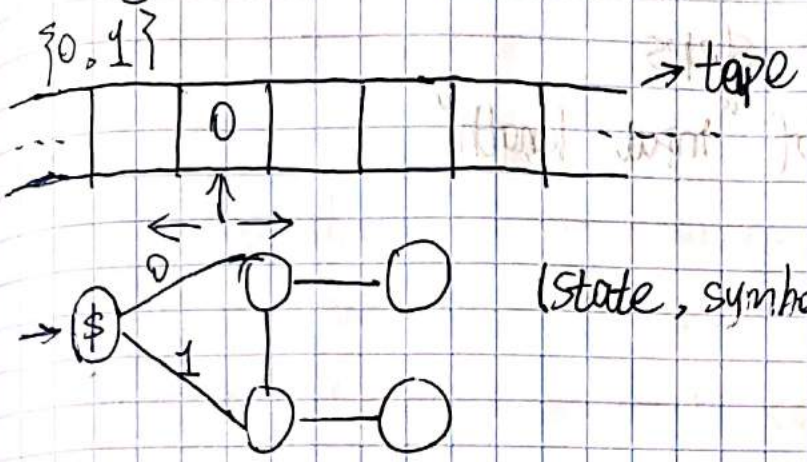
09/09 Lecture 01

- Turing Machine



(State, symbol) $\xrightarrow{f}$ (Symbol $\overset{\text{move}}{\rightleftarrows}$ state)

- Von-Neumann machine:

- An algorithm:

A Turing Machine that halts after a finite # of transitions.

Any Program. ~~return~~ written programming language that halts.

- There are legitimate problems for which there are <u>no algorithms</u>.

- Halting ~~Program~~ Problem: infinit loops

Program input

Halt (P)

: undecidable:

< yes / No

- Test (P):

```
    if Halt(P).
        while true;
    else
        end;
```

Test (test)

- efficiency (complexity) of Algorithm:

• Resources:

- time $\approx$ Number of basic steps as a function of "input length"

ex). list = L = $[m_1, m_2, \ldots, m_m]$

```
Search (L, K)
    for i = 1, ....., m
        if K == L[i]
            return (i, K).
    return (null)
```

constant

• The length of L: m

• Worst-case complexity: $f(m) \approx c \cdot m$

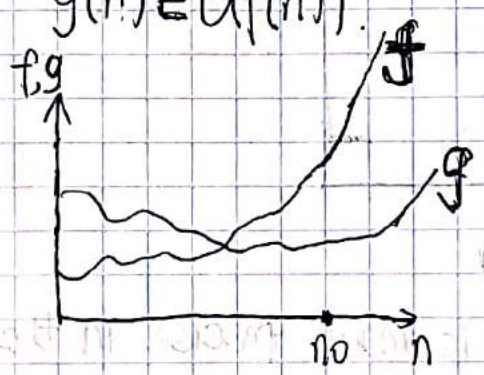
c ↓ constant

• Average-case complexity:

- Time-complexity of an Algorithm on an input $|L| = n$
  a function ~~that~~ $f(n)$ that given the number of steps the algorithm has to take in the worst ~~case~~ case scenario

- Big O notation: upper bounding
  Suppose we have $f(n)$:
  $O(f(n)) = \{g(n) \mid$ there exists constants, $c > 0$, n_0: $g(n) \leq c \cdot f(n)$ for all $n > n_0\}$

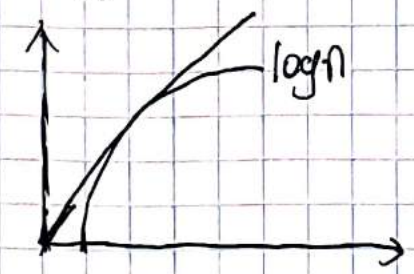
$g(n) \in O(f(n))$



ex). ① $2n+1 \in O(n)$ ② $n^2 \notin O(n)$ / $n = O(n^2)$

~~② $n^2 = O(n)$~~

③ $\log n = O(n)$ / $n \neq O(\log n)$ $\boxed{\log_2 n = \frac{\log_{10} n}{\log_{10} 2}}$



Binary Search

— Binary Search: (L, K)

$L = [n_1 \le \cdots \le n_m]$

n_1	n_2	$\cdots$	$n_{\frac{m}{2}}$	$\cdots$	n_m

$c > 0$

$\lfloor x \rfloor$: $\max\{m \in \mathbb{Z} \mid m \le x\}$
floor fun.
向下取整

if $L[\lfloor \frac{m}{2} \rfloor] == K$
    return $(\lfloor \frac{m}{2} \rfloor, K)$
if $L[\lfloor \frac{m}{2} \rfloor] < K$
    binary search $(L[\lfloor \frac{m}{2} \rfloor + 1 : m], K)$
if $L[\lfloor \frac{m}{2} \rfloor] > K$
    binary search $(L[1 : \lfloor \frac{m}{2} \rfloor - 1], K)$

$\Theta(\log m)$

$\{3, 5, 7, 9, 11, 13, 15\}$ → mid
$X = 200$

— if $\begin{cases} g(n) \in O(f(n)) \\ f(n) \in O(g(n)) \end{cases}$

$\therefore g(n) = \Theta(f(n))$

$3n+1 \in \Theta(n)$.
$n + \log n \in \Theta(n)$.
$n^2 = O(n^2+n)$.

— Big Theta $\Theta(\cdot)$

The big theta is more precise in that we assume that both lower bound and upper bound.

- Big Ω function: lower bounding

$\Omega(f(n)) = \{ g(n) \mid$ there are constants c, n_0: $cf(n) \leq g(n)$, for all $n > n_0 \}$

ex). $n \in \Omega(\log(n))$

- if $\begin{cases} f(n) \in O(g(n)) \\ g(n) \notin O(f(n)) \end{cases} \Rightarrow f(n) < g(n)$.

if $f(n) = \Theta(g(n)) \Rightarrow f(n) \approx g(n)$

$$\underbrace{1 \leq \log(n) \leq \log^2(n) \leq \log^c(n) \cdots \leq n \leq n\log n \leq n^2 \leq n^c}_{\text{Polynomial-time Algorithm "efficient algorithm" ≈ "good algorithm"}} \leq n^{\log n}$$

$c \geq 2$.

$$\underbrace{\leq 2^n \leq 3^n \leq n!}_{\text{non-efficient}} \leq n^n$$

ex. Finding min in an unordered list of number. $\Omega(n)$:
↳ lower bound

09/16

- Comparables:

Objects: $\leq$ is a total orders

a, b are "objects"

(1). $a \leq a$ (reflexive)

(2). $a \leq b$ and $b \leq a$, then $a = b$ (anti-symmetry)

(3). $a \leq b$ and $b \leq c$. then $a \leq c$ (transitive).

(4). for any pair objects, a, b, either $a \leq b$ or $b \leq a$

partial order

total order

- Finding the Minimum

• Input: A list of comparable $L = [a_1 \ldots a_n]$

• Output: Minimum L_i

- Method:

(1). $min = \infty$] c_0

(2). For $i = 1, \ldots, n$
  if $L[i] < min$
    $min = L[i]$] cn

(3). return$(L[i], i)$] $c \cdot n + c = \boxed{O(n)}$

$\boxed{\Theta}(n)$.

$\Theta(1)$??

$\Theta(\cdot)$.

- $(m, i) = Min(L, n)$ ⇝ recurrsive ??

if ~~length(L)=1~~ $n == 1$

then return $(L[1], 1)$.

else $(m', i') = Min(L[2:n])$.

if $m' < L[1]$ return (m', i')

else return $(L[1], 1)$.

$$f(n) = \begin{cases} \text{if } n = 1 & C_1 \\ f(n-1) + C \end{cases}$$

$$\boxed{\begin{aligned} f(1) &= C_1 \\ f(n) &= f(n-1) + C \end{aligned}}$$

difference equation

- $min(L) = min(L_1, min(L_{2:n}))$
- $f(n) = C + f(n-1)$

$= C + (f(n-2) + C) = 2C + f(n-2)$.

$= 2C + (f(n-3) + C) = 3C + f(n-3)$.

$\vdots$

$= (n-1)C + f(n-(n-1))$

$= (n-1)C + f(1)$

$= (n-1)C + C_1$

$= O(n)$.

- $V_1 = (a_1 \dots a_n)$.

$V_2 = (b_1 \dots b_n)$

$V_1 + V_2 = (a_1 * b_1, a_2 \times b_2 \dots a_n \times b_n)$.

arithmetic complexity O_A

~~algebra~~ complexity

algebraic

$O_B(n \cdot K^2)$.

- Bit complexity

- Select Sort:

- Input: list of comparable $L[L_1, \dots L_n]$
- Output: $L[L'_1 \dots L'_n]$: $L'_1 \leq L'_2 \leq \dots \leq L'_n$
- Method:

for $i = 1, \dots, n-1$

$(m', k) = \min(L[i:n])$ $\rightarrow O(n^2)$

$\text{Swap}(L[k], L[i])$ $\rightarrow C$

- 1st iteration: $C(n-1) + C_1$

2nd iteration: $C(n-2) + C_1$ $+$

$n-1$ iteration: $C(2) + C_1$

$\rightsquigarrow (n-1)C_1 + C((n-1) + (n-2) + \dots + 1)$

$C_1(n-1) + C_1\underbrace{(1 + 2 + \dots + n-1)}_{\text{arithmatic progression}}$

$$\begin{cases} S = 1 + 2 + \dots + (n-1) \\ S = (n-1) + \dots + 2 + 1 \end{cases}$$

$\Rightarrow 2S = n + \dots + n \Rightarrow S = \frac{n(n-1)}{2} = O(n) + O(n^2)$

$= O(n^2)$

Merge Sort

— Merge Sort ($L = [a_1, \dots, a_n]$).

- input:
- Output: $[a_1', \dots a_n']$; $a_1' \le \dots \le a_n'$ ⇝ Ready order
- Method:

if $n == 1$ return L

else (1). L_1 = MergeSort ($L_1 \dots L_{\lfloor \frac{n}{2} \rfloor}$)

(2). L_2 = MergeSort ($L_{\lfloor \frac{n}{2} \rfloor + 1} \dots n$).

(3). L = Merge (L_1, L_2). → $C_1 n$

(4). return L.

$f(n) = C + f(\frac{n}{2}) + f(\frac{n}{2}) + C_1 \cdot n$

$$\boxed{f(n) = C_1 n + 2f(\tfrac{n}{2})}$$
$$\boxed{f(1) = 1}$$

$n = 2^k$ $k = \log_2 n$.

$f(n) = C_1 n + 2f(\frac{n}{2})$

$f(2^k) = C_1 \cdot 2^k + 2f(2^{k-1})$

$f(2^0) = 1$

define $g(k) = f(2^k)$.

$$\boxed{g(k) = C_1 \cdot 2^k + 2g(k-1)}$$
$$\boxed{g(0) = 1}$$

$g(k) = C_1 \cdot 2^k + 2(C_1 \cdot 2^{k-1} + 2 \cdot g(k-2))$.

$= C_1 \cdot 2^k + C_1 \cdot 2^k + 2^2 g(k-2)$

$= C_1 \cdot 2^k + C_1 \cdot 2^k + 2^2 (\underbrace{C_1 \cdot 2^{k-2} + 2g(k-3)}_{g(k-2)})$

$= C_1 \cdot 2^k + C_1 2^k + C_1 \cdot 2^k + 2^3 g(k-3)$.

$= 3C_1 \cdot 2^4 + 2^3 g(k-3)$

$= i C_1 \cdot 2^k + 2^i g(k-i)$.

