15-150
Fall 2024

Lecture 6

Asymptotic Analysis

Today

- Big-O complexity classes
- Recurrence relations
- Work and Span
- Application: Sorting

program → recurrence → work/span

Asymptotic

- We assume basic ops take ***constant time***
- Want to find running time *f(n)*, for ***large*** *n*
  - an *estimate*, independent of architecture
- Give big-O classification

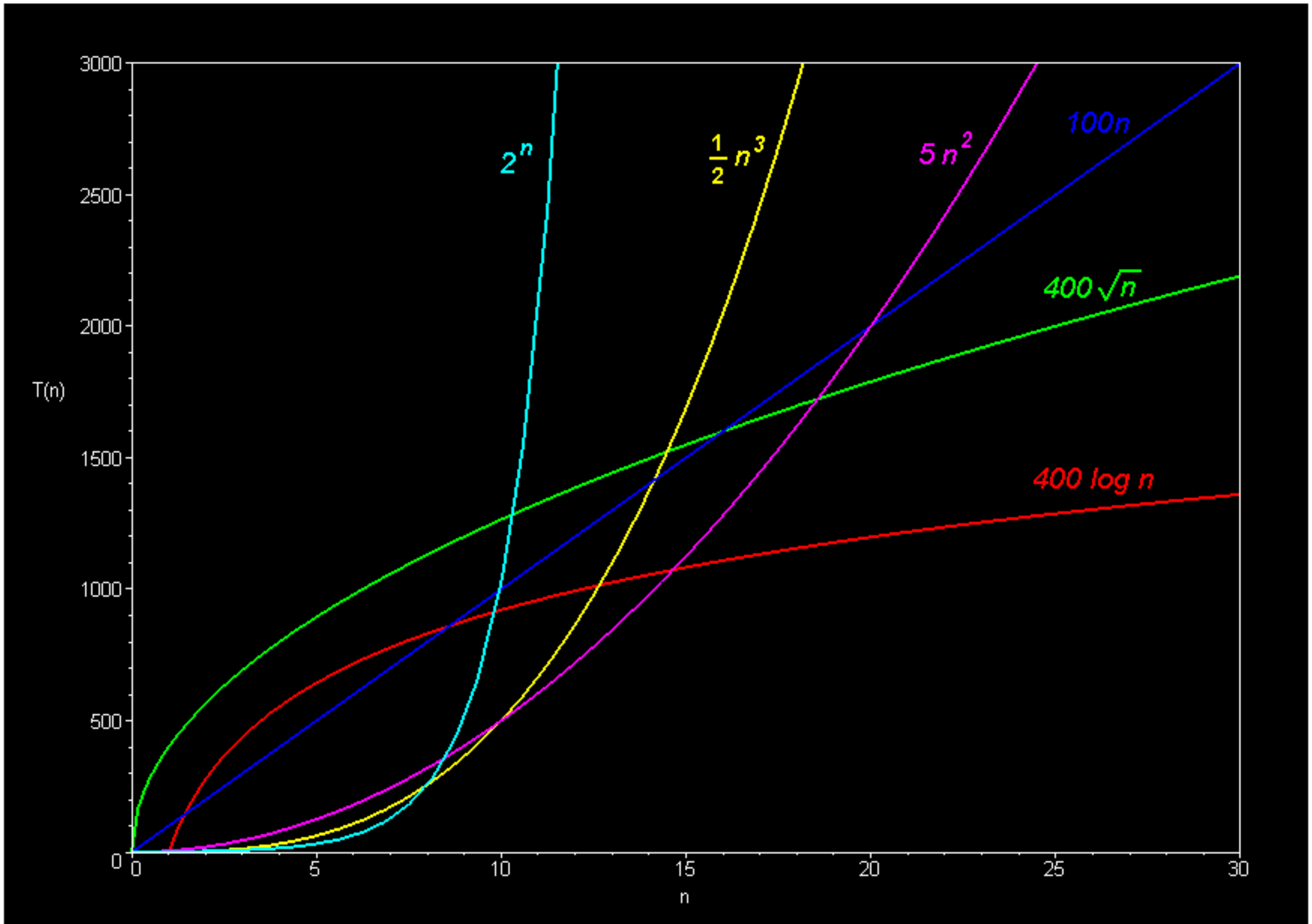
$f(n)$ is $O(g(n))$

if there are N and c such that

$\forall n \geq N,\ f(n) \leq c.g(n)$

The graph below compares the running times of various algorithms.

