Entry

COL 106 Autumn 2017 Minor 1

Welcome to minor 1. The exam is for 1 hour 10 minutes. Please use answering questions. Do not use a pencil.

Before starting the exam, close your eyes and take three deep breath the exam is not an accurate reflection of your understanding of the n you are relaxed, you will likely perform better.

Question Number	Maximum Marks	
1	05	
2	06	
3	05	
4	11	
5	04	
6	03	
7	02	
8	04	

1. [5 points] Analyze the following pseudo code:

```
void fun(int n, int m) {

        if (m <= 0) return;
        if (m > n) return;

        print m;
        fun(n, 2*m);
        print m;
}
```

(a) [2 points] What sequence will be printed if we call `fun(2000, 8)`?

8 16 32 64 128 .... 1028 1028 ---- 128 64 32 16 8

... includes powers of 2 in between.

(b) [3 points] What will be the time complexity (expressed in terms of m and n) for this procedure? Explain your answer.

The procedure runs upto second argument ~~greater~~ smaller than first one.

If fun is called inside the first call to fun, k times, then second argument becomes $2^k * m$.

∴ The recursion terminates when $2^k * m > n \Rightarrow k > \log_2(n/m)$

~~Each~~ $T(n,m) = c + T(n,2m)$

$= c + c + T(n, 2^2 m) = \underbrace{c + c + \dots c}_{k} + T(n, 2^k m)$

∴ Time complexity is $O(\log(n/m))$ ~~assuming n > m for if~~

2. [6 points] Read the following pseudo-code

```
int divide(int n, int s) {

        q ← 0;
        r ← n;

        while (r >= s) {
                q ← q + 1;
                r ← r - s;
        }

        return q;
}
```

Define a loop invariant for the `while` loop (invariant should be true at the beginning of each loop). Prove the loop invariant using methods discussed in class. Using this loop invariant prove that the function `divide` (for positive `n` and `s`) returns the quotient from the division of `n` by `s`.

Loop invariant : At the beginning of each iteration $r + q \times s = n$.

Initial check : Initially $r = n \quad q = 0$

$r + qs = n + 0 = n$

$\therefore$ initially invariant condition is true.

Maintenance : Let invariant be true at i^{th} iteration.

$\therefore \quad r_i + q_i * s = n$

r_i is value of r at beginning of i^{th} iteration

q_i is value of q at beginning of i^{th} iteration.

At $(i+1)^{th}$ iteration,

$r_{i+1} = r_i - s$

$q_{i+1} = q_i + 1$

$r_{i+1} + q_{i+1} \times s = r_i - s + (q_i + 1) \times s$

$= r_i + q_i \times s = n$ by our assumption that invariant is true at i^{th} iteration.

Thus truth at i^{th} iteration $\Rightarrow$ truth at $(i+1)^{th}$ iteration.

Termination : Loop terminates when $r < s$.

By, loop invariant,

$n = qs + r \qquad r \in [0, s)$

According to Euclid's division lemma

$a = qb + c \quad c \in [0, s)$ then q is quotient of a/b and r is remainder.

Thus, using this we know that q at termination is quotient for n/s.

Thus, the divide function returns quotient of n/s. (if it returns).

In case of $s = 0$, the function never returns any value and goes in an endless loop.

3. [5 points] Consider functions *f(n)* and *g(n)* as given below. Use the "most precise" asymptotic notation to show how function *f* is related to function *g* in each case (i.e., $f \in ?(g)$). For example, if you were given the pair of functions $f(n) = n$ and $g(n)=2n+1$ then the correct answer would be: $f \in \Theta(g)$. To avoid any ambiguity between $O(g)$ and $o(g)$ notations due to writing, use *Big-O(g)* instead of $O(g)$.

f(n)	*g(n)*	*Relation* $f \in ?(g)$
2^n	3^n	$f \in o(g)$ ✓
2^n	n^{1000}	$f \in \omega(g)$ ✓
0.5^n	1	$f \in O(g)$ ✓
$n^3 + 2n + 1$	$\frac{1}{100}n^3 + n \log n$	$f \in \Theta(g)$ ✓
$\log_2 n$	$\log_3 n$	$f \in \Theta(g)$ ✓

5

4. [11 points] Recall that queue is a FIFO data structure with two main operations: *enqueue* and *dequeue*. Similarly, stack is a LIFO data structure with two main operations: *push* and *pop*, and additional operations like *isEmpty()*. Your goal is to implement a queue with two stacks so that the amortized time complexity of a sequence of *enqueue* and *dequeue* operations is constant (in the number of stack operation calls). Provide the pseudo-code for the *enqueue* and *dequeue* methods. Also, provide a proof for the amortized time complexity.

