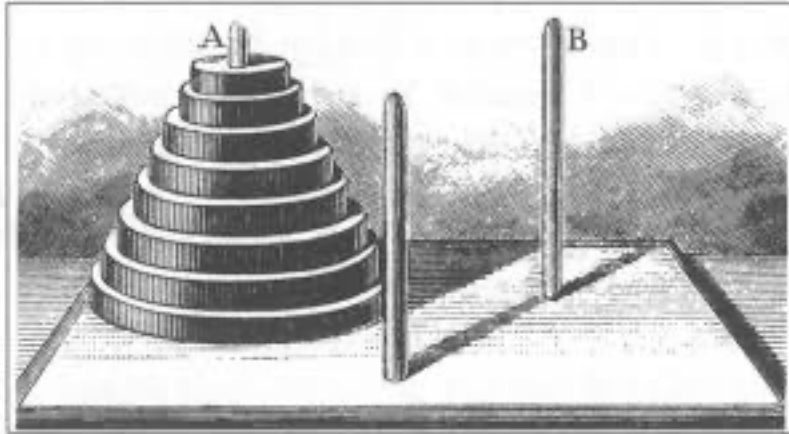


# Hanoi Tower C



河內塔據傳是在越南河內煩天寺中，以三根銀棒和 64 個金盤組成，每個金盤大小不一且從上至下由小到大排列，要求是將所有金盤搬移到另一根上，而搬運的規則是：

- 1.盤子必須在三根銀棒中被移動，不能取下暫置在旁邊。
- 2.每次只能移動一個盤子。
- 3.尺寸較大的金盤永遠在較小的金盤下方。

以遞迴的結果而言，最小移動步數為  $2^n - 1$ ，而 64 層的河內塔如需全部搬移完畢，假設一秒進行一次操作，需約 5849 億年。

和 Recursion 的關係是：我們需要程式來處理遞迴函數，而就需要 Recursion 重複執行。

如何用 Recursion 解決這個問題：設立 A、B、C 三根銀棒，兩根當作輔助，一根作為目標

Hanoi(n-1,A,B,C); 表示將輔助柱當作目標柱。

Hanoi(n-1,B,A,C);表示將輔助柱的盤子搬回。

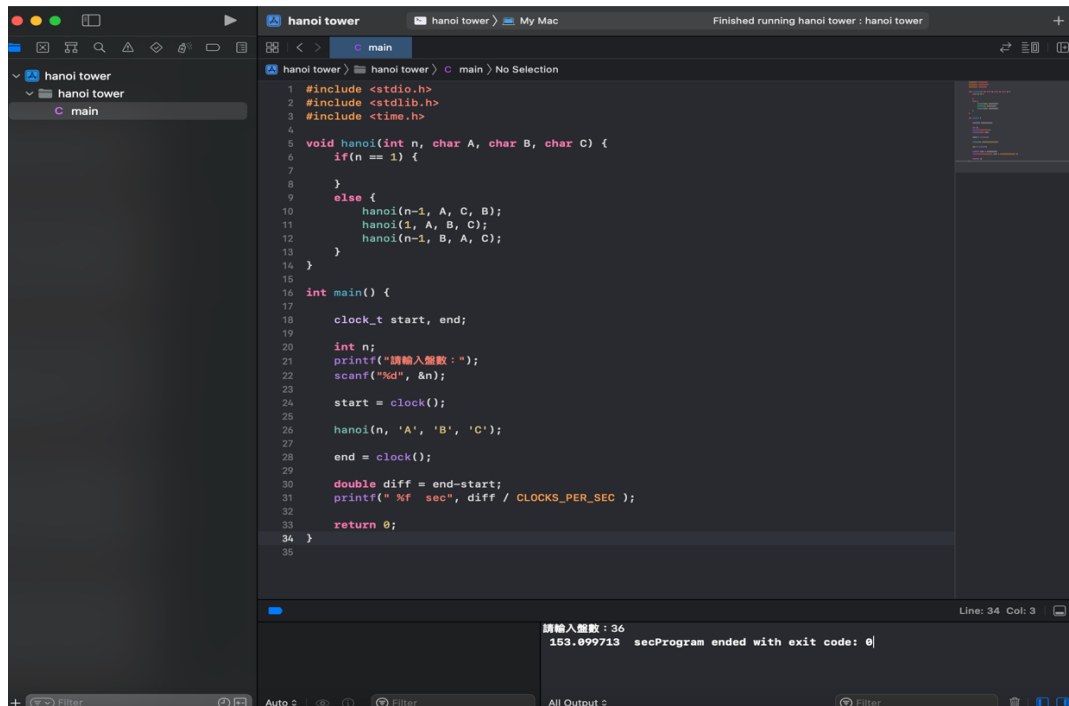
使用的電腦：MacBook Pro 2021

處理器:M1 Pro

執行程式：Xcode

執行結果：

既然 16 層對目前大部分 C P U 而言所需時間很低，這邊直接嘗試 36 層：

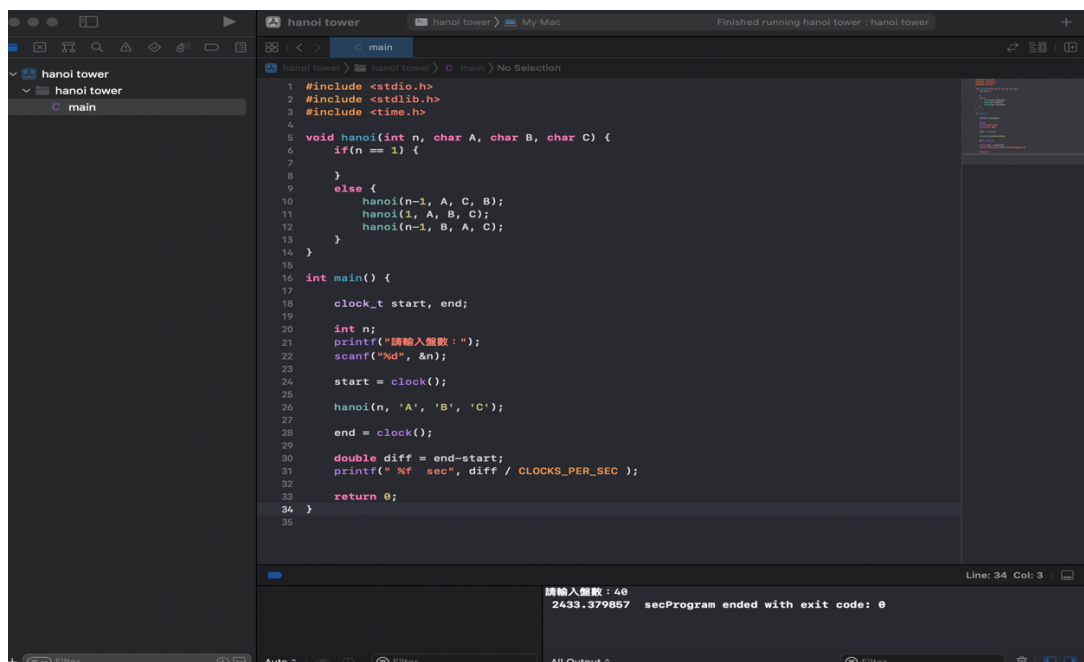


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 void hanoi(int n, char A, char B, char C) {
6     if(n == 1) {
7         printf("%c to %c\n", A, C);
8     }
9     else {
10        hanoi(n-1, A, C, B);
11        hanoi(1, A, B, C);
12        hanoi(n-1, B, A, C);
13    }
14 }
15
16 int main() {
17
18     clock_t start, end;
19
20     int n;
21     printf("請輸入盤數:");
22     scanf("%d", &n);
23
24     start = clock();
25
26     hanoi(n, 'A', 'B', 'C');
