An Introduction to Programming through C++

Professor Abhiram G. Ranade

Department of Computer and Engineering,

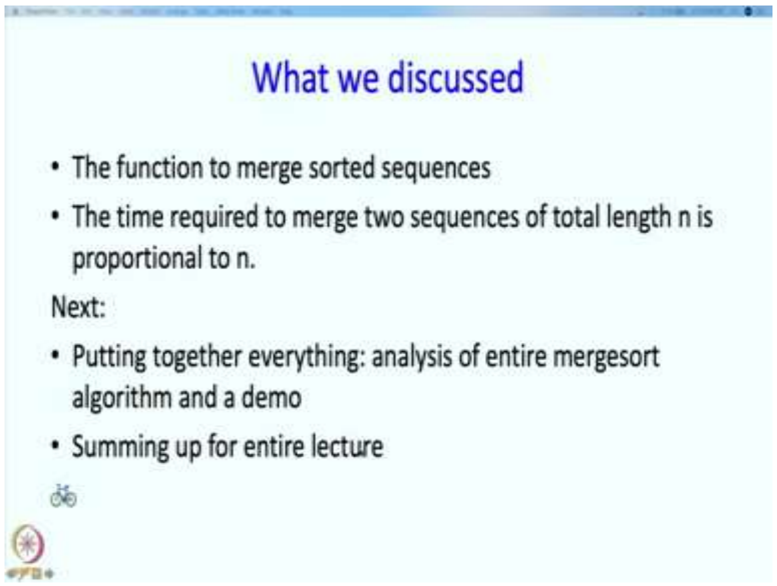
Indian Institute of Technology, Bombay

Lecture Number 18 Part-5

Array and recursion

Mergesort conclusion

(Refer Slide Time: 00:20)



What we discussed

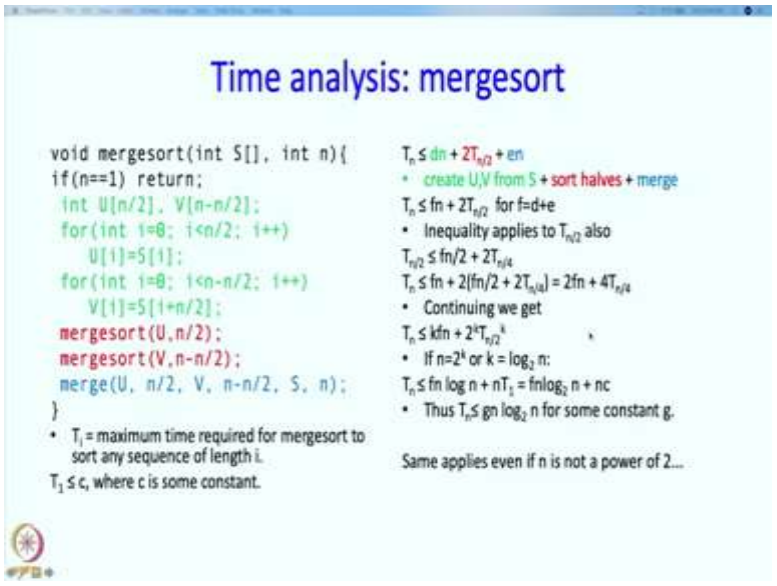
- The function to merge sorted sequences
- The time required to merge two sequences of total length n is proportional to n.

Next:

- Putting together everything: analysis of entire mergesort algorithm and a demo
- Summing up for entire lecture

Welcome back, in the last segment we discussed the merge function, and earlier we had discussed the merge sort function, and now we are going to put together everything and analyze the entire merge sort algorithm. And then we will also do a demo, and this will conclude this entire lecture sequence so we will have some remarks at the end.

(Refer Slide Time: 00:51)



Time analysis: mergesort

```
void mergesort(int S[], int n){
if(n==1) return;
 int U[n/2], V[n-n/2];
 for(int i=0; i<n/2; i++)
    U[i]=S[i];
 for(int i=0; i<n-n/2; i++)
    V[i]=S[i+n/2];
 mergesort(U,n/2);
 mergesort(V,n-n/2);
 merge(U, n/2, V, n-n/2, S, n);
}
```

- T_i = maximum time required for mergesort to sort any sequence of length i.

$T_1 \leq c$, where c is some constant.

