# Lecture-2 01.27.2023

Compare asymptotically

$n$ vs $n+1$ — equivalent

$n^2$ vs $(n+1)^2$ — $n^2$ vs $n^2+2n+1 \Rightarrow \Theta(n^2)$ — equivalent

$n^3$ vs $(n+2)^3$ — same reason as that — equivalent.

$2^n$ vs $2^{n+1}$ — $2^n$ vs $2 \cdot 2^n$ — equivalent

$2^n$ vs $3^n$ — $\lim_{n\to\infty} \frac{2^n}{3^n} \quad (\frac{2}{3})^n \Rightarrow 0$ — $3^n$ larger.

$n!$ vs $(n+1)!$ — $\lim_{n\to\infty} \frac{n!}{(n+1)!} = \frac{n!}{n!(n+1)} = \frac{1}{n+1} = 0$

$n^2$ vs $n^3$

$\log n^2$ vs $\log n^3$ — $\lim_{n\to\infty} \frac{\log n^2}{\log n^3} = \frac{2\log n}{3\log n} = \frac{2}{3}$ — same

$\log_q(n)$ vs $\log_2(n)$

$\log_a n = \frac{\log_2 n}{\log_2 a}$

$\lim_{n\to\infty} \frac{\log_q(n)}{\log_2(n)} = \lim_{n\to\infty} \frac{\frac{\log_2 n}{\log_2 q}}{\frac{\log_2 n}{\log_2 2}} = \frac{\frac{1}{\log_2 q}}{\frac{1}{\log_2 2}} = \frac{\frac{1}{\log_2 q}}{1} = \frac{1}{\log_2 q}$

$\frac{\log_q(n)}{\log_2(n)} \cdot \left(\frac{\frac{1}{\log_q 2}}{\frac{1}{\log_q 2}}\right)$, $\frac{1}{\log_q 2} = 1$

$\lim_{n\to\infty} \frac{\log_q(n)}{\log_2(n)} = \lim_{n\to\infty} \frac{\frac{\log_q(n)}{\log_q 2}}{\frac{\log_2(n)}{\log_q 2}} = \lim_{n\to\infty} \frac{\log_2(n)}{\log_2(n)} = \log_q 2$

$$\lim_{n\to\infty} \frac{\frac{\log_q(n)}{\log_q 2}}{\frac{\log_2 n}{\log_q 2}} = \frac{\log_2 n}{\log_2 n} \lim_{n\to\infty} \frac{\log_2(n)}{1} \cdot \frac{\log_q 2}{\log_2 n} = \log_q 2 = C$$

asymptotically equivalent

one was asymptotically larger than the other one.

$2^8 = 256$

3 2 1

000 000 000

$2^{38} = 256$ Billion

$2^7 = 128$

$2^{17} = 128000$

$2^3 \; 2^{23} = 8000000$

```
for i = 1 to n
    for j = 1 to n
        doit();
```

$\longrightarrow$ doit is $\theta(1)$

doit takes a constant $C_k$ time

$$f(n\ C_n) = n^2 \cdot C_k = \theta(n^2)$$

($C_k$ = assume in seconds)

| | $n^2$ | |
|---|---|---|
| $n = 10$ | 100 times | 1000 seconds |
| $n = 20$ | 400 | 4000 seconds |
| $n = 30$ | 900 | 9000 seconds |
| $n = 40$ | 1600 | 16000 seconds |

(2x, $2^2$x)

```
for i = 1 to n
    for j = 1 to i        j ≤ i
        doit()
```

$\{1, 2, 3, \ldots n\}$ $\quad \frac{n(n+1)}{2} \quad \frac{1}{2}n^2 + \frac{1}{2}n \quad \theta(n^2)$

```
for i = 1 to 10
    doit()        ⟶ (10 seconds)

for j = 1 to n
    doit()
    doit()
    doit()
```

how many times does do it get done.

