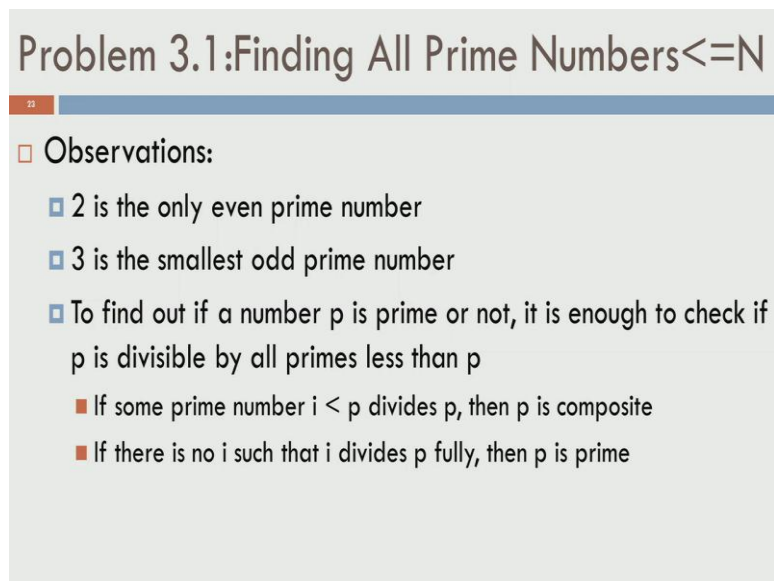


Programming, Data Structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute Technology, Madras

Module – 8C
Lecture – 11
Example: Find prime numbers

(Refer Slide Time: 00:24)



Problem 3.1: Finding All Prime Numbers $\leq N$

22

- Observations:
 - 2 is the only even prime number
 - 3 is the smallest odd prime number
 - To find out if a number p is prime or not, it is enough to check if p is divisible by all primes less than p
 - If some prime number $i < p$ divides p , then p is composite
 - If there is no i such that i divides p fully, then p is prime

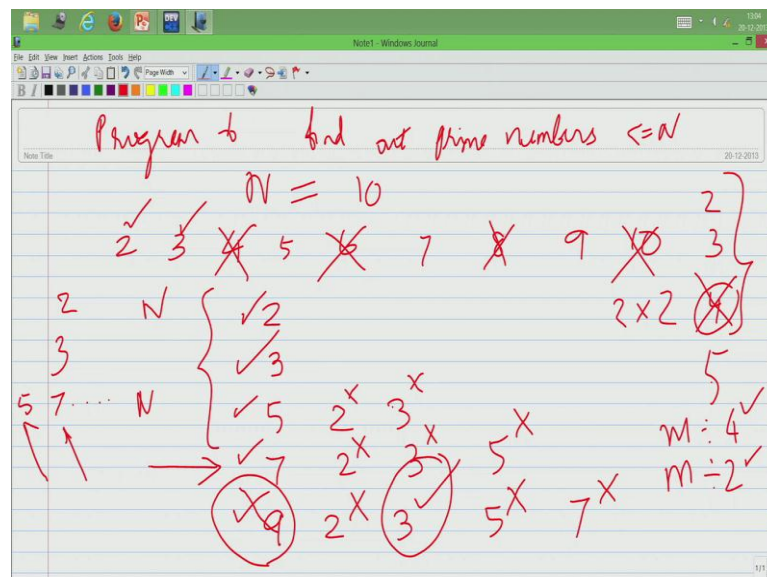
So, now let us move on to another program. And in this program what I am going to do is I am going to post the problem and actually show piece of code running inside the software, running inside the IDE. So, the problem is find all the prime numbers that are less than or equal to N .

So, I am going to give a N . And I want you to find out all the prime numbers that are less than N . So clearly, this is something that you want the computer to do. You want the program to give you this. It is not something that is going to come from the user and it is not something that is as simple as Fibonacci; where you just have to add 2 elements. This requires a little more work. It requires some bit of logic to find out all the prime numbers that are less than or equal to N .

