2IL50 Data Structures

2023-24 Q3

Lecture 2: Analysis of Algorithms

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Analysis of algorithms

the formal way ...

Analysis of algorithms

Can we say something about the running time of an algorithm without implementing and testing it?

InsertionSort(A)

initialize: sort $A[1]$
for $j = 2$ **to** $A.\mathrm{length}$
    $\mathrm{key} = A[j]$
    $i = j - 1$
    while $i > 0$ and $A[i] > \mathrm{key}$
        $A[i + 1] = A[i]$
        $i = i - 1$
    $A[i + 1] = \mathrm{key}$

Analysis of algorithms

Analyze the running time as a function of n (# of input elements)

- best case
- average case
- worst case

An algorithm has worst case running time $T(n)$ if for any input of size n the maximal number of elementary operations executed is $T(n)$.

elementary operations

add, subtract, multiply, divide, load, store, copy, conditional and unconditional branch, return …

Linear Search

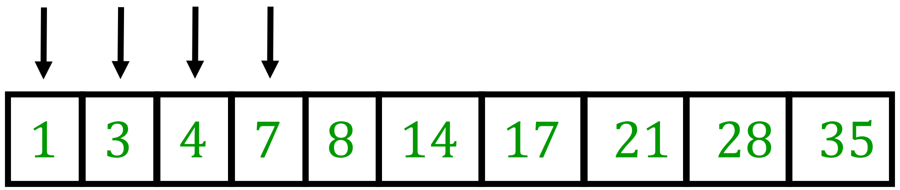
Input: increasing sequence of n numbers $A = \langle a_1, a_2, \dots, a_n \rangle$ and value v

Output: an index i such that $A[i] = v$ or NIL if v not in A

LinearSearch(A, v)

for $i = 1$ **to** n
    if $A[i] == v$
        return i
return NIL

$v = 7$



Running time

- best case: 1
- average case: $n/2$ (if successful)
- worst case: n

Binary Search

Input: increasing sequence of n numbers $A = \langle a_1, a_2, \dots, a_n \rangle$ and value v

Output: an index i such that $A[i] = v$ or NIL if v not in A

BinarySearch(A, v)

$x = 1$

$y = n + 1$

while $x + 1 < y$ and $A[x] \neq v$

  $h = \left\lfloor \frac{x+y}{2} \right\rfloor$

  if $A[h] \leq v$: $x = h$ **else** $y = h$

if $A[x] == v$: **return** x **else return** NIL

$v = 7$

1	3	4	7	8	14	17	21	28	35

Running time

- best case: 1
- average case: $\log n$
- worst case: $\log n$

Analysis of algorithms: example

		$n = 10$	$n = 100$	$n = 1000$
InsertionSort:	$15n^2 + 7n - 2$	1568	150698	1.5×10^7
MergeSort:	$300\, n \log n + 50n$	10466	204316	3.0×10^6

InsertionSort
$6 \times$ faster

InsertionSort
$1.35 \times$ faster

MergeSort
$5 \times$ faster

The rate of growth of the running time as a function of the input is essential!

$n = 1{,}000{,}000$ InsertionSort 1.5×10^{13}

MergeSort 6×10^9 $2500 \times$ faster !

