

COMP3506/COMP7505—Algorithms and Data Structures

School of Information Technology and Electrical Engineering

Week 5 Tutorial Sample Solutions

Question 1

Give a big-Oh characterization, in terms of n , of the running time of the following algorithm.

Algorithm SumEven

Input: An array A storing $n \geq 1$ integers.

Output: The sum of the elements at even cells in A .

$s \leftarrow A[0]$

for $i \leftarrow 2$ to $n - 1$ by increments of 2 do

$s \leftarrow s + A[i]$

return s

$O(n)$

Question 2

Algorithm A executes an $O(\log n)$ -time computation for each entry of an n -element array. What is the worst case running time of Algorithm A ?

$O(n \log n)$

Question 3

Given an n -element array X , Algorithm B chooses $\log n$ elements in X at random and executes an $O(n)$ -time calculation for each. What is the worst-case running time of Algorithm B ?

$O(n \log n)$

Question 4

Suppose you have a deque D containing the numbers $(1, 2, 3, 4, 5, 6, 7, 8)$, in this order. Suppose further that you have an initially empty queue Q . Give a pseudo-code description of a method that uses only D and Q (and no other variables or objects) and results in D storing the elements $(1, 2, 3, 5, 4, 6, 7, 8)$, in this order.

Algorithm numberKick

Input: a deque D containing the numbers $(1, 2, 3, 4, 5, 6, 7, 8)$

Output: a deque containing the numbers $(1, 2, 3, 5, 4, 6, 7, 8)$

$Q = \text{new Queue}$

```
Q.enqueue( D.removeFirst() ) // Q (1)
Q.enqueue( D.removeFirst() ) // Q (1, 2)
Q.enqueue( D.removeFirst() ) // Q (1, 2, 3)
D.addLast( D.removeFirst() ) // D (5, 6, 7, 8, 4)
Q.enqueue( D.removeFirst() ) // Q (1, 2, 3, 5)
Q.enqueue( D.removeLast() ) // Q (1, 2, 3, 5, 4)
Q.enqueue( D.removeFirst() ) // Q (1, 2, 3, 5, 4, 6)
Q.enqueue( D.removeFirst() ) // Q (1, 2, 3, 5, 4, 6, 7)
Q.enqueue( D.removeFirst() ) // Q (1, 2, 3, 5, 4, 6, 7, 8)
```

```
while  $\neg$  Q.isEmpty
do
```

```
    D.addLast( Q.dequeue() )
```

```
done
```

```
return D
```

Question 5

Describe how to implement the stack ADT using two queues. What is the running time of the *push()* and *pop()* methods in this case?

A stack may be implemented using two queues by using the second queue as a buffer. When items are pushed onto the stack they are added onto the end of the queue. Each time an item is popped, the $n - 1$ elements of the first queue must be moved to the second, while the remaining item is returned.

```
public class QueueStack<E> implements IStack<E>
{
    private IQueue<E> q1 = new Queue<E>();
    private IQueue<E> q2 = new Queue<E>();

    public void push( E e )
    {
        q1.enqueue( e )           //O(1)
    }

    public E pop( E e )
    {
        while ( 1 < q1.size() )    // O(n)
        {
            q2.enqueue( q1.dequeue() );
        }
        swapQueues();
        return q2.dequeue();
    }

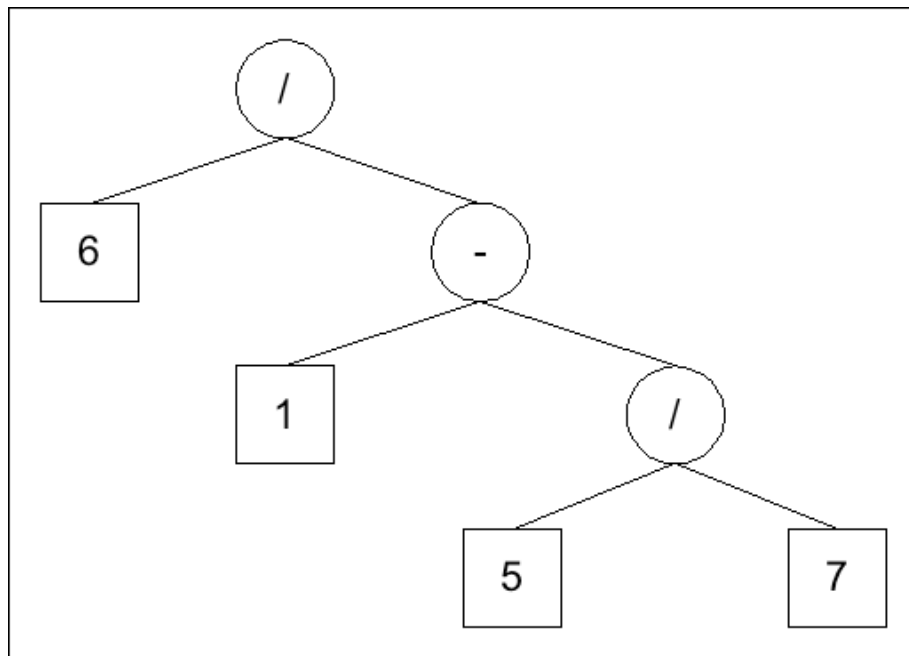
    private void swapQueues()
    {
        IQueue<E> Q = q2;
        q2 = q1;
        q1 = Q;
    }
}
```

