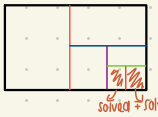
lect.1 (15/8/2023)

Greedy Algorithm : make a decision for the current moment / choose the best choice and don't care about the future

Divide and Conquer (divide → conquer → combine) : have a very big problem → ① divide the problem (ถ้าย่อยก็ได้จนคิดว่าเป็นปัญหาเล็กพอ) → ② solve the very small problem (conquer) → ③ combine solved problems together

: Breaks down a problem into multi sub-problems recursively until it cannot be divided further. These sub-problems are solved first and the solutions are merged together to form the final solution.

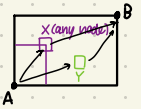


Dynamic Programming : "Current state is depended on previous states"

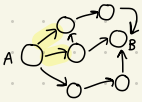
⇒ Solve smaller problems ⇒ use the solution from smaller problems to solve bigger problems.

ex : Shortest Path Problems

$A \to B = \min \begin{pmatrix} A \to X \to B \\ A \to Y \to B \end{pmatrix}$



Network Flow Problems : is to determine the best (optimal) way to route flows through a network to meet certain supply, demand, and capacity constraints



NP Completeness [P≠NP or not]
- Solvable Problems (P-class)
- Unsolvable Problems (EXP-class) : Tower of Hanoi

$= 2^n - 1 = O(2^n)$

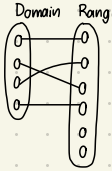
if $n = 100$, $2^{100} = (2^{10})^{10} = (10^3)^{10}$

Analysis --→ Amartized Analysis **Randomized Algorithms** **Approximate Algorithm**

Stable Matching

Matching Problem

↳ Given set D and R find matching M such that...



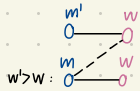
answer as a pair, set of pairs

$M \subseteq D \times R$ (subset of)

$\forall a \in D$, a must appear at most once in M

$\forall b \in R$, b must appear at most once in M

} Matching (≈ Mapping)



- pair(m,w) ∈ M (Answer matching)
- unstable pair

A pair(m,w) is unstable (iff) m prefer other woman(w') to w and that woman(w') also prefer m to her partner(m')

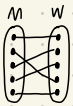
∴ m will try to hangout with w' instead

- Stable matching : A matching with no unstable pair.

Stable Marriage Matching Problem

M = set of n men
W = set of n women
} contain all other n people in order

Given M and W, and their preference lists, find a stable matching of size n.



(EX) Men · Women

a>b>c>d A — a A>C>B>D
a>c>b>d B — b B>D>A>C
b>d>c>a C — c C>D>B>A
b a>d>c D — d B>A>D>C

(EX) Stable Matching

a>b>c A — a A>B>C
a>b>c B — b A>B>C
a>b>c C — c A>B>C
∴ Unstable

(EX) more than 1 stable matchings

b>a>c A — a A>B>C
a>b>c B — b B>A>C
a>b>c C — c A>B>C
∴ Stable ✓

b>a>c A — a A>B>C
a>b>c B — b B>A>C
a>b>c C — c A>B>C
∴ Stable ✓

∴ Is M stable? → NO! : We can easily find unstable pair ; (D,b) (A,a), (B,a) are unstable pairs

Propose and Reject (Gale-Shapley's Algorithm) men propose from left→right and women only reject

index $i_0 : 1 \Rightarrow n$

While (there is a single male m)

- m proposes to the "best" girl $_q$w $_q$ from his list
- If w is single
  - w accepts m $(m_q w) \in M$
- Else (w is not single)
  - x = current w's partner
  - If w prefer x to m
    - w rejects m $(x_q w) \in M$
  - Else
    - w accepts m $_q$ rejects x $(m_q w) \in M_q M = M - (x_q w)$

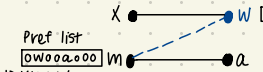
→ Is the answers (matching) stable?

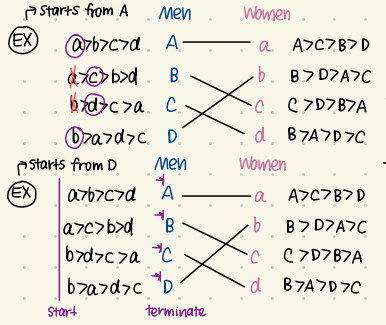
Proof [by contradiction]

Assume that $(m_q w)$ is an unstable pair ∉ M (answer from GS Algorithm)

- So m got rejected from w already
- So w prefers x to m
- $(m_q w)$ is not unstable pair

ตัวอย่างข้อ M = Stable





→ Algorithm gives the same answer in any order

→ Is the GS Algorithm terminated?

└ Yes

→ Time Complexity = $O(n^2)$

observation: After a girl gets a proposal, she will never be single again.

