# 多媒體技術概論 HW4 - 101062124 戴宏穎

A)檔案結構：

```
├── hw4_a.m (Part a)
├── hw4_b.m (Part b)
├── hw4_c.m (Part c)
├── sad.m   (Util for computing sad)
├── full_search.m (implement)
├── logarithmic.m (implement)
├── FST.m (implement for timer)
├── LOGT.m (implement for timer)
├── output
│   ├── caltrain008_full_16_16.bmp
│   ├── caltrain008_full_16_8.bmp
│   ├── caltrain008_full_8_16.bmp
│   ├── caltrain008_full_8_8.bmp
│   ├── caltrain008_log_16_16.bmp
│   ├── caltrain008_log_16_8.bmp
│   ├── caltrain008_log_8_16.bmp
│   ├── caltrain008_log_8_8.bmp
│   ├── caltrain017_full_16_16.bmp
│   ├── caltrain017_full_16_8.bmp
│   ├── caltrain017_full_8_16.bmp
│   ├── caltrain017_full_8_8.bmp
│   ├── caltrain017_log_16_16.bmp
│   ├── caltrain017_log_16_8.bmp
│   ├── caltrain017_log_8_16.bmp
│   └── caltrain017_log_8_8.bmp
└── input
    └── caltrain0XX.bmp
```

B)Objective: 實作兩個 ME (motion estimation) 的演算法，並比較結果。

C)演算法：分別是 Full Search, 2D Logarithmic Search

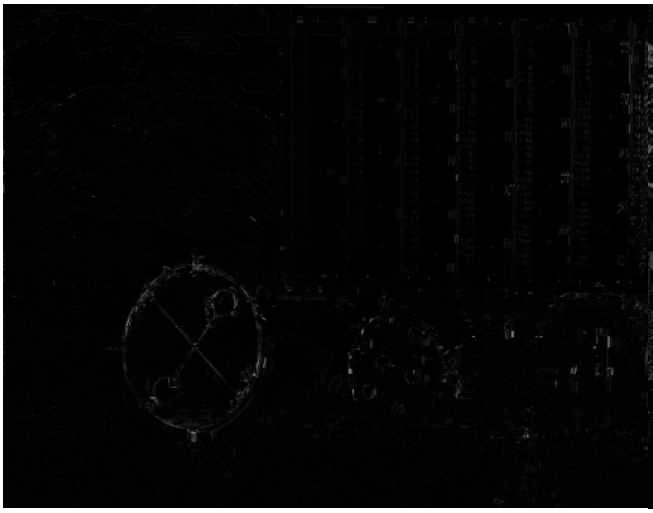
a) Full Search 顧名思義就是全部掃過一遍的意思，把 target frame 上每個 macro block 都對應回 reference frame 上的參考 macro block 再以這個參考 macro block 為中心擴展 d 來框出一塊範圍。接著在裡頭找出與 target frame macro block 差距最小的 SAD ，這樣位移的差值就是我們要的 motion vector。

b) 2D Logarithmic Search ，看到 log 就代表他能用指數方式遞減，因此這個演算法的精髓在於每次會找出包含中心點的五個點 [(0,0),(0,n),(n,0),(0,-n),(-n,0)]，如果周圍的那四個比自己的 SAD 還低，那麼就往那個點偏移，繼續同樣的模式，否則我就縮小自己的範圍再繼續同樣的方式搜尋，直到範圍只剩下自己一個點，這時候會去搜尋以自己點為中心的周圍八個格子，找出最低的 SAD ，算出位移差值就是我們要的 motion vector。