9.5+0.5

q ← The queue to be implemented
s1 ← stack 1 (we will be using it for enqueue operations)
s2 ← stack 2 (we will be using it for dequeue operations)

10

```
algorithm enqueue (object o)
    s1.push(o);
    q.size++;
```

✓

```
algorithm dequeue ()
    if s2.isEmpty() then
        while !(s1.isEmpty()) do
            s2.push(s1.pop())
```

✓

```
if s2.isEmpty() then
    throw EmptyDequeException          // if still empty means both stacks are empty and there are no elements to remove.
else
    q.size--
    return s2.pop()
```

5

The enqueue operation returns after exactly 2 step for any object passed irrespective of the size of queue.

∴ Ammortized time for n operations of push = $\frac{\overbrace{T(1)+T(1)\ldots T(1)}^{n \text{ times}}}{n}$

= O(1) 0.5

For deque operation, suppose our queue has n elements and we need to remove all.

Initially s2 contains c elements ($c \geq 0$) and s1 contains $(n-c)$ elements.

First dequeue call will return in O(1). Similarly second, third uptil c^{th} call to dequeue. ∴ Time to remove first "c" elements = $c\,T(1)$

1.5 Now, for $(c+1)^{th}$ call, we need to transfer all $(n-c)$ elements to s2.

This will take $(n-c)$ steps + 1 step to pop again from s2.

+ ∴ Time = $2T(n-c)+T(1)$

After this all dequeue calls will be returned in O(1).

0.5 ∴ Total time for n dequeue operations = $\underbrace{n}_{\text{n pops from s2}} + \underbrace{2(n-c)}_{\text{transfer operations}}$

+0.5 You need to specify where for a sequence a queue gets empty multiple times.

$\leq 3n$

Ammortised time $\leq \frac{3n}{n} = 3$ ✓

Ammortised time for dequeu = O(1)

Since both enqueue and dequeue run in ammortised time O(1), their combination will run too in O(1) ammortised time.

5. [4 points] You are given a Vector ADT with the following interface:

```
public interface Vector {

/** returns the number of elements in the vector */
public int size();

/** returns whether the vector is empty */
public Boolean isEmpty();

/** returns the element stored at the given rank (rank ∈ 0..size()-1)*/
public Object elemAtRank(int r) throws OutOfBoundException;

/** replaces the element stored at the given rank (rank ∈ 0..size()-1)*/
public Object replaceAtRank(int r, Object e) throws OutOfBoundException;

/** inserts an element at the given rank (rank ∈ 0..size())*/
public void insertAtRank(int r, Object e) throws OutOfBoundException;

/** removes the element stored at the given rank (rank ∈ 0..size()-1)*/
public Object removeAtRank(int r) throws OutOfBoundException;

}
```

Our goal is to define an adaptor for the Deque ADT using the Vector ADT. For each Deque method give the corresponding call to the Vector method that realizes the same functionality.

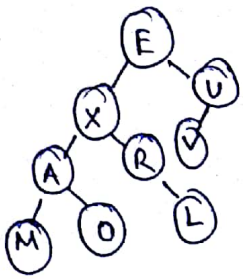
Deque Method	Realization with Vector Methods
`size()`	size()
`isEmpty()`	isEmpty()
`insertFirst(e)`	insertAtRank(0, e)
`insertLast(e)`	insertAtRank(this.size(), e)
`removeFirst()`	removeAtRank(0)
`removeLast()`	removeAtRank(this.size()-1)
`first()`	elemAtRank(0)
`last()`	elemAtRank(this.size()-1)

(9)

6. [3 points] Draw a single binary tree T such that each internal node of T stores a single character and

- a preorder traversal of T yields EXAMORLUV

- an inorder traversal of T yields MAOXRLEVU



3

7. [2 points] Consider a sorted circular doubly linked list of numbers where the head element points to the smallest element in the list.

2

(a) What is the asymptotic complexity of determining whether an element e appears in the list? $O(n)$ $\Theta(n)$

(b) What is the asymptotic complexity of finding the median of the list of numbers? $O(n)$ $\Theta(n)$

(c) What is the asymptotic complexity of finding the smallest element in the list? $O(1)$ $\Theta(1)$

(d) What is the asymptotic complexity of finding the largest element in the list? $O(1)$ $\Theta(1)$

8. [4 points] True/False

(a) If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $h(n)$ is always $\Omega(f(n))$. True

(b) The minimum number of nodes in a binary tree of height d is $d+1$. True

4

(c) The minimum height of tree containing n nodes is $\lceil \log_2 n \rceil$. False (only True for binary trees)

(d) If class A implements interface I, class B extends A, class C extends B, and interface J extends interface I, then C always implements J. False