- Linear -- O(n)
- Quadratic -- O(n^2)
- Cubic -- O(n^3)
- Logarithmic -- O(log n)
- Exponential -- O(2^n)
- Square root -- O(sqrt n)



- ***Ignore*** additive constants

$$n^5+1000000 \quad \text{is } O(n^5)$$

- ***Absorb*** multiplicative constants

$$1000000n^5 \quad \text{is } O(n^5)$$

- Be as accurate as you can

$$O(n^2) \subset O(n^3) \subset O(n^4)$$

- Use and learn common terminology

logarithmic, linear,
polynomial, exponential

work

- $W(e)$, the *work* of e, is the time needed to evaluate e ***sequentially***, on a single processor
  - count each operation as constant-time
  - work = total number of operations
- Often have a function foo and a notion of size for *argument values*, and want to find $W_{foo}(n)$, the work of foo(v) when v has size n

May want *exact* or ***asymptotic*** estimate

Analyzing `rev`

```
(* rev : int list -> int list
   REQUIRES: true
   ENSURES: rev(L) returns a list that consists of
            L's elements in reverse order
*)

fun rev([] : int list) : int list = []
  | rev(x::xs : int list) : int list = rev(xs) @ [x]

(* op @ : int list * int list -> int list *)

infix @

fun @ ([]: int list, r: int list) : int list = r
  | @ (x::l, r) = x :: (l @ r)
```

Analyzing `append`

```
fun @ ([], r)= r
  | @ (x::l, r) = x :: (l@r)
```

size of first list   size of second list

Work of @ $W_{@}(n, m)$

Equation for base case:

$W_{@}(0, m) = c_0$ for some c_0, and all m

Equation for recursive clause for n > 0:

$W_{@}(n, m) = c_1 + W_{@}(n-1, m)$ for some c_1, and all m

Solving: $W_{@}(0, m) = c_0$

$W_{@}(n, m) = c_1 + W_{@}(n-1, m)$

Unrolling:

$$W_{@}(n, m) = c_1 + c_1 + W_{@}(n-2, m)$$

$$= c_1 + c_1 + c_1 + W_{@}(n-3, m)$$

……

$$= n.c_1 + c_0$$

Easy to prove by induction that $W_{@}(n, m) = n.c_1 + c_0$

O(n)

Analyzing `rev`

```
fun rev([])= []
  | rev(x::xs)= rev(xs) @ [x]
```

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) +$ __________

Analyzing `rev`

```
fun rev([])= []
  | rev(x::xs)= rev(xs) @ [x]
```

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Using lemma: for all list values `L`, `length(rev(L)) ≅ length L`

Analyzing `rev`

```
fun rev([])= []
  | rev(x::xs)= rev(xs) @ [x]
```

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

@ is O(n)

$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2.(n-1)$

$\leq c_1 + c_2.n + W_{rev}(n-1)$

Solving: $W_{rev}(0) = c_0$

$W_{rev}(n) \leq c_1 + c_2.n + W_{rev}(n-1)$

Unrolling:

$$W_{rev}(n) \leq c_1 + c_2.n + \{c_1 + c_2 .(n-1) + W_{rev}(n-2)\}$$

$$\leq c_1 + c_2.n + c_1 + c_2.(n-1) + \{c_1 + c_2.(n-2) + W_{rev}(n-3)\}$$

....

$$\leq c_0 + n.c_1 + ((n.(n+1))/2).c_2$$

$O(n^2)$

Analyzing `trev`

```
fun trev([], acc)= acc
  | trev(x::xs, acc)= trev(xs, x::acc)
```

