

Programming, Data Structures and Algorithms
Prof. N.S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 02

Lecture – 29

Example: Polynomial evaluation by direct method

Polynomial evaluation by Horner's method

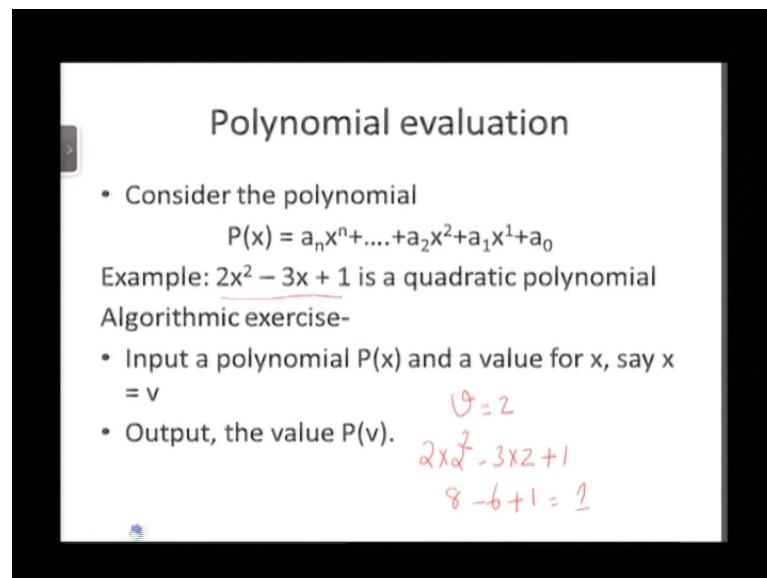
Example: Multiplying two n bit numbers by direct method

Multiplying two numbers by Karatsuba's method

Analysis for Karatsuba's method-recurrence equation

Big-O notation

(Refer Slide Time: 00:11)



Polynomial evaluation

- Consider the polynomial
$$P(x) = a_n x^n + \dots + a_2 x^2 + a_1 x^1 + a_0$$

Example: $2x^2 - 3x + 1$ is a quadratic polynomial
Algorithmic exercise-
- Input a polynomial $P(x)$ and a value for x , say $x = v$
- Output, the value $P(v)$.

$v = 2$
 $2x^2 - 3x + 1$
 $8 - 6 + 1 = 1$

So, let us move ahead to a very closely related topic which is polynomial evaluation where we need similar tricks to improve the efficiency of evaluating polynomials. Consider the polynomial which you see here which is p of x , which suggest that the name of the polynomial is p , and x is a variable. And the polynomial is a n x power n plus a n minus 1 x to the power of n minus 1, all the way upto a 2 x square plus a 1 x plus a naught. So, this is a polynomial of degree n which means that the variable x occurs as x power n ; and a_0, a_1 to a_n are coefficients.

(Refer Slide Time: 01:20)

Outline

- Iterative algorithm
 - Evaluate from right to left
 - In i^{th} iteration
 - Calculate v^i from v^{i-1} (one multiplication) ✓
 - Multiply a_i (one multiplication) ✓
 - Add the result to the current value (polynomial on the first i terms) ✓
 - Total number of arithmetic operations
 - $2n$ multiplications + $n-1$ additions = $3n - 1$ operations
 - Can we improve this*

$i=0, a_0$
 $i=1, x \times a_1 +$
 $i=2, x^2 \times a_2 +$
 $i=3, x^3 \times a_3 +$

As you can see there is an example here; the example is a quadratic polynomial. In other words, a polynomial of, it is a quadratic polynomial, right, which means the polynomial of degree 2. And the algorithmic exercise is an extensively used algorithmic exercise though it looks extremely simple. Input a polynomial, given a polynomial p of x as input, and value v for x ; x is given a value v ; v challenges to output the value p of v .

So, for example, let us take the given polynomial, and let us say v equals 2, in which case we have to compute 2 multiplied by 2 square minus 3 multiplied by 2 plus 1. And this is 8 minus 6 plus 1 which is equal to 1. Observe that there are many ways of evaluating this polynomial, and the most simple way of evaluating this polynomial is the following iterative algorithm where you evaluate the formula, evaluate the polynomial from right to left.

