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

Lecture – 14
Analysis

Welcome to the 14th lecture of the MOOC on Parallel Algorithms. In the 13th lecture we were discussing an optimal algorithm for ranking linkless. Today we shall continue with the discussion we shall do the Analysis of the algorithm, but before we go into the analysis let us do a recapitulation of the algorithm. In this optimal algorithm for list ranking we are given a linked list.

The linked list is given in an array; so we assume that the size of the array is the same as the size of the linked list which is n So, the list has n nodes and the array also has the size of n. The physical order in which the nodes are given in the array is in no way related to the logical order. The first node of the list is accessible through a point ahead and then every node in the list has two pointers a predecessor pointer and a successor pointer.

The goal of the algorithm is to rank the list so that the node pointed to by head will be given a rank of one, then the successor node will be given a rank of two, it is successor will be given a rank of three and so on, until all the nodes in the list of ranked. We know that this algorithm has a sequential time complexity of order n. We have already seen an algorithm that uses pointer jumping for this problem that uses n processors to run in order of log n time on an EREWP ram that algorithm has a cost of order of n log n, but that is not an optimal algorithm.

Here we are attempted to design an optimal algorithm. So, in this algorithm the given array of size n is visualized as a row major representation of a two dimensional array of log n rows and n by log n columns. And then we have n by log n processors here and the model is EREWP ram of the n by log n processors available to us we depute one processor to each column. So, we have exactly as many columns as there are processors and the nodes of the array are given certain labels; initially every node is inactive.

And when we depute the processors to the columns we make them stationed on the 0th element of every column. That is the 0th row of the two dimensional array comes alive.

So, the notes belonging to this row are now labeled active. So, at the beginning of the algorithm the 0th row of the array has notes of label active and every other node in the list is labeled inactive; there are certain other labels too.

(Refer Slide Time: 03:08)

The other labels are ruler, subject and remove and the algorithm proceeds in this fashion. When we are given an array of a list of length n which is to be ranked; we find a successive; we find successive independent sets in this array and remove them. When we remove an independent set for each vertex of the independent set; the weight of the node that is being removed is added to the weight of the predecessor.

This is akin to queue of people waiting at a counter to buy tickets. When the queue is waiting let us say; one person in the queue wants to step out of the queue for a short period. So, the person who wants to buy k tickets can step out of the queue by instructing the person behind him to buy k additional tickets. So, the person who is behind might have been intending to buy l tickets for himself; so he will now by l plus k tickets, l for himself and k for the person ahead of him.

And the person who steps out of the queue will later come back and join the queue after the tickets have been distributed; then his predecessor would be holding l plus k tickets out of the this, k tickets would be taken over by him. So, this way what we are planning to do is to reduce the length of the list. So, the original list has a size of n we find an independent set in the list and remove it from the list.

So, every vertex in the independent set has charged the predecessor to find its own rank. Then the resultant list is recursively ranked let us say; then the vertices of the independent set can come and join back in the list. And then when they join back in the list all of them will acquire their ranks. So, this is the outline of the algorithm; given a list of length n we progressively remove independent sets from the list until the length of the list reduces to n by log n or less.

At this point the number of processes available is at least as many as the number of nodes in the list. So, you can depute one processor pardo; once you have one processor pardo we can use our previous algorithm that runs in order of log n time. And then once the reduced list is ranked the vertices that were spliced out can be reinserted back in the reverse order in which they were removed and when the entire list is repopulated back to it is original form it will be ranked.

So, this is the just of the algorithm in this the part that we have to describe is the first phase; where the vertices are removed to reduce the length of the list from log n to the length of the list from n to n by log n. The second part of the algorithm is nothing, but the previous algorithm which we have already seen. And third phase of the algorithm is an exact replica of the first phase where the vertices are inserted back in the reverse order in which they were removed.

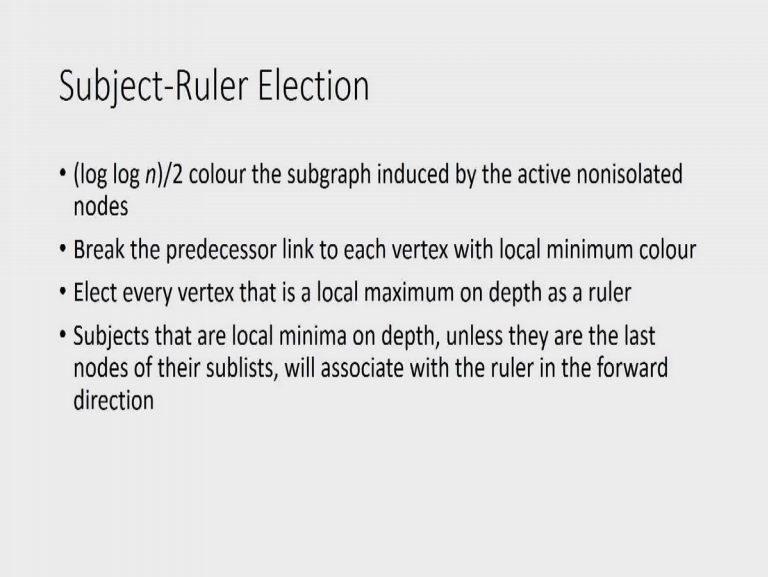
So, let us now consider the first phase which is the part that is to be described. So, the first phase consists of several iterations; in each iteration we have the processors doing in parallel this, we have one processor per column and the processor executes this code. The processor checks whether it is sitting on a ruler node; if it sits on a ruler node then it will remove a subject the ruler is bound to have a subject as we shall see.

So, it will remove a subject and once the subject is spliced out the node checks whether it was the last subject. If it was the last subject of this ruler the ruler will now turn active; that is the status of the node will now be turned active. In the second part of the iteration every active node will check this if the node is active, but isolated; in the sense that it does not have another active node as a neighbor then the node is spliced out.

On the other hand if it is not isolated it is active, but not isolated then it is part of a sub list of active notes. In this case it will take part in a subject ruler election. So, every node which is a part of sub list effective notes will take part in the subject ruler election. And

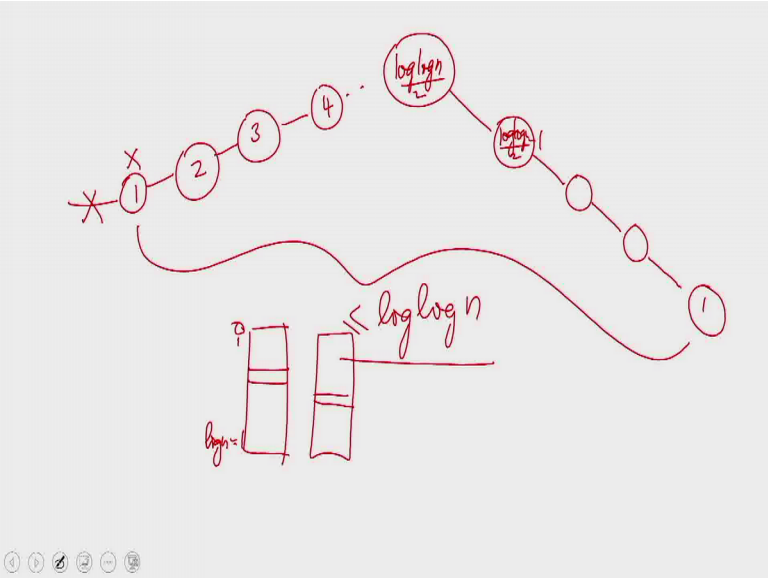
then once the subjects and rulers are elected the subjects will be left in charge of the ruler and the respective processors there will be advancing in their columns. So, this is just of the algorithm as we have seen in the previous class.