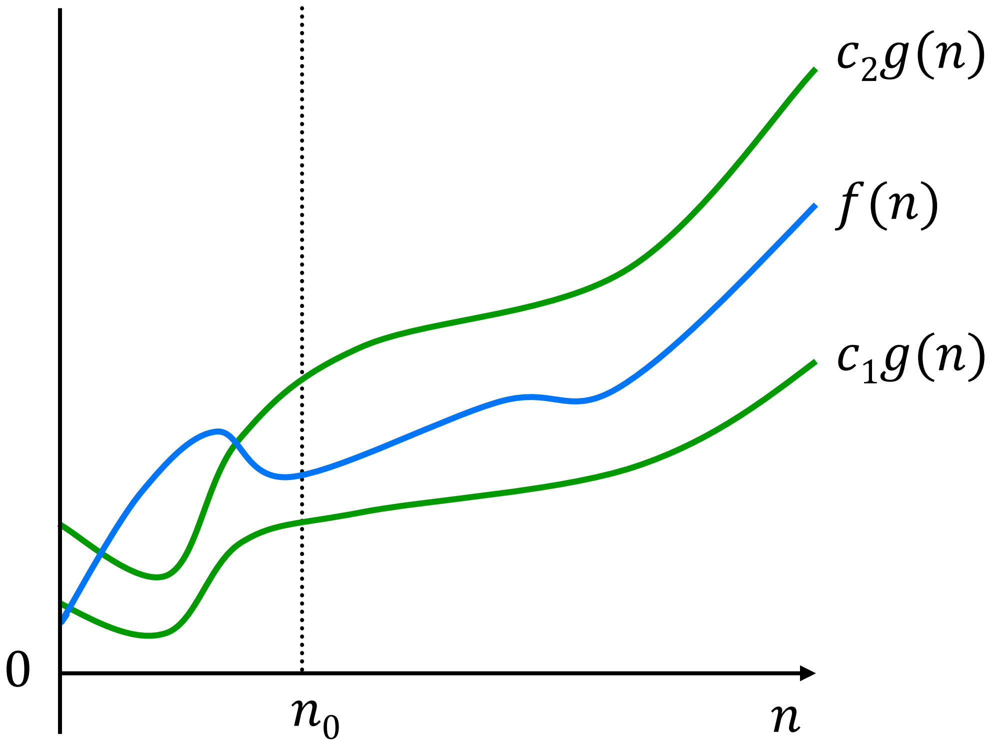
Θ-notation

Let $g(n): N \to N$ be a function. Then we have

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0
such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

"$\Theta(g(n))$ is the set of functions that grow as fast as $g(n)$"

Notation: $f(n) = \Theta(g(n))$



Θ-notation

Let $g(n): N \to N$ be a function. Then we have

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

Claim: $19n^3 + 17n^2 - 3n = \Theta(n^3)$

Proof: Choose $c_1 = 19$, $c_2 = 36$ and $n_0 = 1$.
Then we have for all $n \geq n_0$:

$$
\begin{aligned}
c_1 n^3 &= 19n^3 && \text{(trivial)} \\
&\leq 19n^3 + 17n^2 - 3n && (\text{since } 17n^2 > 3n \text{ for } n \geq 1) \\
&\leq 19n^3 + 17n^3 && (\text{since } 17n^2 \leq 17n^3 \text{ for } n \geq 1) \\
&= c_2 n^3 && \blacksquare
\end{aligned}
$$

Θ-notation

Let $g(n): N \to N$ be a function. Then we have

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0
such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$

Claim: $19n^3 + 17n^2 - 3n \ne \Theta(n^2)$

Proof: Assume that there are positive constants c_1, c_2, and n_0 such that for all $n \ge n_0$

$c_1 n^2 \le 19n^3 + 17n^2 - 3n \le c_2 n^2$

Since $19n^3 + 17n^2 - 3n \le c_2 n^2$

implies $19n^3 \le c_2 n^2 + 3n - 17n^2 \le c_2 n^2 \quad (3n - 17n^2 \le 0)$

we would have for all $n \ge n_0$

$19n \le c_2$.

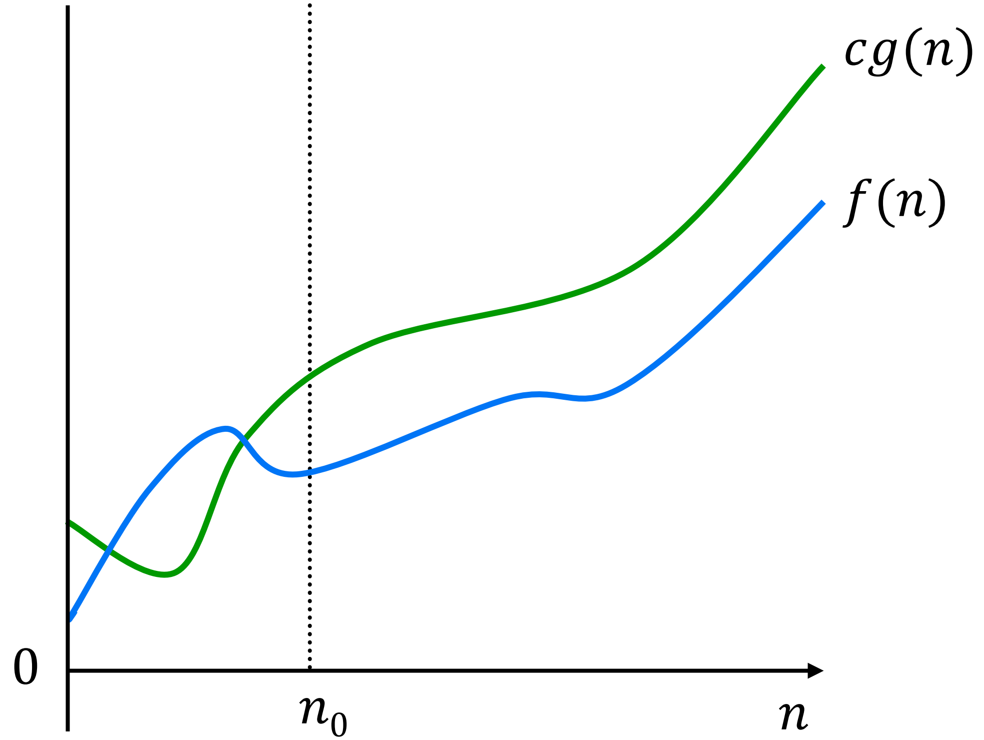
O-notation

Let $g(n): N \to N$ be a function. Then we have

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0
such that $f(n) \leq cg(n)$ for all $n \geq n_0\}$

