Question 1. [10 MARKS]

Consider the following Java code:

```java
public class IntNode {
    public IntNode next;
    public int data;
}
```

In this question you will write a `public static` method called `deleteLast` for the `IntNode` class. This method should take a single `IntNode` as a parameter, representing the start of a linked-list, and its return type should be `void`. You may assume that the `IntNode` parameter represents a linked-list with more than one element.

**a) Implement `deleteLast` as an iterative method. Include the method header and an appropriate comment (Javadoc is not necessary). [5 MARKS]**

```java
// traverse the linked-list until the second-last node is reached, then delete the last one.
public static void deleteLast(IntNode node) {
    while(node.next.next != null) {
        node = node.next;
    }
    node.next = null;
}
```

**b) Implement `deleteLast` as a recursive method. Include the method header and an appropriate comment (Javadoc is not necessary). [5 MARKS]**

```java
// recurse until the list is length 2 - then delete the last node and return.
public static void deleteLast(IntNode node) {
    if(node.next.next == null) {
        node.next = null;
        // no return necessary
    } else {
        deleteLast(node.next);
    }
}
```
Question 2. [10 MARKS]
Assume that each of the following operations is implemented using the most efficient (in the Big-Oh sense) algorithm.
For each, give the worst-case time complexity in Big-Oh (using the smallest, simplest expression), and give a BRIEF explanation of why this performance is produced.

a) Determine whether an unsorted linked-list of length \( n \) contains any duplicate entries. \([2 \text{ MARKS}]\)
Runtime efficiency: \(O(n^2)\)
Explanation:
An element must be compared to each subsequent element, which is \(O(n)\), and there are \( n \) elements.

b) Find the \( m \)th element in a sorted linked list of \( n \) items. (Assume \( m \) is less than \( n \)). \([2 \text{ MARKS}]\)
Runtime efficiency: \(O(m)\)
Explanation:
\( m \) elements need to be traversed sequentially, regardless of \( n \).

c) Determine whether the value \( n \) is a power of 2. \([2 \text{ MARKS}]\)
Runtime efficiency: \(O(\log n)\)
Explanation:
Can be determined by repeatedly dividing \( n \) by 2 until a number \( \leq 1 \) is reached – this takes \( O(\log n) \) divisions.

d) Find the value that occurs most often in a sorted array of \( n \) elements. \([2 \text{ MARKS}]\)
Runtime efficiency: \(O(n)\)
Explanation:
Because the array is sorted, duplicate elements occur together, so in a single pass we can keep track of the most frequent element so far while counting the current element.

e) Print the \( m \)th element of an array of length \( n \). (Assume \( m \) is less than \( n \)). \([2 \text{ MARKS}]\)
Runtime efficiency: \(O(1)\)
Explanation:
Any element in an array can be accessed directly, in constant time.
**Question 3.** [10 MARKS]
The following Java program compiles properly. In the box provided, write the output after running the main method.

```java
public class ExceptionTrace {
    public static void main(String[] args) {
        A a = new A(2);
        B b = new B(2);
        try {
            f(2, a);
            f(2, b);
            f(1, b);
            System.out.println("Done");
        } catch (Exception e) {
            System.out.println("Oops");
        }
    }

    public static void f(int i, A a) throws Exception {
        a.m2(i);

        if (i % 2 == 0) {
            a.m(i);
        } else {
            ((B)a).m();
        }

        System.out.println("End of f.");
    }
}

public class A {
    private int r[];

    public A(int x) { r = new int[x]; }

    public int m(int i) {
        System.out.println("A.m: i="+i);
        return r[i-1];
    }

    public void m2(int i) {
        System.out.println("A.m2: " + r.length);
    }
}

public class B extends A {
    public B(int x) {
        super(x);
    }

    public void m2(int i) {
        System.out.println("B.m2: i="+i);
        super.m(2*i);
    }

    public void m() throws Exception {
        System.out.println("B.m");
        throw new Exception();
    }
}
```

**Output:**

```
A.m2: 2
A.m: i=2
End of f.
B.m2: i=2
A.m: i=4
Oops
```