(Refer Slide Time: 07:37)



Now, the details that we have to describe are about the subject ruler election. To elect the subject subjects and the ruler out of a sub list of consecutive notes all of which are active, what we do is this; first we colour the list with double log n by 2 colours. This can be done by running the symmetry breaking algorithm for two or three steps. So, once the list is double log n by 2 colours we find the local minimum and then disconnect the predecessor link of each vertex with the local minimum colour.

(Refer Slide Time: 08:21)

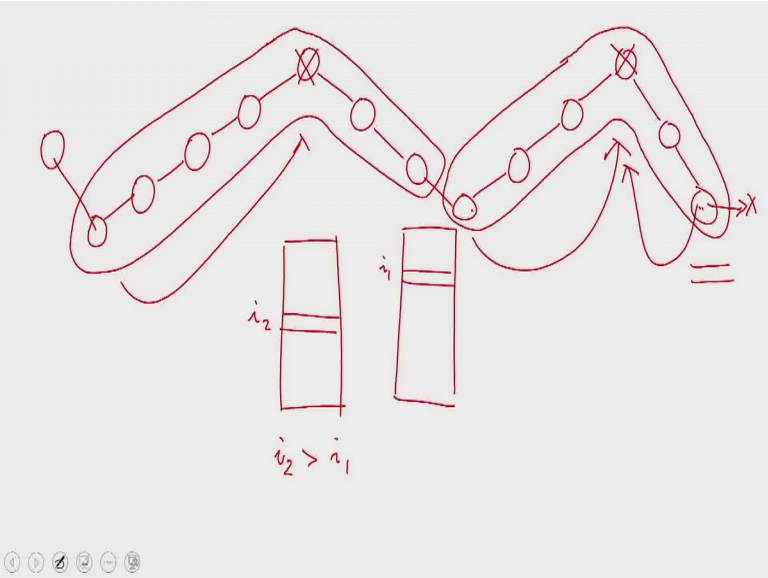


So, as we explained in the previous class when we have local minima identified as notes at which the predecessor link should be cut. The maximum length that we can have is of the sort. The maximum the longest stretch of vertices that we can have would be something like this. So, the number of nodes in this stretch would be less than or equal to double log n.

So, this ensures that the consecutive stretches of active nodes will get broken into sub lists of size double log n at the most. So, this is the first step of the subject ruler election. We break the sub graph induced by the active non isolated nodes into sub lists of size double log n at the most. And then for each vertex we also know the depth of the node. Each node belongs to a separate column these are all active nodes mind you; so, each of them belong to a separate column.

So, in these arrays we would have advanced to various depths. So, we take the depth; so this is at depth 0, this is a depth 2 and so on the depth 1 and so on. And the maximum depth in a column is log n minus 1. So, the various active nodes could be at various depths and then these depths have absolutely no relation to the colours. Therefore, when we find the local maxima on the basis of depth this will have absolutely no relation to the local minima that we found earlier.

(Refer Slide Time: 10:24)



So, we are taking a look at these sub lists once again and then finding the local maximum on the basis of depths. The idea is that a node that has a higher depth. For example; a node that is a higher depth has advanced more in it is column than a node that has a lower depth.

So, this has a lower depth i 1 and this has a higher depth i 2; i 2 is greater than i 1 means; the no depth position i 2 has advanced more in it is column in the sense that it has done more work in it is column. We would like a node that has advanced more in its column to be elected the ruler rather than the subject; so that is what we do here. We consider the depths of the various notes and then every elect every vertex that is a local maximum on depth as the ruler.

So, the local maxima would be chosen as the rulers. Now this is not on the basis of colours, but on the basis of depths. So, in this example these two nodes would be chosen as the local maximum and then the rulers will have subjects assigned to them the subjects of this ruler would be all these vertices.

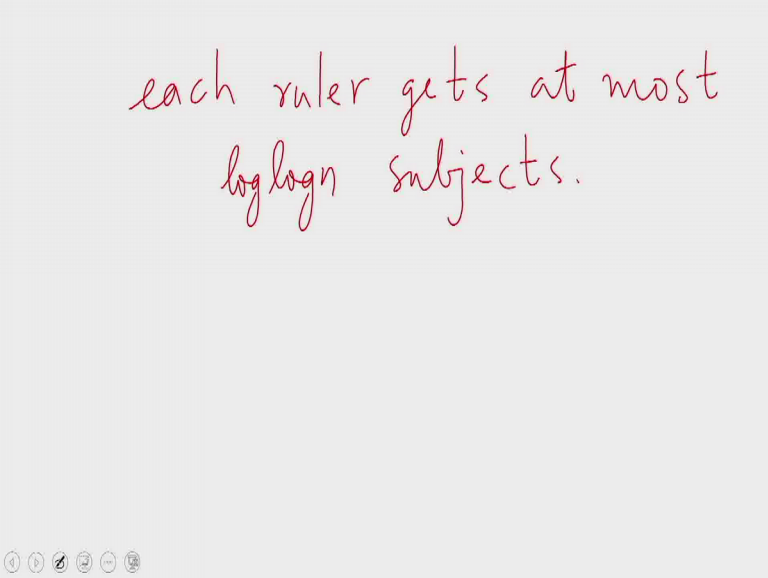
Similarly the subjects of the second ruler would be these. That is what is specified in the last step here, subjects that are local minima on depth unless they are the last notes of their sub list will associate with the ruler and the forward direction that is in this diagram in the right direction. So, this node which happens to be a local minimum will be

associated with this ruler, this node which is also a local minimum is associated with this ruler.

This node is a local minimum; however, since it does not have a ruler in the forward direction that is the forward link has been cut. And therefore, this is the last node of the sub list therefore, this node also will be made a subject of this ruler. So, in every ruler will be assigned some number of subjects and the subjects could be either backwards or forwards.

So, while going backwards all the nodes up to the nearest local minimum will be its subjects. And going forward all the nodes up to, but not including the next local minimum will be its subjects except in this case; where the local minimum going forward happens to be a node without a successor in the sub list. So, in this way we find rulers and subjects.

(Refer Slide Time: 13:29)



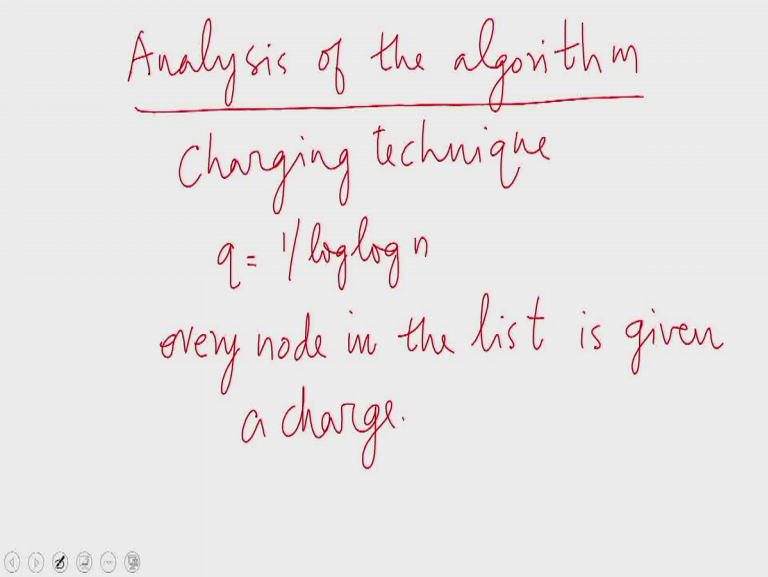
So, this ensures that each subject or rather each ruler gets at most log of log n subjects. So, when a sub list nodes are divided into rulers and subjects. The subject nodes are left in charge of the rulers that is the processes sitting on these rulers are now also responsible for pruning the subjects from the list.

So, they can remove the subjects one at a time. And only after removing these subjects will those processes get to go further in their own columns. But then the processes of the

columns to which the subjects belonged will now leave the subjects and move forward in their respective columns. So, this ensures the sort of work balance because in every set the node with the largest depth has been chosen as the ruler. So, this node has already done a substantial amount of work in its own column.

