I2EA

reviewing techniques from previous lectures

the plan

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting and Order Statistics
3. Probabilistic Analysis of Algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited
13. Some Elementary Number Theory
14. Applications of Number Theory
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

the plan

some real action and fun

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting ad Order Statistics
3. Probabilistic Analysis of Algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited
13. Some Elementary Number Theory
14. Applications of Number Theory
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

the plan

some real action and fun

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting ad Order Statistics
3. Probabilistic Analysis of Algorithms ← probability spaces for algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets ← Tarjan's Analysis of Union-Find
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited ← Schoenhage-Strassen-Multiplication (simplified)
13. Some Elementary Number Theory
14. Applications of Number Theory ← public key cryptography
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

the plan

some real action and fun

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting ad Order Statistics
3. Probabilistic Analysis of Algorithms ← probability spaces for algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets ← Tarjan's Analysis of Union-Find
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited ← Schoenhage-Strassen-Multiplication (simplified)
13. Some Elementary Number Theory
14. Applications of Number Theory ← public key cryptography
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

cannot be done in the given time
unless……

the plan

some real action and fun

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting ad Order Statistics
3. Probabilistic Analysis of Algorithms ← probability spaces for algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets ← Tarjan's Analysis of Union-Find
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited ← Schoenhage-Strassen-Multiplication (simplified)
13. Some Elementary Number Theory
14. Applications of Number Theory ← public key cryptography
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

cannot be done in the given time
unless……
…..you already are familiar with the topic

the plan

some real action and fun

1. Reviewing results and techniques from previous lectures: Oh-Notation, Recursion, Master-Theorem, high performance arithmetic
2. Sorting ad Order Statistics
3. Probabilistic Analysis of Algorithms ← probability spaces for algorithms
4. Balanced Trees: 2/3-trees and AVL trees
5. Heaps and Matrix Algorithms
6. Amortized Analysis of Algorithms and Pattern Matching
7. Recursion Theory and Maintaining Disjoint Sets ← Tarjan's Analysis of Union-Find
8. Graph Algorithms
9. Network Algorithms
10. Hashing
11. Fast Fourier Transform and Multiplication of Polynomials
12. Integer Multiplication Revisited ← Schoenhage-Strassen-Multiplication (simplified)
13. Some Elementary Number Theory
14. Applications of Number Theory ← public key cryptography
15. Greedy Algorithms, Dynamic Programming, Circuit Complexity Revisited

cannot be done in the given time
unless……
…..you already are familiar with the topic

and you are!

The conventional way to start ‚efficient algorithms' lectures:

- O-notation
- divide and conquer
- zooming in
- difference equations
- master theorem

The conventional way to start ‚efficient algorithms' lectures:

- O-notation
- divide and conquer
- zooming in
- difference equations
- master theorem

you know almost all of the above
we review….

The conventional way to start ‚efficient algorithms' lectures:

- O-notation
- divide and conquer
- zooming in
- difference equations
- master theorem

you know almost all of the above
we review….

high performance circuits are efficient hardware algorithms

O-Notation: growth of complexity for large *n*

Consider functions

$$f, g : \mathbb{N} \to \mathbb{R}$$

We write $f(n) = O(g(n))$ if for large n (greater than some n_0) and some positive constant c we have $f(n) \leq c \cdot g(n)$.

$$f(n) = O(g(n)) \leftrightarrow (\exists n_0 \in \mathbb{N}\ ,\ c > 0.\ \forall n \geq n_0.\ \ f(n) \leq c \cdot g(n))$$

O-Notation: growth of complexity for large *n*

Consider functions

$$f, g : \mathbb{N} \to \mathbb{R}$$

We write $f(n) = O(g(n))$ if for large n (greater than some n_0) and some positive constant c we have $f(n) \leq c \cdot g(n)$.

$$f(n) = O(g(n)) \leftrightarrow (\exists n_0 \in \mathbb{N}\ ,\ c > 0.\ \forall n \geq n_0.\ \ f(n) \leq c \cdot g(n))$$

- $$f(n) = 5 \cdot n + 17 \quad , \quad g(n) = n$$

  Then

$$\begin{aligned} 5 \cdot n + 17 \quad & \not\leq \quad 6 \cdot n \\ \leftrightarrow 17 \quad & \leq \quad n \\ 5 \cdot n + 17 \quad & = \quad O(n) \quad \text{with} \quad c = 6 \quad \text{and} \quad n_0 = 17 \end{aligned}$$

O-Notation: growth of complexity for large *n*

Consider functions

$$f, g : \mathbb{N} \to \mathbb{R}$$

We write $f(n) = O(g(n))$ if for large n (greater than some n_0) and some positive constant c we have $f(n) \leq c \cdot g(n)$.

$$f(n) = O(g(n)) \leftrightarrow (\exists n_0 \in \mathbb{N}\ ,\ c > 0.\ \forall n \geq n_0.\ \ f(n) \leq c \cdot g(n))$$

- $$f(n) = a \cdot n + b \quad , \quad g(n) = n$$

  Then

$$\begin{aligned} a \cdot n + b \quad & \leq (a+1) \cdot n \\ \leftrightarrow b \quad & \leq n \\ a \cdot n + b \quad = \quad & O(n) \quad \text{with} \quad c = a+1 \quad \text{and} \quad n_0 = b \end{aligned}$$

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n))\}$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

Proof. Let

$$\begin{aligned} f_1(n) &\leq c_1 \cdot g_1(n) \quad \text{for} \quad n \geq n_1 \\ f_2(n) &\leq c_2 \cdot g_2(n) \quad \text{for} \quad n \geq n_2 \end{aligned}$$

Then

$$\begin{aligned} d \cdot f_1(n) &\leq (d \cdot c_1) \cdot g(n) \quad \text{for} \quad n \geq n_1 \\ f_1(n) + f_2(n) &\leq (c_1 + c_2) \cdot \max\{g_1(n), g_2(n)\} \quad \text{for} \quad n \geq \max\{n_1, n_2\} \\ f_1(n) \cdot f_2(n)) &\leq (c_1 \cdot c_2) \cdot g_1(n) \cdot g_2(n) \quad \text{for} \quad n \geq \max\{n_1, n_2\} \end{aligned}$$

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

applied slowly and carefully

$$\begin{aligned}
((4 \cdot n + 3) + (6 \cdot n - 4)) \cdot n^2 &= ? \\
4 \cdot n + 3 &= O(n) \\
6 \cdot n - 4 &= O(n) \\
(4 \cdot n + 3 + 6 \cdot n - 4) &= O(n) \\
n^2 &= O(n^2) \\
(4 \cdot n + 3 + 6 \cdot n - 4) \cdot n^2 &= O(n^3)
\end{aligned}$$

faster:

use O(g(n)) in expressions as:

'a function f(n) satisfying f(n) = O(g(n))'

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

applied slowly and carefully

$$\begin{aligned}
((4 \cdot n + 3) + (6 \cdot n - 4)) \cdot n^2 &= ? \\
4 \cdot n + 3 &= O(n) \\
6 \cdot n - 4 &= O(n) \\
(4 \cdot n + 3 + 6 \cdot n - 4) &= O(n) \\
n^2 &= O(n^2) \\
(4 \cdot n + 3 + 6 \cdot n - 4) \cdot n^2 &= O(n^3)
\end{aligned}$$

faster:

use O(g(n)) in expressions as:

'a function f(n) satisfying f(n) = O(g(n))'

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

$$\begin{aligned} d \cdot O(g_1(n)) &= O(g_1(n)) \\ O(g_1(n)) + O(g_2(n)) &= O(\max\{g_1(n), g_2(n)\} \\ O(g_1(n)) \cdot O(g_2(n)) &= O(g_1(n) \cdot g_2(n)) \end{aligned}$$

applied slowly and carefully

$$\begin{aligned} ((4 \cdot n + 3) + (6 \cdot n - 4)) \cdot n^2 &= ? \\ 4 \cdot n + 3 &= O(n) \\ 6 \cdot n - 4 &= O(n) \\ (4 \cdot n + 3 + 6 \cdot n - 4) &= O(n) \\ n^2 &= O(n^2) \\ (4 \cdot n + 3 + 6 \cdot n - 4) \cdot n^2 &= O(n^3) \end{aligned}$$

faster:

use O(g(n)) in expressions as:

'a function f(n) satisfying f(n) = O(g(n))'

Lemma: Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ and $d \geq 0$

Then

- $$d \cdot f_1(n) = O(g_1(n))$$
- $$(f_1(n) + f_2(n)) = O(\max\{g_1(n), g_2(n)\})$$
- $$f_1(n) \cdot f_2(n) = O((g_1(n) \cdot g_2(n))$$

$$\begin{aligned} d \cdot O(g_1(n)) &= O(g_1(n)) \\ O(g_1(n)) + O(g_2(n)) &= O(\max\{g_1(n), g_2(n)\} \\ O(g_1(n)) \cdot O(g_2(n)) &= O(g_1(n) \cdot g_2(n)) \end{aligned}$$

applied slowly and carefully

$$\begin{aligned} ((4 \cdot n + 3) + (6 \cdot n - 4)) \cdot n^2 &= ? \\ 4 \cdot n + 3 &= O(n) \\ 6 \cdot n - 4 &= O(n) \\ (4 \cdot n + 3 + 6 \cdot n - 4) &= O(n) \\ n^2 &= O(n^2) \\ (4 \cdot n + 3 + 6 \cdot n - 4) \cdot n^2 &= O(n^3) \end{aligned}$$

$$\begin{aligned} ((4 \cdot n + 3) + (6 \cdot n - 4)) \cdot n^2 &= (O(n) + O(n)) \cdot n^2 \\ &= O(n) \cdot n^2 \\ &= O(n^3) \end{aligned}$$

warning!

faster:

use O(g(n)) in expressions as:

‘a function f(n) satisfying
f(n) = O(g(n))’

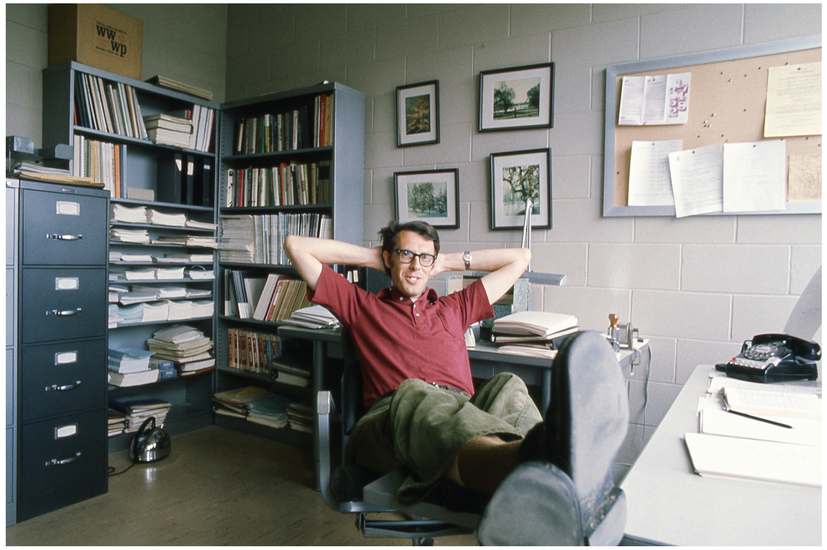
f(n) = O(g(n)) is a relation between functions f and g

= is NOT the usual equality sign

$$2 \cdot n = O(n), n = O(n), n \neq 2 \cdot n$$

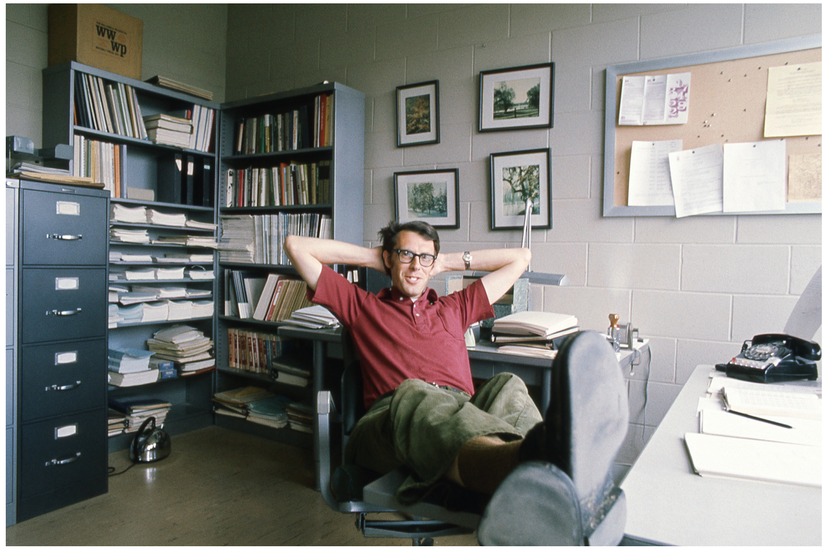
Asymptotic complexity, i.e. measuring complexity in this coarse but simple and fast way was invented in 1964 by Hartmanis and Lewis when creating the field of computational complexity.

In 1971 Hopcoroft and Tarjan reused the notation when creating the field of efficient algorithms.



Asymptotic complexity, i.e. measuring complexity in this coarse but simple and fast way was invented in 1964 by Hartmanis and Lewis when creating the field of computational complexity.

In 1971 Hopcoroft and Tarjan reused the notation when creating the field of efficient algorithms.



promise: you will see the work of these gentlemen in your lectures soon!

a few more definitions

f grows asymptotically

- at most as g

$$f(n) = O(g(n)) \leftrightarrow \exists c > 0, n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) \leq c \cdot g(n)$$

- at least as g

$$f(n) = \Omega(g(n)) \leftrightarrow \exists c > 0, n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) \geq c \cdot g(n)$$

- as g

$$f(n) = \Theta(g(n) \leftrightarrow f(n) = O(g(n) \wedge f(n) = \Omega(g(n))$$

- less than g

$$f(n) = o(g(n)) \leftrightarrow \forall c > 0\ \exists\ n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) \leq c \cdot g(n)$$

- more than g

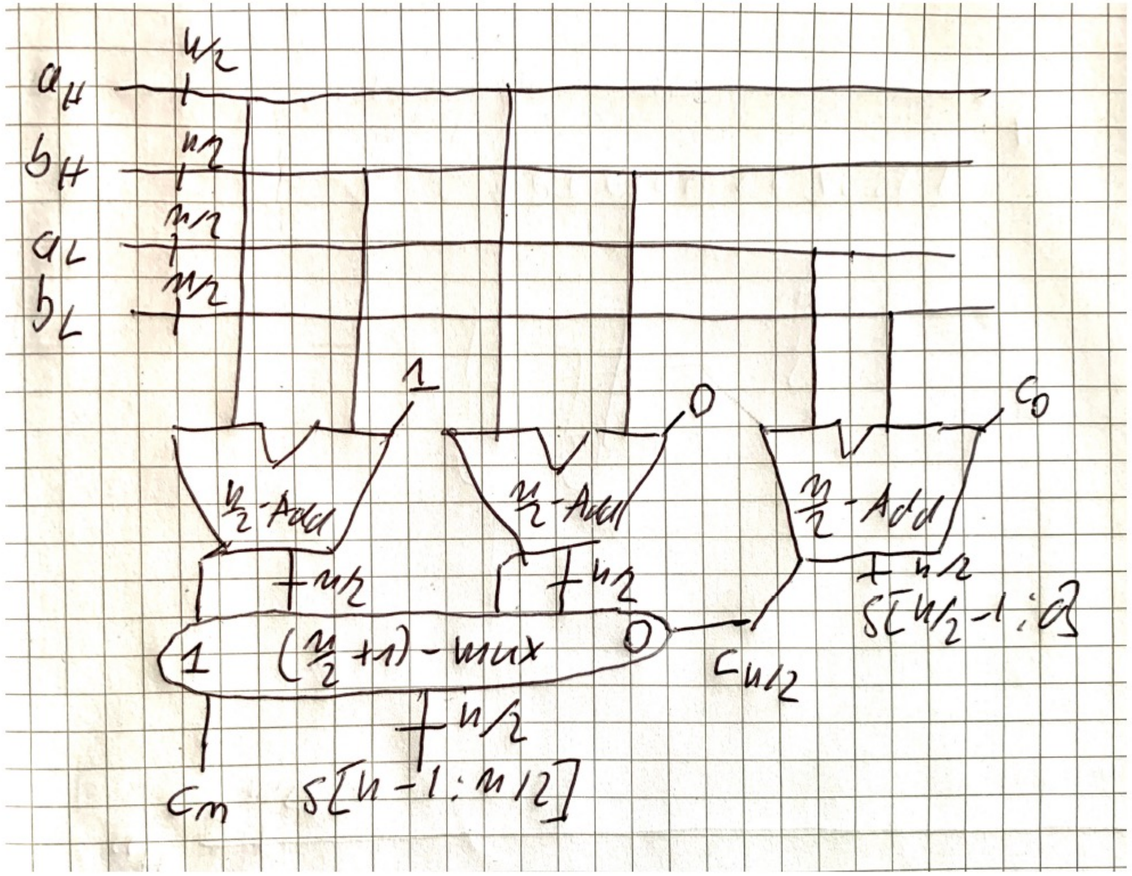
$$f(n) = \omega(g(n)) \leftrightarrow \forall c > 0\ \exists\ n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) \geq c \cdot g(n)$$

