1. 用 recursion，avoid stackoverflow？⇒ naive 會使呼叫層數過多 ↛ stackoverflow

遞迴式：$F(n) = F(n-1) + F(n-2)$

<法一> 記憶化，把算過的數值存在一個陣列裡，避免重複計算 ⇒ O(n)

```
ex: int memo[1000000] = {0};
    int fib(int n){
      if(n<=1) return n;
      if(memo[n] != 0) return memo[n];
      return memo[n] = fib(n-1)+fib(n-2);
    }
```

↗ 沒算過fib才算，否則算過就可直接查表

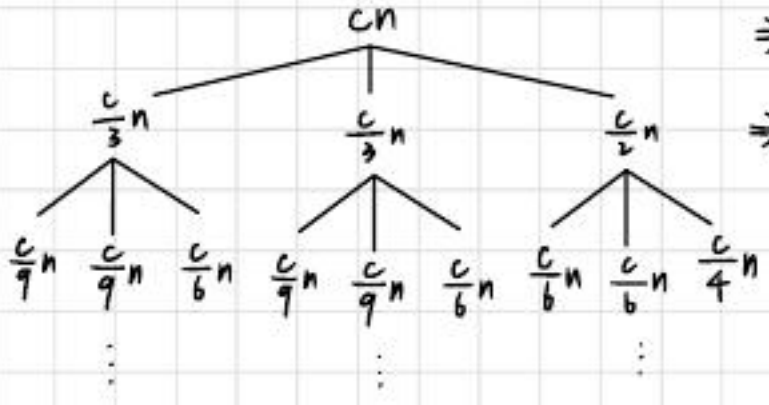
↗ tail recursion

<法二> 使用尾遞迴，讓回傳的東西又有函數自己（∵ compiler可優化）

```
ex: int fib_tail(int n, int a=0, int b=1){
      if(n==0) return a;
      if(n==1) return b;
      return fib_tail(n-1, b, a+b);
    }
```

2. (a) $T(n) = 2T(\frac{n}{3}) + T(\frac{n}{2}) + cn$

<法一> 遞迴樹（∵ 包含2種子問題，不能直接用 Master Theorem）



⇒ cn

⇒ $\frac{c}{3}n + \frac{c}{3}n + \frac{c}{2}n = \frac{7}{6}cn$

⇒ $2(\frac{c}{9}n + \frac{c}{9}n + \frac{c}{6}n) + (\frac{c}{6}n + \frac{c}{6}n + \frac{c}{4}n) = (\frac{7}{6})^2 cn$

樹高：$\log_2 n$

$$T(n) = cn + \frac{7}{6}cn + (\frac{7}{6})^2 cn + \dots + (\frac{7}{6})^{\log_2 n - 1} cn + O(n^{\log_2 3})$$

$$\leq \sum_{i=0}^{\infty} (\frac{7}{6})^i cn + O(n^{\log_2 3}) = \frac{1}{\frac{7}{6} - 1} cn + O(n^{\log_2 3}) \Rightarrow \underline{O(n^{\log_2 3})}\ *$$

<法=> Master Theorem

$T(n) = 2T(\frac{n}{3}) + T(\frac{n}{2}) + cn \leq 3T(\frac{n}{2}) + cn$ ($T(n)$) 令 $cn = f(n)$

$\log_2 3 \Rightarrow f(n) = O(n^{\log_2 3 - \varepsilon})$ for some constant $\varepsilon > 0 \Rightarrow T(n) = \underline{\Theta(n^{\log_2 3})}$ *

2. (b) $T(n) = 2T(\sqrt{n}) + \lg n$

令 $k = \lg n \Rightarrow n = 2^k \Rightarrow T(2^k) = 2T(2^{\frac{k}{2}}) + k$ 令 $S(k) = T(2^k) \Rightarrow S(k) = 2S(\frac{k}{2}) + k$

By Master Theorem, $\log_2 2 = 1 \Rightarrow O(k^1)$ 但 $f(n)$ 不是 $O(k) \Rightarrow S(k) = O(k \log k)$

代回 $\Rightarrow T(2^k) = O(k \log k) \Rightarrow T(n) = \underline{O(\lg n \log \lg n)}$

也就是 $T(n) = \underline{O(\log n \log \log n)}$ * (∵ log通常以2為底)