$T_n \leq dn + 2T_{n/2} + en$
- create U,V from S + sort halves + merge

$T_n \leq fn + 2T_{n/2}$ for f=d+e
- Inequality applies to $T_{n/2}$ also

$T_{n/2} \leq fn/2 + 2T_{n/4}$

$T_n \leq fn + 2(fn/2 + 2T_{n/4}) = 2fn + 4T_{n/4}$
- Continuing we get

$T_n \leq kfn + 2^k T_{n/2^k}$
- If $n=2^k$ or $k = \log_2 n$:

$T_n \leq fn \log n + nT_1 = fn\log_2 n + nc$
- Thus $T_n \leq gn \log_2 n$ for some constant g.

Same applies even if n is not a power of 2...

So, this was our merge sort function. I have just copied it over here for reference, so we are going to analyze the time it takes, so we need some notation, so let us see that T(i) is the maximum time required for merge sort to sort any sequence of length i. So, the time required might be different for different sequences but I am just saying what is the maximum possible time to sort any sequence of length i, so that is that let me call that T(i). So this is an unknown and we are going to put down some conditions on this unknown and then we will be able to estimate this unknown.

In fact this is not just a single unknown. I am defining several unknowns at the same time, so I am defining several unknowns for an unknown for every value of i over here. So we can say something about T1. What is T1? The time required for merge sort to sort a sequence of length 1. Well that is very easy so if you execute this at this point itself we are going to return, so we will take some time, but it will be some fixed amount of time.

So let us say it will take some time C. So, I could write T1=C over here also, but I am just yeah so since everything else is going to be less than I will write this also as less than or equal to.

So, now I claim that T(n) has to satisfy this inequality. So let us see why? So this green part, so T(n) is the time required for executing this entire function for this value of n. So this green part I claim comes from this part, so we are giving mergesort a sequence of length n. So what do we do with it? So we copy parts of it into U and parts of it into V the total number of elements that we are copying is n.

So how much time does this copying take? Well it takes time proportional to n and therefore we can write this as d times n, then we are going to do a merge sort on n/2, a sequence of length n/2. So that I can estimate by 1 of these Tn by 2 and this is really a merge sort of n-n/2, but let us say n is even for simplicity, and then therefore this also takes time n/2 at most and so this time must be 2 times T(n/2).

So for this red part and then finally, we are going to do a merge of these 2 small's sorted sequences. How much time does that take? So in the previous segment we said that the merge itself takes time proportional to n, the proportional to the final sequence that results from merging these two sequences and that we know is n. Yeah, actually I do not need this argument over here so we can drop this argument we do not need it over here. But, yeah so what I have said over here is that the green part is creating U, V sort halves and the merge. So I can write I can simplify this and I can write f for d+e and so I can say that T(n) is fn plus 2 times T(n/2).

As, I as I said earlier we have several such variables T1, T2 T at T3 and similarly over here. So I am really writing not just one inequality, but I am writing lots of inequalities at once. So every n will satisfy something like this, all right but what does that mean? This means that this inequality also applies to T(n/2) and let us say n by 2 is an integer so it applies to T(n/2) also, so what is this inequality for T(n/2), well so just substitute n/2 over here, so fn will get n/2 over here and if I substitute n by 2 I get 2 times T(n/4), but now I can substitute this into this.

So, what do I get? So I will get fn plus this fn by 2, 2 this times 2, so this is the 2 that comes from here. So what is this? So I get an fn over here, then there is this 2 and this 2 cancel. So I will get an extra fn so that is what comes over here and then this 2 and this 2 will give me a 4T(n/4).

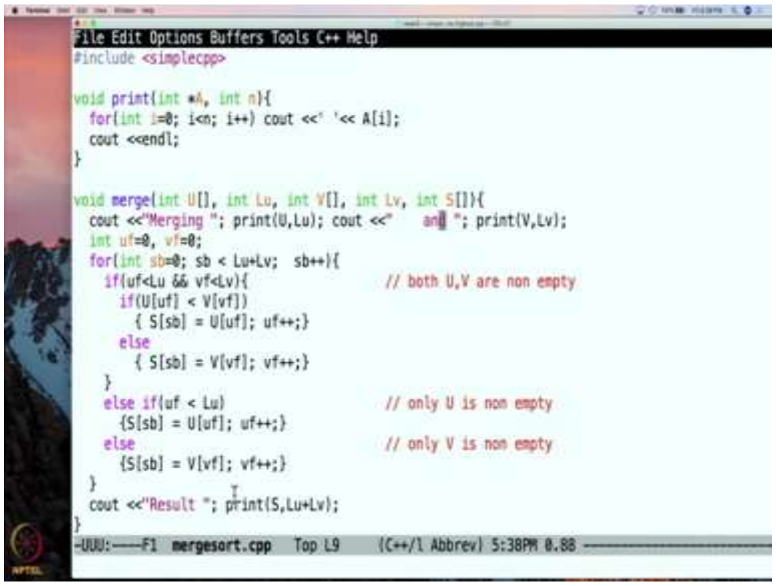
Now, I am going to assume that this n is a power of 2. So I can sort of keep on substituting into this. So I will ask you to work it out but if you keep on substituting into this what we are going to get is that T(n) if I do this k times I will get a k over here and as you can see the power over here is doubling and it is the same in the denominator over here, so I will get something like 2 raise to k times T sub n over 2 to the k so this is a little hard to read, but this is n divided by 2 raise to k over here. This 2 raised to k is in this fraction over here.