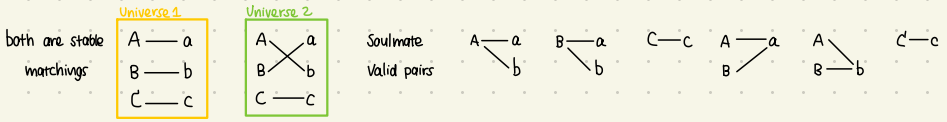
observation2: If all girl gets proposal, all boys will have partners

∴ at most n^2 proposal can be made

---

lect 2 (22/8/2023)

Soulmate or valid couple/pair : pair $(m_q w)$ is a valid pair iff $(m_q w)$ is in any stable matching



claim1 : In GS Algorithm $_q$ all men will get the best soulmate.

proof claim1 (by contradiction)

- Let M be the stable matching from GS Algorithm
- Suppose some men do not get the best woman
  Let $(m_q w)$ be the first rejected proposal from "valid" couple
- There must be another stable matching Θ that $(m_q w) \in \Theta$

[In progress]

M: Universe 1

w>a : x O - - ?? - - O a

m O - - X - - O w : x>m

∴ $(m_q w)$ is first rejection by soulmate

∴ x doesn't propose to a yet.

∴ x prefer w to a

(Contradiction!)

[Done Already]

Θ : Universe 2

w>a : x O ——— O a

m O ——— O w : x>m

valid pairs $(m_q w) \in \Theta$ $(x_q a) \in \Theta$

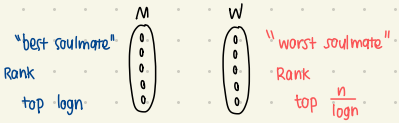
∴ $(x_q w)$ is unstable

∴ Θ is not a stable matching

Inconclusion proposed from soulmate will not be rejected.

claim2 : In GS Algorithm $_q$ all women will get worst soulmate.

- There are nlogn proposals (that all men may propose to women) in total.
- In average, men propose logn times



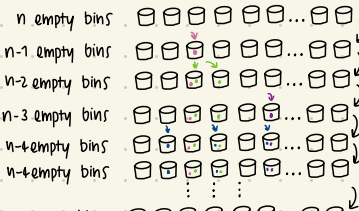
Balls and Bins

Situation: Throw n (same) balls into n (different) bins • Example Question: How many balls (proposals) needed to cover all bins (women)

Random Variables

X = #balls to make all n bins not empty

X_i = #balls to cover 1 more non-empty bins from i-1 non-empty bins to i non-empty bins



	Prob	$E[X_i]$
X_1	$1 = \frac{n}{n}$	1
X_2	$\frac{n-1}{n}$	$\frac{n}{n-1}$
X_3	$\frac{n-2}{n}$	$\frac{n}{n-2}$
X_4	$\frac{n-3}{n}$	$\frac{n}{n-3}$
X_5	$\frac{n-4}{n}$	$\frac{n}{n-4}$
X_{n-1}	$\frac{2}{n}$	$\frac{n}{2}$
X_n	$\frac{1}{n}$	$\frac{n}{1}$

$$X = X_1 + X_2 + X_3 + X_4 + X_5 + \dots + X_{n-1} + X_n$$

$$E[X] = E[X_1 + X_2 + X_3 + \dots + X_n]$$

$$= E[X_1] + \dots E[X_n]$$

$$= \frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} + \dots + \frac{n}{2} + \frac{n}{1}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}\right)$$

↗ harmonic sum

$E[X] \approx n \cdot \log n$

∴ We need about $n \log n$ balls to cover all bins

We need about $n \log n$ proposals to cover all women

Big O Notation

Compare Algorithms

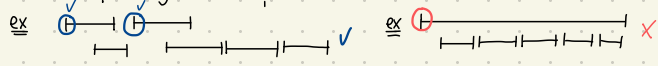
lect3 (29/8/2023)

Greedy Algorithms

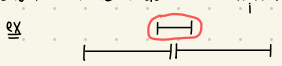
Idea: Choose the best choice at that moment
↳ strategy

Strategies (optimal) maximum subjects

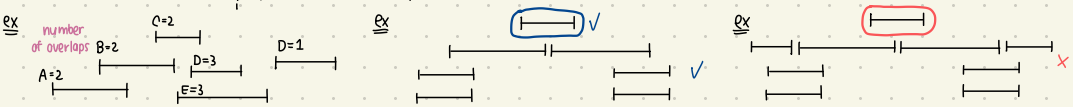
I. Smallest S_i (starting time) $\min_i S_i$



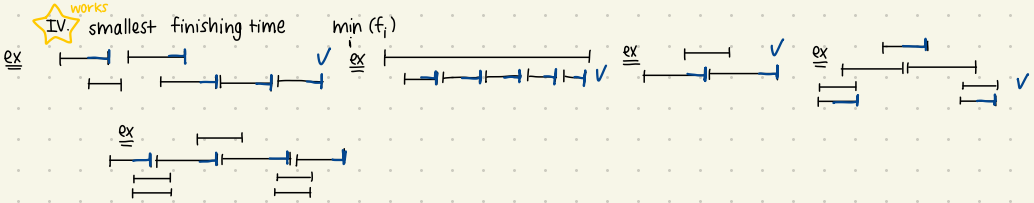
II. Shortest time to do $\min_i (f_i - S_i)$: pick the smallest one and can't be overlapped with other ones.