(Refer Slide Time: 02:53)

Horner's method

$$P(x) = P_0(x) = a_0 + x * P_1(x)$$
$$P_1(x) = a_n x^{n-1} + \dots + a_2 x^1 + a_1$$

In general

$$P_i(x) = a_n x^{n-i} + \dots + a_{i+1} x + a_i$$
$$= a_i + x * P_{i+1}(x)$$
$$P_n(x) = a_n$$

Note the gain : in each iteration one multiplication and one addition, $2n$ operations

You evaluate the polynomial from right to left; what do you mean by right to left? You start off in the i th iteration, for i is equal to 0. You take the value is 0. And this is the value that you have computed. For, i is equal to 1, you take the value x multiplied by the value a_1 and perform an addition. In the iteration number 2, you take the value x , you multiply it by x again which gives you x square, store this and keep it with you, or store it in a temporary variable, multiply this by a_2 and add the value that you have computed so far.

And in iteration number 3, you take the value of x square that has been computed in the previous iteration, multiply it by x , you compute x cube multiplied by a_3 , and perform an addition; this is what I mean by evaluate from right to left. In other words, you can think of it as the polynomial being evaluated from the element of least degree to the element of highest degree; note that a_0 is a constant. The multiplier is x power 0 that is 1, and you evaluate this all the way upto a n x power n .

Let us evaluate this algorithm. Indeed the algorithm is correct because it is quite clear that in every iteration, one part of the polynomial is being calculated on the given value v . So, in every iteration you calculate v power i from v power i minus 1, you multiply it by a_i ; this causes a total of 2 multiplication operations; then you add the result to the current value which gives you the value of the polynomial on the degree i polynomial that is $a_i x$ power i all the way upto term a_0 .

The total number of arithmetic operations that we have performed is $2n$ multiplications. There is in every iteration you perform 2 multiplications and 1 addition. And this counts for $2n$ multiplications plus n minus 1 additions, this is the total of $3n$ minus 1 arithmetic operations which are performed. Let us do see if there is a way of improving this; and this improvement is a very clever trick which is a very important trick which is called the Horner's method for evaluating the algorithm.

(Refer Slide Time: 06:10)

Work Slide

$x = 123456$
 $y = 678914$

$n = 6$

$x = (123 \times 10^3 + 456)$
 $y = (678 \times 10^3 + 914)$

$xy = (123 \times 678) \times 10^6$
 Multiplication of 2, 3 digit #'s

999
 999
 \hline
 1998

To understand the Horner's method of evaluating the algorithm, it is extremely important for us to visualize the polynomial which we are interested in evaluating in a slightly different form; in other words, we rewrite the polynomial. And how do we rewrite the polynomial? Let us just look at p of x , we rewrite it as a 0, the element, the constant term in the polynomial. And in the remaining terms, x is a common factor. If look at all the other terms, x is a common factor; that x is factored out. And the remaining polynomial is called p_1 of x .

In other words, p_1 of x is a n x power n minus 1, all the way down upto a $2x$ plus a 1. Observe that p_1 of x is obtained by taking the first n terms of p of x , and taking out the factor x from them. For example, if you go back and look at p of x , so look at p of x , it has, the first n terms of x as a factor; you just take that out and consider the remaining polynomial; and what we get is p_1 of x .

So, we have already seen that the Horner's method is an optimal method for evaluating a

given polynomial, given as the set of its coefficients in the sense that it uses exactly the required number of arithmetic operations. In other words, to evaluate the polynomial of degree n at a given point for the variable x we need definitely n multiplication operations followed by n addition operations.

And observe that we are not counting the number of operations which are required to compute the higher powers of the value v that has been given. What we have just seen is that the Horner's method uses exactly the minimum number of operations; in other words, it is an optimal algorithm for evaluating a given polynomial at a given point, or a given value for the variable x .