$= k C_1 \cdot 2^k + 2^k g(0)$

$= C_1 k \cdot 2^k + 2^k$

$= 2^k (C_1 \cdot k + 1)$

See next page

ex). for min: tail recussion

$f(n) = C \cdot g(n) + Cf(n-1)$

$= C \cdot g(n) + C(g(n-1) + Cf(n-2))$.

$= C \cdot (g(n) + g(n-1) + \dots)$

— Algorithm Merge: $L_1 = [a_1, \dots a_n]$; $L_2 = [b_1 \dots b_n]$

- Input : two sorted L_1, L_2
- Output: $L\{a_i, b_i\}$ sorted

n_1: $[1, 3_a, 5]$ n_2: $[2, 3_b, 7]$

1	2	3_a	3_b	5	7

$O(n_1 + n_2)$

↳ stable sorting

(1). Compare (expensive)

(2). move/swap (cheapter)

↳ $g(k) = (c_1 \cdot k + 1) 2^k$

$f(2^k) = (c_1 k + 1) 2^k$

$f(n) = (c_1 \log_2 n + 1) \cdot n = O(n \log n)$

- Geometric Progression

$$a + ab + ab^2 + \cdots + ab^n = a(1 + b + b^2 + \cdots + b^n)$$

$$S = 1 + b + b^2 + \cdots + b^n$$

$$bS = b + b^2 + b^3 + \cdots b^n + b^{n+1}$$

$$bS - S = b^{n+1} - 1$$

$$S(b-1) = b^{n+1} - 1$$

$$\Rightarrow S = \frac{b^{n+1} - 1}{b-1} = \frac{1 - b^{n+1}}{1-b}$$

- Convolution of two lists of numbers

$\boxed{a_1 \cdots a_m}$ $\boxed{b_1 \cdots b_m}$

$$f(x) = a_0 + a_1 x + a_2 x^2 \cdots + a_n x^n$$

$\boxed{a_0 | a_1 | \cdots | a_n}$

$$g(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n$$

$$f(x) + g(x) = a_0 + b_0 + (a_1 + b_1)x + \cdots + (a_n + b_n)x^n$$

$O(n)$

- $f(x) \cdot g(x) = C_0 + C_1 x + \cdots + C_{2n} x^{2n}$

$a_0 b_0$						
0	1	2 ...	n ...			2n

$\rightarrow C_0 + C_1 x + \cdots$

$C_0 \downarrow a_0 b_0$ $\quad C_1 \downarrow a_0 b_1 + a_1 b_0$

$\rightarrow C_k = a_0 b_k + a_1 b_{k-1} + a_2 b_{k-2} + \cdots + a_k b_0$

Can I only focus on the last part of $f(x)$ & $g(x)$ to get $O(n^2)$?

$=$

Convolution: $f(n) = 2\sum_{k=0}^{n} [k + (k-1)] = 2\left[\sum_{k=0}^{n} k + \sum_{k=0}^{n} (k-1)\right]$

$\downarrow$

multipule polynomials

$(1+2+\cdots+n)$ $\quad (1+2+\cdots+n-1)$

$$\frac{n(n+1)}{2} + \frac{n(n-1)}{2} = n(n+1) + n(n-1)$$

$$\Rightarrow O(n^2).$$

$a_0 + a_1 x \cancel{+ a_2 x^2}$

$b_0 + b_1 x \cancel{+ b_2}$

Product : $a_0 b_0 + (a_0 b_1 + a_1 b_0) x + (a_1 b_1) x^2 \rightsquigarrow$ 4 multi , 1 add

$S_0 = a_0 b_0$. $\quad \rightarrow S_0 + (S_2 - S_0 - S_1) x + S_1 x^2 \rightsquigarrow$ 3 multi, 4 add

$S_1 = a_1 b_1$

$S_2 = (a_0 + a_1)(b_0 + b_1)$.

$$\begin{cases} a(x) = a_0 + a_1 x + \cdots + a_n x^n + a_{n+1} x^{n+1} + \cdots + a_{2n} x^{2n} \\ b(x) = \underbrace{b_0 + b_1 x + \cdots + b_n x^n} + \underbrace{b_{n+1} x^{n+1} + \cdots + b_{2n} x^{2n}} \end{cases}$$

$$a(x) = \underbrace{a'(x)}_{\deg n} + x^{n+1} \underbrace{a''(x)}_{\deg n-1}.$$

$$b(x) = b'(x) + x^{n+1} (b''(x))$$

$$a(x) \cdot b(x) = [a'(x) \cdot b'(x)] + [S''(x) - S'(x) - S''(x)] x^2 + (a''(x) b''(x)) x^{2n+1}$$

$$\begin{cases} S'(x) = a'(x) b'(x) \\ S''(x) = a''(x) b''(x) \\ S''(x) = a'(x) b''(x) + a''(x) \cdot b'(x) \end{cases}$$

9/23

— $(p_0, p_1, \dots p_n) * (q_0, q_1, \dots q_n) = (r_0, r_1, \dots r_{2n})$

1) $r_0 = p_0 q_0$.

$1+2+\dots+n+(n-1+n-2+\dots 1) = \frac{n(n+1)}{2} = O(n^2)$

2) $r_1 = p_0 q_1 + p_0 q_1$

3) $r_2 = p_0 q_2 + r_1 q_1 + p_2 q_0$

$\vdots$

n, $n-1$, $n-2$ $\vdots$

$r_{2n} = p_n \cdot q_n$

$P(x) = p_0 + p_1 x + \dots + p_n x^n$

$q(x) = q_0 + q_1 x + \dots + q_n x^n$

$r(x) = P(x) \cdot q(x) = r_0 + r_1 x + \dots + r_{2n} x^{2n}$

— $(p_0 + p_1 x) \cdot (q_0 + q_1 x)$

$= \underbrace{p_0 q_0}_{S_0} + \underbrace{(p_0 q_1 + p_1 q_0)}_{S_1 - S_0 - S_2} x + \underbrace{p_1 q_1}_{S_2} x^2$ → 4 multi, 1 add/sub

$S_0 = p_0 q_0$

$S_1 = (p_0 + p_1) \cdot (q_0 + q_1)$ → 3 multi, 4 add/sub

$S_2 = p_1 q_1$

$n = 2k$ $P(x) = \underbrace{p_0 + p_1 x + \dots + p_k x^k}_{P'} + \underbrace{p_{k+1} x^{k+1} + \dots + p_{2k+1} x^{2k+1}}_{x^{k+1} P''}$

$P'(x) = p_0 + p_1 x + \dots + p_k x^k$

$P''(x) = p_{k+1} + p_{k+2} x + \dots + p_{2n-1} x^k$.

$\Rightarrow \begin{cases} P(x) = P'(x) + x^{k+1} P''(x) \\ q(x) = q'(x) + x^{k+1} q''(x). \end{cases}$

$P(x) \cdot q(x) = P'(x) \cdot q'(x) + [P'(x) q''(x) + P''(x) q'(x)] x^{k+1} + P''(x) \cdot q''(x) x^{2k+2}$ ← ①

$= S_0(x) + [S_1(x) - S_0(x) - S_2(x)] x^{k+1} + S_2(x) x^{2k+2}$ ← ②

$$\begin{cases} S_0(x) = P'(x) \cdot q'(x) \\ S_1(x) = [P'(x) + P''(x)][q'(x) + q''(x)] \\ S_2(x) = P''(x) \cdot q''(x) \end{cases}$$

$\Rightarrow f(n) = \#$ of multiplication

$f(n) = 4f(\frac{n}{2}) + \frac{n}{2} \rightsquigarrow O(n^2)$ $\Big\}$ $\rightarrow$ direct for ①

$\Rightarrow f(n) = 3f(\frac{n}{2}) + 4 \cdot \frac{n}{2}$ $\rightsquigarrow$ for ②

$= 3f(\frac{n}{2}) + 2n$ or $f(1) = C$

$n = 2^k$, $f(2^k) = 3 \cdot f(2^{k-1}) + 2^{k+1}$

Define $g(k) = f(2^k)$

$\Rightarrow g(k) = 3 \cdot g(k-1) + 2^{k+1}$ $\quad g(0) = C$

$= 3(g(k-2) + 2^k) = 3^2 \cdot g(k-2) + 3 \cdot 2^k + 3^0 \cdot 2^{k+1}$

$= 3^2(3g(k-3) + 2^{k-1}) + 3 \times 2^k = 3^3 g(k-3) + 3^2 \times 2^{k-1} + 3 \times 2^k + 3^0 \times 2^{k+1}$

$\vdots$

$= 3^k g(0) + 3^{k-1} \times 2^2 + 3^{k-2} \times 2^3 + \cdots + 3^0 \times 2^{k+1}$; $g(k) = S$

$S = \sum_{j=0}^{k} 3^j 2^{k+1-j}$

$\frac{2}{3} S = 3^{k-2} \times 2^2 + 3^{k-3} \times 2^3 + \cdots + 2^k + \frac{2^{k+1}}{3}$

$\Rightarrow \boxed{g(k) = 3^k \times 2 - 2^{k+1} + C_1}$

$3 = 2^{\log_2 3} \Rightarrow g(k) = 2^{k \log_2 3} \times 2 - 2^k \cdot 2$

$$\boxed{f(n) = 2n^{\log_2 3} - 2n}$$

- Vector: $a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$ $\quad a \in R^{n\times 1}$

inner Product: $b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$ $\quad \langle a, b\rangle = a_1 b_1 + \cdots + a_n b_n \rightsquigarrow O(n)$

matrix: $A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & & a_{nm} \end{pmatrix} \rightsquigarrow A \in R^{n\times m}$ $\quad \overset{m\times n}{A} \cdot \overset{m\times 1}{b} \in R^{n\times 1}$

$\rightsquigarrow O(n\cdot m)$.

$$A^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & & a_{n2} \\ \vdots & \vdots & & \vdots \\ a_{1m} & a_{2m} & & a_{nm} \end{pmatrix} \rightsquigarrow A^T \in R^{m\times n}$$

- $A \in R^{n\times m}$ $\quad B \in R^{m\times l}$

$A \cdot B = C \in R^{n\times l}$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \times B = \begin{pmatrix} b_{11} & \cdots & b_{1l} \\ \vdots & & \vdots \\ b_{m1} & & b_{ml} \end{pmatrix} = C = \begin{pmatrix} c_{11} & \cdots & c_{1l} \\ \vdots & c_{ij} & \vdots \\ c_{n1} & \cdots & c_{nl} \end{pmatrix}$$

$\rightsquigarrow c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{im} \cdot b_{mj} \rightsquigarrow O(n)$.

for entire matrix: $O(mnl)$.

— $A \in R^{n \times n}$: $B \in R^{n \times n}$ $\quad A \cdot B = R^{n \times n}$

matrix multi : $O(n^3)$

$N = n^2$: $O(n\sqrt{n})$.

matrix add : $O(n^2)$.

