FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110327 ALGORITHM DESIGN

Year II, Second Semester, Midterm Examination, March 6, 2020 13:00-16:00

ชื่อ-นามสกุล.....เลขประจำตัว.....ตอนเรียนที่.....เลขที่ใน CR58.....

<u>หมายเหตุ</u>

1. ข้อสอบมีทั้งหมด 9 ข้อ ในกระดาษคำถามคำตอบ 7 หน้า
2. **ไม่อนุญาตให้นำตำราและเอกสารใดๆ เข้าในห้องสอบ**
3. **ไม่อนุญาตให้ใช้เครื่องคำนวณใดๆ**
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่เจ้าหน้าที่ควบคุมการสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบและสมุดคำตอบออกจากห้องสอบ
6. ผู้เข้าสอบสามารถออกจากห้องสอบได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. **นิสิตกระทำผิดเกี่ยวกับการสอบ ตามข้อบังคับจุฬาลงกรณ์มหาวิทยาลัย มีโทษ คือ พ้นสภาพการเป็นนิสิต หรือ ได้รับ สัญลักษณ์ F ในรายวิชาที่กระทำผิด และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้**

ห้ามนิสิตพกโทรศัพท์และอุปกรณ์สื่อสารไว้กับตัวระหว่างสอบ หากตรวจพบจะถือว่านิสิตกระทำผิดเกี่ยวกับการสอบ อาจต้องพ้นสภาพการเป็นนิสิต หรือ ให้ได้รับ F และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้

*** ร่วมรณรงค์การไม่กระทำผิดและไม่ทุจริตการสอบที่คณะวิศวกรรมศาสตร์ ***

ข้าพเจ้ายอมรับในข้อกำหนดที่กล่าวมานี้ ข้าพเจ้าเป็นผู้ทำข้อสอบนี้ด้วยตนเองโดยมิได้รับการช่วยเหลือหรือให้ความช่วยเหลือ ในการทำข้อสอบนี้

ลงชื่อนิสิต.....

วันที่.....

1. (10 คะแนน) จงวิเคราะห์ประสิทธิภาพเชิงเวลาของขั้นตอนวิธีแต่ละข้อดังต่อไปนี้

สำหรับบางข้อที่สามารถใช้ Mater Theorem ได้ กำหนดให้ Master Theorem มีนิยามคือ

สำหรับ Recurrence relation $T(n) = aT(n/b) + f(n)$ เมื่อ $a \geq 1$ และ $b > 1$ เป็นค่างคงที่ และ $f(n)$ เป็นค่าฟังก์ชันที่มีค่าเป็นบวกเสมอ เราสามารถคำนวณ $T(n)$ ได้ตามกรณีต่าง ๆ ดังต่อไปนี้

$T(n) = \Theta(n^{\log_b(a)})$	ถ้า $f(n) = O(n^{\log b(a-\varepsilon)})$
$T(n) = \Theta(n^{\log_b(a)} \log^{k+1} n)$	ถ้า $f(n) = \Theta(n^{\log b(a)} \log^{k}(n))$ และ $k \geq 0$
$T(n) = \Theta(f(n))$	ถ้า $f(n) = \Omega(n^{\log b(a+\varepsilon)})$ และ $\varepsilon > 0$ และ $af(n/b) \leq kf(n)$ เมื่อมีค่าคงที่ $k < 1$ และ n ขนาดใหญ่มาก

และให้ถือว่า $T(1) = 1$ เสมอ

ข้อ		Big O หรือ **Θ** (...)

1.	`sum = 0`
	`for(int i=n; i>0; i/=2) {`
	`  for(int j=1; j<n; j*=2) {`
	`  for(int k=0; k<n; k+=2) {`
	`  sum++;`
	`  }`
	`  }`
	`}`
		Θ(n log²n)

2.	`void c2110327() {`
	`  for (int i=1; i<=n; i++)`
	`  for (int j=1; j<=log(i); j++)`
	`  printf("2110327");`
	`}`
		Θ(n log n)

3.	`A(n) {`
	`  if(n<=1) return 1;`
	`  else return A(√n); // √n is square root of n`
	`}`
		O(log log n)

4.	`for( i = 0; i < n; i++ ) {`
	`  for( j = 0; j < n; j++ )`
	`  a[ j ] = randomValue( i );  // randomValue is O(1)`
	`  goodSort( a );  // goodSort is O(n log n)`
	`}`
		O(n² log n)

5.	**การเรียงลำดับข้อมูลของ Quicksort เมื่อใช้ pivot เป็นค่า mean ที่ใช้เวลาหา O(n)**	O(n log n)
6.	`T(n) = T(n-1) + O(1)`	Θ(n)
7.	`T(n)=3T(n/2) + n`	Θ(n^log₂3)
8.	`T(n) = 64T(n/8) + n²log n`	Θ(n² log²n)
9.	`T(n) = T(n/2) + n^(2 - cos n)`	Θ(n^(2 - cos n))
10.	`T(n) = 16T(n/4)+n!`	Θ(n!)

