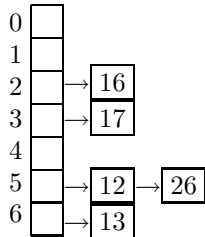


CS2210a Data Structures and Algorithms
Solution for Assignment 3

1. (2 marks) When separate chaining is used, the hash table will look like this:



2. (2 marks) Linear probing

0	26
1	
2	16
3	17
4	
5	12
6	13

3. (2 marks) Double hashing

0	
1	17
2	26
3	16
4	
5	12
6	13

4. (4 marks)

$$\begin{aligned}
 f(n) &= 2f(n-1) \\
 &= 2(2f(n-2)) = 2^2 f(n-2) \\
 &= 2^2(2f(n-3)) = 2^3 f(n-3) \\
 &\vdots \\
 &= 2^{n-2}(2f(n-(n-1))) = 2^{n-1} f(1) = 2^{n-1} \times 2 = 2^n
 \end{aligned}$$

Since $f(n) = 2^n$, then $f(n)$ is $O(2^n)$.

5. (3 marks)

Algorithm InternalLevel (r, k)

In: Root r of a proper binary tree and a positive integer value k

Out: Number of internal nodes at distance k from r .

```
if  $r$  is a leaf then return 0
else { //  $r$  is an internal node
    if  $k = 0$  then return 1
    else {
        nleft = InternalLevel (left child of  $r, k - 1$ )
        nright = InternalLevel (right child of  $r, k - 1$ )
        return nleft + nright
    }
}
```

(3 marks) The algorithm performs a traversal of the tree. Whenever a recursive call is made to visit the left or right subtrees of the current node the value of k must be decreased because the algorithm moves one level down the tree. Therefore, when the value of k is equal to zero, the algorithm has moved to a node at distance k from the root. If while at an internal node the value of k is zero, then that internal node is at distance k from r and so the algorithm must count it by returning the value 1.

Note that the statement

```
if  $k = 0$  then return 1
```

stops the algorithm from exploring the tree below level k , as in that part of the tree there cannot be any internal nodes at distance k from r .

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 is a leaf and it also performs a constant number c' of operations if the current node is an internal node. Now, to take into consideration the recursive calls we need to understand what their purpose is. Observe that the algorithm just implements a traversal of the tree, so the effect of the recursive calls is to make the algorithm visit all the nodes of the tree in the worst case (the worst case is when k is larger than the height of the tree, so the entire tree must be visited); furthermore, each node is visited at most once. Hence, the total number of operations performed by the algorithm is at most

$$c \times \text{number of leaves} + c' \times \text{number of internal nodes} = c \frac{n+1}{2} + c' \frac{n-1}{2}$$

Discarding constants, we see that the time complexity of the algorithm is $O(n)$.