Now, I cannot do this ad infinitum of course. It have to stop then this number this number over here, say becomes 1 and I said n is a power of 2. So when will this number become 1? Well if n is 2 to the k, so if I choose k to be such that n is 2 to the k or in other words, if I choose k to be $\log_2 n$ then this will stop, but at that time what will it be. So at that time it will be k will be log n so this will become fn times log n and this will be n times T1.

But n times T1 is c. So this whole thing is less than fn log n plus c plus nc, but this n log n term is going to dominate and therefore I can write t of n has to be smaller than some constant times n log n. So which is exactly the result that we wanted, so this says that the time taken for sorting is some constant times n log n rather than some constant times n^2, and n log n is really much smaller than n^2, and that explains why or that suggests that merge sort should be much faster.

And I have made an assumption over here that n is a power of 2, but that is only a technical assumption, I mean just to simplify the algebra over here. If it is not a power of 2 all of these things actually work out but then algebra gets messier and so let us not worry about it. The main ideas are already here.

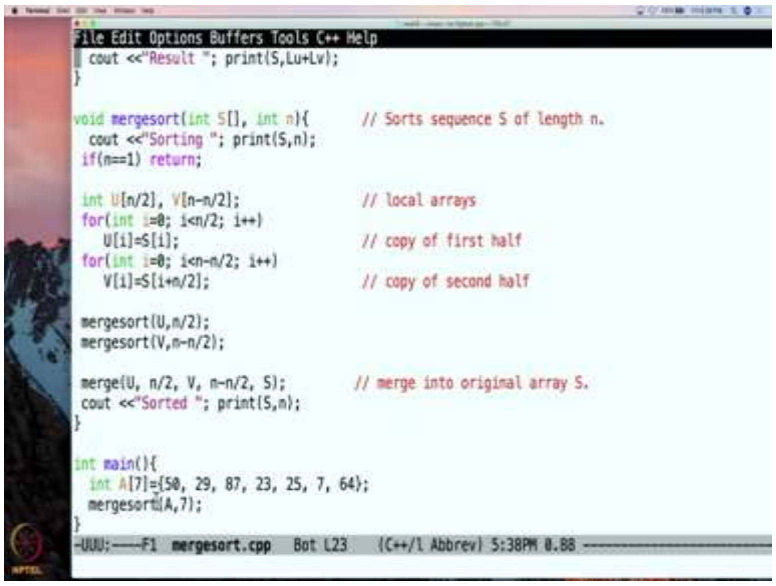
(Refer Slide Time: 08:35)



```
#include <simplecpp>

void print(int *A, int n){
  for(int i=0; i<n; i++) cout <<' '<< A[i];
  cout <<endl;
}

void merge(int U[], int Lu, int V[], int Lv, int S[]){
  cout <<"Merging "; print(U,Lu); cout <<"    and "; print(V,Lv);
  int uf=0, vf=0;
  for(int sb=0; sb < Lu+Lv;  sb++){
    if(uf<Lu && vf<Lv){                 // both U,V are non empty
      if(U[uf] < V[vf])
        { S[sb] = U[uf]; uf++;}
      else
        { S[sb] = V[vf]; vf++;}
    }
    else if(uf < Lu)                    // only U is non empty
      {S[sb] = U[uf]; uf++;}
    else                                // only V is non empty
      {S[sb] = V[vf]; vf++;}
  }
  cout <<"Result "; print(S,Lu+Lv);
}
-UUU:----F1  mergesort.cpp   Top L9     (C++/l Abbrev) 5:38PM 0.88 -------------------------
```

So we are now going to do a demo our program is mergesort. So it is really the same code, but I want to print our our array at different point, so I have written a function print over here, and that gets code, so again in the merge I am printing what array gets passed at the beginning and also the result that is produced at the end.

