Algorithm Description and Algorithm Analysis

The add member problem can be split into two sections. Section A is an algorithm that will search for the appropriate location for the new member to be added. Section B simply moves the members who fall after the new member down the list and inserts the new member at the specified location before returning the result.

ALGORITHM Add(Member)

//Add members to a collection and maintain alphabetical order in a pre-existing MemberCollection object
//Input: An object of Member class
//Output: Array A[0..n-1] that contains all the n members in the MemberCollection object and the jobs are sorted in alphabetical order by full name

```
A ← MemberCollection
v ← MemberCollection.count / 2
While B do S
    if v is v.last
        B ← false
    if A[v] > Member
        v ← v + v / 2
    elseif A[v] < Member
        v ← v – v / 2
    else
        return A
N ← v
For v ← N to M do
    A[v+1] ← A[v]
A[w] ← Member
return A
```

Algorithm analysis:
1. The input parameter Member is a parameter that contains the important information regarding the member who is to be added to the existing MemberCollection object.
2. Determining if "*A*[v] >/< Member" is the main part of this algorithm. Essentially this will determine if the member goes before or after a point "v" which shall be determined as the halfway point of the sector being tested. Practically this operation can be done once per iteration using a switch case statement.
3. The number of times this basic operation occurs will vary depending on various factors including size of the original collection and how far from segment boundaries the desired location will be.

4. Thus, the efficiency of the most complex part of the Add member algorithm is O(log n) as C(n) = 1.4log(1.7n)+1.5
5. In the more simple part of the algorithm which simply adds the member to the location found in the more complex part of the algorithm the efficiency would be described as O(n) where n is the number of members who need to be moved to make space for the new member.
6. Therefore the total complexity of this algorithm should be described as O(log n) + O(n) = O(n)