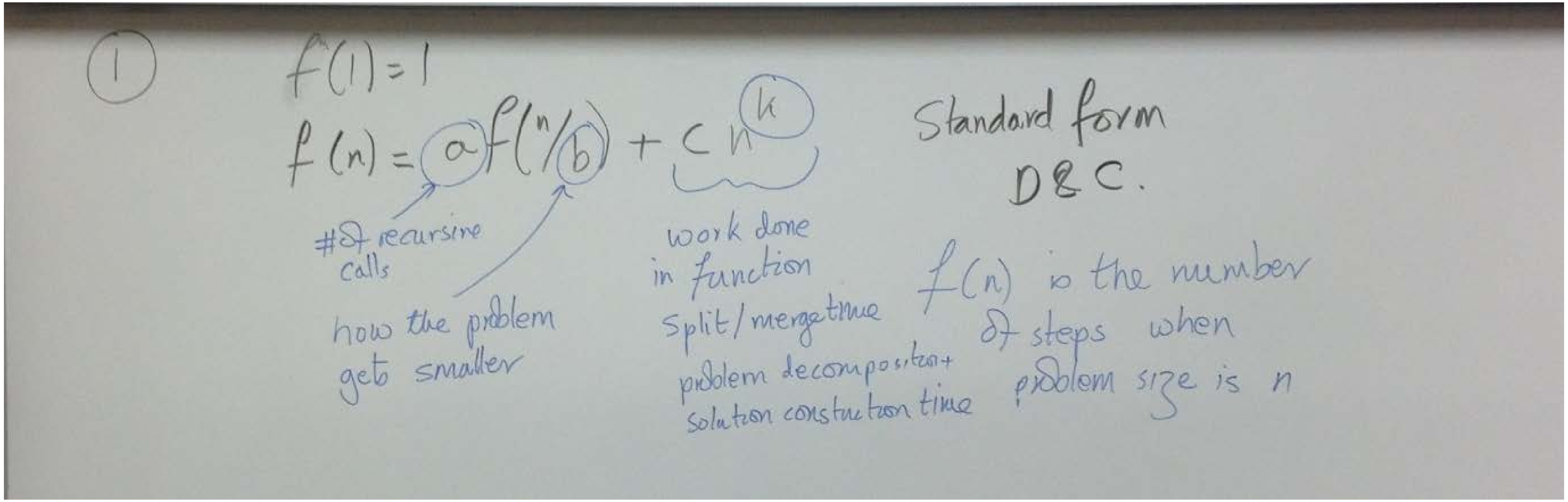
Cs 5050

02 11 14



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

$a > b^k$ $\quad f(n) = n^{\log_b a}$

$a = b^k$ $\quad f(n) = n^k \log n$

$a < b^k$ $\quad f(n) = n^k$

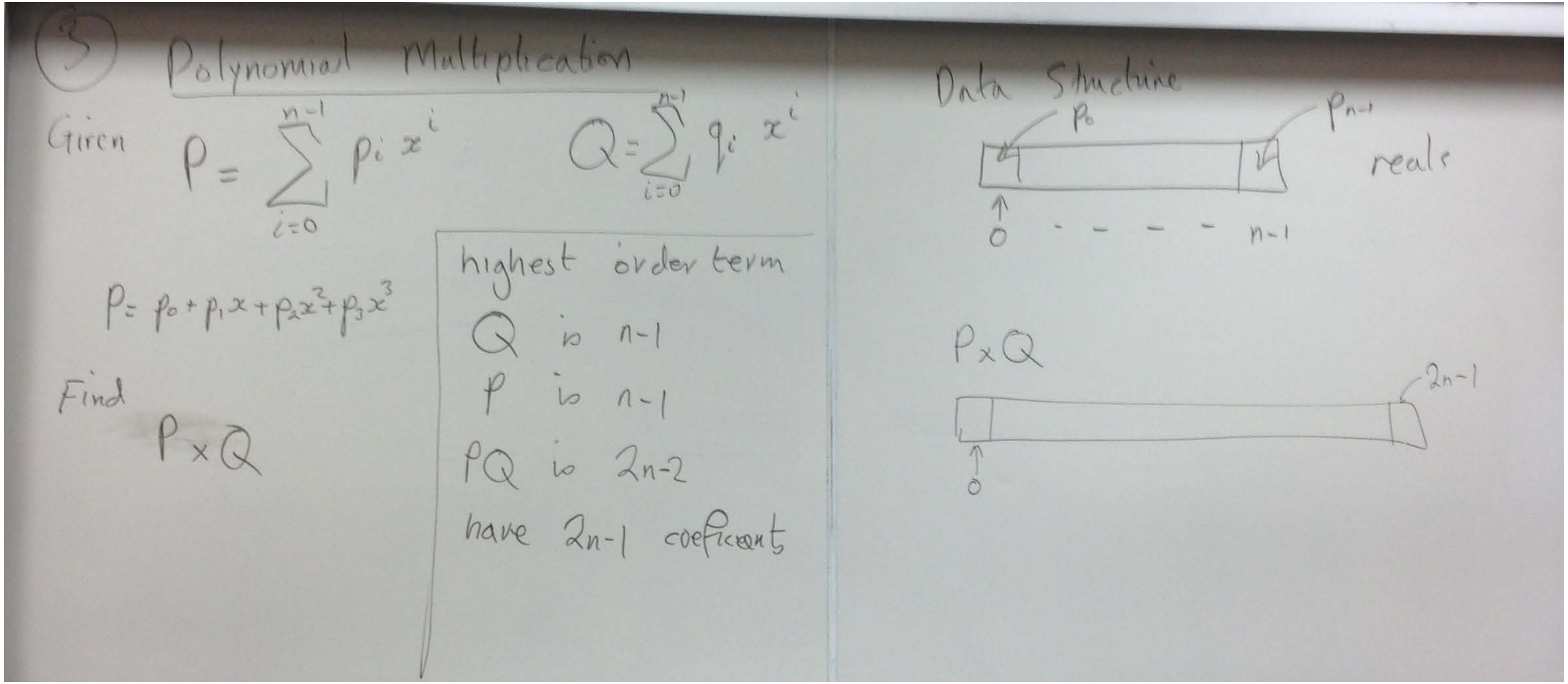
alg.	Solution	a	b	k
