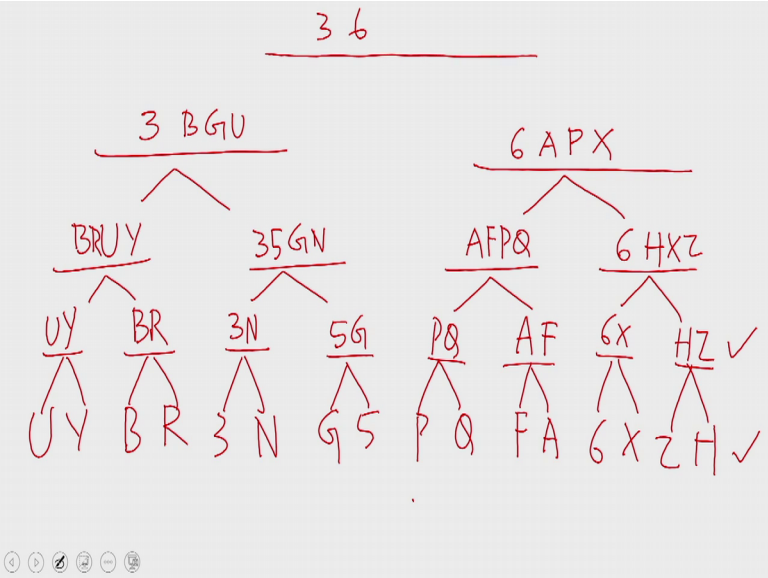
So, all the caches at all the nodes will get updated simultaneously. So, that is the structure of the algorithm. Now let us take an example and see how the algorithm works on real data. So, let us say we have the key values from 0 to 9 followed by A to Z let us say this is the increasing order 0 less than 1 less than 2 less than etcetera. And 9 is the largest then 9 is smaller than A, A is smaller than B and so on. So, we shall assume this linear order for the key values.

So, with these key values let us say we have an input that we want to sort. We will take an input of size 32. So, let me consider one half of the input. Those are the 16 1st 16 elements, then the next 16 elements I will draw in the next screen.

(Refer Slide Time: 25:37)



(Refer Slide Time: 26:07)



So, let us say these are the 32 elements. These elements occupy the cache arrays of the leaves. So, what I have drawn now are the leaves, the caches of the leaves. So, the caches the sample the cache arrays of the leaves are full when the algorithm begins.

So, we assume that at the end of the hypothetical stage 0, all the cache arrays of the leaves which are at level 0 are filled. Then in stage 1 the leaf nodes are supposed to be every 4th element from the right hand side as a sample. But then since the cache array is at the leaves have only 1 element each, you will not be able to pick any element here. So, in the 1st stage actually nothing gets done. Every leaf is attempting to pick the 4th element from the right hand side of it is cache array.

But since the cache array has only one element nothing gets picked. So, no sample is picked. So, the 1st step is essentially our dummy step. We will keep numbering the steps in this fashion, so that the induction holds good. So, for the purpose of the induction we assume that in the 1st step nothing happens. In the 2nd step every leaf node will attempt to pick the 2nd element from the right hand side as a sample, but then again since the cache has only one element nothing gets picked. So, the 2nd step is also a dummy element.

Now, in the 3rd step every leaf node will pick all it is all the elements in all the elements in it is cache array into the sample array which means the elements are copied into the sample arrays of the leaves. So, the sample arrays will look exactly like these. And then

in the 2nd step of stage 3 the sample arrays of the children will be merged into the cache arrays at the parents.

So, now the parents of the leafs are going to get something in their caches. So, the parent of the leftmost nodes v and l will get these elements in sorted order in their cache arrays. So, the parent will end up getting L n V in sorted order in it is cache, and these two nodes will get 9 and E which is the sorted order and here it is 4 n K and here it is 8 and T they are already in sorted order MW C 2 are not in sorted order.

So, they would be put right 2 C J S are in sorted order and then 7 D. So, this is what happens at the end of the 3rd stage. For the 2nd half of the tree we will have U Y BR 3 N 5 G PQ AF 6 X and HZ. So, the cache arrays of the parents of the leafs are filled at the end of the 3rd step. So, we are now at the end of the 3rd step. Then in the 4th step the leaf nodes are done so, there will be no more drawing of samples from the leaf nodes.

