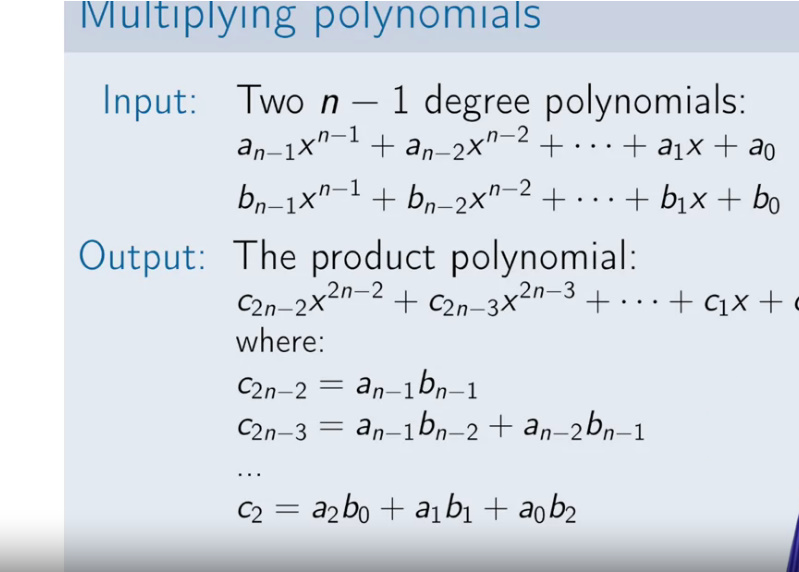
**Polynomial Multiplication**

* In general inseamna sa luam 2 polinoame si sa le inmultim





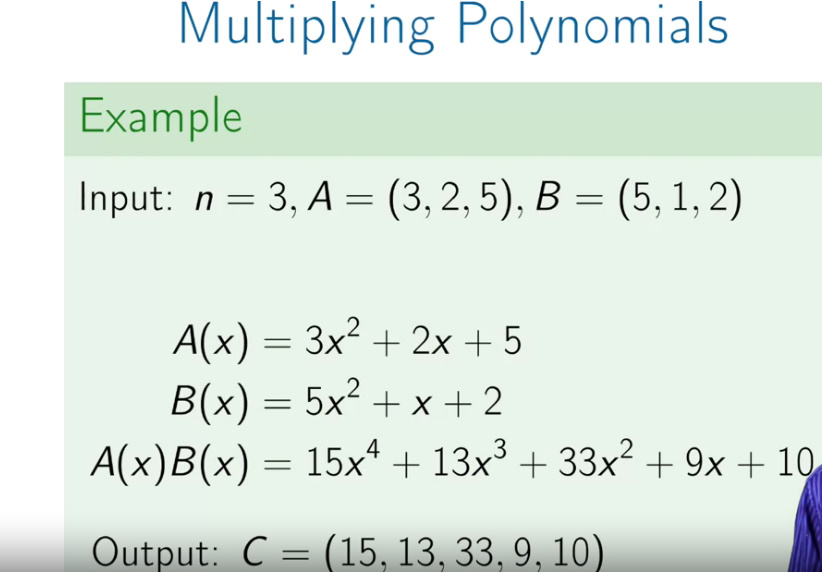
* coeficientul lui c mereu incepe de la 2n – 2 si scade cu 1 mereu pana ajungem la x^0, adica la numarul fara x.

2n – 2 = (n-1) + (n-1) = 2(n-1) = 2n – 2

2n – 3 = n-1 + n – 2 = n – 3

Deci, pentru orice c, trebuie sa avem orice suma de produse, unde fiecare produs are coeficientii ce in suma dau coeficientul lui c.

x ia acelasi coeficient

* 

c(2n - a) – unde **a** incepe de la 2, caci si pozitiile ambelor pot incepe de la 1, si deci doar 2 poate fi 1 + 1, daca ele au ambele pozitii cel putin.

c(2n-2) e primul element dupa inmultirea celor 2 polinoame, ce va avea, de ex pentru n = 10, puterea 2\*10 – 2 = 18, asa cum ambele array vor avea primul element cu puterea 9 si 9 + 9 = 18

* c(2n-2) = a(n-1)b(n-1) = 3x^2 \* 5x^2 = 15x^4

Atentie, a(n-1) se refera la primul element din polinom,care are puterea n-1, a(n-2) la al doilea care are puterea n-2 si tot asa. c(2n-2) se refera la primul termen obtinut dupa multiplicarea polinoamelor.

