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

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

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

    print n;
    fun(4*n, m);
    print n;
}
```

(a) [2 points] What sequence will be printed if we call `fun(8, 4000)`?

8, 32, 128, 512, 2048, 2048, 512, 128, 32, 8 ✓

(2)

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

Taking print steps to take constant time of 1 unit,

For some n, m we call the fun(4*n, m) recursively till x times where $n * 4^x > m \rightarrow x \sim \log_4\left(\frac{m}{n}\right)$ ✓

(3) so for 1 call total time is 1+1+1+1+1 → (+1 for function call) → For print after function call.

(for 2 comparisons; for print before call)

and total calls = $\log_4\left(\frac{m}{n}\right) = \frac{1}{2}\log_2\left(\frac{m}{n}\right)$ → Total time $\frac{5}{2}\log_2\left(\frac{m}{n}\right)$

Time complexity → $O\left(\log_2 \frac{m}{n}\right)$

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

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

    q = 0;
    s = n;

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

    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 `d`) returns the quotient from the division of `n` by `d`.

Let us define the loop invariant to be $s + qd = n$ for all steps where n, d are constant inputs and s & q are variables at each step ✓

(2)

Initial → Initially before loop, $s = n, q = 0$

so $s + qd = n + 0 \times d = n$

So, True. (1/2)

Maintainence → let this be true at the start of ith iteration

At start of ith iteration → So, we take $q = i-1$, $s = n - (i-1)d$

$\Rightarrow s + qd = n - (i-1)d + (i-1)d = n$ (True) (1.5)

At ith iteration $q \leftarrow q+1 \Rightarrow q = i$

$s \leftarrow s-d \Rightarrow s = n - id$

So,

At end of ith iteration → $s + qd = n - id + id = n$

So, loop invariant is true at the end an ith iteration.

Terminating → We say that loop invariant is also true at end as $q = \frac{n}{d}$, $s = 0$?? s can be anything Hw [0, d-1]

$s + qd = 0 + \frac{n}{d} \times d = n$

So, we proved that throughout the loop $s + qd = n$ is true.

Now, For 2 numbers say a, b we can write them as

$a = bx + c$ for $x \geq 1$, $c \geq 0$ if $a \geq b$

and here x = quotient of (a/b) and c is remainder

So, we say in loop invariant $s + qd = n$ (1/2)

q is quotient.

And we return q at end which means we return the quotient.

Hence, Proved.

How s > 0, s < d.

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 $f \in ?(g)$
0.5^n	1	small-o
$\log_2 n$	$\log_3 n$	Θ (Big Theta)
2^n	3^n	small-o
$n^3 + 2n + 1$	$\frac{1}{100}n^3 + n \log n$	Big-O
2^n	n^{1000}	small-ω (small omega)

3

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.

Let Stack1 and Stack2 be objects of 2 stacks.

```
Public void enqueue(Object a) {
    while (Stack2.isEmpty() = false) {
        Object b ← Stack2.pop();
        Stack1.push(b);
    }
    Stack1.push(a);
}

Public E dequeue() throws EmptyQueueException {
    if (Stack2.isEmpty())
        throw new EmptyQueueException;
    else
        while (Stack1.isEmpty() = false) {
            Object a ← Stack1.pop();
            Stack2.push(a);
        }
    Object temp ← Stack2.pop();
    return temp;
}
```

after every enqueue, dequeue gives exception

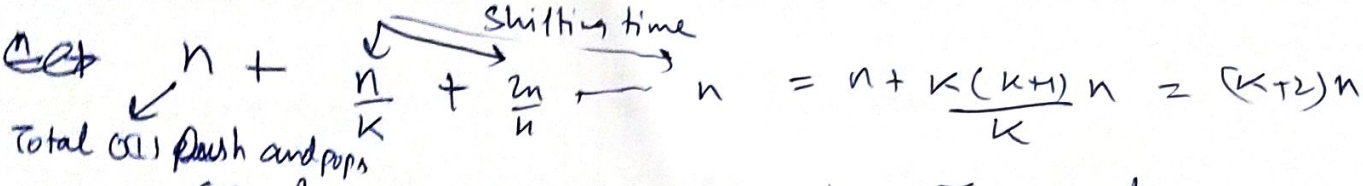
Time complexity :

Suppose there are n operations in all

then for each enque operation we have $O(1)$ steps to enqueue if stack 2 is empty else $O(n)$ steps

Similarly for a deque operation we have $O(1)$ steps if Stack1 is empty else $O(n)$ steps.

so if a deque operation comes k times after each $\frac{n}{n}$ enque operation we have



so, for amortized analysis $T_n = \frac{f(n)}{n}$

$$\rightarrow \frac{(k+2)n}{n} = (k+2) = O(1)$$

Ans So, amortized running time is $O(1)$

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;

}
```

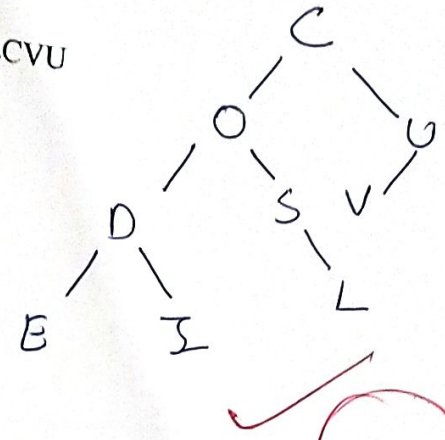
Our goal is to define an adaptor for the Deque ADT using the Vector ADT. For each Deque method give the corresponding call to the Vector method that realizes the same functionality.

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

3.5/4

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 CODEISLUV
- an inorder traversal of T yields EDIOSLCVU



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.

(a) What is the asymptotic complexity of finding the smallest element in the list? Big-O(1)
As we just return Head element

(b) What is the asymptotic complexity of determining whether an element e appears in the list?
Big-O(N) (Although we may apply Binary search but reaching an element will take O(N) steps in List)

(c) What is the asymptotic complexity of finding the median of the list of numbers?
Big-O(N)

(d) What is the asymptotic complexity of finding the largest element in the list?
As we just return Head.Prev element
Big-O(1)

2

8. [4 points] True/False

(a) 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

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

(c) The minimum height of tree containing n nodes is $\lceil \log_2 n \rceil$. True

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

3