merge sort	$n \log n$	2	2	1
k^{th} rank algorithm	n	1	2	1

The solution “cook book”

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

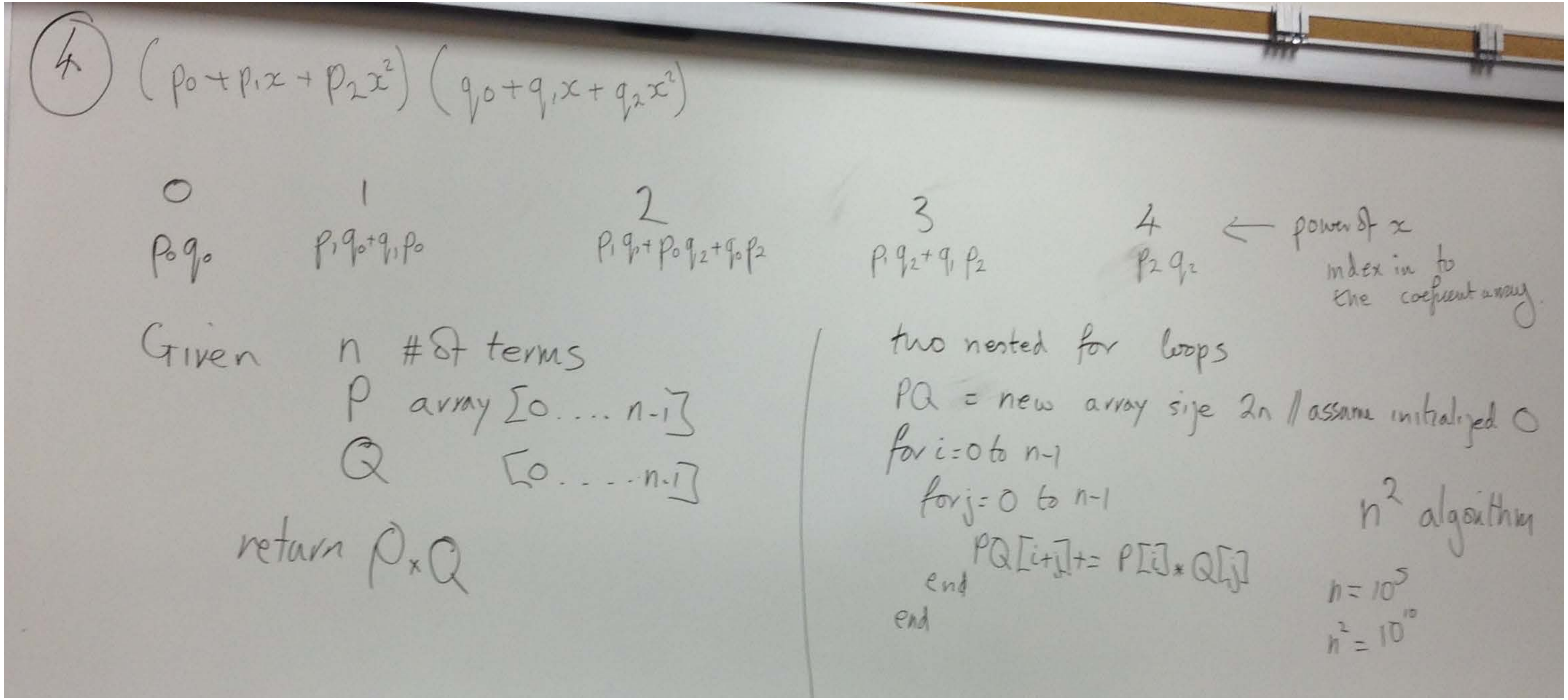
To solve a given recurrence relation, first extract the parameters a, b and k from the relation