* c(2n-3) = a(n-1)b(n-2) + a(n-2)b(n-1) = 3x^2\*x + 2x\*5x^3 = 3x^3 + 10x^3 = 13x^3
* c(2n-4) = a(n-2)b(n-2) + a(n-3)b(n-1) + a(n-1)b(n-3) = 2x\*x + 5\*5x^2 + 3x^2\*2 = 2x^2 + 25x^2 + 6x^2 = 33x^2
* c(2n-5) = a(n-2)b(n-3) + a(n-3)b(n-2) = 2x\*2 + 5\*x = 9x

Atentie! Nu putem alege a(n-4)b(n-1) deoarece polinoamele nu au 4 elemente si nici nu e asa putere la vreun termen.

* c(2n – 6) = a(n-3)b(n-3) = 5\*2 = 10

Ex2:

3x + 4

x + 2

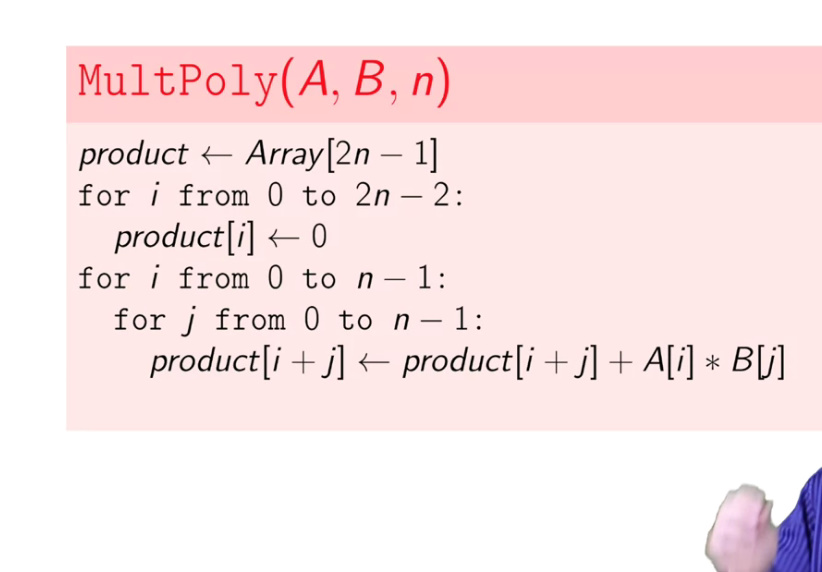
c(2n-2) = a(n-1)b(n-1) = 3x\*x = 3x^2

c(2n-3) = a(n-1)b(n-2) + a(n-2)b(n-1) = 3x\*2 + 4\*x = 6x + 4x = 10x

c(2n-4) = a(n-2)b(n-2) = 4\*2 = 8

(3x + 4 )\*(x + 2) = 3x^2 + 10x + 8

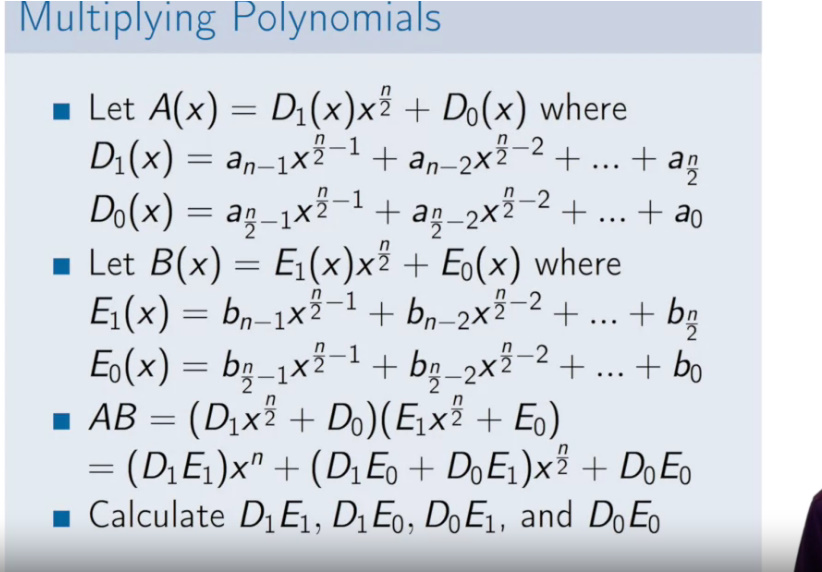
**Naive Algoritm**





* Daca cel mai lung polinom are n termeni, rezultatul mereu va avea 2n – 1 termeni. De ex, mai sus cand polinomnele aveau 2 termeni, produsul lor era format din 2\*2 – 1 = 3 termeni.
* Daca vrem un polinom cu 10 elemente, inseamna ca elementul cu cel mai mare grad va fi primul, x^9, nu x^10! Caci avem si un element cu x^0 la urma
* Acest algoritm se bazeaza pe simpla inmultire a fiecarui termen din primul poligon cu celelalte din al 2 poligon si aflarea sumei lor. Insa, ne asiguram ca fiecare pozitie se aduna cu coeficientul ce are aceesi putere la x.

**Naive Dive and Conquer**



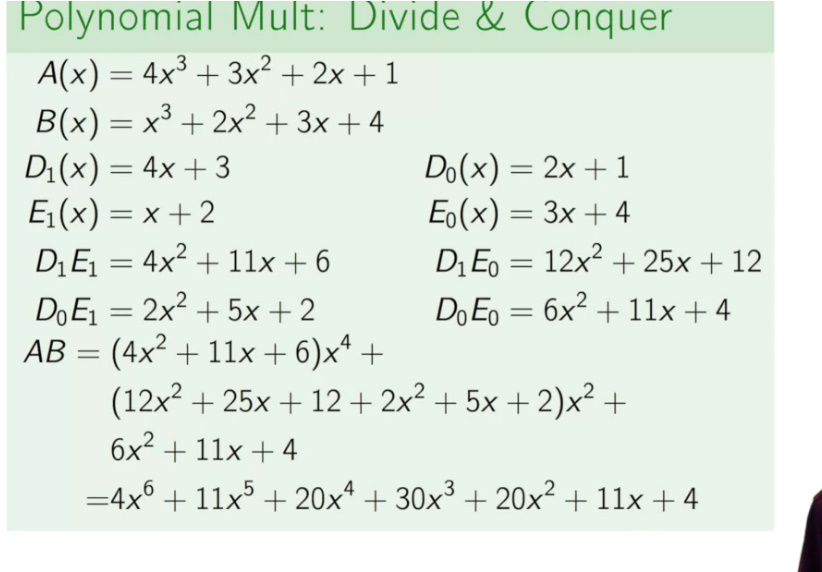


* Divizam fiecare polinom in 2 parti.



* n se refera la nr total de termeni in polinom
* n-1 se refera la primul element din polinom,adica cu x la puterea n-1,cea mai mare putere, n – 2 la al doilea, adica cu x la puterea n - 2 si tot asa
* n/2 – 1 se refera la primul element dupa cel din mijloc, care are x la o putere identica cu x a elementului din mijloc – 1, si apoi la urmatorul dupa el va fi n/2 – 2, adica cu 2 puteri mai mici de cat cea a elementului din mijloc si tot asa.
* Partea din stange e D1, din dreapta D0

Recurenta: 4T(n/2) – 4 + kn deoarece avem 4 submultimi, adica de la fiecare polinom cate 2 jumatati, si kn arata de cate ori se fac aceleasi operatii la jumatati de k ori a cate n instructiuni

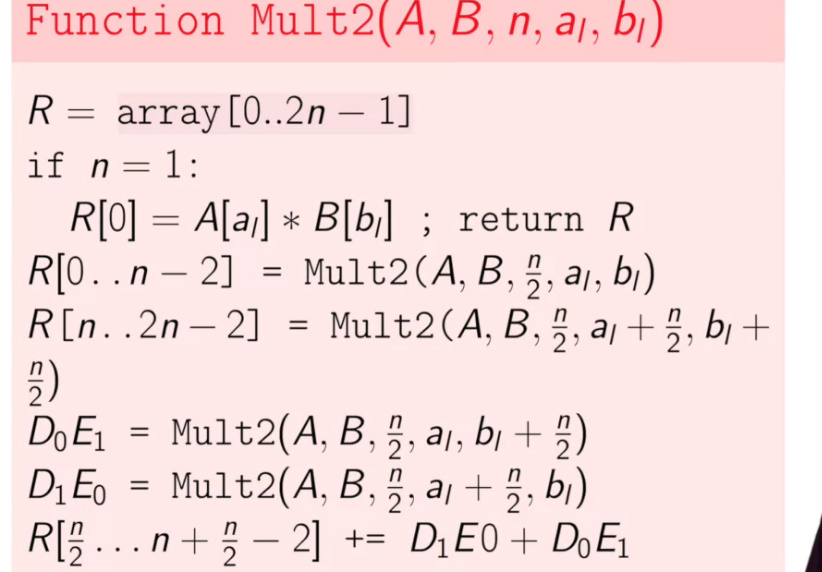
n

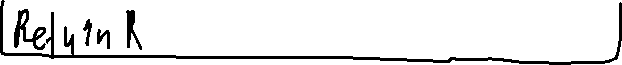
n = 4

D1(x) = 4x^(n/2 - 1) + 3x^(n/2 - 2) = 4x^(1,5-1) + 3x^(1,5-2)= 4x^1 + 3x^0 = 4x + 3

