Exercise 6 CUDA Threads

สิ่งที่ต้องใช้ 1. WinSCP 2. Terminal 3. Code Editor 4. CUDA (nvcc or Nvidia compiler)

1. Create a CUDA program to assign a value to each element of the array of integers X[N] using multiple thread blocks. Each X[i] = 2*i and N can be any positive integer.

ข้อนี้เราจะต้องเขียนโค้ดด้วย CUDA (C++) **โดยให้ค่าแต่ละตัวนั้นจะอยู่ในรูปของ** X[N] **โดยใช้เธรดแบบหลายตัว ใช้พวก** DimGrid , DimBlock **นั่นแหละ โดยแต่ละตำแหน่ง ของ** X[i] **จะเท่ากับ** 2*i **และ** N **จะต้องเป็นค่าบวก**

```
#include <stdio.h>
#define T 16 // As Threads
#define array_size 64

__global__ void vecMultiply(int *A)
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        A[i] = A[i] * 2;
}

int main (int argc, char *argv[])
{
        int i;
        int size = T*sizeof(int);int a[array_size], *devA;
        for (i=0; i< array_size; i++)
        {
                a[i] = i + 1;
        }

        cudaMalloc( (void**)&devA,size);
        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
    dim3 dimBlock(T,T);
    dim3 dimGrid(array_size/T - 1);
    vecMultiply<<<dimGrid,dimBlock>>>(devA);
        printf("Before\n");
        for (i=0; i< array_size; i++)
        {
                printf("%d ", a[i]);
        }
        printf("\n");

        cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
        cudaFree(devA);
        printf("After\n");
        for (i=0; i < array_size; i++)
    {
                printf("%d ",a[i]);
        }
        printf("\n");
```

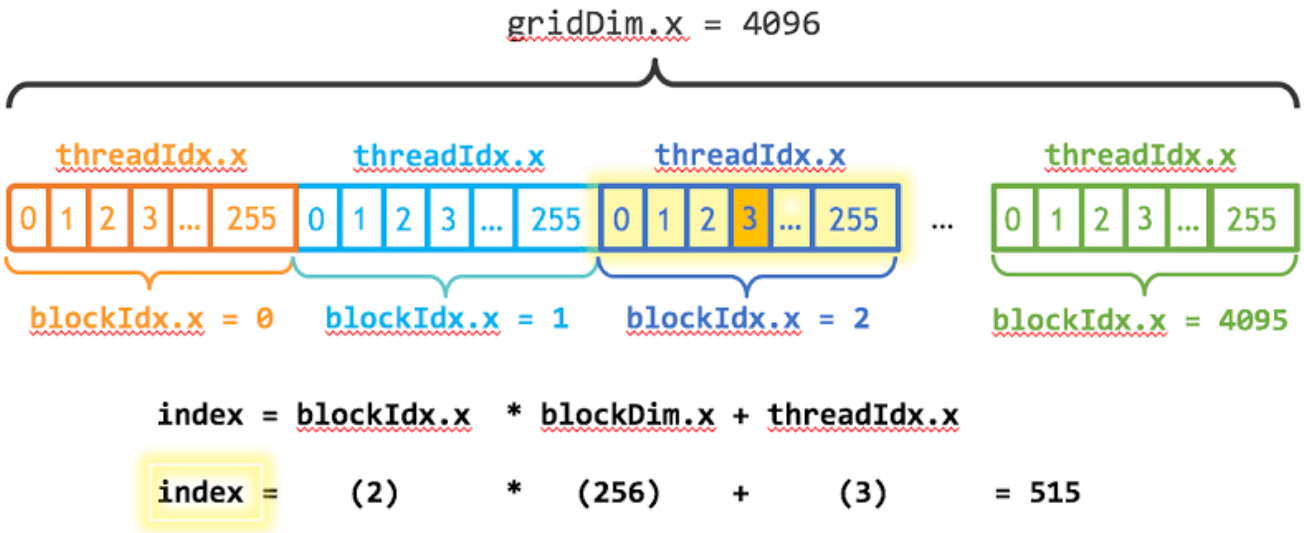
```
    return 0;
}
```

ตรงนี้

```
__global__ void vecMultiply(int *A)
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        A[i] = A[i] * 2;
}
```