Therefore, it can afford to spend some time pruning nodes belonging to some of the other columns. In these columns the processes have not advanced much; that is why they have been elected subjects. So, those processes now get to advanced faster. So, this in that sense ensures that the processes are progressing at a balanced speed. When we do the analysis we find that this will indeed ensure that the length of the list will be reduced from n to n by log n in order of log n time using n by log n processes; so, that is the gist of the algorithm.

(Refer Slide Time: 15:36)



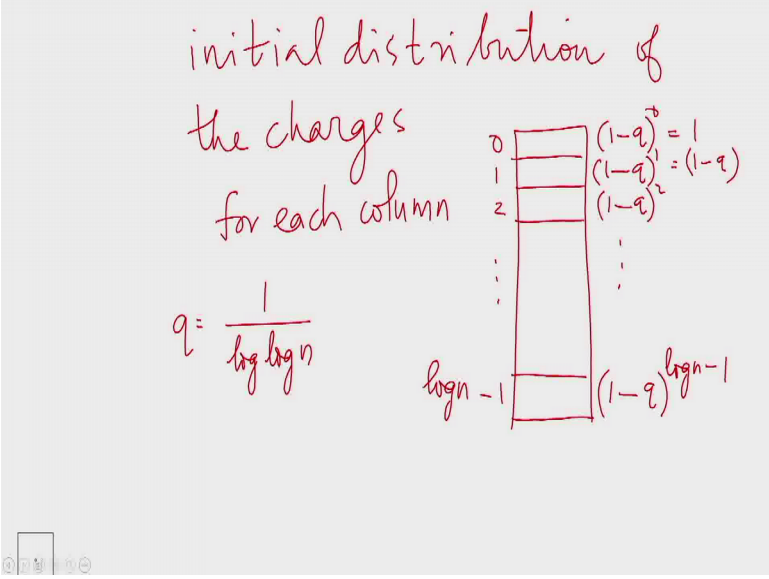
Now, it remains to do the analysis of the algorithm. To do the analysis of the algorithm; we do a charging technique. Let us define q as the quantity 1 divided by double log. And then what we do is this; every node in the list is given a charge. The charges are distributed over the notes merely for the purpose of analysis these charges have nothing to do with the algorithms.

This is a gain that we set up so that at the end of the gain we will have the analysis of the algorithm done. So, the gain that we set up is in this fashion we distribute charges in a certain fashion over the entities of the algorithm. And then when the algorithm performs

various operations we would remove some charges from the system or redistribute the charges.

So, that we can ensure that the with every iteration the charges reduced by a certain fraction. This will ensure this will enable us to ensure that after our log n steps the charges left in the system is in such a way that this ensures the number of nodes remaining is at most n by log n. So, the charging technique enables us to give get this upper bound. So, that now let me describe the distribution of the charges.

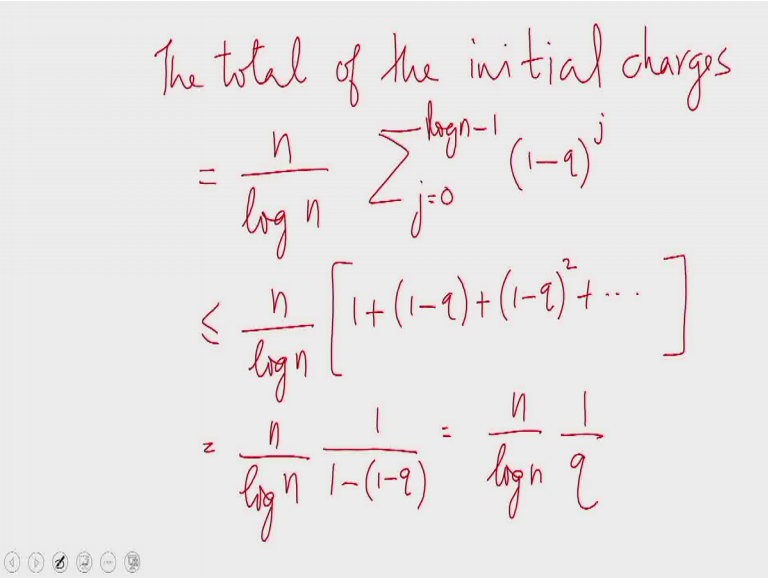
(Refer Slide Time: 17:36)



The initial distribution of the charges is as follows; for each column where the rows are numbered from 0 to log n minus 1, the charges are distributed in this fashion. The 0th element of the column gets a charge of; 1 minus q power 0 which is equal to 1, so the 0th node is going to get a charge of 1 unit. The first element is going to get a charge of 1 minus q power 1 which is 1 minus q.

And the second element is going to get a charge of 1 minus q power 2 and so on. The last node is going to get a charge of 1 minus q power log n minus 1. Here you will recall q is 1 divided by log of log n. So, the charges in each column are distributed in this fashion column after column. Now we can estimate the total amount of charges that we place in the system.

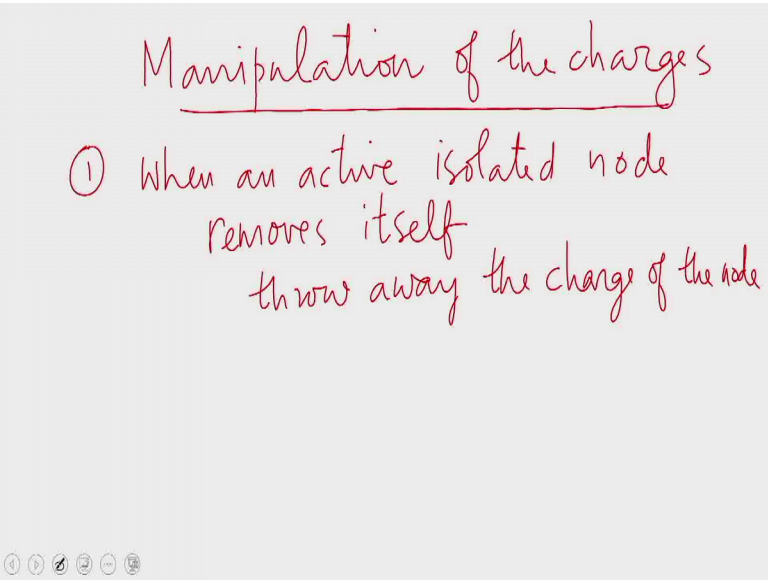
(Refer Slide Time: 19:15)



The total of the initial charges is the number of columns multiplied by the sum of the charges in each column; which you can see is at most n by log n times 1 plus 1 minus q plus 1 minus q squared plus etcetera. There are log n terms in the summation, but let us extend the sum to infinity.

Therefore, we can say this is nothing, but n by log n times 1 divided by 1 minus 1 by q 1 minus; 1 minus q; which is n by log n times 1 by q. So, we find that the total initial charges that we place on the system is at most n by log n times 1 by q.

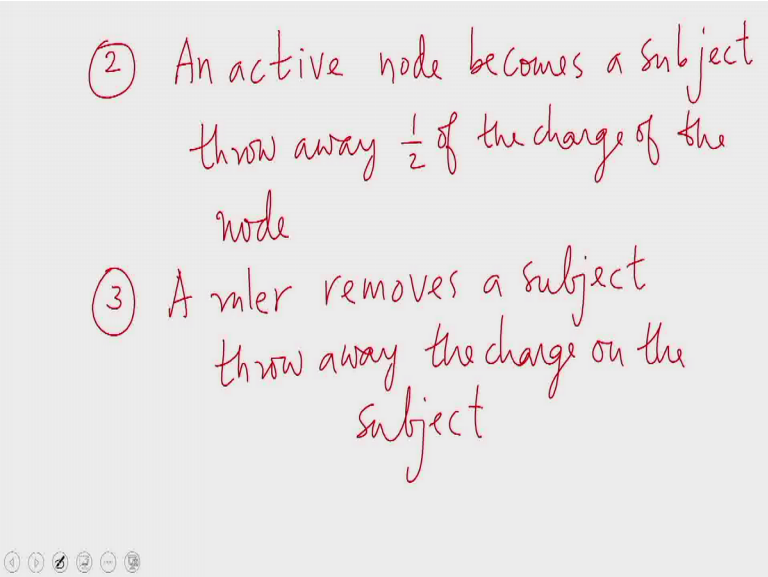
(Refer Slide Time: 20:54)



And then we are going to manipulate the charges in a certain way. When do these charges and their manipulations are all external to the algorithm that is standing outside the algorithm we manipulate these charges as the algorithm performs various operations.