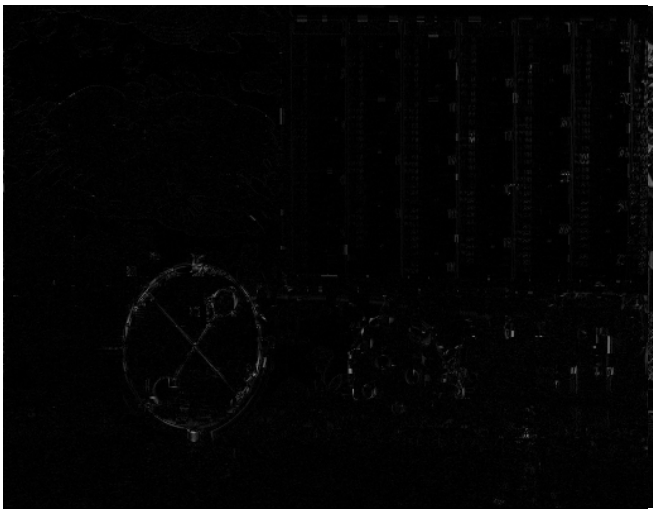
D) Part a：BMP(2) * ME(2) * D(2) * N(2) = 16 張圖片，輸出在 output 資料夾中透過下列十六張圖片，我們可以明顯地發現，017 離原圖較遠，所以白色的部份（誤差）就十分明顯，008 離原圖較近的情況下，確實幾乎快看不出有白色的區塊。兩種演算法相比， Full Search 因為是扎扎實實做，所以明顯的白色誤差比較少。

