Lecture 5 - January 21

Asymptotic Analysis of Algorithms

From Absolute RT to Relative RT
Approximating RT Functions
Asymptotic Upper Bound (Big-O): Def.

Announcements/Reminders

- Assignment 1 released
- ***splitArrayHarder***: an **extended** version released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Contact Information of TAs on common eClass site

Example 2: Counting Number of Primitive Operations

(Exercise)

```
1  boolean foundEmptyString = false;
2  int i = 0;
3  while (!foundEmptyString && i < names.length) {
4    if (names[i].length() = 0) {
5      /* set flag for early exit */
6      foundEmptyString = true;
7    }
8    i = i + 1;
9  }
```

Q. # of times Line 3 is executed?

Q. # of times loop body (Lines 4 to 8) is executed?

Q. # of POs in the loop body (Lines 4 to 8)?

Comparing Algorithms: From Absolute RT to Relative RT

t
exact time taken to execute a PO in some environment
exec.

e.g. Mac M1 t = 2 (ms)
e.g. Mac'14 t = 4 (ms)

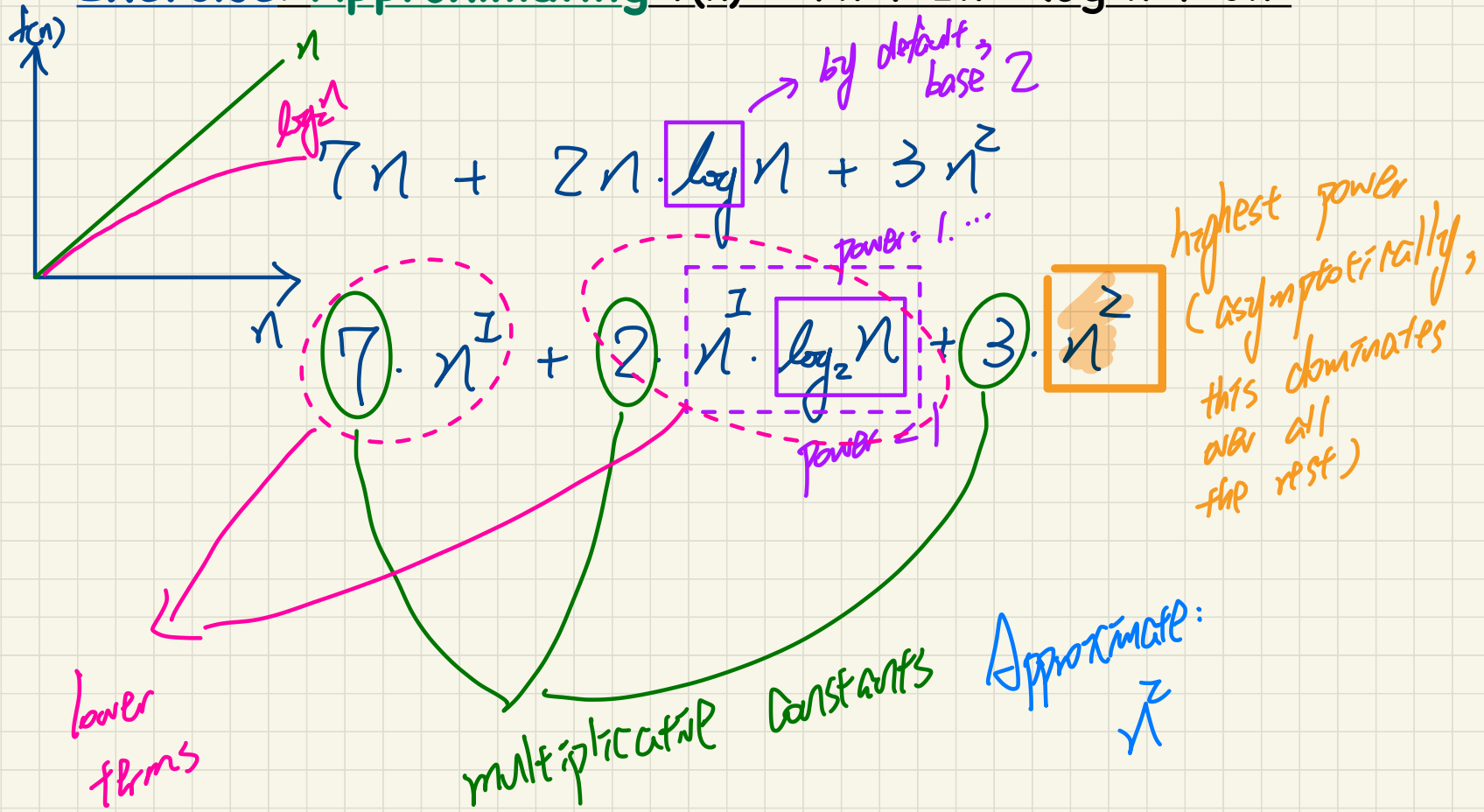
abs. RT	Mac M1	Mac'14
100-2	(100·1-2)·2	(100·1-2)·4
10n+3	(100·10+3) 2	

