EECS2040 Data Structure Hw #1 (Chapter 1, 2 of textbook)
due date 3/21/2022
by 108011235 陳昭維

Part 1 (2% of final Grade)

<Question>

1. (10%) Using the ADT1.1 *NaturalNumber* in the textbook pp.10, add the following operations to the *NaturalNumber* ADT: Predecessor, *IsGreater*, *Multiply*, *Divide*.

<Answer>

```
ADT NaturalNumber is
  object: An ordered subrange of the integers starting at zero and ending at the maximun integer
  (MAXINT) on the computer.
  functions:
    for all x, y ∈ NaturalNumber; TRUE, FALSE ∈ Boolean
    and where +, −, ×, ÷, modulo, <, >, == and = are the usual integer operations

    Zero( ) : NaturalNumber             ::=   0
    IsZero( ) : Boolean                 ::=   if (x==0) IsZero = TRUE
                                              else IsZero = FALSE
    Add(x, y) : NaturalNumber           ::=   if (x+y<=MAXINT) Add = x+y
                                              else Add = MAXINT
    Equal(x, y) : Boolean               ::=   if (x==y) Equal = TRUE
                                              else Equal = FALSE
    Successor(x) : NaturalNumber        ::=   if (x==MAXINT) Successor = x
                                              else Successor = x+1
    Subtract(x, y) : NaturalNumber      ::=   if (x<y) Subtract = 0
                                              else Subtract = x-y
    Predecessor(x) : NaturalNumber      ::=   if (x==0) Predecessor = 0
                                              else Predecessor = x-1
    IsGreater(x, y) : Boolean           ::=   if (x>y) IsGreater = TRUE
                                              else IsGreater = FALSE
    Multiply(x, y) : NaturalNumber      ::=   if (x×y<=MAXINT)
                                                 Multiply = x×y
                                              else Multiply = MAXINT
    Divide(x, y) : NaturalNumber        ::=   if (y==0) Divide = MAXINT
                                              else Divide = (x-(x modulo(y)))÷y
 end NaturalNumber
```

<Question>

2. (10%) Determine the frequency counts for all statements (by step table) in the following two program segments:

<Answer>

Code (a):

```
1 for(i=1; i<=n; i++)
2     for(j=1; j<=i; j++)
3         for(k=1; k<=j; k++)
4             x++;
```

Line	s/e	freq	subtotal
1	1	n + 1	n + 1
2	1	$\frac{(n+3)n}{2}$	$\frac{n^2+3n}{2}$
3	1	$\frac{n^3+6n^2+5n}{6}$	$\frac{n^3+6n^2+5n}{6}$
4	1	$\frac{n(n+1)(n+2)}{6}$	$\frac{n^3+3n^2+2n}{6}$
Total step			$\frac{n^3+6n^2+11n+3}{3}$

Code (b)

```
1 i=1;
2 while(i<=n)
3 {
4     x++;
5     i++;
6 }
```

Line	s/e	freq	subtotal
1	1	1	1
2	1	n + 1	n + 1
3	0	n	0
4	1	n	n
5	1	n	n
6	0	n	0
Total step			3n + 2

<Question>

3. (10%) For the function Multiply() shown below,

(a) Introduce statements to increment count at all appropriate points and compute the count

(b) Simplify the resulting program by eliminating statement and compute the count

(c) Obtain the step count for the function using the step table method.

(d) Repeat (c) but use Θ() notation instead of exact step counts and frequency counts..

```
void Multiply(int **a, int **b, int **c, int m, int n, int p)
{
        for(int i=0;i<m;i++)
            for(int j=0; j<p; j++)
            {
                    c[i][j] = 0;
                    for(int k=0;k<n;k++)
                        c[i][j] += a[i][k] * b[k][j];
            }
}
```

<Answer>

(a)

```
void Multiply(int **a, int **b, int **c, int m, int n, int p)
{
        for(int i=0;i<m;i++) {
            count++;
            for(int j=0; j<p; j++)
            {
                    count++;
                    c[i][j] = 0;
                    count++;
                    for(int k=0;k<n;k++) {
                        count++;
                        c[i][j] += a[i][k] * b[k][j];
                        count++;
                }
                count++;
            }
            count++;
    }
    count++;
}
```