So, what we do is this when an active and isolated node removes itself; we throw away the charge of the node. When an active isolated node removes itself the charge that is sitting on that node will be thrown away from the system. So, this is one kind of charge manipulation that we perform.

(Refer Slide Time: 22:09)

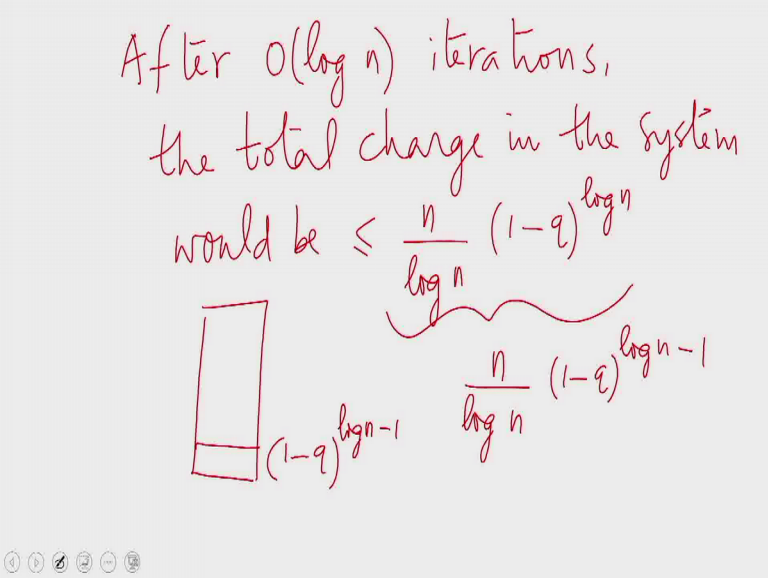


The second kind of charge manipulation is; done when an active node becomes a subject when an active node becomes a subject. Let us say we throw away half of it is charge, we throw away half of the charge of the node this is the second kind of charge removal that we have. And then there is one more kind a ruler removes a subject at this point we throw away the charge on the subject.

Remember when the active node becomes a subject we throw away half of the charge on the node. Then later when this subject that is now the subject of some ruler is in turn removed by that ruler the remaining half of the charges also thrown away. So, that way the charge on a node will be totally removed when it first becomes a subject and then later on gets removed. So, these are the three ways in which we can remove charges from the system.

The first is when an active isolated nor removes itself at this point we throw away the charge on that node entirely at one go. But if an active node becomes a subject and then later on gets removed by a ruler the charge on that node will be removed in two steps. First when it becomes a subject half of the charge will be thrown away, later when it becomes when it is removed the remaining half of the charge will also be thrown away. So, this is how we manipulate the charges that we place on the list initially.

(Refer Slide Time: 24:42)



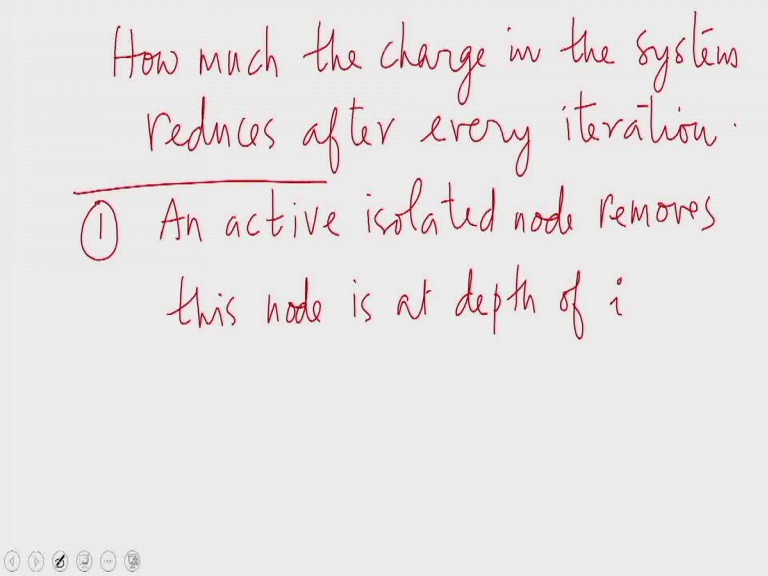
Now, what we want to claim is that; if we remove charges in this fashion after order of log n iterations. The total charge in the system would be less than or equal to n by log n times 1 minus q power log n. Now this would ensure that at most one node is left in each q. This is because in each q the lightest node which happens to be the last node was given an initial weight of 1 minus q power log n minus 1.

So, even if the last node of every q is left the total weight in the list can be no less than n by log n times 1 minus q power log n minus 1. But here we find that the total weight in the list is even less than that. This ensures that the total number of nodes in the list is less than n by log n. That is we do not have even one node per cube even the lightest node per cube cannot be retained and the weight still the weight cannot be as low as n by log n times 1 minus q power log n.

So, once we ensure this we would have guaranteed that the number of nodes remaining in the list is at most n by log n. At this point we will have as many processors as there are

nodes in the list. And therefore, we will be able to enter the second phase of the algorithm. So, this is what now remains to be shown that after order log in iterations the total charge in the system would be at most n by log n times 1 minus q times the q power log n.

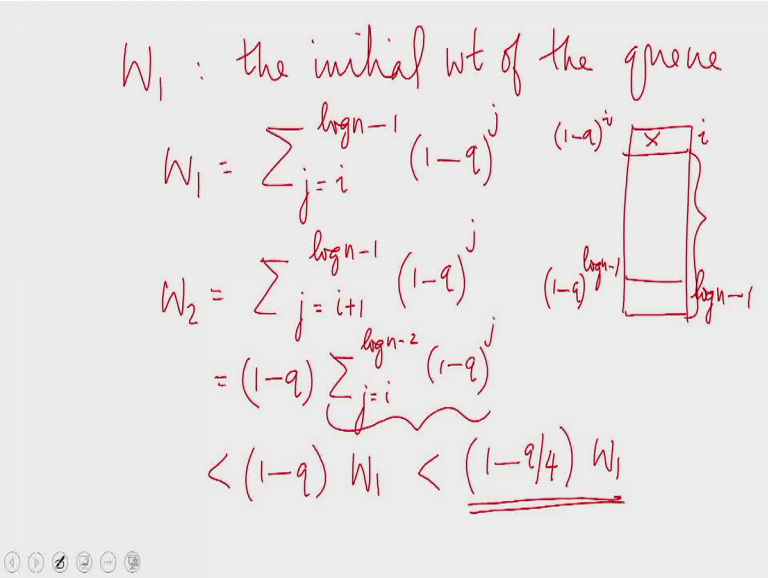
(Refer Slide Time: 27:03)



To prove this we want to establish how much the charge in the system reduces after every iteration; this is what we now want to establish; to establish this let us consider the various cases the first case was where an active isolated note removes itself.

When this happens, let us see what happens to the weight distribution in the corresponding system. So, let us consider an active isolated node let us say this node is at a depth of i.

(Refer Slide Time: 28:23)



In that case W 1 the initial weight of the queue to which this node belongs. Just before this removal happens to be some of j varying from i to log n minus 1 of 1 minus q power j. That is we are looking at a q in which we are at a depth of i and the lowest node is at depth of log n minus 1, all the previous nodes have been removed.

This is the node that is being removed now; so, this node has a weight of 1 minus q power i the last node has a weight of 1 minus q power log n minus 1. Therefore, the total initial weight in the queue just prior to this removal is W 1 which is sigma j varying from i to log n minus 1 of 1 minus q power j. Let us consider the weight W 2 that is there in this queue after this node has been removed.

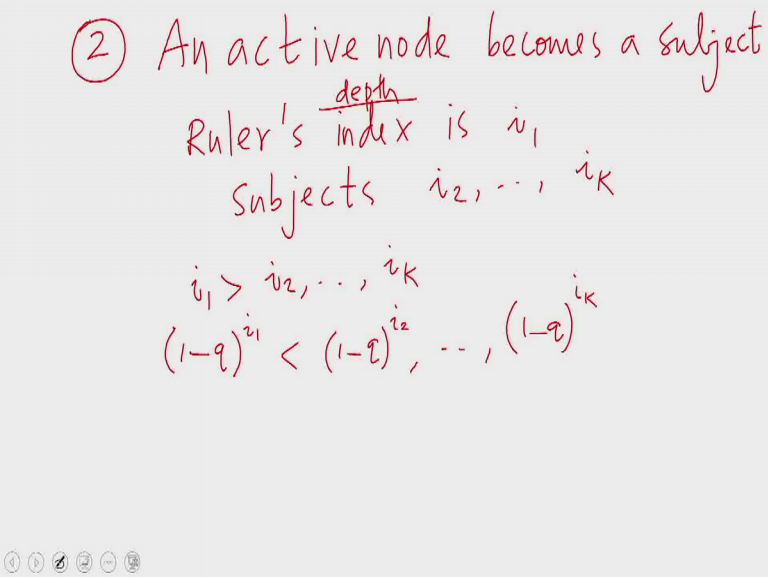
So, we can clearly see that this is nothing, but j varying from i plus 1 to log n minus 1 of 1 minus q power j which is just the sum of these nodes from i plus 1 to log n minus 1, but this we can write as the summation. But we find that this is this quantity is less than W 1.

Because this summation is from i 2 log n minus 2 whereas, in W 1 the summation is from i 2 log n minus 1; therefore, this quantity is less than W 1. So, we can say this is nothing, but less than 1 minus q power the times W 1. So, W 2 is at most 1 minus q times W 1; which is in turn less than 1 minus q by 4 times W 1.

So, what we find is that the weight and the q after the operation is at most 1 minus q by 4 times the weight in the cube before the operation. So, this part of the operation ensures

that the concerned weight reduces by a factor of 1 minus q by 4. So, this is what happens in the first kind of removals; that is when an active isolated node removes itself. The weight of the corresponding q reduces by a factor of 1 minus q by 4.

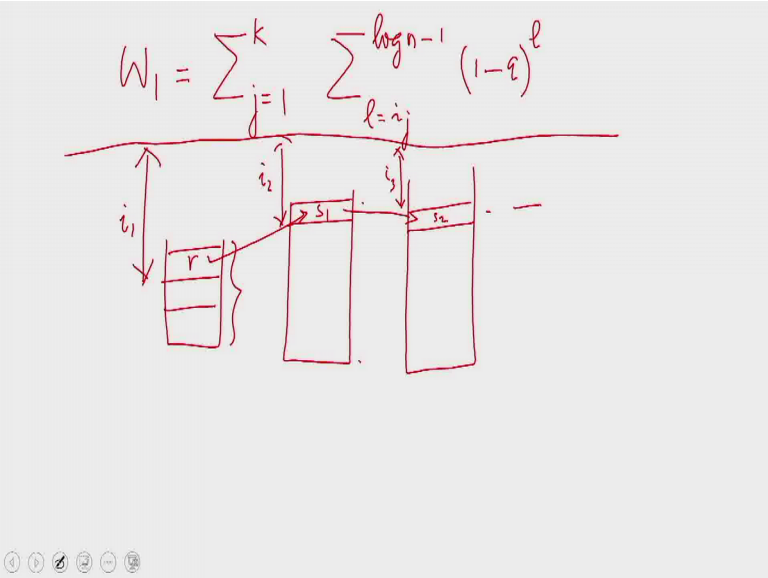
(Refer Slide Time: 31:43)



Now, let us kind consider the second kind of operations this is where an active node becomes a subject. We agreed that when this happens half of the weight of the subject is thrown away. So, let us say the rulers index is i 1 the subjects are of indices i 2 through i k.

So, initially since of course, the rulers index our depth is i 1 we know that i 1 is greater than i 2 through i k. Because we are picking the local maxima as the rulers the ruler's depth would be greater than the depths of the subjects. Therefore, i 1 is greater than i 2 though i k which means; 1 minus q power i 1 is less than 1 minus q power i 2 1 minus q power i k that is the ruler has the least weight of all these notes that we are considering.

(Refer Slide Time: 33:19)

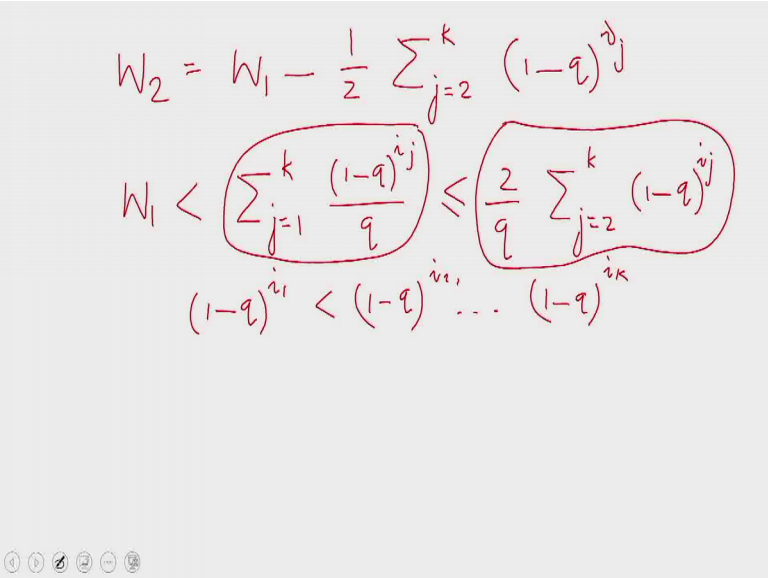


So, in this case let me define W 1 as the total weight of the system before the operation is performed. So, here we have k is used to consider the queues corresponding to the ruler as well as the subjects. So, for j varying from 1 to k, the total weight of these q's would be. That is because in the ruler array this is where the ruler is at this depth this is depth i 1, the subjects would be at lesser depths this would be depth i 2 let us say.

This is the first subject then the second subject would be at a depth of i 3 and so on. So, we are considering a ruler with its subjects of course, the subjects could go backwards as well, but whether it whether they go forwards or backwards their depth would all be greater than the depths of the ruler. Therefore, the depths would be less; therefore, the weights of the subjects would all be greater than the weights of the ruler so the initial weight of the system would be this much.

For each of the queues I have to add up the total weights. The highest weight here is 1 minus q power i 1 going all the way to 1 minus q power log n minus 1 for the this q the highest weight is 1 minus q power i 2. And the lowest weight is 1 minus q power log n minus 1. So, the sums are taken for every single q that is what the quantity W 1 is. So, W 1 is the total weight in the system before the operation.