3. heap為 complete binary tree，每層都是滿的，只有最底層可能不滿，由左到右填充，$H = \lfloor \log_2 n \rfloor$ (n: node數)，根：$H=0$；最底層：$H=H$

Hth層 Max node 數量 $= 2^H$

已知 Full Binary Tree 在高度H時的 Max node 數量 $= 2^{H+1} - 1$

By Mathemetical Theorem,

① 當 $h=0$ 時，最多 $\frac{n}{2}$ 個 nodes 在 0th 層 $\Rightarrow \lceil \frac{n}{2^{h+1}} \rceil = \lceil \frac{n}{2^{0+1}} \rceil = \lceil \frac{n}{2} \rceil$ 成立

(∵一半是leaf，此時的 $h=0$ 為最底層)

除2是因為每個子節點最多只有2個，而第 h 層的 node 來自 (h+1)th

② 當 $h=k$ 時，假設在 kth 層時有最多 $\lceil \frac{n}{2^{h+1}} \rceil$ 個 nodes 成立

③ 當 $h=k+1$ 時，最多有第 k 層÷2個 nodes (∵ complete 且 binary)

即 $\frac{\lceil \frac{n}{2^{k+1}} \rceil}{2} = \lceil \frac{n}{2^{(k+1)+1}} \rceil$ 在 (k+1)th 層，與假設②相符

故 n-element heap has at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes at height h 來

4. ① Radix Sort 只能排序整數、固定長度字串

Comparison-based Sorting 可處理任意 data type，ex: float、double

② 比較排序在一般情況下實用穩定，常數時間開銷較低，而且通用性、易用性在各種環境下一致性高

③ Radix Sort 只適用鍵值大小固定且小的資料，需要額外空間

且當 key 太大，$O(d(n+k))$ 會大於 Comparison-based Sorting 的 $O(n\log n)$

5. $dp[i][j] = \min_{i<k<j} (dp[i][k] + dp[k][j] + (S[j]-S[i]))$

↓ $dp[i][j]$ = 長度 i 到 j 的最小成本

↗ 含 k 個切割點

i = 左界　j = 右界

$S[j]-S[i]$ = 長度 j 到 i 的成本

$S = \{0, 3, 6, 12, 17, 22, 28, 30\}$，Bottom-up DP：

	dp[i][j]	0	3	6	12	17	22	28	30
0	0	0	0	6	18	34	50	72	84
1	3		0	0	9	23	35	59	70
2	6			0	0	11	26	44	56
3	12				0	0	10	26	36
4	17					0	0	11	21
5	22						0	0	8
6	28							0	0
7	30								0

① 對角線皆為 0 ⇒ ∵ 自己 & 自己不可切割

② $dp[i][i+1] = 0$ ⇒ ∵ 兩相鄰點不需切割

③ 長度 2：

$S[0] = 0 \sim S[2] = 6$

(1) $dp[0][2] = dp[0][1] + dp[1][2] + (6-0)$
$= 0 + 0 + 6 = 6$

(2) $dp[1][3] = dp[1][2] + dp[2][3] + (12-3)$
$= 0 + 0 + 9 = 9$

(3) 同理，$dp[2][4] = 17-6 = 11$；$dp[3][5] = 22-12 = 10$

$dp[4][6] = 28-17 = 11$；$dp[5][7] = 30-22 = 8$

④ 長度 3：$dp[0][3] = \min_{0<k<3} (dp[0][k] + dp[k][3] + 12)$

$= dp[0][2] + dp[2][3] + 12 = 18$

⑤ 同理，處理長度為 4~7 的 dp，最終得到 dp[0][7] = 84 ※

dp[0][3] + dp[3][7] + 30

Back Trace:

dp[i][j]	0	3	6	12	17	22	28	30
0	0	0	6	18	34	50	72	84
3		0	0	9	23	35	59	70
6			0	0	11	26	44	56
12				0	0	10	26	36
17					0	0	11	21
22						0	0	8
28							0	0
30								0

令 bestcut[i][j] 為最佳切割點

bestcut	0	3	6	12	17	22	28	30
0	0	0	3	6	6	12	12	12
3		0	0	6	6	17	17	22
6			0	0	12	12	22	22
12				0	0	17	17	22
17					0	0	22	28
22						0	0	28
28							0	0
30								0

記錄最佳切割矣

從 best cut 表中查詢

∵ bestcut[0][7] = 12

① 先切 12 ⇒ (0, 12)

