**Assignment\_PS10Q1 Design Document**

**The basic structure of the player node is:**

*Class PlayerNode:*

def \_\_init\_\_(self, Pid):

self.PId = Pid

self.attrCtr = 1

self.left = None

self.right = None

The binary tree can be chosen to be either balanced or imbalanced. If we choose balanced then the time complexity of Insert is O(n), whereas in an imbalanced binary tree Insert will be O(1) if we just insert the new node at the root.

1. **def \_recordSwipeRec(self, pNode, Pid):**

This function is called for every entry in the input file “**inputPS10Q1.txt** ”. We traverse the tree recursively (pre-order) and compare the input with the PId of nodes, if a match is found we increment the counter attribute of that node.

If the player id is not found in the Binary tree then we need to insert a new node with PId set to input and counter attribute initialized to 1

In the worst case, the function has to traverse the complete tree, and when a match is not found then insert the new node

Time Complexity = Traversal + Insert Operation

= O(n) + O(n)(balanced BT)/O(1)(Imbalanced BT)

= O(n)(worst case)

1. **def \_getSwipeRec(self, pNode):**

Once the BTree is prepared, an entry must be made in the output file about the number of players that came to the hotel today.

This information can be obtained by counting the no of nodes in the binary tree. This requires traversal of the tree, hence the time complexity of this step is O(n).

1. **def \_onPremisesRec(self, pNode, PId):**

This function allows us to check how many players are still on-premises.

This can be achieved by traversing the complete tree and counting the number of nodes for which the counter attribute is odd.

Checking the value of the Counter for all nodes leads to O(n) time complexity

1. **def \_checkEmpRec(self, pNode, EId):**

This function is called when the checkPlay tag is encountered in the Prompts file.

It counts the number of times a particular player swiped today and if the player is currently in hotel or outside.

For search, the whole tree has to be traversed in the worst case. [O(n)]

Then the counter value has to be checked to determine the whereabouts of the player [constant time]

Hence the time complexity of the function is O(n) (worst case)

1. **def \_frequentVisitorRec(self, pNode, frequency):**

This function is called when the freqVisit tag is encountered in the Prompts file.

It generates the list of players who have swiped at least x number of times.

This function also requires tree traversal to check the counter value of each node.

The traversal takes O(n) time and counter value comparison constant time.

Hence the time complexity of this function is O(n) as well.

1. **def printRangePresent(self, StartId, EndId):**

This function is called when the Range tag is encountered in the Prompts file.

It prints the player ids in the range **StartId** to **EndId** and how often they have swiped and if they are inside or outside the hotel.

This function requires Inorder tree traversal to check the PId value of each node.

The traversal takes O(n) time and Pid value comparison constant time.

In the sample, output nodes are printed in sorted order(PId increases as we go down).

The nodes are sorted on the value of PId. O(nlogn).

Hence the time complexity of this function is O(nlogn) if sorted output is required, otherwise O(n).

\*\* This function could be more efficiently implemented if BST was used as inorder traversal produces a sorted result, but it was mentioned in the question to use Binary Tree.