(b)

```
void Multiply (int **a, int **b, int **c, int m, int n, int p)
{
        for(int i=0;i<m;i++) {
            count+=2;
            for(int j=0; j<p; j++)
            {
                    count+=3;
                    for(int k=0;k<n;k++) {
                        count+=2;
                }
            }
    }
    count++;
}
```

(c)

```
void Multiply(int **a, int **b, int **c, int m, int n, int p)
{
        for(int i=0;i<m;i++)
            for(int j=0; j<p; j++)
            {
                    c[i][j] = 0;
                    for(int k=0;k<n;k++)
                        c[i][j] += a[i][k] * b[k][j];
            }
}
```

line	s/e	freq	Subtotal
1	0	1	0
2	0	1	0
3	1	m + 1	m + 1
4	1	mp + m	mp + m
5	0	mp	0
6	1	mp	mp
7	1	mpn + mp	mpn + mp
8	1	mnp	mnp
9	0	mp	0
10	0	1	0
Total step			2mpn + 3mp + 2m + 1

(d)

```
void Multiply(int **a, int **b, int **c, int m, int n, int p)
{
        for(int i=0;i<m;i++)
            for(int j=0; j<p; j++)
            {
                    c[i][j] = 0;
                    for(int k=0;k<n;k++)
                        c[i][j] += a[i][k] * b[k][j];
            }
}
```

Line	s/e	freq	Subtotal
1	0	θ(1)	0
2	0	θ(1)	0
3	1	θ(m)	θ(m)
4	1	θ(mp)	θ(mp)
5	0	θ(mp)	0
6	1	θ(mp)	θ(mp)
7	1	θ(mpn)	θ(mpn)
8	1	θ(mpn)	θ(mpn)
9	0	θ(mp)	0
10	0	θ(1)	0
overall			θ(mpn)

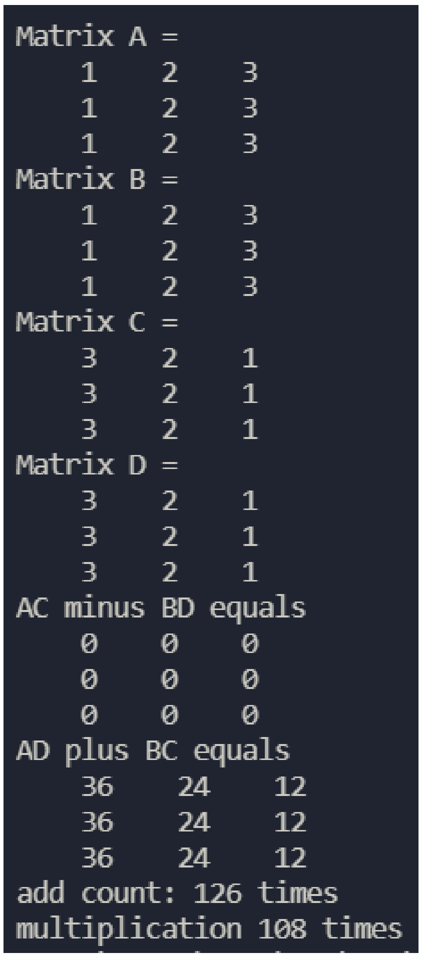
$$\Theta(f(n)) = \Theta(m) \times \Theta(p) \times (\Theta(1) + \Theta(n) \times \Theta(1)) = \Theta(m) \times \Theta(p) \times \Theta(n) = \Theta(mpn)$$