int i = blockIdx.x * blockDim.x + threadIdx.x; ตรงนี้จะเป็นการคิดเลขแบบ multiple thread โดยที่ blockIdx คือ **block Index** จะทำหน้าที่เป็นดัชนีเพื่อเก็บค่าที่ต่างกันในอาเรย์ ส่วน **blockDim** คือ **Block Dimension** ทำหน้าที่เป็นตัวบอกตำแหน่งของเธรดในบล็อกนั้นๆ ส่วน **threadIdx** คือ **Thread Index** ทำหน้าที่เก็บค่าดัชนีสำหรับเธรดในแต่ละบล็อก ส่วนตรงนี้

```
dim3 dimBlock(T,T); //  ก็คือเธรดในบล็อก หรือ ขนาดเธรดในมิติที่เรากำหนด
dim3 blockDim(T,T); // เหมือนกับ dimBlock นั่นแหละ
dim3 dimGrid(array_size/T - 1); // เป็นขนาดเส้นกริดของมิติที่เรากำหนด
dim3 gridDim(array_size/T - 1); // เหมือนกันกับ dimGrid
```



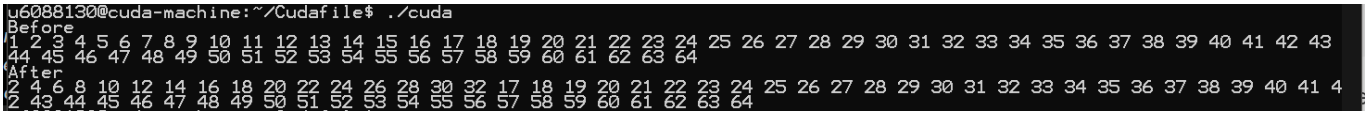
ในตัวอย่างคือเส้นกริดขนาด **4096** โดยที่แต่ละ **blockDim** นั้นในตัวอย่างจะแบ่งโดยใส่ **Thread** ลงใน **Block** ทั้งหมด 256 ตัว (0 - 255) ซึ่ง blockIdx.x 1 ตัว นั้นจะมีค่าเท่ากับ thread 256 ตัว ส่วน threadIdx นั้นจะอยู่ใน blockDim ซึ่งก็คือ (0 - 255) ไปจนครบ blockIdx.x (0 - 4095) หรือ gridDim(4096)

ตัวอย่างในโจทย์คือ threadIdx.x **ตัวที่** 3 **ใน** blockIdx.x **ที่** 2 **นั้น โจทย์ถามหาว่าตัว** threadIdx.x **ตัวนี้อยู่ตรงไหน**

```
    index =  blockIdx.x * blockDim.x + threadId.x
# ตำแหน่งที่จะหา = ตำแหน่ง block ที่threadโดนไฮไลท์(blockIdx.x) * ตัวเธรดที่อยู่ในบล็อกนั้น
```

```
ทั้งหมด(blockDim.x) + ตำแหน่งเธรดที่โดนไฮไลท์ (threadId.x)
    index = 2 * 256 + 3 = 515
```

dim3 คือ integer **ในรูปแบบเวกเตอร์** (Vector) **มาจาก** uint3 (unsigned integer3) **ไว้สำหรับระบุมิติของอาเรย์ เมื่อเราใช้** dim3 **เมื่อไหร่ก็ตาม ค่าที่ยังไม่ได้โดน** assign **จะถูก** assign **เป็น** 1 **เสมอ ส่วนผลลัพธ์ที่ได้นั้นจะเป็นแบบนี้**



2. Write a CUDA program to copy array A into array B in reverse order using multiple thread blocks. Both array A and B are of an arbitrary size N.

ข้อนี้ให้เราทำการคัดลอกค่าในอาเรย์ A **ไปอาเรย์** B **โดยที่ค่าในอาเรย์** B **นั้นต้องเรียงลำดับจากท้ายสุดมาตัวแรก** (Reverse) **โดยที่ขนาดอาเรย์ของทั้งคู่นั้นสามารถกำหนดได้ตามใจตัวเองด้วยขนาด** N

```
#include <stdio.h>
#define T 64 // As Threads
#define array_size 256

__global__ void vecMultiplyReverse(int *A, int *B)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int Reverse = (T - 1) - i;
    B[Reverse] = A[i];
}

int main (int argc, char *argv[])
{
    int i;
    int size = T*sizeof(int);
    int a[T],b[T], *devA,*devB;
    for (i=0; i< T; i++)
    {
        a[i] = i + 1;
    }
    cudaMalloc( (void**)&devA,size);
    cudaMalloc( (void**)&devB,size);
    cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy( devB, b, size, cudaMemcpyHostToDevice);
    dim3 dimBlock(T);
    dim3 dimGrid(array_size/T - 1);
    vecMultiplyReverse<<<dimGrid,dimBlock>>>(devA,devB);
    printf("Before\n");
    for (i=0; i< T; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");

    cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
    cudaMemcpy(b, devB, size, cudaMemcpyDeviceToHost);
```

```
    cudaFree(devA);
    cudaFree(devB);
    printf("After\n");
    for (i=0; i < T; i++)
    {
        printf("%d ",b[i]);
    }
    printf("\n");
}
```