2. (5 คะแนน) สำหรับปัญหา Matrix Chain Multiplication กำหนดให้เรากำลังหาจำนวนครั้งในการคูณน้อยสุดการหาผลการคูณของ matrix จำนวน 6 matrix คือ A1, A2, ... A6 เข้าด้วยกัน โดยที่มีตารางค่าคำตอบ M ดังต่อไปนี้ (กำหนดให้ M[i][j] คือ จำนวนครั้งการคูณน้อยสุดของการคูณ matrix Ai ถึง matrix Aj จงระบุวิธีการใส่วงเล็บที่ทำให้ได้จำนวนครั้งการคูณน้อยสุด

	j=1	j=2	j=3	j=4	j=5	j=6
i = 1	0	750	200	230	370	720
i = 2		0	50	70	170	470
i = 3			0	10	70	320
i = 4				0	20	220
i = 5					0	400
i = 6						0

กำหนดให้ขนาดของ แต่ละ matrix เป็นดังนี้

Matrix	A1	A2	A3	A4	A5	A6
จำนวนแถว	15	10	5	1	2	10
จำนวนคอลัมน์	10	5	1	2	10	20

การคูณ matrix เหล่านี้ทำได้โดยใช้จำนวนครั้งในการคูณเลขน้อยสุดด้วยการใส่วงเล็บดังนี้ (A1 (A2 A3)) ((A4 A5) A6)

(ให้ตอบโดยการใส่วงเล็บ ตัวอย่างเช่น ((A1A2)((A3(A4A5))))

3. (5 คะแนน) ปัญหาการทอนเหรียญเป็นดังนี้ เราต้องการจ่ายเงิน n บาท ด้วยเหรียญต่าง ๆ ให้ใช้จำนวนเหรียญน้อยที่สุด เรามีเหรียญอยู่ k แบบ แต่ละแบบมีค่าแตกต่างกันคือ v[1..k] โดยเรามีเหรียญแต่ละเหรียญไม่จำกัด และรับประกันว่า v[1] = 1 เสมอ เช่น ถ้า n = 11 และ v = [1,3,4,6] เราจะใช้สามเหรียญในการจ่ายเงิน 11 บาท (คือเหรียญ 4 + 4 + 3) กำหนดให้ C(a, b) คือจำนวนเหรียญน้อยสุดที่ใช้ในการทอนเงิน b บาท เมื่อพิจารณาเฉพาะเหรียญ v[1..a] เราจะได้ว่า C(a, b) มี recurrence relation ดังนี้

C(a, b) = b ; if (a == 1 || b == 0)

C(a, b) = C(a-1, b) ; if (a > 1 && b < v[a])

C(a, b) = min(C(a, b – v[a]) + 1, C(a-1, b)) ; if (a > 1 && b >= v[a])

จงเติมตารางสำหรับค่า C(a, b) ด้านล่างนี้ เมื่อกำหนดให้ v มีค่าดังนี้ v = [1,3,4,7,12]

a\b	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1	2	1	2	3	2	3	4	3	4	5	4	5	6	5	6	7	6
3	1	2	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5
4	1	2	1	1	2	2	1	2	3	2	2	3	3	2	3	4	3	3
5	1	2	1	1	2	2	1	2	3	2	2	1	2	2	2	2	3	3

4. (5 คะแนน) จาก Recurrence Relation ต่อไปนี้ จงระบุว่า หากเราเขียนโปรแกรมเพื่อคำนวณ Relation ดังกล่าวด้วยวิธีการแบบ Dynamic Programming แล้ว เราจะต้องใช้ตารางขนาดกี่มิติในการเก็บข้อมูล และ การคำนวณนั้นทำได้เร็วสุดมีประสิทธิภาพเชิงเวลาเป็นเท่าไร