But the parents of the leaf nodes will attempt to draw every 4th element from the right hand side of the cache arrays as samples. But then these nodes have only two elements each. So, nothing gets picked. Therefore, the 4th step is also a dummy step like the 1st and the 2nd steps. But we will keep it in, so that the induction holds good. Now in the 5th step these nodes, that is the parents of the leaves will attempt to pick every 2nd element from the right hand side as a sample.

So, in particular the node which contains LV will pick L as a sample. And the node which contains 9 E will pick 9 as a sample. And then, in the 2nd step of the algorithm 9 and L will be merged at the parent at the end of the 5th step we are at the 5th step now at the end of the 5th step, the cache of this node will contain 9 and L. That is because L is chosen as the sample of the left child and 9 is chosen as the sample of the right child.

So, the parent will end up getting 9 L. Here we have 4 and 8 coming in. And here we have 2 and M and here we have 7 and J. In the 2nd half of the tree we have a U and B. And here we have 3 and 5. And here we have A and P. And here it is 6 and H. So, at the end of the 5th stage this is how the cache arrays will look.

So, by the end of the 3rd stage, the leaf nodes were done they are not drawing samples anymore. So, now we have we are done with the 5th stage. Now in the 6th stage nodes at level 2, the leafs are at level 0 these are at level 1 and these are at level 2. The nodes at

level 2 will attempt to pick every 4th element as a sample. But then since they have only 2 elements each nothing will be picked as a sample. But then their children that is the nodes which are at level 1 will pick every element as a sample.

So, L and V will come as the samples from the left most node, and 9 and E will come as the samples from the next node. Now in the 2nd step of the stage, this will be merged at the parent. So, the parent at level 2 is now going to contain 9 E L V. And here, we are going to get 4 K 4 4 8 K T. Then 2 CM W here and 7 D J S here, at this point all these level one nodes are done. So, this is in the left half of the tree in the right half of the tree we have a B R UY 35GN AFPQ 6 HXZ.

So, at the end of the 6th step, all these nodes are done. So, that is the tree as it looks like at the end of the 6th stage. Now we come to the 7th stage, in the 7th stage levels 0 and 1 are done. But level 2 can still emit samples. So, here level 2 will pick every 4th element from the right hand side as a sample. So, from this node we will get 9. And from the next node we are getting 4. So, these will merge to form the cache at the parent.

So, at the parent now we have 4 and 9 forming the cache, and here we have 2 and 7 forming the cache. That is because 2 is picked as the sample from this node and 7 is picked as a sample from this node. In the right half of the tree we will have 3 and 5 coming here 3 and B coming here, and here we have 6 and A coming from the children. So, this is what happens at the end of the 7th stage.

So, at the end of the 7th stage levels 2 and 3 are still active. Level 2 is emitting samples and level 3 has not started drawing samples yet. And then in stage number 8, the level the level 2 nodes will be picking every 2nd element from the right hand side as samples. So, the node which contains 9 ELV will pick nine and l as samples and the node which contains 4 8 KT will pick 4 and K as samples.

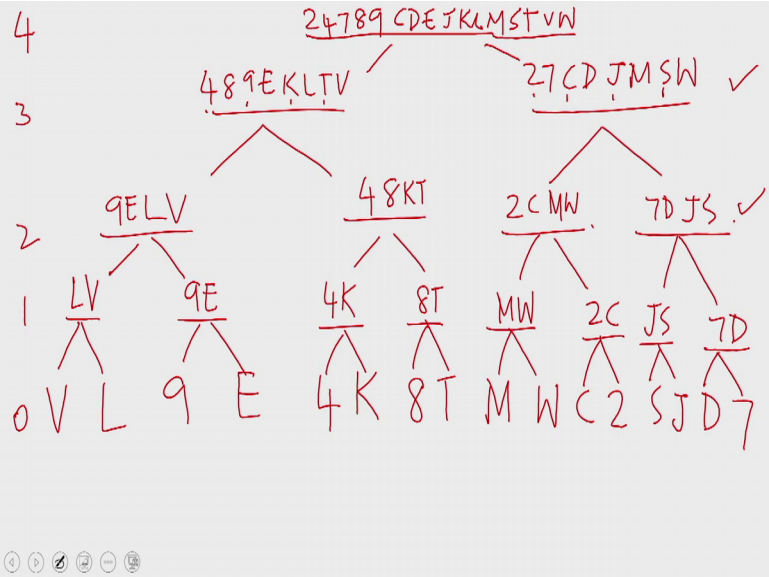
So, in the 2nd step this will be merged at the parent. Now, at the parent since this node that is the level 3 node is picking every 4th element from the right hand side as a sample nothing will be picked. So, the 4th level will not be populated yet. So, in the 8th step we have, a 4 K and 9 L coming up that will be 4 9 KL here we have sorry it should have been 2 and 7 here.