III. smallest "conflict" $\min_i$ (number of overlapped tasks to task i)



IV. works smallest finishing time $\min_i (f_i)$



Greedy Algorithm, G

- Sort all tasks by f_i (finishing time) from small to large } O(nlogn)
- Re-indexing all tasks (i.e., $f_1 \leq f_2 \leq f_3 \leq f_4 \leq \dots \leq f_n$)
- For i=1 to n → O(n)
  - pick task i if not overlapped to all selected tasks → O(1)·O(n) = O(n)

or, formally
if $S_i > f_a \quad \forall a \in A$
$A = A \cup \{i\}$

or, better
if $S_i > f$

∴ Time = O(nlogn) = O(n) if sorted already

math induction
p(1) is true p(k) is true
then p(k+1) is true

Show that G (result of our Greedy algorithm) is optimal

Proof: Let O be the result of any other algorithm.

$G = \langle g_1, g_2, g_3, \dots, g_p \rangle$ g_i is a task in G ; $O = \langle o_1, o_2, o_3, o_4, \dots, o_q \rangle$ o_i is a task in O

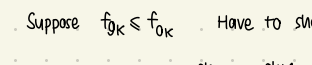
and $f_{g_1} < f_{g_2} < f_{g_3} < \dots < f_{g_p}$ $|G| = p$ and $f_{o_1} < f_{o_2} < f_{o_3} < \dots < f_{o_q}$ $|o| = q$

Goal: $f_{g_i} \leq f_{o_i} \quad \forall i$ (G always stays ahead)

Consider g_1 $f_{g_1} \leq f_x \quad \forall x \in U$ (universe)

$\therefore f_{g_1} \leq f_{o_1}$

Suppose $f_{gk} \leq f_{ok}$ Have to show: $f_{gk+1} \leq f_{ok+1}$



$f_{ok} \leq f_{ok+1}$ (because they are not overlapped)

$f_{gk} \leq f_{ok} \leq f_{ok+1}$ ... from induction hypothesis

$\therefore O_{k+1}$ is a table that G can choose as well

$f_{gk+1} \leq f_x \quad \forall x \in U'$

$\therefore f_{gk+1} \leq f_{ok+1}$

∴ From math induction, we can conclude that $\forall i \; f_{g_i} \leq f_{o_i}$

g always stays ahead of O

Interval Scheduling Problem

Given n intervals/tasks,

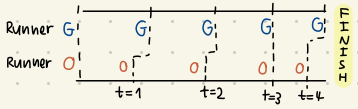
- task i started at time S_i, finished at time f_i
- 2 tasks is compatible if they are not overlapped

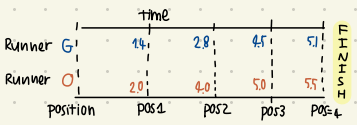
Goal: Find the maximum set of mutually compatible tasks
(longest set of tasks that none of them are overlapped)

G : |G| < |O| : No

Proof : How to prove that Greedy is the best algorithm? (optimal)

1. Greedy always stays ahead





2. Exchange Argument : try to exchange things / make a better result from left to right

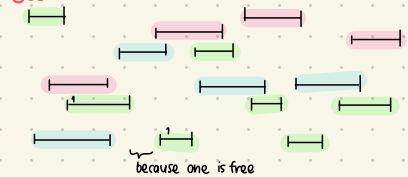
Class Scheduling Problem

Given n intervals / tasks,
- task i is started at time s_i
  is finished at time f_i

Goal : Find the minimum number of rooms to handle all classes.

no. of overlaps at any moment (depth) (d) $\leq$ no. of rooms $\leq$ no. of class $\leq$ n (ถ้าทุกคลาสชนกันหมด)

Q) Is it enough to use d rooms??

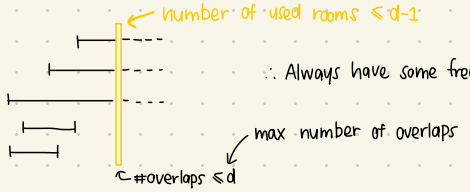


Algorithm G:
- Sort all classes by s_i
- Reindex $(s_1 \leq s_2 \leq \dots \leq s_n)$
- Assign the smallest-index available room {1, 2, 3, ..., d}

d = 1
2
3

Show that d rooms (rooms # overlapped classes) are enough

Proof Consider that at any moment when new class starts



Note :
Contradiction if number of used rooms > d.

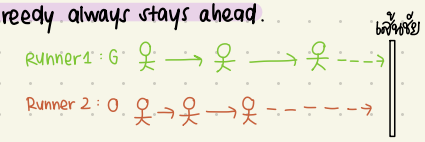
Q) You drive from bangkok to Chiangmai

lect 4 (5/9/2023)

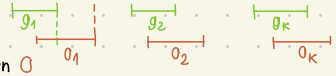
Greedy (Idea)
- Sort by measurement
- Pick the best choice on by one

To prove that Greedy is Optimal

I) Greedy always stays ahead.