<Question>

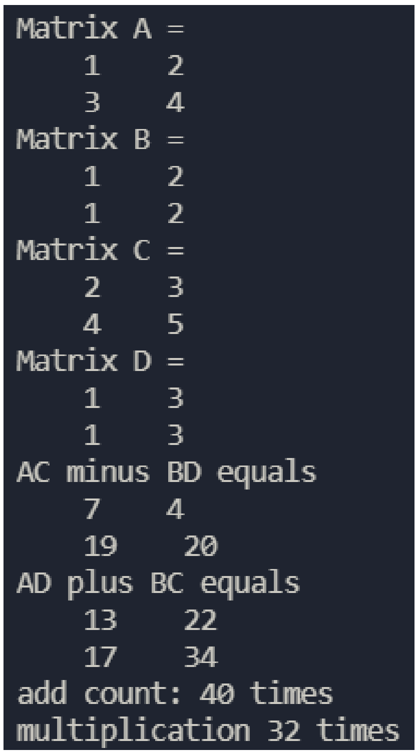
4. (10%) A complex-valued matrix X is represented by a pair of matrices (A, B) where A and B contains real values.
(a) Write a C++ program that computes the product of two complex-valued matrices (A, B) and (C, D), where (A, B) * (C, D) = (A+iB)*(C+iD) = (AC-BD) + i(AD + BC).
(b) Determine the number of additions and multiplications if the matrices are all nxn.

<Answer>

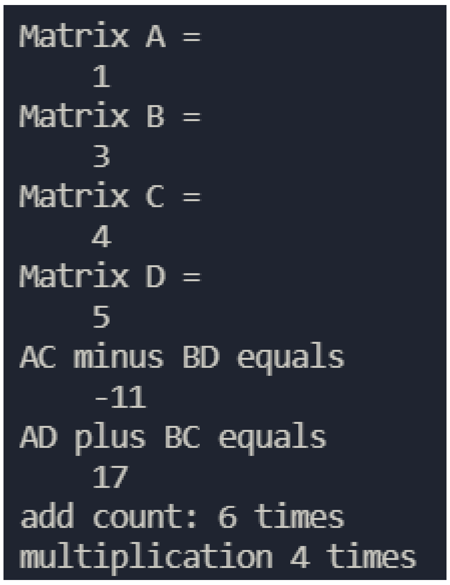
(The code is attached in elearn with pdf file)



```
Matrix A =
    1    2    3
    1    2    3
    1    2    3
Matrix B =
    1    2    3
    1    2    3
    1    2    3
Matrix C =
    3    2    1
    3    2    1
    3    2    1
Matrix D =
    3    2    1
    3    2    1
    3    2    1
AC minus BD equals
    0    0    0
    0    0    0
    0    0    0
AD plus BC equals
    36     24     12
    36     24     12
    36     24     12
add count: 126 times
multiplication 108 times
```



```
Matrix A =
    1    2
    3    4
Matrix B =
    1    2
    1    2
Matrix C =
    2    3
    4    5
Matrix D =
    1    3
    1    3
AC minus BD equals
    7    4
    19     20
AD plus BC equals
    13     22
    17     34
add count: 40 times
multiplication 32 times
```



```
Matrix A =
    1
Matrix B =
    3
Matrix C =
    4
Matrix D =
    5
AC minus BD equals
    -11
AD plus BC equals
    17
add count: 6 times
multiplication 4 times
```

For every (AC-BD) + i(AD + BC) to be performed, four matrix multiplication has to be performed to generate AC, BD, AD, BC term, and then two matrix addition has to be performed to generate the (AC – BD), (AD + BC) term.

For my code, each matrix multiplication performs n^3 times of entries multiplication and entries addition, and each matrix addition performs n^2 times of entries addition.

So, for my total code, there is $2n^2 + 4n^3$ entries addition and there is $4n^3$ entries multiplication to perform.

For N = 3, $2n^2 + 4n^3 = 18 + 108 = 126$ addition and $4n^3 = 108$ multiplications.
For N = 2, $2n^2 + 4n^3 = 8 + 32 = 40$ additions and $4n^3 = 32$ multiplications.
For N = 1, $2n^2 + 4n^3 = 2 + 4 = 6$ additions and $4n^3 = 4$ multiplications.

<Question>

5. (10%) The Tower of Hanoi is a classical problem which can be solved by recurrence. There are three pegs and N disks of different sizes. Originally, all the disks are on the left peg, stacked in decreasing size from bottom to top. Our goal is to transfer all the disks to the right peg, and the rules are that we can only move one disk at a time, and no disk can be moved onto a smaller one. We can easily solve this problem with the following recursive method: If N = 1, move this disk directly to the right peg and we are done. Otherwise (N >1), first transfer the top N − 1 disks to the middle peg applying the method recursively, then move the largest disk to the right peg, and finally transfer the N −1 disks on the middle peg to the right peg applying the method recursively. Let T(N) be the total number of moves needed to transfer N disks.