∵ bestcut[0][3] = 6 ↓ 切 6

(0, 6) ↓ 切 3

(0, 3) & (3, 6)

& (6, 12)

② (12, 30)

再切 22

(22, 30)

(12, 22) 切 17

(12, 17) & (17, 22)

切 28

(22, 28) & (28, 30)

A : 12, 6, 3, 22, 17, 28 ※

6.(a) 設 dp[i][j][k] 為 "前 i 個字符的序列 A，前 j 個字符的序列 B 前 k 個字符的序列 C，對應的 LCS 長度"

$$dp[i][j][k] = \begin{cases} dp[i-1][j-1][k-1] + 1 & , \text{if } A[i] = B[j] = C[k] \\ \max(dp[i-1,j,k], dp[i,j-1,k], dp[i,j,k-1]) & , \text{o.w.} \end{cases}$$

and $dp[i][0][k] = dp[0][j][k] = dp[i][j][0] = 0$

6.(b) 設 dp[i][j] 為 "以 i_{th} 字符結束的前綴與以 j_{th} 字符結束的前綴的最長相同 LRS 序列長度"

$$dp[i][j]=\begin{cases} dp[i-1][j-1]+1 & \text{if } S[i]=S[j] \text{ 且 } i\neq j \\ \max(dp[i-1][j], dp[i][j-1]) & \text{o.w.} \end{cases}$$

and 0 if $i=0 \vee j=0$

i \ j		Yj	A	T	T	A	A	T	A	T
0	Xi	0	0	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1	1	1
2	T	0	0	0	1	1	1	2	2	2
3	T	0	0	1	1	1	1	2	2	3
4	A	0	1	1	1	1	2	2	3	3
5	A	0	1	1	1	2	2	2	3	3
6	T	0	1	2	2	2	2	2	3	4
7	A	0	1	2	2	3	3	3	3	4
8	T	0	1	2	3	3	3	4	4	4

Bottom Up 依序填表，

例如 $dp[1][4]$ 來自 $dp[0][3]+1$，

因 A 和 A 對應；$dp[1][6]=\max(dp[1][5], dp[0][6])$ 以此類推

Back Trace 是利用 $dp[8][8]=4$，往左、上、左上角找，若該元素 = 左上角元素 +1 則輸出

A T A T

A: ATAT，長度 4 ※

b、(c) LPS 即反轉後的原字串 + 原字串 ⇒

$$dp[i][j]=\begin{cases} dp[i-1][j-1]+1 & \text{if } S[i]=S[j] \\ \max(dp[i-1][j], dp[i][j-1]) & \text{o.w.} \end{cases}$$

and 0 if $i=0 \vee j=0$

i \ j		Yj	c	h	a	r	a	c	t	e	r
0	Xi	0	0	0	0	0	0	0	0	0	0
1	r	0	0	0	0	1	1	1	1	1	1
2	e	0	0	0	0	1	1	1	1	2	2
3	t	0	0	0	0	1	1	1	2	2	2
4	c	0	1	1	1	1	1	2	2	2	2
5	a	0	1	1	2	2	2	2	2	2	2
6	r	0	1	1	2	3	3	3	3	3	3
7	a	0	1	1	2	3	4	4	4	4	4
8	h	0	1	2	2	3	4	4	4	4	4
9	c	0	1	2	2	3	4	5	5	5	5

Bottom Up 依序填表，

例如 $dp[1][4]$ 來自 $dp[0][3]+1$，

因 r 和 r 對應；$dp[1][6]=\max(dp[1][5], dp[0][6])$ 以此類推

Back Trace 是利用 $dp[9][9]=5$，往左、上、左上角找，若該元素 = 左上角元素 +1 則輸出

c a r a c

A: carac，長度 5 ※

7. 令 v_i 為 ith 物品的 value，w_i 為 ith 物品的 weight，W 為 max capacity，B[k,w] 表示前 k 個物品在容量 w 下的 Max value

令 $B[k,w] = \begin{cases} B[k-1,w] & \text{if } w < w_k \quad \text{以 weight 為主方法} \\ \max(B[k-1,w], B[k-1,w-w_k]+b_k) & \text{o.w.} \end{cases}$

b_k = 第 k 個物品的 value

初使目等若 k=0、w=0 ⇒ 0；B[0,v] 都為 0

item	w_i	v_i
1	100	1
2	150	2
3	200	3
4	300	4

	item \ w	0	100	150	200	250	300	350
	0	0	0	0	0	0	0	0
X	1	0	1	1	1	1	1	1
V	2	0	1	2	2	3	3	3
V	3	0	1	2	3	3	4	5
X	4	0	1	2	3	3	4	5

