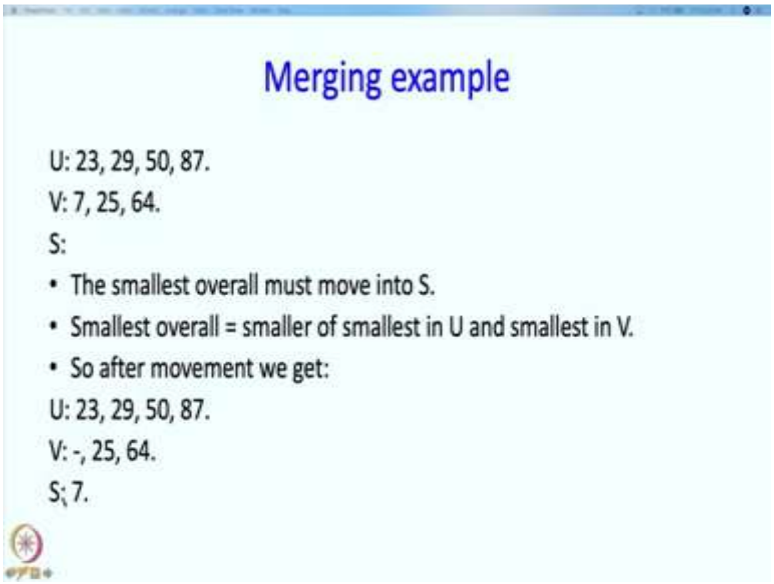
An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 18 Part- 4
Arrays and recursion
Merge function

Welcome back, in the previous segment we discussed the outline of merge sort and we also discussed the code for it.

(Refer Slide Time: 0:21)

In this segment we are going to discuss the merge function. So let us do an example first.

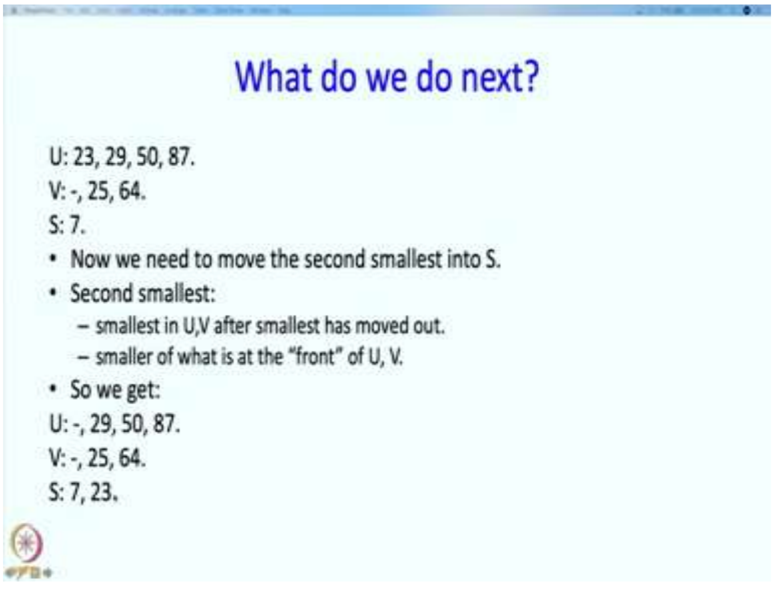
(Refer Slide Time: 0:32)



So let us take the U sequence as 23, 29, 50, 87, so this is the sorted the result of sorting. That U that we had got by splitting our original sequence and V is the other part again sorted. So how do we merge this? So we have to produce the sequence S. And if you think about it what do we want in S? Well the first element should be the overall smallest. Now how do we get the overall smallest? We need to look at both these sequences in great detail. Well no. Because they are sorted it is the smaller of the smallest which is present in the first position and the smallest over here which is also present in the first position.