D0(x) = 2x(n/2 - 1) + x(n/2 - 2) = 2x^(2 - 1) + x^(2 – 2 ) = 2x + 1

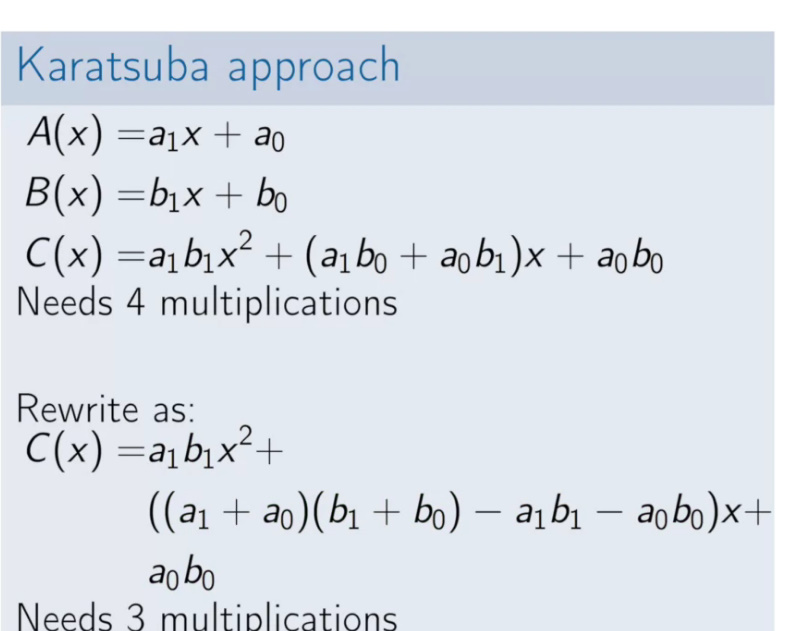
Cand vom face apel la functia data prima oara, a1 si b1 vor fi 0 ambele.

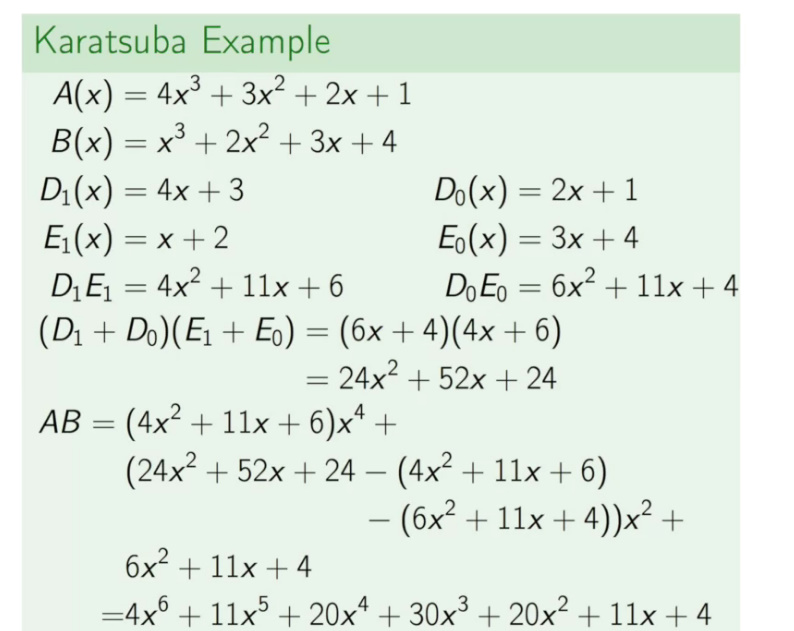
**

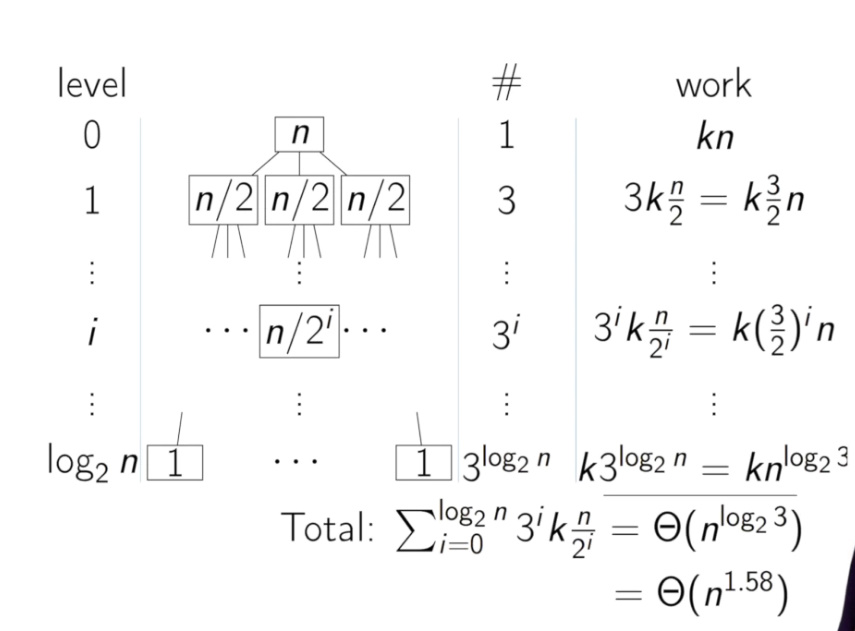


avem R[0..n – 2] si R[n..2n-2], deoarece avem nevoie ca elementul n – 1 sa ramana intact.

