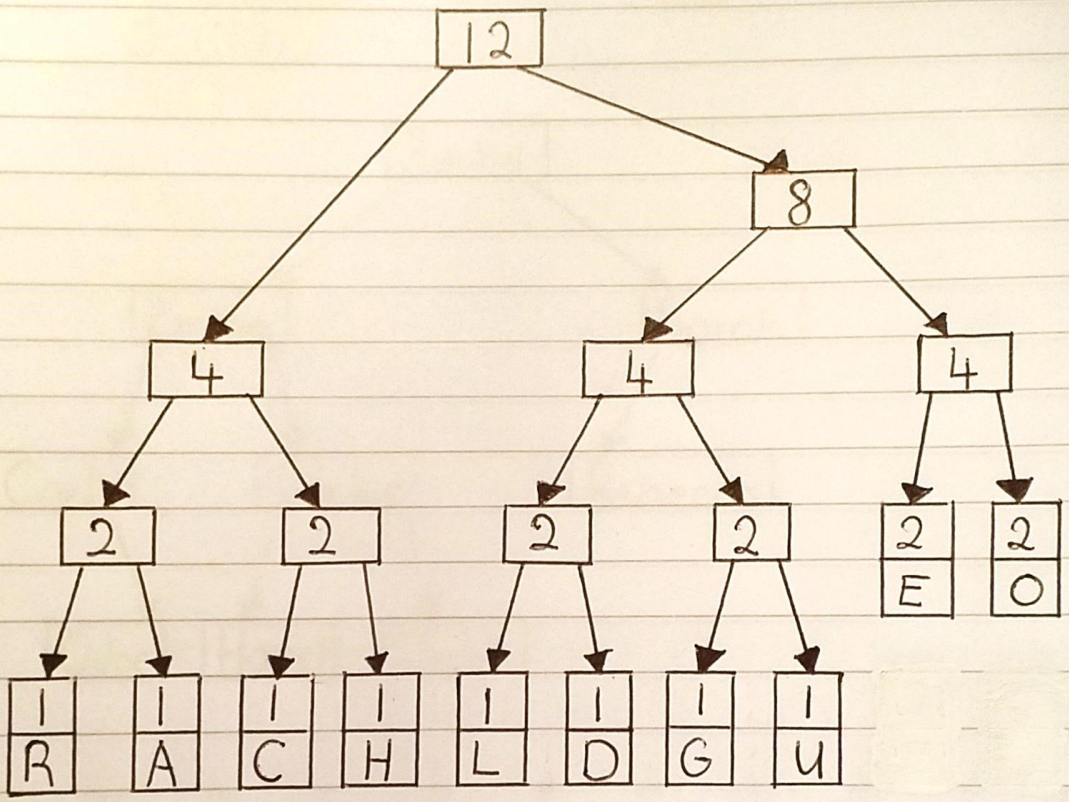
Rachel Doogue , C00237335

Q1. R = 1 A = 1 C = 1 H = 1 E = 2 L = 1

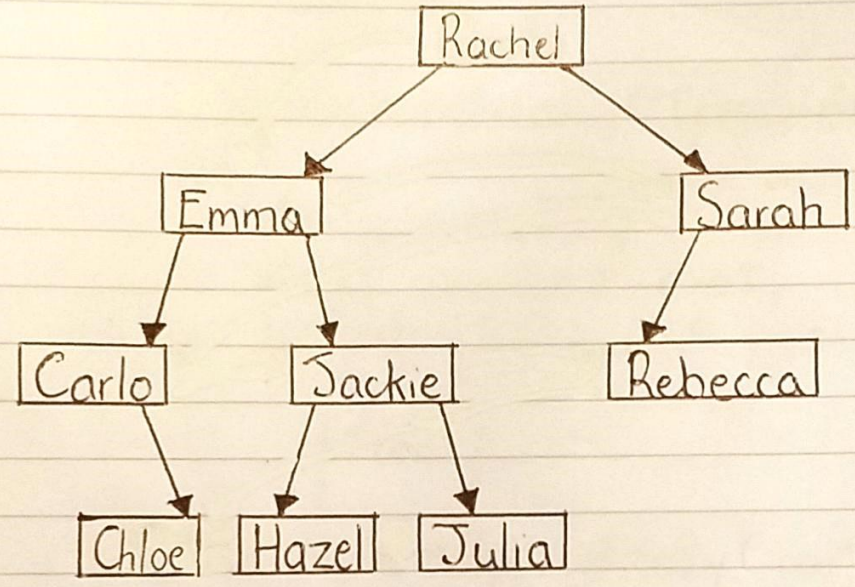
D = 1 O = 2 G = 1 U = 1



Note: I did not include 'space'

Rachel Doogue , C00237335

Q2a. 1. Rachel
2. Emma
3. Carlo
4. Jackie
5. Chloe
6. Sarah
7. Hazel
8. Rebecca
9. Julia



