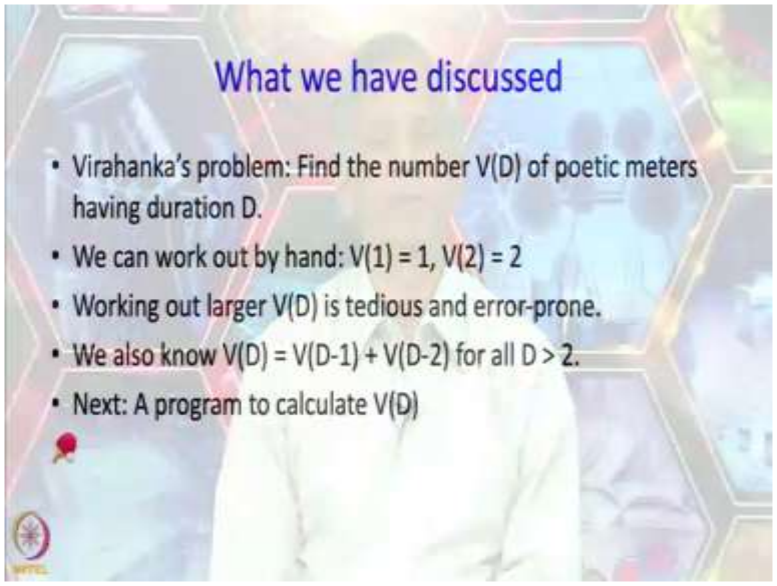
An introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 12_Part 2
Virahanka Numbers (Recursive Program)

Welcome back, in the last segment we discussed Virahanka's problem.

(Refer Slide Time: 00:27)

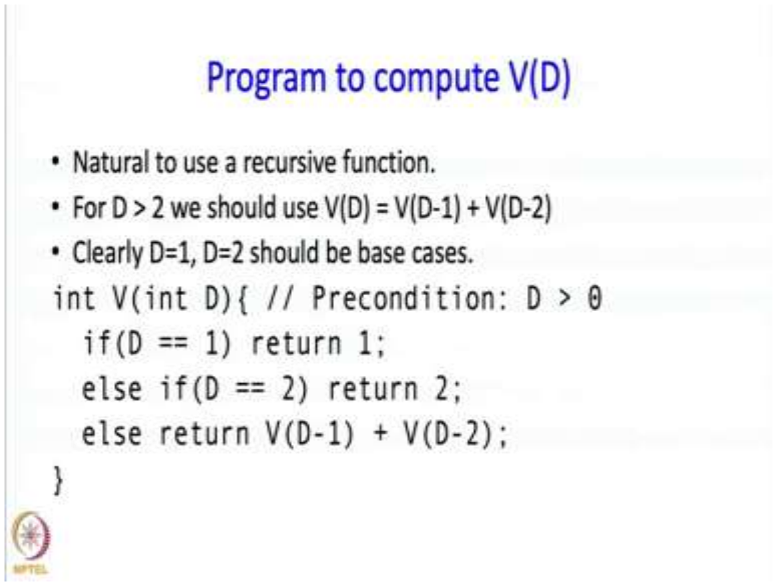


What we have discussed

- Virahanka's problem: Find the number V(D) of poetic meters having duration D.
- We can work out by hand: V(1) = 1, V(2) = 2
- Working out larger V(D) is tedious and error-prone.
- We also know V(D) = V(D-1) + V(D-2) for all D > 2.
- Next: A program to calculate V(D)

So this was about finding the number of poetic meters having duration D. In this segment we are going to discuss a program to calculate V(D).

(Refer Slide Time: 00:39)



Program to compute V(D)

