在生活中，我們有許多關於事情難易的形容，例如「難如上青天」、「反掌折枝之易」。隨著現代科技的發展，許多過去生活經驗中複雜的問題都變得更簡單了；例如，稅務計算、指紋辨識、基因比對等等都受惠於資訊科技而變得比較容易。也因此，「電腦萬能」成了許多人的共通看法。但是從資訊理論的基本角度來看這個問題是如何呢？現實世界中，問題之難易可能因人而異；因此，在探究電腦潛在能力之前，亦需對電腦的基本定義有所依據。現今電腦系統雖有各種不同的計算速度，但是本質上仍屬同一模式(稱為 Turing machine，詳細請參考[3])。在沒有可跳脫此模式，並可供廣泛使用之其他模式電腦之前，我們就以今日電腦所遵循 Turing machine 為基準來介紹 NP-Complete 的概念。

1999 年，IBM「深藍」擊敗人類西洋棋王，為人類運用電腦的深度與強度注入更多期待。隨著這樣的期望，我們要問有沒有什麼事情電腦會幫不上忙。舉一個資訊系老師都很想要擁有的東西為例—寫個母程式，只要將學生交過來的程式設計作業讀進去，之後可以判斷出程式作業是否正確。很可惜，這個在數十年前已經經由數學證明是不可行的，也就是電腦沒辦法提供老師們這項便利。在本文，我們僅就電腦能力可以處理的問題，介紹問題的複雜度概念。

首先，考慮兩個$n\times n$矩陣相乘的問題：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

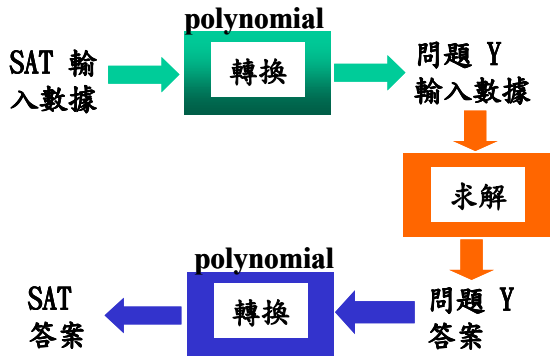
若是採用一般$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$方式，要算出一個$c_{ij}$所需的乘法與加法次數分別為$n$個，因此算出整個矩陣所需之計算步驟大約為$n^3$。接著，我們看一個更「簡單」的分割問題 (Partition problem) [1, 4]：給定一個整數集合$A = \{x_1, x_2, \ldots, x_n\}$，是否能將之分割成兩個子集合，使得其所含數字總和相等？例如，$A = \{3, 4, 6, 9, 14\}$可以分割成$A_1 = \{3, 6, 9\}$與$A_2 = \{4, 14\}$，而其個別數字總和均為 18。要確定是否可以成功分割，若數字個數不多，可以簡單推算出來；但是當數字多達數百或數千個之後，就不易使用手工計算。因此，設計演算法 (algorithm) 透過電腦程式來幫忙，列出各種可能分割組合再進行驗證，就成了可行的方式。那麼所需之計算步驟是多少呢？不考慮細項，單是可能組合有2^n種。如果針對不同的n值，我們將上述兩個問題的n^3與2^n表列如下：

n	10	20	40	60	80	100
n^3	1,000	8,000	64,000	$\sim 10^5$	$\sim 10^5$	$\sim 10^6$
2^n	1024	$\sim 10^6$	$\sim 10^{12}$	$\sim 10^{18}$	$\sim 10^{24}$	$\sim 10^{30}$

這些數字說明2^n係以呈指數成長；相對地，n^3的成長就和緩多了。若是以電腦一秒鐘可以處理10^{12}個計算步驟估計，那麼需要10^{18}秒 (約$10^{10} = 10$億年) 才能處理$n = 100$的「簡單」分割問題！即使電腦速度可以快上一千倍，也只不過去掉三個 0，因此分割問題好像是不簡單的。

當然，面對此情況，我們首先會想：是否可以設計個聰明演算法，而不是土法煉鋼式地列舉並檢驗所有組合呢？當然可以嘗試看看。在繼續探討這個想法前，我們先就以上的數字成長幅度，說明兩個關於所需計算步驟的重要概念：諸如 n^2, $n\log n$, n^{20}, … 等等的稱為多項式 (polynomial) 形式，至於 2^n, 5^n, $n!$ 之類者稱為指數 (exponential) 形式。如上表可知，一個演算法若其所需計算步驟是屬於指數形式，遇到大一點的問題(即 n 值比較大)，即無法在短時間內得到解答；反之，若是屬於多項式，所需耗費時間就比較小了。針對某一問題，若是所設計解題方法，無論處理如何的輸入數據，其所需計算步驟均是多項式形式，我們將它定義為有效率的 (efficient)，有時也簡稱為 P 演算法(P algorithm)。

