CS2210 Data Structures and Algorithms
Solution for Concept Assignment 2

1. (1 mark) When separate chaining is used, the hash table will look like this:

0	$\rightarrow$ 0
1	$\rightarrow$ 8 $\rightarrow$ 43
2	
3	$\rightarrow$ 3 $\rightarrow$ 10
4	
5	
6	

2. (1 mark) Linear probing

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

3. (4 marks) Double hashing

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

4. We solve the equation using repeated substitution. Remember that n is even.

$$\begin{aligned}
&f(n) = f(n-2) + c_1 n + c_2 \\
&f(n-2) = f(n-4) + c_1(n-2) + c_2 \\
&f(n-4) = f(n-6) + c_1(n-4) + c_2 \\
&\quad\vdots \\
&f(4) = f(2) + c_1 \times 4 + c_2 \\
&f(2) = f(0) + c_1 \times 2 + c_2 \\
&f(0) = c_0
\end{aligned}$$

Substituting the value of $f(0)$ in the equation for $f(2)$, then substituting the value of $f(2)$ in the equation for $f(4)$, and so on we get:

$$\begin{aligned}
f(n) &= c_1 n + c_2 + c_1(n-2) + c_2 + c_1(n-4) + c_2 \cdots + c_1 \times 4 + c_2 + c_1 \times 2 + c_2 + c_0 \\
&= \sum_{i=1}^{n/2} (c_1 \times 2i + c_2) + c_0 = \frac{\frac{n}{2}(\frac{n}{2}+1)}{2} \times (2c_1) + c_2 \frac{n}{2} + c_0 \\
&= \frac{c_1}{4} n^2 + \left(\frac{c_1}{2} + \frac{c_2}{2}\right) n + c_0
\end{aligned}$$

Discarding multiplicative constants, and then getting the larger function between n^2 and n we get that $f(n)$ is $O(n^2)$.

5.(*i*) **Algorithm** `numOdd` (r)

In: *Root r of a tree*
Out: *Number of internal nodes with odd degree*

```
if r.isLeaf() then return 0
else {
    odd = 0
    children = r.getChildren()
    if (r.numChildren() % 2) == 1 then ++odd
    for (Node u : children)
        odd ← odd + numOdd(u)
    return odd
}
```

5.(*ii*) To compute the time complexity of the algorithm, first we ignore the recursive calls. In each invocation, the algorithm performs a constant number c of operations if the current node r is a leaf. If r is an internal node, then the **else** statement is performed. In each iteration of the **for** loop a constant number c' of operations is performed (ignoring recursive calls) and the loop is repeated degree(r) times. Hence, the total number of operations performed by the **for** loop is $c' \times \text{degree}(r)$. Outside the **for** loop an additional constant number c'' of operations is performed, so for an internal node r the algorithm performs $c' \times \text{degree}(r) + c''$ operations.

To take into consideration the recursive calls we need to understand what their purpose is. Observe that in the worst case the algorithm implements a traversal of the tree, so the effect of the recursive calls is to make the algorithm visit each node of the tree **once**. Hence, the algorithm performs one recursive call per node and so, the total number of operations performed by the algorithm is

$$\sum_{\text{leaves } u} c + \sum_{\text{internal nodes } u} (c' \times \text{degree}(u) + c'') = c \times \#\text{leaves} + c'' \times \#\text{internal nodes} + c' \sum_{\text{internal nodes } u} \text{degree}(u)$$

$$= c \times \#\text{leaves} + c'' \times \#\text{internal nodes} + c'(n-1)$$

Discarding constants, we get that the order of the time complexity is $O(\#\text{leaves} + \#\text{internal nodes} + n)$. Since $\#\text{leaves} + \#\text{internal nodes} = n$, the time complexity of the algorithm is $O(n)$.

6. Outside the while loop a constant number c_1 of operations is performed.

   In each iteration of the while loop a constant number c_2 of operations is performed. To calculate the number of iterations performed by the while loop, consider the following table. The values of i and j are the values that the variables have at the end of an iteration. So after iteration 1 the values of i and j are equal to 1, after the second iteration the value of i is 2 and the value of j is 3, and so on.

Iteration	i	j
1	1	1
2	2	$1+2$
3	3	$1+2+3$
4	4	$1+2+3+4$
$\vdots$		
$n-1$	$n-1$	$1+2+3+\cdots+n-1 = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$
n	n	$1+2+3+\cdots+n-1+n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

Therefore, the total number of iterations performed by the while loop is n and so the total number of operations performed by the algorithm is $f(n) = c_1 + c_2 n$ is $O(n)$.