$W_{trev}(0, m) = c_0$, for some c_0 and all m

$W_{trev}(n, m) = c_1 + W_{trev}(n-1, m+1)$, for some c_1 and all m

Can prove by induction that $W_{trev}(n, m)$ is $O(n)$

```
datatype tree = Empty | Node of tree * int * tree

(* sum : tree -> int
   REQUIRES: true
   ENSURES:  sum t returns the sum of all the integers in t
*)

fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Analysis of `sum`

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$$W_{sum}(0) = c_0$$
$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

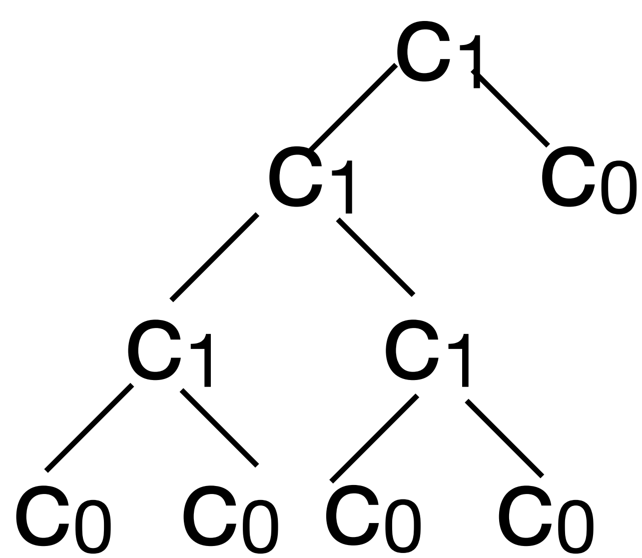
n_l: number of ints in the left subtree of t

n_r: number of ints in the right subtree of t

Solving: $W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

Tree method: write down work that occurs at each node and leaf



$W_{sum}(n) = c_1.n + c_0.(n+1)$ $O(n)$

Opportunity for parallelism

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$$S_{sum}(0) \;=\; c_0$$
$$S_{sum}(n) \;=\; c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$$

number of ints in the left subtree of t

number of ints in the right subtree of t

No balance assumption

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$$S_{sum}(0) = c_0$$

$$S_{sum}(n) \leq c_1 + \max(S_{sum}(n-1), S_{sum}(0))$$

number of ints in the left subtree of t

number of ints in the right subtree of t

No balance assumption

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$S_{sum}(0) = c_0$

$S_{sum}(n) \leq c_1 + S_{sum}(n-1)$ $\quad O(n)$

Assuming balance

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$$S_{sum}(0) = c_0$$

$$S_{sum}(n) \approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2))$$

number of ints in the left subtree of t

number of ints in the right subtree of t

$$S_{sum}(0) = c_0$$
$$S_{sum}(n) = c_1 + \max(S_{sum}(n/2), S_{sum}(n/2))$$

Unrolling:

$$\begin{aligned} S_{sum}(n) &\leq c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\ &\leq c_1 + S_{sum}(n/2) \\ &\leq c_1 + c_1 + S_{sum}(n/4) \\ &\quad \ldots\ldots \\ &\leq c_0 + (\lfloor \log n \rfloor + 1).c_1 \end{aligned}$$

$O(\log n)$

Opportunity for parallelism

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let n be the number of integers in a tree t

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

If tree is balanced span is O(log n)

Without that assumption it is O(n)

Using depth as a measure of size

```
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

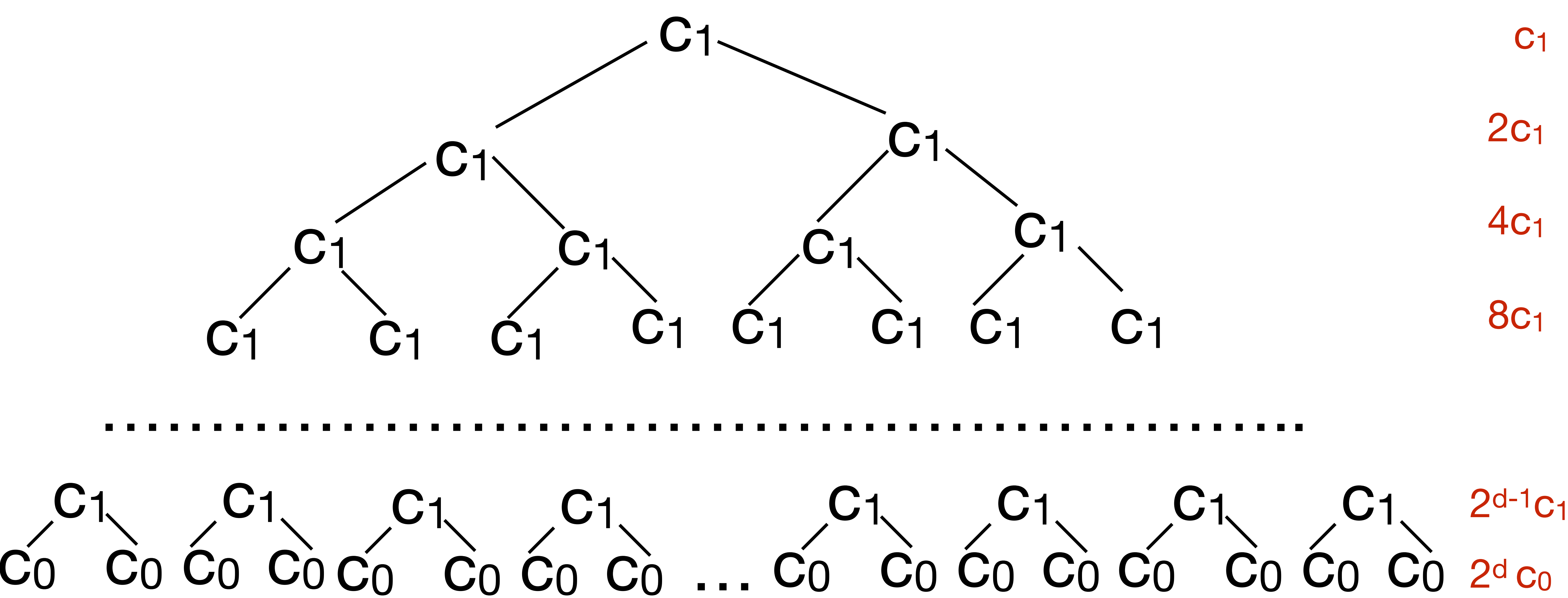
d-1 or smaller

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + S_{sum}(d-1)$

O(d) d is log(n) for balanced trees

Tree method for balanced trees



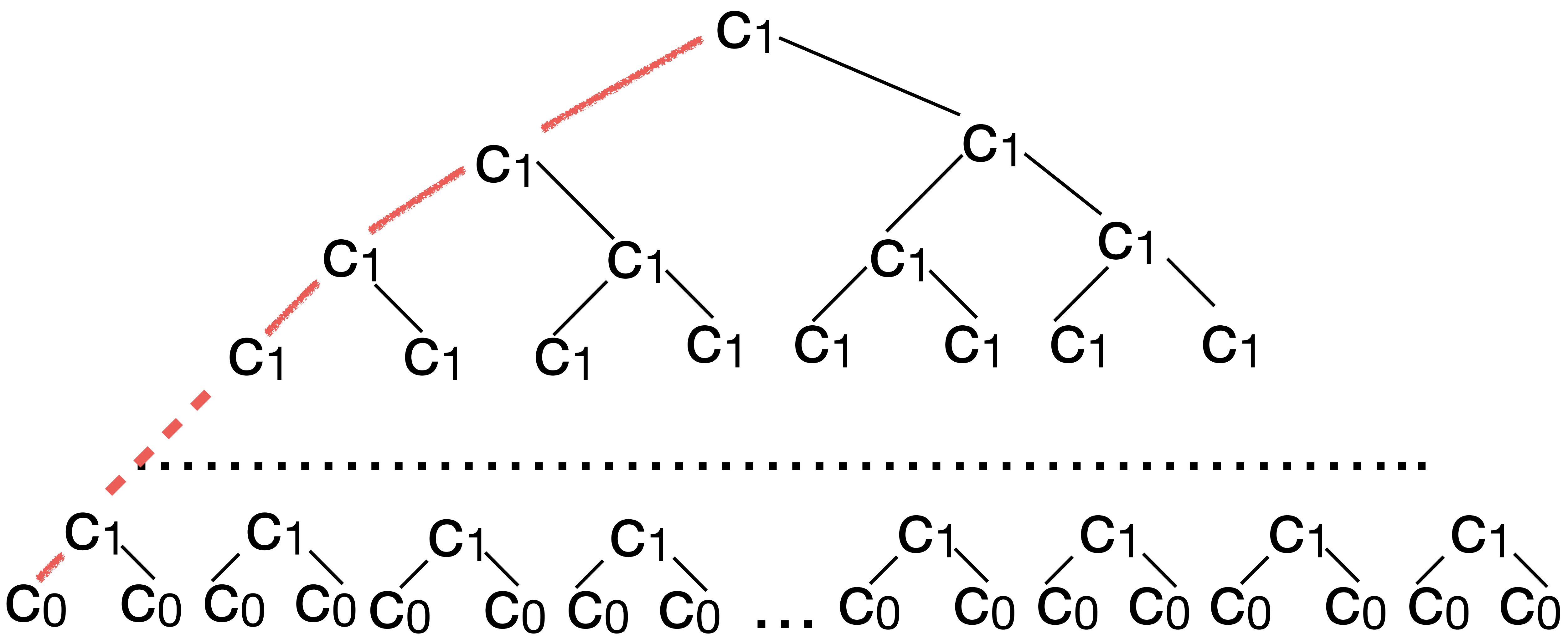
Tree method

$$S_{sum}(0) = c_0$$
$$S_{sum}(n) = c_1 + \max(S_{sum}(n/2), S_{sum}(n/2))$$

$$W_{sum}(n) = c_1.(1+2\ldots+2^{d-1}) + c_0.2^d \leq c.2^{d+1} \quad O(n)$$

↑ max(c1,c2)

Tree method for balanced trees



$$S_{sum}(n) = c_1.(1+1\ldots+1) + c_0 \leq c.(d+1)$$

$$S_{sum}(0) \;=\; c_0$$
$$S_{sum}(n) \;=\; c_1 + \max(S_{sum}(n/2), S_{sum}(n/2))$$

$$W_{sum}(n) \;=\; c_1.(1+2\ldots+2^{d-1}) + c_0.2^d \leq c.2^{d+1}$$

$$S_{sum}(n) \;=\; c_1.(1+1\ldots+1) + c_0 \leq c.(d+1) \quad O(\log n)$$

Sorting

Comparison

```
datatype order = LESS | EQUAL | GREATER

fun compare(x:int, y:int):order =
   if x<y then LESS else
   if y<x then GREATER else EQUAL
```

```
compare(x,y) = LESS                  if x<y
compare(x,y) = EQUAL                 if x=y
compare(x,y) = GREATER               if x>y
```

```
(* ins: int * int list -> int list
   REQUIRES: L is sorted
   ENSURES: ins(x,L) ==> a sorted permutation of x::L
*)


fun ins (x, []) = [x]
  | ins (x, y::ys) = (case compare(x,y) of
                        GREATER => y::ins(x,ys)
                        | _ => x::y::ys)


(* isort: int * int list -> int list
   REQUIRES: true
   ENSURES: isort(L) ==> a sorted permutation of L
*)

fun isort [] = []
  | isort (x::xs) = ins(x, isort xs)
```

```
fun ins (x, []) = [x]
  | ins (x, y::ys) = (case compare(x,y) of
                        GREATER => y::ins(x,ys)
                      | _ => x::y::ys)
```

$W_{ins}(n)$ the work for ins(x, L) L has length n

$W_{ins}(0) = c_0$
$W_{ins}(n) = c_1 + W_{ins}(n-1)$, for the first clause
$W_{ins}(n) = c_2$, for the second clause

$W_{ins}(n)$ is $O(n)$

```
fun isort [] = []
  | isort (x::xs) = ins(x, isort xs)
```

$W_{isort}(0) = c_0$

$W_{isort}(n) = c_1 + W_{ins}(n\text{-}1) + W_{isort}(n\text{-}1)$, for $n > 0$

$\leq c_1 + c_2.n + W_{isort}(n\text{-}1)$

$W_{isort}(n)$ is $O(n^2)$

standard results

- $T(n) = c + T(n-1)$ — $O(n)$
- $T(n) = c + n + T(n-1)$ — $O(n^2)$
- $T(n) = c + T(n \text{ div } 2)$ — $O(\log n)$
- $T(n) = c + 2\,T(n \text{ div } 2)$ — $O(n)$
- $T(n) = c + n + T(n \text{ div } 2)$ — $O(n)$
- $T(n) = c + n + 2T(n \text{ div } 2)$ — $O(n \log n)$
- $T(n) = c + k\,T(n-1)$ — $O(k^n)$