在電腦發展歷程中，不管是實用或理論，學者們提出或處理過許許多多問題。這些問題中，有一些是可以設計出 P 演算法，例如矩陣相乘，另外，如何將數個整數由小到大排列的排序問題(sorting)，也有許多精采的 P 演算法[4]。然而，研究學者們同時卻也遭遇一些特殊問題，無人能設計出 P 演算法，前述之分割問題即是其中之一。當研究學者為這些問題所困惑的同時，更多找不到有效率演算法的問題陸陸續續被提出。到了 1971 年，學者 Stephen Cook 在國際學術會議上發表一項重要理論，此理論的中心是所謂的 Satisfiability problem (簡稱 SAT)：給定二元邏輯式 $F(x_1, x_2,\ldots, x_n)$，是否存在某組 $x_1, x_2,\ldots, x_n$ 的 ture/false 組合 (共 2^n 組可能) 使得 $F(x_1, x_2,\ldots, x_n)$ 為 true？如果針對某問題 Y，一直找不到 P 演算法，但是卻可以進行如下圖的轉換機制，其中來回兩次轉換所需步驟皆為多項式形式。這樣的轉換也就是說問題 Y 為 SAT 提供另一種的解題選擇，當然問題 Y 的複雜度也就不亞於 SAT。也因此，如果可以為問題 Y 設計出 P 演算法，那同樣 SAT 亦存在有 P 演算法，因為轉換、求解、再轉換三階段均可運用有效率之演算法處理。此理論提出之後，許多問題的研究都依此方式進行，因為問題 Y 以 SAT 為試金石說明其難度至少與 SAT 一樣，再引申而言，其他問題亦可採用問題 Y 為試金石。如此相生，學者們認知到之前一些特殊問題形成“落難”的等價群(equivalence class)，彼此間可以進行著轉換；亦即，若能夠為其中某一問題設計出有效率演算法，則所有問題的疑問都解決了。但是，目前為止，尚無學者能夠為其中任何一個問題設計出 P 演算法。



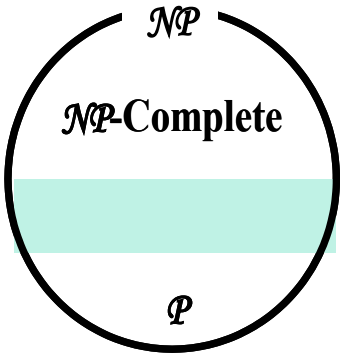
計算複雜度相關理論之問題歸類至此可以整理簡述如下：

1. 一個問題，如果存在有 P 演算法，則屬於集合 $\mathcal{P}$。

2. 針對一個問題給定一組答案，如果可以在多項式計算步驟內驗證其對錯，則此問題屬於 $\mathcal{NP}$。分割問題中，若給兩個子集合，要驗證兩個子集合內元素總和是否相同非常容易，因此屬於 $\mathcal{NP}$。很顯然，一個問題若屬於 $\mathcal{P}$，那自然也屬於 $\mathcal{NP}$。

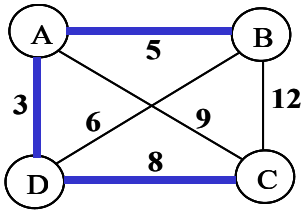
3. 若問題 Y 屬於 $\mathcal{NP}$，若是可以進行來自 SAT 或某個已知為 $\mathcal{NP}$-Complete 問題之轉換，則此問題亦稱為 $\mathcal{NP}$-Complete。

綜合而言，問題複雜度的群落關係如下圖所示，$\mathcal{P}$與 $\mathcal{NP}$-Complete 均為 $\mathcal{NP}$之子集合。目前眾多學者在努力的是釐清中間灰色地帶，解答“$\mathcal{NP}=\mathcal{P}$?”這個久懸的謎題。(1)為某個 $\mathcal{NP}$-Complete 問題找到有效率的演算法，則屬於 $\mathcal{NP}$-Complete 的所有問題也屬於 $\mathcal{P}$，也就是整個 $\mathcal{NP}$所涵蓋的問題都是容易處理的，即 $\mathcal{NP}=\mathcal{P}$；(2)證明某一個 $\mathcal{NP}$-Complete 問題根本就不可能存在 P 演算法，那麼就是 $\mathcal{NP}$-Complete 與 $\mathcal{P}$涇渭分明，$\mathcal{NP}$-Complete $\cap\ \mathcal{P}=\phi$ 且 $\mathcal{NP}$-Complete $\cup\ \mathcal{P}=\mathcal{NP}$，即 $\mathcal{NP}\neq\mathcal{P}$。無論是哪個方向的嘗試都極具挑戰性，因此誰可以解開謎團，有機會獲得資訊領域最高榮譽之 Turing Award，也許還會以其名設個獎項。