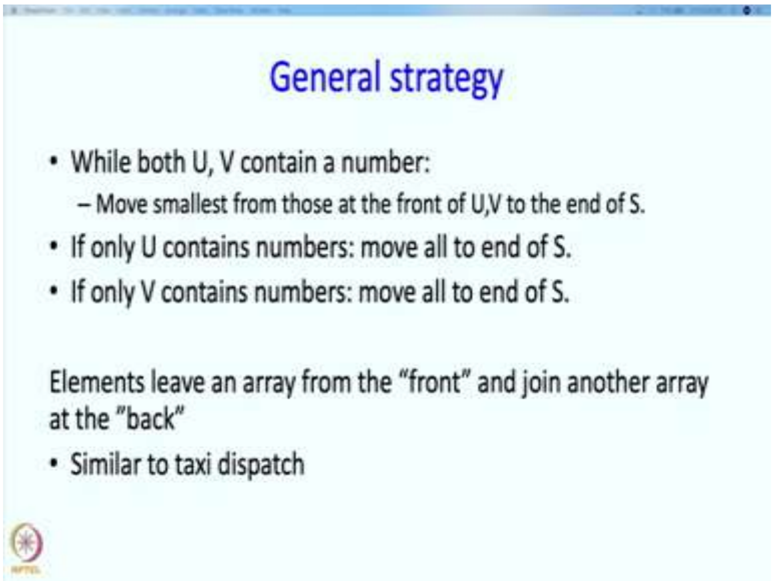
So if I want the smallest in these two sequences I just need to look at these two positions. So if I look at these two positions then I know that the smaller is 7, so that 7 must come down over here. So what we get after this is this picture, so we started over here we looked at the two elements which are (respects) respectively smaller in U and V. And we do not have to look at the rest rest of the elements to decide which is the smallest in this entire entire set. So we picked this 7 which is the smallest over here and we move it to S.

(Refer Slide Time: 2:07)



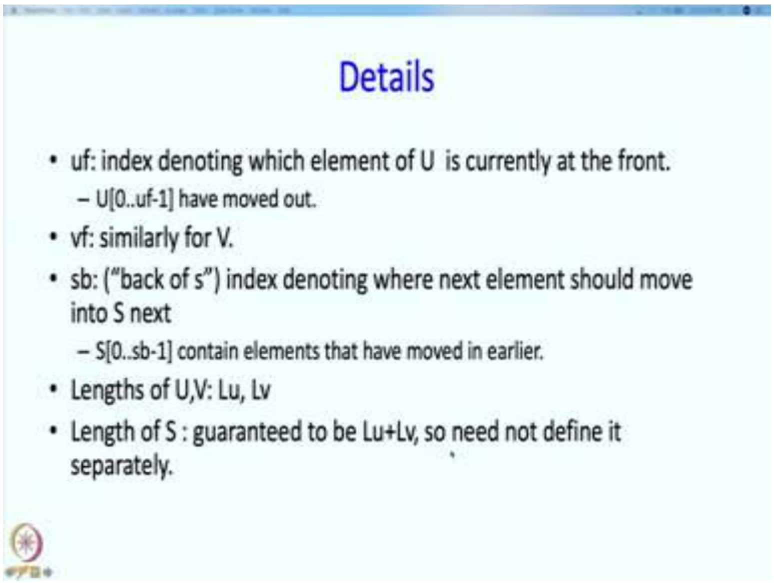
And this we are going to continue, so what do we do next? So this is our situation, we again want the smallest of these two. So we really want the second smallest, in what was originally U and V. But now that S has moved out we can say look what is the current situation of U and V and we want the smallest amongst those. So how do we get that? So the second smallest is the smallest in U, V after the smallest has moved out. So which is exactly this position over here? And to get that we simply have to ask what is the smaller among what is at the front of U and V? So the front of U is this so here there is 23 the front of V is now this because this element has moved out. So we have to now pick the smaller of these two elements and move that at the end of S. So if we do that we get this, so the smaller has moved to the end of S or the back of S and the fronts, the elements which were originally at difference have gone away and the fronts have sort of advanced.

(Refer Slide Time: 3:30)



So what is the general strategy? So while both U and V contain a number, move the smallest from those at the front of U, V to the end of S. If U contains only U contains numbers, then that means everything in V has already been moved to S and so therefore move everything in U to the end of S similarly for V. So basically elements are leaving the arrays U, V from the front and joining the array S at the back. Now you have encountered this, this is really what was happening in this taxi dispatch problem. So the code is also going to be somewhat similar.

