CAB301 – ALGORITHM DESCRIPTION AND ANALYSIS

Assessment 1

Matthew Chambers – n11318546

Add Algorithm Description

The algorithms functionality can be separated into four distinct steps. First, before we continue in adding a member to collection we must first check that the collection is not full. The second step in the algorithm is another check to ensure that the member wanting to be added to the collection is not already in it, to avoid duplicates. With these checks in place, we can use a sorted approach to determine the correct position where the new member should be inserted into the collection. In the final step, it shifts members to the right to make room for the member being added, then adds them to the collection.

Within the add algorithm, a call to a linear search method is used as a pre check to insertion to ensure that the member wanting to be added is not already in the collection. This algorithm works by first utilising a sequential comparison which iterates through the member collection. Then, for each of the members present in the array it checks against the last name and first name. If a match is found the algorithm returns true which when being used in the add method does not allow for the member to be added since it is a duplicate. If no match is found, the algorithm returns false so that the Add method can continue in adding the member to the collection.

```
ALGORITHM Add(member)
    if not IsFull()
        if not Search(member.LastName, member.FirstName)
            i ← 0
            while i < count and members[i].CompareTo(member) < 0
                i ← i + 1

            j ← count - 1
            while j >= i
                members[j + 1] ← members[j]
                j ← j - 1

            members[i] ← member
            count ← count + 1

ALGORITHM IsFull()
    return count = capacity

ALGORITHM Search(lastname, firstname)
    i ← 0
    while i < count
        if members[i].LastName = lastname and members[i].FirstName =
firstname
            return true
        i ← i + 1
    return false
```

Add Algorithm Analysis

1. The input parameter ‘member’ represents the member to be added to the collection. Additionally the parameter ‘count’ defined in the ‘MemberCollection’ class is used to set the problem size as it indicates the current amount of members in the collection.
2. the comparison operation within the search method is the basic operation of this algorithm as it checks to see if the member wanting to be inputted to the collection already exists within it. It is this operation that it has the most influence on the computation time as it determines if the member will not be added, significantly decreasing the run time.
3. The number of times this basic operation occurs is dependent on the number members already in the collection. If the member is already in the collection, then the operation will only occur once. If the member is not already in the collection, then the basic operation will occur for however many members there are in the collection or ‘count’ times.
4. Given that the amount of comparisons grows linearly with the amount of members in the collection, it can be determined that the big O notation of the Add algorithm is O(n) where n is the number of members in the collection.
5. If the collection is sorted, a potential improvement to the algorithm would be to utilise a binary search in favour of a linear search to improve computation time. Additionally, another improvement would be to use a different data structure for the member collection that automatically adjust its size negating the need for the shifting process.
6. Therefore, for its current implementation the add algorithm is suitable and provides a good time complexity of O(n). However, for large collections it would be in the best interest to implement further optimisations to ensure running time is of an acceptable level.