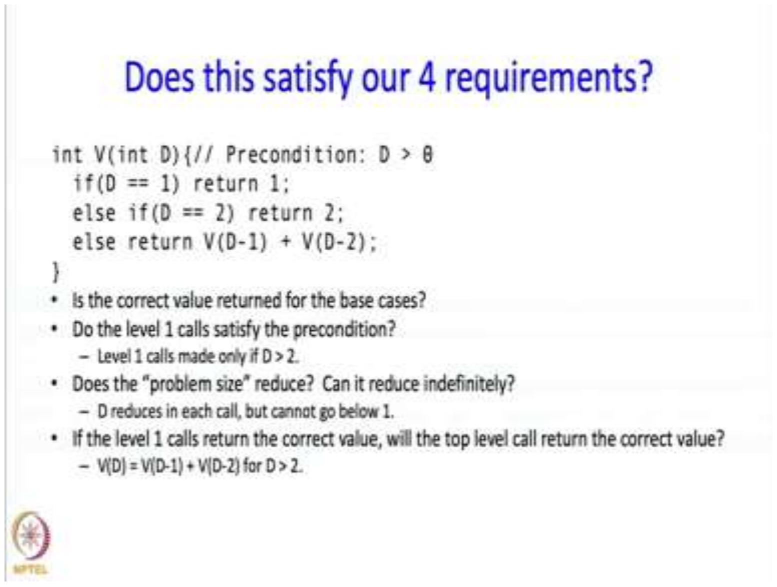
- Natural to use a recursive function.
- For D > 2 we should use V(D) = V(D-1) + V(D-2)
- Clearly D=1, D=2 should be base cases.

```
int V(int D){ // Precondition: D > 0
  if(D == 1) return 1;
  else if(D == 2) return 2;
  else return V(D-1) + V(D-2);
}
```

So it seems that it is natural to use recursive function, okay. So since you were told that for D greater than 2, V(D)=V(D-1)+V(D-2) the natural, the recursion sort of stairs is in the face and we were also told that for D equal to 1 what the values were for D equal to 1 and D equal to 2 and therefore D should be the base cases. So, this might be a natural program to write.

So V(D) which is what we want to calculate, where the precondition is the D is bigger than 0. So if D is equals to 1 then we return 1. If D equals 2 then we return 2, else we use this relationship, that is it. So, the question is, is this a reasonable program? We just wrote it but can we be sure or can we be, can we somehow check whether this is a good program?

(Refer Slide Time: 01:54)



Does this satisfy our 4 requirements?

```
int V(int D){// Precondition: D > 0
  if(D == 1) return 1;
  else if(D == 2) return 2;
  else return V(D-1) + V(D-2);
}
```

- Is the correct value returned for the base cases?
- Do the level 1 calls satisfy the precondition?
  - Level 1 calls made only if D > 2.
- Does the "problem size" reduce? Can it reduce indefinitely?
  - D reduces in each call, but cannot go below 1.
- If the level 1 calls return the correct value, will the top level call return the correct value?
  - V(D) = V(D-1) + V(D-2) for D > 2.

Well, so here is our program and in the last lecture we listed out some 4 requirements that we should check when we write a recursive (algo) recursive function. So what were those? So first all, our concern was, is the correct value return for the base cases? Okay first of all are there base cases? So are there base cases? Yes, there are 2 base cases, okay, sorry before that this is one base case and is the (current) correct value being returned? Yes, because it was specified in the problem definition or we worked it out and for D equal to 2 is the correct value being returned? Yes, we worked that out and that was 2 okay. Now does the level 1 call satisfy the precondition? So which are the level 1 calls?

So this is one level 1 call, this is another level one call and what is the precondition? D should be bigger than 0. So when we make this level 1 call, is this satisfied? Well, originally this was satisfied for D. If D was 1, then we would have returned 1, if D was 2, we would have returned. So when we come to this point, we know that D has to be bigger than 3, 3 or bigger. So then V(3) minus 1 or something bigger than 3 even this number will be positive. So then that means this precondition will be satisfied, and this number will also be positive because we said that D has to be bigger than or equal to 3 at this point. So, even for this case, we will, even for this case the precondition will be satisfied.

Now thus the problem size reduce? Can it reduce indifferently? So clearly in this case the problem size is D, there is no other candidate really. So does it reduce? When we make the recursive calls as it is reducing in the level 1 calls? Yes, it is going to D minus 1 and D minus 2. But as it reduces can it go below 1? Can it go to 0? Well, we just said that, that when we make the recursive calls the preconditions are satisfied and therefore they 0 is a limit, so the recursive are the size cannot go to 0, cannot go below 1 and therefore the set of recursive calls will have to terminate in some sense.