input size
assume n=100

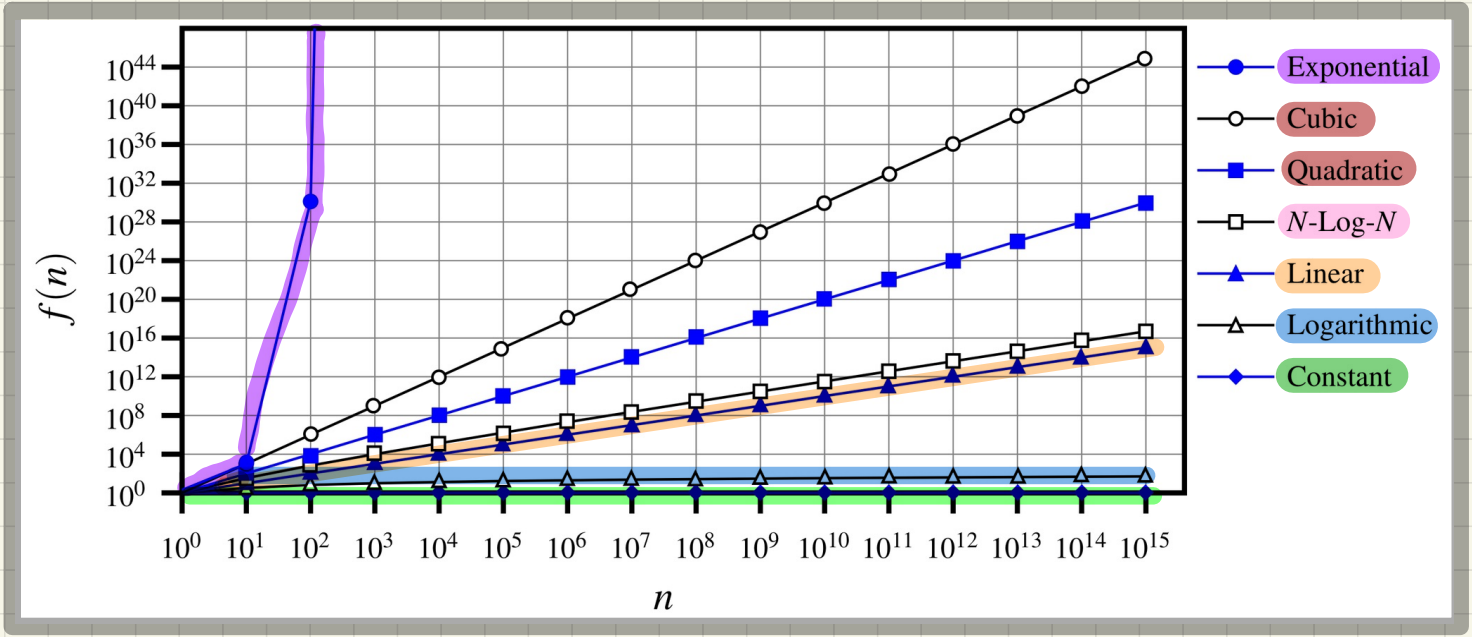
for the same alg., t is not useful (∵ we're supposed to compare it with another alg.).

to compare the RT of two alg. : not necessary → same conclusion can be drawn anyway.
to know the precise value of t (e.g. 2, 4)

Exercise: Approximating $f(n) = 7n + 2n \cdot \log n + 3n^2$



RT Functions: Rates of Growth (w.r.t. Input Sizes)



RT $f(n) = 1000$ $f(1M) = 1000$

$f(1) = 1000$ $f(1B) = 1000$

$f(100) = 1000$

Comparing Relative, Asymptotic RTs of Algorithms

Q1. Compare:

$RT_1(n) = 3n^2 + 7n + 18 \approx n^2$

$RT_2(n) = 100n^2 + 3n - 100 \approx n^2$

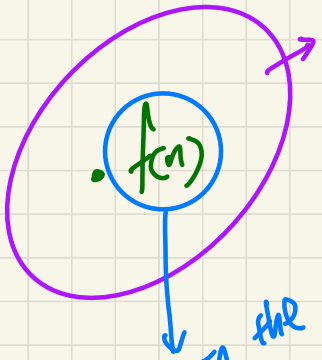
↳ equally efficient, asymptotically.