**Faster Divice and Conquer**

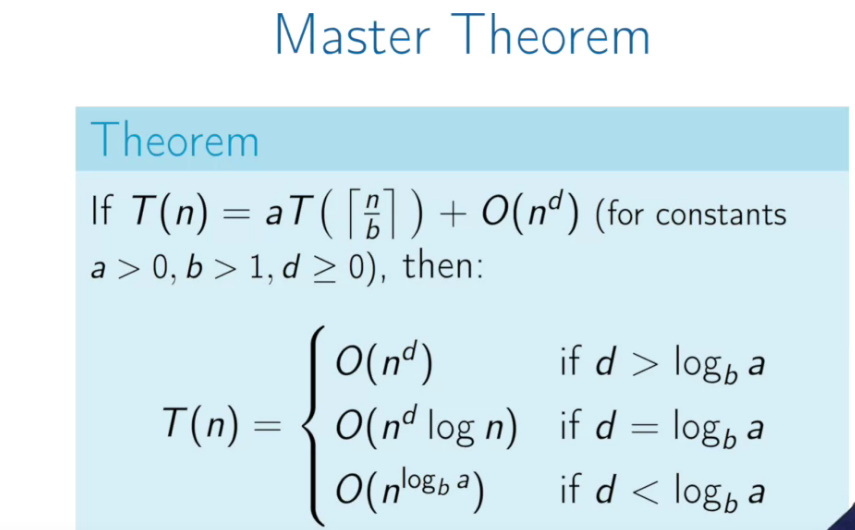


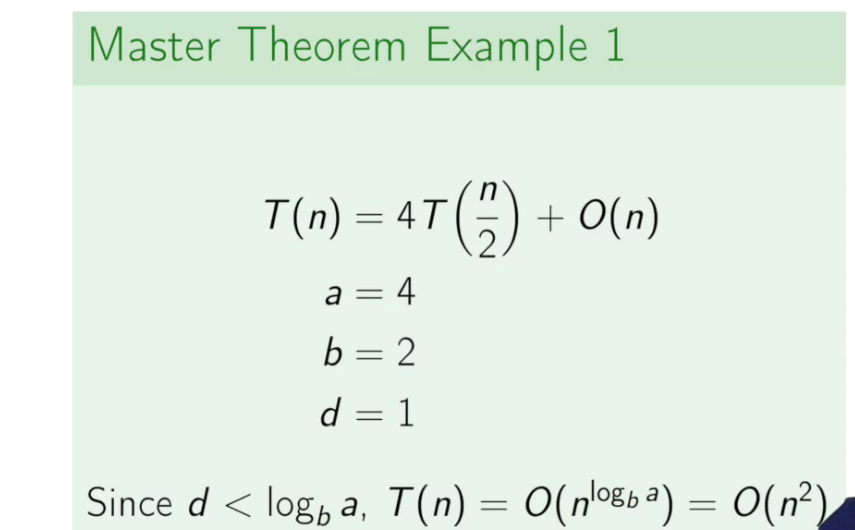


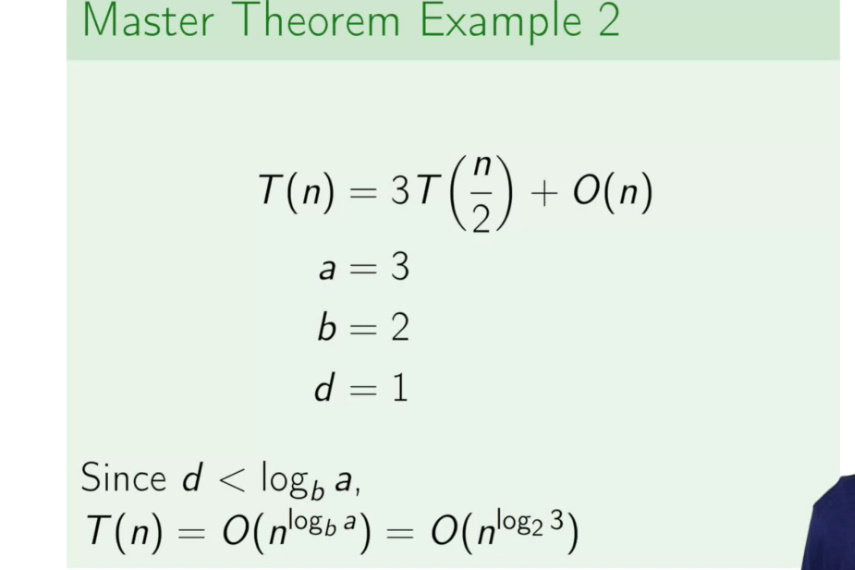