(Refer Slide Time: 08:50)

Multiplying n digit numbers

Multiply two n digit numbers

- 1234 and 5678
- How many single digit multiplications?
- 16, each digit in one is multiplied with all digits in the other.
- For 2 n digit numbers by the same reasoning
 n^2 is a formula for the number of single digit multiplications

Can we come up with an algorithm with a formula that has a smaller value for all n except for a few small n ?

Handwritten diagram showing 1234 and 5678x with arrows indicating digit-by-digit multiplication.

So, let us move on and ask some questions about efficient algorithms for the most simplest looking exercise of multiplying n digit numbers. Here is the exercise. We have to write a program to multiply 2 n digit numbers. And here, we deviate slightly from the module that we have been looking at so far when we assume that any 2 numbers can be multiplied in, any 2 positive integers can be multiplied in unit time. We do not, we discard that assumption and now explore the challenge of multiplying 2 n digit numbers.

In other words, the 2 numbers have n digits and we want to count the total number of single digit multiplications that are required to multiply the 2 given numbers. So, let us look at this example which we see here which is multiplying the numbers 1234 and 5678. Both of them as we can see are 4 digit numbers, and the straight forward high

school multiplication approach is to write down 1 2 3 4 5 6 7 8, and now observe that 8 has to be multiplied to each of these 4 digits, and then 7 has to be multiplied with each of these 4 digits; 6 has to be multiplied with each of the 4 digits; and 5 has to be multiplied with each of the 4 digits.

In other words, each digit in 1 must be multiplied with all the digits in the other, in the straight forward multiplication method which we are all very familiar with. And therefore, if the 2 numbers have n digits each, we essentially are performing n square multiplication operations, single digit multiplication operations. This example, it is of course, 16 multiplication operations. The question now is, can we come up with an algorithm which uses a lesser number of single digit multiplication operations? Let us try to pose this question very clearly, and that is exactly what has been written as the question at the end of the slide - can we come up with an algorithm and along with the algorithm we must write down a formula for the number of multiplication operations performed by that algorithm.

And that formula must be smaller than n square, for all values of n except for some finite numbers of n . I repeat this. The exercise now is to come up with an improved algorithm. What do we mean by an improved algorithm? We want to come up, or design and come up with, or design an algorithm; not just design the algorithm, but we also want to analyse the algorithm and come up with the formula for the number of operations which are performed by the algorithm.

What property do we desire from such a formula? We desire that the formula must be smaller than n square at all values of n , except for a few small values of n ; meaning, maybe for the first 10 numbers n , that is for upto 10 digit numbers, maybe n square is smaller than the formula that we have come up with. But, beyond 10, it must be larger than the formula that we come up with. In other words, our formula must be smaller, or the formula associated with our algorithm must be smaller. So, there are 2 tasks here - one to design the algorithm, and two to analysis the algorithm and write down the formula for the number of single digit multiplication operations that it performs.

(Refer Slide Time: 13:24)

Karatsuba's Method

Multiply x and y , two given n bit integers

Think of

$$x = x_1 10^{n/2} + x_0 \text{ and } y = y_1 10^{n/2} + y_0,$$

where x_0 and y_0 are less than $10^{n/2}$.

$$xy = (x_1 10^{n/2} + x_0)(y_1 10^{n/2} + y_0)$$
$$= z_2 10^n + z_1 10^{n/2} + z_0$$

Note that multiplication by powers of 10 is adding least significant

Is this going to help?

So, we have already seen a very straight forward method; let us do something more sophisticated and interesting and exciting. And this is the Karatsuba's method. So, let us just recall the whole exercise. We want to multiply x and y which are $2n$ digit integers; these are $2n$ digit integers. And the way we do this is to look at x and y in a slightly different way. So, what we do is we consider x to be in the form x_1 multiplied by 10 to the power of $n/2$ plus x_0 , and y to be y_1 to the power of 10 to the $n/2$ plus y_0 .