(a) Prove that T(N) = 2T(N −1) + 1 with T(1) = 1.

(b) Unfold this recurrence relation to obtain a closed-form expression for T(N). (T(N) is expressed in terms of function of N.)

<Answer>

(a)

Denote the left, middle, right peg as L, M, R and consider all N pegs are at the L peg initially.
For N = 1, it takes 1 step to move the only disk from peg L to peg R. This is our fundamental case, T(1) =1.

Then consider there now is N disks, we can decompose the problem into three parts.
(1) We can move the upper N-1 disks to peg M, it takes T(N-1) steps to do so.
(2) Then we can move the bottom disk remaining in peg L to the peg R, this takes T (1) = 1 step to do so.
(3) Then we move the N-1 disks to the peg R, this then again takes T(N-1) steps.

In conclusion, we can move N disks from peg L to peg R in T(N-1) + 1+ T(N-1) = 2 T(N-1) + 1 steps.

(b)

$$
\begin{gathered}
\mathrm{T(N)} = 2T(N-1)+1 \\
= 2(2T(N-2)+1)+1 \\
= 2^2(2(T(N-3)+1)+3 \\
= \cdots \\
= 2^kT(N-k)+2^{k-1}+2^{k-1}+\cdots+2^1+1 \\
= 2^{N-1}T(1)+2^{N-2}+\cdots+2^1+1 \\
= 2^N-1
\end{gathered}
$$

<Question>

6. (10%) For the polynomial represented by dynamic array of tuples,

(a) Design an algorithm for performing multiplication of two polynomials.
(b) Then analyze its time complexity using Θ() notation, assuming the number of terms of the two polynomials are m and n, respectively.

<Answer>

Similar to the textbook, we assume the terms for each polynomial is ordered in exponent descending order, First, we initialize an array with size of the sum of both highest exponents.

```
int array[termArray[0].exp + b.termArray[0].exp] = {0};
```

This is done in the situation of the previous assumption with descending exponent order.

Then use the array as a map or dictionary, where the index is the key for exponent, and the value is the coefficient, and then use 2 for loop with range of first polynomial term numbers and the second polynomial term numbers respectively.

```
for(int pos1 = 0; pos1 < terms; pos1++)
        for(int pos2 = 0; pos2 < b.terms; pos2++)
                array[termArray[pos1].exp+b.termArray[pos2].exp – 1] += (termArray[pos1].coef *
b.termArray[pos2].coef);
```

Then initialize a polynomial c as the return polynomial.

```
Polynomial C;
```

Then again we use a for loop starting from the last index of the mapping array to the first index, if the value in the array is not zero then create new term using *NewTerm* function with (coefficient = array[index], exponent = index + 1).

```
for(int index = (termArray[0].exp+b.termArray[0].exp – 1); index >= 0; index--) {
        If(Array[index] != 0)
                c.NewTerm(array[index], index+1);
}
```

Then we return the polynomial C.

```
return c;
```

First the time complex complexity for the first for loop is $\theta(m \times n)$

```
for(int pos1 = 0; pos1 < terms; pos1++)
        for(int pos2 = 0; pos2 < b.terms; pos2++)
                array[termArray[pos1].exp+b.termArray[pos2].exp – 1] += (termArray[pos1].coef *
b.termArray[pos2].coef);
```

Then the time complexity of the second loop is $\theta(m + n)$

```
for(int index = (termArray[0].exp+b.termArray[0].exp – 1); index >= 0; index--) {
        If(Array[i] != 0)
                c.NewTerm(array[index], index+1);
}
```

The return time complexity of the second loop is $\theta(1)$

```
return c;
```

We have to take doubling array into consideration, the maximal number of term is $m \times n$ terms thus the time spent on doubling the array is $\theta(m \times n)$

The total time complexity is $\theta(m \times n) + \theta(m + n) + \theta(m \times n) + \theta(1) = \theta(m \times n)$

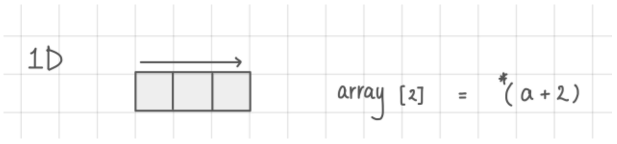
<Question>

7. (10%) Obtain an addressing formula for the element a[i1][i2]…[in] in an array declared as a[u1][u2]…[un]. Assume a column-major representation of the array with one word per element and a the address of a[0][0]…[0]. In a column-major representation, a two-dimensional array is stored sequentially by columns rather than by rows.