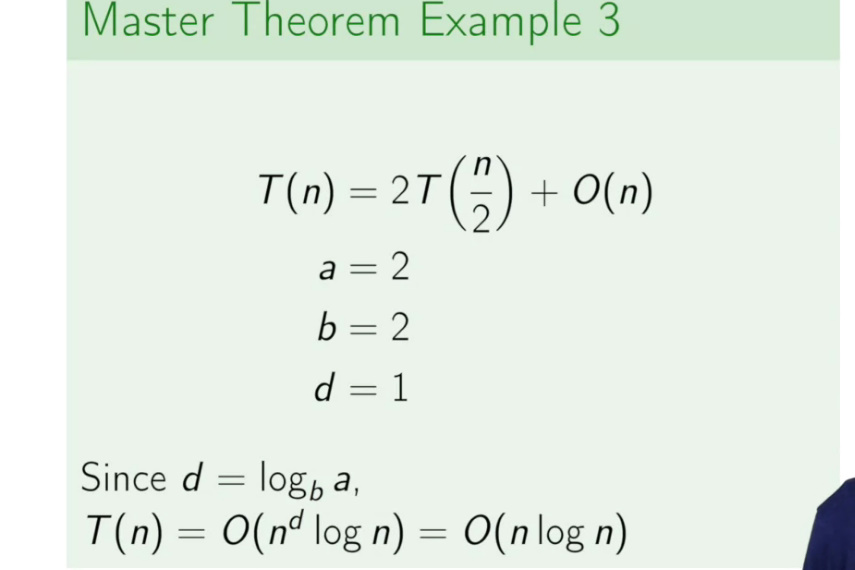
**Master Theorem**

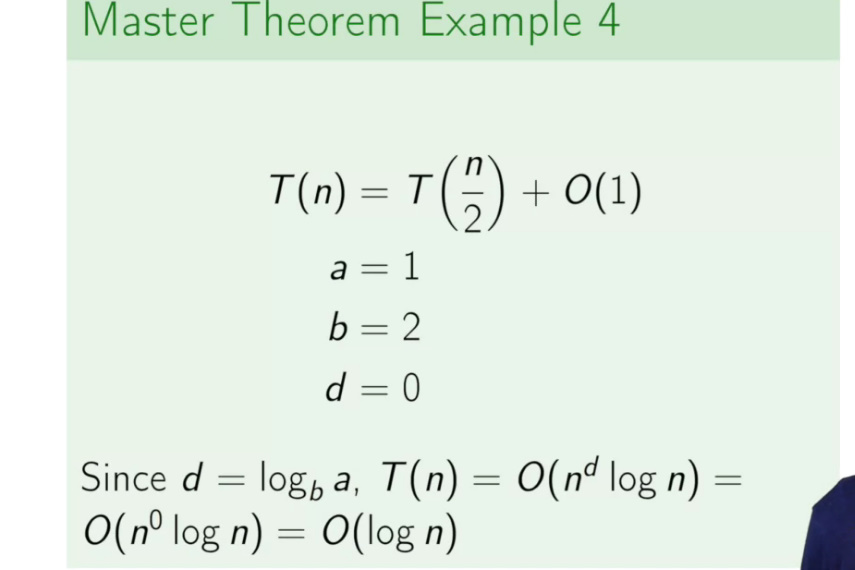
E folosita pentru a transforma T(n) in O(n)