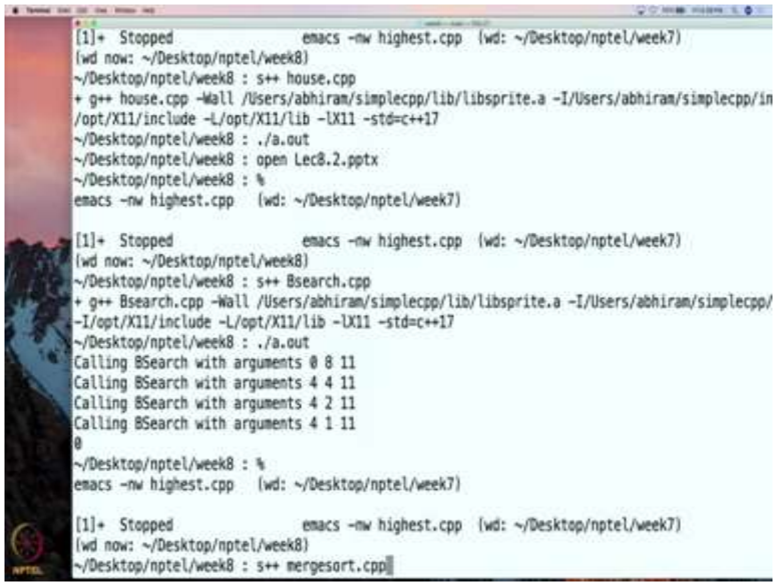
(Refer Slide Time: 09:04)



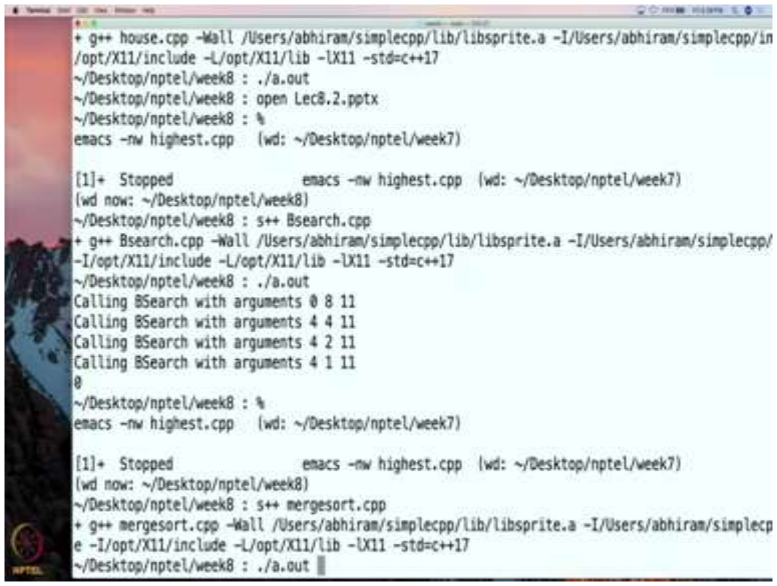
```
  cout <<"Result "; print(S,Lu+Lv);
}

void mergesort(int S[], int n){        // Sorts sequence S of length n.
  cout <<"Sorting "; print(S,n);
 if(n==1) return;

 int U[n/2], V[n-n/2];                 // local arrays
 for(int i=0; i<n/2; i++)
   U[i]=S[i];                          // copy of first half
 for(int i=0; i<n-n/2; i++)
   V[i]=S[i+n/2];                      // copy of second half

 mergesort(U,n/2);
 mergesort(V,n-n/2);

 merge(U, n/2, V, n-n/2, S);          // merge into original array S.
 cout <<"Sorted "; print(S,n);
}

int main(){
  int A[7]={50, 29, 87, 23, 25, 7, 64};
  mergesort(A,7);
}
-UUU:----F1  mergesort.cpp   Bot L23    (C++/l Abbrev) 5:38PM 0.88 -------------------------
```

And mergesort also I am printing a message saying and sorting this array over here, and at the end I am printing a message saying sorted. And this is the array to be sorted this is mergesort and that is the main program. So let us compile this and run it, and let us run it.