So, let us start with some very simple observations. So, we know that 2 is the only prime

number that is even. So, 2 is the smallest prime number. And we also know that is the only even number that is prime. All the other even numbers are not prime, because 2 is a factor. Three is the smallest odd prime number. And from now on, we know that 2 and 3 are prime numbers. We will have to start checking from 4 onwards, if they are prime number or not. So, 4 is not a prime number because it is a multiple of 2 it is an even number. Five is a prime number. 6 is not because it is even; 7 is a prime number and 8 is not because it is even. 9, even though it is a odd number is not a prime number, because it is 3 times 3 and so on. So, what we are going to do is we are; so I am going to layout a technique to solve this problem. So, let us see how to do this using paper and pencil first. And then we will move on and see how to write a program for this.

(Refer Slide Time: 02:02)



So, I am going to assume that N equals 10. So, I want all the prime numbers that are less than or equal to 10. If 10 is a prime number I want to use, I want to declare that 10 is a prime number also. So, 10 is not. But whatever N I take, I want to print all prime numbers that are less than or equal to N. So, this is the program to find out prime numbers less than or equal to N.

So, since I start with N equals to 10, so let us make use of the observations. So, what could I have? I could have 2, 3, 4, 5, 6, 7, 8, 9 and 10. I already know that 2 is a prime

number and 3 is a prime number. I also know that all the even numbers are not prime. I know that all the even numbers are not prime. So, I am going to start with 2 and 3. And I have all the odd numbers that are less than or equal to N . So, in this case it is 2, 3, 5, 7 and 9. And of these, I already know that 2 is a prime number and 3 is a prime number. Let us see how to find out if 5 is prime or not. So, one thing we are going to do is we are going to assume that 5 is a prime number initially, until we have a proof that 5 is not prime.

So, in this case 5 is guilty of being prime, until it is proven otherwise. So, what are the different things that we can do to check whether 5 is prime or not. Of course, I can go from; if 5 is not a prime, then one of these numbers 2, 3 or 4, we should be able to divide 5 by that and get no remainder. So, let us do that. Let us say I do 5 divided by 2; the remainder is one. I do 5 divided by 3; the remainder is 2. I do 5 divided by 4; the remainder is one. So, all these numbers which are less than 5, I divide them into 5. It leaves the remainder. So 2, 3 and 4 are not factors of 5. So, 5 is actually prime.

But, one thing we have to observe is that if you look at 4, right, 4 is already 2 times 2. So, if I want to check whether 5 is divisible by 4 or not, right, let us assume that some number m is divisible by 4. m is actually divisible by 4. But, if m is divisible by 4 it is also actually divisible by 2. The reason is 4 is actually 2 times 2. So, I do not have to really check whether m is divisible by 4 or not, if I have already checked if m is divisible by 2. So for 5, I am going to check only if it is divisible by 2 or 3; I will not check if it is divisible by 4. So, I go and check whether 5 is divisible by 2; it is not. Is 5 divisible by 3; it is not. I started with the assumption that 5 is prime. So, it is not changing. So, 5 is indeed prime.

Let us do the same thing for 7. So, what I am going to do is I am going to; so as, for 7 also I can start from 2, check for 3, check for 4, check for 5, 6 and so on. But, I really do not have to check for 4 and 6 divisibility, because if for 4 I already know 2 is a factor, for 6 I already know that 2 and 3 are factors. So, therefore I do not have to check 7 divisible by 6 or 4.

