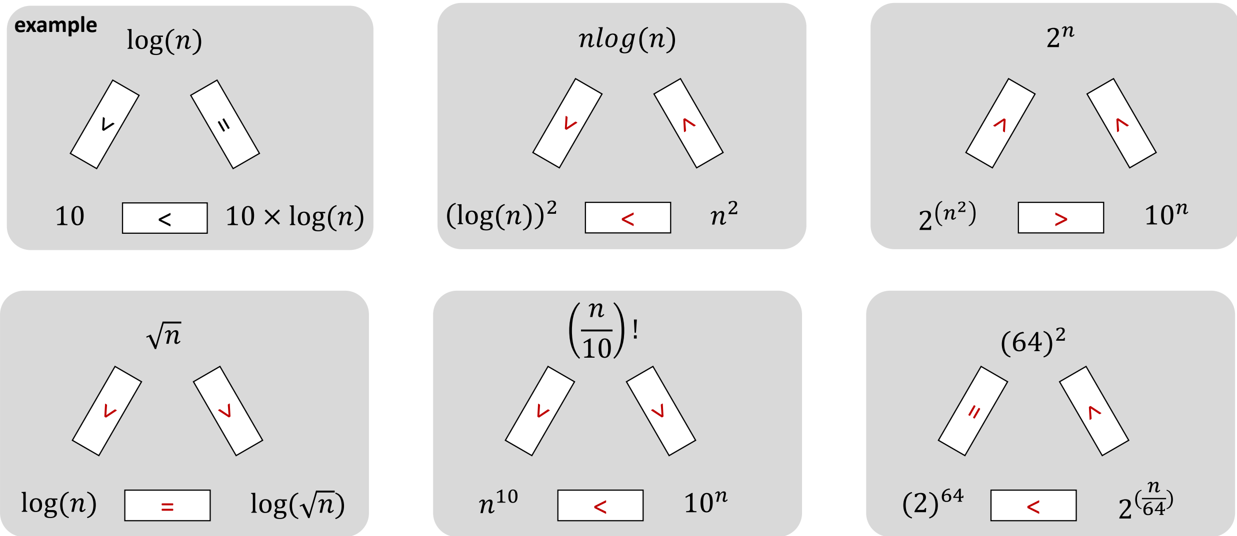
Data Structure Midterm Examination

3:30pm-5:20pm (110 minutes), Monday, April 27, 2015

1. Please find the **asymptotic order** of the following function groups:



2. Please consider the **KMP** algorithm.

A. Please analyze the **failure function** of the following pattern string.

'a'	'a'	'b'	'a'	'a'	'a'	'b'	'a'	'a'	x
-1	0	-1	0	1	1	2	3	4	__5__ if (x == 'a') __2__ if (x == 'b') __-1__ otherwise

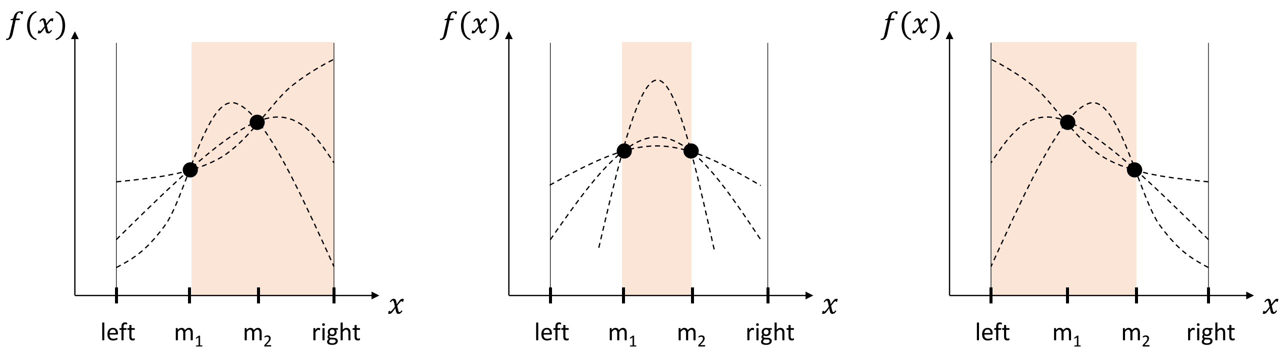
B. Please compose a pattern string that exhibits the following failure function. Please try to compose as long a string as possible and mark an 'X' to denote the position (if any) where the failure function becomes invalid.