(Refer Slide Time: 14:27)

Reducing the number of multiplications

$$z_2 = x_1 y_1, \text{ both multiplicands smaller than } 10^{n/2}$$

one $n/2$ digit multiplication

$$z_1 = x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$$

one $n/2$ digit multiplication

Reason: x_1 and x_0 are both smaller than $10^{n/2}$, so both have less than $n/2$ digits, sum can have maximum of $n/2$ digits.

$$z_0 = x_0 y_0 \text{ both smaller than } 10^{n/2}$$

(Refer Slide Time: 14:28)

Gain in Arithmetic

The running time is given by the following formula which is called a *recurrence equation*

$$T(n) = 3 T(n/2), T(2)=4$$

$T(2)=4$ gives the number of single digit multiplications of two 2 digit numbers, the *boundary condition*.

Specifies the count as a formula involving the count when the number of digits is $n/2$

So, let us do a small exercise. So, let us consider x . Let us consider the x to be equal to 3 4 5 6 and y to be equal to 6 7 8 9 1 4. So, the total number of digits in both these cases is 6. We write x to be 123 multiplied by 10 to the power of 3 plus 456. Clearly this can be very easily done, right. 6 is divisible 2, right; that gives an exponent value of 3; and therefore, you can write this as 123 into 1000 plus 456. Similarly, we can write down y to be equal to 678 multiplied by 1000 plus 914.

What are we interested in? We are interested in the value x multiplied by y , which is this term multiplied by this term; this is the product of the right hand sides. I hope the approach is clear. So, the most important thing is that 123 multiplied by 1000, and 678 multiplied by 1000 are very easy to evaluate; and they do not require multiplication operations. 123 multiplied by 1000 just involves us writing down 123 followed by 3 0s. Similarly, 678 multiplied by 1000 just requires writing down 678 followed by 3 0s.

If one wants to compute the product of 123000 and 678000, then one only needs to evaluate 123 multiplied by 678. Once you have computed these, this number we had it with 6 0s. Let me write this just for the purposes of visualization in this form; that we just need to evaluate 123 multiplied by 678. Observe that we want to multiply 2 6 digit numbers, and because we know that we can pad the 6 0s, this is the multiplication of 3 digit numbers. This is the multiplication of 2 3 digit numbers. This is the whole idea that

we have reduced the size of the problem.

This is extremely important. We have reduced the size of the multiplication problem in one step by rewriting the number in this form from a 6 digit multiplication to 3 digit multiplication. But, of course, there are some tricks in world. It is not so straight forward. As you can see, there are some more multiplications to be done. So, let us go back to our slide and look at it in the formal way.

As we have just seen, we have seen that x can be written in this form and y can be written in this form; we did see this through an example. We saw this through this example. Now, let us just focus on the values of y_{naught} , y_1 , x_1 and x_{naught} . You have already seen that x_{naught} and y_{naught} are smaller than 10 to the power of n by 2 . Now, let us look at x , y , the product of we are interested in, and see how this can be rewritten.

This can be rewritten by just expanding this term out. When you try to expand this term out let us assume that the coefficients that one gets are z_2 multiplied by 10 power n ; that is a product of x_1 and y_1 is written down as z_2 . Then there is a term where the exponent is 10 to the power of n by 2 . This is the product of x_1 into y_{naught} plus x_{naught} into y_1 , and then there is product of x_{naught} and y_{naught} which we have written as z_{naught} .

We have already seen the multiplication by powers of 10 is adding least significant 0 s. Now, is this going to help in reducing the total number of multiplication operations, single digit multiplication operations? So, let us just focus on z_2 which we have already seen that is x_1 multiplied by y_1 ; z_1 which is x_1 into y_{naught} plus x_{naught} into y_1 ; and z_{naught} which is x_{naught} multiplied by y_{naught} .

Most important thing is that this value, this is 10 to the power of n by 2 ; this is selected so that in the sub problems there are just n by 2 digits that need to be multiplied; the number of, in every number that means to be multiplied, there are just n by 2 digits. Now, let us just look at the second term which is the most important thing. Observe that as of now there are 4 multiplications to be done, that is z_2 ; this is a single multiplication; z_1 , it looks here that this is one multiplication and this is the second multiplication and then this is the fourth multiplication.