$N = n^2$: $O(N)$.

$$A = \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \begin{array}{c|c|c} A_{11} & A_{12} & \}5 \\ \hline A_{21} & A_{22} & \}3 \end{array} \\ 8 \times 9 \end{array} \times B = \begin{array}{c} \begin{array}{cc} 7 & 8 \end{array} \\ \begin{array}{c|c|c} 4 & B_{11} & B_{12} \\ \hline 5 & B_{21} & B_{22} \end{array} \\ 9 \times 15 \end{array} = C = \begin{array}{c} \begin{array}{cc} 7 & 8 \end{array} \\ \begin{array}{c|c|c} 5 & C_{11} & C_{12} \\ \hline 3 & C_{21} & C_{22} \end{array} \\ 8 \times 15 \end{array}$$

$$\begin{cases} C_{11} = A_{11}B_{11} + A_{12} \cdot B_{21} \\ C_{12} = A_{11}B_{12} + A_{12}B_{22} \\ C_{21} = A_{21}B_{11} + A_{22}B_{21} \\ C_{22} = A_{21}B_{12} + A_{22}B_{22} \end{cases}$$

$C \in R^{n \times n}$ $\quad C_{ij} \in R^{\frac{n}{2} \times \frac{n}{2}}$

to get the $n \times n$ matrix, do $8 \cdot \underbrace{\frac{n}{2} \times \frac{n}{2}}_{\text{multi}} + 4 \cdot \underbrace{\frac{n}{2} \times \frac{n}{2}}_{\text{add}}$

$\Rightarrow \boxed{f(n) = 8 \cdot f(\frac{n}{2}) + c \cdot n^2 \rightsquigarrow O(n^3).}$ $\rightsquigarrow$ ① $\leftarrow$ 8 multi 4 add.

$\leftarrow$ 7 multi 18 add (in textbook)

S_1 $\boxed{f(n) = 7f(\frac{n}{2}) + 18 \cdot (\frac{n}{2})^2}$ $\rightsquigarrow$ ②

S_2

$\vdots$

S_7

$\hookrightarrow O(n^{\log_2 7})$

$n^2 \log n$?

- Sorting - Quick Sort:

3	5	1	2	9	1	4	2

1	2	1	2	3	5	9	4

Partition: Pivot

选第一个数
也可以选最后一个或者其他

$\rightsquigarrow O(n)$

$n+1$

1 1 2 2 3 4 5 9

↑ ↑ ↑

□ 1 [ ] $\rightsquigarrow f(n) = f(n-1) + Cn \rightsquigarrow O(n^2) \rightarrow$ Worst case

[ ] Pivot [ ] $\rightsquigarrow f(n) = 2 \cdot f(\frac{n}{2}) + Cn \rightsquigarrow O(n\log n) \rightsquigarrow$ median case

$\frac{n}{2}$ $\frac{n}{2}$

- Average Case Complexity analysis (choose 1st to be pivot)
  - Assumption: All $n!$ order are equally likely $\Rightarrow$ every element in the list is equally likely to be the first element.
  - Analysis: expected (mean) of # operation

    $O(n\log n)$

$P[\text{If the pivot is the } k\text{'th element}] = \frac{1}{n}$

$k-1$	P	$n-k$

expect number of operation $C(X)$

$C(k+1)$ $\quad C(n-k) \quad + n+1$

$$C(n) = \sum_{k=1}^{n} \frac{1}{n}\left[C(k-1) + C(n-k) + n+1\right] \; ; \; C(1) = 1$$

$$= \frac{\sum C(k-1) + \sum(C(n-k) + (n+1))}{n} = n+1 + 2 \cdot \frac{C(1) + C(2) + \cdots + C(n-1)}{n}$$

$$n\,C(n) = n(n+1) + 2[C(1) + \cdots + C(n-1)]$$

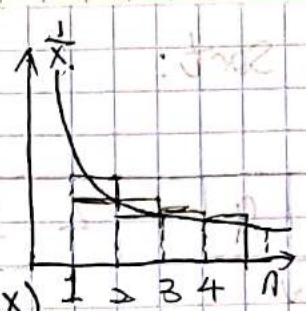
$$(n-1)\,C(n-1) = (n-1)n + 2[C(1) + \cdots C(n-2)]$$

$$nC(n) - (n-1)C(n-1) = 2n + 2C(n-1)$$

$nC(n) = (n+1)C(n-1) + 2n \rightsquigarrow \div\, n(n+1)$. See next page!

~~$\frac{C(n)}{n+1} = \frac{C(n-1)}{}$~~

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2}{n+1}$$
$$= \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$
$$= \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$
$$\vdots$$
$$\frac{C(2)}{1} + 2\underbrace{\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\right)}_{H_n} = O(\ln x).$$



$$\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n-1}\right) \geq \int_1^n \frac{1}{x}\,dx \geq \left(\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}\right)$$
$$\Rightarrow O(H_n) \geq \ln x \geq O(H_n)$$

$$\frac{C(n)}{n} = O(H_n).$$
$$= O(\ln x).$$
$$\Rightarrow C(n) = O(n \ln n).$$

- Randomized Algorithm:

Choose k equally likely to be $1, 2, \ldots, n$ and use $P = L[k]$ → Pivot

---

$A \cdot B = C \rightsquigarrow O(n^3)$. $O(n^{2+\alpha})$.

Is $A \cdot B \overset{?}{=} C$

Choose a random vector $x \in R^n$, $ABx \overset{?}{=} Cx$

$$\underbrace{ABx}_{n^2} \overset{?}{=} \underbrace{Cx}_{n^2}$$

$$\underbrace{A}_{n^2}\,\underbrace{y}$$

$O(kn^2)$. $\Pr[AB \neq C] = O(2^{-k})$.

↳ ??

- linear Programming

$\min \langle a, x \rangle$ s.t. $\langle b_i, x \rangle \geq c_i \quad i = 1, \dots, n$

- The Simplex-method:
  - worst case: exponential time
  - avg. case: simplex polynomial time

09/30

- The Fast discrete Fourier Transform

$P(X) = P_0 + P_1 X + \cdots + P_n X^n$

$[P_0, P_1, \ldots, P_n]$

- Evaluation : $a : P(a)$. $\Omega(n)$.
- Horner's method:

$$P(X) = P_0 + X(P_1 + P_2 X + P_3 X^3 + \cdots + P_n X^{n-1})$$
$$= P_0 + X(P_1 + X(P_2 + P_3 X + \cdots + P_n X^{n-2}))$$
$$\vdots$$
$$= P_0 + X(P_1 + X(\cdots + X(\underbrace{P_{n-1} + \underbrace{X P_n}_{r}}_{r}))\cdots))$$

$P(a)$ $\quad r = a * P[n]$.

$r = (r + P[n-1]) \cdot a$

Horner's Method

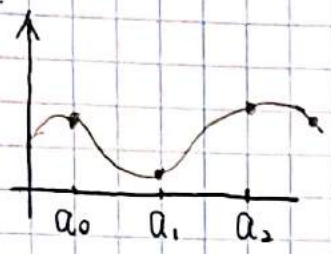
```
r = a * P[n]
For k = n-1 down to 1 do
    r = (r + P(k)) * a
output P(a) = r
```

$\rightsquigarrow O(n)$.

- interpolation: $O(n^2)$.

$$\begin{cases} P(a_0)=b_0 \\ P(a_1)=b_1 \\ \vdots \\ P(a_n)=b_n \end{cases} \Rightarrow \text{find } [P_0, P_1, \dots P_n]$$

$a_i \neq a_j$



$$\begin{cases} P(a_0)=P_0+P_1a_0+\cdots+P_na_0^n=b_0 \\ P(a_1)=P_0+P_1a_1+\cdots+P_na_1^n=b_1 \\ \vdots \\ P(a_n)=P_0+P_1a_n+\cdots+P_na_n^n=b_n \end{cases}$$

Vandermode Matrix

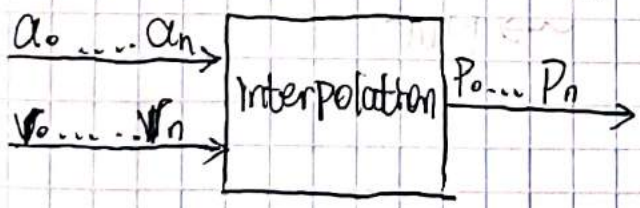
$$\begin{pmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^n \\ 1 & a_1 & a_1^2 & \cdots & a_1^n \\ \vdots & & \vdots & & \\ 1 & a_n & a_n & \cdots & a_n^n \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix}$$

$$A\underset{\sim}{P}=\underset{\sim}{b} \Rightarrow \underset{\sim}{P}=A^{-1}b$$

$\prod_{i\neq j}(a_i-a_j)$

matrix multiplication $\Leftrightarrow$ ~~immension~~ matrix inversion

$$\underset{n^{2.37}}{O(f(n))} \Leftrightarrow \underset{n^{2.37}}{\Theta(f(n))}$$

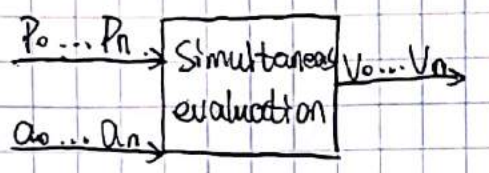


— Lagrange interpolation Formula:

$$P(X) = \frac{(X-a_1)(X-a_2)\cdots(X-a_n)}{(a_0-a_1)(a_0-a_2)\cdots(a_0-a_n)} \cdot b_0 + \frac{(X-a_0)(X-a_2)(X-a_3)\cdots(X-a_n)}{(a_1-a_0)(a_1-a_2)\cdots(a_1-a_n)} \cdot b_1 +$$

$$\frac{(X-a_0)(X-a_1)\cdots(X-a_{n-1})}{(a_n-a_0)(a_n-a_1)\cdots(a_n-a_{n-1})} b_n \rightsquigarrow O(n^2).$$

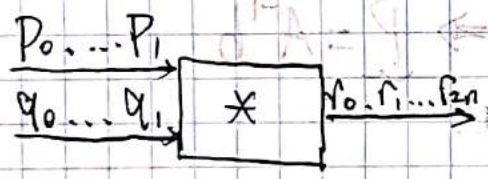
— Simultaneas evaluation: $\underbrace{P(a_0)\cdot P(a_1)\cdots P(a_n)}_{O(n^2).}$



— $P(X) = P_0 + P_1X + \cdots + P_nX^n$

$q(X) = q_0 + q_1X + \cdots + q_nX^n$

$r(X) = r_0 + r_1X + \cdots + r_{2n}X^{2n}$



— Algorithm:

(1). Choose $2n+1$ distinct number $\boxed{a_0 \ldots a_{2n}}$ $\rightsquigarrow O(n)$.

special numbers

(2). Simultanely compute $P(a_0)$ $\rightsquigarrow O(n^2)$

$\vdots$

$P(a_{2n})$.

(3). " " $q(a_0)$

$\vdots$

$q(a_{2n})$

(4). $r_0 = P(a_0)q(a_0)$ $\rightsquigarrow O(n)$

$r_1 = P(a_1)q(a_1)$.

$r_{2n} = P(a_{2n})q(a_{2n})$

(5). interpolate $r(x)$ from $r(a_i) = P(a_i)\,q(a_i)$. $\rightsquigarrow O(n^2)$

- special a_i's are the $2n$'th roots of unity (one) on the complex plane. $i = \sqrt{-1}$, $a + bi = \rho e^{i\theta}$

$n=2 \longrightarrow \sqrt{1} : 1, -1$

$n=4 \longrightarrow \sqrt[4]{1} : 1, i, -1, -i$

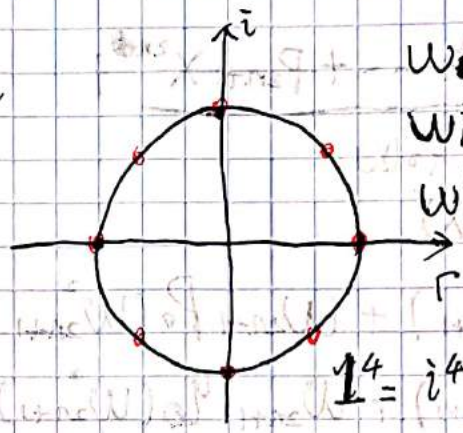
$n=8$

$P_0 \qquad P(1)$

$P_1 \xrightarrow{\text{DEF}} P(i)$

$P_2 \qquad P(-1)$

$P_3 \xleftarrow{\text{IDEF}} P(-i)$



$\omega_{1,n} = e^{i\frac{2\pi}{n}}$

$\omega_{1,n}^2 = e^{i\frac{2\cdot 2\pi}{n}}$

$\vdots$

$\omega_{1,n}^k = e^{i\frac{k\cdot 2\pi}{n}}$

$1^4 = i^4 = (-1)^4 = (-i)^4 = 1$

2). compute $\boxed{\begin{array}{l} V_k = P(\omega_{1,2n+1}^k), \quad k = 0, \dots, 2n \\ U_k = q(\omega_{1,2n+1}^k), \quad k = 0, \dots, 2n \end{array}}$ $\rightsquigarrow O(n\log n)$

discrete Fourier transform (DFT)

3). $W_k = V_k \cdot U_k$ for $k = 0, \dots, 2n$.