-1	-1	0	-1	0	1	2	3	0	1
a	b	a	c	a	b	a	c	X	X

3. Please consider the infix expression A*((B-A)+3@C/D), in which '@' is a binary operator whose priority is higher than '+' and '-' but lower than '*' and '/'. Please fill in the following table that shows the procedure of infix-to-postfix using a stack.

	Stack State	Output State
A		A
*	*	A
(	*(	A
(	*((	A
B	*((	AB
-	*((-	AB
A	*((-	ABA
)	*(	ABA-
+	*(+	ABA-
3	*(+	ABA-3
@	*(+@	ABA-3
C	*(+@	ABA-3C
/	*(+@/	ABA-3C
D	*(+@/	ABA-3CD
)	*	ABA-3CD/@+
(End)		ABA-3CD/@+*

4. **Ternary Search** can be used to find the maximum of a bell-shape function. Let $f(x)$ be a bell-shape function defined on some interval [left, right] and m_1 and m_2 be two arbitrary points in the interval such that $left < m_1 < m_2 < right$. The values of $f(m_1)$ and $f(m_2)$ can exhibit three possibilities, each of which indicates a reduced interval that the maximum lies in, as depicted as follows.



if $f(m_1) < f(m_2)$
the maximun lies in [m_1, right]

if $f(m_1) == f(m_2)$
the maximun lies in [m_1, m_2]

if $f(m_1) > f(m_2)$
the maximun lies in [left, m_2]

A. Let n = (left-right+1) be the problem size. Please analyze the step count per execution (**s/e**) of the following iterative version of the **Ternary Search** algorithm:

		s/e
1:	**int** TernarySearch(**int** left, **int** right)	--
2:	{	0
3:	**while**(1){	1
4:	**If** (right - left <= 2)	1
5:	**return** integer x that has the greatest f(x), left <= x <= right	1
6:	**int** m1 = left + (right - left)/3;	1
7:	**int** m2 = right - (right - left)/3;	1
8:	**if** (f(m1) < f(m2))	1
9:	left = m1;	1
10:	**else if** (f(m1) > f(m2))	1
11:	right = m2;	1
12:	**else**	0
13:	{ left = m1; right = m2; }	1
14:	}	0
15:	}	0

B. Please show a **recursive version** of the ternary search algorithm. You can directly quote the iterative version of code using line numbers.

1:	**int** TernarySearch(**int** left, **int** right)	
2:	{	
3:	// **while**(1){ // remove the loop	
4:	**If** (right - left <= 2) // this serves as the boundary condition	1
5:	**return** integer x that has the greatest f(x), left <= x <= right	1
6:	**int** m1 = left + (right - left)/3;	1
7:	**int** m2 = right - (right - left)/3;	1
8:	**if** (f(m1) < f(m2))	1
9:	left = m1;	1
10:	**else if** (f(m1) > f(m2))	1
11:	right = m2;	1
12:	**else**	0
13:	{ left = m1; right = m2; }	1
14:	**return** TernarySearch(left, right); // recursive call	T(2n/3)
15:	}	

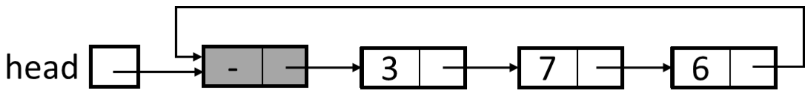
C. Please analyze the time complexity of the recursive **Ternary Search** using the **O** notation. Try to show as tight a bound as you can.