Final, the last check that we need to make is that if we assume that the level 1 calls returned the correct value, then can you argue that the top level calls that the top level call is correct? Okay so this is the top level call V(D) and we are assuming that these values are correct V(D) minus 1, okay. So we have D minus 1 plus V(D) minus 2 so this is correct and this is correct. So in that case are we supposed to return this? Well we got to this only if D is bigger than 2 and in that case, indeed we are supposed to return this. And therefore, the top level call is going to be correct if our level 1 calls are correct. So this really does look like a good recursive function. Yes, so that is what I had written down over here.

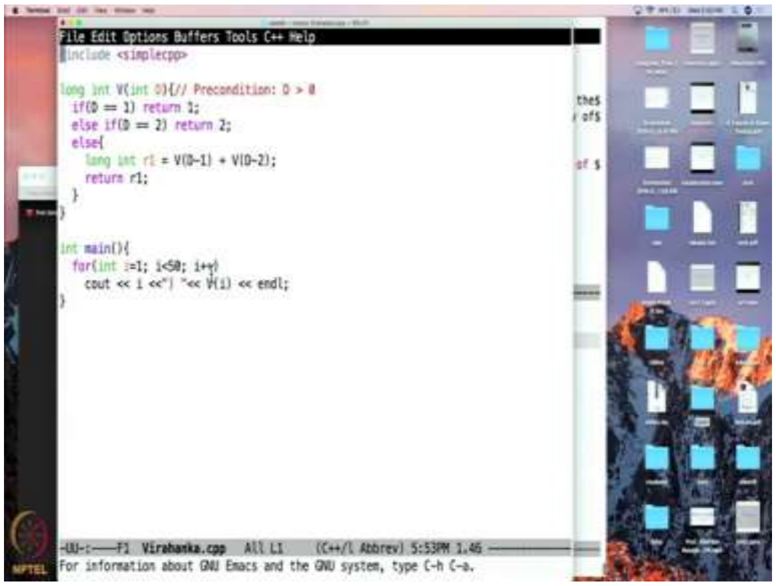
(Refer Slide Time: 05:41)

Demo

- Virahanka.cpp

All right so let us now do a demo.

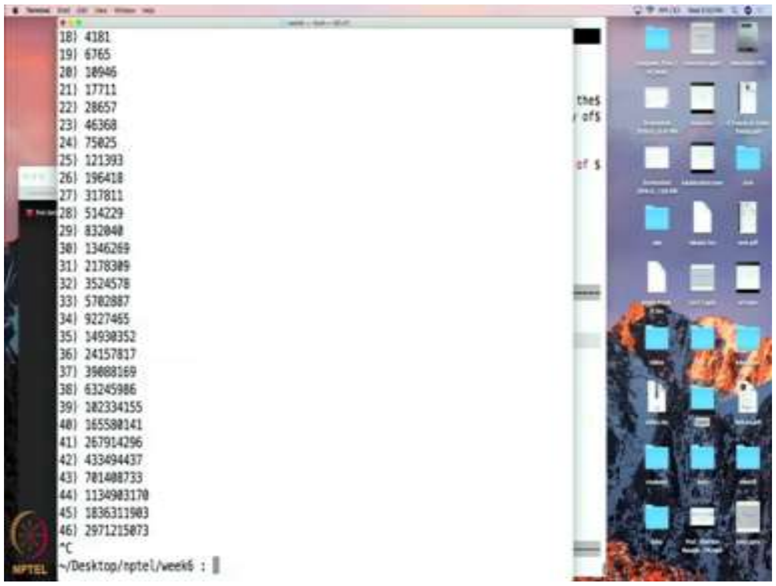
(Refer Slide Time: 05:43)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

long int V(int D){// Precondition: D > 0
  if(D == 1) return 1;
  else if(D == 2) return 2;
  else{
    long int r1 = V(D-1) + V(D-2);
    return r1;
  }
}

int main(){
  for(int i=1; i<50; i++)
    cout << i <<") "<< V(i) << endl;
}
-UU-:----F1  Virahanka.cpp   All L1     (C++/l Abbrev) 5:53PM 1.46
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Okay so this is the program that I showed to you and I have the main program and I am calling V of i. So I am just doing this for all value of i between 1 and 50 okay. So let us see how this goes.

