**EECS2040 Data Structure Hw #1 (Chapter 1, 2 of textbook) due date 4/8/2021 by S108061217, 鍾永桓**

**Part 1 (40% of Hw1)**

**1(20%) Using the ADT1.1 *NaturalNumber* in the textbook pp.10, add the following operations to the *NaturalNumber* ADT: Predecessor, *IsGreater*, *Multiply*, *Divide*.**

1.ANS

Predecessor(x):NaturalNumber :: = if(x == 0) Predecessor = 0;

else Predecessor = x-1

IsGreater(x,y): NaturalNumber :: =if(x>y)IsGreater = true

else IsGreater = false

Multiple(x,y): NaturalNumber :: =if(x\*y<=MAXINT)Multiple=x\*y

else Multiple = MANINT

Divide(x,y):NaturalNumber ::=Divide = x/y

2(20%) Determine the frequency counts for all statements (by step table) in the following two program segments:

code (a): code (b)

1. for(i=1;i<=n;i++) 1 i=1;
2. for(j=1;j<=I;j++) 2 while(i<=n)
3. for(k=1;k<=j;k++) 3 {
4. x++; 4 x++;

5 i++;

6 }

2.ANS

（ａ） s/e freq subtotal

for(int i=1;i<=n;i++) 1 n+1 n+1

for(j=1;j<=i;j++) 1 (n+3)\*n/2 (n+3)\*n/2

for(k=1;k<=j;k++) 1 n(n+1)(n+2)/6 n(n+1)(n+2)/6

x++ ; total 1/3\*n^3+2\*n^2+11/3\*ｎ＋１

（ｂ） s/e freq subtotal

i=1; 1 1 1

while(i<=n) 1 n+1 1

{ 0

x++; 1 n n

i++; 1 n n

} total: 3n+2

3(20%) For the function Multiply() shown below,

1. Introduce statements to increment count at all appropriate points and compute the count
2. Simplify the resulting program by eliminating statement and compute the count
3. Obtain the step count for the function using the frequency method.

3.ANS

(a)

void Multiply(int \*\*a,int \*\*b, int \*\*c, int m, int n, int p)

{

int count = 0;

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)

{

count=0;

for(int i=0;i<m;i++)

{

count+=2;

for(int j=0; j<p; j++)

{

count+=3;

c[i][j] = 0;

for(int k=0;k<n;k++)

{

count+=2;

c[i][j] += a[i][k] \* b[k][j];

}

}

}

count++;

}

(c)

2\*m\*n\*p+3\*m\*p+2\*m+1

4(20%) A complex-valued matrix X is represented by a pair of matrices (A, B) where A and B contains real values. Write a 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). Determine the number of additions and multiplications if the matrices are all nxn.

4.ANS

additions: 4\*n^3-2\*n^2

multiplications:4\*n^3

5(20%) 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.)

5.ANS

1. To move all disks to the right peg, we need to move the largest disk from the left peg to the right peg. However, we cannot move the largest onto others. Therefore, we have to move other disks to the middle peg which needs at least T(n-1) moves. Then we move the largest to the right peg and it needs only one move, Finally, we move the n-1 disks from the middle to the right, it needs T(n-1) moves. The total number of moves is 2\*T(n-1)+1
2. T(1) = 1

T(2) = 2\*T(1)+1

T(3) = 2\*T(2)+1=4\*T(1)+3

T(n) =2\*T(n-1)+1=….=2^(n-1)\*T(1)+2^(n-1)-1=2^n-1

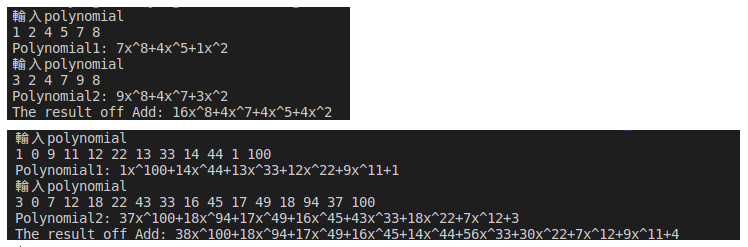
**Part 2 Coding (60% of Hw1)**

**execution trace**

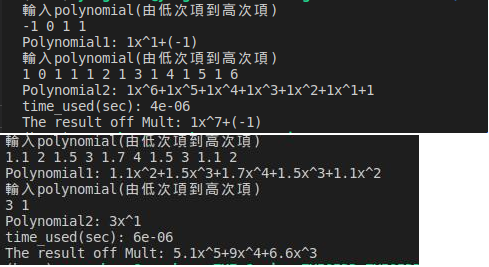
(1)