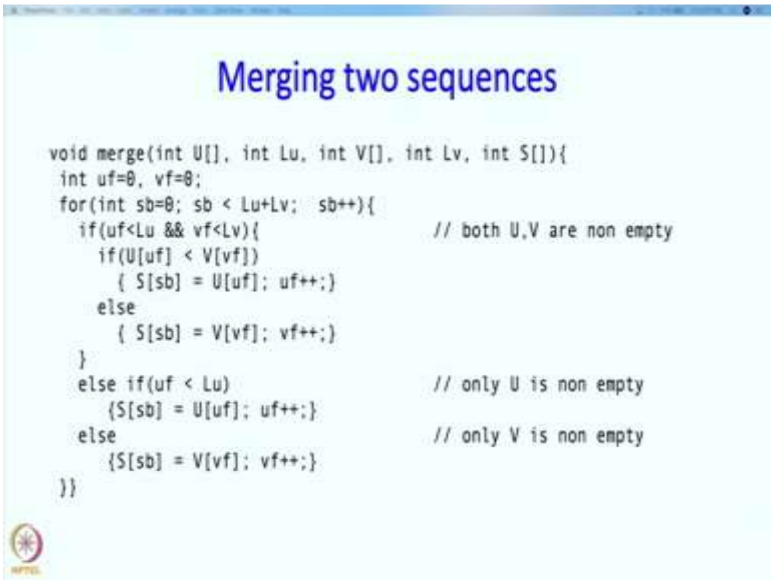
(Refer Slide Time: 4:15)



So what are the details? So we are going to maintain a variable uf which is going to denote the the element of U, the index at which that the front is currently. Similarly, so what that means is that element 0 through uf-1 have already moved out and therefore the front has now advanced to index uf. Similarly, there is vf and then we also want to have an index sb to the back of s. So this is the index denoting where the next element should move into s.

So again 0 to sb-1 contain elements that have already moved in earlier. So this is this is really pretty similar to the taxi dispatch as you might be noting. So we also need to keep track of the lengths we need to know the lengths of U and V and so let us say we use the variables Lu and Lv to denote the lengths. We should we could have a variable to denote the length of S but we know that S is going to be exactly equal to the lengths of Lu and Lv because we created U and V in that manner. So we need not define a variable for it separately.

(Refer Slide Time: 5:43)



So that is basically it, we can get started on our function for doing the merging. So again let me explain U is the first sequence that we want to merge its length is Lu, V is the second its length is Lv. And S is the array into which the result is supposed to go. And uf and vf are the positions of the fronts. So initially the front is at the 0th index here and the front of V is also at the 0th index.

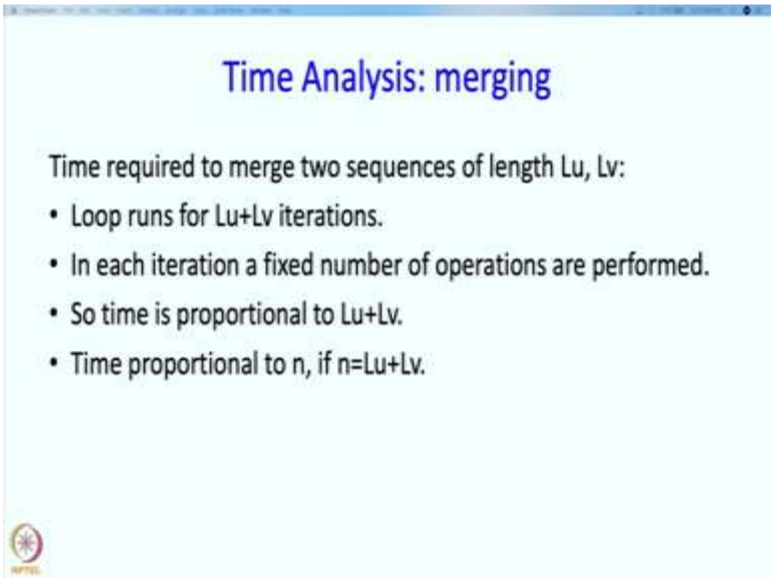
We need an element for the back but that will be a part of our main loop. So we are going to move elements at the back and so the back is going to keep on advancing, so it is going to be the 0. It will become it will keep on increasing and when we have moved Lu plus Lv elements then we are going to stop. So that is how the whole overall structure is going to be. So how do we do this movement? Well if both U and V are non-empty and when will they be non-empty? Well if uf is smaller than Lu so the front is still pointing to a valid element in U. And if the front is pointing to a valid element in V, then that means both U and V are non-empty.