"$O(g(n))$ is the set of functions that grow at most as fast as $g(n)$"

Notation: $f(n) = O(g(n))$



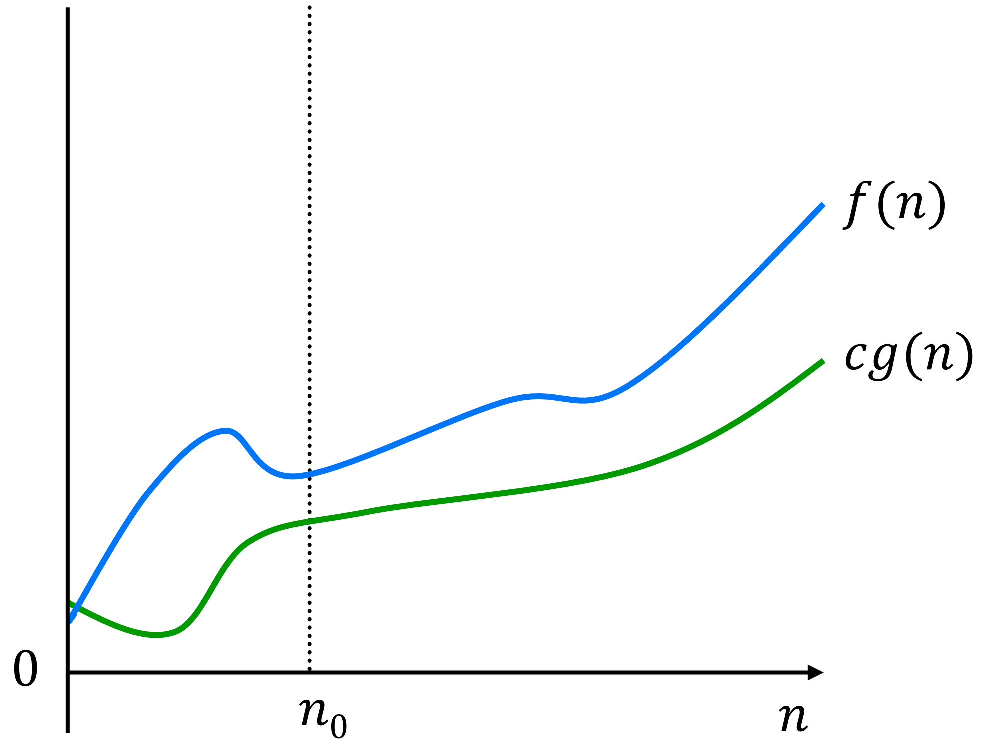
Ω-notation

Let $g(n): N \to N$ be a function. Then we have

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0
such that $cg(n) \le f(n)$ for all $n \ge n_0\}$

"$\Omega(g(n))$ is the set of functions that grow at least as fast as $g(n)$"

Notation: $f(n) = \Omega(g(n))$



Asymptotic notation

$\Theta(\ldots)$ is an asymptotically tight bound — "asymptotically equal"

$O(\ldots)$ is an asymptotic upper bound — "asymptotically smaller or equal"

$\Omega(\ldots)$ is an asymptotic lower bound — "asymptotically greater or equal"

other asymptotic notation

$o(\ldots) \rightarrow$ "grows strictly slower than"
$\omega(\ldots) \rightarrow$ "grows strictly faster than"

More notation …

$f(n) = n^3 + \Theta(n^2)$	means	there is a function $g(n)$ such that $f(n) = n^3 + g(n)$ and $g(n) = \Theta(n^2)$
$f(n) = \sum_{i=1}^{n} O(i)$	means	there is one function $g(i)$ such that $f(n) = \sum_{i=1}^{n} g(i)$ and $g(i) = O(i)$
$O(1)$ or $\Theta(1)$	means	a constant
$2n^2 + O(n) = \Theta(n^2)$	means	for each function $g(n)$ with $g(n) = O(n)$ we have $2n^2 + g(n) = \Theta(n^2)$

Quiz

1. $O(1) + O(1) = O(1)$ true
2. $O(1) + \cdots + O(1) = O(1)$ false
3. $\sum_{i=1}^{n} O(i) = O(\sum_{i=1}^{n} i)$ true
4. $O(n^2) \subseteq O(n^3)$ true
5. $O(n^3) \subseteq O(n^2)$ false
6. $\Theta(n^2) \subseteq O(n^3)$ true
7. An algorithm with worst case running time $O(n \log n)$ is always slower than an algorithm with worst case running time $O(n)$ if n is sufficiently large. false

Quiz

8. $n \log^2 n = \Theta(n \log n)$ false
9. $n \log^2 n = \Omega(n \log n)$ true
10. $n \log^2 n = O(n^{4/3})$ true
11. $O(2^n) \subseteq O(3^n)$ true
12. $O(2^n) \subseteq \Theta(3^n)$ false