Rachel Doogue, C00237335

Q2b.

```
struct treenode
{
  String data;
  struct treenode *lchild;
  struct treenode *rchild;
}Treenode;

SortAlphabetically(Treenode * root)
{
  string array[9];
  string pointer = root;
  for(int i = 0; i<9; i++)
  {
    if(root)
    {
      SortAlphabetically(root -> lchild)
      root -> data
      for(int j = 0; j<9; j++)
      {
        if(data == array[j]
        {
          data = Null
        }
        else
```

Rachel Doogue, C00237335

Q2c.

```
struct treenode
{
  String data
  struct treenode * lchild
  struct treenode * rchild
}

int LeafCount(treenode * root)
{
  if(root == Null)                                          O(n)
  {
    return 0
  }
  else if(root -> lchild == Null && root -> rchild == Null)  O(logn)
  {
    return count++
  }
  else                                                      O(n^2)
  {
      LeafCount(root -> lchild)
      LeafCount(root -> rchild)
  }
}

output count
```

Count = 4

Rachel Doogue, C00237335

Q2d.

```
struct treenode
{
    String data
    struct treenode * lchild
    struct treenode * rchild
}

String HighestLeafValue (treenode * root)
{
    if (root == Null)                                          O(n)
    {
        return 0
    }
    else if (root → lchild == Null && root → rchild == Null)   O(logn)
    {
        if (root → data > highestValue)                        O(n)
        {
            highestValue = root → data
        }
    }
    else                                                       O(n²)
    {
        HighestLeafValue (root → lchild)
        HighestLeafValue (root → rchild)
    }

    output highestValue
```

Highest Leaf Value = Rebecca

Note: initalise highestValue as String

Rachel Doogue, C00237335

Q2e.

```
struct treenode
{
    String data
    struct treenode * lchild
    struct treenode * rchild
}

void
~~String~~ FindName (treenode * root)
{
O(n)    if (root == Null)
        {
            return 'Not Found'
        }
O(n)    else
        {
O(n)        if( findName== root)
            {
                return findName +'Found'
            }
O(logn)     else if (findName < root)
            {
                FindName(root->lchild)
                return 'left,'
            }
O(n^2)      else if(findName > root)
            {
                FindName(root -> rchild)
                return 'right,'
            }
        }
```

Find: Jackie Output: left, right, Jackie Found

Rachel Doogue, C00237335

Q2f. SortAlphabetically =

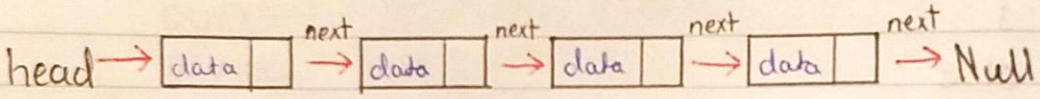
LeafCount = $O(n^2)$

HighestLeafValue = $O(n^2)$

FindName = $O(n^2)$

Rachel Doogue, C00237335

Q3. To create a list for my friends telephone numbers I used a singly linked list. This is a list of nodes. Each contains data and the address of the next node. Each node points to one list element. Singly linked lists are unidirectional.



Pseudocode

```
AddFriend ()
{
    cin string
    node current = head
    create newnode
    while (current ->next != Null)
    {
        current = current → next
    }
    newnode = string
    current -> next = newnode
}
```

- FindFriend()

```
{
  cin string
  node current = head
  while (current -> next != Null)
  {
    if (current -> data == string)
    {
      output  current -> data
    }
    else
    {
      current = current -> next
    }
  }
}
```

- Print()

```
{
  node current = head
  while (current != Null)
  {
    cout current -> data
    current = current -> next
  }
}
```

```
DeleteFriend()
{
  cin string
  node current = head
  while(current -> next != Null)
  {
    if (current -> data == string)
    {
      prev = current
      current = current -> next
      string = current -> data
    }
  }
}

UpdateFriend()
{
  cin stringFind
  cin stringRplacement
  node current = head
  while (current -> next != Null)
  {
    if(current -> data == stringFind)
    {
      current -> data = stringReplacement
    }
    current = current -> next
  }
}
```

- Size()

```
{
node current = head
int count
while (current != Null)
{
    current = current → next
    count++
}
cout count
}
```

Summary

- AddFriend() will a friend/contact to the end of the linked list.
- FindFriend(), when you enter a friend it will search for that friend in the linked list. Once found it returns the data from the node.
- Print() will output the data from each node.
- DeleteFriend() lets you enter a friends details to be deleted when found. All data is then shifted left.
- UpdateFriend() takes in the original details and the new details. It searches for the original details. Once it is found it is replaced with the new details.
- Size() outputs the number of contacts in the linked list.