Analysis of Algorithms

How does an algorithm scale w/ increasing input size?

in tables:

list of topics you already know about

key words

space complexity

time complexity

asymptotic notation: big O (O), big theta (Θ), big omega (Ω)

types of algorithms – searching, sorting

classes of runtimes: n^2, $2n^2$, n, $n + 100$

worst case / best case / average case

polynomial vs exponential
efficient vs brute force

Today: #primitive operations alg uses
asymptotic notation intuition
exponent / logarithm rules

Algorithm 1

input: integer array of length n

initialize new int array B of length n (n assignments)

```
for i in 1, 2, ..., n:          # outer loop
    total = 0                   # assignment
    for j in 1, 2, ..., n:      # inner loop; assign
        total = total + A[i]    # retrieval arithm retrieval
    B[i] = total                # assign; retrieval
```

4 ops

How many primitive operations does Alg 1 use on input of size n?

Assume 1 primitive op for:

- basic arithmetic +, -, *, /
- variable assigment
- variable retrieval

total ops for inner loop: $5n$

number ops per outer loop: $4 + 5n$

overall # prim ops: $n(4+5n)+n$

$= 4n+5n^2+n = 5n^2+5n$

↳ n^2?

Def of exponiation:

$a^n = \underbrace{a \cdot a \cdot a \cdots a}_{n \text{ times}}$

properties:

$a^{n+m} = a^n a^m$ ←

$a^{n \cdot m} = (a^n)^m$

$(ab)^n = a^n b^n$

$$\frac{2^{2n}}{2^n} = 2^{2n-n} = 2^n$$

$= 2^n$

$$\frac{2^{2n}}{2^n} = \frac{2^{n+n}}{2^n} = \frac{2^n 2^n}{2^n}$$

def of logarithm: for $b \neq 1$, $b \in \mathbb{R}$, and $x \in \mathbb{R}^{\geq 0}$, the logarithm of x base b is $y \in \mathbb{R}$ such that $b^y = x$.

ex $\log_{\underset{b}{2}} \overset{x}{16} = \overset{y}{4}$ $\quad 2^{(4)} = 16$

Properties of Logarithms:

$$\log_b xy = \log_b x + \log_b y$$

$$\log_b \frac{x}{y} = \log_b x - \log_b y$$

$$\log_b x^y = y \log_b x$$

$$\log_b x = \frac{\log_c x}{\log_c b}$$