<Answer>

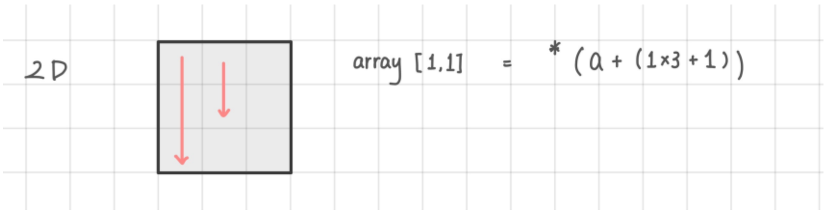
For 1D -array if I want to access the element a[i1] then I can get it by the below address formula

$$\mathrm{a} + \text{memory unit} \times i_1$$



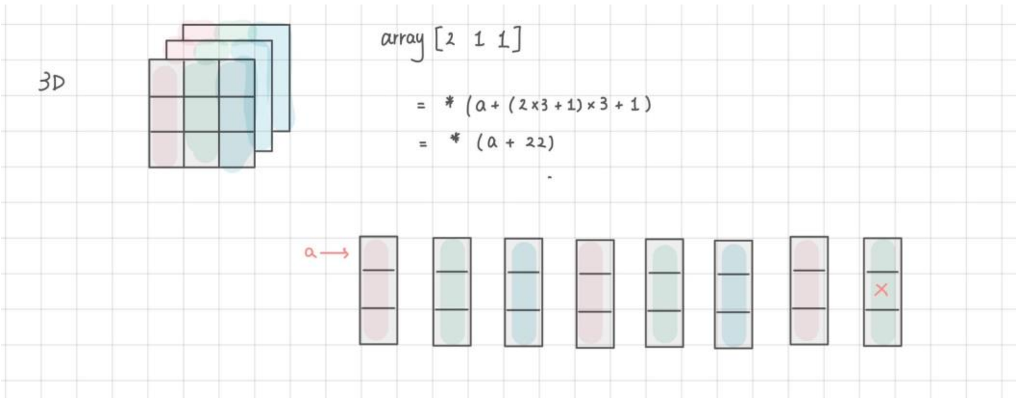
For 2D-array if I want to access the element a[i1][i2] then I can get it by the below address formula

$$\mathrm{a} + \text{memory unit } \times ((i_1) \times u_2 + i_2)$$



For 3D-array if I want to access the element a[i1][i2][i3] then I can get it by the below address formula

$$\mathrm{a} + \text{memory unit } \times ((u_2 i_1 + i_2) \times u_3 + i_3)$$



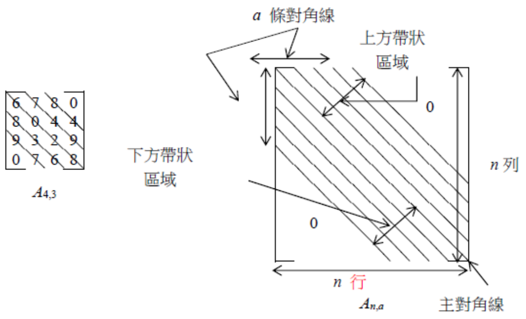
So a recursive pattern can be observed that if we want to access the element a[i1][i2]…[in], then we can get the address by formula:

$$\mathrm{a} + \text{memory unit } \times ((u_2 i_1 + i_2) \times u_3 + \cdots i_{n-2}) u_{n-1} + i_{n-1}) u_n + i_n)$$

$$= \mathrm{a} + \text{memoru unit } \times \sum_{j=1}^{n} i_j\, a_j \text{ where } \begin{cases} a_j = \prod_{k=j+1}^{n} u_k \quad 1 \le j < n \\ a_n = 1 \end{cases}$$

<Question>

8. (15%) A square band matrix $A_{n,a}$ is an n x n matrix A in which all the nonzero terms lie in a band centered around the main diagonal. The band includes a-1 diagonals below and above the main diagonal as shown below.