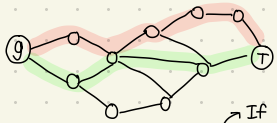


→ Interval Scheduling.
- whole answer from G = $\langle g_1, g_2, g_3, g_4, \dots, g \rangle$
- whole answer from O = $\langle o_1, o_2, o_3, o_4, \dots \rangle$



Math Induction : $f_{g_i} \leq f_{o_i} \quad \forall_i \quad \therefore$ G is better than O

Shortest Path (Dijkstra's Algorithm)

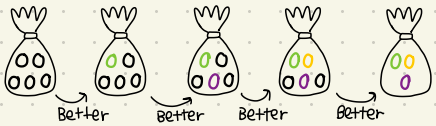


G always stays ahead

II) Exchange Argument
If you make some changes, it's better. (or equal) → make a property (prove the property) → greedy follows the property

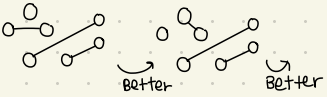
↳ Properties Idea: Having A is better than having B

ex

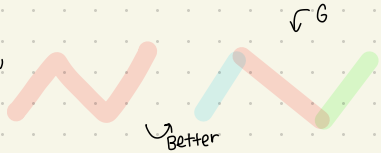


(เปลี่ยนลูกสีเรื่อยๆ หลุดลิ้นที่เปลี่ยนก็จะดีขึ้น)

ex ถ้าสลับคู่กันจะดีขึ้น



ex

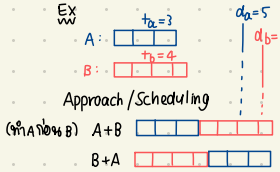


Interval Scheduling
- Property : Interval A and B ($f_A \leq f_B$) the smallest finishing time, the better.

Scheduling to Minimize Lateness
There are n tasks. For task i ; time to do the task (t_i), deadline of the task (d_i)

Goal : Find a scheduling that minimize the maximum lateness after doing all tasks.

Ex



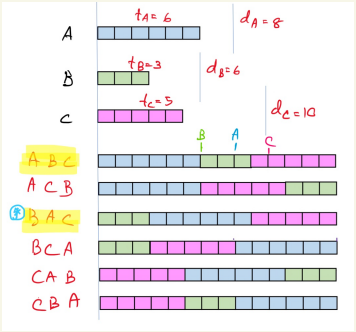
lateness

$late_i = \begin{cases} 0 & \text{if not late} \\ submit_i - d_i & \text{if late} \end{cases}$

max(late_A, late_B)

max(0, 7−6=1) = 1

max(7−5=2, 0) = 2



maximum lateness

max ↓

(0, 9−6=3, 14−10=4) = 4

(0, 14−6=8, 11−10=1) = 8

(1, 0, 4) = 4

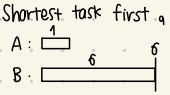
(6, 0, 0) = 6

(3, 8, 0) = 8

(6, 2, 0) = 6

Strategy of minimize maximum lateness

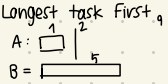
1. Shortest task first $\min_i t_i$ ("bad")



X A+B max late $(0, 7-6=1) = 1$

✓ B+A max late $(0, 0) = 0$

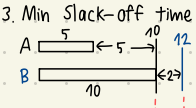
2. Longest task First $\max_i t_i$

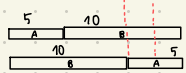


✓ A+B max late $(0, 0) = 0$

X B+A max late $(6-2=4, 0) = 4$

3. Min Slack-off time $\min_i (d_i - t_i)$





✓ A+B late $(0, 15-12=3) = 3$

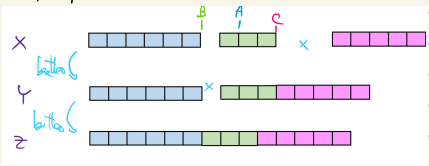
X B+A late $(5, 0) = 5$

4 Earliest deadline $\min_i d_i$

G
- Sort all tasks by deadline (small → large)
- Reindex $(d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n)$
- Do all tasks from 1 to n

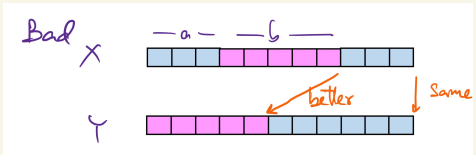
- To show G is the best, we have to provide some Properties.