T(n) = T(2n/3) + 1
= T(4n/9) + 1 + 1
= 1 + … + 1   (a total of **O**(log(n)) ones)
= **O**(log(n))

5. Some languages allow array index to start from any arbitrary integer. Please consider a three-dimension array Z[1….20][20…70][1…15] in the row-major order with one-byte element size in this type of language. Assume Z[10][20][1] is stored at address 2000.
   A. What is the address of Z[10][30][10]?
      (10, 30, 10) – (10, 20, 1) = (0, 10, 9)
      2000 + 0*(51*15*1) + 10*(15*1) + 9*(1) = 2000+159 = 2159

   B. What is the array index at the location 2050?
      Z[10][20][1] → 2000
      Z[10][23][1] → 2045
      Z[10][23][6] → 2050

6.
   A. Please depict a **circular, singly linked list of integers with a header node**.



   B. Please design an algorithm that can **sort the abovementioned type of list** using pseudo code.

```
void Sort()
{
    if(head->link->link == head) return;
    Node* min_ptr = head->link;
    int min_val = head->link->data;

    for(Node* a=head->link; a!=head; a=a->link) {
        for(Node* b=a; b!=head; b=b->link) {
            if(b->data < min_val){
                min_ptr = b;
                min_val = b->data;
            }
        }
        swap(a->data, min_ptr->data);
```

```
        }
    }
```

7. You’re asked to perform **postfix evaluation (note: NOT the infix-to-postfix)** using **two queues**, q1 and q2, without any stack. A queue supports **add()**, which adds an element at the rear of the queue, **remove()**, which take an element from the front of the queue, and **size()**, which reports the number of elements in the queue. Please show your algorithm using pseudo code.

```
Eval (Expression e)
{
    Queue q1, q2;
    Token x;
    while(not end of the expression e){
        x=NextToken(e);
        if (x is an operand) {
            q1.add(x);
        }else{
            Let n be the number of operands required by the operator x.
            for(int i =0; i<(q1.size())-n; i++)     q2.add(q1.remove());
            Perform the operation x using the operand(s) in q1 and obtain a result;
            for(int i =0; i<q2.size(); i++)          q1.add(q2.remove());
            q1.add(the result);
        }
    }
    return the result in q1;
}
```

8. Short answer questions / explanation of terminologies
   A. What issue does **C++ template** aim to address?
      Classes and functions with different data types contain duplicate code.

   B. What are the pros and cons of **data encapsulation?**
      Pros:
      1. Encapsulation allows us to change one part of code without affecting other part of code.
      2. Encapsulation prevents bugs caused by unexpected access to internal data of an object.

3. Encapsulation hides object details and thus makes an object easy to use.

Cons:

1. Encapsulation tends to increase code size because of the need to access private data indirectly.
2. Encapsulation tends to decrease performance because of the need to access private data indirectly.

C. Is it possible that an **O**(2^n) (i.e., exponential-time) algorithm outperforms an **O**(n) (i.e., linear-time) algorithm in terms of speed? How can this occur?

Yes. This situation can occur when

1. n is not large enough, or
2. the input data does not cause the worst-case time complexity

9. Please prove that

if $\mathrm{F(n)} = \mathbf{O}(\mathrm{G(n)})$, then $(\mathrm{F(n)} + \mathrm{G(n)}) = \mathbf{O}(\mathrm{G(n)})$.

$\mathrm{F(n)} = \mathbf{O}(\mathrm{G(n)})$

→ There exists c and n0 such that $\mathrm{F(n)} \le \mathrm{c} \times \mathrm{G(n)}$ for all $\mathrm{n} \ge \mathrm{n0}$