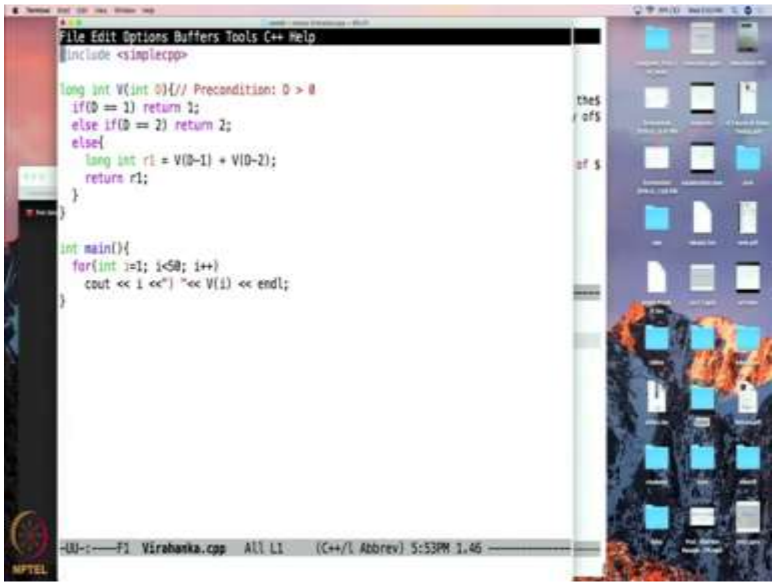
(Refer Slide Time: 05:58)



```
18) 4181
19) 6765
20) 10946
21) 17711
22) 28657
23) 46368
24) 75025
25) 121393
26) 196418
27) 317811
28) 514229
29) 832040
30) 1346269
31) 2178309
32) 3524578
33) 5702887
34) 9227465
35) 14930352
36) 24157817
37) 39088169
38) 63245986
39) 102334155
40) 165580141
41) 267914296
42) 433494437
43) 701408733
44) 1134903170
45) 1836311903
46) 2971215073
^C
~/Desktop/nptel/week6 : 
```

So let us run it, okay. So it is calculating answers pretty fast, but well now it seems to have slowed down, slowed down, okay. So let me terminate it over here. Oh no-no 45, I think 44 it took a long time, 45 maybe it will take even longer and even longer for 46. So let me not make you wait through all of this but it is clearly the program has slowed down over here. So let me terminate and (I) yeah.

(Refer Slide Time: 06:48)

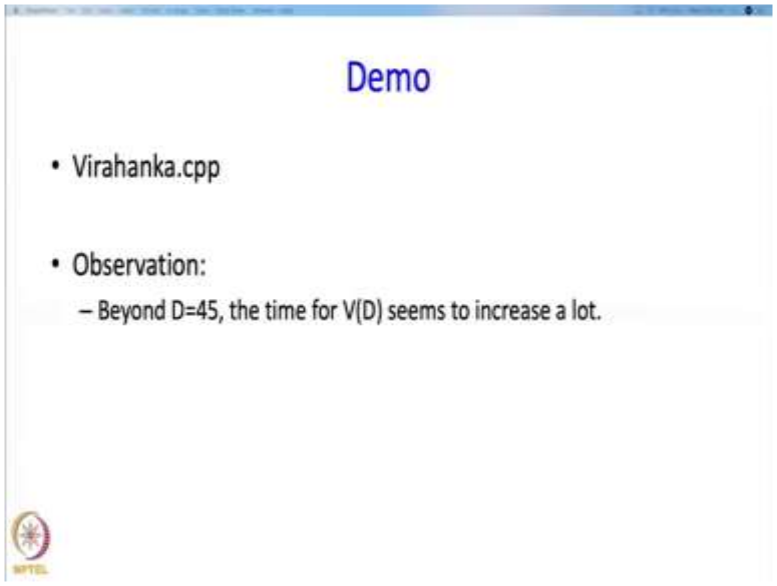


```
File Edit Options Buffers Tools C++ Help
include <simplecpp>

long int V(int D){// Precondition: D > 0
  if(D == 1) return 1;
  else if(D == 2) return 2;
  else{
    long int r1 = V(D-1) + V(D-2);
    return r1;
  }
}

int main(){
  for(int i=1; i<50; i++)
    cout << i <<") "<< V(i) << endl;
}

-UU-:----F1  Virahanka.cpp   All L1     (C++/l Abbrev) 5:53PM 1.46
```

I forgot to tell you one thing which is that when we write this program, we are going to use long int okay, why? Because as you saw, the numbers were getting bigger than what can be held in a single word. So this will be long int and because of that you could print you could get all those numbers correctly, okay.

