1) $h(K) = K \bmod 7$

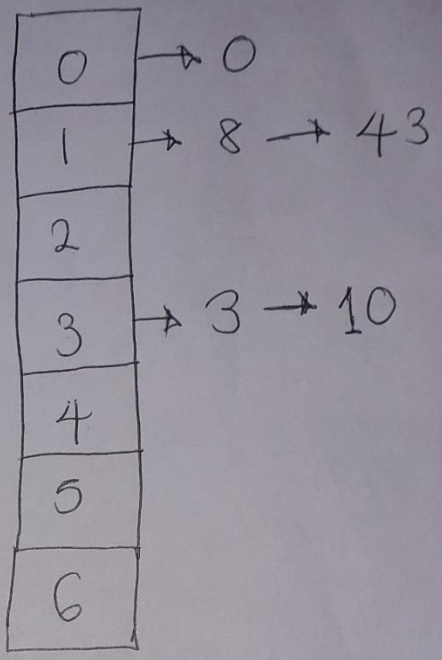
$h(3) = 3\%7 = 3$

$h(10) = 10\%7 = 3$

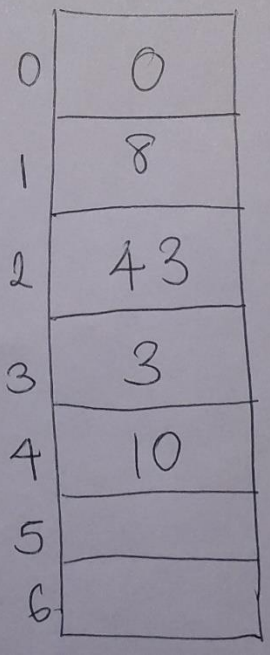
$h(8) = 8\%7 = 1$

$h(43) = 43\%7 = 1$

$h(0) = 0\%7 = 0$



2)



3) $h(K) = K \bmod 7$ $\quad h'(K) = 5 - (K \bmod 5)$

$h(3) = 3$

$h(10) = 3 + h'(10) = 8$ index 4

$h'(10) = 5 - (10 \% 5) = 5$

$h(8) = 1$

$h(43) = 1 + h'(43) = 3$ index 4 $+ h'(43) = 5$ index 6

$h'(43) = 5 - (43 \% 5) = 2$

$h(0) = 0$

0	0
1	8
2	
3	3
4	10
5	
6	43

4. $f(n) = f(n-2) + c_1 n + c$ for $n > 0$

$f(n) = f(n-2) + n$.

After removing constants the time complexity of $f(n) = O(n)$.

∴ the time complexity of $f(n-2)$ should also be of order n because 2 is a constant.

5.(i)

```
public class Odd {

	public int numOdd(Node n) {

		if (n.isLeaf()) {
			return 0;
		}

		int oddNodes = 0;
		int childCount = n.numChildren();
		Node[] children = n.getChildren();

//		check current node
		if (childCount % 2 == 1) {
			oddNodes++;
		}

//		check children
		for (Node node : children) {
			oddNodes += numOdd(node);
		}
		return oddNodes;
	}
```

}

5(ii)

Let all external loop operations $= C_1$

all internal loop operations $= C_2$.

C_2 is within a loop that executes (# of degree of node) times $= n-1$

$\therefore f(n) = C_1 + C_2(n-1)$ excluding recursive call.

The recursive call is invoked once every node.

$\therefore$ the function has to be added n times

$= \sum^{n} C_1 + C_2(n-1)$

After removing constants, we have to add $n-1$, n times.

$= n(n-1) = n^2 - n$

n^2 is larger $\therefore$ the function is $O(n^2)$

6)

let all external operations $= C_1$

let all internal while loop operations $= C_2$.

$= C_1 + C_2\left(\frac{n(n+1)}{2}\right)$

$= C_1 + C_2\left(\frac{n^2+n}{2}\right)$

$= C_1 + \frac{C_2 n^2 + Cn}{2}$

Eliminate Constants

$n^2 + n$

∴ The function is $O(n^2)$