However, a small trick of rewriting z_1 into x_1 plus x_{naught} into y_1 plus y_{naught} minus z_2 minus z_{naught} . Let us see this z_2 is x_1 minus x_1 y_1 , z_{naught} is x_{naught}

multiplied by y naught, if you evaluate this and subtract z 2 and z naught we get exactly what we want which is the value of z 1. And now observe that this is a single multiplication operation. And what we are going to see is that all these multiplication operations involve just n by 2 digits.

So, let us just go to the example, right. So, remember that we wanted to multiply these 2 6 digit numbers; we wrote the number down as 123 into 1000; how many digits are left now? Half the digits are left. If you notice here there are n by 2 digits. There are just 3 digits here. Similarly, there are also 3 digits here. Further observe that these 2 numbers are obtained by looking at the 3 digits of x , the first, the least significant 3 digits of x and the least 3 digits of y .

Therefore, those 2 values have to be lesser than 1000. They cannot be 1000. They have to be lesser than 1000 because they involve only 3 digits; same with this term, right. Therefore, what is the conclusion? The conclusion is that we now have 4 3 digit numbers. We started off wanting to multiply 2 6 digit numbers; we have broken it down into 4 3 digit numbers. And going back to our previous slide, we are now wanting to evaluate, we have now observed that x 1 y 1 y naught y 1, all are n by 2 digits; in that example there were 3 digits.

Now, let us just look at the reduction in the number of multiplication operations. Both, x 1 and y 1 as we have seen are smaller than 10 to the n by 2; that is they have less than n by 2 digits; same with y naught and y 1. Secondly, let us just look at the sum of the 2 of them. So, this is just 2 n by 2 digits 1, n by 2 digit multiplication; z naught which is x naught into y naught, both are smaller than n by 2. This is again 1 n by 2 multiplication.

We only have to worry about and reason about this multiplication operation. Let us just observe that x 1 and x naught both are numbers smaller than 10 to the n by 2, 10 to the power of n by 2. When you add both of them how many digits can you get? So, that is you have 2 values which are 10 to the power of n by 2, and when you add the 2 of them how many digits can you have? You can have a maximum of n by 2 digits because both of them are smaller than n by 2.

So, let us just go back to our example. Observe that 456 and 914 are 3 digit numbers. When you add both of them, at the most you can get a 4 digit number, right. So, let us just see this; you can have 999. You can have 999 a 3 digit number, added with another 999 which is another 3 digit number, this is 1998. Observe that it is a 4 digit number.

Therefore, the most important point that is being made is that if you add 914 with 678 you can at most get another 4 digit number and not a 6 digit number, right. So, that is the most important thing. So, you do not get a 6 digit number.

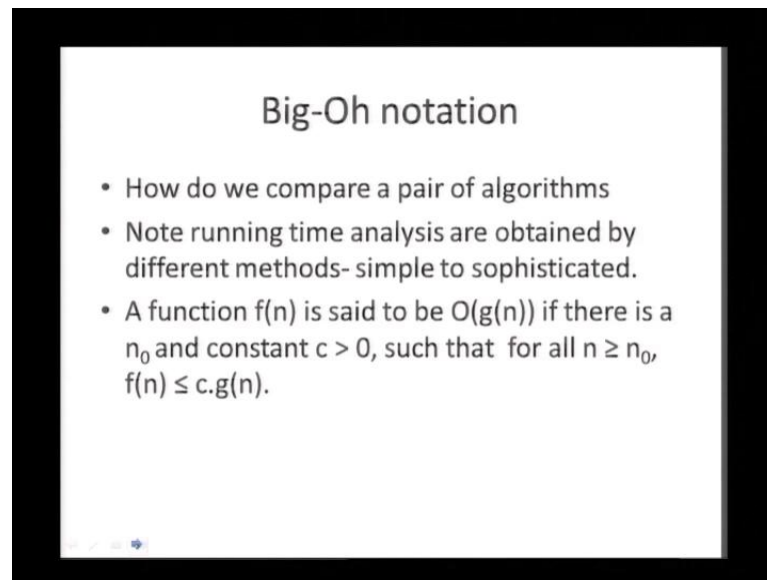
So, let us just see what the gain in the arithmetical. The gain in the arithmeticals we have 3 multiplications of n by 2 digit numbers, we have 4 additions and 2 subtractions, we have 1 multiplication by 10^n , 1 multiplication by 10^{n+2} , and the last 2 do not are not counted as multiplications, they are just shifting operations or adding 0s, least significant 0s.