มาดูตรงนี้

```
__global__ void vecMultiplyReverse(int *A, int *B)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int Reverse = (T - 1) - i;
        B[Reverse] = A[i];
}
```

ตรงนี้คือ Kernel ที่เอาไว้ Reverse **ค่านั่นแหละ ตัวอย่าง**

```
A         : 1 2 3 4 5 6 7 8 9 10
A Reverse : 10 9 8 7 6 5 4 3 2 1
```

แบบนี้เป็นต้น ซึ่งเราจะใช้ Multiple Thread **หรือตรงบรรทัด int i = blockIdx.x * blockDim.x + threadIdx.x; อันนี้แหละ และผลลัพธ์ที่จะได้เป็นแบบนี้**

```
u6088130@cuda-machine:~/Cudafile$ ./cuda1
Before
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
After
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

3. Given two arrays A[N] and B[N], write a CUDA program to create the array C[N] such that

```
C[i] = A[i] + B[i];  // if i is even
C[i] = A[i] - B[i]; // if i is odd
```

เราจะใช้อาเรย์สามตัวในข้อนี้โดยมีข้อกำหนดว่าถ้าเป็นเลขคู่ให้เอาอาเรย์ A **บวกกับอาเรย์** B **แต่ถ้าเกิดเป็นเลขคี่ให้เราเอาอาเรย์** A - B **แทน**

```
#include <stdio.h>
#define T 16 // As Threads
#define array_size 64

__global__ void vecMultiplyReverse(int *A, int *B, int *C)
{
```

```
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i%2 == 0)
    {
        C[i] = A[i] + B[i];
    }
    else if(i%2 != 0)
    {
        C[i] = A[i] - B[i];
    }
}

int main (int argc, char *argv[])
{
    int i;
    int size = T*sizeof(int);
    int a[T],b[T],c[T], *devA,*devB,*devC;
    for (i=0; i< T; i++)
    {
        a[i] = i + 2;
        b[i] = i + 1;
    }

    cudaMalloc( (void**)&devA,size);
    cudaMalloc( (void**)&devB,size);
    cudaMalloc( (void**)&devC,size);
    cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy( devB, b, size, cudaMemcpyHostToDevice);
    cudaMemcpy( devC, c, size, cudaMemcpyHostToDevice);
    dim3 dimBlock(T);
    dim3 dimGrid(array_size/T - 1);
    vecMultiplyReverse<<<dimGrid,dimBlock>>>(devA,devB,devC);
    printf("Before A: \n");
    for (i=0; i< T; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");

    printf("Before B: \n");
    for (i=0; i< T; i++)
    {
        printf("%d ", b[i]);
    }
    printf("\n");

    cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
    cudaMemcpy(b, devB, size, cudaMemcpyDeviceToHost);
    cudaMemcpy(c, devC, size, cudaMemcpyDeviceToHost);
    cudaFree(devA);
    cudaFree(devB);
    cudaFree(devC);
    printf("After\n");
    for (i=0; i < T; i++)
    {
```

```
            printf("%d ",c[i]);
        }
        printf("\n");