Question 6

Draw an arithmetic expression tree that has four external nodes, storing the numbers 1, 5, 6, and 7, (with each number stored in a distinct external node, but not necessarily in this order), and has three internal nodes, each storing an operator from the set $\{+, -, \times, /\}$, so that each value of the root is 21. The operators may return and act on fractions, and an operator may be used more than once.

Hints:

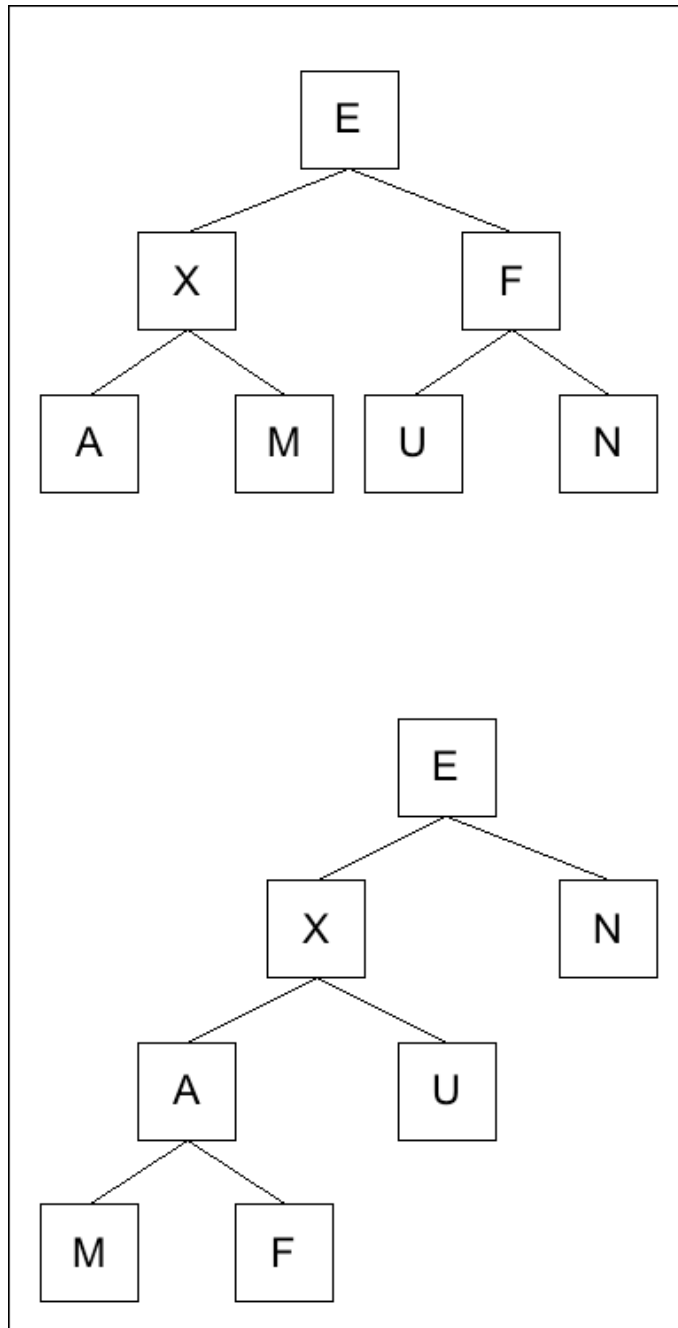
Use of number 1 suggests division e.g. $1/5$ or subtraction e.g. $6 - 1$



Question 7

Draw a (single) binary tree T such that

- Each internal node of T stores a single character
- A *preorder* traversal of T yields EXAMFUN
- An *inorder* traversal of T yields MAFXUEN



Question 8

Describe an algorithm for computing the number of descendants of each node of a binary tree, based on the Euler tour traversal.

How would you override the abstract visit methods of the class presented in lectures?

Algorithm visitExternal(p, r)

Input: a position p and result object r

Output: result object r is modified

r .finalResult = **new** Integer(0)

Algorithm visitRight(p, r)

Input: a position p and result object r

Output: result object r is modified

$d \leftarrow 0$

if r .leftResult **then**

$d \leftarrow d + 1 + r$.leftResult

if r .rightResult **then**

$d \leftarrow d + 1 + r$.rightResult

r .finalResult = d

The visitExternal algorithm returns 0 as final result, as each external object has 0 descendants.

The visitRight algorithm (analogous to a *postorder* visit) sums the descendants of the left child (if it exists) and the descendants of the right child (if it exists) – adding those children – then stores that sum into the final result.