Ackermann

Ackermann 函數的時間複雜度如下

A (0.n)　=> O(1)

A (1.n)　=> O(n)

A (2.n)　=> O(2^n)

A (3.n)　=> O(2^{2^n})

A (4.n)　=> O(2^{2^{2^n}})

通過實際演算來理解 Ackermann 函數的計算過程，A(m,n) = 2 2

A (2,2)

A (1, A(2,1))

A (1, A(1, A(2,0)))

A (1, A(1, A(1, 1)))

A (1, A (1, A (0, A (1,0))))

A (1, A(1, A(0, A (0,1))))

A (1, A (1, A (0, 2)))

A (1, A(1, 3))

A (1, A (0, A (1,2)))

A (1, A (0, A (0, A(1, 1))))

A (1, A (0, A (0, A(0, A(1, 0)))))

A (1, A (0, A (0, A(0, A(0, 1)))))

A (1, A (0, A(0, A(0, 2))))

A (1, A (0, A (0, 3)))

A (1, A (0, 4))

A (1, 5)

A (0, A (1, 4))

A (0, A (0, A (1,3))

....

A (0, A (0, 5))

A (0, 6)

=7

Ackermann4.sln 為使用遞迴版本

計算 A(2,2)只需要 0 微秒，A(3,3)則需要 32 微秒

但是計算 A(4,1)就會因為遞迴次數溢位而無法執行

而經過測試 A(3,n)，如果 n>=9 也無法執行

Ackermann3.sln 為 z 非使用遞迴的版本

計算 A(0,0)時需要 607 微秒，計算 A(1,0)時則需要 514 微秒

但是計算 A(1,1)實則會因為超出陣列的範圍而無法繼續執行

實際在製作時會因為陣列長度而影響，如果用預設長度會

導致在儲存資料時不夠用，但是改成動態記憶體配置也會

因為記憶體不夠而失敗。