Analysis of algorithms

Analysis of InsertionSort

```
InsertionSort(A)
initialize: sort A[1]
for j = 2 to A.length
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Get as tight a bound as possible on the worst case running time.

- ➡ lower and upper bound for worst case running time

Upper bound: Analyze worst case number of elementary operations

Lower bound: Give “bad” input example

Analysis of InsertionSort

InsertionSort(A)

initialize: sort $A[1]$ $\quad O(1)$
for $j = 2$ **to** $A.\text{length}$
  $\text{key} = A[j]$
  $i = j - 1$
  } $O(1)$
  while $i > 0$ and $A[i] > \text{key}$
    $A[i+1] = A[i]$
    $i = i - 1$
  } worst case: $(j-1) \cdot O(1)$
  $A[i+1] = \text{key}$ $\quad O(1)$

The worst case running time of InsertionSort is $\Theta(n^2)$.

Upper bound: Let $T(n)$ be the worst case running time of InsertionSort on an array of length n. We have

$$T(n) = O(1) + \sum_{j=2}^{n} \{O(1) + (j-1) \cdot O(1) + O(1)\} = \sum_{j=2}^{n} O(j) = O(n^2)$$

Lower bound: Array sorted in decreasing order ➡ $\Omega(n^2)$

Analysis of MergeSort

MergeSort(A)

// divide-and-conquer algorithm that sorts array $A[1{:}n]$

if $A.\text{length} == 1$ — $O(1)$

  skip

else

  $n = A.\text{length};\ n_1 = \left\lfloor \frac{n}{2} \right\rfloor;\ n_2 = \left\lceil \frac{n}{2} \right\rceil$ — $O(1)$