Property 1 : No idle time is better



Submissions of all tasks are earlier in Z. (equal to) time(Z≤Y<X) better

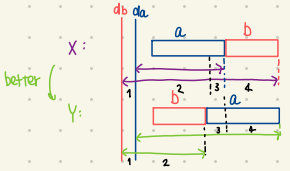
Property 2 : No interleave is better



Y is better than X (or equal to)

- All tasks' submission date is smaller (or equal)

Property 3 : Do task with earlier deadline first



follow the deadline

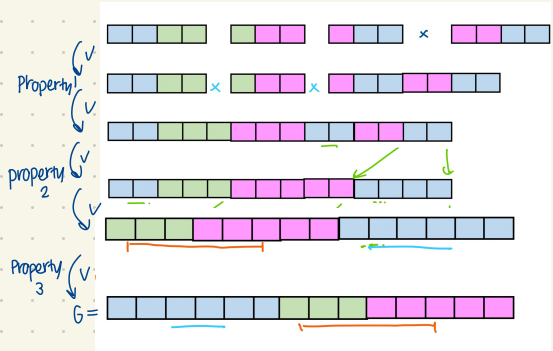
follow the Property 3

Goal Show that Y is better than X

maximum Y-lateness ≤ maximum X-lateness

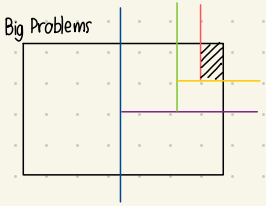
$$\max\begin{pmatrix}②+③+④\\①+②\end{pmatrix} \leq \max\begin{pmatrix}②+⑦\\①+②+③+④+⑤\end{pmatrix}$$

longest



lect.5 : divide & conquer

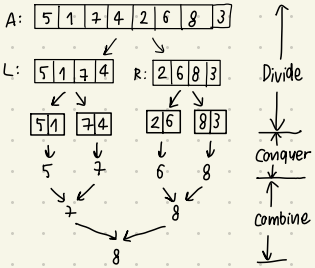
Divide & Conquer



#1) Divide : seperate problems to smaller part
#2) Conquer → Solve Small Problems (smallest one)
#3) combine → combine

Find Max

Given an array of n items, find the maximum.



T(n) (suppose)

Time	FindMax (A[1...n])	
O(1)	if (n==1) return A[1]	Conquer
O(1)	if (n==2)	
O(1)	return max(A[1], A[2])	
O(1)	L = A[1... $\frac{n}{2}$]	Divide
O(1)	R = A[$\frac{n}{2}$+1 ... n]	
$T(\frac{n}{2})$+O(1)	m_1 = FindMax(L)	
$T(\frac{n}{2})$+O(1)	m_2 = FindMax(R)	
O(1)	return max(m_1, m_2)	Combine

running time

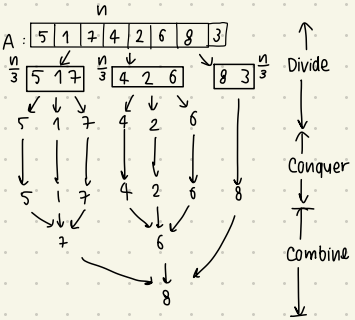
$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + O(1) + O(1) + \dots$

$T(n) = 2 \cdot T(\frac{n}{2}) + O(1)$ and $T(2) = 1$

$\therefore T(n) = O(n)$ $T(1) = 1$

FindMax (ver2)

Split into 3 pairs, instead of 2.



$T(n) = 3 \cdot T(\frac{n}{3}) + O(1)$ and $T(n) = O(1)$

$T(2) = O(1)$

$T(3) = O(1)$

$\therefore T(n) = O(n)$

Merge Sort

Def T(n)

	MSort(A[1...n])	
O(1)	if (n==1) return A[1]	Conquer
O(1)	if (n==2)	
O(1)	if A[1] < A[2]	
O(1)	return A[1...2]	
O(1)	else	
O(1)	swap(A,1,2)	
O(1)	return A[1...2]	
$T(\frac{n}{2})$	L = MSort(A[1 ... $\frac{n}{2}$])	Divide
$T(\frac{n}{2})$	R = MSort(A[$\frac{n}{2}$+1 ... n])	
$T_{combine}$	A = combin(L,R)	Combine
O(1)	return A	

$\therefore T(n) = 2T(\frac{n}{2}) + T_{combine} + O(1) \dots$ when $T(1) = T(2) = O(1)$

Calculation

when $T(1)=O(1)$, $T(1)=1$

① $T(n)=T(n-1)+O(1)$

Remove $O(1)$ → $T(n)=T(n-1)+1$: can put any number in n

$=T(n-2)+1+1$

$=T(n-3)+1+1+1$

$=T(n-4)+1+1+1+1$

$\vdots$

$=T(n-k)+\underbrace{1+1+1+\dots+1}_{k}$

$=T(n-k)+k$

$\vdots$

$=T(1)+n-1$

$\therefore T(n)=n$

Put $O(1)$ back → $T(n)=O(n)$

$T(k)=T(k-1)+1$
$T(2)=T(2-1)+1$
$T(n-2)=T(n-3)+1$

② $T(n)=2T(\frac{n}{2})+O(1)$

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

$=2(2T(\frac{n}{4})+1)+1$

$=2(2(2T(\frac{n}{8})+1)+1)+1$

$=2(2(2(2T(\frac{n}{16})+1)+1)+1)+1$

$\vdots$

$=2^k T(\frac{n}{2^k})+2^{k-1}+\dots+8+4+2+1$

when $2^k=n$ or $k=\log_2 n$ $=2^k T(\frac{n}{2^k})+(2^k-1)$

$=n\cdot T(1)+n-1$

$\therefore T(n)=2n-1$

$\therefore T(n)=O(2n-1)=O(n)$ → $O(1)$ back

---

③ $T(n)=2T(\frac{n}{2})+O(n)$

* $T(n)=2T(\frac{n}{2})+n$

$=2(2T(\frac{n}{4})+\frac{n}{2})+n$

$=2(2(2T(\frac{n}{8})+\frac{n}{4})+\frac{n}{2})+n$ ($\frac{n}{2}\times 2=n$)