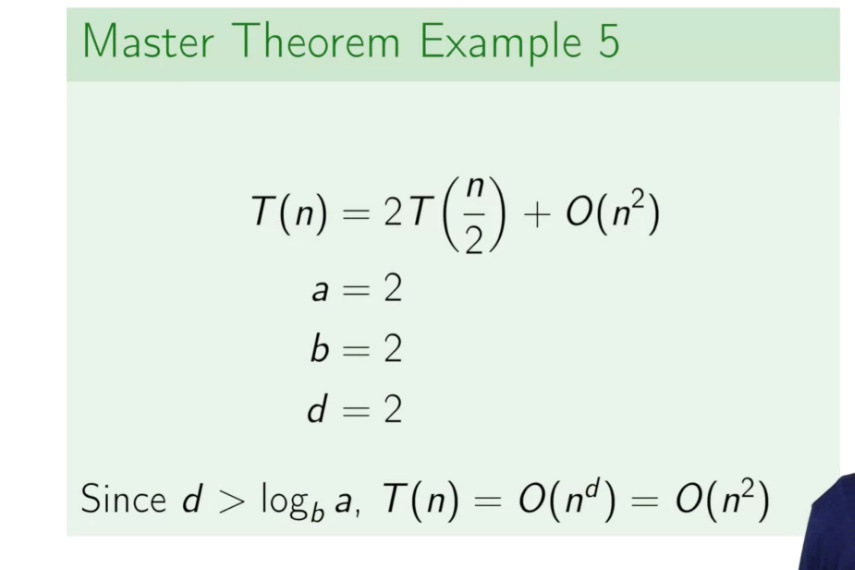




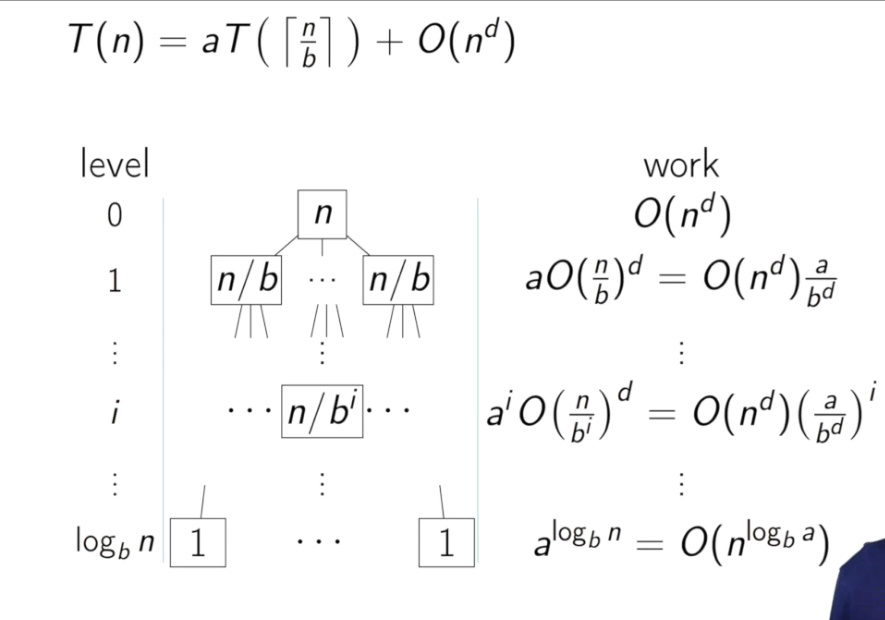


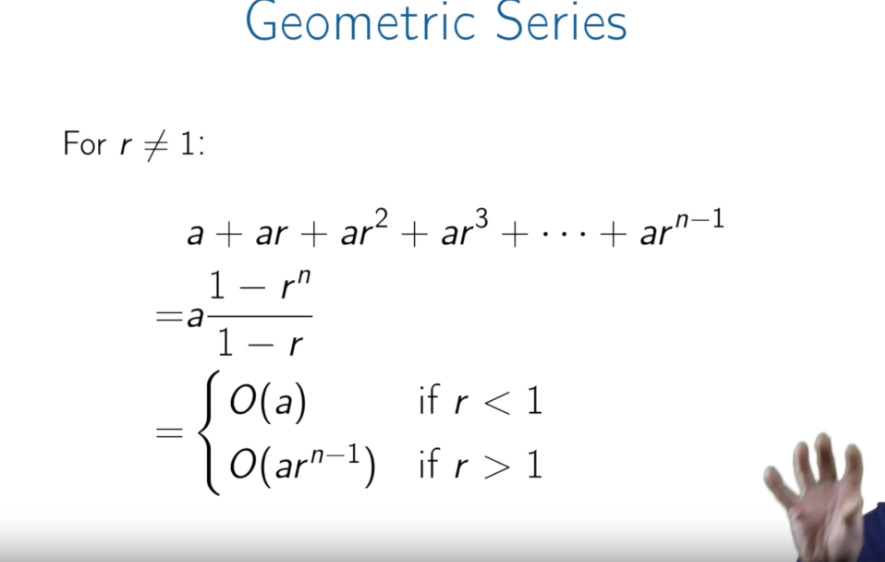


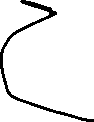
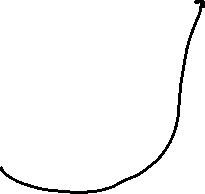


