15-150
Fall 2025

Lecture 6 - Part 2

Cost Analysis

Today

- Work (sequential runtime) and span (parallel runtime)
- Recurrence relations
- Exact and approximate solutions
- Improving efficiency

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

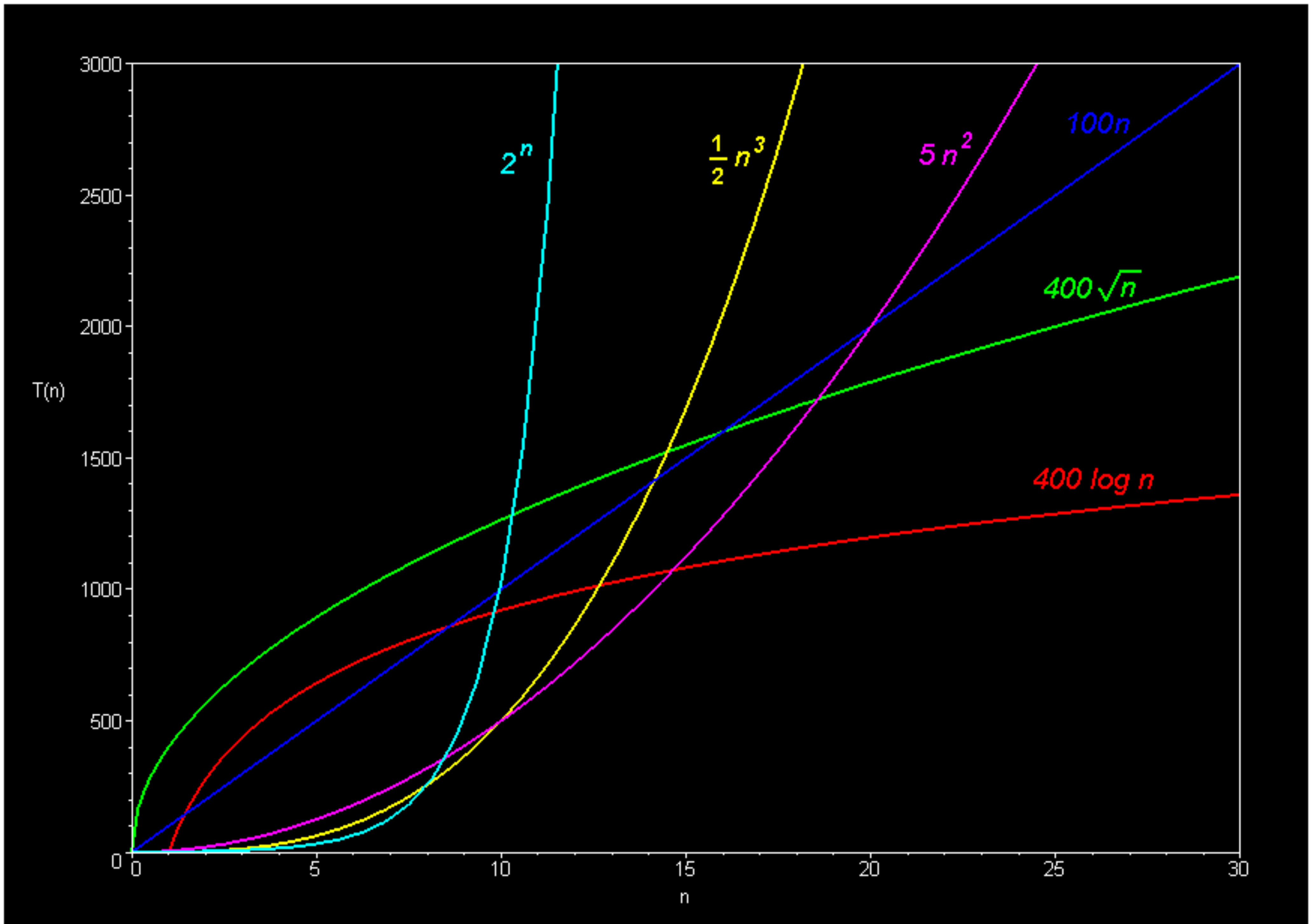
f(n) is O(*g(n)*)

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 -- $O(\text{sqrt } n)$



- ***Ignore*** additive constants

$n^5+1000000$ is $O(n^5)$

- ***Absorb*** multiplicative constants

$1000000n^5$ 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

Appending lists

```
(* @ : int list * int list -> int list
   REQUIRES:  true
   ENSURES:  @(l,r) returns the list consisting of l
             followed by r
   NOTE: this is also predefined in SML as the right-
         associative infix  operator @.
*)

infixr (op @);

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

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

Evaluating @

```
[1,2] @ [5,~6,7]   ==> 1 :: ([2] @ [5,~6,7])

                   ==> 1 :: (2 :: ([] @[5,~6,7]))

                   ==> 1 :: (2 :: [5,~6,7])

                   ==> 1 :: [2, 5,~6,7]

                   ==> [1, 2, 5,~6,7]
```

The last 2 lines are not really “steps”.
They are just different representations of the same value

Appending lists

```
(* @ : int list * int list -> int list
   REQUIRES:  true
   ENSURES:  @(l,r) returns the list consisting of l
             followed by r
   NOTE: this is also predefined in SML as the right-
         associative infix  operator @.
*)

infixr (op @);

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

What is the time complexity?   For a list with *n* elements, O(*n*)

For a list of length *len*, O(*len*)

Analyzing `append`

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

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: $W_@(0, m) = c_0$

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

Unrolling:

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

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

……

$= n.c_1 + c_0$

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

O(n)

To be continued next week!