$f(n) = 3n + 10$ → $\theta(n)$ → twice the input size will take twice as long.

| | | | |
|---|---|---|---|
| | $n = 100$ | $f(n) = 310$ | 3100 seconds = $f(n) \cdot 10$ seconds |
| 2× | | 3× | |
| | $n = 200$ | $f(n) = 610$ | 6100 seconds |
| 2× | | | |
| | $n = 300$ | $f(n) = 910$ | 9100 seconds |
| | $n = 400$ | $f(n) = 1210$ | 12,100 seconds. |

if a program processes 5000 items in 3 seconds.

| 5000 / 3 seconds | $\theta(1)$ | $\theta(n)$ | $\theta(n^2)$ | $\theta(n^3)$ | $\theta(2^n)$ |
|---|---|---|---|---|---|
| 10000 | } | 6 sec | 12 sec | 24 sec | } $\cdot 2^{5000}$ |
| 20000 | } | 12 sec | 48 sec | 192 sec | } $\cdot 2^{15,000}$ |
| 30000 | } | 18 sec | 108 sec | 648 sec | |
| 40000 | } | 24 sec | 192 sec | 1536 sec | |

$f(n) = (n-19)^2 \cdot 3$ $\quad O(n^2)$

| $n$ | running time | |
|---|---|---|
| $n = 20$ | 3 | $20 \cdot 2 = 40$ |
| $n = 40$ | $(40-19)^2 \cdot 3 = 21^2 \cdot 3 = 1323$ | |
| $n = 80$ | $(80-19)^2 \cdot 3 = 11163$ | |

$n = 1000$ $\quad = (1000-19)^2 \cdot 3 = 2887083$ $\quad$ 4x

$n = 2000$ $\quad = (2000-19)^2 \cdot 3 = 11773083$ $\quad$ 4x $\quad$ 9x

2x, 3x

$n = 3000$ $\quad = (3000-19)^2 \cdot 3 = 26659083$ $\quad$ 4x

2x

$n = 4000$ $\quad = (4000-19)^2 \cdot 3 = 47545083$

So that gives you an idea of what happens as you start getting some additional growth. And this then is a predictor once you understand the asymptotic growth rate you can start looking at what happens when your input sizes get larger and larger

How to drop your grades

1. not linear axes.
2. using a built in list.
3. not using table to support assumption bounds.
4. Having a graph / table that is inconsistent.

six pages.

probably three pages for problem number one

and three pages for problem number two.

32 million seconds / year $\approx 2^{25}$ seconds / year

the Sun is going to explode and the Earth will cease to exist in 4 billion years

4 billion $= 2^{32}$

$2^{25} \cdot 2^{32} = 2^{57}$ seconds

$2^n$

5000 3 sec

5001 6 sec

5002 12 sec

5003 24

5004 48

5005 96

5006 192

5057 → * Boom *

10000 →

0 digits $2^0$ value = 1 value 0

1 digit $2^1$ value = 2 value 0 1

2 digit $2^2$ value = 4 00 01 10 11

3 digit $2^3$ value = 8 000 001 010 011
100 101 110 111

$2^{10}$ ) $10^X$ ) $1024 = 1k$ $\times 2^{10}$
1 million

30 bit there are 1 Billion values I can represent.

you can brute force 56 bit Password in ~1 hour

How long would it take to Brute Force the 128-bit password?