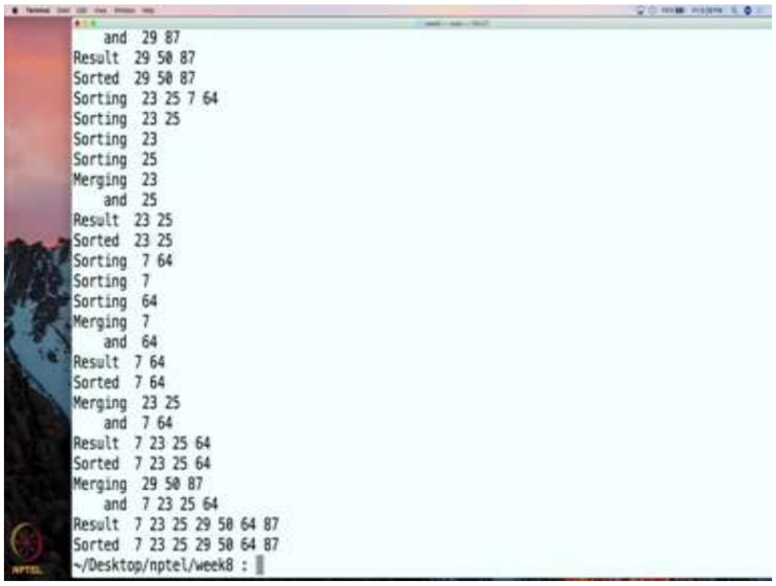
(Refer Slide Time: 09:23)



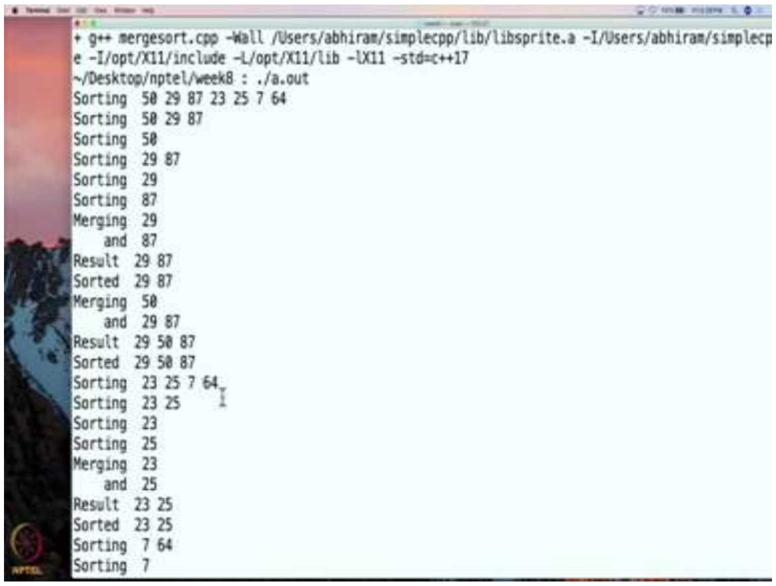
```
[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : open Lec8.2.pptx
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ Bsearch.cpp
+ g++ Bsearch.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/
-I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
Calling BSearch with arguments 0 8 11
Calling BSearch with arguments 4 4 11
Calling BSearch with arguments 4 2 11
Calling BSearch with arguments 4 1 11
0
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ mergesort.cpp
```



```
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : open Lec8.2.pptx
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ Bsearch.cpp
+ g++ Bsearch.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/
-I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
Calling BSearch with arguments 0 8 11
Calling BSearch with arguments 4 4 11
Calling BSearch with arguments 4 2 11
Calling BSearch with arguments 4 1 11
0
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ mergesort.cpp
+ g++ mergesort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecp
e -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
```



```
    and  29 87
Result  29 50 87
Sorted  29 50 87
Sorting  23 25 7 64
Sorting  23 25
Sorting  23
Sorting  25
Merging  23
    and  25
Result  23 25
Sorted  23 25
Sorting  7 64
Sorting  7
Sorting  64
Merging  7
    and  64
Result  7 64
Sorted  7 64
Merging  23 25
    and  7 64
Result  7 23 25 64
Sorted  7 23 25 64
Merging  29 50 87
    and  7 23 25 64
Result  7 23 25 29 50 64 87
Sorted  7 23 25 29 50 64 87
~/Desktop/nptel/week8 :
```



```
+ g++ mergesort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecp
e -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
Sorting  50 29 87 23 25 7 64
Sorting  50 29 87
Sorting  50
Sorting  29 87
Sorting  29
Sorting  87
Merging  29
    and  87
Result  29 87
Sorted  29 87
Merging  50
    and  29 87
Result  29 50 87
Sorted  29 50 87
Sorting  23 25 7 64
Sorting  23 25
Sorting  23
Sorting  25
Merging  23
    and  25
Result  23 25
Sorted  23 25
Sorting  7 64
Sorting  7
```