$W_k = r(\omega_{1,2n+1}^k)$ $\rightsquigarrow O(n)$

4). interpolate to get $r_0, \dots, r_{2n}$. IDFT

If $\alpha^n = 1$, $1 = (\alpha^n)^2 = \alpha^{2n}$ $\rightsquigarrow O(n\log n)$

- $P(X) = P_0 + P_1 X + P_2 X^2 + \cdots + P_n X^{n+1}$

$P_e(X^2) = P_0 + P_2 X^2 + P_2 X^4 + \cdots + P_{2n} X^{2n}$

$\rightarrow$ even

$P_o(X^2) = P_1 X + P_3 X^3 + \cdots + P_{2n+1} X^{2n+1}$

$\rightarrow$ odd

$X P_o(X^2) = X \underbrace{(P_1 + P_3 X^2 + \cdots + P_{2n+1} X^{2n})}_{P_o(X)}$

$\Rightarrow P(X) = P_e(X) + X P_o(X)$

$P(W_{2n+1}^2) = P_e(W_{2n+1}^2) + W_{2n+1} P_o(W_{2n+1}^2)$

$q(W_{2n+1}^2) = q_e(W_{2n+1}^2) + W_{2n+1} q_o(W_{2n+1}^2)$. with roots of 1

$\boxed{T(n) = 2T(\frac{n}{2}) + O(n)}$ $\boxed{T(n) = O(n \log n)}$

- EX). $\begin{cases} P(X) = P_0 + P_1 X \\ q(X) = q_0 + q_1 X \end{cases}$ want $r(X) = P(X) * q(X) = r_0 + r_1 X + r_2 X^2 + r_3 X^3$

$\begin{cases} W_4^0 = 1 \\ W_4^1 = i \\ W_4^2 = -1 \\ W_4^3 = -i \end{cases}$ $r_e = r_0 + r_2 X^2$ $r_e(t) = r_0 + r_2 t^2$

$X r_o = X(r_1 + r_3 X^2)$ $r_o(t) = r_1 + r_3 t^2$

$r_e(\overset{\rightarrow 1}{W_4^0}) = r_0 + r_1 \cdot 1^2$

$r_e(\overset{\rightarrow i}{W_4^1}) = r_0 + r_1 (i)^2$

$r_e(\overset{\rightarrow -1}{W_4^2}) = r_0 + r_1 (-1)^2$

$r_e(\overset{\rightarrow -i}{W_4^3}) = r_0 + r_1 (-i)^2$

$$V(1, i, i^2, i^3) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \xrightarrow{\text{inverse}}$$

interpolate:

$$\left[V(1, w_n, w_n^2, \dots, w_n^{n-1})\right]_{kl} = \left(e^{\frac{2\pi}{n} \cdot i}\right)^{kl}$$

$$\left[V(1, w_n, w_n^2, \dots, w_n^{n-1})\right]^{-1}_{kl} = \frac{1}{n}\left(e^{-\frac{2\pi}{n} i}\right)^{kl}$$

- Number theoretic algorithms/cryptography
  integers

u: BUY_APPLE_STOCK $\xrightarrow{E(u)=Au+B \bmod (u)}$ E

$\Downarrow$

encrypt $E(u,K)$.

XZYGPOZXCTY

$E^{-1}(u,K)$.

- O_A & O_B

$\underbrace{b}_{n\,bit} * \underbrace{a}_{n\,bit} \rightsquigarrow O_B(n^2) \rightsquigarrow$ bits

$O_A(1) \rightsquigarrow$ Algorithm

- K^n length$(K) = \lg K \rightsquigarrow$ 几位数? $\begin{cases} 0 \to 2 \\ 100 \to 3 \\ 1000 \to 4 \end{cases}$

• K & n are n bit numbers
  $\hookrightarrow n \simeq \log k$

$\boxed{K^m} \rightsquigarrow \underbrace{K \cdot K \cdot K \cdots K}_{m \text{ times}} \rightsquigarrow O(m \cdot \log^2 k)$.

$\|m\| = \log m$
$\|K\| = \log K$
$O(m \log k) = O(2^{\|m\|} \cdot \log k)$

- Better Case: $K^{16} \bmod 23$

$K^2 = K \cdot K \bmod 23$
$K^2 \cdot K^2 = K^4 \bmod 23 \rightsquigarrow \log m$
$K^4 \cdot K^4 = K^8 \bmod 23$
$K^8 \cdot K^8 = K^{16} \bmod 23$

K^{23} $23 = 16 + 4 + 2 + 1$

$K^{16} \cdot K^4 = K^{20}$ ⋮
$K^{20} \cdot K^2 = K^{22}$ ⋮
$K^{22} \cdot K = K^{23} \bmod 23$

$O(\log m \times \max(\log K, \log N)^2)$

$\hookrightarrow O(\log m \cdot \lg N^2) \rightarrow O(\|m\| \cdot \|N\|^2)$

- K^m

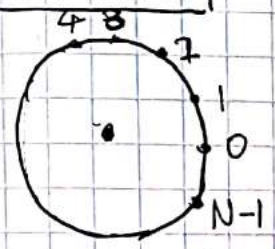
$\|K\| = n \rightarrow K$ is n bit number ; $n \approx \log_2 K$

$\|K \cdot K\| \approx n+1$

$\|K \cdot K \cdot K\| \approx n+2$

$\|\underbrace{K \cdot K \cdots K}_{m}\| \approx n+m$ bits

- $\boxed{K^m \bmod N}$



$(a \bmod N)$. a/N · remainder

$17 \bmod 3 = 2$.

$a \bmod b$ = remainder of division of a/b

- $\|2^{10} \bmod 15\| \leq \|15\|$

$\boxed{\|K^m \bmod N\| \leq \|N\| \approx \log N}$

- $ax = b \bmod N$.

10/07

- Master Method:

$$f(n) = a \cdot f(n/b) + g(n) \quad , \varepsilon > 0$$

1). If $g(n) = O(n^{\log_b a - \varepsilon})$, then $f(n) = O(n^{\log_b a})$

2). If $g(n) = \Theta(n^{\log_b a})$, then $f(n) = O(n^{\log_b a} \cdot \log n)$

3). If $g(n) = \Omega(n^{\log_a b + \varepsilon})$ and $af(n/b) < c \cdot g(n)$, hence $f(n) = O(g(n))$

①. binary search:

$$f(n) = f(n/2) + C$$

- $a=1$ $b=2$ $g(n)=C$, $\rightsquigarrow n^{\log_2 1} = n^0 = C$, It's 2 case: $O(\log n)$

②. $f(n) = 7f(n/2) + n^2$

- $a=7$ $b=2$ $\log_2 7 = 2.8\cdots$ $g(n) = n^2 = n^{\log_2 7 - 0.8\cdots}$

$f(n) = O(n^{\log_2 7}) \rightsquigarrow$ case 1

③. $f(n) = 4 \cdot f(\frac{n}{2}) + \underbrace{\frac{n^2}{\log n}}_{g(n)}$

$a=4$ $b=2$

$g(n) < n^{\log_2 4} (= n^2)$.

$n^{2-\varepsilon} < \frac{n^2}{\log n} < n^2$ for any $\varepsilon > 0$

Master method does not apply!

$O(n^{2.81} \log n)$

$= O(n^{\log 7} \log n)$

- For sorting with comparison:
  - require $\Omega(n\log n)$ comparison.

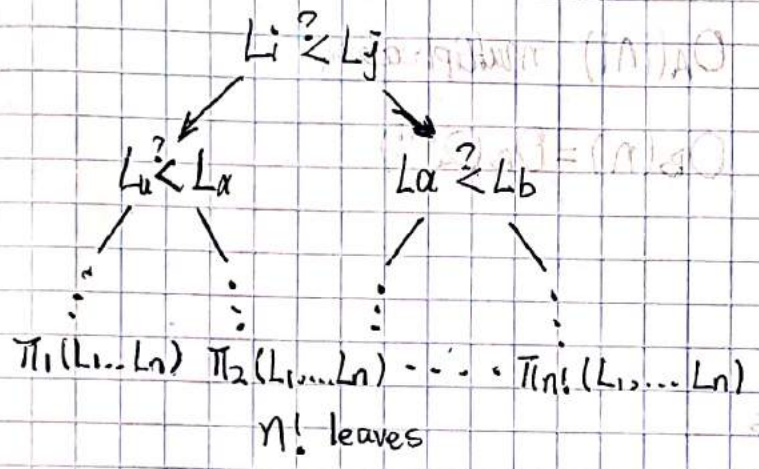
$[L_1, \dots, L_n] \Leftrightarrow [L[1] < L[2] < \dots < L[n]]$

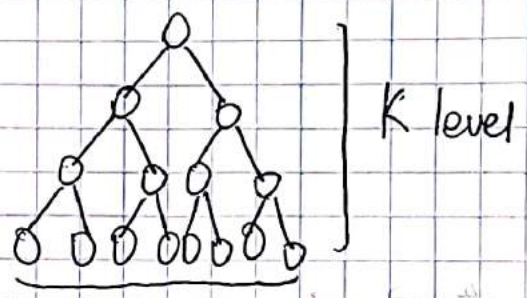
L_{in} $\qquad$ L_{out} 结果要排好大小

$L_{in} = \pi([L[1] < \dots < L[n]])$

# of permutation of n distinct items is $n!$

- Comparison Sorting algorithm:





$N = 2^k$ leaves $\Rightarrow$ depth $= \log_2 k$

- A Binary Tree with N leaves has <u>at least</u> depth $\log_2 N$
- depth $= \log n!$ comparison

- $n! < n^n$
  $\hookrightarrow \log n^n = n\log n$
- Stirling formula:

$$\left(\frac{n}{e}\right)^n \sqrt{2\pi n} < n! < e\left(\frac{n}{e}\right)^n \sqrt{n}$$

$$\Rightarrow n\log n - n + C\log n < \log n! < n\log n + 1 - \lg - \frac{1}{2}\log n$$

$$\log(n!) = \Theta(n\log n)$$

— Number Theory:

- $m^n = \underbrace{m \times m \times \cdots \times m}_{n \text{ times}}$ $\quad O_A(n) = O(2^{|n|})$

$\text{length}(n) = |n| = \lg n$.

- Repeated ~~Seq~~ Squaring:

$n = b_1 b_2 b_3 \cdots b_k \quad b_i \in \{0, 1\}. \quad k = |n| = O(\lg n)$

- $m = 23$

$$\begin{cases} m \\ m^2 \\ m^4 \\ m^8 \\ m^{16} \end{cases}$$

$O_A(|n|)$ multiplication

$O_B(n) = O_B(2^{|n|})$

$m^{16} \cdot m^4 \cdot m^2 \cdot m^1$

— $10^{1,000,000,000}$ = output has n bits

$= 2^{|n|}$ bits.