Recurrence Relation	ขนาดของตาราง (มิติ)	ประสิทธิภาพเชิงเวลา (ตอบเป็น O หรือ θ)
(ตัวอย่าง) F(n) = F(n-1) + F(n-2)	1	O(n)
B(n, k) = k * B(n-1, k) + B(n-1,k-1)	2	O(nk)
C(n) = max(C(1), C(2), ..., C(n-1))	1	θ(n)
D(n,b,c) = D(n-1,b-1,c-1) + D(n-1,b-1,c+1) + D(n-1,b+1,c+1) + D(n-1,b+1,c-1)	3	O(nbc)
E(n) = E(n/2) + a[n] // ให้ a[n] เป็นอาเรย์ 1 มิติ	1	O(log n)
G(n, k) = max(G(n-1, 1) + a[1], G(n-1, 2) + a[2], ..., G(n-1,k-1) + a[k-1])	2	O(nk²)

5. (10 คะแนน) ปัญหาเชิญคนมางานปาร์ตี้ เป็นดังนี้ มีแขกทั้งหมด n คน กำกับด้วยหมายเลข 1 ถึง n เราต้องการเชิญคนเหล่านี้มางานปาร์ตี้ อย่างไรก็ตาม แขกบางคนไม่ถูกกัน ให้ dislike(a, b) เป็นฟังก์ชันที่คืนค่า true เมื่อแขก a และ b นั้นเกลียดกัน แขกที่เราเชิญมา จะต้องไม่มีคู่ไหนเลยที่เกลียดกัน

5.1. จงเขียน Recurrence relation สำหรับการคำนวณหาจำนวนคนมากที่สุดที่เราสามารถเชิญมางานปาร์ตี้ได้ กำหนดให้ MaxGuest(i, S) คือ จำนวนคนที่สามารถเชิญมาร่วมงานได้มากที่สุด เมื่อเราพิจารณาเฉพาะคนหมายเลขที่ 1 ถึง i โดยที่ S คือคนหมายเลย i+1 ถึง n ที่เราเลือกเชิญมาก่อนแล้ว และ S ไม่มีคนที่เกลียดกันเลย ซึ่งคำตอบที่เราต้องการคือ MaxGuest(n, {})

// กรณีพื้นฐาน (trivial case)
MaxGuest(0, S) = len(S)

// ทั่วไป (trivial case) MaxGuest(i, S) = max (MaxGuest(i-1, S.push_back(i)), MaxGuest(i-1, S), MaxGuest(i-1, ไม่เอา)) ; ∀j dislike(i, sj) == 0 max (MaxGuest(i-1, S), MaxGuest(i-1, ไม่เอา)) ; ∃j dislike(i, sj)

5.2. กำหนดให้ prefer(a) คือค่าที่บอกว่าแขก a นั้นเป็นที่ชื่นชอบขนาดไหน เราต้องการให้ผลรวมของค่า prefer ของแขกที่เชิญทั้งหมดนั้นมากที่สุด กำหนดให้ Best(i, S) คือค่ามากสุดของผลรวมของค่า prefer เมื่อเราพิจารณาเฉพาะคนหมายเลขที่ 1 ถึง i โดยที่ S คือคนหมายเลย i+1 ถึง n ที่เราเลือกเชิญมาก่อนแล้ว และ S ไม่มีคนที่เกลียดกันเลย ซึ่งคำตอบที่เราต้องการคือ Best(n, {}) จงเขียน Recurrence relation นี้

// กรณีพื้นฐาน (trivial case) Best(0, S) = 0
// ทั่วไป (trivial case) Best(i,S) = max (prefer(i) + Best(i-1, S.push_back(i)), Best(i-1, S), prefer(i) + Best(i-1, ไม่เอา)) ; ∀j dislike(i, sj) == 0 = max (Best(i-1, S), prefer(i) + Best(i-1, ไม่เอา)) ; ∃j dislike(i, sj)

สำหรับข้อ 6 – 9 นั้น จะเป็นการออกแบบอัลกอริทึม ในแต่ละข้อนั้นสามารถอธิบายอัลกอริทึมที่ออกแบบด้วย รหัสเทียม (pseudo code) หรือว่า programming language ภาษาใดที่เคยเรียนมาก็ได้ แต่<u>ทุกข้อให้ระบุประสิทธิภาพเชิงเวลาด้วย</u>