    }
```

มาดูตรง Kernel ตรงนี้

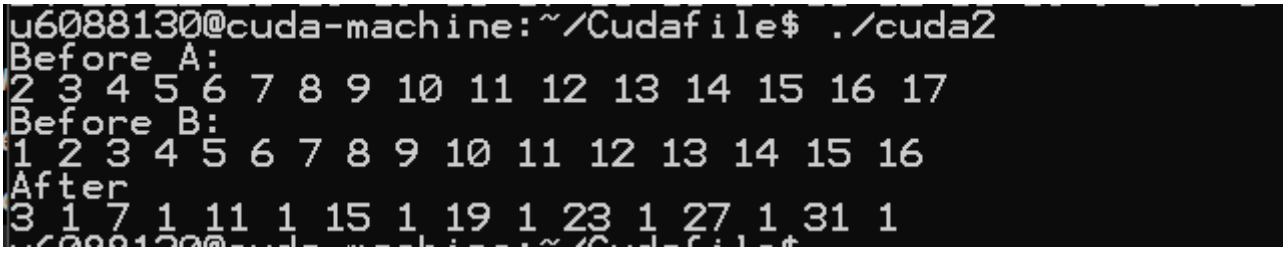
```
__global__ void vecMultiplyReverse(int *A, int *B, int *C)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i%2 == 0) // ตรงนี้คือถ้าเราหารด้วย2ลงตัวโดยไม่เหลือเศษ(Mod) จะเป็นเลขคู่
    {
        C[i] = A[i] + B[i];
    }
    else if(i%2 != 0) // ตรงนี้คือถ้าเราหารด้วย2ไม่ลงตัวโดยเหลือเศษ(Mod) จะเป็นเลขคี่
    {
        C[i] = A[i] - B[i];
    }
}
```

```
for (i=0; i< T; i++)
{
    a[i] = i + 2; //ตำแหน่งอาเรย์ A
    b[i] = i + 1; //ตำแหน่งอาเรย์ B
}
```

ที่ต้องบวกด้วยเลขที่ต่างกันนั้นเพื่อจะให้เห็นผลลัพธ์นั่นเอง ตัวอย่าง

```
Index   : 0 1 2 3 4  5  6 7   8  9
A       : 2 3 4 5 6  7  8 9  10 11
Pos|Neg : + - + - +  -  + -   +  -
B       : 1 2 3 4 5  6  7 8   9 10
Result  : 3 1 7 1 11 1 15 1  19  1
```

ผลลัพธ์ที่ได้



4. Create a CUDA program to increase the value of each element in an NxN matrix by one. That is Ai,j = Ai,j + 1, for each i,j = 0 to N-1 using multiple thread blocks. N could be any

arbitrary size.

ข้อนี้ใช้ CUDA นั่นแหละ โดยในโค้ดจะให้บวกค่าเข้าไปอีก 1 ลงในช่องอาเรย์แต่ละตัวโดยเมทริกซ์มีขนาด NxN โดยที่ อาเรย์ A ตำแหน่ง (i,j) จะเท่ากับ A(i,j) + 1 ซึ่งในแต่ละตำแหน่งของ i และ j จะเริ่มตั้งแต่ 0 ถึง N - 1 โดยใช้ Multiple Thread Block และ N มีขนาดเท่าไหร่ก็ได้

```
#include <stdio.h>
#define T 8 // As Threads
#define N 16


__global__ void vecMatrix(int *A, int *B)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int id = (i * N) + j;
    if(i < N && j < N)
    {
        B[id] = A[id] + 1;
    }
}

int main (int argc, char *argv[])
{
        int i,j;
    int size[N*N];
    int A[N][N];

    int sizearr = N*N *sizeof(int);

    int *Adefault,*B;

        for (i=0; i< N; i++)
        {
        for(j = 0 ; j<N ; j++ )
        {
            A[i][j] = ((i*i) +1) * (j+1);
            printf("%5d ", A[i][j]);
        }
    }
    printf("\n");
        cudaMalloc( (void**)&Adefault,sizearr);
    cudaMalloc( (void**)&B,sizearr);
    cudaMemcpy( Adefault, A, sizearr, cudaMemcpyHostToDevice);

    dim3 dimBlock(T,T);
    dim3 dimGrid((N+ dimBlock.x - 1)/ dimBlock.x ,(N + dimBlock.y - 1) /
dimBlock.y);
    vecMatrix<<<dimGrid,dimBlock>>>(Adefault,B);
    cudaMemcpy(size, B, sizearr, cudaMemcpyDeviceToHost);
    cudaFree(Adefault);
    cudaFree(B);
```

```
            printf("Result\n");
        for (i=0; i < N * N; i++)
        {
                    printf("%5d ",size[i]);
            }
            printf("\n");

