Practice Questions

1) What is the Big-Oh complexity of the following methods? Defend your answer. Assume all n are positive.

```java
public int methodA (int n){
    int count = 0;
    for (int i = 0; i < n; i++){
        count++;
    }
    count += 5;
    for (int i = n; i >= 0; i--){
        count++;
    }
    return count;
}
```

\(n + n = \Theta(n)\)

```java
public int methodB (long n){
    int count = 0;
    for (int i = 0; i < n; i *= 2){
        count++;
    }
    for (int i = 0; i < n / 100000; i++){
        count++;
    }
    return count;
}
```

\(O(\log(n)) + O(n/100000) = \Theta(n)\)
public int methodC (int n){
    int value = 0;
    for (int i = 1; i < n; i++){
        for (int j = 0; j < i; j++){
            value += i - j;
        }
    }
    return value;
}

O(n^2)

2) A Deque (short for double-ended queue) is a data structure that permits insertion and deletion at both the front and the back in O(1) time. Explain why a singly-linked list is not suitable for storing elements in a deque.

Impossible to add onto the back in O(1) time.
3) Deques are usually implemented using a doubly-linked list, as shown below with the following class skeletons. A class skeleton shows the fields and methods in a class, without writing all the code.

```java
public class Node{
    private Node next;
    private Node prev;
    private String value;

    public Node(String str){
        value = str;
        next = null;
        prev = null;
    }

    public Node getNext();
    public Node getPrev();
    public void setNext(Node n);
    public void setPrev(Node n);
}

public class Deque{
    private Node first;
    private Node last;

    public Deque(){
        first = null;
        last = null;
    }

    public void pushFront(Node n);
    public void pushBack(Node n);
    public Node popFront();
    public Node popBack();
    public Node peekFront();
    public Node peekBack();
    public boolean isEmpty();
}
```

Implement the Deque method `pushBack` to place the parameter Node `n` onto the back of the deque.

```java
public void pushback(Node n){
    if (first == null){
        first = n;
        last = n;
    } else {
        last.setNext(n);
        n.setPrev(last);
        last = n;
    }
}
```
4) Below are some class and method definitions. `hasDuplicates` returns true if and only if its list parameter contains links with duplicate values.

```java
public class Node{
    public String value;
    public Node next;
    public Node(String str){
        value = str;
        next = null;
    }
    public void setNext(Node node){
        next = node;
    }
    public boolean hasNext(){
        return (next != null);
    }
}

public boolean find(Node n, String s){
    if (n.value == s) return true;
    if (n.hasNext()){
        return find(n.next, s);
    } else return false;
}

public boolean hasDuplicates(Node n){
    if (n.hasNext()){
        if (find(n.next, n.value)) return true;
        else hasDuplicates(n.next);
    } else return false;
}

Write a recurrence relation and its big-Oh solution for `hasDuplicates` where \( T(n) \) is the time for `hasDuplicates` to execute on an n-node list. Assume all necessary packages are imported. Justify your answer.

\[
T(n) = O(n) + T(n-1)
\]
\[
T(1) = O(1)
\]