So, let us just go back and calculate this. You see 1 subtraction as second subtraction, then you see 2 additions here, and then there is a third addition and a fourth addition, and there are 3 multiplication operations. So, therefore, let us formally analyse the gain in arithmetic. The running time of the algorithm that we have just written where we want to calculate the total number of multiplications that is our measure of running time, let us not count the number of additions and the shift operations, let us just count the number of multiplications.

The number of multiplications on $2n$ digit numbers is given by 3 multiplications on $2n$ by 2 digit numbers. And if you multiply 2 2 digit numbers we can take this to be equal to 4. This is easy to check. And this is called the boundary condition for this recurrence equation; what is a recurrence equation? The value for multiplying $2n$ digit numbers. The number of multiplications, single digit multiplications to multiply $2n$ digit numbers is given by 3 times the number of multiplications is to multiply $2n$ by 2 digit numbers.

The solution for this is given by n to the power of $\log_3 2$, the solution for this is given by the value n to the power of $\log_3 2$, and observe that $\log_3 2$, n to the power of $\log_3 2$, and observe that $\log_3 2$ is smaller than 2. Therefore, $n^{\log_3 2}$ is smaller than n^2 ; and n^2 is a total number of single digit multiplications in the simple method.

(Refer Slide Time: 29:55)

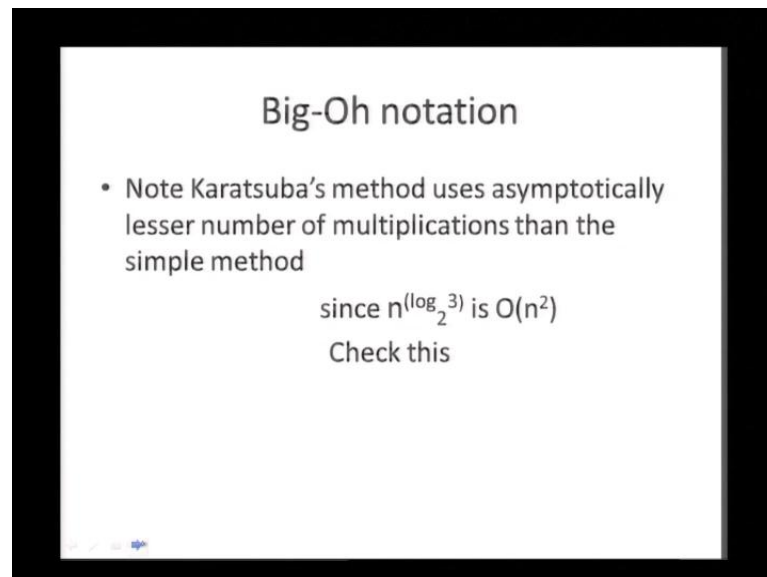


Big-Oh notation

- How do we compare a pair of algorithms
- Note running time analysis are obtained by different methods- simple to sophisticated.
- A function $f(n)$ is said to be $O(g(n))$ if there is a n_0 and constant $c > 0$, such that for all $n \geq n_0$, $f(n) \leq c.g(n)$.

This bring us to the last module here which is the concept called a big oh notation where the focus is how do we compare a pair of algorithms. Observe that we have done analysis of running time of these algorithms by different methods; we use simple methods and sophisticated methods.

(Refer Slide Time: 30:20)



Big-Oh notation

- Note Karatsuba's method uses asymptotically lesser number of multiplications than the simple method

since $n^{(\log_2 3)}$ is $O(n^2)$
Check this

So, let us just look at the Karatsuba's method. We know that the Karatsuba's method uses 10 to the power of log 3 base 2 multiplications, and this is smaller than n square. How does one capture this? We capture this using the big oh notation which will be the focus

of the next lecture.

Thank you.