  copy $A[1{:}n_1]$ to auxiliary array $A_1[1{:}n_1]$ — $O(n)$

  copy $A[n_1 + 1{:}n]$ to auxiliary array $A_2[1{:}n_2]$ — $O(n)$

  MergeSort(A_1); MergeSort(A_2) — ??

  Merge(A, A_1, A_2) — $O(n)$

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right)$$

MergeSort is a recursive algorithm

➡ running time analysis leads to recursion

Analysis of MergeSort

Let $T(n)$ be the worst case running time of MergeSort on an array of length n.

We have

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

frequently omitted since it (nearly) always holds

often written as $2T(n/2)$

Solving recurrences

Solving recurrences

Easiest: Master theorem
caveat: not always applicable

Alternatively: Guess the solution and use the substitution method to prove that your guess is correct.

How to guess:

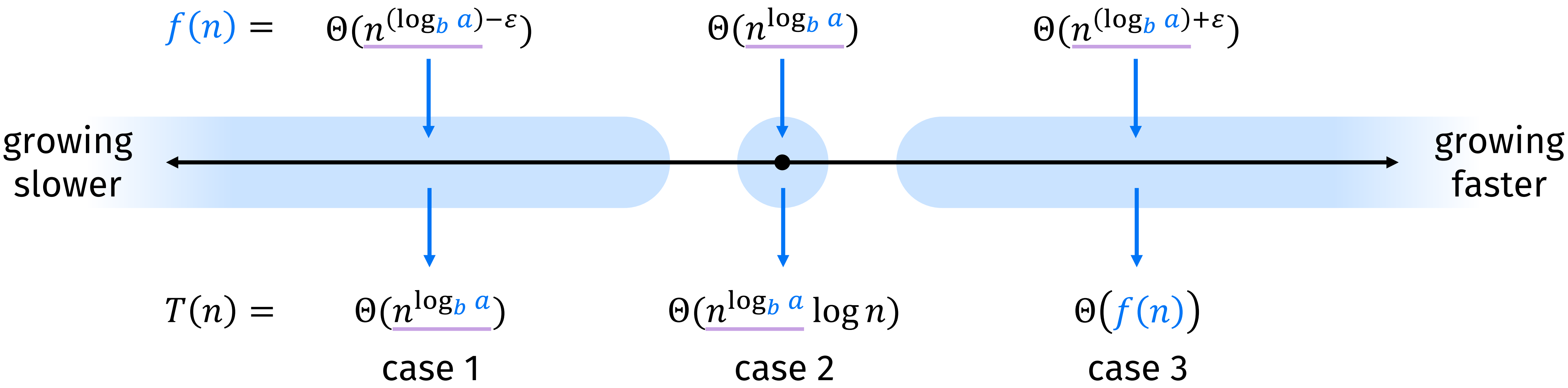
1. expand the recursion
2. draw a recursion tree

The master theorem

Let a and b be constants, let $f(n)$ be a function,
and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

Watershed function: $n^{\log_b a}$



The master theorem

Let a and b be constants, let $f(n)$ be a function,
and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) \;=\; aT\boxed{(n/b)} + f(n)$$ can be rounded up or down