Q2: Compare:

$RT_1(n) = n^3 + 7n + 18 \approx n^3$

$RT_2(n) = 100n^2 + 100n + 2000 \approx n^2$

↳ RT_2 more efficient (taking less time) asymptotically.



$f(n)$: RT function

↳ input size ⇝ relative RT

$O(g(n))$

e.g. $f(n) = 7n - 2$

$g(n)$: reference function → $g(n) \rightsquigarrow c \cdot g(n)$ (modifies the slope of $g(n)$)

(further manipulation on $g(n)$ expected)

$f(n)$ being in the family means that it can somehow be upper-bounded by $g(n)$

Goal: Prove $f(n)$ is $O(g(n))$

Asymptotic Upper Bound: Big-O

upper bound effect.

$f(n) \in O(g(n))$ if there are:

- A real *constant* $c > 0$
- An integer *constant* $n_0 \geq 1$

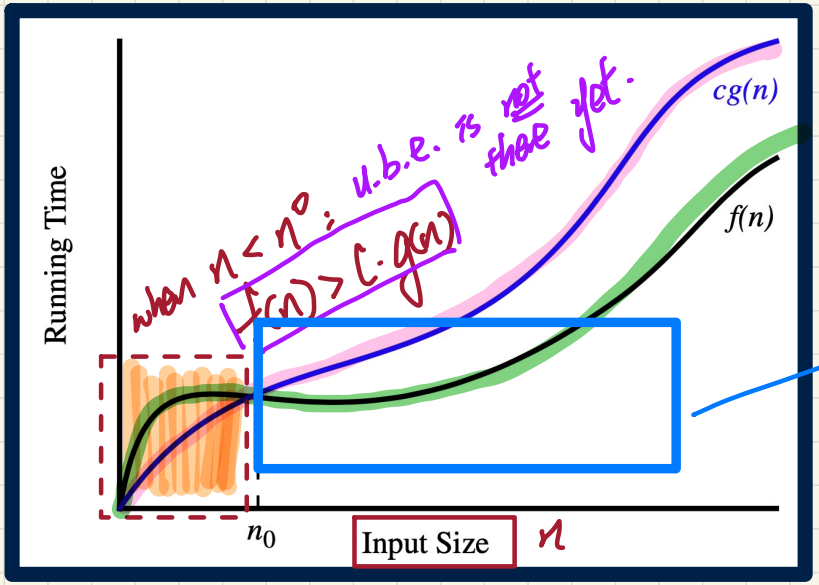
such that:

$$f(n) \leq c \cdot g(n) \quad \text{for } n \geq n_0$$

multiplicative constant applied to g(n) to change its slope

starting point of the u.b.e.

u.b.e.



Example:

f(n) = 8n + 5

g(n) = n

Prove:

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

Choose c = 9.

What about n0?

starting from $n = n_0$,

$f(n) \leq c \cdot g(n)$

upper bound effect is there!

Can n0 be 1?

$f(1) \leq 9 \cdot 1$

13 ↓ False

no u.b.e yet.

$* \; 1^0 = 1^2 = \cdots = 1^d = 1$

Proving f(n) is O(g(n))

We prove by choosing

$$\begin{array}{|l} c \\ n_0 \end{array} \; = \; \begin{array}{l} |a_0| + |a_1| + \cdots + |a_d| \\ \hline 1 \end{array}$$

If $f(n)$ is a polynomial of degree d, i.e.,

$$f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + \cdots + a_d \cdot n^d$$

and $a_0, a_1, \ldots, a_d$ are integers (i.e., negative, zero, or positive),

then $f(n)$ is $O(n^d)$. → $g(n)$

$** \; a_0 \le |a_0|$, $a_1 \le |a_1|$, ⋮ , $a_d \le |a_d|$

(1) $f(1) \le c \cdot 1^d$

(2) $f(n) \le c \cdot n^d$ $(n > 1)$

Upper-bound effect: $n_0 = 1$? $\quad [f(1) \le (|a_0| + |a_1| + \cdots + |a_d|) \cdot 1^d]$

$$f(1) = a_0 \cdot 1^0 + a_1 \cdot 1^1 + \cdots + a_d \cdot 1^d$$

$$\overset{*}{=} (a_0 + a_1 + \cdots + a_d) \cdot 1^d \overset{**}{\le} (|a_0| + |a_1| + \cdots + |a_d|) \cdot 1^d$$

Upper-bound effect holds? $\quad [f(n) \le (|a_0| + |a_1| + \cdots + |a_d|) \cdot n^d]$