divide and conquer

- divide problem into smaller subproblems
- solve subproblems
- get solution of original problem from solution of subproblems
- usually applied recursively

divide and conquer

- divide problem into smaller subproblems
- solve subproblems
- get solution of original problem from solution of subproblems
- usually applied recursively

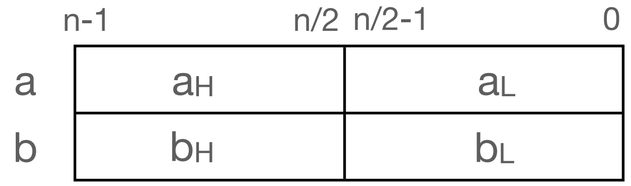


- I2CA exercise 6
- conditional sum adder

Karatsuba Multiplier

I2CA slide set 20

$$
\begin{aligned}
a_H &= a[n-1:n/2] \\
a_L &= a[n/2-1:0] \\
b_H &= b[n-1:n/2] \\
b_L &= b[n/2-1:0]
\end{aligned}
$$



$$
\begin{aligned}
\langle a \rangle \cdot \langle b \rangle &= (\langle a_H \rangle \cdot 2^{n/2} + \langle a_L \rangle) \cdot (\langle b_H \rangle \cdot 2^{n/2} + \langle b_L \rangle \\
&= A \cdot 2^n + B \cdot 2^{n/2} + C \\
A &= \langle a_H \rangle \cdot \langle b_H \rangle \\
C &= \langle a_L \rangle \cdot \langle b_L \rangle \\
B &= \langle a_H \rangle \cdot \langle b_L \rangle + \langle a_L \rangle \cdot \langle b_H \rangle \\
\longrightarrow &= (\langle a_H \rangle + \langle a_L \rangle) \cdot (\langle b_H \rangle + \langle b_L \rangle) - A - C \\
\langle a_H \rangle + \langle a_L \rangle \, , \langle b_H \rangle + \langle b_L \rangle &\in B_{n/2+1}
\end{aligned}
$$

divide and conquer

- divide problem into smaller subproblems
- solve subproblems
- get solution of original problem from solution of subproblems
- usually applied recursively

- I2CA exercise 6
- conditional sum adder

telescoping

- reduce problem of size n to 1 problem of size cn, c<1
- solve subproblem
- get solution of original problem from solution of subproblem
- usually applied recursively

telescoping

wiring

$$x'_i = x_{2i+1} \circ x_{2i},$$
$$y_{2i} = x_{2i} \circ y'_{i-1},$$
$$y_{2i+1} = y'_i.$$

- I2CA slide set 18
- parallel prefix circuit

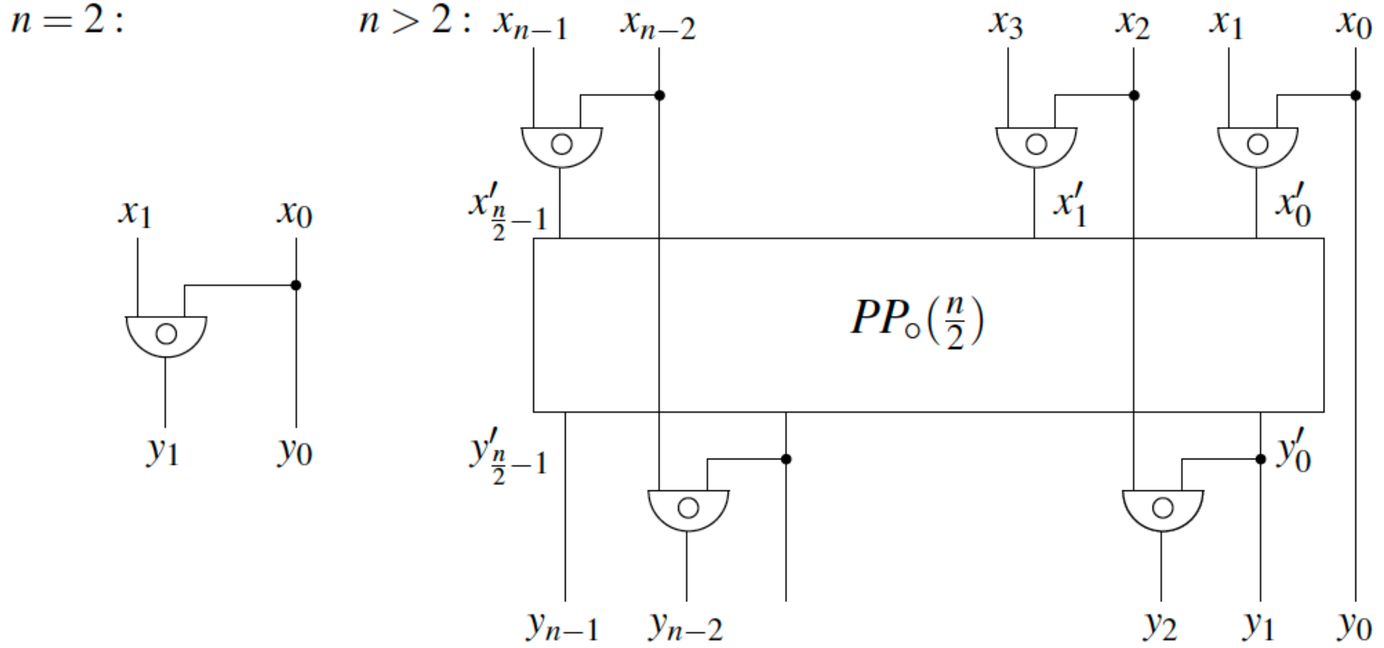


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

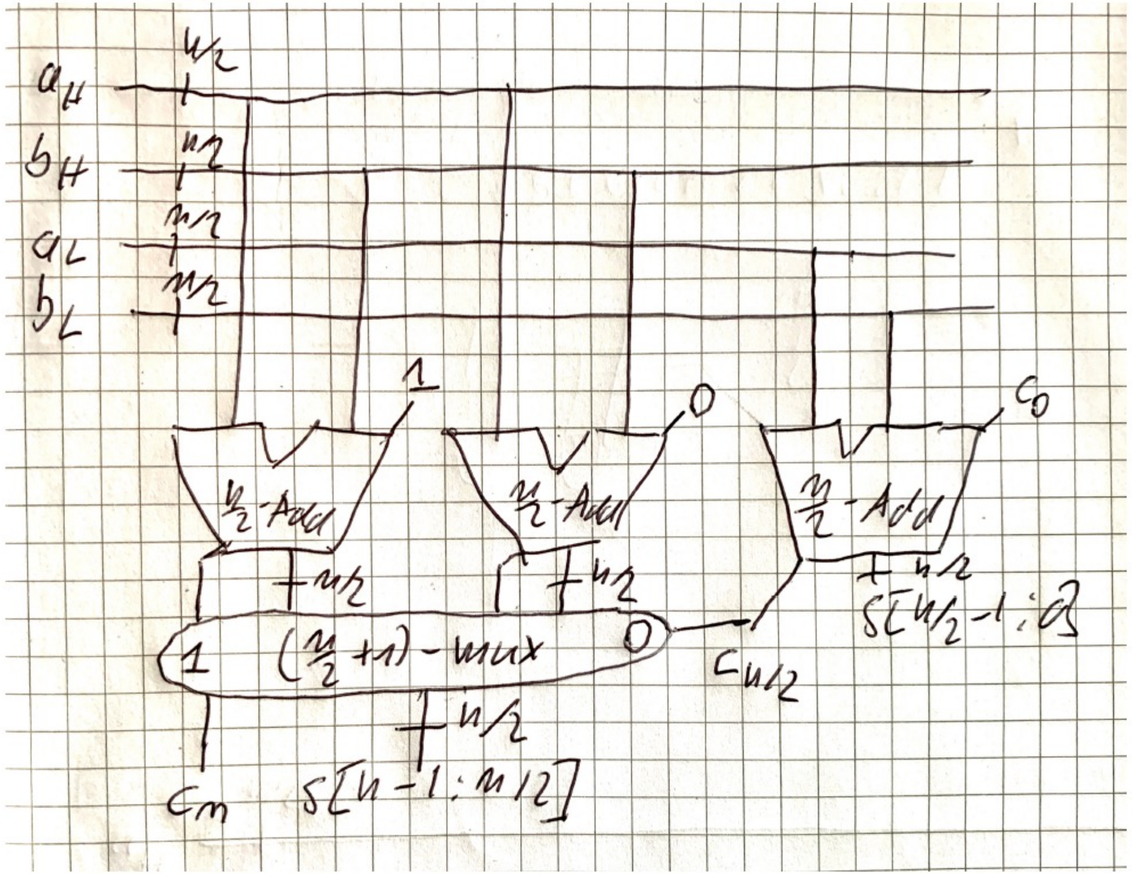
analyzing effort:
read off difference equation from construction

analyzing effort:
read off difference equation from construction

cost

$$c(n) = 3c(n/2) + a \cdot n$$

$$c(1) = c(FA)$$

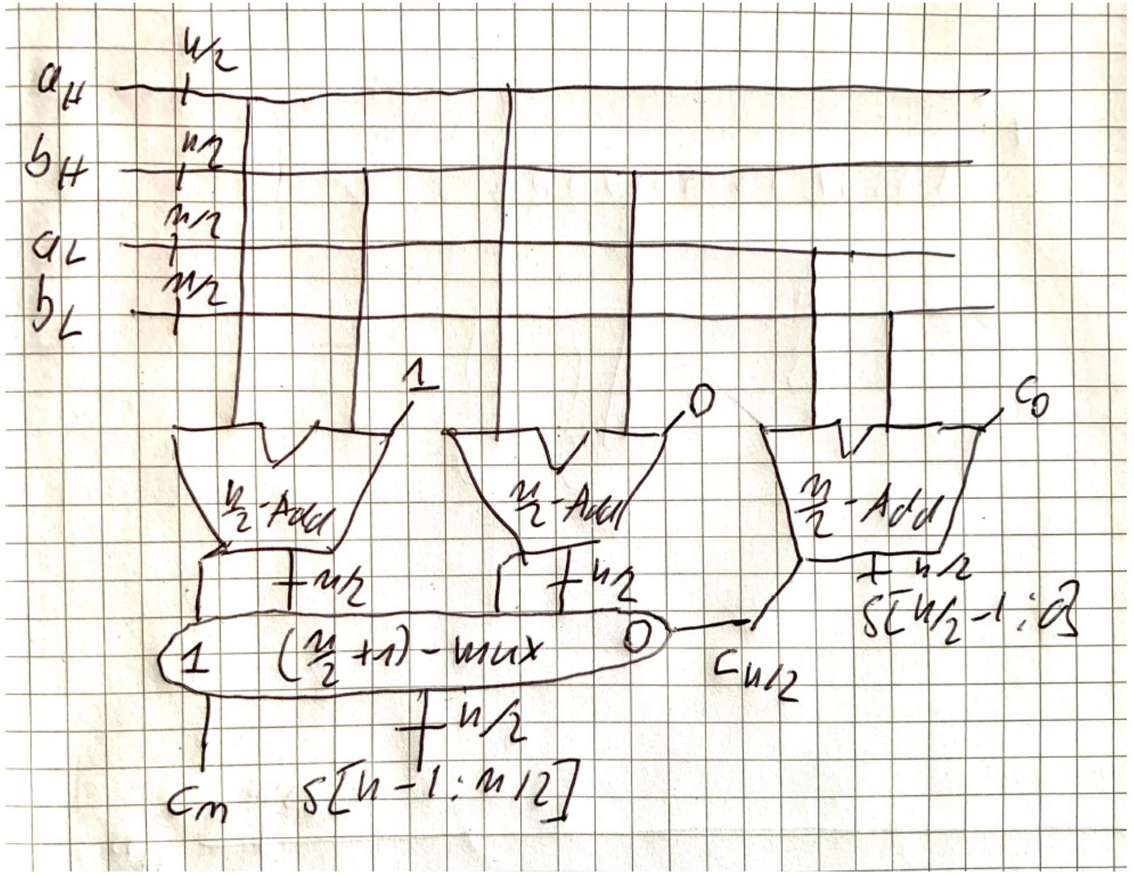


analyzing effort:
read off difference equation from construction

cost

$$c(n) = 3c(n/2) + a \cdot n$$

$$c(1) = c(FA)$$



delay

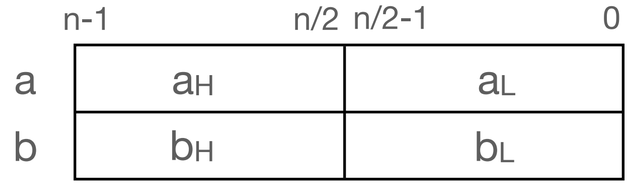
$$d(n) = 3d(n/2) + 3$$

$$d(1) = d(FA)$$

Goal: multiply with asymptotically much less than $O(n^2)$ gates

using subtraction

$$
\begin{aligned}
a_H &= a[n-1:n/2] \\
a_L &= a[n/2-1:0] \\
b_H &= b[n-1:n/2] \\
b_L &= b[n/2-1:0]
\end{aligned}
$$



$$
\begin{aligned}
\langle a\rangle \cdot \langle b\rangle &= (\langle a_H\rangle \cdot 2^{n/2} + \langle a_L\rangle) \cdot (\langle b_H\rangle \cdot 2^{n/2} + \langle b_L\rangle \\
&= A \cdot 2^n + B \cdot 2^{n/2} + C \\
A &= \langle a_H\rangle \cdot \langle b_H\rangle \\
C &= \langle a_L\rangle \cdot \langle b_L\rangle \\
B &= \langle a_H\rangle \cdot \langle b_L\rangle + \langle a_L\rangle \cdot \langle b_H\rangle \\
&= (\langle a_H\rangle + \langle a_L\rangle) \cdot (\langle b_H\rangle + \langle b_L\rangle) - A - C \\
\langle a_H\rangle + \langle a_L\rangle \,, \langle b_H\rangle + \langle b_L\rangle &\in B_{n/2+1}
\end{aligned}
$$

c(n) = cost of n-multiplier constructed here

$$c(1) = 1 \quad , \quad c(n) = 2 \cdot c(n/2) + c(n/2+1) + O(n)$$

adders, subtractors

$$c(n+1) = c(n) + O(n)$$

$d, e \in \mathbb{B}^{n+1}$

(n+1)-multiplier from n-multiplier

$$
\begin{aligned}
\langle d[n:0]\rangle \cdot \langle e[n:0]\rangle &= (d_n \cdot 2^n + \langle d[n-1:0]\rangle) \cdot (e_n \cdot 2^n + \langle e[n-1:0]\rangle) \\
&= d_n \cdot e_n \cdot 2^{2n} + d_n \cdot \langle e[n-1:0]\rangle \cdot 2^n + e_n \cdot \langle d[n-1:0]\rangle \cdot 2^n + \\
&\quad \langle d[n-1:0]\rangle \cdot \langle e[n-1:0]\rangle \\
&= d_n \cdot e_n \cdot 2^{2n} + \langle d_n \wedge e[n-1:0]\rangle \cdot 2^n + \langle e_n \wedge d[n-1:0]\rangle \cdot 2^n + \\
&\quad \langle d[n-1:0]\rangle \cdot \langle e[n-1:0]\rangle
\end{aligned}
$$

$$c(n) = 3 \cdot c(n/2) + r \cdot n$$

difference equations

$$
\begin{aligned}
d(2) &= 1,\\
d(n) &= d(n/2)+2,\\
c(2) &= 1,\\
c(n) &\le c(n/2)+n.
\end{aligned}
$$

