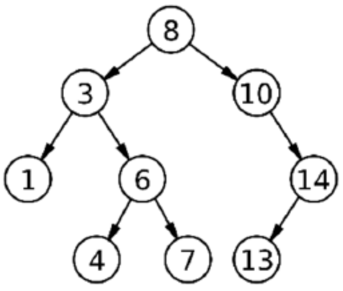
```
Assignment 3
© Mukesh Kumar Angrish
Written by: Mukesh Kumar Angrish (Student ID - 40203596)
```

Part I.

1. Write an efficient pseudo-code for a method named BottomUpLevelTraversal which is a variation of BST level order traversal. It is similar to the level order traversal; except, that nodes on deeper levels are visited before nodes on shallower levels. Your method must not modify the tree. Also, if the tree has n nodes, what is the running time of your new traversal method?

A sample tree where BottomUpLevelTraversal will print 4,7,13,1,6,14,3,10,8.



G: Graph with vertices and edges
Q: Queue, initially with source vertex (root of the tree in this case)
discovered: a list of all discovered/visited nodes in the graph

```
function recursiveBFS(G, Q, discovered):
    if Q.isEmpty() then
        return
    endif
    v := pollQueue(Q)

    for u in reverseList(AdjacentList(u)):
        if not discovered[u] then
            discovered[u] := true
            Enqueue(Q, u)
    recursiveBFS(G, Q, discovered)
    print(v)
```

2. You are given a hash table with size 7 and a hashing function h(k) = k%7. Output the state of the table after inserting the below mentioned values in that specific order when collisions are handled using:

The values to be inserted are 19, 26, 13, 48, 17.

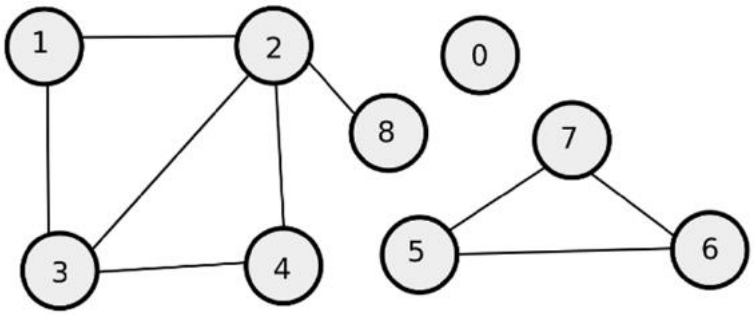
a. Separate chaining
h(0)={}
h(1)={}
h(2)={}
h(3)={17}
h(4)={}
h(5)={19, 26}
h(6)={13, 48}

b. Linear probing
h(0)={13}
h(1)={48}
h(2)={}
h(3)={17}
h(4)={}
h(5)={19}
h(6)={26}

c. Double hashing using a second hash function h’(k) = 5 – (k % 5)

This fails since the double hashing function returns the same index over and over for insertion of 26

3. Develop a well-documented pseudo code that determines number of connected components in any given graph. A graph is connected when there is a path between every pair of vertices. A graph G is said to be disconnected if there exist at least two nodes in G such that no path in G has those nodes as endpoints. A connected component (or just component) of a graph is a sub graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super graph. For instance, the below mentioned graph will yield 3 as it has three connected components.



```
vertex: source vertex
visited: a list of all discovered/visited nodes in the graph
vertexList: list of vertices in a single connected component
function DFSRecursive(vertex, visited, vertexList):
        visited[v] := true
        add(vertexList, v)
        while u in adj(v) do
                if not visited[u] then
                        DFSRecursive (vertex, visited, vertexList)
                endif
        endwhile
```

```
G: Graph with vertices and edges
visited: a list of all discovered/visited nodes in the graph
components: list of connected components
function connectedComponents(G, visited):
        components := emptyList()
        foreach vertex in G do
                vertexList := emptyList()
                if not visited[vertex] then
                        DFSRecursive(vertex, visited, vertexList)
                add(components, vertexList)
            endif
        endfor
        return size(components)
```

Part II.

Watchable.java is an interface for objects that can be watched.

The TVShow.java program defines a TVShow class with different attributes corresponding to the information regarding a TV Show. The TVShow class also has multiple accessors and mutators in order to provide flexibility during object creation and modification. There are also methods to convert object to string and to compare different TVShows objects (such as equals and isOnSameTime)

The ShowList.java class implements the ADT list by using a chain of nodes about TVShow objects. The ShowList class also has multiple accessors and mutators in order to provide flexibility during object creation and modification. There are also methods to convert object to add, remove and modify the nodes of the list.

ProcessWishlist.java uses ShowList class to process files and produces list of shows whether or not there are conflict between the shows a user is watching and wishes to watch.