Now, we have 2 M 7 J coming up, so we have 2 7 J M here. And here we have BU and 3 G coming up. So, that will be 3 B GU and here we have AP and 6 X coming up. 6 A P X would be sorted order. So, this is how the tree looks like at the end of the 8th stage. Now when we come to the 9th stage the level 3 nodes will pick every 4th element as a sample and send upwards.

So, now the 4th level will be populated for the 1st time it will be getting 2 and 4, at the 4th level in the 1st half of the tree. In the 2nd half of the tree we will have a 3 and 6 coming to the 4th level that is for the 4th level. Now for the 3rd level we get samples from the 2nd level. But the 2nd level is now picking every element into the sample array.

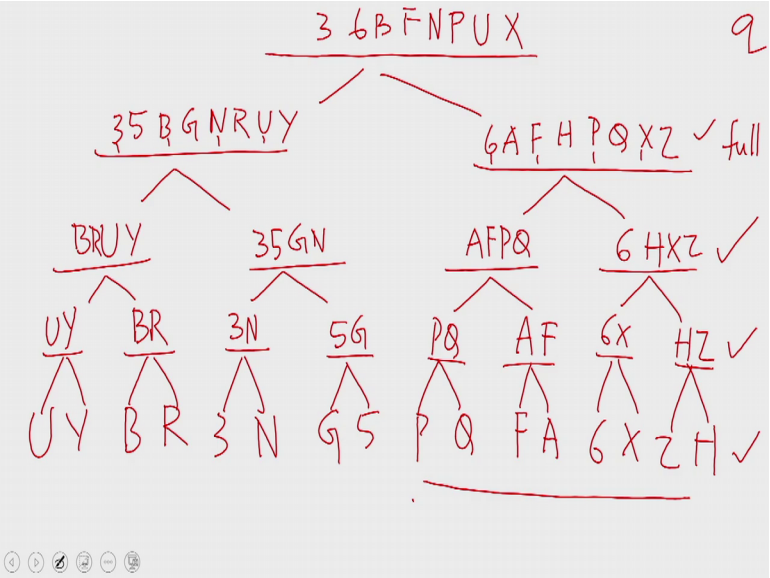
So, the sample array of the node which contains 9 ELV in the cache will be the same that is 9 ELV all the elements are now moving upwards to the 3rd level, similarly 4 K 4 8 KT.

(Refer Slide Time: 38:15)



Will be sending all those elements upwards; so, at level 3 now we will have all the elements in sorted order. That is we will have, 4 8 9 E K L T V and here at the next node we will have all the elements in sorted order which is 2 7 CD J M SW that is in the 1st half of the tree.

(Refer Slide Time: 38:51)