Watershed function: $n^{\log_b a}$

Then we have:

1. If $f(n) = O(n^{(\log_b a)-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a}\boxed{\log^k n})$, for some constant $k \geq 0$, then $T(n) = \Theta(n^{\log_b a}\log^{\boxed{k+1}} n)$.
   allows for extra log factors
3. If $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ for some constant $\varepsilon > 0$,
   and if $af(n/b) \;\leq\; cf(n)$ for some constant $c < 1$ and all sufficiently large n,
   then $T(n) = \Theta(f(n))$.

The master theorem: Example

$T(n) = 4T(n/2) + n^3$

Master theorem with $a = 4$, $b = 2$, and $f(n) = n^3$

$\log_b a = \log_2 4 = 2$ ➡ watershed function = n^2

➡ $n^3 = f(n) = \Omega(n^{2+\varepsilon})$ with, for example, $\varepsilon = 1$

Case 3 of the master theorem gives $T(n) = \Theta(n^3)$, if the regularity condition holds.

choose $c = \frac{1}{2}$ and $n_0 = 1$

➡ $af(n/b) = 4(n/2)^3 = n^3/2 \le cf(n)$ for $n \ge n_0$

➡ $T(n) = \Theta(n^3)$

The substitution method

The Master theorem does not always apply

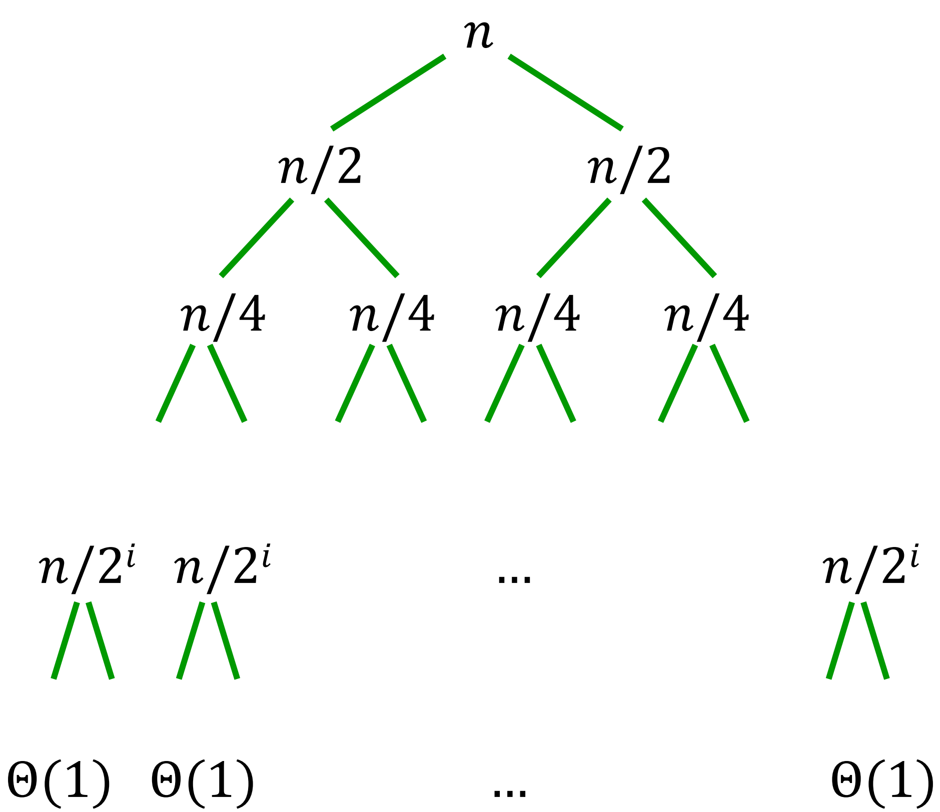
In those cases, use the substitution method:

1. Guess the form of the solution.
2. Use induction to find the constants and show that the solution works

Use expansion or a recursion tree to guess a good solution.

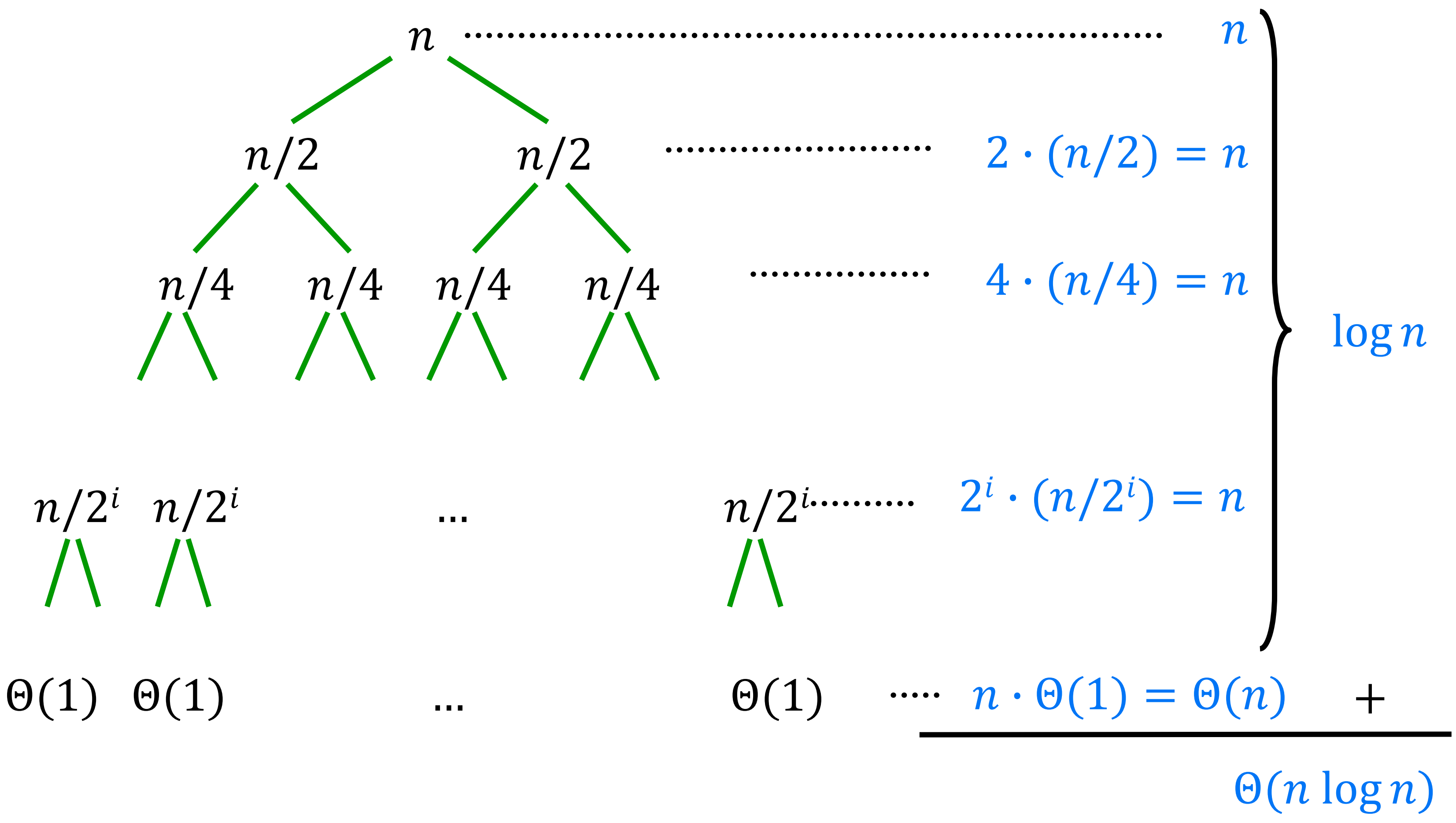
