1

How well did the data structures perform for the assigned application?
The ADT implented in this assignment was the Binary Search Tree. The assigned application was inserting, removing and retrieving data of an oscar nominee type. The oscar nominee data is a struct consisting of a character array and number of votes of the character array, i.e. nominee votes. The data structure performed these assigned tasks with better efficiency compared to some other data structures used in this class. Compared to a linear linked list the sequential search with tail optimization still performs in linear time, however the binary search tree performs in a logarithmic time efficiency when doingthe remove and retrieve operation. In fact other than the in-order traversal the binary tree performs much better than a linear structure, given enough data items.

2

Would a different data structure work better? If so, which one and why.
Given the problem type to create a data set of oscar nominees, the better data structure might be to use a hash table where the category can be the basis of the hash function. Since the categories are a finite set, the hash table would be fully used in memory. Each category can correspond to a different index in a hash table. Since the hash table performs the adding a new nominee in $O(1)$ given that all nominees are known and that the collision resolution technique used is quadratic probing.

3

What was efficient about this program design?
Using a binary tree the efficiency relied on the fact that the use of memory in the heap was efficient. In that they do not reserve more memory than needed. Compared to a linked list with a one pointer, the binary tree may use an extra child pointer but the data type has better algorithms for performing tasks of retrieving, inserting and removal. This efficiency only works in the best case scenario, which in this case is given that the data inserted creates a full binary tree that is height balanced. Given that the height of a binary tree is, on average $\lceil \log_2(n+1) \rceil$ the average number of comparisons for removing, inserting and retrieving also are $O(\log(n))$ since it is binary data structure using a binary search. Since the data provided is searched based on alphabetical order, the data when inserting or searching has to be different enough (alphabetically compared) that the tree is balanced for example, if the nominees for the best movie were, Matrix, Matrices, Matlock, this would be a linked list.

4

What was not efficient about this program design?
Continuing from the previous question given that the data is sorted alphabetically, the operations of retrieving, inserting, removing and traversing are all performed in the worst case function, which is a linear time. This then becomes a linked list with an extra pointer over head per node. Thus the data has to be sufficiently randomized. Further the implentation of each function is recursive, so to make sure that stack over flow does not occur tail call optimizations have to made so to that it will retain the same efficiency as iteration.

5

What would you do differently?
In this assignment some recursive functions do not have tail call optimization so adding tail-optimization would be implemented. A get height function would also be useful to check if a tree is balanced, so checking if the left height of a subtree matches the right subtree height. Further adding functions like getting a user to write to a file first might be a better way to read in data to binary tree. For example once a user writes

form continued on next page...

to a file adding the nominee and number of votes, the file can then be read randomly so for large trees, this might be a better way to maintain the data. In applications like storing movies in a database with a binary tree structure, reading a file with a sorting algorithm to minimize the height of the tree would ensure the logarithmic time performance.