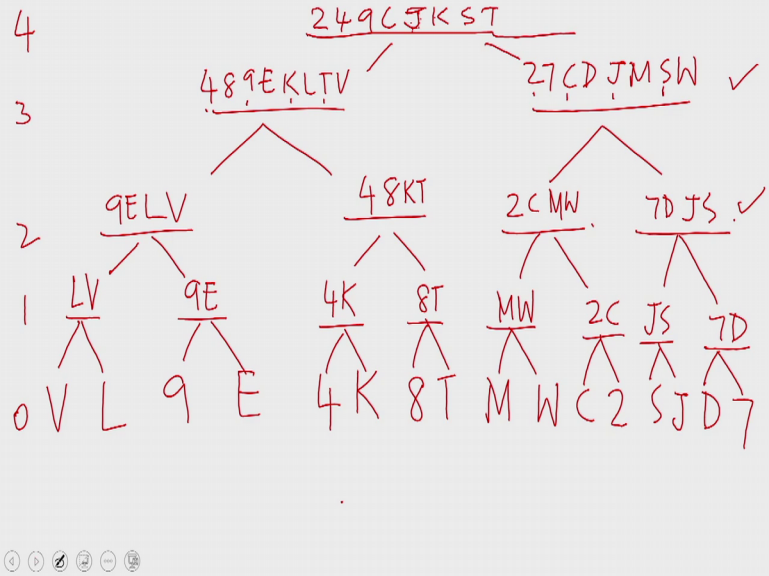
In the 2nd half of the tree we have, 3 5 BG NR UY these are the elements coming from the 2 children. AF P Q and 6 H XZ merged will form the cache which will replace 6 AP X. So, 6 AP X will be replaced with 6 A F H P Q X Z with this the 2nd level is also done. So, by the end of the 9th stage, you can see that the 3rd level is full. That is the 3rd level has acquired all the elements in the leaves of the sub tree that is rooted there which means the sub tree that is rooted at this node contained 6 AF H P Q X Z.

This is the sorted order of the actual set. The actual set is listed at the leaf in the order P Q F A 6 X Z H this gets sorted into 6 A F H PQ X Z at this node. Similarly at every level 3 node we have now collected all the leaf, or the leaf condensed of the sub tree rooted at that node in sorted order. So, level 3 nodes are now full. They are not done with sampling yet level 2 done the level 2 nodes are done with sampling, but level 3 nodes have just become full in their caches. But they have still more sampling to go.

So, now we come to the 10th stage, in the 10th stage we will be picking every 4th every 2nd element from these nodes. The level 3 nodes as samples. So we will have 4 9 K and T as the samples from here, and 2 C J S as the samples from here level 4 nodes will attempt to pick every 4th element from the right end, but since they have only 2 elements each, they will not be able to pick anything. Therefore, there will be no data moving from the 4th level to the 5th level yet. But the caches will now be replaced with the sorted order of these elements, 2 4 7 8 9 C D E J K L M S T B W.

This node has collected these sorry we will be picking only the marked elements. This will happen later, at this point we will be sorting 4 9 KT with 2 C J S. So, we have 2 4 9 C J K.

(Refer Slide Time: 42:03)



So, at the end of the 9th phase that is what we have the nodes at level 3 have become full. Now we come to the 10th stage, in the 10th stage we will be picking every 2nd element from the 3rd the level 3 nodes as samples. And they will be transport or upwards at the level 4 nodes since we have only 2 elements each, nothing will be picked as a sample. So, nothing will be sent to the 5th level yet.

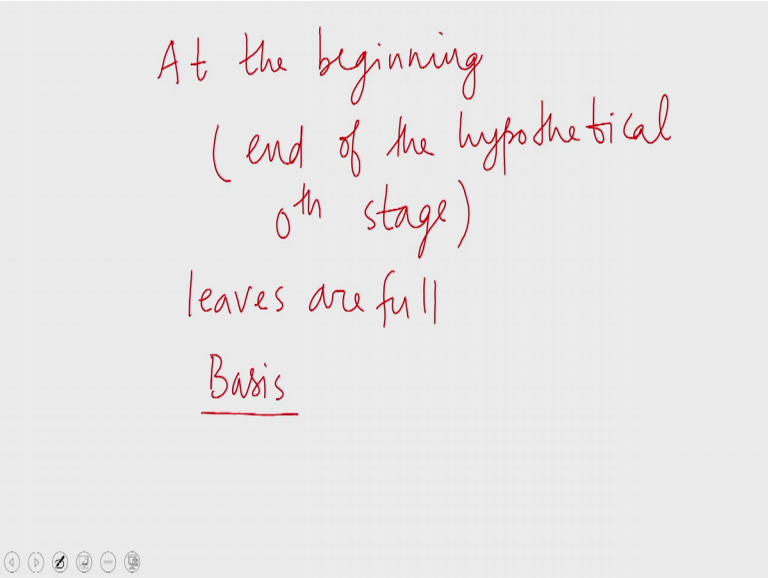
From the level 3 node here we will be picking, 4 time K T and 2 C J S respectively to send upwards. So, we will have 2 4 9 C J K S T at this node. And here we have 3 B N U going up and 6 F P X going up. So, we will have 3 6 B F N P U X at this node. So, at the end of the 10th stage this is what the array looks like. And then in the 11th stage, the level 4 nodes will be picking every 4th element. Which means it will pick 3 and N to send up and this node will pick 2 and J to send up.

