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

```
0    
1   15
2    
3    
4    
5   33 \rightarrow 12 \rightarrow 47
6    
```

2. Linear probing:

```
0    
1   12
2   15
3   47
4    
5   5
6   33
```

3. Double hashing:

```
0   33
1   15
2    
3   47
4   12
5   5
6    
```

4. We solve the equation using repeated substitution.

\[
\begin{align*}
  f(n) &= f(n - 1) + n - 2 \\
  f(n - 1) &= f(n - 2) + n - 3 \\
  f(n - 2) &= f(n - 3) + n - 4 \\
  &\vdots \\
  f(3) &= f(2) + 1 \\
  f(2) &= f(1) + 0 \\
  f(1) &= 1 \\
\end{align*}
\]

Substituting the value of \( f(1) \) in the equation for \( f(2) \), then \( f(2) \) in the equation for \( f(3) \), and so on we get

\[
\begin{align*}
  f(n) &= (n - 2) + (n - 3) + (n - 4) \cdots + 1 + 0 + 1 = 1 + \sum_{i=1}^{n-2} i \\
        &= \frac{(n - 2)(n - 1)}{2} + 1 = \frac{n^2}{2} - \frac{3n}{2} + 2
\end{align*}
\]

Discarding constant terms, and then getting the larger function between \( n^2 \) and \( n \) we get that \( f(n) \) is \( O(n^2) \).
5.(i) **Algorithm** isSymmetric \((r)\)

**In:** Root \(r\) of a tree

**Out:** True if the tree is symmetric, or false if it is not

\[
\begin{align*}
\text{if } r \text{ is a leaf then return true} & \quad \text{// a leaf is a symmetric tree} \\
\text{else} \{ & \\
\quad u \leftarrow \text{first child of } r \\
\quad \text{For each child } u \text{ of } r \text{ do } \{ & \\
\quad\quad \text{if } u.\text{value} \neq u.\text{value} \text{ then return false} & \quad \text{// children of } r \text{ do not have same value} \\
\quad\quad \text{if isSymmetric}(u) = false \text{ then return false} & \\
\quad\} & \\
\text{return true} & \\
\}\end{align*}
\]

5.(ii)

- The worst case for the algorithm is when the input tree is symmetric, as in this case the conditions of the if statements inside the for loop will always be false, and so the algorithm will not terminate early.

- 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 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 \(O(n)\).