Alright, so at this point we have a puzzle, we wanted to calculate the first 50 Virahanka numbers but the program seemed to be taking too much time at about 45, okay so we need to figure out why that is the case.

(Refer Slide Time: 07:29)

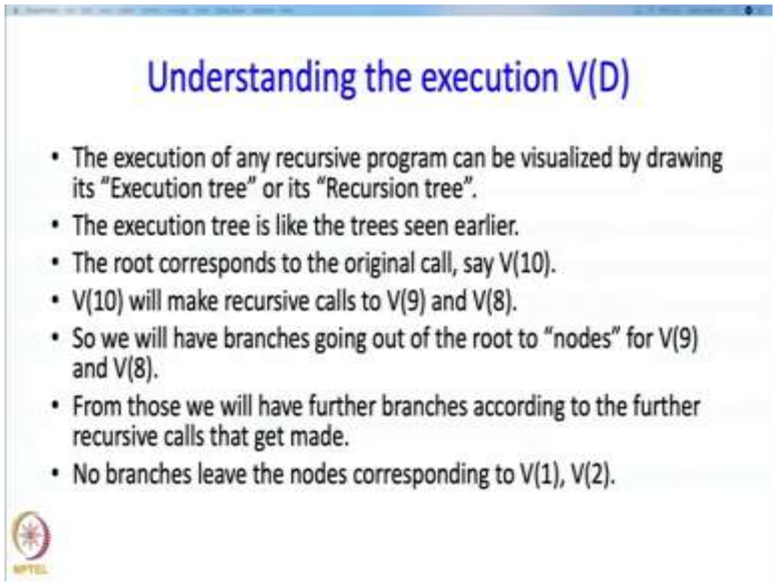


Demo

- Virahanka.cpp
- Observation:
  - Beyond D=45, the time for V(D) seems to increase a lot.

Yeah beyond D equals to 45 the time for V(D) seems to increase a lot.

(Refer Slide Time: 07:34)

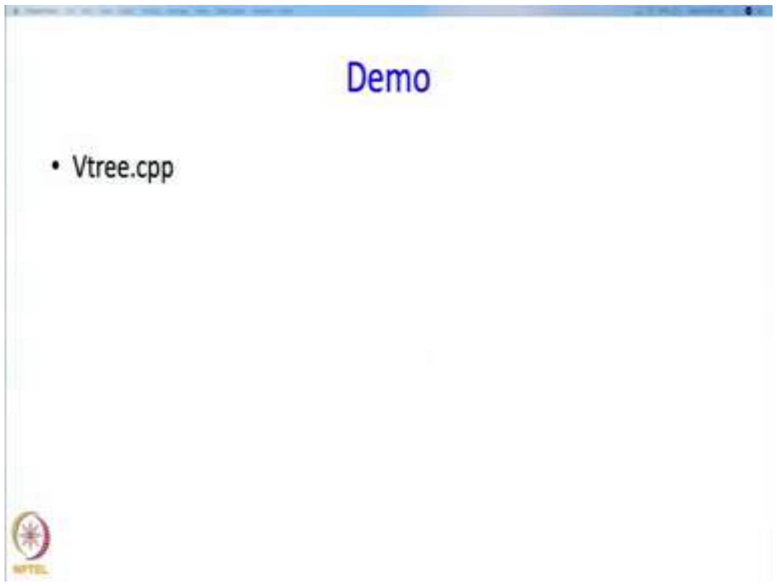


Understanding the execution V(D)

- The execution of any recursive program can be visualized by drawing its "Execution tree" or its "Recursion tree".
- The execution tree is like the trees seen earlier.
- The root corresponds to the original call, say V(10).
- V(10) will make recursive calls to V(9) and V(8).
- So we will have branches going out of the root to "nodes" for V(9) and V(8).
- From those we will have further branches according to the further recursive calls that get made.
- No branches leave the nodes corresponding to V(1), V(2).