008

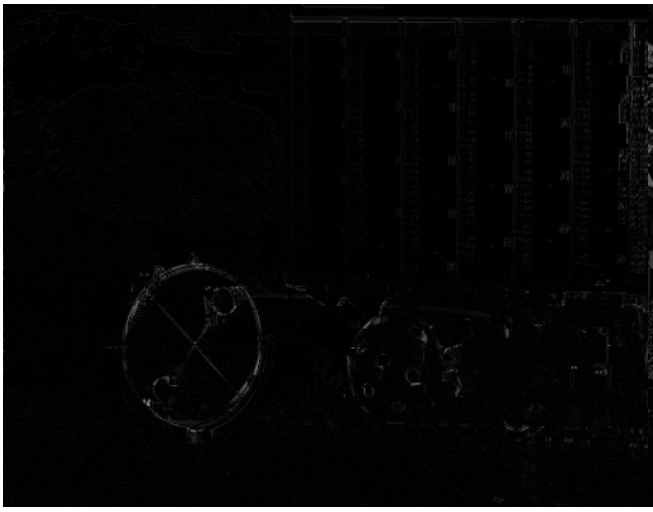
Full Search N = 8 D = 8



Full Search N = 16 D = 16



Full Search N = 16 D = 8

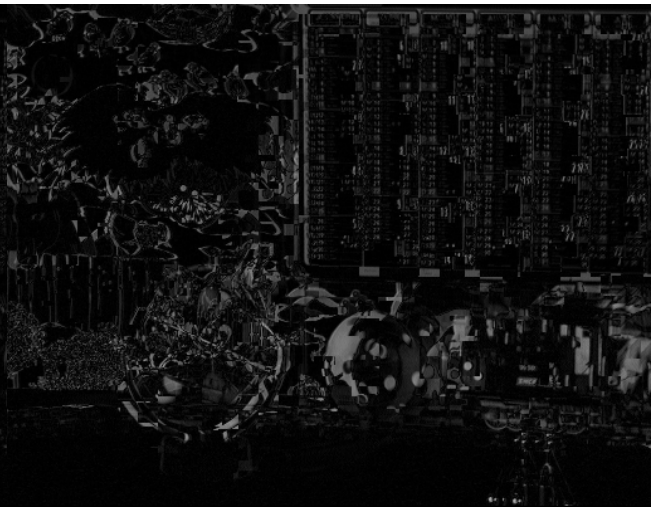


017

Full Search N = 8 D = 8

Full Search N = 8 D = 16

Full Search N = 16 D = 8



008