∴ 128 bit Password $2^{128-56}$ · 1hr

$2^{74}$ hours

$\Theta(n!)$ 5000 3sec

factorial 5001 $3 \cdot 5001$

5002 $3 \cdot 5001 \cdot 5002$

permutation /ˌpɜːmjuˈteɪʃ(ə)n/ n. [数]排列；[数]置换

so there gives you an idea of why we care about running time.

sorting an array

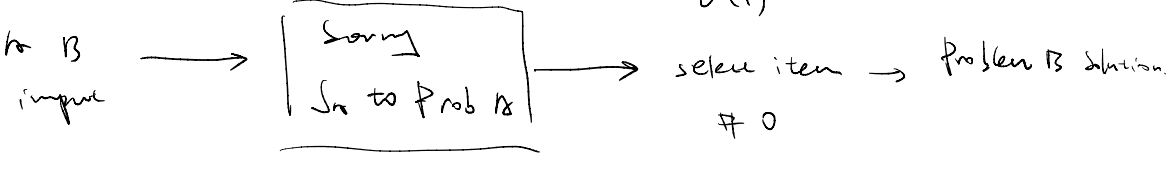
Problem A is just as hard or possibly harder than problem B within $\theta(1)$

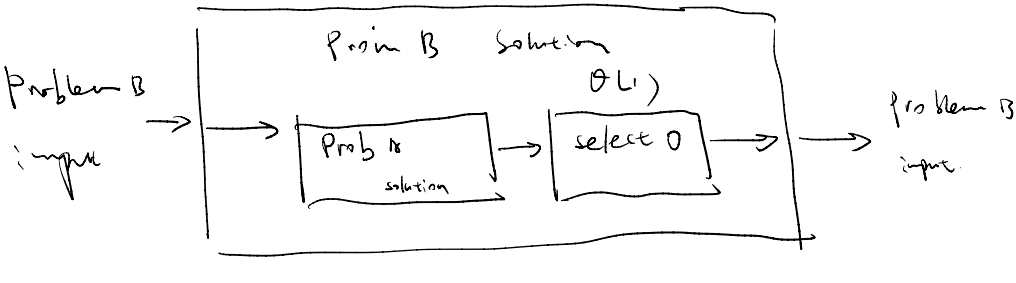
finding the min element in the array.

Assume $S_A$ is a solver for Problem A

if $S_A + \theta(1)$ extra work can solve Problem B then Problem A is just as hard or possibly harder [within $\theta(1)$] of Problem B.