Okay, so for that why that is happening? Let us try to understand the execution of V(D) okay. Now the execution of any recursive program can be visualized by drawing its execution tree or in this case what might be called its recursion tree, okay. The execution tree is really like the tree seen earlier, okay. The root correspond to the original call say V(10), then V(10) will make recursive calls to V(9) and V(8). So we will have branches going out to nodes for V(9) and V(8). From those we will have further branches going according to the further recursive calls that get made and V(1) and V(2) returned without recursion. So, no branches leave the nodes corresponding to V(1) and V(2). So what I am going to do now is to show you this recursion tree, okay.

(Refer Slide Time: 08:43)

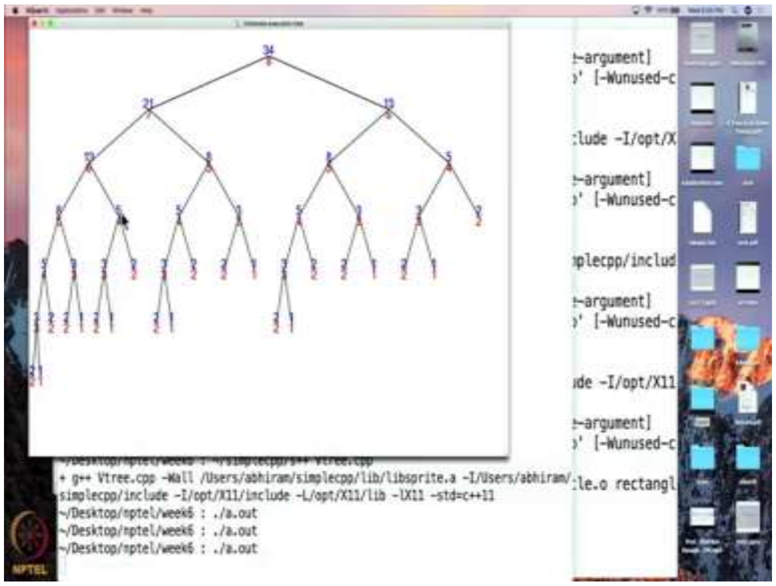


Demo

- Vtree.cpp

So for sum execution of Virahanka and in fact that I am going to do by running another program.

(Refer Slide Time: 08:54)



```
+ g++ Vtree.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++11
~/Desktop/nptel/week6 : ./a.out
~/Desktop/nptel/week6 : ./a.out
~/Desktop/nptel/week6 : ./a.out
```

So I have written a program which will draw the execution tree or the recursion tree of Virahanka, okay. So let us see how it gets drawn. So this is the recursion tree, okay let me explain what is going on. So there are 2 sets of numbers appearing on every node, okay. So the red numbers are the numbers which are the arguments to the call.

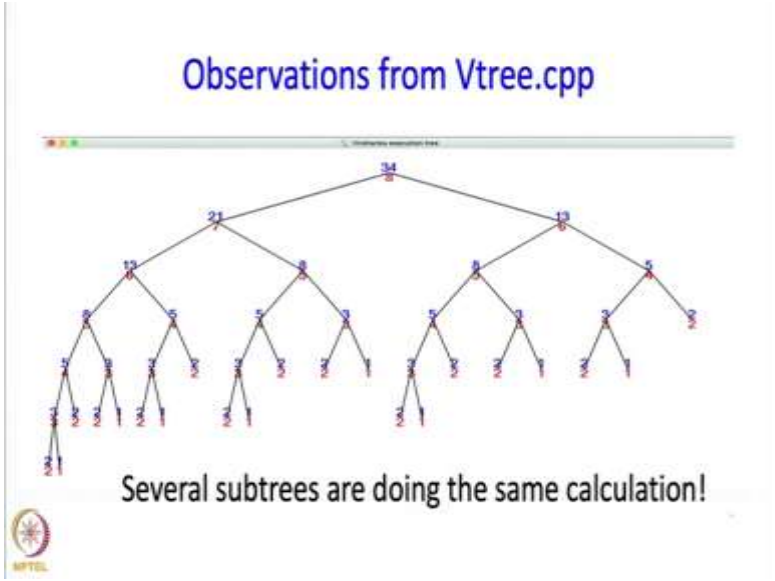
So this is asking for V(8) to be calculated. In order to calculate V(8) there is a call made to V(7) and also to V(6) then to calculate V(7) there is make a call is made to V(6) and also to V(5) and so on. And the blue numbers are the results, okay so these are the arguments, the red numbers are the arguments to Virahanka and the blue numbers are the results of the cause or this is V(8), this is V(7), this is V(6) and so on, okay. Let me run it again so that you can see the order in which the numbers get printed.