By the source code,the execution time of Mult :4\*b.terms\*terms+1(b.terms and terms are the number of terms of these two Polynomial ),so we can estimate the time complexity of it to be Θ(b.terms\*terms). The execution of Eval : terms, so we can estimate th e time complexity of it to be Θ(terms).

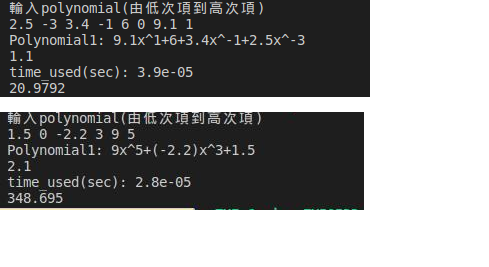
Add:



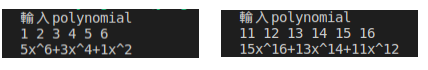
Mult:



Eval:

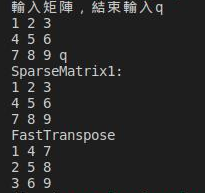
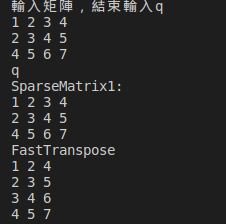


Input(>>) and Output(<<):

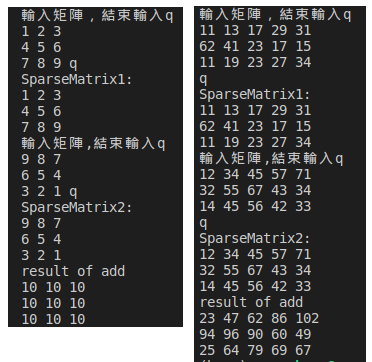


(2)

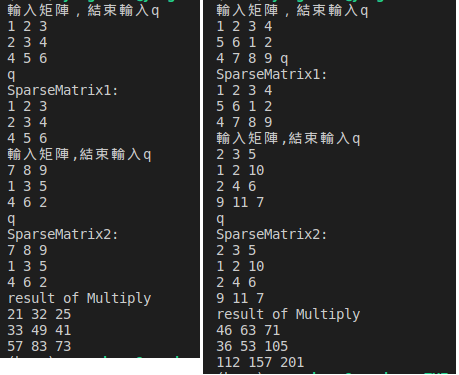
FastTranspose:



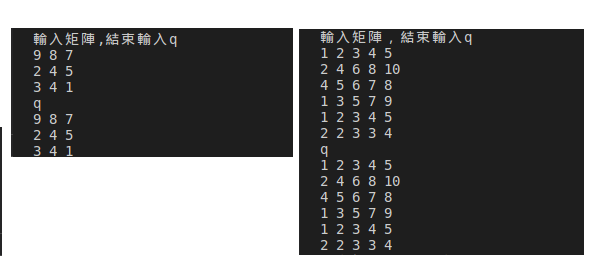
Add:



Mult:



Input(>>) and Output(<<):

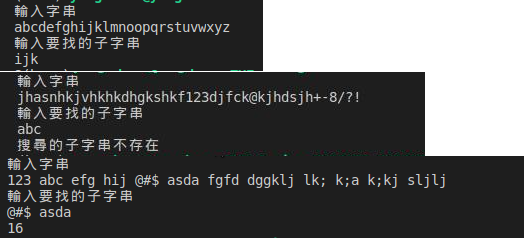


(3)

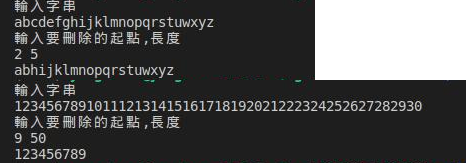
FailureFunction:



FastFind:



Delete:



CharDelete:

