

Cs 5050

02 11 14

①  $f(1) = 1$

$$f(n) = a f(n/b) + c n^k$$

# of recursive calls →  $a$   
how the problem gets smaller →  $n/b$

work done in function  
Split/merge time  
problem decomposition + solution construction time →  $c n^k$

Standard form D & C.

$f(n)$  is the number of steps when problem size is  $n$

$f(n)$  is the number of steps the algorithm takes to solve a problem of size  $n$

Most divide and conquer algorithms have this form.

The generate  $a$  recursive calls, each of problem size  $n/b$ .

At each call they require  $cn^k$  steps to decompose the problem into smaller problems and/or combine the sub solutions together.

(2) if	$a > b^k$	$f(n) = n^{\log_b a}$	alg.	Solution	a	b	k
			merge sort	$n \log n$	2	2	1
	$a = b^k$	$f(n) = n^k \log n$	$k^{\text{th}}$ rank algorithm	$n$	1	2	1
	$a < b^k$	$f(n) = n^k$					

The solution “cook book”

there are three cases based on the relationship among the parameters a, b and k

To solve a given recurrence relation, first extract the parameters a, b and k from the relation  
 second determine which case applies, third substitute the parameters in the solution form

3

## Polynomial Multiplication

Given

$$P = \sum_{i=0}^{n-1} p_i x^i$$

$$Q = \sum_{i=0}^{n-1} q_i x^i$$

$$P = p_0 + p_1 x + p_2 x^2 + p_3 x^3$$

Find

$$P \times Q$$

highest order term

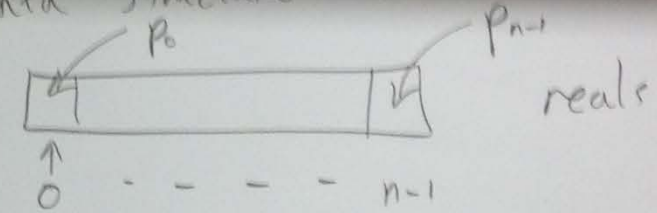
Q is  $n-1$

P is  $n-1$

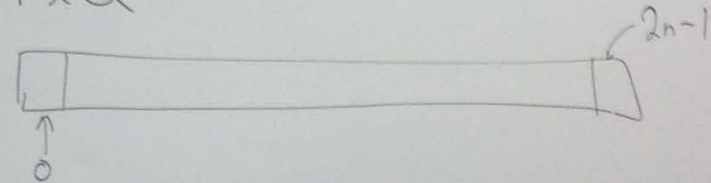
PQ is  $2n-2$

have  $2n-1$  coefficients

Data Structure



$P \times Q$



A fundamental problem to study: given two polynomials P and Q determine the polynomial that is the product of multiplying P and Q

Data structure: a 1D array where the index into the array is the power of x for that location, so  $P[0]$  is the coefficient for  $x^0$

$$(4) (p_0 + p_1x + p_2x^2)(q_0 + q_1x + q_2x^2)$$

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ p_0q_0 & p_1q_0 + q_1p_0 & p_1q_1 + p_0q_2 + q_0p_2 & p_1q_2 + q_1p_2 & p_2q_2 \end{array}$$

4 ← power of x  
index in to  
the coefficient array.

Given  $n$  # of terms  
P array  $[0 \dots n-1]$   
Q  $[0 \dots n-1]$   
return  $P \times Q$

two nested for loops  
PQ = new array size  $2n$  // assume initialized 0  
for  $i = 0$  to  $n-1$   
  for  $j = 0$  to  $n-1$   
    PQ[i+j] += P[i] \* Q[j]  
  end  
end

$n^2$  algorithm

$$n = 10^5$$

$$n^2 = 10^{10}$$

Example poly multiply problem where P and Q have three terms.

Algorithm is two nested for loops where each  $p_i$  times  $q_j$  goes into location  $i+j$  in the solution polynomial

Algorithm is  $n^2$  since we have two nested loops each of size  $n$ .

5

$$P = \sum_{i=0}^{n-1} p_i x^i$$

$$P = P_L + x^{n/2} P_H$$

$$P_L = \sum_{i=0}^{n/2-1} p_i x^i$$

$$P_H = \sum_{i=0}^{n/2-1} (p_{i+n/2}) x^i$$

$$P = \underbrace{[p_0 \dots p_{n/2-1}]_{P_L}}_{P_L} x^{n/2} \underbrace{[p_{n/2} \dots p_{n-1}]_{P_H}}_{P_H}$$

2 polynomials with  $n/2$  terms

Can we do better? Does not appear so. Try Divide and Conquer. Split each poly into low and high order terms

⑥  $PQ = (P_L + x^{n/2} P_H)(Q_L + x^{n/2} Q_H)$

$PQ = \underbrace{P_L Q_L}_{\text{polymult}} + \underbrace{(P_L Q_H + P_H Q_L)}_{\text{only need the sum!}} x^{n/2} + P_H Q_H x^n$

$f(1) = 1$

$f(n) = 4f(n/2) + n^1$

$f(n) = n^{\log_2 4} = n^2$

Algorithm  
 polymult( $P, Q, n$ )  
 if  $n=1$  return  $P_0 \times Q_0$   
 else  
 $P_L \leftarrow P[0 \dots n/2-1]$   
 $P_H \leftarrow P[n/2 \dots n-1]$   
 $Q_L$   
 $Q_H$   
 compute  
 answer

Diagram illustrating the recursive splitting of polynomials  $P$  and  $Q$  into low ( $L$ ) and high ( $H$ ) parts:

- $P$  is split into  $P_L$  (low) and  $P_H$  (high).
- $Q$  is split into  $Q_L$  (low) and  $Q_H$  (high).
- The product  $PQ$  is computed as  $P_L Q_L + (P_L Q_H + P_H Q_L) x^{n/2} + P_H Q_H x^n$ .
- The diagram shows the recursive splitting of  $P$  and  $Q$  into  $L$  and  $H$  parts, with the final result being the sum of the products of these parts.

Multiply the two divided polynomials and show that to compute  $PQ$  we generate 4 problems of  $\frac{1}{2}$  size  
 The recurrence relation counts the number of steps.  
 When we solve using the cook book we get order  $n^2$







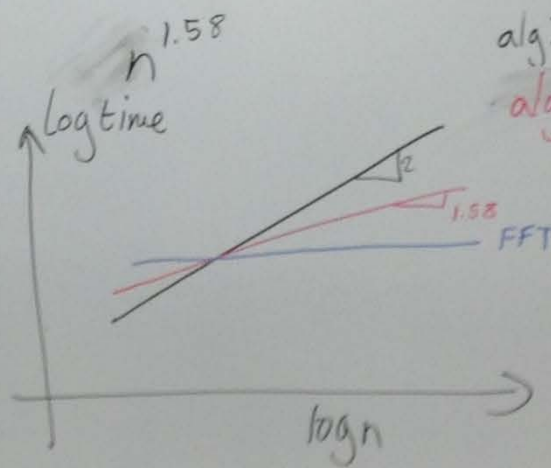
3 sub problems of  $1/2$  size

$$f(1) = 1$$

$$f(n) = 3f(n/2) + n$$

a                  b                  c

$$f(n) = n^{\log_2 3}$$



$$\text{alg1} \approx n^2$$
$$\text{alg2} \approx n^{1.58}$$

With 3 subproblems we get an algorithm that takes about  $n^{1.58}$  rather than  $n^2$   
Much more efficient for large  $n$