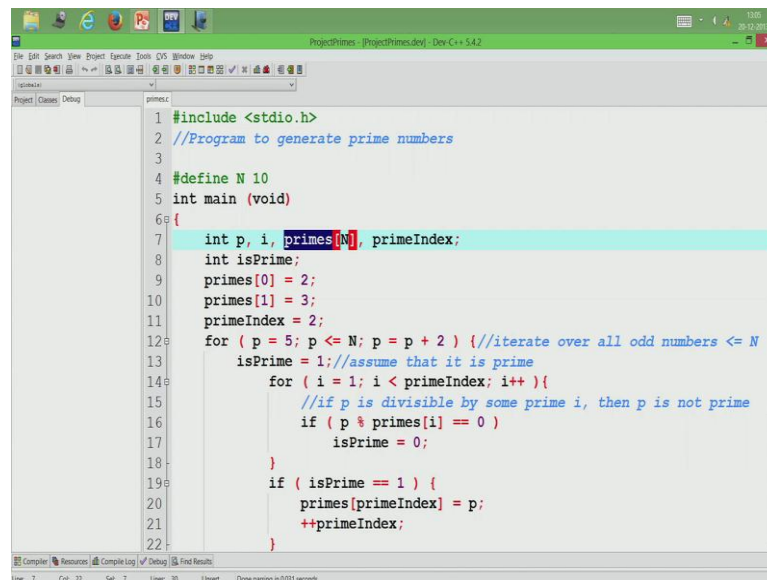
So, to put it more succinctly what we are checking is, for any number N you are going to

check only if it is divisible by other prime numbers that are strictly less than N or not. For 7, the list of prime numbers that are less than 7 or 2, 3 and 5, I do not have to check other numbers like 4 and 6 and so on. We will only check prime numbers that are less than 7. So, what are the prime numbers that are less than 7? They are 2, 3 and 5. So, again as before I will assume that 7 is actually prime, until it is proven otherwise... So, I will go and check whether it is divisible by 2; it is not. I can check whether 7 is divisible by 3; it is not. Is 7 divisible by 5; it is not. So, you have checked all the prime numbers that are less than 7. So, at this point 7 is actually guilty of being prime and it does not change.

Let us do this little thing for 9. I start with number. So, what are the prime numbers that are less than 9? They are 2, 3, 5 and 7. I will assume that 9 is a prime number to start with. I check whether it is divisible by 2; it is not. I can check whether it is divisible by 3 or not. In this case, 9 is actually divisible by 3. So, it is divisible. It is not divisible by 5 and it is not divisible by 7. So, I started with the assumption that 9 is actually prime, however 9 was divisible by 3. The moment it is divisible by one prime number, even one prime number, less than the N that I am looking at, then this assumption that this number is prime is not true anymore. So, I have to change that assumption. So, 9 was assumed to be prime and it is not prime anymore, because it is divisible by 3.

So, in summary what I am going to do is I am going to start with number 2 and 3. I will assume that they are already prime and I am going to check. For given N , I am going to check only odd numbers starting from 5, 7 and so on up till N . So, if N is odd I will include N also; if N is even I will go only till N minus one. And for each of these numbers, I am going to look at all the prime numbers that are less than that number and see if that number divides into this current number under consideration or not. And I do this till end.

(Refer Slide Time: 08:43)

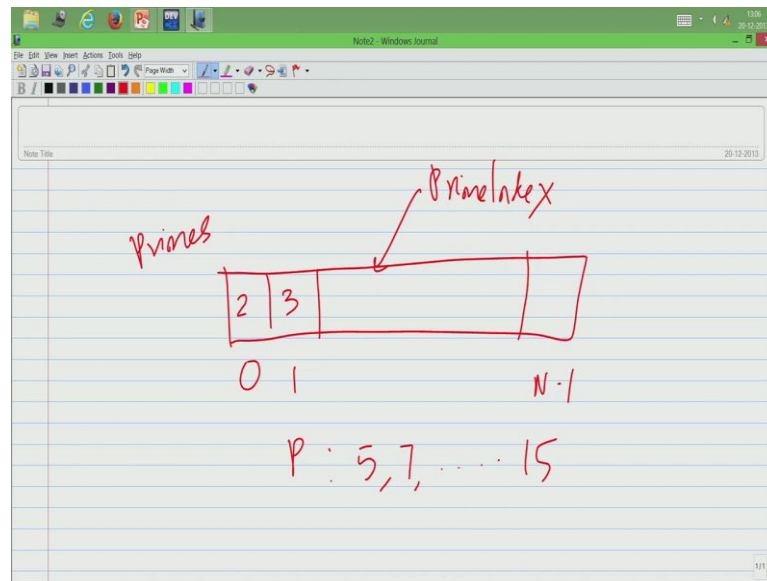


```
1 #include <stdio.h>
2 //Program to generate prime numbers
3
4 #define N 10
5 int main (void)
6 {
7     int p, i, primes[N], primeIndex;
8     int isPrime;
9     primes[0] = 2;
10    primes[1] = 3;
11    primeIndex = 2;
12    for ( p = 5; p <= N; p = p + 2 ) { //iterate over all odd numbers <= N
13        isPrime = 1; //assume that it is prime
14        for ( i = 1; i < primeIndex; i++ ){
15            //if p is divisible by some prime i, then p is not prime
16            if ( p % primes[i] == 0 )
17                isPrime = 0;
18        }
19        if ( isPrime == 1 ) {
20            primes[primeIndex] = p;
21            ++primeIndex;
22        }
23    }
```