second determine which case applies, third substitute the parameters in the solution form



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



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

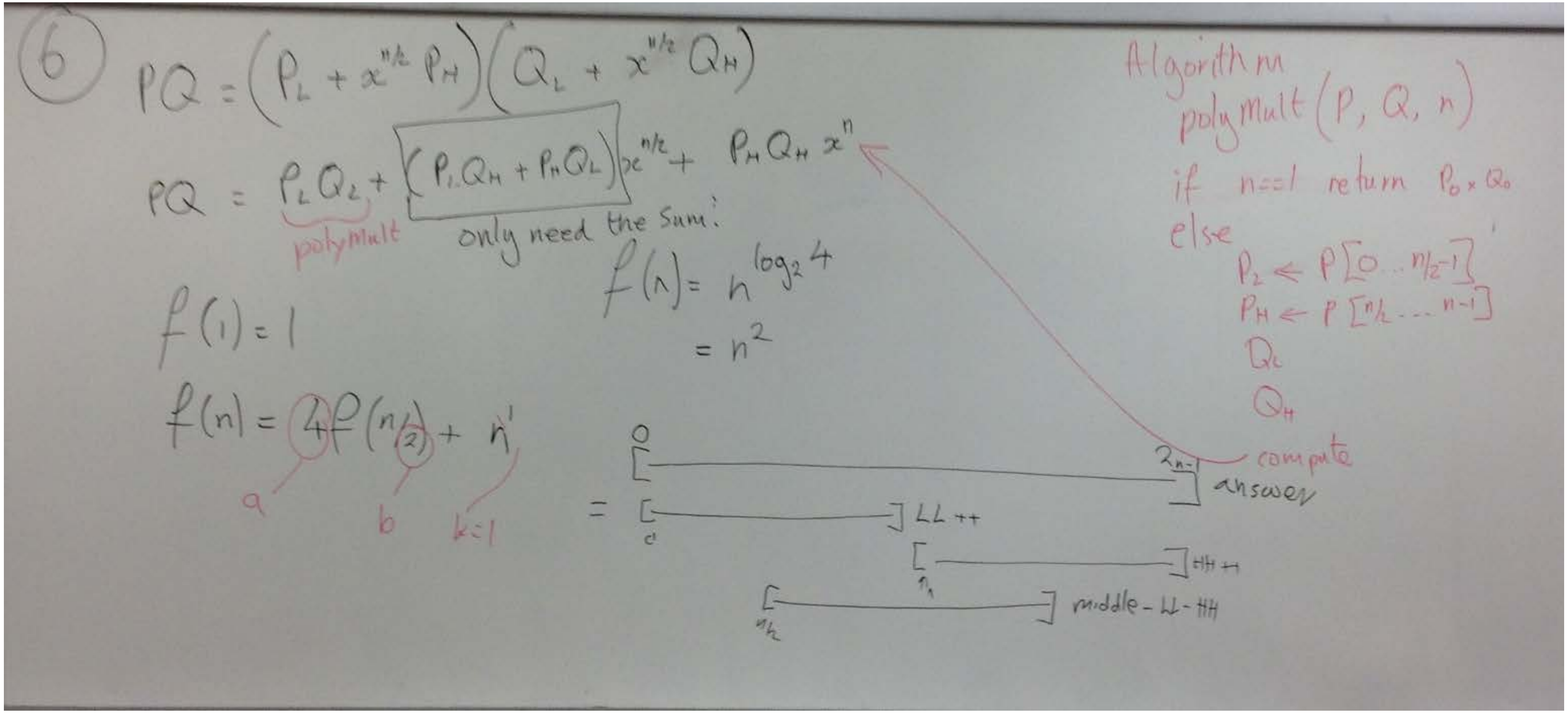
$$P = \sum_{i=0}^{n-1} p_i x^i \qquad P_L = \sum_{i=0}^{i=n/2-1} p_i x^i$$