$\theta(1) + Prob\ A \geq Prob\ B$



$11 \% 7 = 4$

$-2 \% 7 = 5$

$(11 + -2) \% 7 = 9 \% 7 = 2$

$(4 + 5) \% 7 = 9 \% 7 = 2$

$(a+b) \% 7 = (a \% 7 + b \% 7) \% 7$

$(7k + n) \% 7 = n$

$a \% 7 = n$

$7 + a \% 7 = n$

$2 \times 7 + a \% 7 = n$

$(k 7 + n) \% 7 = n$

$7 \% 5 = 2$

$-4 \% 5 = 1$

$(7 + -4) \% 5 = 3 \% 5 = 3$

$(7 + -4) \% 5 = (7 \% 5 + -4 \% 5) \% 5 = (2 + 1) \% 5 = 3 \% 5 = 3$

$(7 \cdot -4) \% 5 = (-28) \% 5 = -3 \% 5 = 2$

$(7 \cdot -4) \% 5 = [(7 \% 5) \times (-4 \% 5)] \% 5 = (2 \times 1) \% 5 = 2$

$9135267 \equiv 4 \pmod 9$

$\times\ 6953472218 \equiv 2 \pmod 9$

---

$\equiv 8 \pmod 9$

$318 \pmod 9$

$(3\times10^2 + 1\times10^1 + 8\times10^0) \pmod 9$

$1 \equiv 1 \pmod 9$

$10 \equiv 1 \pmod 9$

$100 \equiv 1 \pmod 9$

$1000 \equiv 1 \pmod 9$

$10^n \equiv 1 \pmod 9$

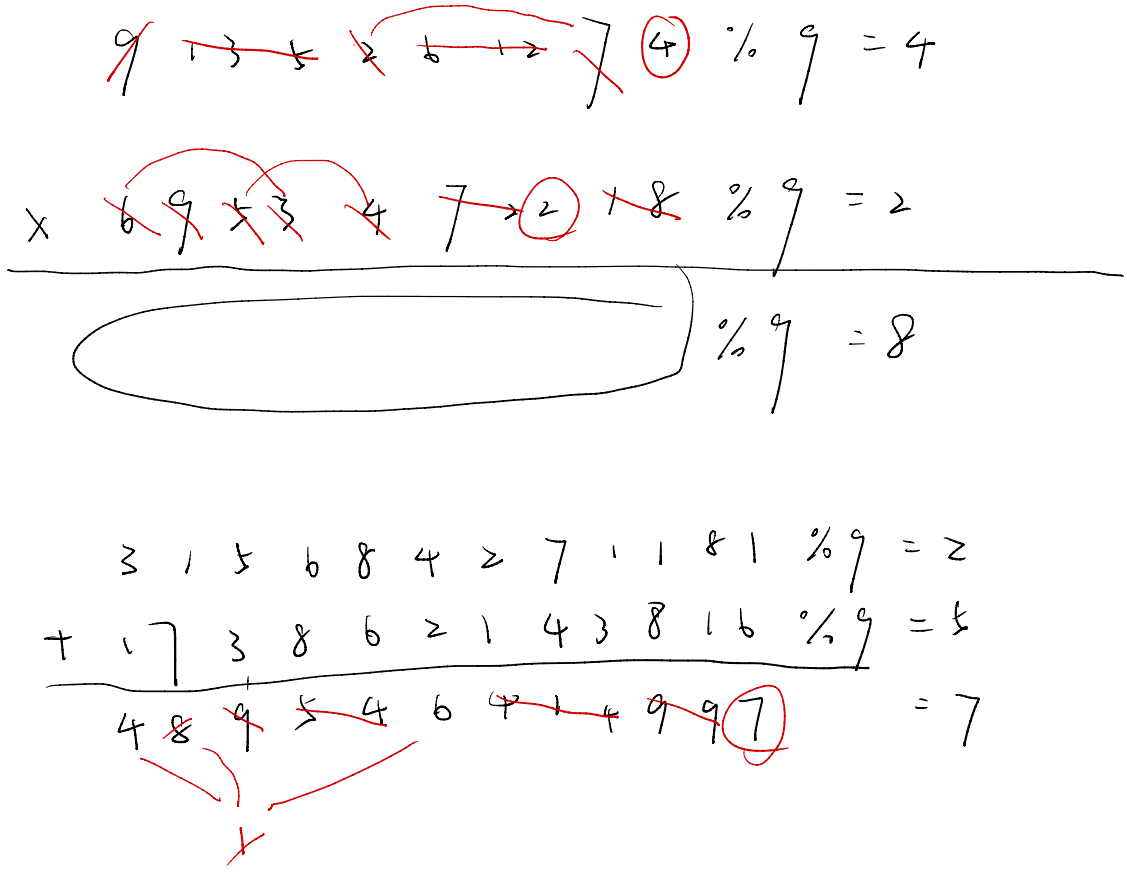
$(3\times10^2 \pmod 9 + 1\times10^1 \pmod 9 + 8\times10^0 \pmod 9) \pmod 9$

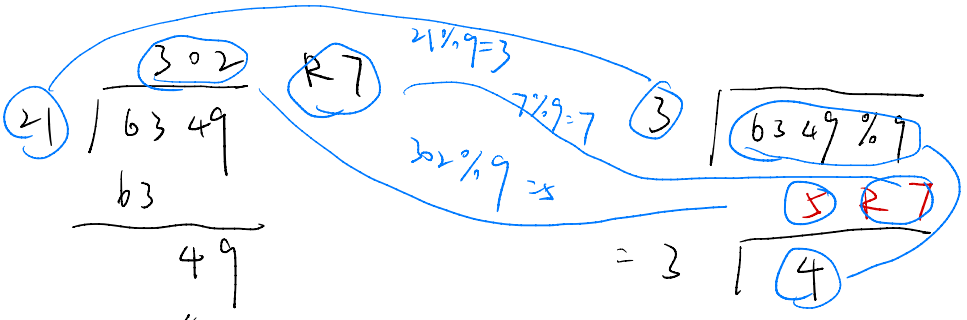
$(3\times1 + 1\times1 + 8\times1) \pmod 9$