Recursion trees

$T(n) = 2T(n/2) + n$



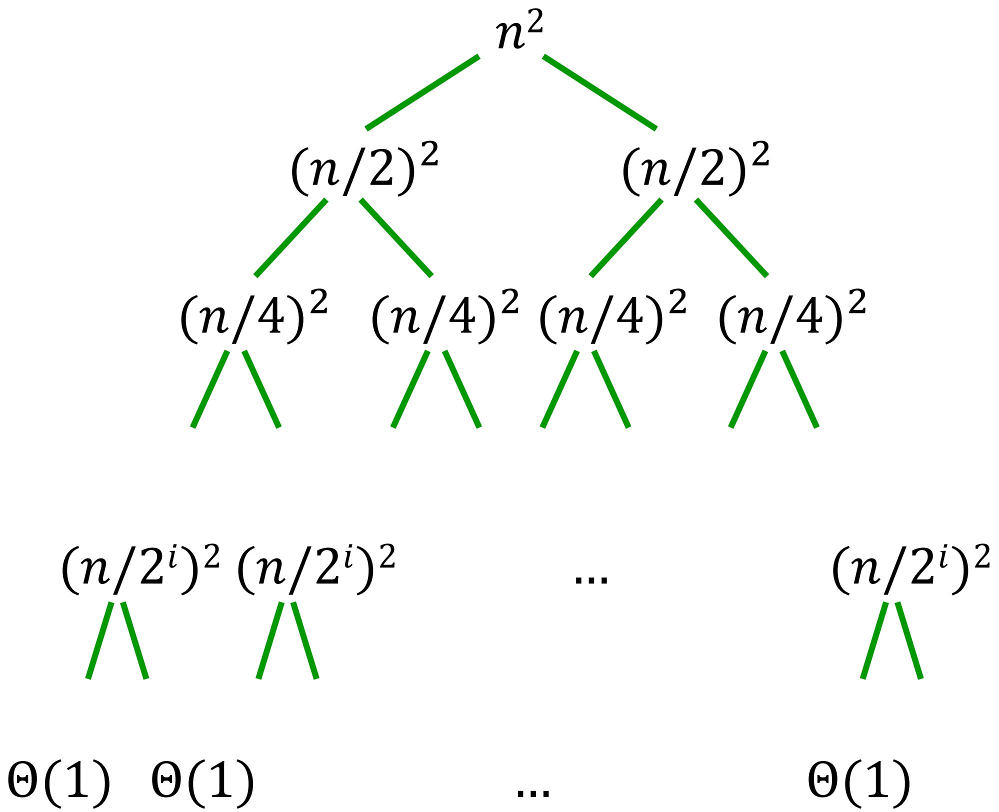
Recursion trees

$T(n) = 2T(n/2) + n$



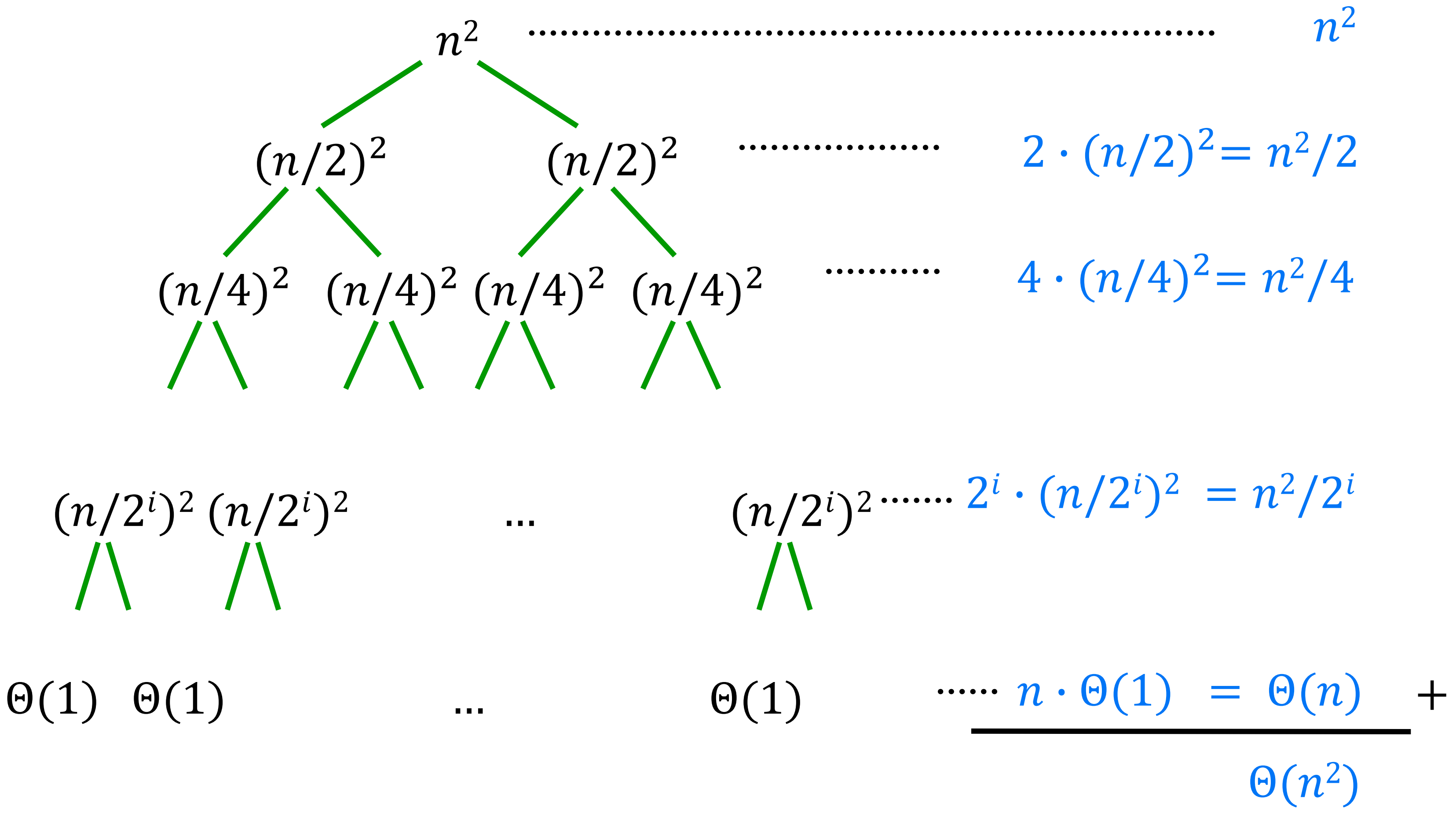
Recursion trees

$T(n) = 2T(n/2) + n^2$



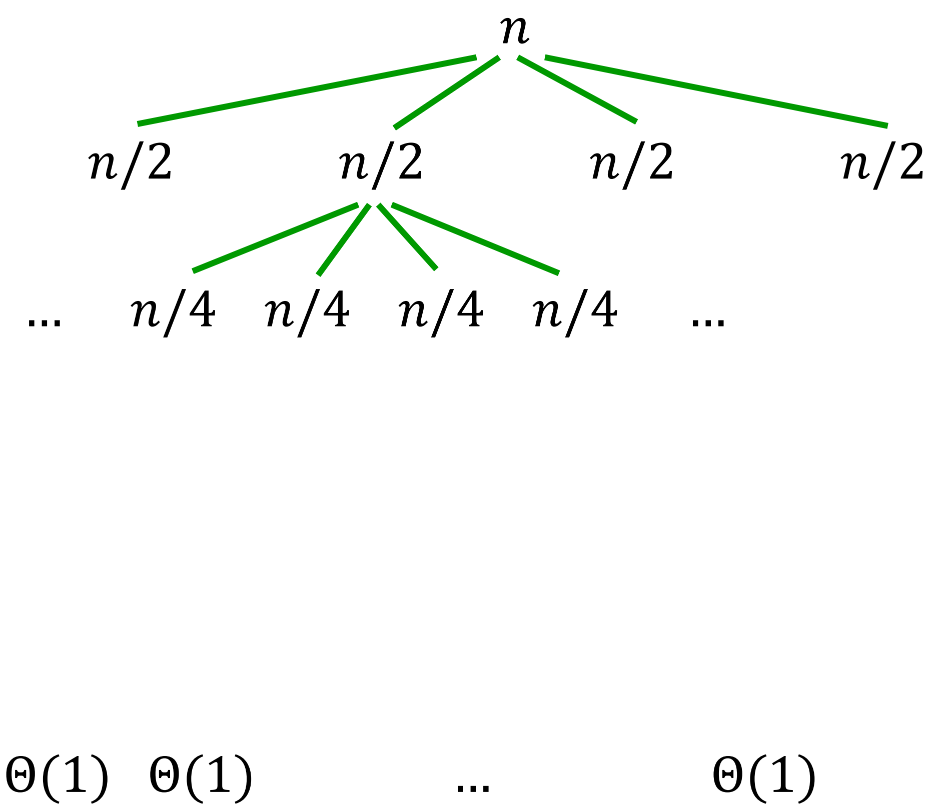
Recursion trees

$T(n) = 2T(n/2) + n^2$



Recursion trees

$T(n) = 4T(n/2) + n$



Recursion trees

$T(n) = 4T(n/2) + n$

n .... n

$n/2 \quad n/2 \quad n/2 \quad n/2$ .... $4 \cdot (n/2) = 2n$

... $n/4 \quad n/4 \quad n/4 \quad n/4$ ... . $16 \cdot (n/4) = 4n$

$\Theta(1) \quad \Theta(1) \quad \ldots \quad \Theta(1)$ .... $n^2 \cdot \Theta(1) = \Theta(n^2)$ +

$\Theta(n^2)$

The substitution method

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & \text{if } n > 1 \end{cases}$$

Claim: $T(n) = O(n \log n)$

Proof: by induction on n

to show: there are constants c and n_0 such that
$T(n) \leq c\, n \log n$ for all $n \geq n_0$

$n = 1$ ➡ $T(1) = 2 \leq c\, 1 \log 1$ ➡ $n_0 = 2$

$n = n_0 = 2$ is a base case

Need more base cases? $\lfloor 3/2 \rfloor = 1, \lfloor 4/2 \rfloor = 2$ ➡ 3 must also be base case

Base cases:

$n = 2$: $T(2) = 2T(1) + 2 = 2 \cdot 2 + 2 = 6 = c\, 2 \log 2$ for $c = 3$

$n = 3$: $T(3) = 2T(1) + 3 = 2 \cdot 2 + 3 = 7 \leq c\, 3 \log 3$

The substitution method