    }
```

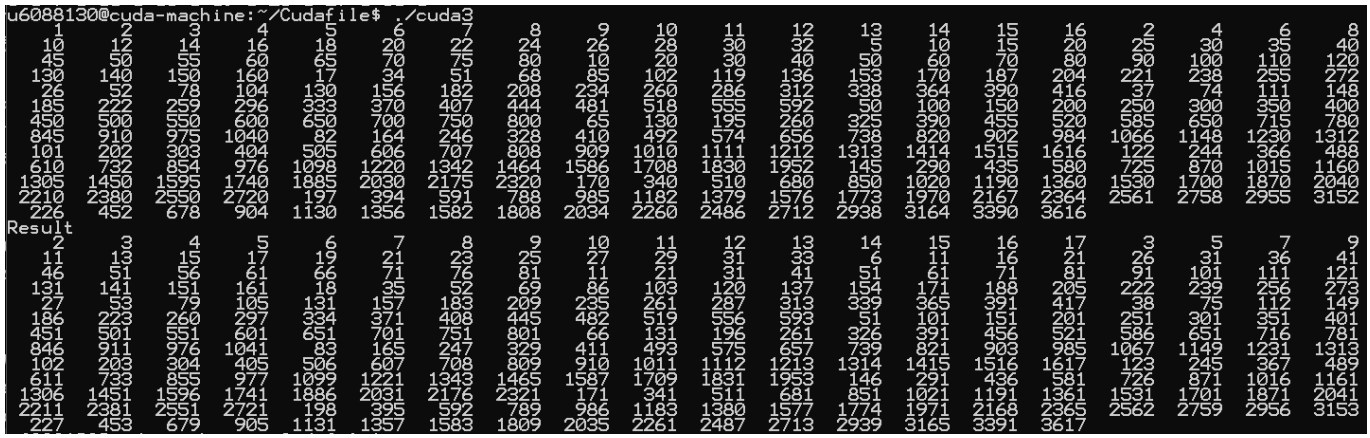
มาดูตรงนี้

```
__global__ void vecMatrix(int *A, int *B)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int id = (i * N) + j;
    if(i < N && j < N)
    {
        B[id] = A[id] + 1;
    }
}
```

ตรงนี้คือการกำหนดให้ Matrix **มีขนาด** NxN **และบวกค่าเข้าไปอีก** 1

```
    dim3 dimBlock(T,T);
    dim3 dimGrid((N+ dimBlock.x - 1)/ dimBlock.x ,(N + dimBlock.y - 1) /
dimBlock.y);
```

ตรงนี้จะทำการแบ่งขนาด Block **และเส้นกริดให้โดยอัตโนมัติ ผลลัพธ์ที่ได้จะออกมาแบบนี้**



5. Write a CUDA program to create the transpose matrix B from a square matrix A of an arbitrary size N using multiple thread blocks. Recall that if B is the transpose of A, then Bi,j = Aj,i.

โจทย์ข้อนี้จะให้เอาข้อเมื่อกี้นี้แหละมาทำทรานโพสเมทริกซ์นั่นแหละ ไม่มีอะไรมาก ไปดูโค้ดเลย

```
#include <stdio.h>
#define T 8 // As Threads
#define N 16

__global__ void vecMatrix(int *A, int *B)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y+ threadIdx.y;
    int width = gridDim.x * T;
    for( int j = 0; j<T; j+=N )
    {
        B[x*width + (j+y)] = A[(y+j)*width + x];
    }

}
int main (int argc, char *argv[])
{
        int i,j;
    int size[N*N];
    int A[N][N];

    int sizearr = N*N *sizeof(int);

    int *Adefault,*B;

        for (i=0; i< N; i++)
        {
        for(j = 0 ; j<N ; j++ )
        {
            A[i][j] = ((i*i) +1) * (j+1);
            printf("%5d ", A[i][j]);
        }
    }
    printf("\n");

        cudaMalloc( (void**)&Adefault,sizearr);
    cudaMalloc( (void**)&B,sizearr);
    cudaMemcpy( Adefault, A, sizearr, cudaMemcpyHostToDevice);

    dim3 dimBlock(T,T);
    dim3 dimGrid((N+ dimBlock.x - 1)/ dimBlock.x ,(N + dimBlock.y - 1) /
dimBlock.y);
    vecMatrix<<<dimGrid,dimBlock>>>(Adefault,B);
    cudaMemcpy(size, B, sizearr, cudaMemcpyDeviceToHost);
    cudaFree(Adefault);
    cudaFree(B);

    printf("Result\n");
    int newline = 0;

    for (i=0; i < N * N; i++)
    {
        newline++;
```

```
        printf("%3d ",size[i]);
        if(newline == N)
        {
            newline = 0;
            printf("\n");
        }
        }
        printf("\n");