So, let us see a small program segment that I have written to do that. So, in the process I also want to show you some features of the IDE that I am using which is called Dev C plus plus. So as before, I start with hash include stdio dot h, because I want to be able to print the prime numbers later. And I also have hash define N equals 10. So, I define N to be 10. I want to print all the prime numbers starting at 2 up to 10; whatever prime numbers are there I want to find them out. So, in the worst case all of them could be prime numbers. So, I have an array called primes. So, I have this array called primes. And what it is going to keep is all the prime numbers that are less than or equal to N. And you are actually allocating N numbers or N locations for prime numbers. So upfront, if I give you a number 100 you do not know how many prime numbers are there, I am assuming that all hundred could be prime numbers and I assign or I create an array of size hundred. So, they are of course going to get index from 0 to 99.

So, I look at this line; primes of 0 is 2. So, the first prime number is 2. I put that at location 0. Primes of one is 3. So, the next prime number is 3. So, I already have 2 locations filled up; namely location 0 and location 1. I can start at location 2 and start filling up the prime numbers.

(Refer Slide Time: 10:26)



So, the array that we are going to maintain; so this is the setup. We are going to maintain an array. And this array is going to be called primes. It is going to have locations from 0 to N minus one. Location 0 is going to contain 2; location one is going to contain 3. And prime index is a variable that I am going to use to index into the array called primes. So, that is prime index.

And p is an iterator that is going to run from 5, 7 and so on. So, if N is fifteen it will go till fifteen; 5, 7, up to fifteen. So, that is what this loop does. So, p equal to 5; p less than or equal to N. So, the nice thing that I have done here is I have incremented p by 2 already. So, if I start with 5, if I go up to N in increments of 2 and starting with 5 and then it will be 7, 9 and so on. I am looking only at odd numbers. I will never have even numbers.

So, I also want to show you something else. So, on the left side if you see there are line numbers. So, the line number hash include gets line number one; this for loop is starting at line number 12; this for loop is starting at line number 14 and so on. So, I do not know if you have been paying attention so far when I showed the IDE. So, this for loop starts at this line and ends at this line. And we can see a thin line running from f to close bracket.

Similarly, this if statement here; you can see a thin line running from here to here. And for this for loop it is running from somewhere. You do not see the end of it. So, this is something that many IDEs give you for tracking source code and being able to write code easily.

So, you see the line numbers on the left side. You also see what is called the gutter. So, gutter is a small gully on the left side. So, on the gutter you see the line numbers. You also see small rectangles on the left side. So, these are called foldables. So, if you see this for loop, right, I can click on this line number 14 and it folded. After 4teen you see line number 19 now. So, line numbers 14 up to 8een are now hidden. If I click on this minus symbol next to 19, it again folded. So, I can click on it again and it will expand. So, I can click to expand or fold. So, let us assume that these are folded now. So, if you do that what the IDE does is it just shows you opening and closing braces without the statements inside.

So, let us see what the loops starting at twelve and ending at line number twenty 3 is supposed to do. We are going to start with p equals 5. I will assume that 5 is prime and then I will check whether it is prime or not. Then I will either write it into the array or not. Then I will assume that 7 is prime. I will check whether it is actually prime or not, I will register it if it is prime, then I move on to 9 and so on up till N is over.