— $m^n \bmod k$ ⇝ trepend squares: $O_B(\log n \cdot \log k) = O_B(|n| \cdot |k| \cdot |m|^2)$

$$\begin{cases} a \pm b \bmod n \\ a \times b \bmod n \\ a^b \bmod n. \end{cases}$$

- $10^{1,000,000,000} \bmod 973$

- Modular Arithmetic:

$n=3 \to 3k+0: -3, 0, 3, 6$
$\to 3k+1: -5, -2, 1, 4$
$\to 3k+2:$

$\mathbb{Z}_n = \{[0], [1], \dots, [n-1]\}$ $\{[0], [1], [2]\}$

- Properties:

$+: [a]+[b]=[a+b]$

$*: [a]*[b]=[a*b]$

- $(\mathbb{Z}_n, +, *)$.

- $(\mathbb{Z}_n, +)$:

$a+0=a$ ← identity → $a\times(b\times c)=(a\times b)\times c$. $a\times 0=0$

$a+(-a)=0$ $(-a=n-a)$ $a\times b=b\times c$

$(a+b)+c=a+(b+c)$ ← Associate → $a\times 1=a$

$a+b=b+a$

- $(\mathbb{Z}_n, *)$:

- for each $a\neq 0$ does there exist:?
a^{-1}: $a\times(a^{-1})=1 \pmod n$

- mod 10
$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

$1^{-1}=1$

$2^{-1}=X$

$3^{-1}=7$

$4^{-1}=X$

$5^{-1}=X$

$6^{-1}=X$

$7^{-1}=3$

$8^{-1}=X$

$9^{-1}=9$

1, 3, 7, 9 inverse (mod 10) → no common divisor with 10

2, 4, 5, 6, 8 no inverse (mod 10) → has common divisor with 10

$2\times 5=10$ mod 10 $4\times 5=0$ mod 10

$a\times a^{-1}=1 \pmod n$.
$3\times 7=1 \pmod{10}$.

$a\times(a^{-1})=1 \pmod n$
↓
$3\times 7=1 \pmod{10}$

- Prime Number:

  P if the only divisor of P are 1 and P. 2|10

- two integer a, b ; k is <u>common divisor</u>

  $\Rightarrow k|a$ and $k|b$

- a and b are relatively prime if their only common divisor 1

  48 & 25 are relatively prime number $\Rightarrow \gcd(a,b)=1$.

- Greast Common divisor of a & b : $\gcd(a,b)$

  $\gcd(8,12)=4$

- If P is Prime number : $\begin{cases} \gcd(P,a)=1 \\ \gcd(P,kP)=P \end{cases}$

- $(Z_n,+)$; $(\overset{\text{take "0" out}}{Z_n \setminus \{0\}}, *)$

	$(Z_n,+)$	$(Z_n \setminus \{0\}, *)$	
Associative:	✓	✓	semigroup
identity:	0 ✓	1 ✓	semigroup
inverse:	✓	X	group
communicative:	$a+b=b+a$ ✓	$a \times b = b \times a$ ✓	Abelian group

- $(\overset{\text{P is Prime}}{Z_P} \setminus \{0\}, *)$; $Z_P = \{\overset{\text{take "0" out}}{\cancel{[0]}}, [1], \dots [p-1]\}$

  • $P=7$

  $\{\cancel{[0]}, [1], [2], [3], [4], [5], [6]\}$

  $\downarrow$ 1, 4, 5, 2, 3, [6]

  • If P is a Prime, then $Z_P' = Z_P \setminus \{0\}$

- $\{Z_n \setminus \{0\}, *\}$ not a group

  $Z_n' = \{k \mid \gcd(k,n)=1, 1 \le k \le n\}$.

  $\{Z_n', *\}$ is a group

- Prime factorial : $n = p_1^{n_1} \cdot p_2^{n_2} \cdots p_k^{n_k}$

$$48 = 2^4 \cdot 3^1$$

- To find the 1st prime factor in the worst case, require $O(\sqrt{n})$

$n \approx 2^{1{,}000{,}000}$

$\sqrt{n} \approx 2^{500{,}000}$

$\sqrt{n} = O(2^{|n| \cdot \frac{1}{2}}) = 2^{\frac{|n|}{2}} = 1.41^{|n|}$

- $g(a,b) = gcd(b,a)$, $a \geq b$

- recursive algorithm:

base: if $b=0$, $gcd(a,b)=a$

if $b=1$, $gcd(a,b)=1$

$a \div b$ = int : $a = bq + r$ $(0 \leq r < a) \Leftrightarrow gcd(a,b) = gcd(b,r)$

a mod b : $r = a \bmod b$.

- Claim: $gcd(a,b) =$ [redacted] $gcd(b, b \bmod a)$

$k|a$ $a = b \cdot k \Rightarrow b \cdot q + r = qhk + r$

$k|b$ $b = h \cdot k$

$r = dk - qhk$

$= (d - qh) \cdot k$

$\rightsquigarrow k|r$

- ex). $gcd(90, 21) \rightsquigarrow q = 4$, $r = 6$.

$= gcd(21, 6) \rightsquigarrow q = 3$, $r = 3$

$= gcd(6, 3) \rightsquigarrow q = 2$, $r = 0$

$= gcd(3, 0)$

$= 3$

```
Algorithm Euclid(a,b), a>b
if b==0 return(a)
if b==1 return(1).
return(gcd(b, a mod b))
```

$3 = gcd(3,0) = 3 \times 1 + 1 \times 0$

$= gcd(6,3) = 1 \times 6 + 3 \times (1 - 2 \times 1)$

$= gcd(21,7) = -1 \times 1 + 7 \times (1 - 3 \times (-1))$

$= gcd(90,21) = 90x + 21(-1 - 4 \times 4)$

- Bezout's Theorem:

If $\gcd(a,b)=g$, then there are integers x, y s.t.

$$\boxed{g = ax + by}$$

- $\gcd(a,0) = a = a\cdot x + 0\cdot y \Rightarrow x=1 \,\&\, y=1$
- $\gcd(a,1) = 1 = 1\cdot x + a\cdot y \Rightarrow x=1 \,\&\, y=0$

- $\gcd(a,b) = \gcd(b, \underbrace{b \bmod a}_{r}) \Rightarrow a = qb + r \,;\, 0 \le r \le a$

$\gcd(a,b) = \gcd(b,r) = g$

$$\Rightarrow g = bx + ry$$
$$= bx + (a-bq)\cdot y$$
$$= \underbrace{ay}_{x'} + \underbrace{b(x-qy)}_{y'} = ax' + by'$$

- $ax \equiv 1 \bmod n$

$\gcd(a,n) \begin{cases} \neq 1 \rightsquigarrow \text{no solution} \\ = 1 \rightsquigarrow \end{cases}$

$1 = ax + \underbrace{ny}_{\to 0} \bmod n$

- $m^n \bmod L$.

$E(m) = m^k \bmod n$

10/14

— $GCD(m, n) = g = mx + ny$

① $ax = 1 \mod n$

$gcd(a, n) = g = ax + ny \mod(n)$

② $M^k \mod n$

$\mathbb{Z}_n = \{0, 1, \dots, n-1\}$

$(\mathbb{Z}_n +)$

$(\mathbb{Z}_n \times) \rightarrow a^{-1}$ exist $\Leftrightarrow gcd(a, n) = 1$

$(\mathbb{Z}_n^* \times) \rightarrow a^{-1}$ always exists

$\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$

$\mathbb{Z}_n^* = \{x \mid x = 0, 1, \dots n-1$ and $gcd(x, n) = 1$

Crytography:

Private key

Public key

$M = (\underbrace{0}_{m_1}\underbrace{1}_{m_2}\underbrace{0}_{}\cdots\underbrace{1}_{m_n})$ → Message ⇝ Sequence of 0 & 1

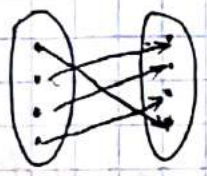
$\underbrace{100110}_{S}\ \underbrace{011100}_{E}\ \cdots\cdots$

$$M \xrightarrow{S;\,k} E \xrightarrow{S^{-1}} M$$

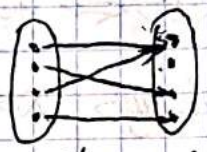
Message → function; k → Parameter

S^{-1} → $S^{-1}: P_B: E \to M.$

S is a one-to-one function



$S(x) = S(y) \Rightarrow x = y$



not one-to-one

(1). Bad example

$E = S_K(M) = M + K \bmod N$

$M = P_\ell: E - K \bmod N.$

$\ell = -K$

Alice sends message to Bob

$$A \xrightarrow{\begin{array}{l} E_1 = M_1^K \bmod N \\ E_2 \\ \vdots \\ E_n \end{array}} B$$

(2).

$S(M, K) = M^K \bmod N \Rightarrow P_\ell: E^\ell \bmod N$

need to find ℓ s.t. $(M^K)^\ell = M \bmod N$

$M^{K\ell} = M \bmod N$

(3).

• Bodean Algebra:

$x \cdot y = \begin{cases} 1 & \text{if both } x \,\&\, y \geq 1 \\ 0 & \text{otherwise} \end{cases}$

$x + y = \begin{cases} 1 & \text{if exists one of } x \text{ or } y = 1 \\ 0 & \end{cases}$

XOR

$x \oplus y \begin{cases} 1 & \text{if exactly one of } x \text{ or } y = 1. \\ 0 & \text{o.w.} \end{cases}$

x	y	$x \cdot y$	$x+y$	$x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

$$\begin{array}{ll} M & (10111010) \\ K \oplus & (11011011) \\ \hline E & 01100001 \\ \oplus & 11011011 \\ \hline K^{-1} = K & 10111010 \end{array}$$

- $\begin{cases} f(x) = \text{convex} \\ g(x) = \text{convex} \end{cases} \rightsquigarrow f(w_1 g(w_2)) = \text{non-convex}$

- Difficult Part:

1. # of parameters is large.
2. Gradient Descent $\quad W_{k+1} \leftarrow -\alpha_k \nabla F(W_k) + W_k$

   α_k ↳ step length / learning rate
3. Stochastic Gradient Descent:
4. early-stopping

- Chain Rule: $y = f(x)$, $z = g(y)$ $\rightsquigarrow \frac{dz}{dx} = \frac{dy}{dx} \frac{dz}{dy}$

$E = S_K(M) = M \oplus K$

$P_l(E) = M$

$P_l(E)$

- Crytographic Hash Function

Password : Bank check your hash code, and hold your hash code of password.

1. Fist Pass → Pre-imogine resistence : $E = S_K(M)$

   from E it should be computational hard to find u.

2. Given K it should be computational easy to compute $E = S_K(u)$

3. Random Property : Knowing E

   Should give you any useful information about M.

   S_K(Gerge) = XZ21@0.