$=2(2(2(2T(\frac{n}{16})+\frac{n}{8})+\frac{n}{4})+\frac{n}{2})+n$

$\vdots$

when $2^k=n$ or $k=\log_2 n$ $=2^k\cdot T(\frac{n}{2^k})+\underbrace{n+\dots+n+n+n+n}_{k\text{ times}}$

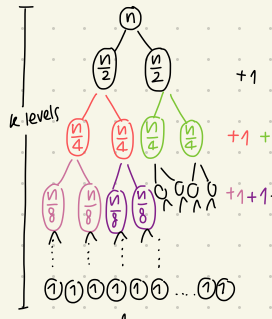
$=2^k\cdot T(\frac{n}{2^k})+kn$

$T(n)=n\cdot T(1)+n\log_2 n$

$T(n)=n+n\log_2 n$

* $T(n)=O(n\log n)$ [any base is fine]

④ $T(n)=2T(\frac{n}{2})+1$ and $T(1)=1$



$=1=2^0$ ($n=2^k$, $k=\log_2 n$)

$=2=2^1$

$=4=2^2$

$=8=2^3$

$\vdots$

2^{k-1}

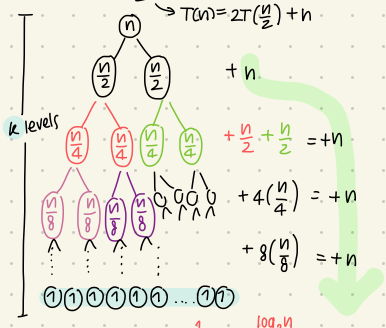
$T(n)=2^k\cdot T(1)+2^0+2^1+2^2+2^3+\dots+2^{k-1}$

$=\sum_{i=0}^{k}2^i=2^{k+1}-1$ ($2^k=n$)

$\therefore T(n)=2n-1$

$T(n)=O(2n-1)=O(n)$

⑤ $T(n)=2T(\frac{n}{2})+O(n)$ and $T(1)=1$ → $T(n)=2T(\frac{n}{2})+n$

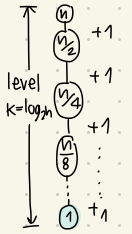


$\therefore T(n)=2^k T(1)+k\cdot n$ ($k=\log_2 n$)

$=n(\log_2 n+1)$

$T(n)=O(n\log n)$

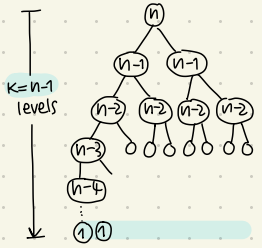
⑥ $T(n)=T(\frac{n}{2})+1$ and $T(1)=1$



$T(n)=1\cdot T(1)+k$

$T(n)=O(\log_2 n)$

⑦ Tower of hanoi: $T(n)=2T(n-1)+1$; $T(1)=1$



$\therefore T(n)=2^k T(1)+1+2+4+\dots+2^{k-1}$

$=\sum_{i=0}^{k}2^i=2^{k+1}-1$

$=2^n-1$

$\therefore T(n)=O(2^n-1)$

$=O(2^n)$

$T(n)=\square T(\square)+\square$

$T(n)=T(n-1)+1=O(n)$

$T(n)=T(n-1)+n=O(n^2)$

$T(n)=2T(n-1)+1=O(2^n)$

$T(n)=T(\frac{n}{2})+1=O(\log n)$

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

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

$T(n)=3T(\frac{n}{3})+1=O(n)$

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

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

$T(n)=3T(\frac{n}{2})+1=O(n^{\log_2 3})$

$T(n)=4T(\frac{n}{2})+1=O(n^2)$

Integer Multiplication

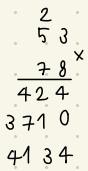
input: number of bits

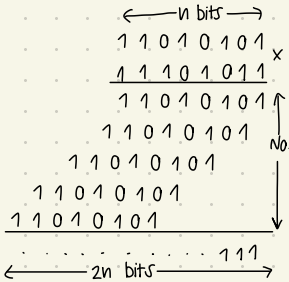
Calculate $z = x \cdot y$ (n-bit integers)

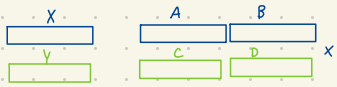
?? number of bits in $z \Rightarrow 2n$ or less.

?? Running Time $\Rightarrow O(n^2)$

↳ can we calculate faster ??





