For all of below queries, heap is build for “n” swipes entries. So time complexity of this tasks is also needs to be considered along with time complexity analysis for performing queries.

1. Upheap Time (T1) = O(log n)
2. Heap Build Time (T2)
   1. Upheap is called after inserting every node. So total “n” calls will be required.
   2. T2 = O(nlog n)
3. How many employees came today (T3)
   1. It requires traversing entire list of “n” node sequentially towards end of list.

def getSwipeRec(self,eNode):  
 if eNode is not None : -------------------🡪 1  
 return 1 + self.getSwipeRec(eNode.right)---🡪 1 + (N -1) + 1 + 1  
 return 1------------------------------------------------🡪 1

* 1. Running Time Complexity T3 = 1 + 1 + 1 + (N-1) + 1 + 1 **= N-4 ~ O(n)**

1. Number of employees that have come today and are currently in office?
   1. Here also entire list of “n” nodes to be traversed sequentially and additional time to check and validate swipe count (attCtr) values for each node.

def onPremisesRec(self, eNode):  
 if eNode is not None: --------------------------------- 🡪 1  
 if eNode.attCtr % 2 == 0: -------------------------- 🡪 1 + 1  
 return self.onPremisesRec(eNode.right)-----🡪 1 + (N-1) + 1  
 else:  
 return 1 + self.onPremisesRec(eNode.right)-🡪 1 + (N-1) + 1 + 1  
 return 1----------------------------------------------------🡪 1

* 1. Time Complexity T4 = 1+1+1+(N-1)+1+1+1+(N-1)+1+1+1 **= 2N-7 ~ O(n)**

1. Check specific employee whereabouts?
   1. For specific employee this function is getting called.
   2. In each call, it traverse the list and compare Node EMPID and get swipe counter.

def checkEmpRec(self, eNode, EId):  
 if eNode is not None : ---------------------------------------🡪 1  
 if eNode.EmpId != EId:------------------------------------🡪 1 + 1  
 return self.checkEmpRec(eNode.right,EId)--------🡪 1 + (N-1) + 1  
 else:  
 return eNode.attCtr-------------------------------------🡪 1 + 1  
 return 0----------------------------------------------------------🡪 1

* 1. Time Complexity For single call = 1 + 1 + 1 +1 + (N-1) + 1 + 1+1 = N-6 ~ O(n)
  2. Best Case if employee id is the root node and got matched in first call ~ O(1)
  3. Worst case if employee id is at the leaf node ~ O(n)

1. List of employees that have swiped (in or out) more than x number of times?
   1. This function require traversing entire list.
   2. In each call,
      1. it compare data node (swipe counter) with input.
      2. Return Tuple which gets appended to List

def frequentVisitorRec(self,eNode,visit):  
 if eNode is not None :  
 if eNode.attCtr > visit:  
 return [(eNode.EmpId, eNode.attCtr)] + self.frequentVisitorRec(eNode.right, visit)🡪  
 else:  
 return self.frequentVisitorRec(eNode.right, visit)  
 return []

* 1. Time Complexity will be O(n)

1. Which employee ids within a range of IDs attended office, the swipe counter for them, and whether they are inside or outside office?
   1. This function uses queue representing max heap tree for inorder traversing
   2. Time Complexity depends
      1. Size of range
      2. Find Employee IDs in those range and get Swipe count.
   3. In worst case time complexity of this function will O(n2) where range includes all elements and employee id is the leaf node.