$$P = P_L + x^{n/2} P_H \qquad P_H = \sum_{i=}^{i=n/2-1} \left(p_{i+n/2}\right) x^i$$

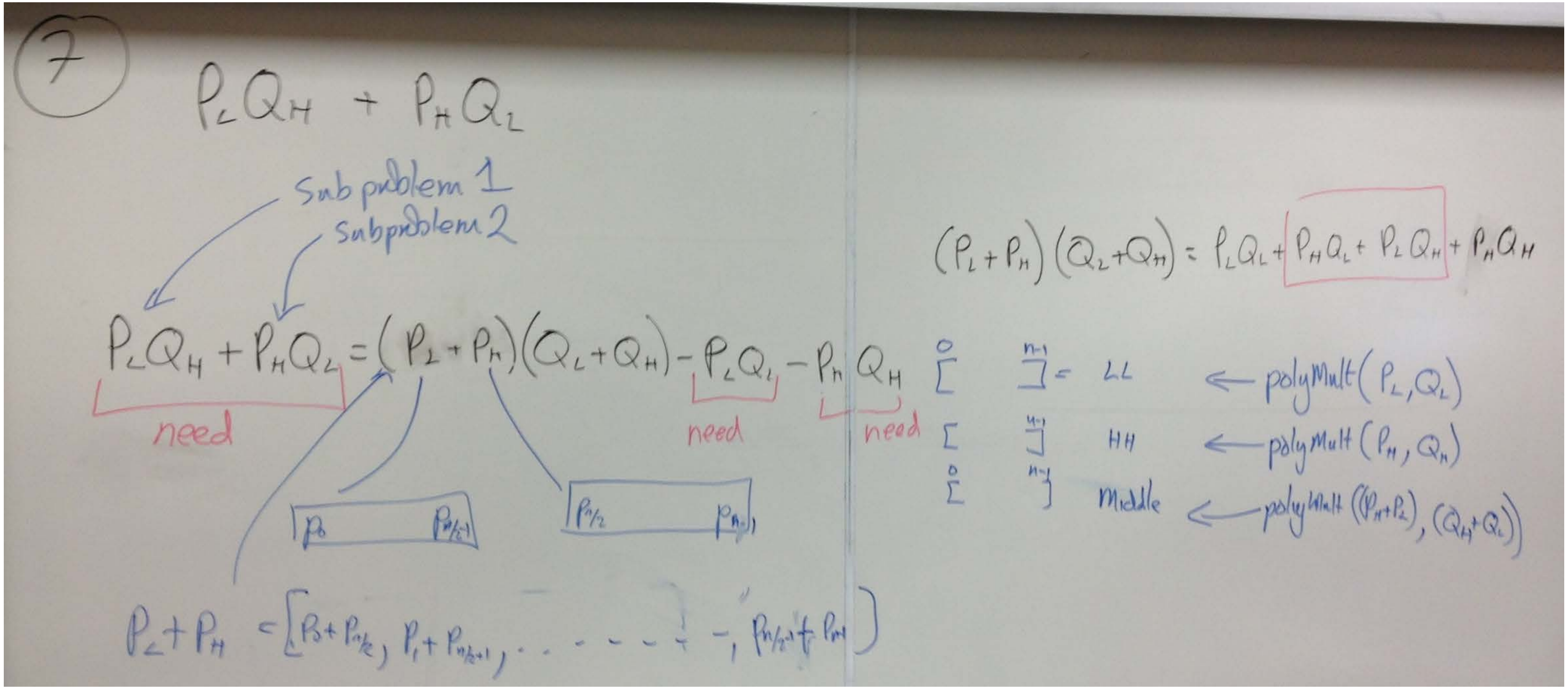
$$P = \underbrace{\left[p_0 \cdots\cdots\cdots p_{n/2-1} x^{n/2-1}\right]}_{P_L} \underbrace{\left[p_{n/2} x^{n/2} \qquad p_{n-1} x^{n-1}\right]}_{P_H}$$

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



Multiply the two divided polynomials and show that to compute PQ we generate 4 problems of ½ size
The recurrence relation counts the number of steps.
When we solve using the cook book we get order n^2



To improve efficiency need to generate less subproblems!