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & \text{if } n > 1 \end{cases}$$

Claim: $T(n) = O(n \log n)$

Proof: by induction on n

to show: there are constants c and n_0 such that
$T(n) \leq c\, n \log n$ for all $n \geq n_0$

choose $c = 3$ and $n_0 = 2$

Inductive step: $n > 3$

$$\begin{aligned} T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq 2\, c\, n/2 \log n/2 + n && \text{(induction hypothesis)} \\ &\leq c\, n\, ((\log n) - 1) + n \\ &\leq c\, n \log n \end{aligned}$$

■

The substitution method

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & \text{if } n > 1 \end{cases}$$

Claim: $T(n) = O(n)$

Proof: by induction on n

Base case: $n = n_0$

$$T(2) = 2T(1) + 2 = 2c + 2 = O(2)$$

Inductive step: $n > n_0$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &= 2O(\lfloor n/2 \rfloor) + n \qquad \text{(induction hypothesis)} \\ &= O(n) \end{aligned}$$

■

Never use O, Θ, or Ω in a proof by induction!

Tips

Analysis of recursive algorithms:
find the recursion and solve with master theorem if possible

Analysis of loops: summations

Some standard recurrences and sums:

$$T(n) = 2T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n \log n)$$

$$\sum_{i=1}^{n} i = \frac{1}{2} n(n+1) = \Theta(n^2)$$

$$\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1) = \Theta(n^3)$$