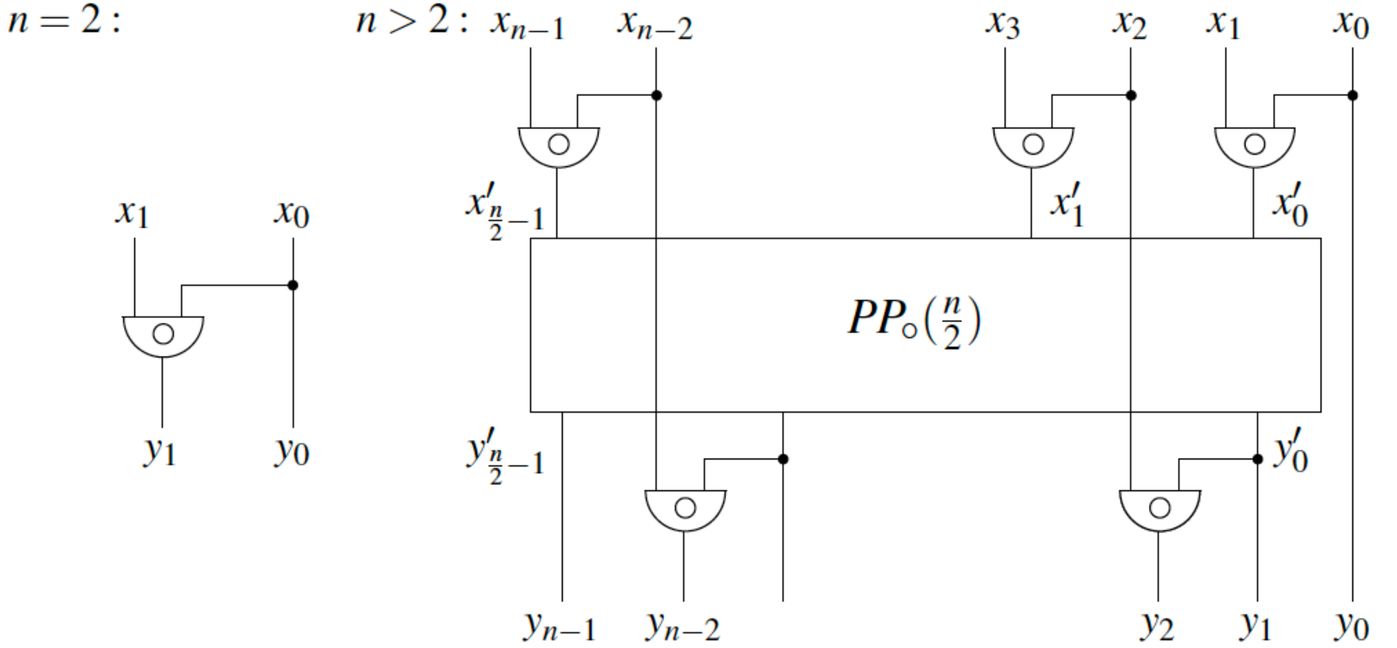


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

solving difference equations

- expand x times… n becomes smaller
- until you can ‚see‘ what happens after x expansions
- determine x such that n= base case
- obtain ‚guess‘
- prove correctnes of guess by induction

difference equations

$$
\begin{aligned}
d(2) &= 1,\\
d(n) &= d(n/2)+2,\\
c(2) &= 1,\\
c(n) &\le c(n/2)+n.
\end{aligned}
$$

expand d

$$d(n) = d(n/2^x)+2\cdot x.$$

stop recursion at

$$
\begin{aligned}
n/2^x &= 2\\
2^x &= n/2\\
x &= \log n - 1.
\end{aligned}
$$

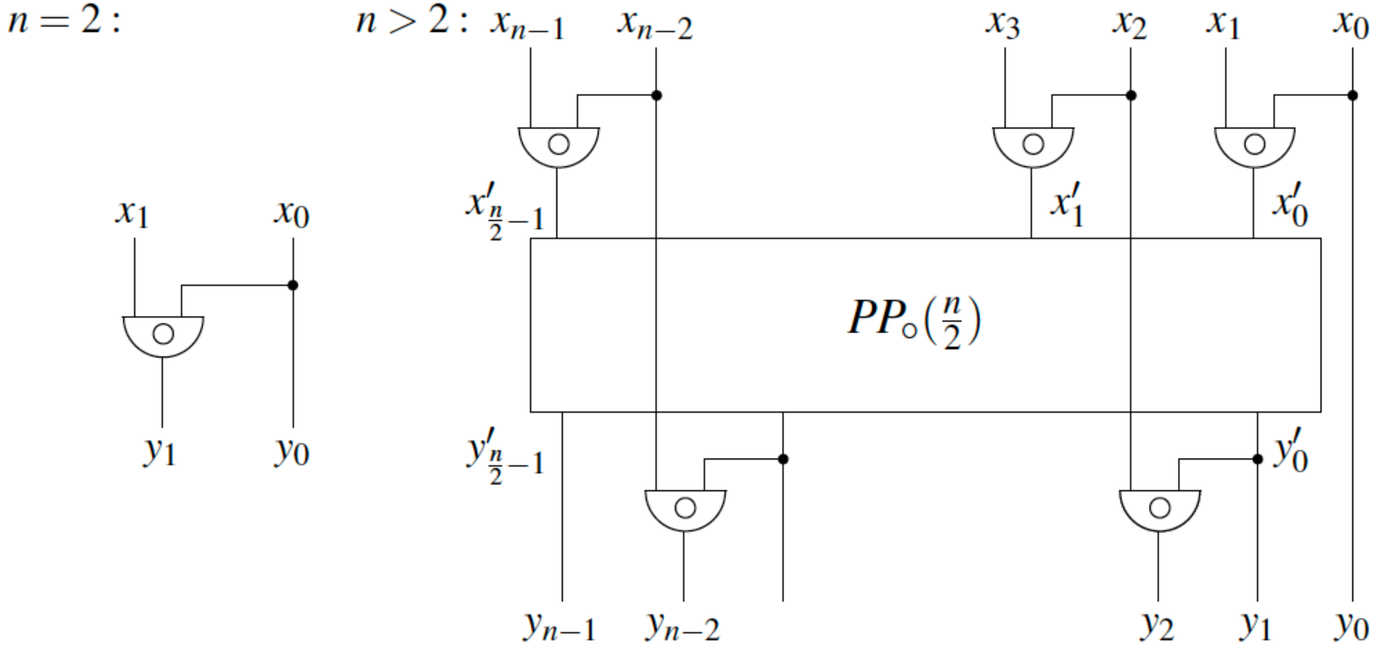


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

difference equations

$$
\begin{aligned}
d(2) &= 1,\\
d(n) &= d(n/2)+2,\\
c(2) &= 1,\\
c(n) &\le c(n/2)+n.
\end{aligned}
$$