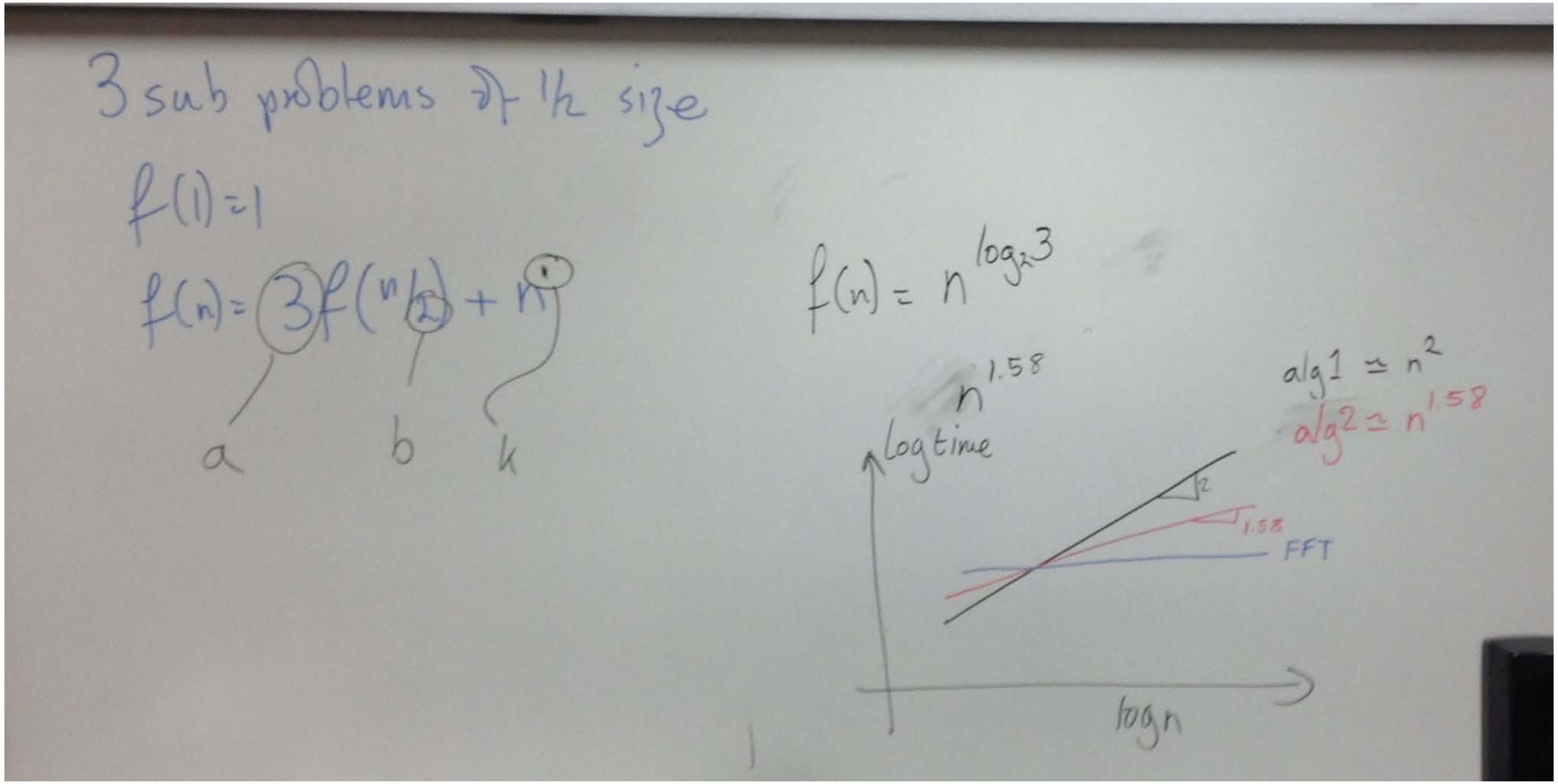
Notice in the previous decomposition we only need the sum: $P_LQ_H+P_HQ_L$ not each individual solution!

Try adding then multiplying See right upper expression.

Note that adding the high order coefficients to the low order coefficients is very counter-intuitive!

See lower left expression.

Now we have only three sub problems to solve



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