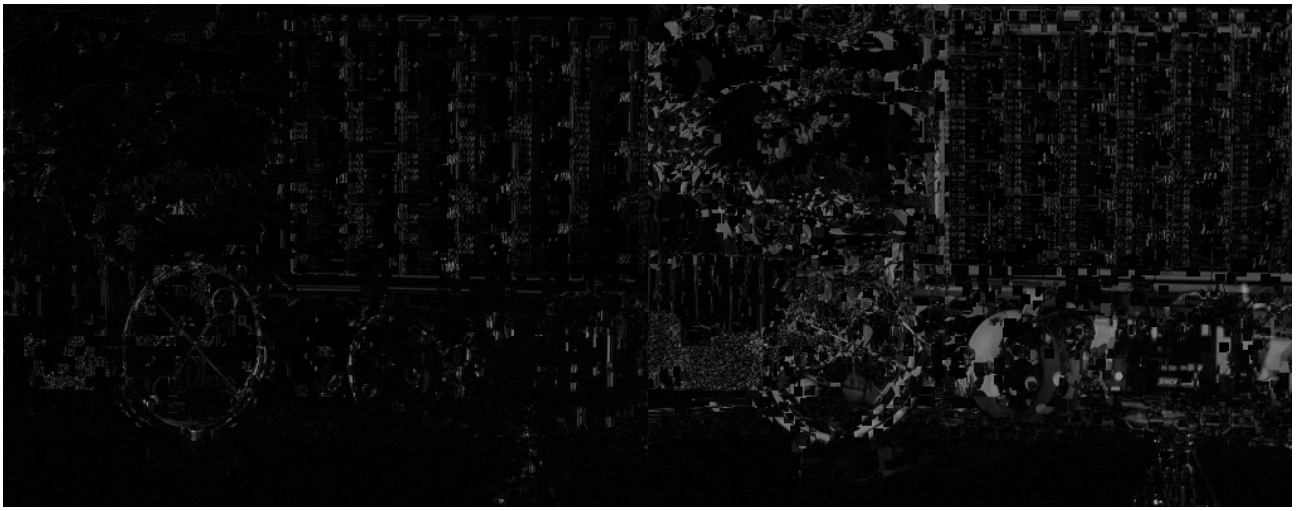
Full Search N = 16 D = 16

017

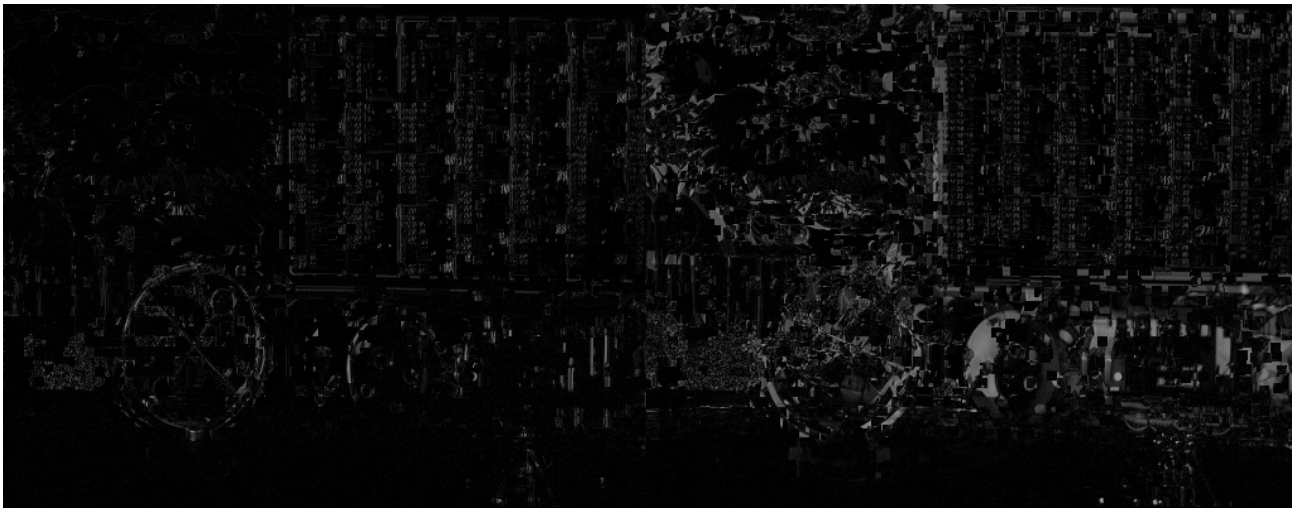
Full Search N = 16 D = 16

008

2D Log N = 8 D = 8



2D Log N = 8 D = 16



2D Log N = 16 D = 8

017

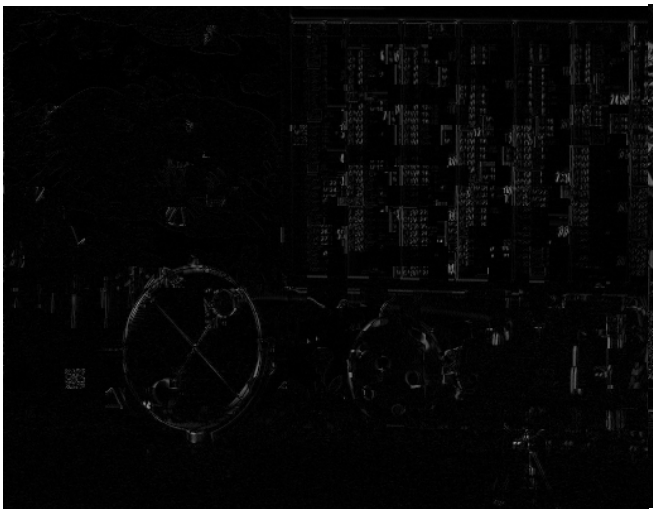
2D Log N = 8 D = 8

2D Log N = 8 D = 16