(Refer Slide Time: 35:42)



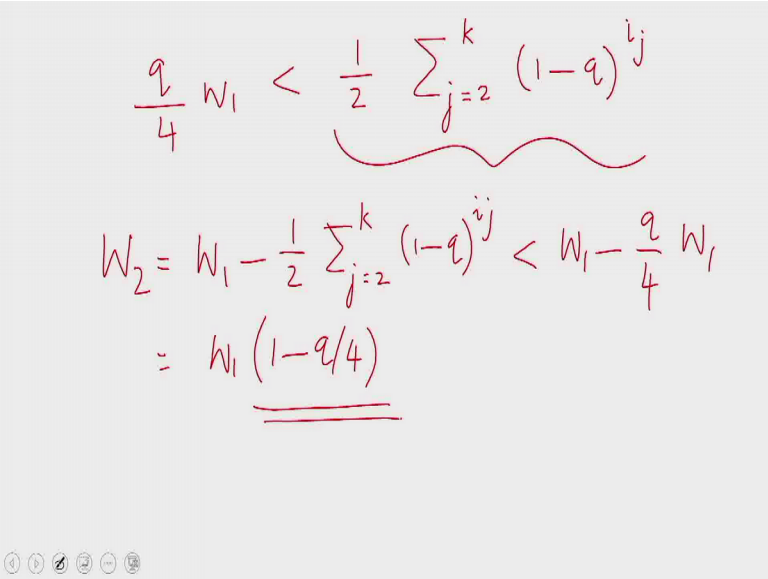
The weight in the system after the operations is going to be W 1 minus half of 1 minus q power i j. This is because when a node becomes a subject half of it is weight is thrown away. So, the subjects in this had weights of 1 minus q power i 2, 1 minus q power i 3 etcetera going up to 1 minus q power i k. Half of all these weights have been thrown away which means the net weight in the system would be that is in all these q's put together would be W 1 minus half of the sum varying from j equal to 2 to k of 1 minus q power i j.

So, this is what W 2 is, but then summing over the various q's involved in the system we find that W 1 is less than sigma j varying from 1 to k of 1 minus q power i j divided by q. That is within each q we are summing up all the way to the infinity. That is starting from 1 minus q power i j we keep summing up to the infinity. Then we find that W 1 is at most this much which we can say is less than or equal to 2 by q times sigma j varying from 2 to k 1 minus q power i j.

Now, why would this be, because of the various sums 1 minus q power i 1 is less than 1 minus q power i 2 etcetera. Therefore, out of some of all these that is out of the some if I remove the first term which is 1 minus q power i 1 by q, then the total sum will not even be halved. Therefore, if I remove the first term and then double the remaining then the net would. In fact, be greater than or equal to the original quantity. Therefore, we have

this inequality this circle part is less than or equal to 2 by q times the sum of the second term through the kth term.

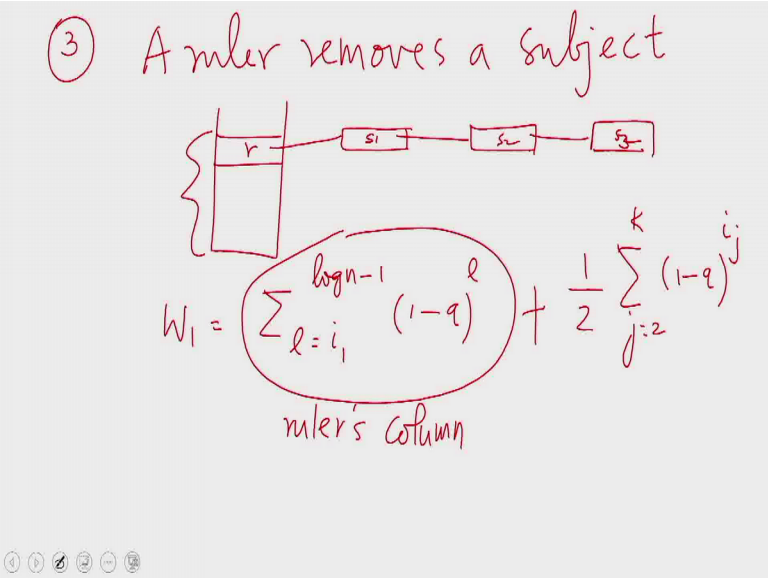
(Refer Slide Time: 38:37)



So, we have that W 1 is less than or equal to this quantity which tells us that; q by 4 times W 1 is less than half of sigma j varying from 2 to k of 1 minus q power i j. But this exactly is the weight that we remove from the system by removing that weight we get W 2. So, W 2 is W 1 minus this quantity half of sigma j varying from 1 to k 2 to k of 1 minus q power i j which is; less than W 1 minus q by 4 W 1.

Since this is negative here the inequality turns therefore, we have W 2 is 1 minus q by 4 times W 1. So, what this establishes is that in the second case also the total weight involved in the operation reduces by a factor of 1 minus q by 4. This is exactly identical to the term that we got in the first case. In the first case we found that the total weight involved in the system reduces by a factor of 1 minus q by 4 and similarly here too we obtain a similar factor.

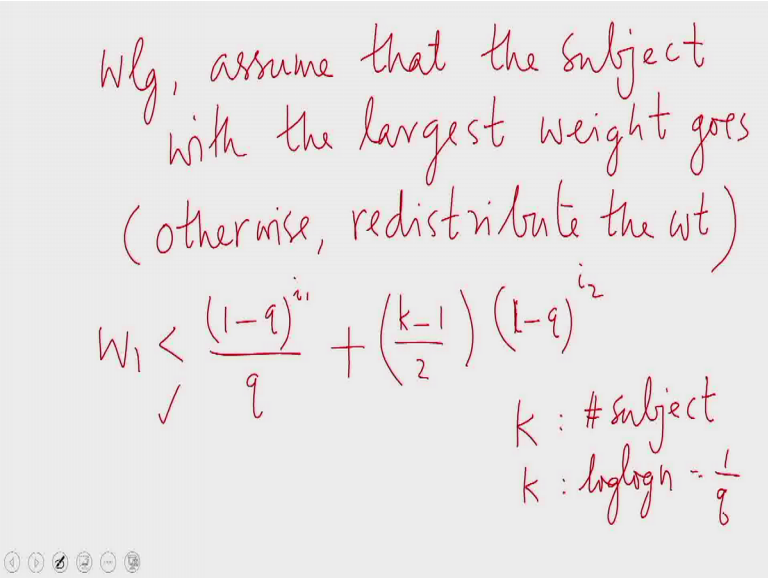
(Refer Slide Time: 40:17)



Now, coming to the third case a ruler removes a subject. So, here we have the column corresponding to the ruler, the process of sitting on the ruler is in charge of this entire column. In addition to that this ruler is also responsible for certain subjects. But this processor is not responsible for the columns of those subjects, but only to those subjects themselves. So, this is the initial system that we have the entire column corresponding to the ruler plus the subject notes.

Therefore, the total weight in this system initially is this is the total weight corresponding to the ruler's column plus we also have to consider the total weight of the subjects. So, this is the total weight of the system initially, the total weight of the rulers column plus the weight of these subjects. All these subjects have lost half their weights so all that they have now are half their original weights. So, here we assume that the ruler removes a subject.

(Refer Slide Time: 42:28)



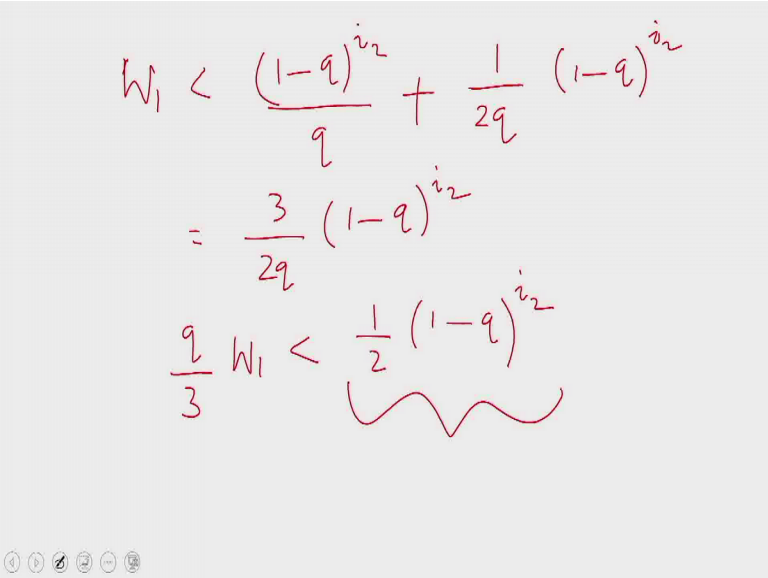
So, let us without loss of generality assume that the subject with the largest weight is removed. We can make this assumption without loss of generality because if this actually does not happen we redistribute the weights. That is if you find that the algorithm is actually removing a subject with a different weight we would swap the weights.