expand d

$$d(n) = d(n/2^x)+2\cdot x.$$

stop recursion at

$$
\begin{aligned}
n/2^x &= 2\\
2^x &= n/2\\
x &= \log n - 1.
\end{aligned}
$$

conjecture

$$d(n) = d(2)+2\cdot(\log n - 1) = 2\cdot\log n - 1,$$

easy proof by induction

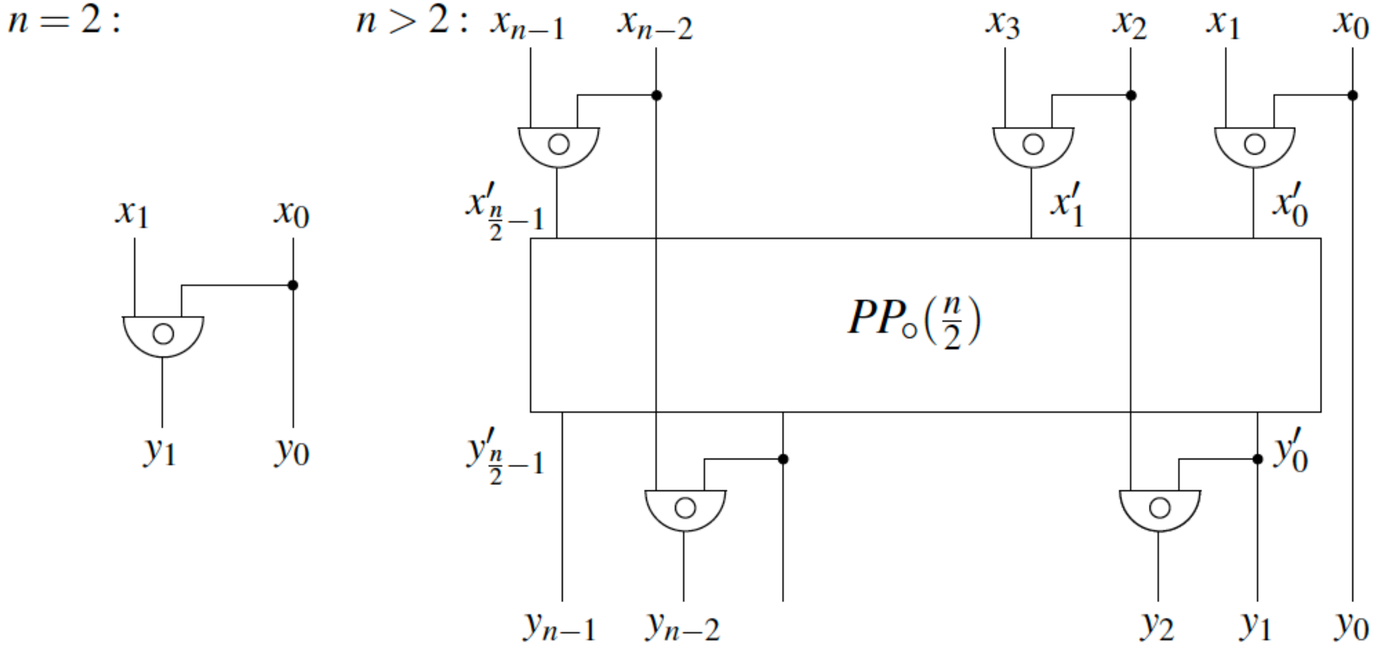


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

difference equations

$$
\begin{aligned}
d(2) &= 1,\\
d(n) &= d(n/2)+2,\\
c(2) &= 1,\\
c(n) &\le c(n/2)+n.
\end{aligned}
$$

expand c

$$
\begin{aligned}
c(n) &\le c(n/2)+n\\
&\le c(n/4)+n+n/2\\
&\le c(n/8)+n+n/2+n/4\\
&\le c(n/x)+n\cdot\sum_{i=0}^{x-1}(1/2)^i\\
&\le c(2)+n\cdot\sum_{i=0}^{\log n-2}(1/2)^i.
\end{aligned}
$$

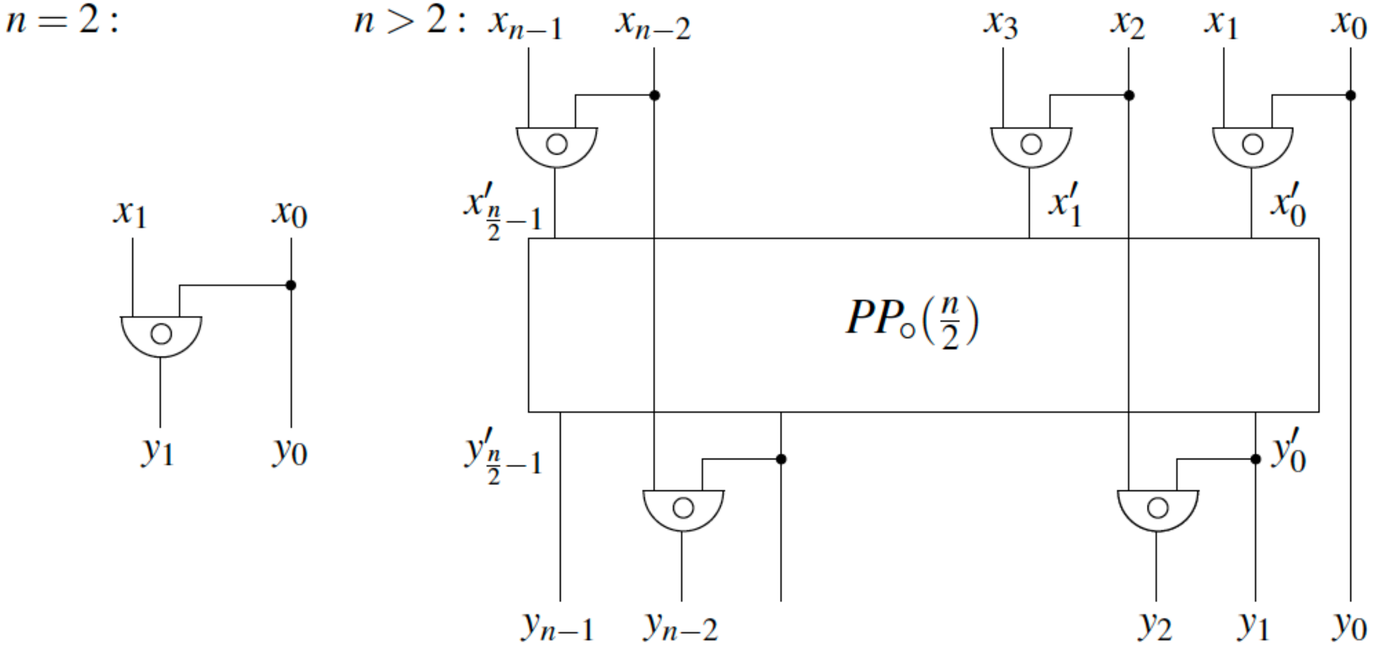


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

difference equations

$$
\begin{aligned}
d(2) &= 1,\\
d(n) &= d(n/2)+2,\\
c(2) &= 1,\\
c(n) &\le c(n/2)+n.
\end{aligned}
$$

expand c

$$
\begin{aligned}
c(n) &\le c(n/2)+n\\
&\le c(n/4)+n+n/2\\
&\le c(n/8)+n+n/2+n/4\\
&\le c(n/x)+n\cdot\sum_{i=0}^{x-1}(1/2)^i\\
&\le c(2)+n\cdot\sum_{i=0}^{\log n-2}(1/2)^i.
\end{aligned}
$$

conjecture

$$\boxed{c(n)\le 2\cdot n,}$$

easy proof by induction

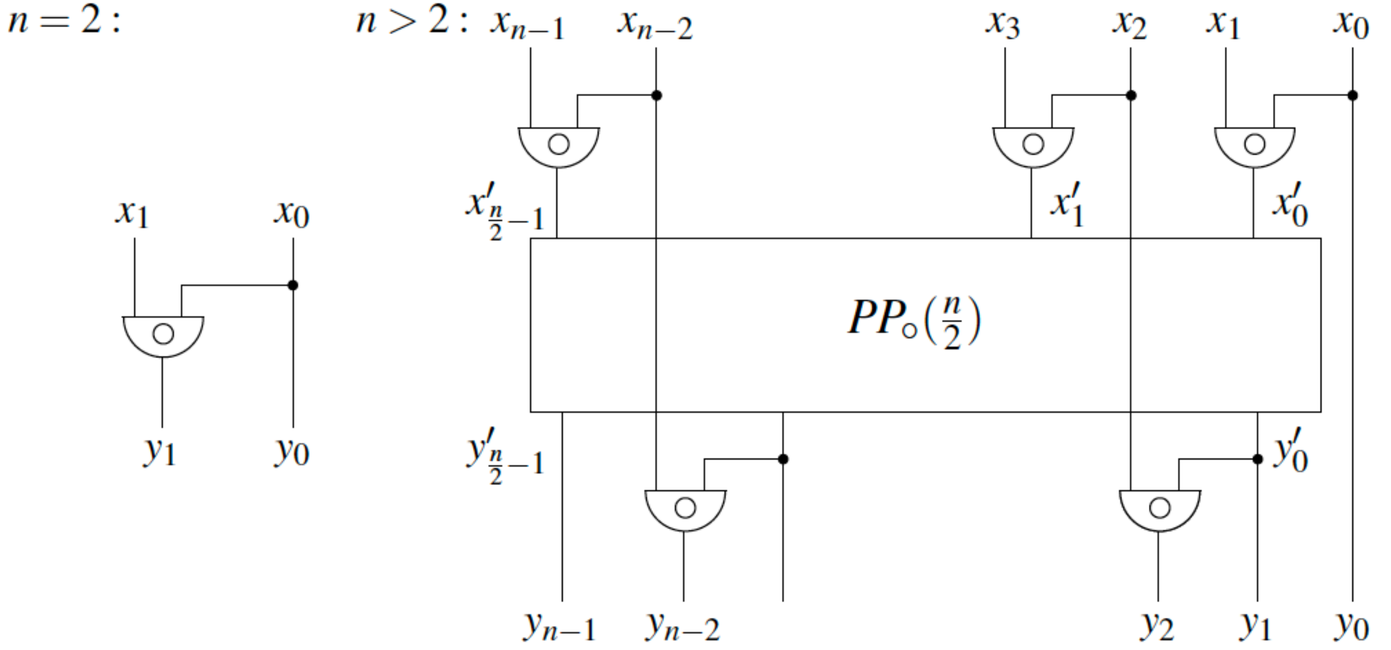
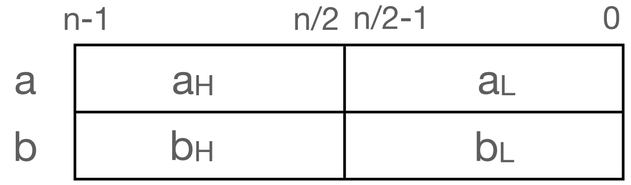