6. (10 คะแนน) มีอาเรย์ A[1..5][1..n] ซึ่งเป็นอาเรย์ 2 มิติขนาด 5 แถว n คอลัมน์ เราต้องเลือกข้อมูลบางตัวมาจากอาเรย์นี้ โดยต้องเลือกข้อมูล 1 ตัวมาจากแต่ละคอลัมน์ โดยมีกฎดังนี้ 1) ต้องเลือกทีละตัว จาก คอลัมน์ 1 ไปยังคอลัมน์ n 2) ในคอลัมน์ที่ 1 จะเลือกข้อมูลตัวใดก็ได้ 3) ถ้าคอลัมน์ที่ i เลือกข้อมูลจากแถวที่ j แล้วในคอลัมน์ที่ i+1 สามารถเลือกได้เฉพาะแถวที่ j-1, j, j+1 (เมื่อมีแถวดังกล่าวอยู่) เท่านั้น เราต้องการเลือกให้ได้ผลรวมมากที่สุด จากกฎดังกล่าว หากเรากำหนดให้ B(r, c) คือผลรวมมากสุดของข้อมูลที่เลือก เมื่อเราได้ทำการเลือกจนถึงการเลือกข้อมูลแถวที่ r จากคอลัมน์ที่ c เราจะได้ Recurrence relation ของ B(r,c) ดังนี้

B(r, c) = A[r][c] ; if (c == 1)

B(r, c) = max(B(r, c-1), B(r+1, c-1)) + A[r][c] ; if (c > 1 && r == 1)

B(r, c) = max(B(r-1, c-1), B(r, c-1), B(r+1, c-1)) + A[r][c] ; if (c > 1 && r > 1 && r < 5)

B(r, c) = max(B(r-1, c-1), B(r, c-1),) + A[r][c] ; if (c > 1 && r == 5)

จงเขียนรหัสเทียมในการคำนวณค่า B ใด ๆ โดยใช้วิธี Dynamic Programming แบบ bottom up และให้เก็บผลลัพธ์ลงในตัวแปร B

```
Solve(n, A[1..5][1..n])

    initialize  B [1...5] [1...n]   every entries equal to 0.

    for i in [1...5] {
        B[i][1] = A[i][1]
    }

    for col in [2...n] {
        for row in [1...5] {
            if (row == 1) {
                B[row][col] = max( A[row][col-1], A[row+1][col-1] ) + A[row][col]
            }
            else if (row == 5) {
                B[row][col] = max( A[row-1][col-1], A[row][col-1] ) + A[row][col]
            }
            else {
                B[row][col] = max( A[row-1][col-1], A[row][col-1], A[row+1][col-1] ) + A[row][col]
            }
    }

  return B;    // ต้องคืนค่า B ซึ่งเป็นตาราง 2 มิติกลับมา
```

ยกตัวอย่างประกอบ

ประสิทธิภาพในการทำงานของอัลกอริทึมนี้คือ O(n)

7. (10 คะแนน) กำหนดให้มีอาเรย์ A[1..n] โดยที่ค่าใน a มีเฉพาะตัวเลข 0, 1, 2 เท่านั้น และ a ถูกเรียงจากน้อยไปมากแล้ว เราต้องการทราบว่าในอาเรย์ a มีตัวเลข 1 อยู่กี่ตัว (รับประกันว่า a มีขนาดมากกว่า 3 ช่อง และมีตัวเลข 0, 1, 2 อย่างน้อยเลขละ 1 ตัวแน่นอน) ตัวอย่างเช่น A = [0, 1, 1, 2, 2, 2] จะต้องตอบว่า 2 เพราะมี 1 อยู่ 2 ตัว จงออกแบบอัลกอริทึมสำหรับแก้ปัญหาดังกล่าว

```
Count(A, n) {
    b = lower_bound( A, 1, n )
    e = upper_bound( A, 1, n )
    return e-b
}

lower_bound( A, start, stop ) {
    if (start == stop) return start
    m = (start + stop) / 2
    if ( 1 <= A[m] ) return lower_bound( A, start, m );
    return lower_bound( A, m+1, stop);
}
upper_bound ( A, start, stop ) {
    if (start == stop) return start;
    m = (start+ stop) /2 ;
    if ( 1 >= A[m] ) return upper_bound ( A, m+1, start);
    return upper_bound (A, start, m) ;
}
```

1 2 3 4 5

0 1 1 1 1 1

ยกตัวอย่างประกอบ

ประสิทธิภาพในการทำงานของอัลกอริทึมนี้คือ O(log n)