2D Log N = 16 D = 8

008

2D Log N = 16 D = 16

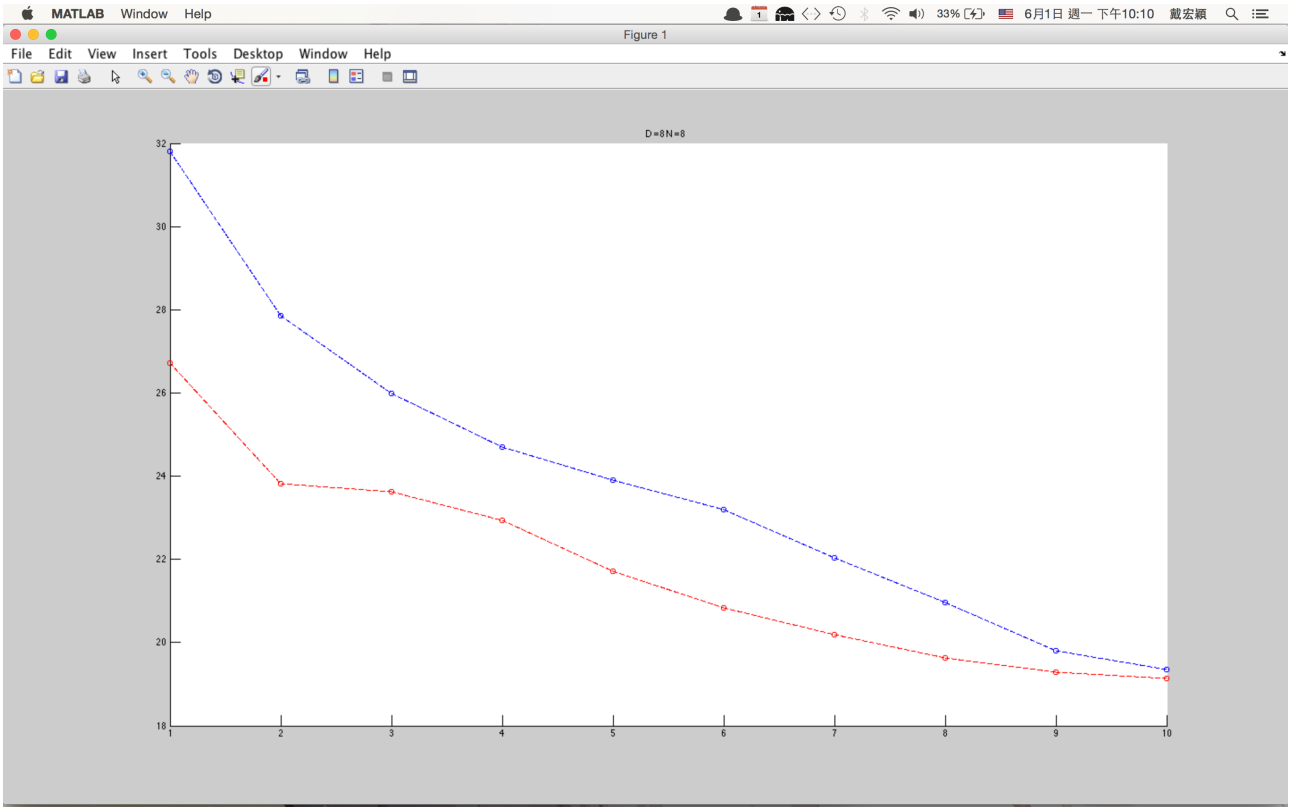


017

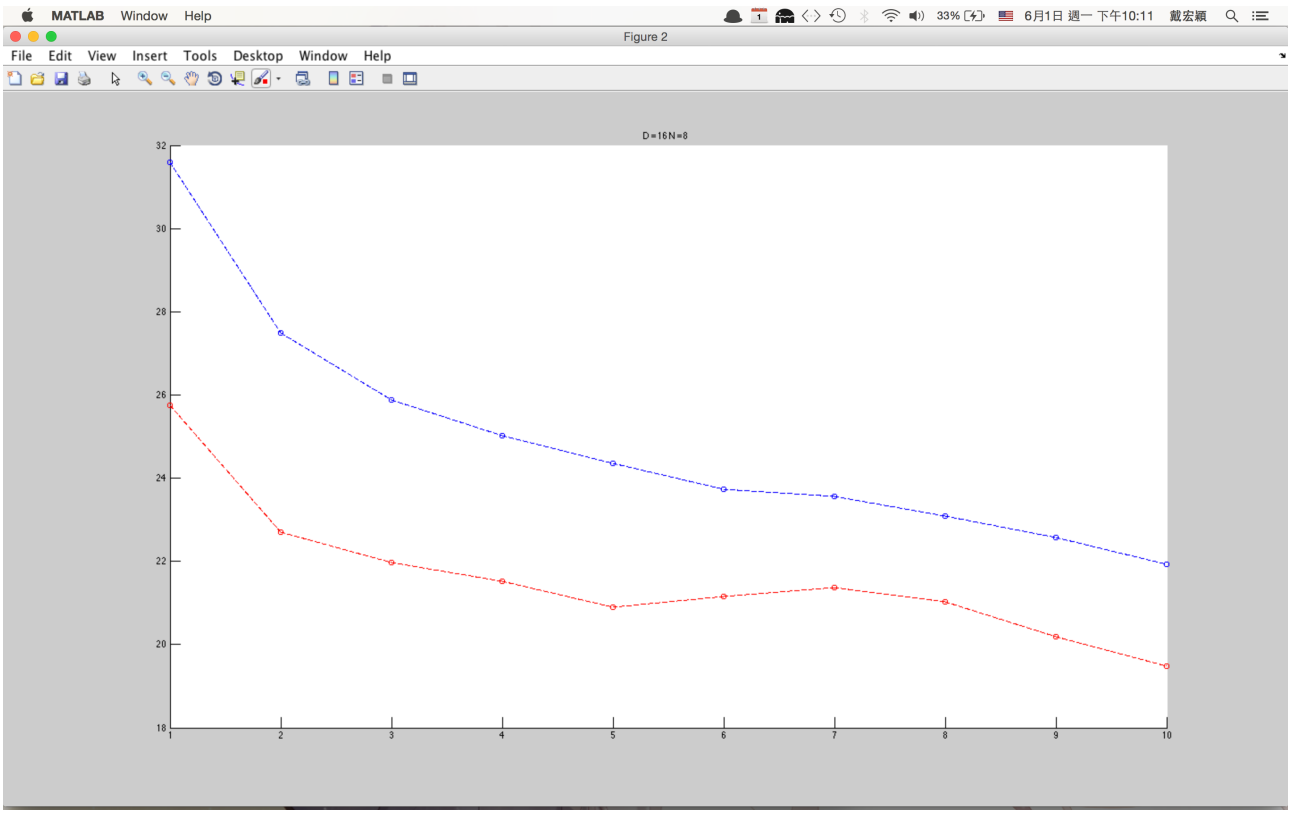
2D Log N = 16 D = 16

E) Part b：比較兩種演算法的 PSNR 看哪一種比較好。（紅色是 2D-log, 藍色是 Full），PSNR 越高代表失真越少，想當然爾，我們可以預期 Full Search 的失真是比 2D-Log 少的，看輸出的圖也是如此。