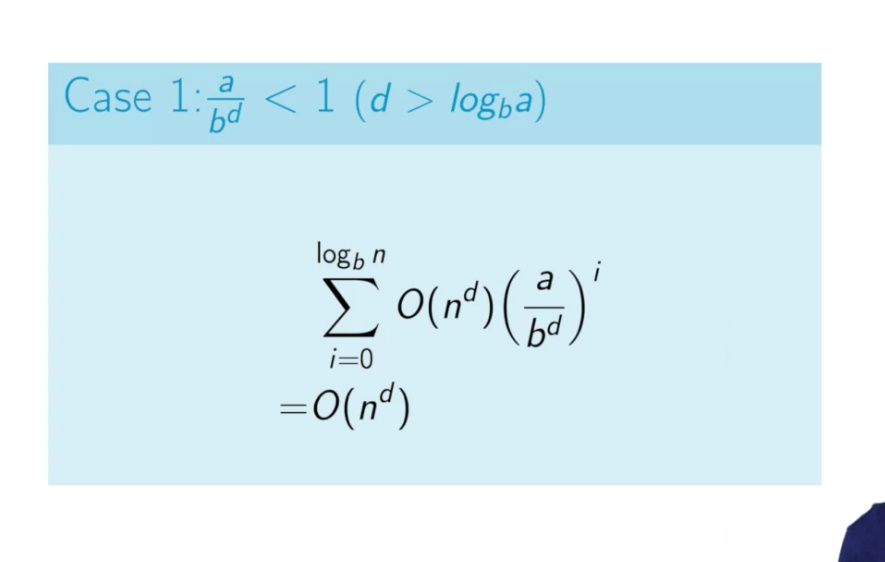
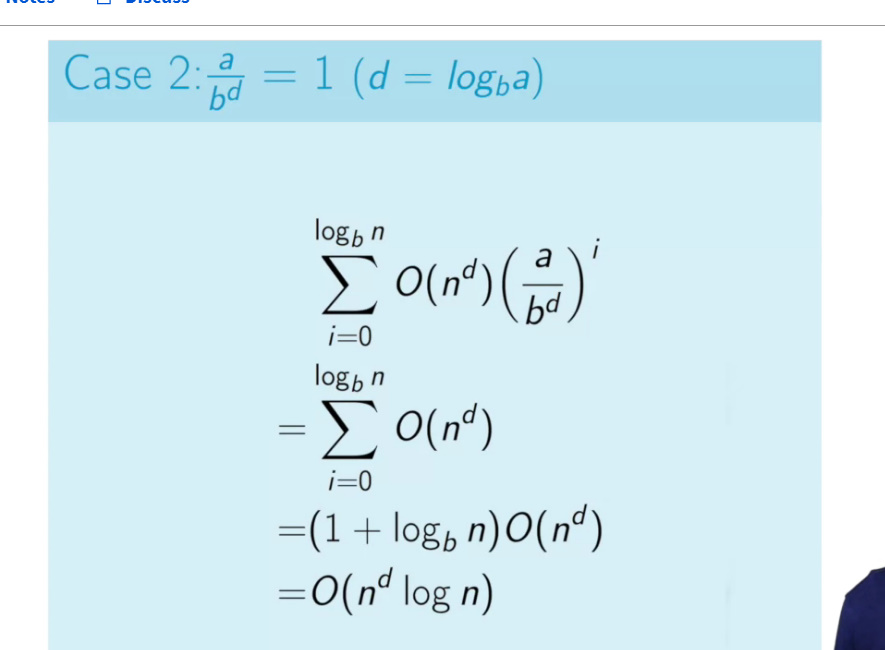


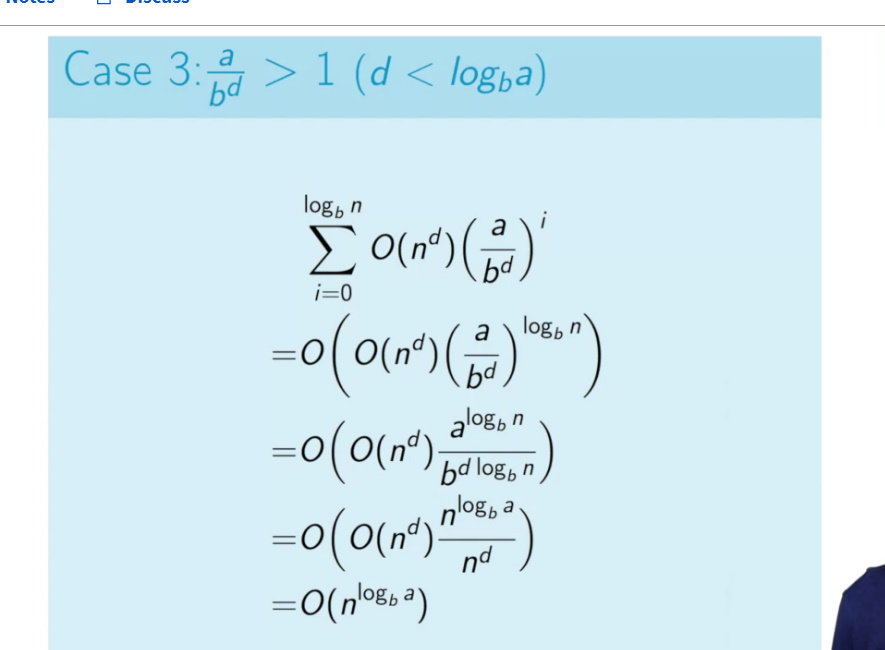
**Demonstrare**

****

****



****

****