Fig. 35. Recursive construction of an n-bit parallel prefix circuit of the function $\circ$ for an even n

Goal: multiply with asymptotically much less than $O(n^2)$ gates

using subtraction

$$
\begin{aligned}
a_H &= a[n-1:n/2] \\
a_L &= a[n/2-1:0] \\
b_H &= b[n-1:n/2] \\
b_L &= b[n/2-1:0]
\end{aligned}
$$



$$
\begin{aligned}
\langle a\rangle \cdot \langle b\rangle &= (\langle a_H\rangle \cdot 2^{n/2} + \langle a_L\rangle) \cdot (\langle b_H\rangle \cdot 2^{n/2} + \langle b_L\rangle \\
&= A \cdot 2^n + B \cdot 2^{n/2} + C \\
A &= \langle a_H\rangle \cdot \langle b_H\rangle \\
C &= \langle a_L\rangle \cdot \langle b_L\rangle \\
B &= \langle a_H\rangle \cdot \langle b_L\rangle + \langle a_L\rangle \cdot \langle b_H\rangle \\
&= (\langle a_H\rangle + \langle a_L\rangle) \cdot (\langle b_H\rangle + \langle b_L\rangle) - A - C \\
\langle a_H\rangle + \langle a_L\rangle \,, \langle b_H\rangle + \langle b_L\rangle &\in B_{n/2+1}
\end{aligned}
$$

c(n) = cost of n-multiplier constructed here

$$c(1) = 1 \quad , \quad c(n) = 2 \cdot c(n/2) + c(n/2+1) + O(n)$$

adders, subtractors

$$c(n+1) = c(n) + O(n)$$

$d, e \in \mathbb{B}^{n+1}$

(n+1)-multiplier from n-multiplier

$$
\begin{aligned}
\langle d[n:0]\rangle \cdot \langle e[n:0]\rangle &= (d_n \cdot 2^n + \langle d[n-1:0]\rangle) \cdot (e_n \cdot 2^n + \langle e[n-1:0]\rangle) \\
&= d_n \cdot e_n \cdot 2^{2n} + d_n \cdot \langle e[n-1:0]\rangle \cdot 2^n + e_n \cdot \langle d[n-1:0]\rangle \cdot 2^n + \\
&\quad \langle d[n-1:0]\rangle \cdot \langle e[n-1:0]\rangle \\
&= d_n \cdot e_n \cdot 2^{2n} + \langle d_n \wedge e[n-1:0]\rangle \cdot 2^n + \langle e_n \wedge d[n-1:0]\rangle \cdot 2^n + \\
&\quad \langle d[n-1:0]\rangle \cdot \langle e[n-1:0]\rangle
\end{aligned}
$$

$$c(n) = 3 \cdot c(n/2) + r \cdot n$$

solving the difference equation

$$c(1) = 1 \quad , \quad c(n) = 3 \cdot c(n/2) + r \cdot n$$

solving the difference equation

$$c(1) = 1 \quad , \quad c(n) = 3 \cdot c(n/2) + r \cdot n$$

expand

$$\begin{aligned}
c(n) &= 3 \cdot c(n/2) + r \cdot n \\
&= 3 \cdot (3 \cdot c(n/4) + r \cdot n/2) + r \cdot n \\
&= 3^2 \cdot c(n/2^2) + r \cdot n \cdot (1 + n/2) \\
&= 3^2 \cdot (3 \cdot c(n/2^3) + r \cdot n/2^2) + r \cdot n \cdot (1 + 3/2) \\
&= 3^3 \cdot c(n/2^3) + r \cdot n \cdot (1 + 3/2 + (3/2)^2)
\end{aligned}$$

solving the difference equation

$$c(1) = 1 \quad , \quad c(n) = 3 \cdot c(n/2) + r \cdot n$$

expand

$$\begin{aligned} c(n) &= 3 \cdot c(n/2) + r \cdot n \\ &= 3 \cdot (3 \cdot c(n/4) + r \cdot n/2) + r \cdot n \\ &= 3^2 \cdot c(n/2^2) + r \cdot n \cdot (1 + n/2) \\ &= 3^2 \cdot (3 \cdot c(n/2^3) + r \cdot n/2^2) + r \cdot n \cdot (1 + 3/2) \\ &= 3^3 \cdot c(n/2^3) + r \cdot n \cdot (1 + 3/2 + (3/2)^2) \end{aligned}$$

3

guess and prove by induction

$$c(n) = 3^x \cdot c(n/2^x) + r \cdot n \cdot \sum_{i=0}^{x-1} (3/2)^i$$

solving the difference equation

$$c(1) = 1 \quad , \quad c(n) = 3 \cdot c(n/2) + r \cdot n$$

expand

$$\begin{aligned} c(n) &= 3 \cdot c(n/2) + r \cdot n \\ &= 3 \cdot (3 \cdot c(n/4) + r \cdot n/2) + r \cdot n \\ &= 3^2 \cdot c(n/2^2) + r \cdot n \cdot (1 + n/2) \\ &= 3^2 \cdot (3 \cdot c(n/2^3) + r \cdot n/2^2) + r \cdot n \cdot (1 + 3/2) \\ &= 3^3 \cdot c(n/2^3) + r \cdot n \cdot (1 + 3/2 + (3/2)^2) \end{aligned}$$

3

guess and prove by induction

$$c(n) = 3^x \cdot c(n/2^x) + r \cdot n \cdot \sum_{i=0}^{x-1} (3/2)^i$$

$x = \log(n)$:

$$\begin{aligned} c(n) &= 3^{\log(n)} \cdot c(n/n) + r \cdot n \cdot \frac{(3/2)^{\log(n)} - 1}{3/2 - 1} \\ &\leq 3^{\log(n)} \cdot c(1) + 2 \cdot r \cdot n \cdot \frac{3^{\log(n)}}{2^{\log(n)}} \\ &= 3^{\log n} \cdot (1 + 2 \cdot r) \end{aligned}$$

solving the difference equation

$$c(1) = 1 \quad , \quad c(n) = 3 \cdot c(n/2) + r \cdot n$$

expand

$$\begin{aligned} c(n) &= 3 \cdot c(n/2) + r \cdot n \\ &= 3 \cdot (3 \cdot c(n/4) + r \cdot n/2) + r \cdot n \\ &= 3^2 \cdot c(n/2^2) + r \cdot n \cdot (1 + 3/2) \\ &= 3^2 \cdot (3 \cdot c(n/2^3) + r \cdot n/2^2) + r \cdot n \cdot (1 + 3/2) \\ &= 3^3 \cdot c(n/2^3) + r \cdot n \cdot (1 + 3/2 + (3/2)^2) \end{aligned}$$

guess and prove by induction

$$c(n) = 3^x \cdot c(n/2^x) + r \cdot n \cdot \sum_{i=0}^{x-1} (3/2)^i$$

$x = \log(n)$:

$$\begin{aligned} c(n) &= 3^{\log(n)} \cdot c(n/n) + r \cdot n \cdot \frac{(3/2)^{\log(n)} - 1}{3/2 - 1} \\ &\leq 3^{\log(n)} \cdot c(1) + 2 \cdot r \cdot n \cdot \frac{3^{\log(n)}}{2^{\log(n)}} \\ &= 3^{\log n} \cdot (1 + 2 \cdot r) \end{aligned}$$

$$c(n) = O(n^{log(3)})$$

master theorem

- prefabricated solutions to a large family of difference equations
- several versions in literature

master theorem

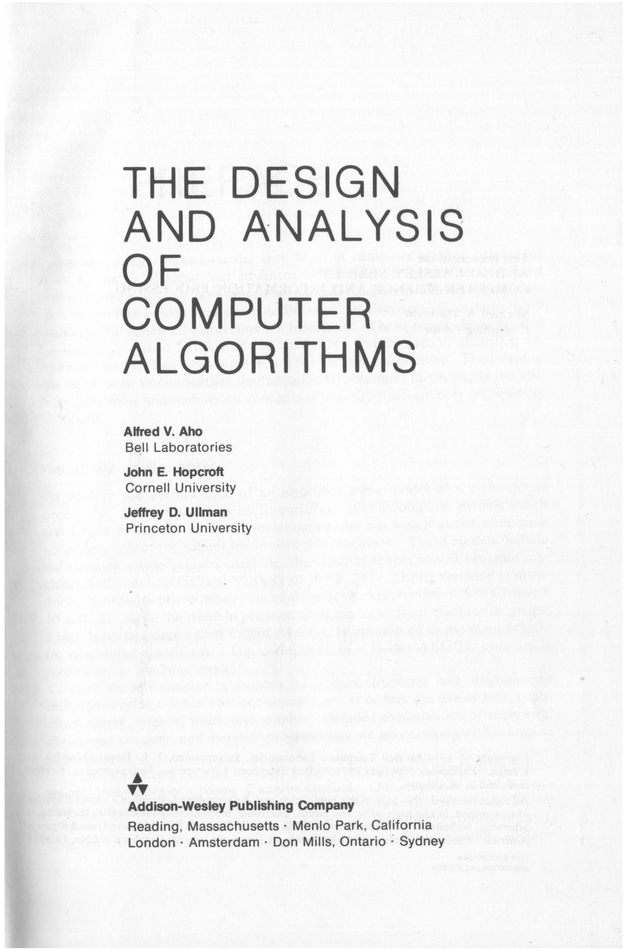
- prefabricated solutions to a large family of difference equations
- several versions in literature
- here: version of THE classical textbook of EA, 1974

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for} \quad n = 1, \\ aT(n/c) + bn, & \text{for} \quad n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if} \quad a < c, \\ O(n \log n), & \text{if} \quad a = c, \\ O(n^{\log_c a}), & \text{if} \quad a > c. \end{cases}$$



proof of master theorem

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for } n = 1, \\ aT(n/c) + bn, & \text{for } n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c. \end{cases}$$

$$\begin{aligned} T(n) &= a \cdot T(n/c) + b \cdot n \\ &= a \cdot (a \cdot T(n/c^2) + b \cdot n/c) + b \cdot n \\ &= a^2 \cdot T(n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^2 \cdot (a \cdot T(n/c^3) + b \cdot n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^3 \cdot T(n/c^3) + b \cdot n \cdot (1 + a/c + (a/c)^2) \\ T(n) &= a^x \cdot T(n/c^x) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

proof of master theorem

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for } n = 1, \\ aT(n/c) + bn, & \text{for } n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c. \end{cases}$$

$$\begin{aligned} T(n) &= a \cdot T(n/c) + b \cdot n \\ &= a \cdot (a \cdot T(n/c^2) + b \cdot n/c) + b \cdot n \\ &= a^2 \cdot T(n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^2 \cdot (a \cdot c(n/c^3) + b \cdot n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^3 \cdot T(n/c^3) + b \cdot n \cdot (1 + a/c + (a/c)^2) \\ T(n) &= a^x \cdot T(n/c^x) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

$x = \log_c n$:

$$\begin{aligned} T(n) &= a^{\log_c(n)} \cdot T(n/n) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= (c^{\log_c a})^{\log_c n} \cdot b + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= b \cdot n^{\log_c a} + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

proof of master theorem

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for } n = 1, \\ aT(n/c) + bn, & \text{for } n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c. \end{cases}$$

$$\begin{aligned} T(n) &= a \cdot T(n/c) + b \cdot n \\ &= a \cdot (a \cdot T(n/c^2) + b \cdot n/c) + b \cdot n \\ &= a^2 \cdot T(n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^2 \cdot (a \cdot c(n/c^3) + b \cdot n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^3 \cdot T(n/c^3) + b \cdot n \cdot (1 + a/c + (a/c)^2) \end{aligned}$$

$$T(n) = a^x \cdot T(n/c^x) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i$$

$x = \log_c n$:

$$\begin{aligned} T(n) &= a^{\log_c(n)} \cdot T(n/n) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= (c^{\log_c a})^{\log_c n} \cdot b + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= b \cdot n^{\log_c a} + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

- $a < c$

$$\log_c a < 1 \ , \ n^{\log_c(a)} = o(n) \ , \ \sum_{i=0}^{x-1} (a/c)^i = O(1) \ , \ T(n) = O(n)$$

proof of master theorem

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for } n = 1, \\ aT(n/c) + bn, & \text{for } n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c. \end{cases}$$

$x = \log_c n$:

$$\begin{aligned} T(n) &= a^{\log_c(n)} \cdot T(n/n) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= (c^{\log_c a})^{\log_c n} \cdot b + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= b \cdot n^{\log_c a} + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

$$\begin{aligned} T(n) &= a \cdot T(n/c) + b \cdot n \\ &= a \cdot (a \cdot T(n/c^2) + b \cdot n/c) + b \cdot n \\ &= a^2 \cdot T(n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^2 \cdot (a \cdot c(n/c^3) + b \cdot n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^3 \cdot T(n/c^3) + b \cdot n \cdot (1 + a/c + (a/c)^2) \\ T(n) &= a^x \cdot T(n/c^x) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

- $a = c$

$$\log_c a = 1 \ , \ n^{\log_c(a)} = n \ , \ \sum_{i=0}^{x-1} (a/c)^i = O(\log_c(n)) \ , \ T(n) = O(n \cdot \log_c(n))$$

$$\begin{aligned} 2^{\log n} &= n \\ &= c^{\log_c n} \\ &= (2^{\log c})^{\log_c n} \\ &= 2^{(\log c) \cdot \log_c n} \\ \log_c n &= O(\log n) \end{aligned}$$

proof of master theorem

Theorem 2.1. Let a, b, and c be nonnegative constants. The solution to the recurrence

$$T(n) = \begin{cases} b, & \text{for } n = 1, \\ aT(n/c) + bn, & \text{for } n > 1 \end{cases}$$

for n a power of c is

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c. \end{cases}$$

$x = \log_c n$:

$$\begin{aligned} T(n) &= a^{\log_c(n)} \cdot T(n/n) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= (c^{\log_c a})^{\log_c n} \cdot b + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \\ &= b \cdot n^{\log_c a} + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

$$\begin{aligned} T(n) &= a \cdot T(n/c) + b \cdot n \\ &= a \cdot (a \cdot T(n/c^2) + b \cdot n/c) + b \cdot n \\ &= a^2 \cdot T(n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^2 \cdot (a \cdot c(n/c^3) + b \cdot n/c^2) + b \cdot n \cdot (1 + a/c) \\ &= a^3 \cdot T(n/c^3) + b \cdot n \cdot (1 + a/c + (a/c)^2) \\ T(n) &= a^x \cdot T(n/c^x) + b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i \end{aligned}$$

- $a > c$

$$\begin{aligned} b \cdot n \cdot \sum_{i=0}^{x-1} (a/c)^i &= b \cdot n \cdot \frac{(a/c)^{\log_c n} - 1}{(a/c) - 1} \quad \text{(geometric sum)} \\ &= O\left(\frac{n \cdot a^{\log_c n}}{c^{\log_c n}}\right) \\ &= O(a^{\log_c n}) \\ &= O(n^{\log_c a}) \quad \text{(above)} \end{aligned}$$

$a^{\log_c n} = n^{\log_n a \cdot \log_c n} = n^{\log_c a}$