So, we will have 2 J 3 and N going up. So, at the 5th level we will have the sorted order of those elements. That is what happens in the 11th stage at level 4 nodes. At level 3 nodes, we will be picking every element from 3 and then all of them will be sent up sorry, we are at the 12th step, in the 12th step in the 12th stage. We will be sending every

element from the level 3 nodes upwards. So, level 4 node will now end up containing every single element.

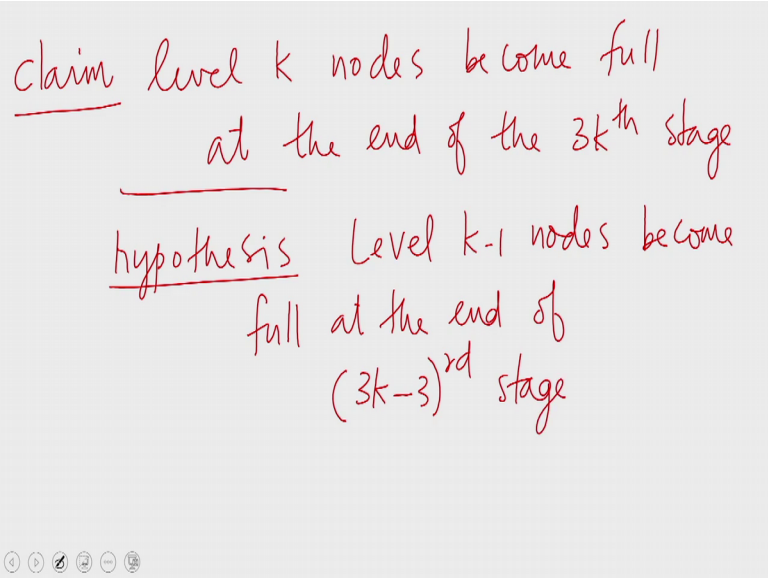
So, a level 4 node becomes full at the 12th stage. So, that is how the algorithm proceeds. So, I am leaving the rest of the tree for you to sketch up.

(Refer Slide Time: 44:49)



So, you can see that, at the beginning of the algorithm which I can assume is the end of the hypothetical 0th stage, for the sake of the induction I will assume that we are at the end of the hypothetical 0th stage. We assume that we are at the end of the hypothetical 0th stage at this point we find that the leaves are full. So, this forms the basis.

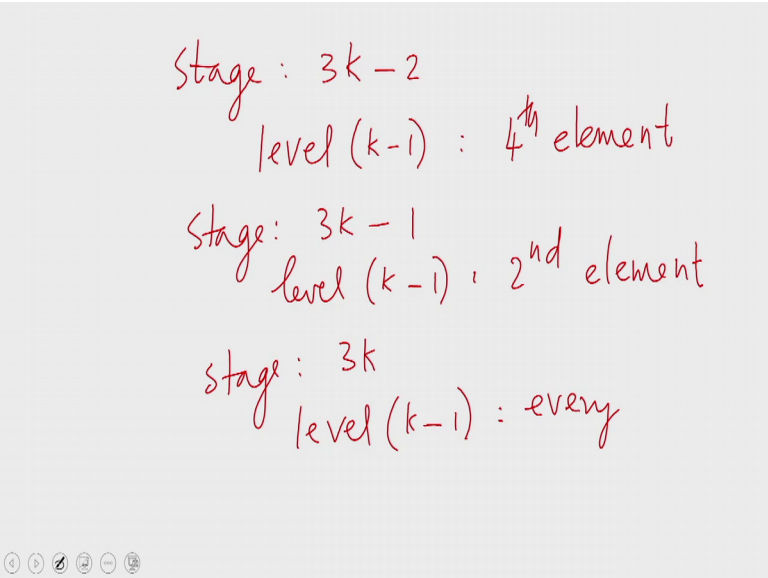
(Refer Slide Time: 45:35)



So, inductively I want to argue that, level k nodes become full, at the end of the 3 k th stage. This is what we want to claim inductively, for which the fact that at the beginning we had the leaves full will form the basis.