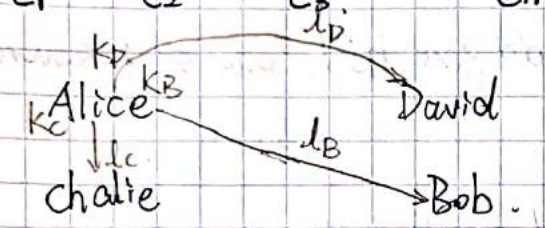
   S_K(gerge) = XF~~21~~@0. → bad similarity far away

4. If $E = S_K(M)$, it should be difficult to come up with another M' where $E = S_K(M')$.

5. Collision Resistence : If should be difficult to find M and M' s.t. $S_K(M) = S_K(M')$.

- Merkle - Damgonal Method

$$\begin{tikzcd}
M_1, & M_2, & M_3 \ldots & M_n \arrow[dl] \\
S_K(M_1) \arrow[d] & S_{e_1}(M_2) \arrow[d] & S_{e_2}(M_3) \arrow[d] & S_{e_{n-1}}(M_n) \arrow[r] & g(e_n) = E \\
e_1 & e_2 & e_3 &
\end{tikzcd}$$

- 

- Navie Private key:

- Three-way-Pass Method (Private key).

$$E = S_K(M).$$

$$M = P_l(E).$$

$$M = P_l(S_K(M))$$

$$M = P_K(P_{i'}(P_j, (S_j(S_i(S_K(M))))) \rightsquigarrow ①$$

- requirement: The S_i, $P_{i'}$ Pair for any key (i, i) should be <u>commutative.</u>

$$① = S_i P_{i'} S_j P_K P_i (S_i(M))$$

- Alice (K_A, l_A)

$E_1 = S_{KA}(M) \rightsquigarrow$ Bob receives

$E_2 = S_{KB}(E_1) \rightsquigarrow$ Alice receives

$E_3 = P_{lA}(E_2) \rightsquigarrow$ Bob receive

$M = P_{lB}(E_3)$.

Bob (K_B l_B)

$$\underbrace{P_{lB}(\underbrace{P_{lA}(\underbrace{S_{lB}(\underbrace{S_{KA}(M)}_{E_1})}_{E_2})}_{E})}$$

Multiple Class

y
B
R
G

$$\frac{e^{\langle b, x\rangle}}{1+e^{\langle b, x\rangle}}$$

$$P_K = \frac{e^{\langle b, x\rangle}}{\sum_{k=1}^{K} e^{\langle b, x\rangle}} \rightsquigarrow \text{softmax}$$

$M = \underbrace{P_{l_B}}_{2.}(S_{K_B}(\underbrace{P_{l_A}}_{2.}(S_{K_A}(M))))$.

Alice $K_A\ l_A$

David $K_D\ l_D$.

$P_{l_A}(S_{K_A}(M)) = M$

• requirement: knowing $l_A(P_l(\cdot))$ should not make it easy to find K, $S_K(\cdot)$

l = Public key

K = Private key

Charlie $K_C\ l_C$.

Bob $K_B\ l_B$

Alice $\xrightarrow{M\ E = P_{l_B}(M)}$ Bob.

$M = S_{K_B}(E)$.

• Singing: M M' ⇝ Alice creates two messages to Bob.

$\downarrow$

$E' = S_{K_A}(M')$

$$\begin{cases} E = P_{l_B}(M E') \\ M E' = S_{K_B}(E) \\ M' = P_{l_A}(E'). \end{cases}$$

$$\begin{cases} E = P_{l_B}(S_{K_A}(M)) \\ M = P_{l_A}(S_{K_B}(E)) \end{cases}$$

NP- Complete problem

easy way, but no efficiency

- RSA:

$S_K: M^K \bmod N$

$P_l: E^l \bmod N$ $\quad (M^K)^l \bmod N = M$

→ Euler's φ function

- $\varphi(n) = \#$ of m's where $1 \le m \le n$ and $\gcd(m, n) = 1$

$\varphi(10) = 4$

If P is prime number, $\varphi(P) = P - 1$

If $n = P_1^{e_1} P_2^{e_2} \cdots P_n^{e_n}$, then $\varphi(n) = n(1 - \frac{1}{P_1})(1 - \frac{1}{P_2}) \cdots (1 - \frac{1}{P_n})$.

Speical case $n = P \cdot q$, then $\varphi(n) = (P-1)(q-1)$

↳ both P & q are Prime numbers

- $O(\sqrt{n}) = O((2^{|n|})^{\frac{1}{2}}) = 1.41^{|n|}$
- Computing prime factorial of $n = P_1^{e_1} P_2^{e_2} \cdots P_K^{e_K}$ is not known to be easy;
- Computing $\varphi(n)$ is not known to be easy.

- Euler's Theorem:

For any integer n and m where $\gcd(m, n) = 1$, then $m^{\varphi(n)} = 1 \pmod n$

- special case: if P is prime number,

(Fermat's Theo). then $m^{P-1} = 1 \bmod P$

- RSA Public Key System

1. Choose very large random Prime numbers P & q, $n = P \cdot q$
2. Compute $\varphi(n) = (P-1)(q-1)$
3. Find a small number l where $\gcd(l, \varphi(n)) = 1$
4. Find K s.t. $K \cdot l = 1 \bmod (\varphi(n))$.

5. Publish n & l. Keep k secret.
$p, q, \varphi(n)$

~~* We know~~

$E = M^{l} \bmod n$

$M = E^{k} \bmod n$.

- Theorem: $M^{kl} = M \bmod n$: $M^{kl} = M^{1+h\varphi(n)}$
$= M \cdot M^{h\varphi(n)}$
$= M \cdot (M^{\varphi(n)})^{h}$
$= M \cdot 1 = M$

- $\gcd(l, \varphi(n)) = 1$

$k, x : lk + \varphi(n) \cdot x = 1$

$kl = 1 + h \cdot \varphi(n)$

10/21

— RSA:

1. select large (1000–2000 bits)

   Prime number P, q and compute $n = p \cdot q$

2. Compute $\phi(n) = (p-1)(q-1)$, and find a number l s.t. $\gcd(l, \phi(n)) = 1$

3. Use Euclid's algorithm to find k s.t. $k \cdot l = 1 \bmod \phi(n)$

4. Publish n, and l, (if someone wants to send you message M, then send $M^l \bmod n$).

5. Keep k and P, q secret.

   To decrypt: $E^k \bmod n$

   $M^{kl} = M \bmod n$

— Questions:

1. How do you test if P is prime number?

2. How do you defined Prime number?

• Answer:

1. Euclid's theorem: $a^{\varphi(n)} = 1 \bmod n$: $\gcd(a, n) = 1$

   Fermat's little theorem: $a^{P-1} = 1 \bmod P$

$P \rightarrow \boxed{P = \text{Prime?}}$ → yes / → No

① choose a random number $1 < a < P$

② check to see if $a^{P-1} = 1 \bmod P$

↳ $a^{P-1} \neq 1 \bmod P \Rightarrow P$ is not prime

if $a_1, a_2 \ldots a_k$, $a_i^{P-1} = 1 \bmod P$, then P [P is not prime] $\approx 2^{-k}$

$O(|P|^3)$ • carmichael # P · which is not prime, but $a^{P-1} = 1 \bmod P$ for all $1 \leq a < P$. $561 = 3 \times 11 \times 17$

ex. $\underbrace{2^{1023}+1 \cdots 2^{1024}-1}_{p,\,q \text{ Prime}}$, How Plentiful are primes?

- Prime Numbe Theory:

$\pi(n) = \#$ of Primes $\leq n$. then

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln(n)} = 1$$

$$\pi(n) \approx \frac{n}{\ln(n)}$$

- p is choosen randomly, $\leq n$, $P[p \text{ is Prime}] \approx \frac{n(\ln(n))}{n} \approx \frac{1}{\ln(n)}$

- Data Structure encode:

sets or multisets

$\{1, 3, 7, 2, 5\}$ $\{1, 1, 1, 2, 2, 3, 5, 5\}$

1. Specify operations.
2. Abstract data type (ADT). $\begin{cases} \text{Type of object (comparable or not)} \\ \text{What operations?} \end{cases}$

- ADT: A, B
  - $A \cap B$? • $A \cup B$
  - Search (membership)
    $x \overset{?}{\in} A$ and find it
  - Insert or delete:
    $A \cup \{x\}$ $A \setminus \{x\}$.
  - Find max or min: (comparable)
  - Find next or previous one (comparable)
  - Delete max or Delete min (comparable).

$O(n)$

$O(\log n) \rightarrow$ worst case

Array

— Array (sorted)

A:

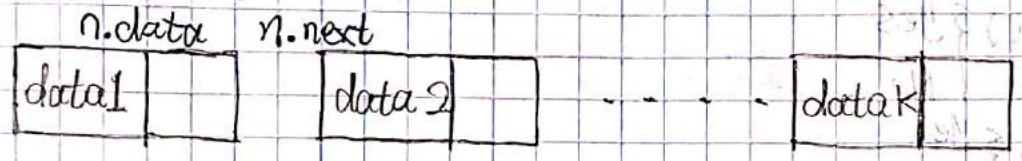
0	a
1	b
2	c
3	d
⋮	
n	w

A[i]

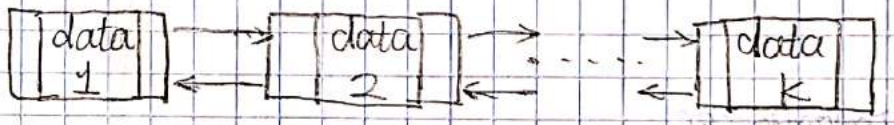
- Search sorted arrays: $O(\log n)$ ⇝ Binary Search
- insert / delete : $O(n)$

Linked list

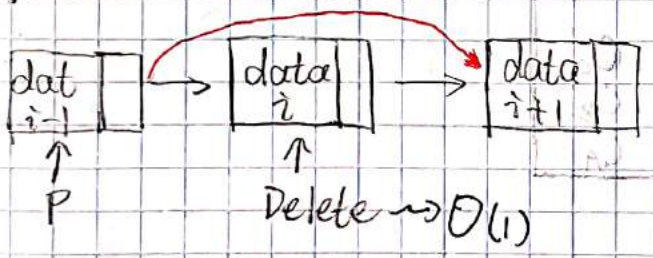
— Linked list:



- "w" linked list

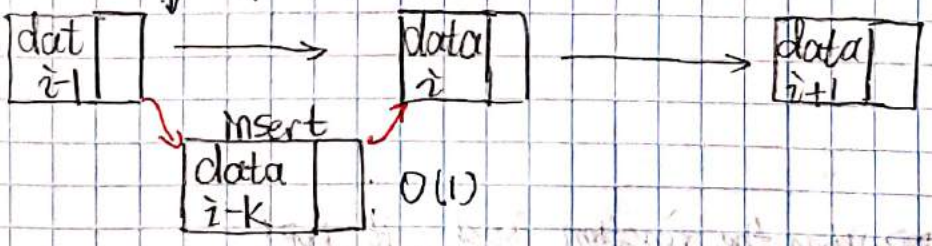


—



P.next = n.next

— free(n) 不清 .next



newn.next = P.next

P.next = newndata

Stack

— Stack:

① top(s) = top element
↳ stack

② push(s, a) : a is added to stack, new top is a.
↓ new data

③ pop() ⇝ top

- returns top element
- new top is element before top.

④ is_empty() { yes / No

⑤. is_full() { yes / No

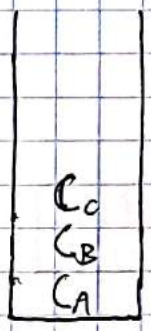
- last in-First out (LIFO)

— Application:

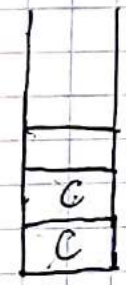
1. Well-formed parentheses

illegal {)((/ ((()))

{ for left Paren, Push (A (B (C
{ for right Paren, Pop



ex). (( )))(



2. function calls:

main()
⋮
f() → g()

→ POP: When the function finish, it pop!

— mergeSort(L)

L_1 = mergeSort $L[1, \frac{n}{2}]$

L_2 = mergeSort $L[\frac{n}{2}+1, n]$ ⇝ stack has $O(\log n)$

merge(L_1, L_2).

— QuickSort (L).

i = Split[L] ⇝ [小于 i] i [大于 i]

QuickSort([L[1] .... L[i-1])

QuickSort(L[i-1] ... L[n]).