2-2=0

5-3=2

因此題 weight 較 value 大，以 value 為主

A: value=5 *

$$dp[v,i] = \begin{cases} 0 & \text{if } v=0 \\ \infty & \text{if } i=0, v>0 \\ \min(dp[v,i-1], dp[v-v_i, i-1]+w_i) & \text{o.w.} \end{cases}$$

w_i → ith 物品 weight

v_i → ith 物品 value

v \ i	0	1	2	3	4
0	0	0	0	0	0
1	∞	100	100	100	100
2	∞	∞	150	150	150
3	∞	∞	250	200	200
4	∞	∞	∞	300	300
5	∞	∞	∞	350	350
6	∞	∞	∞	450	450
7	∞	∞	∞	∞	500
8	∞	∞	∞	∞	600
⋮	⋮	⋮	⋮	⋮	⋮

item	w_i	v_i
1	100	1
2	150	2
3	200	3
4	300	4

因 W 背包容量大有 350，由 dp[5][4] 往回找

350 − 200 = 150　150 − 150 = 0

⇒ item 3 + item 2

⇒ value = 2+3 = 5 *

8. step1：先照結束時間排序 (ascending)

	activity	start	end	value
j=1	5	1	3	1
2	4	1	4	3
3	1	2	4	3
4	3	3	4	2
5	6	3	5	4
6	2	5	5	5

令 dp[j] 為選擇 activity j 的 Max value，p(j) 表活動 j 前結束的最後一個 index ⇒ $dp[j] = Max(dp[j-1], v_j + dp[p(j)])$

v_j：第 j 個活動的 value

使用 binary search (and dp[0] = 0)

activity	p(j)
5	0
4	0
1	0
3	1
6	1
2	3

(不能選 5 會重疊)

	activity	start	end	value	p(j)	狀態計算	dp[j]
j=1	5	1	3	1	0	max(0, 1+0)	1
2	4	1	4	3	0	max(1, 3+0)	3
3	1	2	4	3	0	max(3, 3+0)	3
4	3	3	4	2	1	max(3, 2+1)	3
5	6	3	5	4	1	max(3, 4+1)	5
6	2	5	5	5	3	max(5, 5+3)	8

最佳解 dp[6] = 8

option one ⇒ 選 activity (2, 4, 3) + activity (5, 5, 5)

(如果 start time 不可與 end time overlap)

	activity	start	end	value	p(j)	狀態計算	dp[j]
j=1	5	1	3	1	0	max(0, 1+0)	1
2	4	1	4	3	0	max(1, 3+0)	3
3	1	2	4	3	0	max(3, 3+0)	3
4	3	3	4	2	1	max(3, 2+1)	3
5	2	5	5	5	3	max(3, 5+3)	8
6	6	3	5	4	1	max(8, 4+1)	5

8. 如果 start time 可以 finish time overlap

① 先照 finish time sorting，

令 s_i 為 activity i 的 start time，f_i 為 activity i 的 finish time，

$S_{ij} = \{a_k : f_i < s_k < f_k < s_j\} \Rightarrow S_{ij}$ is the subset of activities in S that can start after activity a_i finishes and finish before activity a_j starts。$f_0 = 0$ and $s_{n+1} = f_{n+1}$，then $S = S_{0,n+1}$

令 $H(i) = \max\{k = \{1,2,\dots,i-1\}\} \mid f_k \le s_i\}$

↳ 在 i 之前可選擇的 activity　　↓

前一個 finish time <= 當前 start time

令 $A(i)$ = 考慮 activity i 的最大 value，

令 w_i 為 activity i 的 weight/value

$$A(i) = \begin{cases} 0, & i = 0 \\ \max\{A(i-1), w_i + A(H(i))\} & o.w. \end{cases}$$

activity	1	2	3	4	5	6
start	1	1	2	3	3	5
finish	3	4	4	4	5	5
weight	1	3	3	2	4	5
H(i)	0	0	0	1	1	5
A(i)	1	3	3	3	5	10

A: Max value = 10，選 (1,3,1) + (3,5,4) + (5,5,5)