In this case what should we do? We should check which one is smaller. If the U side has the smaller element than the V side what should we do? Well we should move the U side element to the back of S. And we should advance the pointer the front of uf. You should advance the back of uf as well but that will get advanced at the end of the loop anyway because of the statement over here. So we are not going to do that explicitly here. If, on the other hand, this V had the smaller element, then we should do the same thing but with V rather than with U.

So the front element of V is going to move to the back of S and the front for V is going to advance. So if one of those U and V is empty so let us say U is not empty. So uf is smaller than U, so that means V is now empty. So in that case, we do not have to do any comparisons we just move whatever is at front of U to the back and then we just advance the front for U. Otherwise, it means that only V is non-empty, so then we are going to do the same thing whatever is at front of V is going to be moved to the back of S. And we are going to advance the front of V, so that is it I should have yeah so this brace closes this brace and this brace is closing this brace over here.

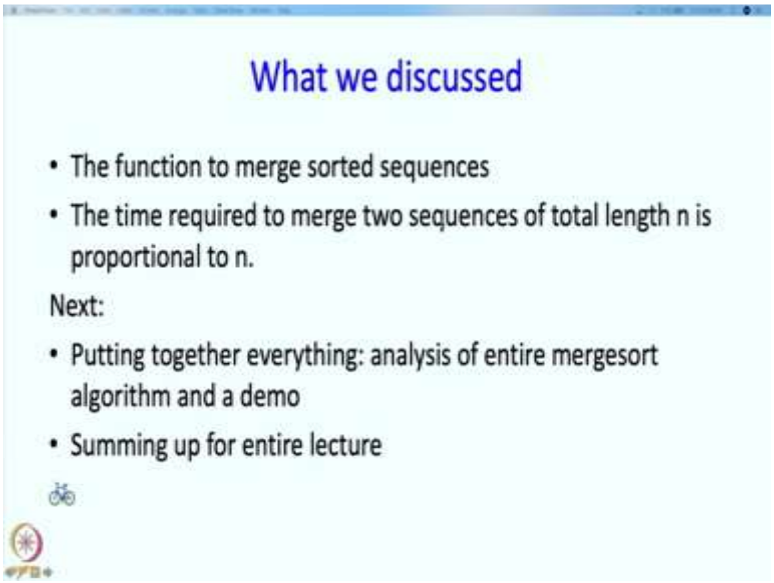
I do not want to put it later on because I just want to keep it otherwise the font becomes too small. But anyway this is this is the code for merging these two sequences. Now you can check okay so yeah so we already discussed what exactly happens. And so now we are we can analyze the time it takes okay. So how much time does this take? Well it has this main loop so this is the main loop and how many iterations is the main loop going to run? It is going to run Lu plus Lv iterations and therefore its time is going to be proportional to Lu pus Lv since in each iteration we are going to do some fixed amount of work, the work may be different, depending upon whether both queues are non empty or only one is non-empty. But it is going to be some fixed amount of work.

(Refer Slide Time: 10:09)



So if we are merging two sequences of length Lu, Lv the loop runs for Lu+Lv iterations. We do a fixed amount of work in each iteration and so the time is proportional to Lu+Lv. So I could say that the time I can say that the time is proportional to n if Lu+Lv is equal to n or if the final sequence has length n then the time is proportional to Lu+Lv. So now we have decided what the time taken for merging is.

(Refer Slide Time: 10:46)



So what have we discussed in this segment? We discussed how to merge sorted sequences and we have discussed that the time taken to merge two sequences of total length is proportional to n. In the next segment we are going to put together everything and we are going to do the analysis of the entire merge sort. And we will also have a demo of the merge sort and then we will also conclude for this entire lecture sequence, so we will take a quick break.