8. (10 คะแนน) เราต้องการนับจำนวน string ความยาว n ที่ประกอบด้วยตัวเลข 0 หรือ 1 เท่านั้นที่มีเงื่อนไขคือ string นั้นจะต้องไม่มีเลข 1 ติดกัน เช่น เมื่อ n เป็น 3 จะมี string ที่ตรงกับเงื่อนไขคือ “000”, “001”, “010”, “100” และ “101” เท่านั้น รวมทั้งหมด 5 แบบ จงออกแบบอัลกอริทึมที่นับจำนวน string ดังกล่าวเมื่อระบุค่า n เป็น input ด้วยวิธี Divide & Conquer หรือ Dynamic Programming

8.1. จงกำหนดนิยามของ Recurrence relation ของปัญหานี้ โดยให้นิยาม function ที่ใช้ พร้อมทั้ง parameter ที่เกี่ยวข้อง

count (n) n is the size of the string

count number of possible string length n

8.2. จงเขียนสมการของ Recurrence relation ในข้อ 8.1

$$count(i) = \begin{cases} 2 & ; i = 1 \\ 3 & ; i = 2 \\ count(i-1) \times 2 - (count(i-1) - count(i-2)) & ; i > 2 \end{cases}$$

8.3. จงออกแบบอัลกอริทึมสำหรับปัญหานี้ (ไม่จำเป็นต้องทำตามข้อ 8.1 หรือ 8.2 ก็ได้ ขอให้ทำงานได้ถูกต้อง)

```
count ( n ) {
    vector <int>  dp(n+1) ;
    dp[1] = 2 ;   dp[2] = 3 ;
    for(int  i=3 ; i<=n ; i++) {
        dp[i] = dp[i-1] * 2 - (dp[i-1] - dp[i-2])
    }
    return  dp[n] ;
}
```

ยกตัวอย่างประกอบ

ประสิทธิภาพในการทำงานของอัลกอริทึมนี้คือ O(n)

9. (10 คะแนน) เกมตำรวจจับโจรเป็นดังนี้ มีโจรและตำรวจเข้าแถวเป็นเส้นตรงอยู่ เราสามารถแทนตำแหน่งของโจรและตำรวจด้วย อาเรย์ A[1..n] โดยที่ A[i] มีค่าเป็น 0 หมายความว่าตำแหน่งที่ i มีโจรอยู่ แต่ถ้า A[i] มีค่าเป็น 1 แสดงว่าตำแหน่ง i มีตำรวจยืนอยู่ ตำรวจแต่ละคนสามารถจับโจรได้คนเดียวเท่านั้น และตำรวจ ที่อยู่ ณ ตำแหน่ง i สามารถจับโจรที่อยู่ที่ตำแหน่งตั้งแต่ i-k ถึง i+k ได้เท่านั้น เราอยากทราบว่าจาก A ที่กำหนดให้ มีโจรโดนจับมากสุดกี่คน ตัวอย่างเช่น A = [0, 1, 1, 1, 0, 0, 0] และ k = 1 จะจับโจรได้สองคน คือ โจรที่ตำแหน่ง 1 และ 5 (ตำรวจ ณ ตำแหน่ง 3 ไม่สามารถจับโจรได้เลย) หรือให้ A = [1, 0, 0, 0, 1, 1, 1, 0, 0] และ k = 2 จะสามารถจับโจรได้มากสุด 4 คน จงออกแบบอัลกอริทึมสำหรับแก้ไขปัญหานี้ Greedy

```
Catch(A[1..n], k)
    int result = 0
    for( int i=1;    i<=n;    i++) {
        int pos,
        if( A[i]==1 ) {
            bool check_Left_theif = false, check_Right_theif = false ;
            for( pos = max(1, i-k); pos < i; pos++) {
                if(A[pos]==0 ) {
                    check_Left_theif = true;
                    break ;
                }
            }
            if( check_Left_theif ) {
                A[pos] = -1;
                result++;
            }
            else {
                for( pos = i+1    ; pos<=min(n,i+k); pos++) {
                    if(A[pos]==0 ) {
                        check_Right_theif = true;
                        break ;
                    }
                }
                if( check_Right ) {
                    A[pos]=-1;
                    result++;
                }
            }
        }
    return    result ;
```

ยกตัวอย่างประกอบ

ประสิทธิภาพในการทำงานของอัลกอริทึมนี้คือ O(nk)