15-150
Fall 2026

Lecture 6

Asymptotic Analysis

Today

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

program → recurrence → work/span

Asymptotic analysis

How does the running time scale with the size of the input?

Asymptotic analysis

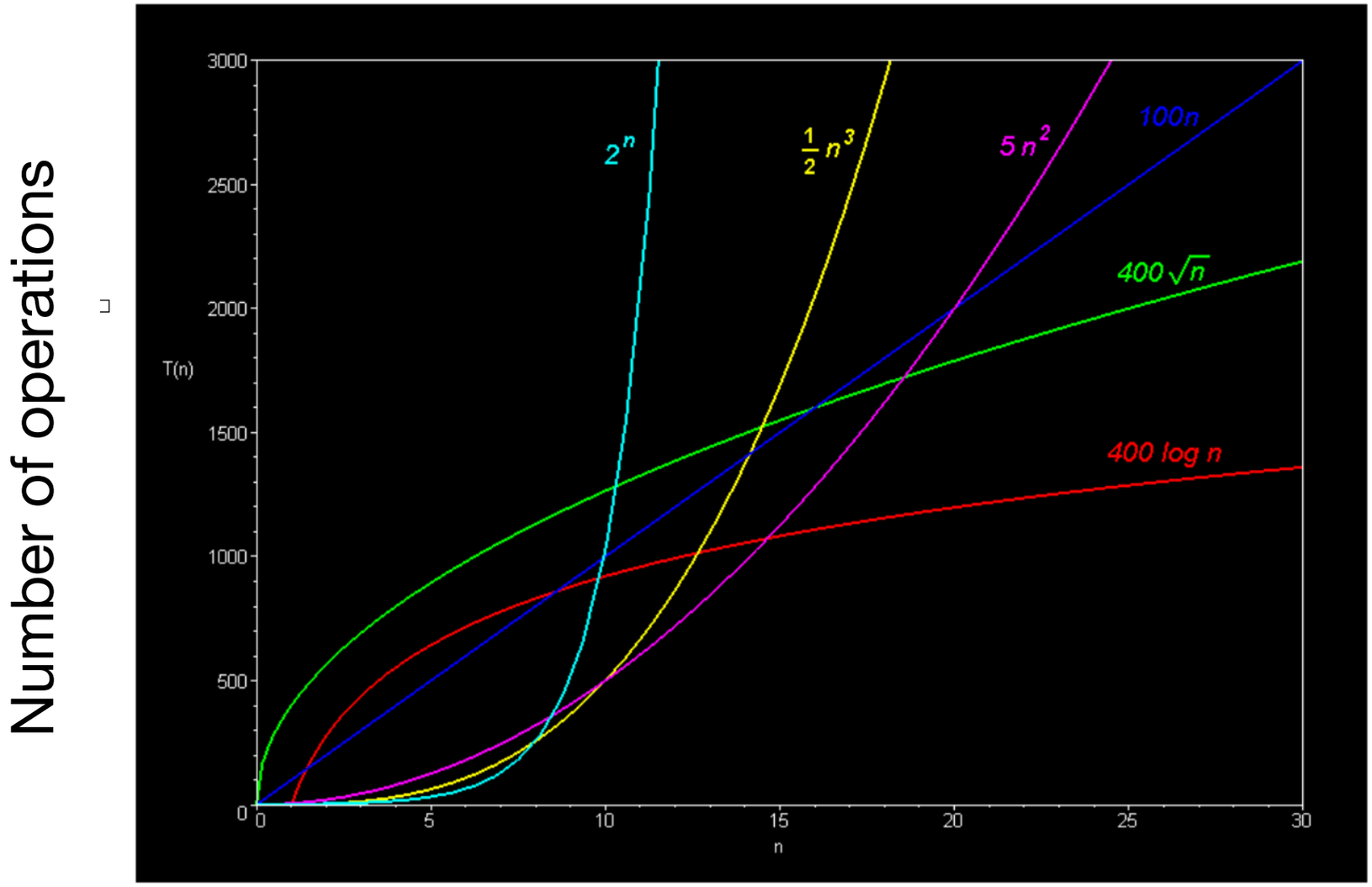
- We assume basic ops take ***constant time***
- Want to find running time *f(n)*, for ***large*** *n*
    - an *estimate*, independent of architecture
- Give big-O classification

f(n) is O(*g(n)*)

if there are *N* and *c* > *0* such

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

Some examples



- ***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

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]
```

previously defined as `append` in Lecture 4

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

infixr @

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

Analyzing @

```
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:

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

Equation for recursive clause for n > 0:

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

size of `x::l`

size of `l`

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

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

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

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

Unrolling:

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 $W_{@}(n, m) = n.c_1 + c_0$

$W_{@}(n, m)$ is O(n)

Analyzing `rev`

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

$W_{\texttt{rev}}(0) = c_0$

$W_{\texttt{rev}}(n) = c_1 + W_{\texttt{rev}}(n\text{-}1) + \_\_\_\_\_$

Analyzing `rev`

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

$W_{\texttt{rev}}(0) = c_0$

$W_{\texttt{rev}}(n) = c_1 + W_{\texttt{rev}}(n\text{-}1) + W_{@}(\quad,\quad)$

Analyzing `rev`

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

$$W_{\texttt{rev}}(0) = c_0$$
$$W_{\texttt{rev}}(n) = c_1 + W_{\texttt{rev}}(n\text{-}1) + W_{@}(n\text{-}1, 1)$$

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

Analyzing `rev`

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

$W_{\texttt{rev}}(0) = c_0$

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

Since @ is O(n), $W_{@}(n\text{-}1, 1) \leq c.(n\text{-}1)$ for some c

Analyzing `rev`

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

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

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

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

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

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

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

Unrolling:

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

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

....

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

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

Analyzing `trev`

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

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

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

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