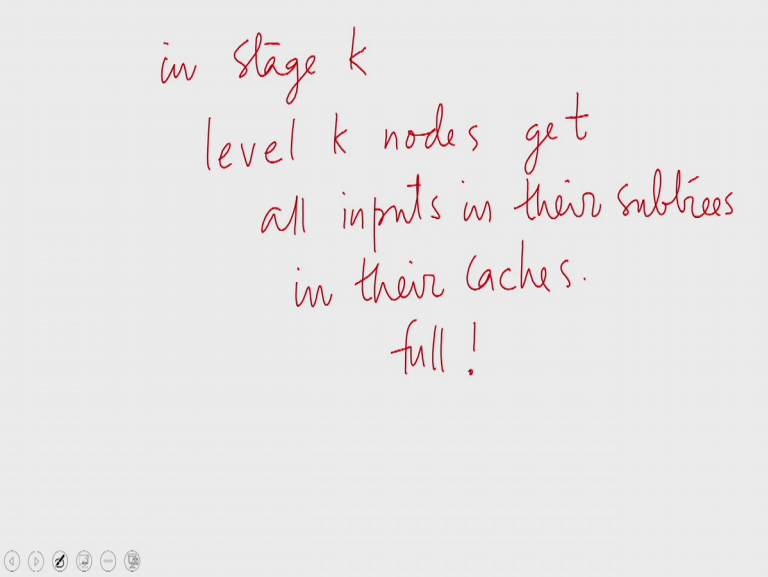
We assume that the leaves are at level 0, and at the end of the 0th stage which is a hypothetical stage the leaves are all full. So, that forms the basis. Now by inductive hypothesis, we have that level k minus 1 nodes become full at the end of the 3 k minus 3rd step 3rd stage.

(Refer Slide Time: 46:59)



Then from the algorithm we see that in stage number 3 k minus 2 level k minus 1 nodes, fix every 4th element as a sample. In stage number 3 k minus 1, level k minus 1 nodes pick every 2nd element as a sample.

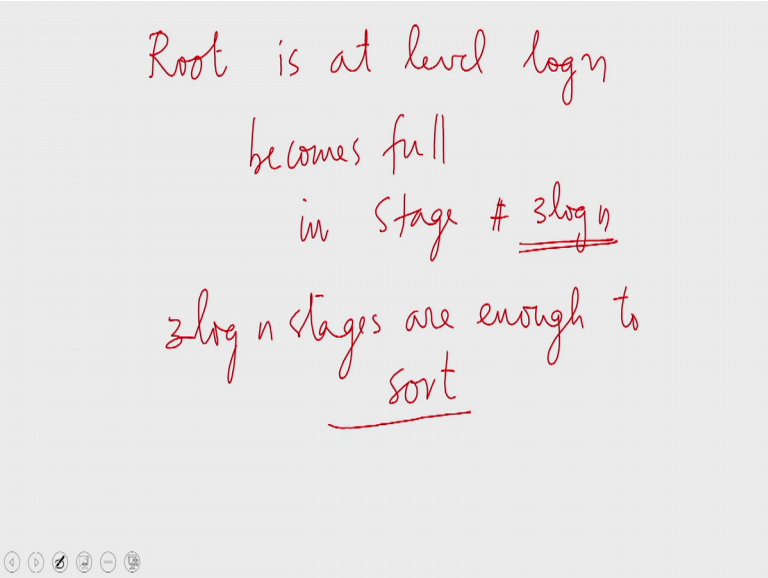
(Refer Slide Time: 48:33)



And in stage 3 k level k minus 1 nodes will pick every element as a sample. But mind you this is out of the caches that have already become full, which means all the nodes and all the elements that were placed at the leaves of a sub tree of a level k minus 1 node, has now actually reached the level k minus 1 node. Therefore, in stage 3 k when we pick all of them into the sample array, the sample arrays will become full. Now in the 2nd step of 3rd stage 3 k all these elements will be merged into the caches of their respective parents. And these parents are at level k.