27
28     end = clock();
29
30     double diff = end-start;
31     printf(" %f sec", diff / CLOCKS_PER_SEC );
32
33     return 0;
34 }
```

Line: 34 Col: 3

請輸入盤數: 36  
153.099713 secProgram ended with exit code: 0

好像很輕鬆，那來試試 40 層：

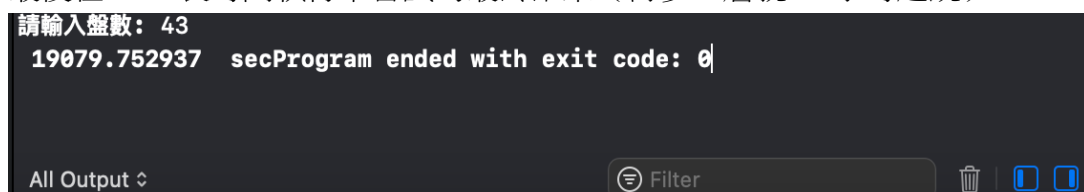


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 void hanoi(int n, char A, char B, char C) {
6     if(n == 1) {
7         printf("%c to %c\n", A, C);
8     }
9     else {
10        hanoi(n-1, A, C, B);
11        hanoi(1, A, B, C);
12        hanoi(n-1, B, A, C);
13    }
14 }
15
16 int main() {
17
18     clock_t start, end;
19
20     int n;
21     printf("請輸入盤數:");
22     scanf("%d", &n);
23
24     start = clock();
25
26     hanoi(n, 'A', 'B', 'C');
27
28     end = clock();
29
30     double diff = end-start;
31     printf(" %f sec", diff / CLOCKS_PER_SEC );
32
33     return 0;
34 }
```

Line: 34 Col: 3

請輸入盤數: 40  
2433.379857 secProgram ended with exit code: 0

最後在 CPU 長時間執行下嘗試的最終結果（再多一層就 10 小時起跳）：



```
請輸入盤數: 43
19079.752937 secProgram ended with exit code: 0
```

All Output ⌵

Filter