}
```

ข้อแตกต่างระหว่างข้อที่ 4 กับ 5 คือ

```
// ข้อ 4.
__global__ void vecMatrix(int *A, int *B)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int id = (i * N) + j;
    if(i < N && j < N)
    {
        B[id] = A[id] + 1;
    }
}
```

และ

```
// ข้อ 5
__global__ void vecMatrix(int *A, int *B)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y+ threadIdx.y;
    int width = gridDim.x * T;
    for( int j = 0; j<T; j+=N )
    {
        B[x*width + (j+y)] = A[(y+j)*width + x];
    }

}
```

จากข้อ 4. โดยการเอา if condition **ออกแล้วใส่** for loop **เข้าไปแทนเพื่อทำการ** Transpose **ตัวเมตริกซ์นั่นเอง ผลลัพธ์จะได้แบบนี้**

```
u6088130@cuda-machine:~/Cudafile$ ./cuda4
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16    2    4    6    8
  10   12   14   16   18   20   22   24   26   28   30   32    5   10   15   20   25   30   35   40
  45   50   55   60   65   70   75   80   10   20   30   40   50   60   70   80   90  100  110  120
 130  140  150  160   17   34   51   68   85  102  119  136  153  170  187  204  221  238  255  272
  26   52   78  104  130  156  182  208  234  260  286  312  338  364  390  416   37   74  111  148
 185  222  259  296  333  370  407  444  481  518  555  592   50  100  150  200  250  300  350  400
 450  500  550  600  650  700  750  800   65  130  195  260  325  390  455  520  585  650  715  780
 845  910  975 1040   82  164  246  328  410  492  574  656  738  820  902  984 1066 1148 1230 1312
 101  202  303  404  505  606  707  808  909 1010 1111 1212 1313 1414 1515 1616  122  244  366  488
 610  732  854  976 1098 1220 1342 1464 1586 1708 1830 1952  145  290  435  580  725  870 1015 1160
1305 1450 1595 1740 1885 2030 2175 2320  170  340  510  680  850 1020 1190 1360 1530 1700 1870 2040
2210 2380 2550 2720  197  394  591  788  985 1182 1379 1576 1773 1970 2167 2364 2561 2758 2955 3152
 226  452  678  904 1130 1356 1582 1808 2034 2260 2486 2712 2938 3164 3390 3616
Result
  1   2   5  10  17  26  37  50  65  82 101 122 145 170 197 226
  2   4  10  20  34  52  74 100 130 164 202 244 290 340 394 452
  3   6  15  30  51  78 111 150 195 246 303 366 435 510 591 678
  4   8  20  40  68 104 148 200 260 328 404 488 580 680 788 904
  5  10  25  50  85 130 185 250 325 410 505 610 725 850 985 1130
  6  12  30  60 102 156 222 300 390 492 606 732 870 1020 1182 1356
  7  14  35  70 119 182 259 350 455 574 707 854 1015 1190 1379 1582
  8  16  40  80 136 208 296 400 520 656 808 976 1160 1360 1576 1808
  9  18  45  90 153 234 333 450 585 738 909 1098 1305 1530 1773 2034
 10  20  50 100 170 260 370 500 650 820 1010 1220 1450 1700 1970 2260
 11  22  55 110 187 286 407 550 715 902 1111 1342 1595 1870 2167 2486
 12  24  60 120 204 312 444 600 780 984 1212 1464 1740 2040 2364 2712
 13  26  65 130 221 338 481 650 845 1066 1313 1586 1885 2210 2561 2938
 14  28  70 140 238 364 518 700 910 1148 1414 1708 2030 2380 2758 3164
 15  30  75 150 255 390 555 750 975 1230 1515 1830 2175 2550 2955 3390
 16  32  80 160 272 416 592 800 1040 1312 1616 1952 2320 2720 3152 3616
```

อะไรคือการ Transpose

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^{\mathrm{T}}$$

Transpose **คือการเปลี่ยนข้อมูลในเมทริกซ์จาก** Row **เป็น** Column **หรือ** Column **เป็น** Row **ประมาณนี้**