So first all the red numbers will get printed that is because first I will know only the arguments. So in order to calculate V(8), I need to calculate V(7). But the blue numbers will get printed when from bottom up so to say as soon as this Virahanka returns then you know that the result over here is a 1. When this returns you know that the V(3) is 3 and so on. So here finally when everything is done 34 will get printed indicating that V(8) is 34. So let me just run it again so that you can see the order in which things are happening.

Okay so right now only on the left side, only the arguments are known and now as things get built up then the results are also known, that the values of V are also known, okay. So now let us look at this and let us see if we can find anything interesting, indeed there is something very interesting in this. So we want to calculate V(7) and this sub-tree is what is useful for calculating V(7). So this sub-tree is responsible for calculating V(6). But even while calculating V(7) we had to calculate V(6), we did calculate V(6).

So this is that calculation and what I want you to notice is that this calculation is exactly the same as this calculation, and that is really not good because this work is exactly the same as this work and we duplicated it over here. But that is not all, things get much worse. So for example, let us look at V(5), this sub-tree sorry, let us look at V of what this number is? V(4) okay so let us look at V(4). So we want to calculate V(4), so that is what this sub-tree is doing. But V(4) is being calculated here as well and here and here and here. So V(4), the calculation of V(4) is being done in many-many-many places. So clearly something is wrong, I mean if you want to calculate V(45) in this manner, several-several Virahanka numbers will be calculated far too many times and in fact that is why it was taking so long to calculate the Virahanka of 45. Of course everything is taking long but that length really shows up when you get to 45, okay. Alright, so what can we do?

(Refer Slide Time: 13:26)

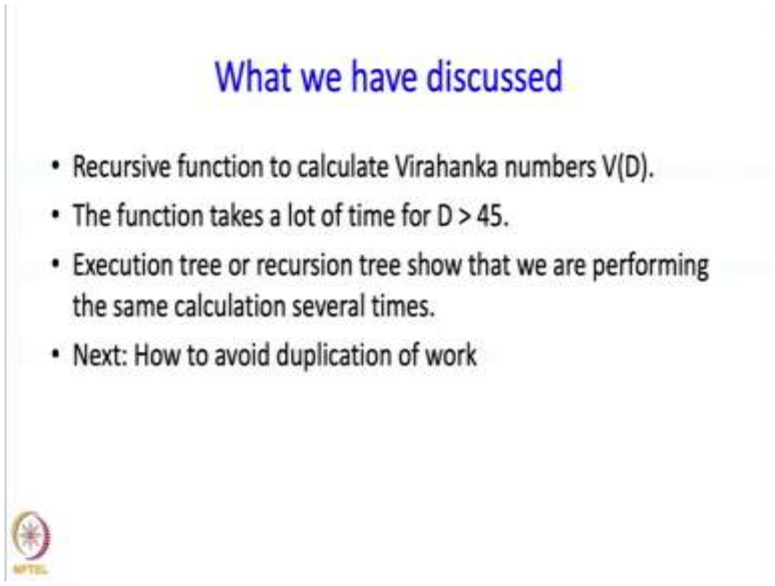


Observations from Vtree.cpp

Several subtrees are doing the same calculation!

Okay so first of all let us go back to our slides. Yeah so, we said that the (several) that several sub-trees are doing the same calculation.

(Refer Slide Time: 13:43)



What we have discussed

- Recursive function to calculate Virahanka numbers V(D).
- The function takes a lot of time for D > 45.
- Execution tree or recursion tree show that we are performing the same calculation several times.
- Next: How to avoid duplication of work

And so what have we discussed at this point? We have discussed recursive function to calculate Virahanka numbers okay, V(D) and we observed that the time takes, function takes a lot of time for D greater than 45 and when we plotted the execution tree or the recursion tree it shows that we are performing the same calculation several times. So in the next segment we are going to see how to avoid this duplication of work. So before that will take a quick break.