CS180 Discussion 1B
Week 2: Algorithm Analysis

Ling Ding

Email: lingding@cs.ucla.edu

Many slides courtesy of former CS 180 TA Sherley Chen

Announcements

- **HW1 due:**
  - **11:59PM July 2nd, 2021**
  - Submit via CCLE-Gradescope
- **No homework this week**
- **No lecture next Monday (July 5th, 2021)**

Outline

- Algorithm Analysis
  - Computational Tractability
  - Asymptotic order of growth
  - Survey of common running times
- Exercises

Algorithm Analysis

(See separate slides)

Exercises

Asymptotic Bounds for Some Common Functions

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.
- Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n.
- Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants a, b > 0.

  can avoid specifying the base
- Logarithms. For every x > 0, $\log n = O(n^x)$.

  log grows slower than every polynomial
- Exponentials. For every r > 1 and every d > 0, $n^d = O(r^n)$.

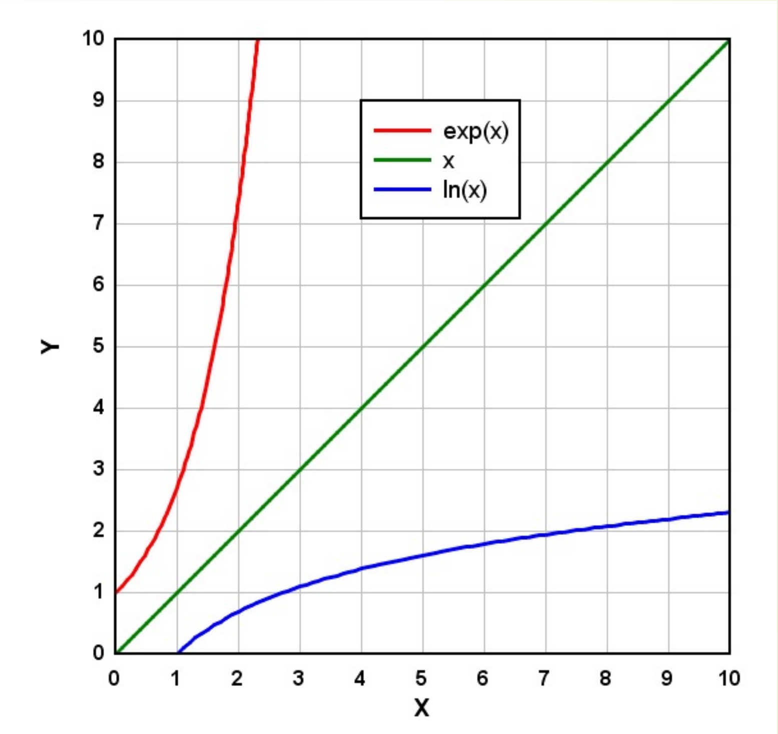
  every exponential grows faster than every polynomial

Asymptotic notations

- **Time complexity by counting operations**
- **big O notation**
  - upper bound
  - $f(n) = O(g(n))$
    - if there exists some constant c>0 such that, *for all large n*, f(n) ≤ c g(n).
    - $\boldsymbol{\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c}$
- Ω, Θ
  - $f(n) = \Omega\big(g(n)\big)$ lower bound
  - $f(n) = \Theta\big(g(n)\big)$
    - $f(n) = O\big(g(n)\big), f(n) = \Omega\big(g(n)\big)$

Special functions

- **Polynomial time**
  - $\boldsymbol{O(n^a)}$: polynomials, $\boldsymbol{a}$ is independent of n
    - $\boldsymbol{O(n)}$ linear
    - $\boldsymbol{O(n^2)}$ quadratic
    - $\boldsymbol{O(n^3)}$ cubic
  - $\mathbf{O(\log n)}$: logarithms
    - $O(\log\ n) = O(n^{\varepsilon})$
      - logarithms grow more slowly than polynomial
- **Non-polynomial time**
  - $O(n!)$: factorial
  - $O(3^n)$: exponential



Practice 1: big-O (functions)

$2n$	
$2n+3$	
$2n+10{,}000{,}000{,}000$	
$2n-10{,}000{,}000{,}000$	
$3n^2+2n+3$	
$3n^2+10000000000n+3$	
$2\log(n)$	
$2\log_{100}(n)$	
$2n+\log(n)$	
3^n+n^{100}	

$$\log_a b = \frac{\log_c a}{\log_c b}$$

Practice 1: big-O (functions)

$2n$	$O(n)$
$2n+3$	$O(n)$
$2n+10{,}000{,}000{,}000$	$O(n)$
$2n-10{,}000{,}000{,}000$	$O(n)$
$3n^2+2n+3$	$O(n^2)$
$3n^2+10000000000n+3$	$O(n^2)$
$2\log(n)$	$O(\log(n))$
$2\log_{100}(n)$	$O(\log(n))$
$2n+\log(n)$	$O(n)$
3^n+n^{100}	$O(3^n)$

$$\log_a b = \frac{\log_c a}{\log_c b}$$

Practice 2: big-O (Pseudo code)

- Assumption about unit operations:
  - compare two numbers
  - math operations (+, /, log, …)
  - assign value to an array element
  - …

Practice 2: big-O (Pseudo code)

- (1)

```
for (int i=1; i<=n; i++
{
      sum = sum + i;
}
```

```
operations:
i=1
loop n times:
      sum = sum + i
      i=i+1
      i<=n?
3n+1
```

O(n)

- (2)

```
for (int i=1; i<=n; i++)
{
      sum = sum + i;
      System.out.println("
Hello");
}
```

O(n)

Practice 2: big-O (Pseudo code)

- (3)

```
for (int i=1; i<=n; i=i+2)
{
      sum = sum + i;
}
```

```
n=5, i=1,3,5
n=10, i=1,3,5,7,9
loop times: n/2
```

O(n)

- (4)

```
for (int i=1; i<=n; i=i*2)
{
      sum = sum + i;
}
```

```
n=5, i=1,2,4
n=20, i=1,2,4,8,16
```

O(log n)

Practice 2: big-O (Pseudo code)

- (5)

```
for (int i=1; i<=n; i=i*3)
{
      sum = sum + i;
}
```

n=5, i=1,3
n=20, i=1,3,9
loop times: $\log_3(n) - 1$

O(log n)

- (6)

```
for (int i=1; i<=n; i++)
{
   for(int j=1; j<=n; j++)
      {
            A[i,j]=i*j;
      }
}
```

$O(n^2)$

Practice 3: Order functions

- Take the following list of functions and arrange them in ascending order or growth rate. That is, if function g(n) immediately follows function f(n) in your list, then it should be the case that f(n) is O(g(n)).

(a) n^2

(b) n^3

(c) $100n^2$

(d) $n \log n$

(e) 2^n

(f) 2^{2^n}

Thank you!