Data Structures and Object Oriented Programming

Practice Midterm

Contents

1 True or False (10 points)

Use the true definition of Big-O (i.e. upper bound, even if a very loose one).

(a) Specifications for abstract data types are not concerned with the efficiency of implementations.

(b) An interface in Java can be an extension of another interface

(c) $2^n + 52n + 1000 = O(n^{2015})$

(d) If x and y are references to objects of the same type, then the expression x == y will evaluate to true if and only if x and y refer to the same object.

(e) $5n + 8n^2 + 100n^3 = O(n^4)$

(f) An algorithm that requires O(nlog n) time is always faster than one that requires $O(n^2)$ time

(g) Adapter Design Pattern can be used to implement a Queue.

(h) $n^2 + n + logn = \Theta(n^2)$

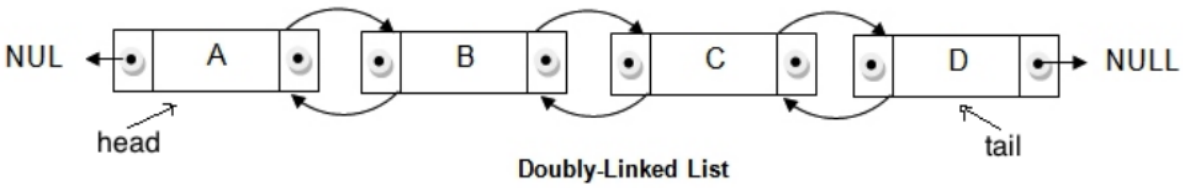
(i) If one implements a queue using a circular array, then that implementation can be coded in such a way that $\Theta(enqueue) = \Theta(dequeue) = \Theta(1)$.

(j) Removing an element from the beginning of a Doubly Linked List with head is as efficient as removing an element from the beginning of a Singly Linked List with head.

(k) Using dummy sentinel nodes for linked lists simplifies the code for most operations by avoiding special cases.

2 Short Answers (25 points)

(a) (6 points) Consider the following Doubly Linked List. You are given references to the Nodes B and C. Is it possible to swap the nodes B and C in the list, using only the references to these nodes? If yes, then write down the list of assignment statements that will correctly swap B and C. If not, justify your answer by saying what information is missing to perform the swap operation.



(b) (5 points) What is the worst-case time complexity in terms of Θ of the following loop? Justify your answer.

```
for (int i=0; i<n; i++) {
    for(int j=1; j<n; j=j*2) {
        //do some computation
    }
    for(int k=1; k<n; k++) {
        //do some computation
    }
}
```

(c) (6 points) Consider the following method (Assume that Stacks S1, S2 and S3 are empty when passed to this method)

```
public void SecretMethod(Stack<String> S1, Stack<String> S2, Stack<String> S3)
{
    S1.push("blue");
    S1.push("red");
    S1.push("green");
    S1.push("yellow");
    S1.push("black");
    S1.push("orange");

    for(int i=0; i<3; i++) {
        S2.push(S1.pop());
    }

    for(int i=0; i<2; i++) {
        S3.push(S2.pop());
    }

    for(int i=0; i<2; i++) {
        S1.push(S3.pop());
    }
}
```

Write down the contents of S1, S2 and S3 (from top of the stack to bottom) after the above method is executed.

(d) (3 points) Is it possible to achieve O(1) time for Stack operations push() and pop() if we use a linear array to implement the Stack? Explain.

(e) (4 points) Name two advantages and two disadvantages of LinkedLists over Arrays.

(f) (1 point) What is the average-case asymptotic complexity of linear search (in an array or a list)?
i) $O(n^2)$ ii) O(logn) iii) O(n) iv)O(nlogn) v)O(1)

3 Coding (15 points)

Given below is a partially implemented Iterator for a **Circular** Doubly Linked List (named CircularDLL) with no dummy nodes. Implement the following methods for the Iterator class.

(a) *public E next() throws NoSuchElementException* : Returns the next item in the list and moves the iterator ahead by one position. (5 points)

(b) *public int previousIndex()* : Returns the index of the previous element. (3 points)

(c) *public void removeNNodes(int n)* : Remove 'n' number of nodes from the Iterator's current position, (last element that was returned by a call to the next). This is similar to the normal Iterator remove() operation, except that it removes 'n' nodes instead. For simplicity, you may assume that this particular iterator always moves forward. (7 points)

You may assume that the following methods have been implemented in the Node class and use the same in your implementation.

```
public Node(E element)
public void setPrev(Node p)
public void setNext(Node n)
public void setElement(E e)
public Node getNext()
public Node getPrev()
public E getElement()
```

You may also assume that you have access to the following variables in the CircularDLL class:

Node head : reference to head
Node tail : reference to tail
int nelems : a variable that holds the number of the elements in the list

```
protected class MyListIterator implements ListIterator<E> {

    private boolean forward;
    private Node left;
    private Node right; /* Cursor sits between these
    *left and right nodes
    */
```

```
private int index; // Tracks current position.
          //what next() would return

/**
 * Constructor for the iterator. The iterator initially
 * sits between the head and tail.
 */
public MyListIterator() {
   index = 0;
   left = tail;
   right = head;
   forward = true;
}
```