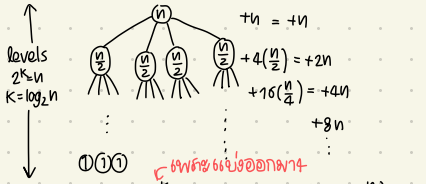


$z = X \times Y$

$= (A \cdot 2^{n/2} + B) \times (C \cdot 2^{n/2} + D)$

$z = AC \cdot 2^{n/2} + AD \cdot 2^{n/2} + BC \cdot 2^{n/2} + BD$

$T(n) = 4 \cdot T(\frac{n}{2}) + O(n)$; summation ; $T(n) = O(1)$



①①①

เพราะจะแบ่งออกมาๆ

$T(n) = 4^K T(1) + n(1+2+4+8+\ldots+\frac{n}{2})$

$2^K \cdot 2^K$

$\therefore T(n) = n^2 + n(n-1) = O(n^2)$

$T(n) = O(n^2)$

Lect 6 (19/9/2023) Merge sort, Counting Inversions, Closest Pair / Midterm : Tue 26th Sep Room 202-507 / Open Note A4 5 แผ่น 10 หน้า

Merge Sort

Input | 5 | 1 | 7 | 4 | 2 | 6 | 8 | 3 |

Output | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Sorting Algorithm

→ Bubble/Selection/Insertion = $O(n^2)$ [slow]

→ Merge / Quick / Heap = $O(n \log n)$

Divide & Conquer

Heap → ใส่เลขอะไรก็ได้ ถ้าเอาออกมา 1 เลขจะเป็นเลขที่น้อยที่สุด

Idea : 5 1 7 4 2 6 8 3

split data into 2 ชิ้น — divide

5 1 7 4 | 2 6 8 3

conquer ↓ ↓ ↓ ↓

1 4 5 7 | 2 3 6 8

combine

1 2 3 4 5 6 7 8

* Assume that is sorted (Idea of divide & conquer : We can assume anything (i.e. ผู้จำหมอนสมาธิ))

↱ สมมุติ Assume (1)(2)(3) ต้อง Ensure property (1)(2)(3)

* Return/Ensure everything is assumed correctly (ในกรณีนี้ก็คือ เลขถูก sort ทั้งหมดแล้ว / assume sorted array)

return Sorted Array

Running Time

- Merge Sort :

↱ สมมุติว่าใช้ Sort ยี่ห้ออะไรสักอย่าง — Selection/Insertion

combine

(I) $T(n) = 2T(\frac{n}{2}) + O(n^2) = O(n^2)$

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

(II) L : 1 4 5 7 (i) R : 2 3 6 8 (j)

A : 1 2 3 4 5 6 7 8

combine part

```
i=0, j=0
While (i ≤ n/2) && (j ≤ n/2)
    if L[i] < R[j]
        A.append(L[i]), i++
    else
        A.append(R[j]), j++
Then, put all the rest to A
```

combine in linear time

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

Merge : Combine

input : 2 sorted arrays L and R / output : sorted array A

input L : 1 4 5 7 R : 2 3 6 8

output A : 1 2 3 4 5 6 7 8

(I) $A = [L, R]$ A : 1 4 5 7 2 3 6 8

$A = Sort(A)$ A : 1 2 3 4 5 6 7 8

?? Selection/Insertion Sort ? = $O(n^2)$

Quick Sort

A : 10 5 8 4 2 12 7 9

Pivot = 7

L : 5 4 2 (น้อยกว่า 6.5) — a

R : 10 8 12 9 (มากกว่า 6.5) — n-a-1

Divide

Conquer (ไม่ต้องสนใจ)

2 4 5 | 7 | 8 9 10 12

Combine

2 4 5 7 8 9 10 12

→ Select Pivot

→ loop เข้า element ที่น้อยกว่า Pivot ใส่ซ้าย / ที่มากกว่าใส่ขวา } Divide

```
Qsort A[1...n]
if (n==1) return A[1]
r = random(1,n)        } select a random pivot
p = A[r]
for i=1 to n
    x = A[i]
    if x < p
        L.append(x)
    else
        R.append(x)
L = Qsort(A[1...n/2])
R = Qsort(A[n/2+1...n])
A = [L, p, R]   combine
return A
```

$T(n) = T(a) + T(n-a-1) + O(n) \approx O(n \log n)$

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

Counting Inversion : Given an array A, how many inversions in A

ex B : 1 2 5 4 3

ex C : 2 5 1 4 3

ex D : 2 1 5 3 4

D : 1 2 / 2 1 and 3 4 5 / 5 3 4

count inversion from the dots

Inversion is a pair $\langle a_i, a_j \rangle$ where $i<j$ but $a_i > a_j$

Traditional Algo = $O(n^2)$

```
for i=1 to n
  for j=i+1 to n
    if A[i] > A[j]
      inv++
return inv
```

5 1 7 4 2 6 8 3

5 1 7 4 | 2 6 8 3

Assume : inv_L , inv_R = free

Return : inv_A

$\therefore inv_A = inv_L + inv_R + inv_X$ (free, free; inv_X = inversion cross between L and R)