So, the loop running from line number twelve to twenty 3 is only iterating over all the odd numbers. For each odd number, we assume that it is prime. So this, we are going to track by using a variable called prime. So, you can think of this as a question. Is 5 prime? And one means it is prime and 0 is usually used for false. So, one is usually used for true and 0 is usually used for false. So, is 5 prime; I will assume that 5 is actually prime and have to run a loop to actually verify if it is prime or not.

So, let us look at the loop from line number 4teen to 8een. So, in line number 4teen you have a loop for i equals 1; i less than primeIndex; i plus plus. So, if p is divisible by sum p of i, then p is not prime. So, we are checking that in line number 16 to 17. So, if p is actually divisible by a prime number that is less than it, that is what you have in line number 16, then your is Prime becomes 0. You start with is 5 prime; yes. And this if

condition checks whether that has to be changed or not. And when this for loop is over, you come and check whether your assumption is true or not. If your assumption still holds true; if the current p is not divisible by any prime number less than it, then it is still prime and you update it, you increment the prime index by one. Otherwise, you do not increment the prime index, you do not record the value and so on. And line number twenty 4 is just a loop which prints all the prime numbers.

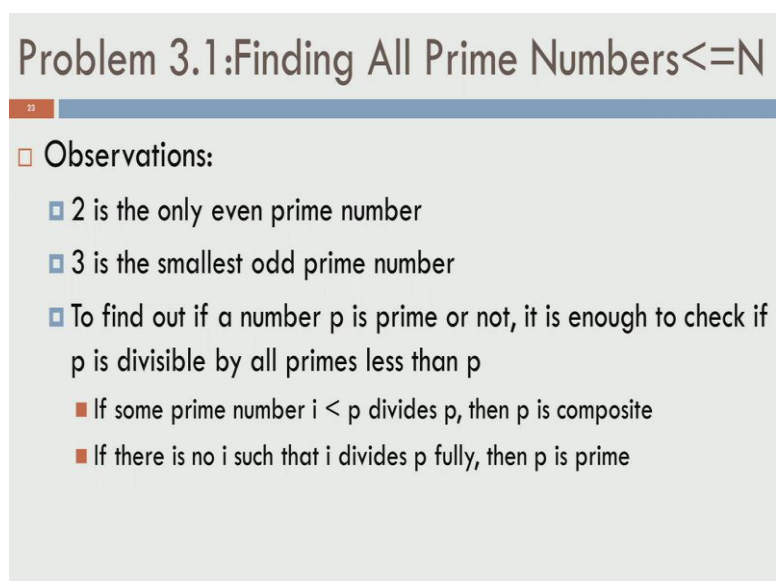
So, we have this setup now. I want to run it, compile it, run it and show what it does. So, see that N is equal to 10. I am going to compile it. So, I see that there are no errors and I run it. You can see that I printed values 2, 3, 5 and 7. So, it does not have any number which is not a prime number. So, as I said earlier I want to go and change this. Let us I want to print all the numbers from 2 to fifteen which are prime. So, the first thing I did is I edited it and then I saved it. So, let us say I change it to 15. You can see that there is a star on the top, which says it is not saved yet. The first thing I do is save. Let us say I do not compile it, I just run it. I saved it, but I did not compile it. So, it is still executable that I have; still holding the whole value of N which is 10. So, it is printing only prime numbers up to 10. So, to correct that what I have to do is I have to compile it once more. I compile it once more. Now the compilation is over, now I run it. You see that the print is 2, 3, 5, 7, 11 and 13. So, I have all the prime numbers from 2 to thirteen included in this.

Then let us say I want to change this number and I want the first hundred numbers. Not, the first hundred numbers and whatever is prime in the first hundred integers I want to print those. I change N to hundred, I save it, I compile it first. So, I compile it and then I run it. You can see that numbers are starting from 2 it goes from 3, 5, 7, 11 and so on up till 97. So, this is what I was talking about earlier. Usually you test your program with small values and once you are comfortable about that, then you go ahead and put larger values.