That is a largest weight of any subject will now be placed on this element which is being removed and it is weight will be placed on the node of which we remove the weight. Therefore, we can still claim that the subject with the largest weight is being removed. Now, this ensures that W 1 is less than 1 minus q power i 1 by q, this is a simplification of the first term the first term is the summation from I want to log n minus 1 of 1 minus q power l.

If I take the summation all the way to infinity I will have 1 minus q power i 1 divided by q, so the first term reduces to this. In the second term I have the summation from 2 to k of 1 minus q power i j. So, this is 1 minus q power i to 1 minus q power i 3 etcetera summed up and then divided by 2 there are k minus 1 terms here.

But since we have assumed that 1 minus q power i 2 which happens it happens to be the largest weight of all the subjects is the one which is being removed I can assume that this is at most k minus 1 by 2 times, 1 minus q power i 2 1 minus q power i 2 we assume is the largest of all these terms. So, I am replacing this with a larger sum which is k minus 1 by 2 into 1 minus q power i 2 this inequality holds good.

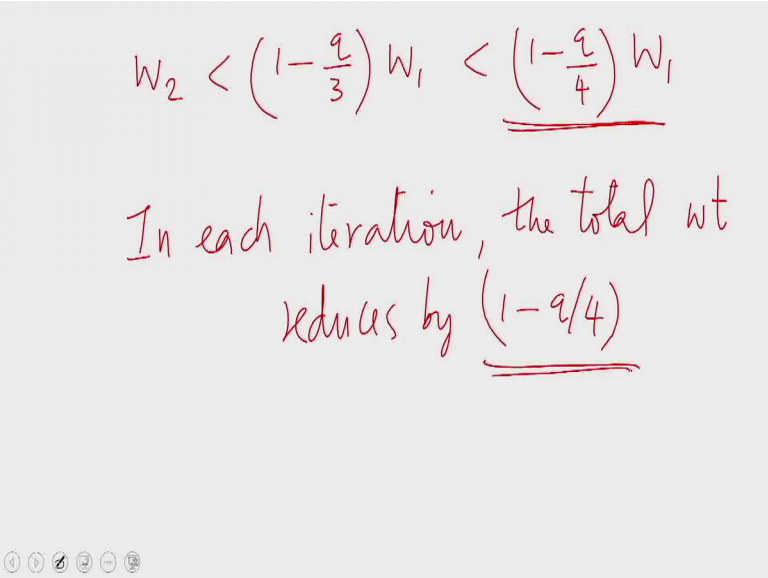
(Refer Slide Time: 45:08)



So, now what we have established is that W 1 is at most this quantity which further simplifies to this W 1 is less than 1 minus q power i 2 by q. Here we are replacing the first term here with 1 minus q power i 2 by q the first term is here is 1 minus q power i 1 by q. But since i 1 is greater than i 2 1 minus q power i 2 by q is larger than this; so, I am replacing the first term with a larger quantity. And in the second term if we replace k minus 1 by 2 with k by 2 and then observe that k here is the number of subjects that the ruler has.

And that k is utmost double log n which is 1 by q I can simplify this term to 2 by q times pardon me; 1 by 2 q times 1 minus q power i 2 k minus 1 by 2 is less than k by 2 when you substitute k equal to 1 by q this term becomes 1 by 2 q times 1 minus q power i 2 which is 3 by 2 q times 1 minus q power i 2 which means rearranging we find that q by 3 times W 1 is less than half times 1 minus q power i 2. This is the weight of the subject that is being removed this is the weight of the heaviest subject this is what is being removed from the system.

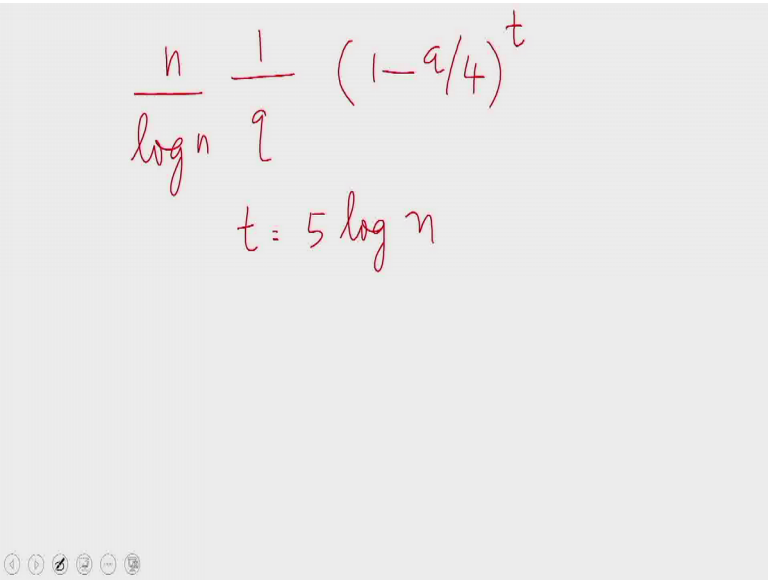
(Refer Slide Time: 47:18)



Which means the new weight W 2 is less than 1 minus q by 3 times W 1 which is further less than 1 minus q by 4 times W 1. So, in this case also we establish that W 2 is at most 1 minus q by 4 times W 1. Therefore, combining all the three cases we can claim that in each iteration.

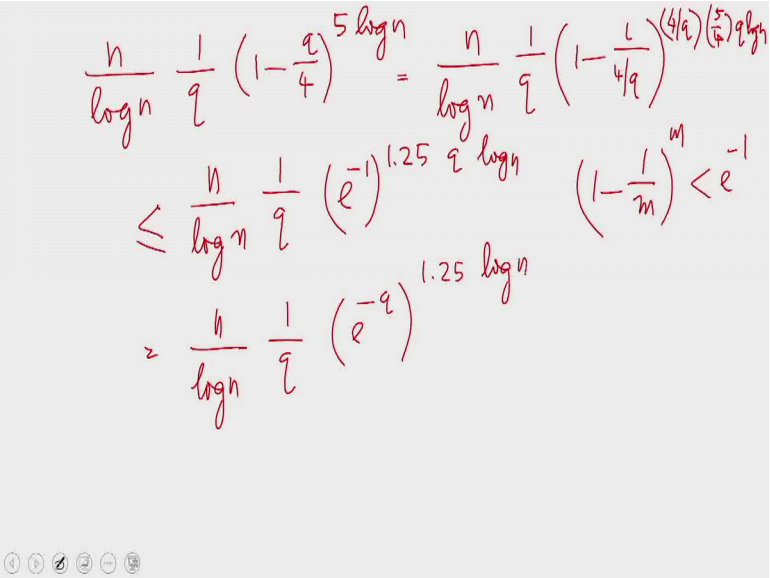
The total weight in the system reduces by 1 minus q by 4. This is because every vertex should belong to one of the three systems in every single iteration. Therefore, in every single iteration the weight reduces by 4 minus q by 4.