以下我們來看看幾個問題，他們都已知是屬於 $\mathcal{NP}$-Complete 或 $\mathcal{P}$：

1. 最小涵蓋樹 (Minimum spanning tree problem, MST)：有 n 個網路點，任兩點之間有一網路佈建成本，如何規劃出最小成本之網路，使得任意點都為此網路所涵蓋。以右圖為例，藍色線就是最佳的一個解。這個問題基本上就是要找出 n-1 條線來涵蓋所有點使得總成本最小。乍看之下，有點複雜，基本做法是列出所有可能組合，大概會有 $C_{n-1}^{n(n-1)/2}$ 種，這是個階乘式的數字，亦即指數形式的。但是在文獻中，有幾個非常漂亮，所需計算步驟大約 $n^2 \log n$ 之演算法[3]可以得到最佳解，所以 MST $\in \mathcal{P}$。

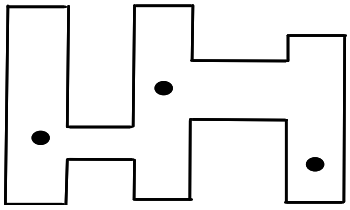


2. 背包問題(Knapsack problem)：尋寶者背著可承受一定重量(W)的背包入寶山，眾多寶石(n 個)各有其重量(w_i)與價值(v_i)，他(她)希望可以在重量限制下，拾取寶石以使總價值最大[4]。這樣一個單純的“貪心”問題事實上是屬於 $\mathcal{NP}$-Complete 的。

3. 穩定配對(Stable matching problem) ：我們不直接介紹這個問題，而是想一下大學聯考完之後的“學生—學校”分發問題，思索一下，電腦程式是如何進行分發的？似乎有點複雜。由於分發問題可以視為穩定婚姻理論之應用，而後者是有個非常有名的 n^2 演算法[3]，所以屬於 $\mathcal{P}$。

4. 最長共同字串問題 (Longest common subsequence problem)：這是文字(件)比對的一項關鍵問題，目前在基因序列比對中也被廣泛運用。給定兩個字串 $S1$ = cbbcd 與 $S2$ = cbacad，其最長共同字串為 cbcd。如果要嘗試各種長度組合，所需時間非常可觀。經過仔細觀察遞迴性質，可以設計出計算步驟為 n^2 之演算法 [4]。

5. 警勤配置問題(Art gallery problem)：給定一個樓層的平面圖，至少需要配置幾員警衛方能監控整個樓層的所有位置？例如右圖中總共配置了三位。這個問題從視覺上做判斷好像可以輕易找出答案，但是當問題規模變大以後，就較為複雜。此問題業已經證明為 $\mathcal{NP}$-Complete [4]。



從上面幾個例子，我們會覺得有一些結構複雜的問題是容易的，一些看似簡單的問題卻反而屬於 $\mathcal{NP}$-Complete。一般而言，面臨一個未曾被討論過的問題我們會嘗試去設計 P 演算法，如果都無法成功，可以轉向懷疑此問題是 $\mathcal{NP}$-Complete。要進行證明，那就是找個合適且已知為 $\mathcal{NP}$-Complete 的問題，進行先前介紹過的轉換機制。當然，要提出 P 演算法或證明 $\mathcal{NP}$-Complete 都不一定很容易，這就是學術研究的挑戰與趣味之所在。舉例而言，因數分解(Factorization problem)就是個懸案：給定整數 n，若已知其可分解成 p 與 q 均為質數的乘積($n = p \times q$)，但未知 p 與 q 值，請問如何求出 p 與 q 的值？這個問題是很多密碼技術的理論基礎，目前沒有人可以證明它是屬於 $\mathcal{NP}$-Complete，但是至今亦尚無人可以提出 P 演算法，所以才會說要在短時間內破解密碼是不容易的(如果 n 是 300 位數，則可能要百萬年才能解開來)。

結束前，我們再次澄清討論問題時常被提及的口誤：「這個問題是 $\mathcal{NP}$，所以很難。」這句話是有誤的，因為一個問題屬於 $\mathcal{NP}$，也可能是比較簡單的 $\mathcal{P}$。以目前而言，「這個問題是 $\mathcal{NP}$-Complete，所以很難。」說法是可接受的，至少尚未有智者可以提出 P 演算法。

註：為說明方便，因此簡略 $\mathcal{NP}$-complete 與 $\mathcal{NP}$-hard 之差異以及與 input length 相關之 pseudo-polynomial time 嚴謹性，詳細定義請參考[1]。

參考文獻：

[1] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freedman, San Francisco, CA, 1979.

[2] D. Gusfield and R.W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, Cambridge, Mass, 1989.

[3] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979.

[4] R.C.T. Lee, S.S. Tseng, R.C. Chang and Y.T. Tsai, *Introduction to the Design and Analysis of Algorithms*, 旗標出版股份有限公司, 2001 年 9 月出版。