— Algorithm Quicksort (L, low, high): L = [low high]

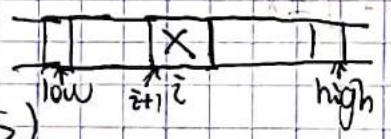
1. create empty stack S

2. Push(low, S) ⇝ Push low in the stack

3. Do a) if high > low:

i = Partition(L, low, high).

if i-low > high-i ⇝



Push(low, S), Push(i-1, S).

else: Push(i+1, S), Push(high, S)

else high = POP(S).

low = POP(S).

while(!empty(S)).

---

if high > low:

i = Partition(L, low, high).

if i-low > high-i:

Push(low, S), Push(i-1, S).

else: Push(i+1, S), Push(high, S)

else: high = POP(S)

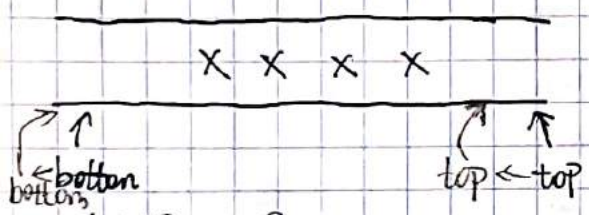
low = POP(S)

Queue

— Queue :

1. Create empty Q :
2. addQ(Q, x)
3. x = Remove Q(Q).
4. is_empty() { True / False
5. is_full() { True / False

• LILQ / FIFQ ⇝ last in - last Out



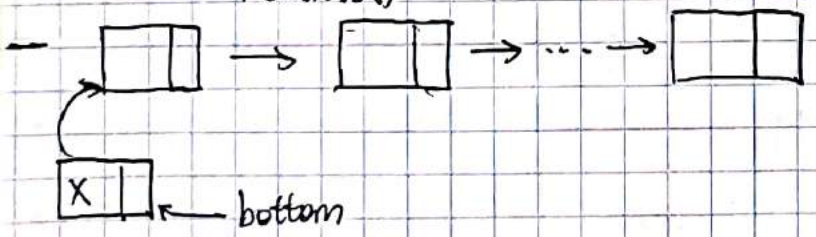
• AddQ(X, Q) :

Q[bottom - -] = X

• Delete(Q) :

X = Q[top]

return(X)



• AddQ(X, Q) :

newnode(X)

X.next = bottom

bottom = X

• Delete Q(Q) :

X = top.data

top = top.previous

free() → ?

return X

- Priority Queue:

1. items have comparable <u>keys</u>.

Operations:

2. Insert(Q, X[K])
3. Find max(Q) → most important thing
4. x = Exactmax(Q)
5. <u>IncreasePriority(X, K).</u> → ?

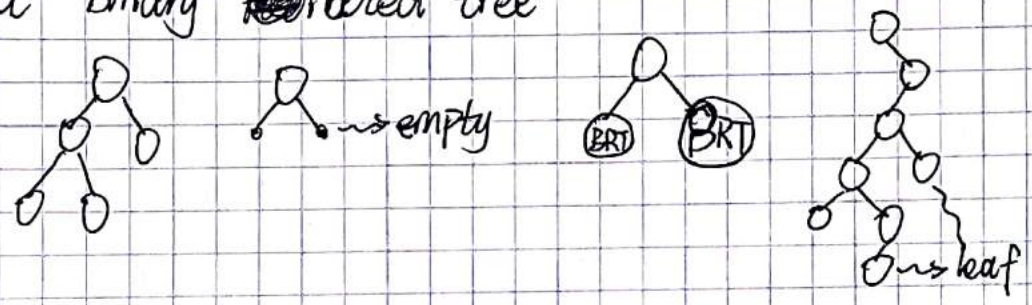
10/28

— Priority Queues: S

1. Insert(S, X) — $O(\log|S|)$
2. findMaxMax(S) ⇝ highest Priority element — $O(\cancel{\log}|S|) = O(1)$ for heap
3. m = ExtractMax(S) ⇝ remove the highest from queue — $O(\log|S|)$
4. IncreasePriority(X, K) ⇝ talk it later class — $O(\log|S|)$

— heap:

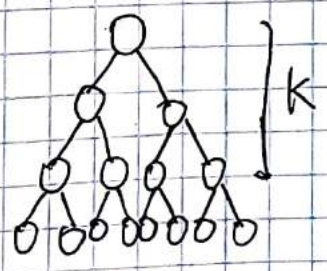
- a binary rooted tree



leave(n) = dist(n, root)

Depth(t) = max level (n) nodes

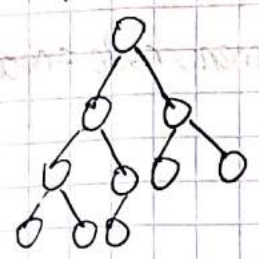
1. Full Trees:
   - All nodes have two children except leave
   - All leaves are at the same level



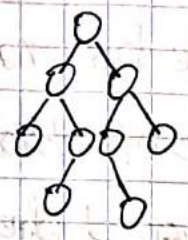
$N = \sum 2^K = 2^{K+1} - 1$

Depth = $K = \log_2(N+1)$

2. Complete Tree:



complete

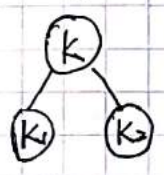


not complete

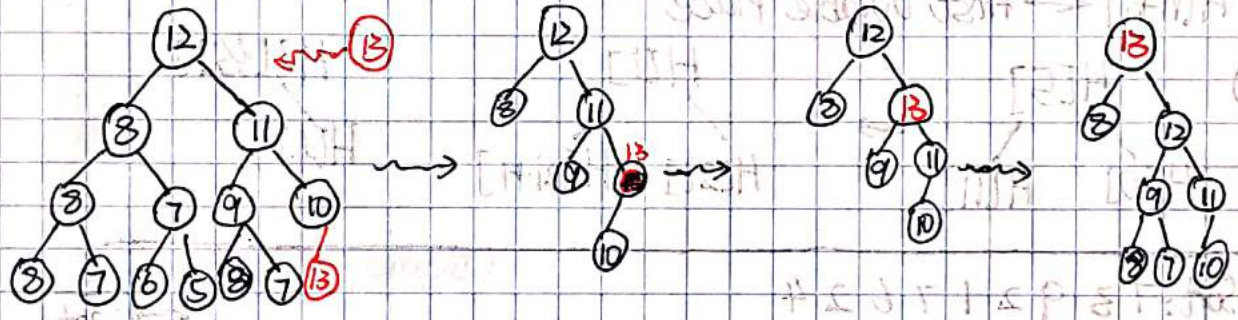
Depth = $O(\log N)$

- heap:
- A <u>heap</u> is complete binary tree where they key at node n. is larger than or equal to the key of its two children.

heap

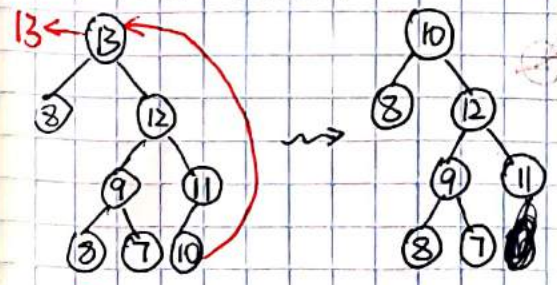


$\begin{cases} K \geq K_1 \\ K \geq K_2 \end{cases}$

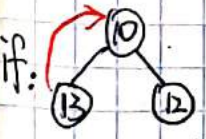


- We have access to the root
- We have access to the first a variable place
- If we have access to a node, then we have access to its children (if exists) and parent (if exists)

- m = Find Max(S)



if:



Compare with 13 & 12, then Swap

- Extract max(S):

1). tmp = key(root)

2). move last key to root and delete the last node ; current node = root

3). while [key(current node) < key in one of children]

{then swap(key, current node), key (largest children)}

current node = largest child

O(log N)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H:	12	10	11	8	7	9	~~10~~ 11	8	7	6	5	8	7	~~11~~ 10	

① H[1] → root

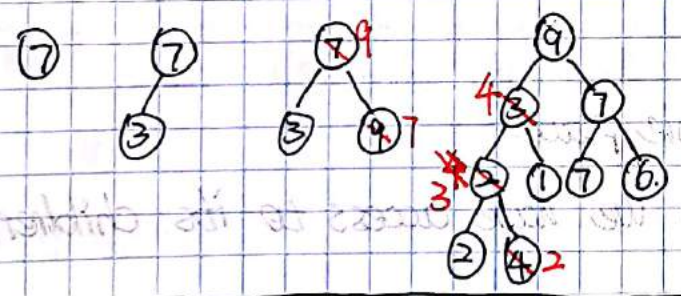
② H(n+1) ← First aviable Place

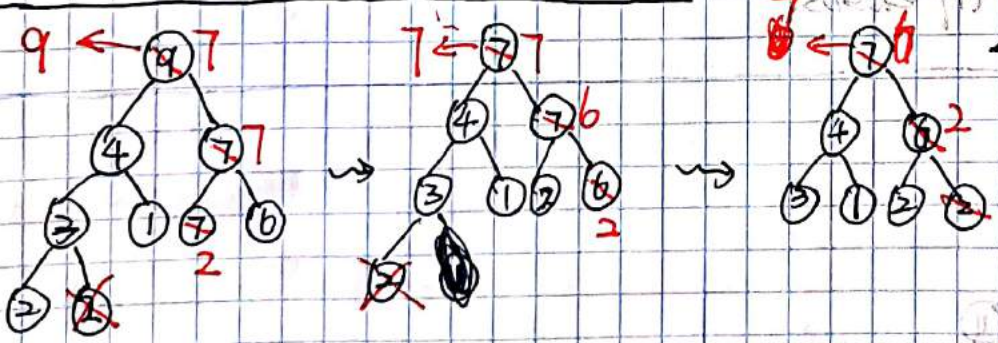
③ H[5] → H[10], H[11]

H[i] → H[2i], H[2i+1]

H[⌊i/2⌋] → H[i]

- Sort: 7 3 9 2 1 7 6 2 4 ← sort





← extract Max 9 7 7 6

— First Phase: build heap

$$\left(\sum_{k=1}^{N} \log k\right) + \sum_{k=N}^{1} \log k = \underbrace{2\sum_{k=1}^{N} \log k}_{\substack{\log(1\times 2\times 3\times\cdots\times N) \\ = \log(N!) \\ \approx O(N\log N)}} \leq 2\sum_{k=1}^{N} \log N \rightarrow N\log N$$

---

1. Search(S, X) ⇝ search everything
2. Insert(S, X)
3. Delete(S, X)

---

4. Findnext(S, X).
   Findprev(S, X).
5. Find(S, a, b)

S is a set of comparables

Two Approaches:

1. Hash Table (don't assume comparable)
2. Binary Search Tree (require comparable)

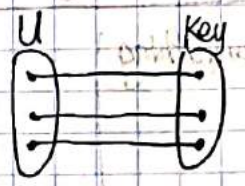
— Hash Table / Hash Function

- Direct addressing:

$U = \{\text{all objects}\}$ ex. words over $\{a, b, \dots, z, _\} \to 27$

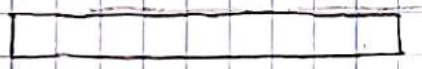
the set of all sequences of letters of length 10:

$|U| = 27^{10} = 2 \times 10^{14}$

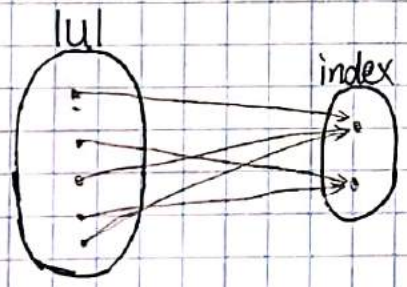


insert $O(1)$

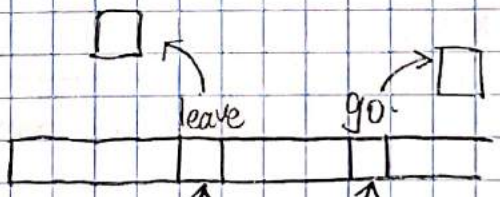
- If your universe large, like store a 100x100 "black-white" picture, your universe is 2^{10000}, it's really large!