(a) How many elements are there in the band of $A_{n,a}$?

(b) What is the relationship between i and j for element A_{ij} in the band of $A_{n,a}$?

(c) Assume that the band of An,a is stored sequentially in an array b by diagonals starting with the lowermost diagonal. Thus $A_{4,3}$ above would have the following representation:

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]	b[10]	b[11]	b[12]	b[13]
9	7	8	3	6	6	0	2	8	7	4	9	8	4
A_{20}	A_{31}	A_{10}	A_{21}	A_{32}	A_{00}	A_{11}	A_{22}	A_{33}	A_{01}	A_{12}	A_{23}	A_{02}	A_{13}

Obtain an addressing formula for the location of an element Aij in the lower band of An,a, e.g., LOC(A20) = 0, LOC(A31) = 1 in the example above.

<Answer>

(a) For $n \times n$ matrix, there are $2n - 1$ diagonals, so for $A_{n,a}$ there are $(2n-1)-(2a-1)$ zero diagonals, so elements number in the band is $n^2 - 2\sum_{i=1}^{n-a} i = n^2 - (n-a+1)(n-a)$

$$= 2an - a^2 + a - n$$

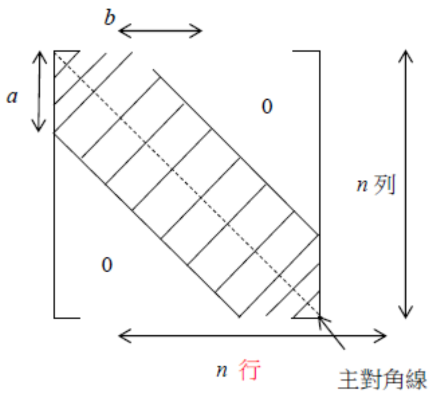
(b) The **absolute value of the difference between j and i** has to be smaller than **a** to be inside the band. And **for j > i,** the element is in the upper band, for **i < j**, the element is in the lower band and for **i = j** is in the midband.

(c)

$$\mathrm{LOC}(A_{ij}) = \begin{cases} \sum_{k=n-a+1}^{n-i+j-1} k \quad + j\ modulo(n-i+j) & for\ i \geq j\ and\ (i-j) < a \\ \sum_{k=n-a+1}^{n} k + \sum_{k=n-j+i+1}^{n-1} k + i\ modulo(n-j+i) & for\ i < j\ and\ (j-i) < a \\ -1 & for\ |i-j| \geq a \end{cases}$$

<Question>

9. (15%) A generalized band matrix $A_{n,a,b}$ is an n x n matrix A in which all the nonzero terms lie in a band made up of a-1 diagonals below the main diagonal, the main diagonal, and b-1 above the main diagonal as shown below.



An, a, b

(a) How many elements are there in the band of $A_{n,a,b}$?

(b) What is the relationship between i and j for element A_{ij} in the band of $A_{n,a,b}$?

(c) Obtain a sequential representation of the band of $A_{n,a,b}$ in one-dimensional array c.

<Answer>

(a) For $n \times n$ matrix, there are $\sum_{i=n-a+1}^{n-1} i + \sum_{i=n-b+1}^{n-1} i + n = \frac{(a+b)(2n+1)-a^2-b^2-2n}{2}$ element in the band.

(b) If $i < j$ then it is in the upper band if and only if $(j-i) < b$, if $i > j$ then it is in the lower band if and only if $(i-j) < a$.

(c)

$$\mathrm{LOC}(A_{ij}) = \begin{cases} \sum_{k=n-a+1}^{n-i+j-1} k \quad + j\ modulo(n-i+j) & for\ i \geq j\ and\ (i-j) < a \\ \sum_{k=n-a+1}^{n} k + \sum_{k=n-j+i+1}^{n-1} k + i\ modulo(n-j+i) & for\ i < j\ and\ (j-i) < b \\ -1 & for\ A_{ij}\ not\ in\ the\ band \end{cases}$$