$T(n) = 2T(\frac{n}{2}) +$ Time to count inv_X

* How to count inv_X (across L and R)

5 1 7 4 2 6 8 3 , $inv_X = 8$

Ⅰ
```
invx = 0
for i=1 to n/2
  for j 1 to n/2
    if L[i] > R[j]
      invx = invx + 1
return invA = invL + invR + invx
```

Time = $O(\frac{n}{2} \times \frac{n}{2}) = O(\frac{n^2}{4}) = O(n^2)$

$T(n) = 2T(n) + O(\frac{n^2}{4}) = O(n^2)$

Ⅱ 5 1 7 4 2 6 8 3 , $inv_X = 8$

Return 1 4 5 7 | 2 3 6 8 ①

i → → → → j → → → → ②

$inv_A = inv_L + inv_R + inv_X$

Goal : $inv_X = 8$

① Sort
- L is Sorted
- R is Sorted

② inv_X
```
i=1, j=1
while (i≤n/2) && (j≤n/2)
  if L[i] > R[j]
    i++
  else
    invx = invx + (n/2 - i + 1)
    j++
```

$T(n) = 2T(\frac{n}{2}) +$ ① sort + ② inv_X

$\therefore T(n) = O(n \log^2 n)$

Ⅲ We can also do sort in linear time doing each step

5 1 7 4 2 6 8 3

5 1 7 4 | 2 6 8 3

Assume → sorted L → inv_L : 1 4 5 7 | 2 3 6 8

Return → sorted A → inv_A : 1 2 3 4 5 6 7 8

$inv_A = inv_L + inv_R + inv_X$

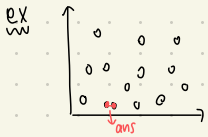
Assume L is sorted
R is sorted
i=1, j=1
while (i≤n/2) && (j≤n/2)
if L[i] < R[j]
A.append(L[i])
i++
else
$inv_X = inv_X + (\frac{n}{2} - i + 1)$
A.append(R[j])
j++
return A, $inv_A = inv_L + inv_R + inv_X$

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

$= O(n \log n)$

Closest Pair Problem : Given n points (x_i, y_i) coordinate find which pair is the closest [find the distance of the closest pair]

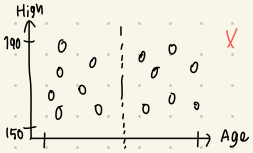
ex



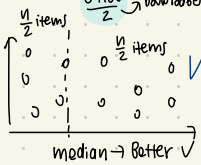
Traditional Approach

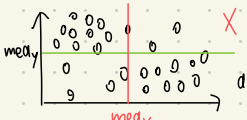
```
for i=1 to n
  for j= i+1 to n      = O(n²)
    max d_ij
```

Divide and Conquer Approach



$\frac{0+100}{2}$ ไม่ค่อยสะดวกเท่าไหร่



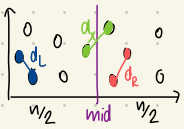


data maybe dense in just some part

$T(n) = 4T(\frac{n}{4}) + \square$

$= 4T(\frac{n}{2}) + \square$ ต้องการอันนี้ แต่บางส่วนอาจจะมาแทนไม่เท่าอีกส่วน

Find the closest pair

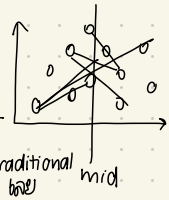


Ans $d_{ALL} = \min(d_L, d_R, d_X)$

How to find d_X ??

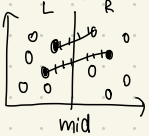
① All left to All right $\Rightarrow O(\frac{n}{2} \times \frac{n}{2}) = O(n^2)$

$T(n) = 2T(\frac{n}{2}) + O(n^2) = O(n^2)$ เท่ากับ traditional ไม่ดี

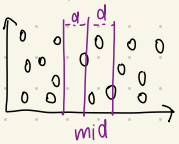


Some pairs are too far.

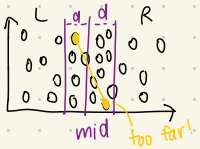
How to improve ?



$d = \min(d_L, d_R)$ ↳ distance ของคู่ที่สั้นสุดด้านซ้าย

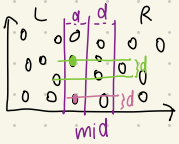


$T(n) = 2T(\frac{n}{2}) + \square$ ← How many pair to compute

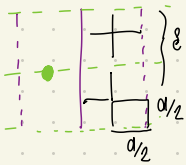


Possible that many items $\approx O(\frac{n}{2})$ ↳ are inside the band

$\therefore T(n) = 2T(\frac{n}{2}) + O(\frac{n^2}{4})$



How many pairs inside the box??



$\frac{\sqrt{2}}{2}d \approx 0.717d < 1d \leq d_R$

Impassible the 2 points are in the same small box

$\therefore$ 1 small box has at most 1 point

$T(n) = 2T(\frac{n}{2}) + O(\frac{n}{2} \cdot 8)$ 8 กล่องเล็ก

$= O(n \log n)$