$n \le |U|$



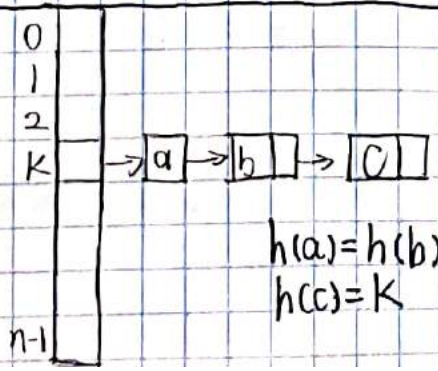
many-to-one
onto
subjective function



- go $\xrightarrow{h}$ 193275
- leave $\xrightarrow{h}$ 27932 collision
- Plate $\xrightarrow{h}$ 193275

— How to solve Collision?

1. Chaining:



$h(a) = h(b) = K$
$h(c) = K$

• load factor: m = # of items in the table

n = Size of length of array

$\alpha = \frac{m}{n}$ ⇝ load factor

ex). $\begin{cases} m = 100{,}000 \\ n = 300{,}000 \\ \alpha = 1/3 \end{cases}$ $\begin{cases} \text{average:} \\ O(1+\alpha) \rightsquigarrow \text{insert / search / delete} \end{cases}$

$\begin{cases} m = 100{,}000 \\ n = 50{,}000 \\ \alpha = 2 \end{cases}$

- $h(K) = K \bmod 2^{l}$ ⇝ Bad hash function

$973\underbrace{15163}_{l} \bmod 2^{5}$ ⇝ cause collision problem

A good hash function should use all of the bits!

- Fixed Hash Function

1. Division Method:

$\boxed{h(K) = K \bmod m}$ depend on how you choose m

① Bad choice: $m = 2^{l}$

② Good choice: m = Prime, but not close to a power 2.

2. Multiplication Method:

$\boxed{h(K) = \lfloor n(KA \bmod 1) \rfloor}$

choose $0 < A < 1$ $A = \frac{\sqrt{5}-1}{2}$

n = Size of array

ex). $\begin{cases} 3.25 \bmod 1 \\ 3.25/1 = 3 \end{cases}$

• If $K = 905232$ $n = 83$, $KA \bmod 1 = 0.1437\cdots\cdots$

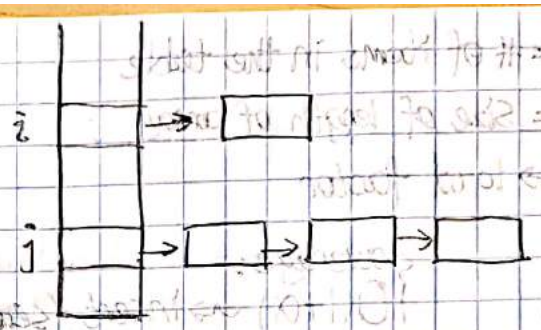
$83(KA \bmod 1) = 11$

11/04

hash

- $h(k) = \lfloor m(Ak \bmod 1) \rfloor \rightarrow i$

  ↳ is your email



$j = h(k_1) = h(k_2) = \cdots = h(k_m)$

1. Use many different hash function
2. For each excution of the program randomly choose a

$H = \{h_1, h_2 \ldots h_N\}$, $\{k_1, k_2 \ldots k_m\}$

We say H has universal property if k_1 & k_2 are ~~random~~ any two keys (distinct), then if h is chosen randomly from H, then $P[h(k_1) = h(k_2)] = \frac{|H|}{m}$

- One possible set of H with universal property:

1. Choose a base prime number P

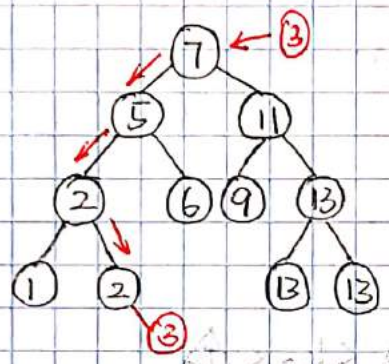
   $P \geq |U| \rightsquigarrow$ P is very large prime

2. m is the size of the hash table. $(m \ll P)$
3. For every $a \in \{1, \ldots P-1\}$, $b = \{0, 1, \ldots, P-1\}$, $h_{ab}(k) = (ak + b \bmod P) \bmod m$

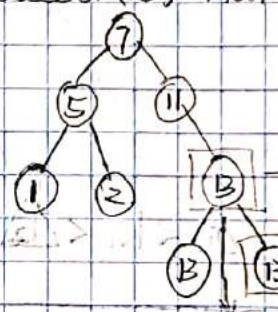
- Binary Search Tree: (BST)

1. Binary Tree with data composable of comparable objects.
2. For each node with K, every node in left sub tree has a smaller or equal keys
   : : : : , : : right : : larger : :
3. The left & right sub trees are also BST.
4. The empty tree is a BST and a single node in BST.

ex).



Predeccessor(13) = max{key: key < 13}.



→ 7 ⇝ Predeccessor.

⇝ No Predeccessor

successor(13) = 13

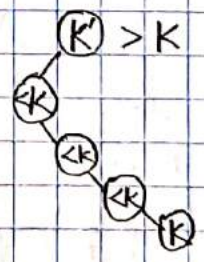
- depth = d

(1). Search = O(d)

(2). insert = O(d) (may increase depth by 1)

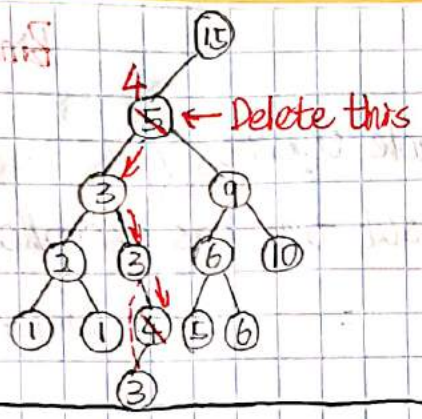
(3). Min / Max = O(d)
   ↳ keep going left / right

(4). successor / predeccessor:

- successor(K) = min{key: key ≥ K}
  - case①: right sub tree exists ⇝
  - case②: right sub tree doesn't exist ⇝ 

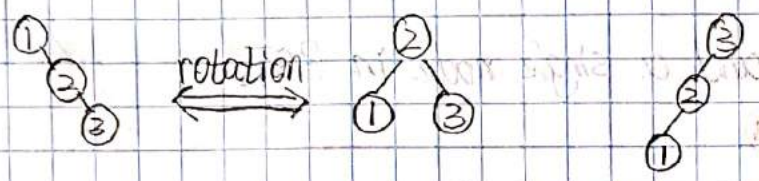
- Predeccessor(K) = max{key: key < K}
  - case①:
  - case②:
- If K = Pre(K'), then K' = succ(K)

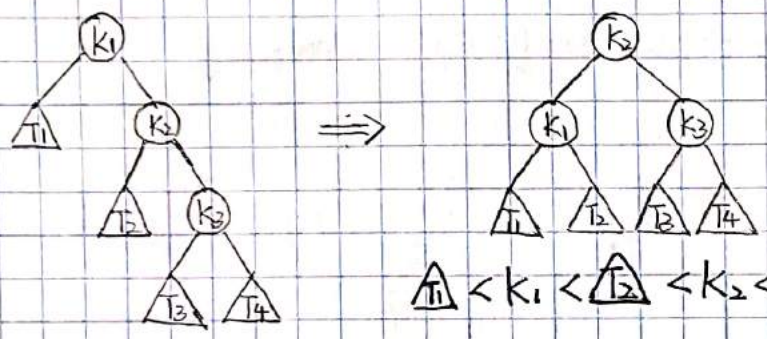
(5). delete:



• Delete = $O(d)$, may $d = d-1$

- rotation:

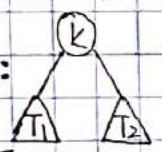




$T_1 < K_1 < T_2 < K_2 < T_3 < K_3 < T_4$

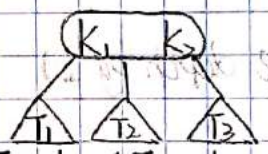
TWO - Three Trees

• 2-node:



$T_1 \le K \le T_2$

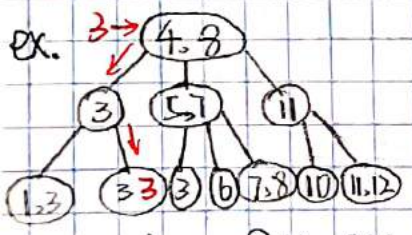
• 3-node:



$K_1 < K_2$

$T_1 \le K_1 \le T_2 \le K_2 \le T_3$

ex.



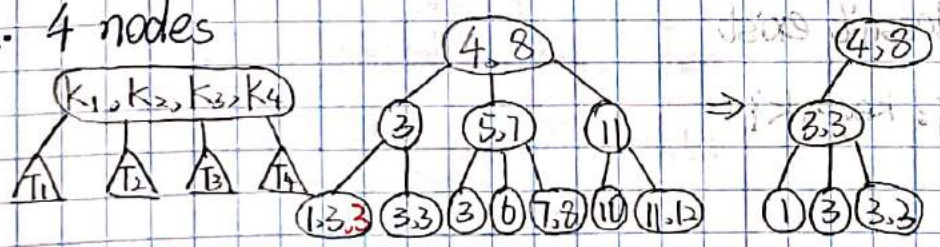
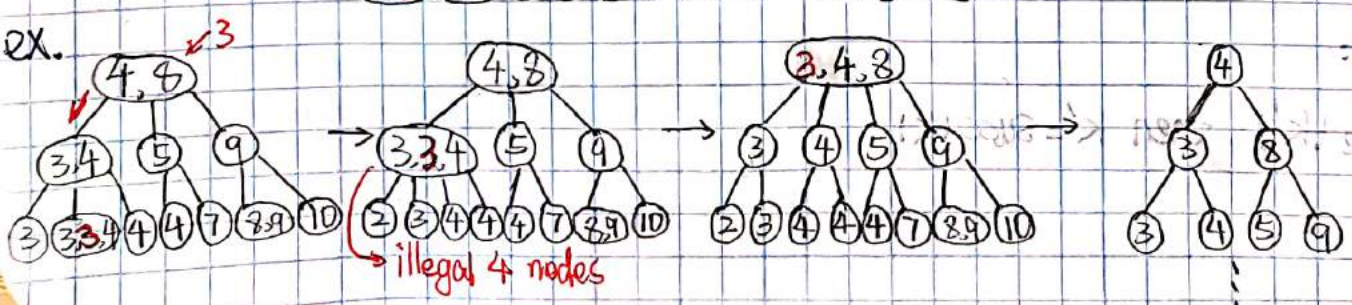
if N nodes, depth $\ge \log_2 N$

$\log_3 N \le$ depth $\le \log_2 N$

depth $= O(\lg N)$

- Search $= O(\lg N)$

ex. 4 nodes



ex.