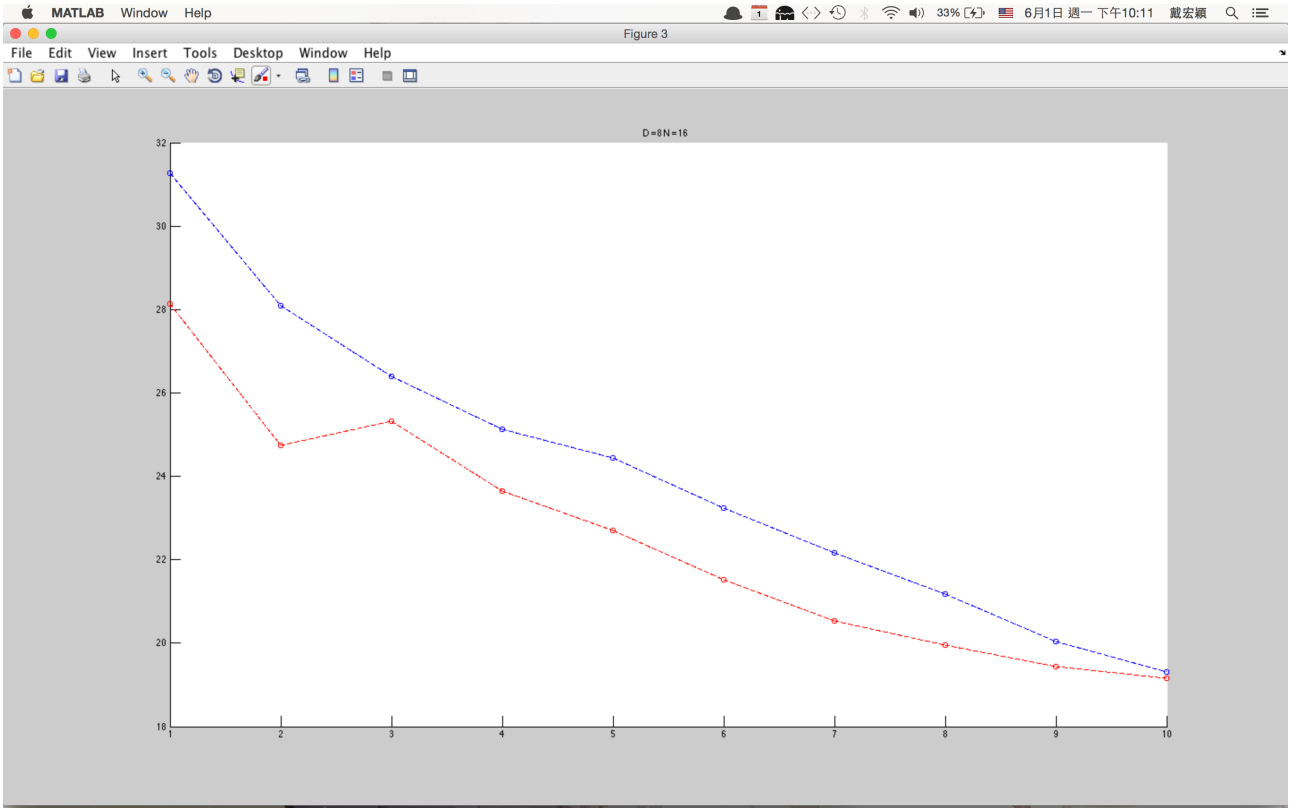
N = 8, D = 8



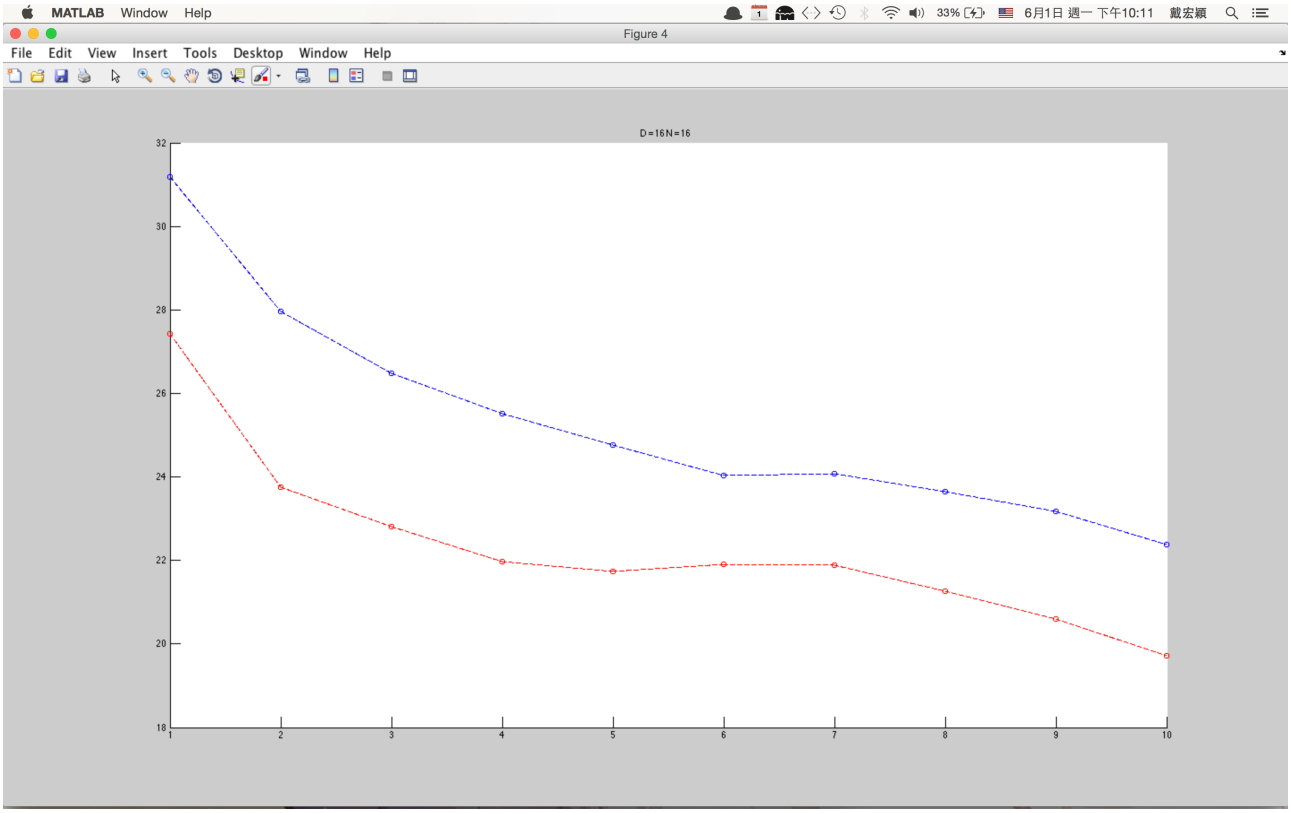

N = 8, D = 16



N = 16, D = 8




N = 16, D = 16



F) Part c：算一下時間複雜度，測一下時間是否符合猜測

Define:

Length: L

Range: D

Macro block size: N

Full Search:

對每塊都掃過：O(L/N)*O(L/N) ，方便計算 L 取一樣大

列出每個 SAD 起點：O(N+2*D)*O(N+2*D)

對每個 SAD 起點做 SAD：O(N)*O(N)

時間複雜度：O(L/N)*O(L/N)*O(N+2*D)*O(N+2*D)*O(N)*O(N)

= O(L^2)*O((N+2D)^2)

2D Log:

對每塊都掃過：O(L/N)*O(L/N)，方便計算 L 取一樣大

找最小的 SAD ：O(log(D))

計算該 SAD ：O(N*N)

時間複雜度：O(L/N)*O(L/N)*O(log(D))*O(N*N) = O(L^2) * O(log(D))

由上面我們可以發現對 Full Search 而言，當 N 變兩倍時，速度應該會快四倍，當 D 變兩倍時，速度會慢四倍，N, D 同時增長時，剛好效應會差不多抵消，可以和下面的輸出有類似的比擬。

對 2D Log 而言，因為 D 在 log 項，因此影響不大，反而是 N 變兩倍時的影響可以讓速度快四倍。

這邊是實際跑出來的數據：

```
Full Search N = 8, D = 8, T = 4.890421
2D Log N = 8, D = 8, T = 0.606508
Full Search N = 8, D = 16, T = 17.757685
2D Log N = 8, D = 16, T = 0.683937
Full Search N = 16, D = 8, T = 1.645538
2D Log N = 16, D = 8, T = 0.183595
Full Search N = 16, D = 16, T = 5.977531
2D Log N = 16, D = 16, T = 0.216315
```