$(3+1+8) \pmod 9$

$(3+9) \pmod 9$

$(3 \pmod 9) + 9 \pmod 9 = 3 \pmod 9 = 3$






remainder of 7

$21 \times 302 + 7 = 6349$

$(3 \times 5 + 7) \% 9 = 4$

$(15 + 7) \% 9$

$= 22 \% 9$

$= 4$

casting out 9's

discrete by Pr — we don't know how to back

$2^0 \% 11 = 1$

$2^1 \% 11 = 2$

$2^2 \% 11 = 4$

$2^3 \% 11 = 8$

$2^4 \% 11 = 5$

$2^5 \% 11 = 10$

$2^6 \% 11 = 2 \cdot (2^5 \% 11) \% 11 = 2 \cdot 10 \% 11 = 9$

$2^7 \% 11 = 2 \cdot (2^6 \% 11) \% 11 = 2 \cdot 9 \% 11 = 7$

$2^8 \% 11 = 3$

$2^9 \% 11 = 6$

$2^{10} \% 11 = 1$

$2^n \% 11 = 3$

$2^{n+1} \% 11 = 2 \cdot (2^n \% 11) \% 11 = 2 \cdot 3 \% 11 = 6$

$2^{35} \% 11 = ((2^5 \% 11)(2^{10} \% 11)(2^n \% 11)) \% 11 = (10 \times 1 \times 10) \% 11 = 1$

$2^5 \% 11 = 2 \cdot (2^4 \% 11) \% 11$
$= 2 \cdot 5 \% 11$
$= 10$

$3^{137} \% 11$

$3^0 \% 11 = 1$

$3^1 \% 11 = 3$

$3^2 \% 11 = 9$

$3^3 \% 11 = 27 \% 11 = 5$

$3^4 \% 11 = 3\,(3^3 \% 11) \% 11 = 3 \times 5 \% 11 = 4$

$3^5 \% 11 = 3\,(4) \% 11 = 1$

$3^8 \% 11 = \left[(3^4 \% 11)\,(3^4 \% 11)\right] \% 11 = (4 \times 4) \% 11 = 16 \% 11 = 5$

$3^{16} \% 11 = (5 \times 5) \% 11 = 25 \% 11 = 3$

$3^{32} \% 11 = (3 \times 3) \% 11 = 9 \% 11 = 9$

$3^{64} \% 11 = (9 \times 9) \% 11 = 81 \% 11 = 4$

$3^{128} \% 11 = (4 \times 4) \% 11 = 16 \% 11 = 5$

$3^{137} \% 11 = (3^{128} \cdot 3^8 \cdot 3^2 \cdot 3^1) \% 11 = (5 \cdot 5 \cdot 9 \cdot 3) \% 11$

$= (25 \times 27) \% 11$

$= (3 \times 5) \% 11$

$= 15 \% 11$

$= 4$

It's the fundational for the few elementary exchanges in RSA. When you bring up web browser and it says https and it encrypts it. your browser and the host that you're talking to does what is called a diffie-hellman key to exchange to determine what they're going to use for the encryption key and a key exchange is an idea that I can talk to you if we can exchange information and everybody else in this classroom can listen to us and even though they hear everything that we have said they will not know what our secret encryption key is that we're going to use for encrypting our session. and it relies on being able to raise a value to a power module of something.

It's easy to go forward I could figure out what $\sum^{5} \% 11 = 10$. but we don't know how to efficiently go backwards that's called the discrete log problem.