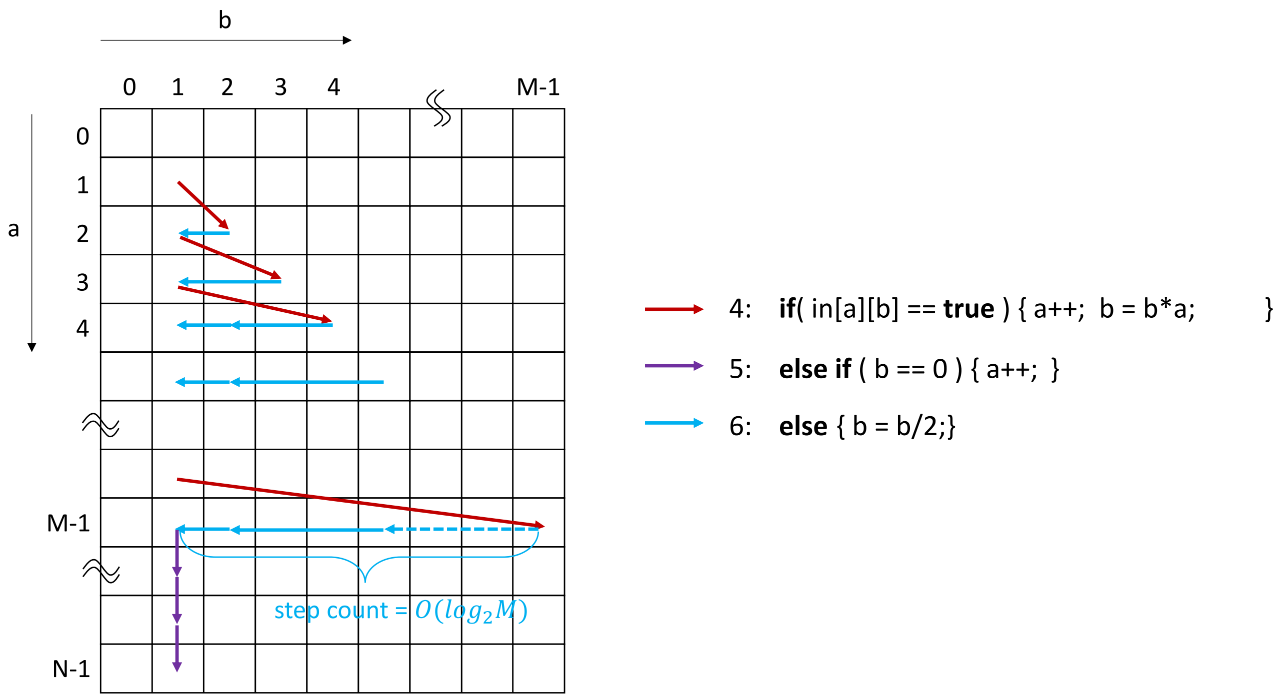
Let c' = (c+1)

$(\mathrm{F(n)} + \mathrm{G(n)}) \le (\mathrm{c} \times \mathrm{G(n)} + \mathrm{G(n)}) = \mathrm{c'} \times \mathrm{G(n)}$ for all $\mathrm{n} \ge \mathrm{n0}$

10. Please analyze the **worst-case time complexity** of the following procedure with **brief explanation**. Please find as tight a bound as you can.

```
// in[][] is an N-by-M input array
int a=1, b=1;
while ( a<N && b<M ) {
    if( in[a][b] == true )  { a++;    b = b*a;    }
    else if ( b == 0 )      { a++;                }
    else                    { b = b/2;            }
}
```

If N>M, the following figure depicts the worst case scenario.



The asymptotic time complexity is $\mathrm{N} + \sum_{k=1}^{M} log_2 M = \mathrm{N} + log_2(M!) = N + Mlog_2 M$

If N≤M, the asymptotic time complexity is $N + log_2(N!) = N + Nlog_2 N = Nlog_2 N$

11. We want to design a **circular queue** class that is implemented in terms of a **16-element integer array** and can store up to **15 integers**. Please show **add()**, **remove()**, and **size()** operations using **pseudo code**. These operations should all be of **O**(1) time complexity.

```
class circular_queue{
      …
      void add(int e);
      void remove();
      int size();
private:
      int data[16];
      int front=0
      int back=0;
}

class circular_queue::size()
{
      if(back >= front) return (back-front);
      else return (16+back-front);
```

```
}

void circular_queue::add(int e)
{
	if((back+1)%16 == front)
		throw “add element to a full queue”;
	data[back] = e;
	back = (back+1)%16;
}

int circular_queue::remove()
{
	if(back == front)
		throw “remove element from an empty queue”;
	int e = data[back];
	front = (front+1)%16;
	return e;
}
```