Divide & Conquer 的策略

設一問題輸入為 I，依下面 steps 處理該問題：

① Divide：把 I 分割成多個子問題 $I_1, I_2, \dots, I_m$

② Conquer：遞迴呼叫來解 $I_1, \dots, I_m$，得到 $a_1, \dots, a_m$ 之 m 個解

③ Combine：將 $a_1, \dots a_m$ 合併回原問題解

Merge sort

Problem：給定 n 筆資料 $A[1, \dots, n]$，求由小到大排序後的 $A^*[1, \dots, n]$

Algorithm：① Divide：將 $A[1, \dots, n]$ 分割成 $A[1, \dots m]$ 和 $A[m+1, \dots, n]$，其中：$m = \lfloor \frac{n}{2} \rfloor$

② Conquer：利用 Merge_Sort 求得排序好的 $A^*[1, \dots, m]$ 和 $A^*[m+1, \dots, n]$

③ Combine：I. Create 一個新的陣列 $B[1 \dots n]$

II. 利用 i 指向 $A^*[1, \dots, m]$ 前端，若 $A[i] < A[j]$ 將 $A[i]$ 放入 B 中，重複直至掃過 $A^*[1, \dots, m]$ 或 $A^*[m+1, \dots n]$

j 指向 $A^*[m+1, \dots n]$ 前端　　$A[j] < A[i]$ 將 $A[j]$ 放入 B 中

III. 將 $A^*[1, \dots, m]$ 或 $A^*[m+1, \dots n]$ 剩餘元素放入 B

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

$\therefore T(n) = \theta(n \lg n)$

Maxium Subarray Problem：

Problem：給定一整數數列：$A[1, \dots, n]$，求其具最大整數和的值為何？

Algorithm：I. bruteforce：列舉所有 subarray 的和，取其最大值

等同取兩數作 subarray 的頭尾

再算出 $A[i, \dots, j]$ 的和，取最大值　$O(C_2^n) \cdot O(n) = O(n^3)$

II. bruteforce 改良：先做 preprocessing 計算：$B[j] = \sum_{k=1}^{j} A[k]$

則 $A[i, \dots, j]$ 可用 $B[j] - B[i-1]$ 得到 $\Rightarrow O(1)$

會為：$O(n^2)$

III. Divide & Conquer：① Divide：將 $A[1, \dots, n]$ 分割成 $A[1, \dots m]$ 和 $A[m+1, \dots, n]$，其中：$m = \lfloor \frac{n}{2} \rfloor$

② Conquer：遞迴呼叫求解 $A[1, \dots, m]$ 和 $A[m+1, \dots, n]$，得到左右兩 maxiam subarray 的值：l, r

③ Combine：找出橫跨左右兩 subarray 的 maxium subarray，令其值為 s

則結果為：$\max(l, r, s)$

$\Rightarrow T(n) = 2T(\frac{n}{2}) + \theta(n) \Rightarrow T(n) = \theta(n \lg n)$

IV. Dynamic Programming:

① characterize the structure of an OPT

定義子問題為：求 A[1,...,i] 且含有 A[i] 的 maximum subarray

設其最優解為 OPT，其對應 optimal value 為 d_i

必為兩種 case：

(1) OPT 不包含 A[i-1]，則 OPT 會為 A[i]，$d_i = A[i]$

(2) OPT 包含 A[i-1]，則 optimal value 為：$d_i = A[i] + d_{i-1}$

而 $\max_{1 \le i \le n}(r_i)$ 為其解

② Recursively define the value of an optimal solution

$$d_i = \begin{cases} A[1] & \text{if } i=1 \\ \max(A[i], A[i]+d_{i-1}) & \text{otherwise} \end{cases}$$

Time Complexity: $O(n) \times O(1) = O(n)$

③ Compute the value of an optimal solution

e.g. $A = [5, -7, -1, 2, 3, -1, 2, -3]$

	5	-7	-1	2	3	-1	2	-3
d_i	5	-2	-1	2	5	4	6	3

Algorithm:

```
Max_Subarray_DP(A)
  max = A[1] // 記錄 d_i 最大值
  d = A[1] // d_i，∵只會用到 d_{i-1} ∴不用用 table
  for i = 2 to n
    if d + A[i] < A[i]
      d = A[i]
    else d = d + A[i]
    if d > max
      max = d
  return max
```

Matrix Multiplication

Problem: 設 $A \in F^{n \times n}$，$B \in F^{n \times n}$ 則 $AB=C$，求得 C 共需做 n^3 次純量乘法

欲減少純量乘法次數至 $O(n^{\lg 7})$

Algorithm:

① 無用的 divide & conquer

利用方塊矩陣特性：

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{11} & B_{22} \end{bmatrix} \quad \text{則：} \quad C = AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{11}B_{21} & A_{21}B_{12} + A_{11}B_{22} \end{bmatrix}$$

則：$T(n) = 8T(\frac{n}{2}) + O(n^2)$

$\therefore T(n) = \theta(n^3)$ | 無用 |

② Strassen's Method:

利用數學技巧減少子問題個數

則：$T(n) = 7T(\frac{n}{2}) + O(n^2)$

$\Rightarrow T(n) = \theta(n^{\lg 7})$

Partition Problem:

Problem: 給定一 array，設經由重新排列後，可使所有在一數 pivot 前的都小於等於 pivot

而所有在一數 pivot 後的都大於等於 pivot，則完成，且分割成 pivot 前，後兩个 subarray

Algorithm: 和 quicksort 中一回合行為類似，設 pivot 為 p，先將 p swap 到 A[1]

利用兩指標 i 和 j 分別指向 A[2], A[n]，i 向右移动到 A[i] > pivot 時，j 向左移动到 A[j] < pivot 時

若 i, j 都停則 swap(A[i], A[j])，直至 j < i 時，pivot 和 A[j] swap 即完成

Time complexity: $T(n) = \theta(n)$