(Refer Slide Time: 48:23)



Now, we know that we started with an initial weight of n by log n times 1 by q. If we continue these for 3 steps after t steps we will have a total weight of 1 minus q by 4 power t. Let us take t equal to 5 log n we claim that after t equal to 5 log n steps the total weight in the system will be less than n by log n times 1 minus q power log n. Once we establish this we have achieved what we wanted.

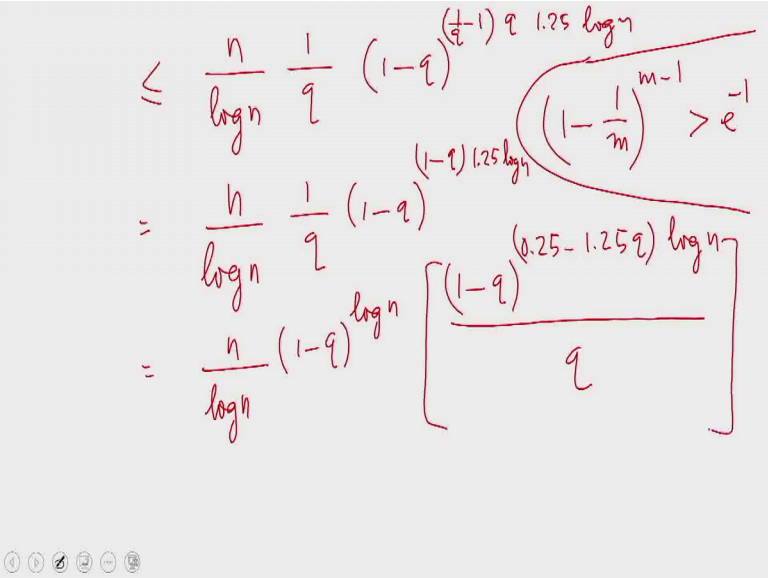
(Refer Slide Time: 49:01)



So, let us evaluate this quantity n by log n times 1 by q times 1 minus q by 4 power 5 log n while remembering that q is 1 by double log n. So, this is nothing, but n by log n times 1 by q times 1 minus 1 by 4 by q power 4 by q times 5 by 4 q log n which we know is less than or equal to n by log n times.

1 by q times e power minus 1 the whole power 1.25 q log n. That is because 1 minus 1 by m the whole power m is less than e power minus 1; which is n by log n times 1 by q times e power minus q power 1.25 log n.

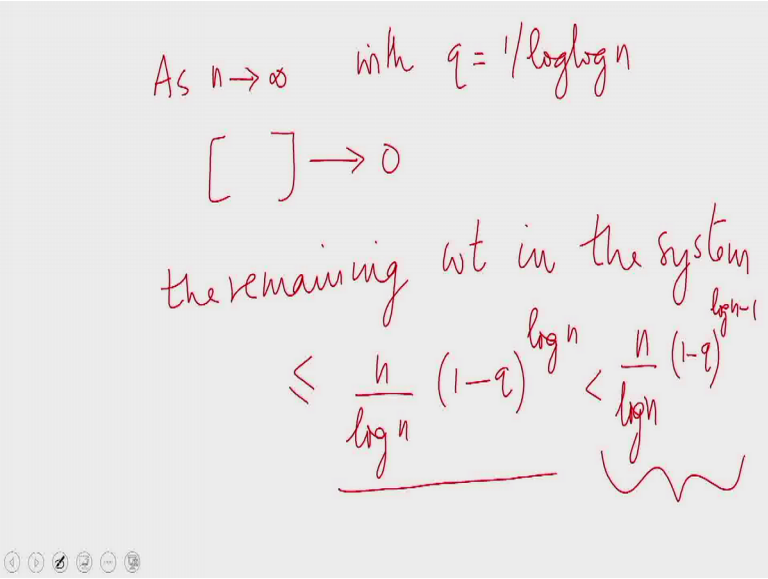
(Refer Slide Time: 50:29)



Which can be written as n by log n times 1 by q times 1 minus q power 1 by q minus 1 into q into 1.25 log n. That is because 1 minus 1 by m power m minus 1; this is greater than e power minus 1.

Therefore, this simplifies to n by log n times 1 by q times 1 minus q power 1 minus q 1.25 log n. Or in other words n by log n times 1 minus q power log n into 1 minus q power 0.25 minus 1.25 q times log n right.

(Refer Slide Time: 52:05)

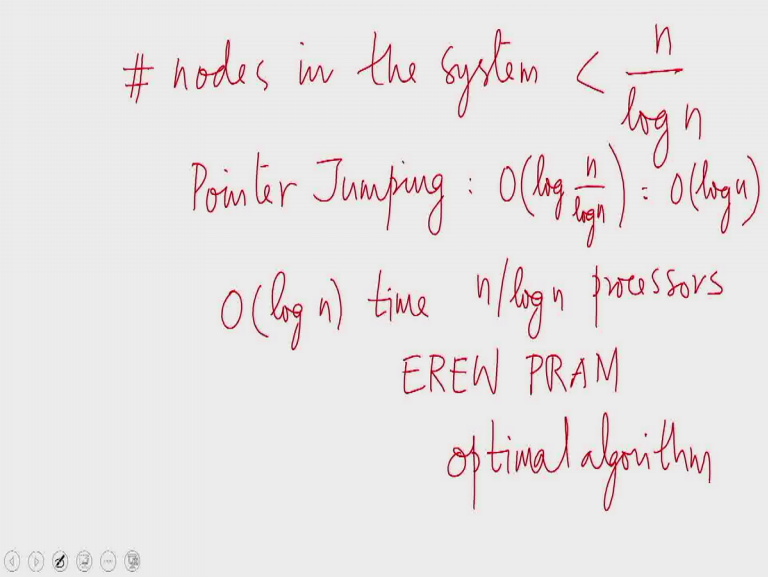


But, as n tends to infinity with q equal to 1 divided by double log n; you can show that the quantity in the square bracket goes to 0. So, this quantity goes to 0 as n tends to infinity; which means for sufficiently large n the remaining weight in the system when the quantity within the square bracket goes below 1.

Since it is a rapidly reducing function you find that for sufficiently large n the remaining weight in the system becomes less than or equal to n by log n times 1 minus q power log n; which is less than the total weight in the system that would have been the had we kept exactly 1 item per column the heavy is the lightest weight in the column.

So, even if every column retained the lightest element the wage in the system would have been only would have in this much. But here we have a weight which is even less than that.

(Refer Slide Time: 53:39)



That means the number of nodes in the system is now less than n by log n and n by log n less the number of processes we have therefore, now with n by log n processes we will be able to execute this algorithm execute the previous algorithm. So, now, the number of nodes have reduced to n by log n with n by log n processors we run the pointer jumping algorithm which is the second phase of the algorithm.

The pointer jumping algorithm forms the second phase of our algorithm here this will run in order of log of n by log n time which happens to be order of log i. And then the

third phase of the algorithm is an exact replica of the first phase except that the nodes are scheduled in the reverse order there is a nodes are put back in the list in the reverse order.

Therefore, the third phase also runs in order of n by log n runs in order of log n time using n by log n processes. Therefore, finally, the algorithm runs in order of log n time within by log n processes to complete the list ranking. So, once again to do a recap; the algorithm involves removing a series of independent sets from the list. The number of such independent sets to be removed as order of log n 5 log n at the most as we have shown.

Once all these independent sets have been removed the size of the list will be at most n by log n on this reduced list we will run the part jumping algorithm which runs in order of log n time. Since now we have the number of nodes equal to the number of processors. After this the third phase starts in which the vertices are inserted back into the list in the reverse order in which they were removed when the list is completely populated we have the original list with the rank for every vertex computed.

The net time taken by the algorithm is order of log n and the number of process used is n by log n and the model that we have used is EREW P. Since the cost of the algorithm is order of n this is an optimal algorithm. In the next couple of lectures we shall see several applications of this ranking problem in all of which we will be using this algorithm as a subroutine. So, that is it from this lecture hope to see you in the next lecture.

Thank you.