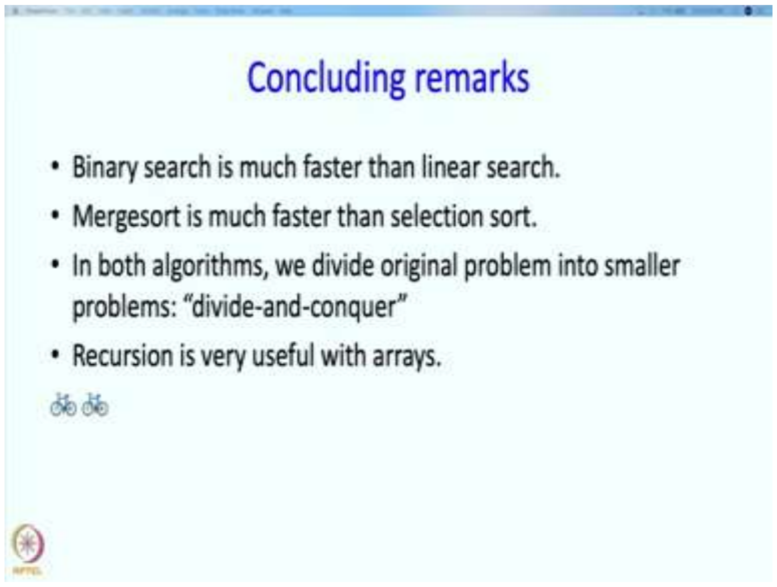
So there is a certain amount of output over here, let us start at the beginning and let us see what it is doing. So the first call to mergesort was this these this in the array. Now, if you remember what this was doing, it was splitting the problem into two parts and it was recursing. So indeed that is what is happening over here. This is the first half on which it was splitting, and it is recursing on it but what happens next? It splits on this and it recurses is on this, but if you recurse on 50 there is nothing to be done and so, this comes back to these returns immediately.

Next, the recursion and this is taken up. So that causes the smaller problems of size 1 to be created so that those also get returned immediately, so now we have two sorted sequences which were created and they are so we have a call here on the merge function. So, the result of merging these two things is 29 87.

So this was returned to this sorting call, so there was a mergesort call with this as arguments that led to this all of this this merge business, and therefore the result of this sorting call is all of this. So you can keep going in this manner you can look through this just to see, just to understand how this sorting is happening or better still you really should draw out that recursion tree to see, to understand how the sorting actually happens.

Or you can just say that look I do not need to know these details, I need to know and the way I think about recursion is I just look at the top level there the top level call and the recursive calls and I assume that, the recursive calls happen correctly if in that case can I be sure that the top level calls also happen correctly, so that is also fine, but if you need to know if you want to know what the details are you can certainly run this program maybe add even more print statements to it, so that you can see what is going on.

(Refer Slide Time: 11:54)



Concluding remarks

- Binary search is much faster than linear search.
- Mergesort is much faster than selection sort.
- In both algorithms, we divide original problem into smaller problems: "divide-and-conquer"
- Recursion is very useful with arrays.

Alright so, that really concludes this lecture. And what did we do in this lecture? So we studied binary search, and we wrote the code for it, and we also observed, and we also analyzed as to why it is going to be faster than linear search.

And similarly we wrote the code for merge sort, and we analyzed that it has to be much faster than the selection sort code that we wrote earlier in the course. Both of these algorithms have a similar idea. Which is that we take our original problem, the problem that is given to us and we divide it into two parts, and in the case of binary search we only work on one of those parts, we somehow know and that is, what the sorting gives us that we do not need to work on one part at all.

In the case of mergesort we have to work on both parts, but in any case we are looking at a big problem as being decomposed into smaller problems. And this idea of viewing and solving problems in this manner is often called divide and conquer, so what you have seen are examples of divide and conquer.

And, recursion works nicely to divide and conquer but recursion can also work quite nicely in other cases. So in any case we now have seen two examples of recursions with arrays and you can certainly note that both of these are very nice applications, and very nice algorithms, and very useful algorithms. So we will stop over here but I will, as always urge you to look at the problems on at the end of this chapter. Thank you.