So, what we find is that in stage 3 k, level k nodes get all the inputs in their sub trees in their caches in other words these nodes of full that establishes the induction. So, what we have shown is that at the end of the 3rd 3 k th stage every node at level k has got it is cache full.

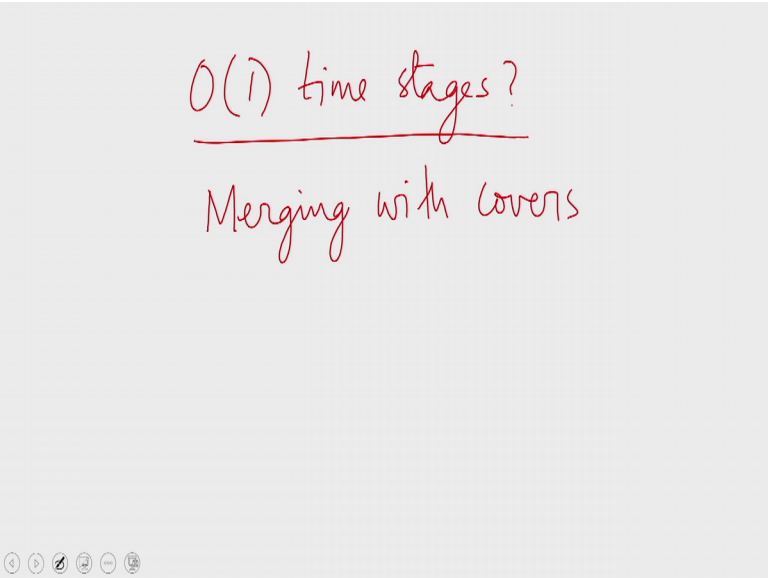
(Refer Slide Time: 49:21)



So in particular the root, which is at level log n becomes full. In stage number 3 log n which means if we run the algorithm for 3 log n stages, at the end of this stage the cache of the route will become full.

In other words the cache of the root will contain all the input elements in sorted order. So, at this point all we have to do is to output the cache of the root which means sorting will be accomplished in 3 log n stages. Now what we have to establish is that each stage can be executed in order 1 time.

(Refer Slide Time: 50:33)



But how do we achieve order 1 stages, to achieve order 1 stages we have to do what is called merging with covers. We say that an array is a cover of another one, in particular we said that an array c is a d cover of array b if between any two elements of c there are at most d elements in a. So, what we are going to show is that if 2 sorted arrays a and b have a cover c that is in particular let us say c is a d cover of both a and b and in addition let it be let it be that c is ranked into both a and b. Then if you have 1 process of a element of a b and c then, you will be able to merge a and b in order of d time.

In particular when I take d is d as a constant we will find that 2 arrays a and b can be merged in constant time, when we have a cover of a and b in particular a constant cover. So, in our case we will see that, we have 3 covers for all the merges that we want to perform. Where do we want to merge the perform merges when we merge the samples to form the new cache of the parent we will require covers. But then we shall show that covers are readily available.

We shall show that, the old cache at a node is a cover for the new samples of the children. In particular these are 3 covers. Therefore, the children's samples can be merged into the new cache of the parent using the old cache at the parent as a 3 cover. Therefore, the merge can be executed in order 1 time. Provided we have enough number of processors for every sample element at the children, and every cache element the old cache element at the parent.

If we can show that the total number of cache elements and the sample elements, over the active band of vertices, we say there are vertexes active if it is not full yet, and still emitting samples. So, if you consider the band of vertices which are still active. And we sum up the total sizes of the samples, and caches at these nodes if and we find that if that that is order n. Then we find that with order n processors, we will be able to merge the samples of the children into the caches at the parents in order of 1 time. That is the crux of the argument that establishes that a stage can be executed in order 1 time. We shall see more of more of the details in the next lecture. So, that is it from this lecture hope to see you in the next.