So, let us say I want thousand; up to thousand I want all the all the prime numbers to be printed. So, again I change it to thousand, I compile it first, then I run it. And now you can see that I have a screen full of prime numbers. I start with 2 and the last prime number less than thousand seems to be 997. So, you can go and verify that the number

before that is 991 and so on. You can go and verify that. So, one thing that I want you to observe is that the line numbers are a very useful thing. So, use them when you want to discuss programs and so on. And you can also use this collapse and expand. If you do not want to see all the code you collapse and expand. So, when I want to see what the outer most loop is doing, I collapse everything else inside. So, now this is very clear. So, I am I iterating over all the odd numbers less than or equal to N. right. And if I want to see how I am actually checking whether a number is prime or not, I expand this loop on line number 14 and I see that I am checking the current p against all the prime numbers that are less than this. And finally, if the assumption is still true, right, I record it; otherwise, I do not record it. So, if isPrime is 0, you see that there is no else clause. So, we are not recording it and I print everything and return 0.

(Refer Slide Time: 19:32)



Problem 3.1: Finding All Prime Numbers $\leq N$

- Observations:
 - 2 is the only even prime number
 - 3 is the smallest odd prime number
 - To find out if a number p is prime or not, it is enough to check if p is divisible by all primes less than p
 - If some prime number $i < p$ divides p, then p is composite
 - If there is no i such that i divides p fully, then p is prime

So, we wrote a small program which printed all the prime numbers less than or equal to N. So, at line number twelve we had this outer most loop running which is iterating over all the odd numbers. At line number 14 we had this primality check.

(Refer Slide Time: 19:38)

Code Segment

```
24
12:   for ( p = 5; p <= N; p = p + 2 ) { //iterate over all odd numbers <= N
        isPrime = 1; //assume that it is prime
14:       for ( i = 1; i < primeIndex; ++i ) {
                //if p is divisible by some prime i, then p is not prime
                if ( p % primes[i] == 0 )
                        isPrime = 0;
        }
19:       if ( isPrime == 1 ) {
                primes[primeIndex] = p;
                ++primeIndex;
        }
}
```

And line number 19 goes and verifies our initial assumption whether it is actually prime or not. So, there are lot of things it you can do to this program. You can make this program more efficient.

(Refer Slide Time: 19:59)

Exercise

- Can make the program more efficient
 - ▣ Check only till \sqrt{N}
 - ▣ If p is divisible by some prime number, the loop in Line 14 still runs till all prime numbers less than it are checked
- Each of these techniques will reduce number of steps
- Can also combine both

So, for example, when we check fifteen we would have check 2, 3, 5, 7 9 and so on, if

you did not account for just the prime numbers. If you check only the prime numbers, you would have still checked 2, 3, 5, 7, 11 and 13 also to check for 15. But, there is a small mathematical property that if you are looking at a prime number it is enough to check prime numbers that are less than or equal to square root of N. You do not even have to go to the last prime number that is less than N. So, you can make that change.

The other thing is if p is divisible by some prime number, the loop will still check and keep running. So, for example, if you have 9 you would have checked it against 2. When you check against 3, you already know that it is not prime; 9 is not prime. You still go ahead and check it with 5 and 7. What is the point? We already know that with 3, 9 is not a prime anymore because 3 is a factor of 9. So, you can use these 2 tricks. Go only till prime numbers that are square; less than square root of N. And at some point, if your assumption is violated why even bother looking at all the entries. So, you can break right then and there and say that the number is not prime any more. So, you can combine each of these techniques. So, you can have this each of these techniques in place or it even combine them both to give you a better program.

So, this code segment can be modified to do these 2 changes. So, do not go till all the prime numbers less than N; go only till square root. And even in that, at some point if we find out that a number is composite do not go any further, you can break out of the loop. So, you may have to make changes in 2 places. So, you have to make changes here, so that you do not run everywhere. And at some point, once you get if isPrime; so if isPrime become 0, which means it is already a composite number. You do not run the loop any more. You can stop and do something about it. So, I give you these 2 things as exercises; go and do this.

So, thank you very much.