**Algorithm Design:**

We are implementing greedy algorithm to select the mobile ‘parts manufacturing’ and ‘assembling’ in such a way that total production time is minimized.

In AVL tree, every time a new data node is inserted, rebalancing is done to ensure tree height is maintained, hence time complexity always remains O(log N). Therefore, greedy algorithm has been implemented using AVL tree.

Our algorithm constitutes 2 parts:

1. Inserting data into AVL tree.
2. Calculate Production time and idle time.

**AVL Tree Structure:**

**MobileNode**:

MobileID

PartsManufTime

AssembleTime

Left Node

Right Node

**Algo part 1 - Inserting data into AVL tree:-**

Read file ‘InputPS1.txt’ and perform below algorithm repeatedly to insert each data record into AVL tree with greedy approach:

(In AVL Tree, while inserting data nodes, we are considering parts manufacturing time as 'key' here)

**insert(MobileID, PartsManufTime, AssembleTime):**

/\* Accept the current Mobile Tree \*/

tree 🡨Current MobileNode

/\* Create new node as per new Mobile details \*/

newMobileNode 🡨 MobileNode(MobileID, PartsManufTime, AssembleTime)

if tree is None:

MobileNode <-- newMobileNode // Set as root node

else if PartsManufTime < tree.PartsManufTime:

//Attach new mobile to left subtree

MobileNode.left.**insert**(MobileID, PartsManufTime, AssembleTime)

else if PartsManufTime > tree.PartsManufTime:

//Attach new mobile to right subtree

MobileNode.right.**insert**(MobileID, PartsManufTime, AssembleTime)

else:

// If Parts Manufacture Time is equal, then decide where to add node based on Assemble time

if AssembleTime > tree.AssembleTime:

MobileNode.left.insert(MobileID, PartsManufTime, AssembleTime)

else:

MobileNode.right.insert(MobileID, PartsManufTime, AssembleTime)

/\* After every new Mobile node insert to mobileTree, check for AVL tree balance.

**rebalance**()

**rebalance():**

**update\_heights**() // Update the height after addition of each node.

**update\_balances**() // Update the balances to see if Tree is balanced or not after addition.

while balance < -1 or > 1:

if balance > 1:

MobileNode.left.balance < 0: //then, we are in case II and need to do left rotate first

MobileNode.left.**leftrotate**()

**update\_heights**()

**update\_balances**()

**rightrotate**() // becomes case I, do right rotate

**update\_heights**()

**update\_balances**()

if balance < 1:

MobileNode.left.balance > 0: //we are in case III and need to do right rotate first

MobileNode.left.**rightrotate**()

**update\_heights**()

**update\_balances**()

**leftrotate**() // becomes case IV, left rotate.

**update\_heights**()

**update\_balances**()

**update\_heights():**

if not MobileNode == None:

height = max(MobileNode.left.height, MobileNode.right.height) + 1

else:

height = -1

**update\_balances():**

if not MobileNode == None:

balance = MobileNode.left.height - MobileNode.right.height

else:

balance = 0

**lrotate():**

# Rotate left pivoting on self

A <- MobileNode

B <- MobileNode.right.MobileNode

T <- B.left.MobileNode

MobileNode <- B

B.left.MobileNode <- A

A.right.MobileNode <- T

**rrotate():**

# Rotate left pivoting on self

A <- MobileNode

B <- MobileNode.left.MobileNode

T <- B.right.MobileNode

MobileNode <- B

B.right.MobileNode <- A

A.left.MobileNode <- T

**Algo part 2 - Calculate production order, production time and idle time:-**

productionOrder = [] // Captures the order in which mobiles should be produced to optimize production time as per Greedy algorithm.

productionTime = [] // captures the production time.

idleTime = [] // captures the idle time of assembly unit.

//inorder\_traverse will return list of all mobile nodes in parts manufacturing time order

Mobilelist = **inorder\_traverse**(tree)

for mobile in Mobilelist:

productionOrder.append(mobile.MobileID)

partsRunningTime += mobile.PartsManufTime

if productionTime < partsRunningTime:

#Assembling unit will be idle. Calculating idle time.

currentIdleTime = partsRunningTime - productionTime

productionTime += mobile.AssembleTime + currentIdleTime

idleTime += currentIdleTime

else if productionTime >= partsRunningTime:

productionTime += mobile.AssembleTime

**inorder\_traverse**(MobileNode tree):

{

if (tree == null):

return []

inlist[] = [] // take the list

if (tree != null) {

inorder(tree.left); // process left node first as inorder traversal does

process tree // process parent to add mobile nodes into inlist[]

inorder(tree.right);

}

return inlist[];

}

Print('Mobiles should be produced in the order: ' + str(productionOrder)[1:-1] + '\n')

Print('Total production time for all mobiles is: ' + str(productionTime) + '\n')

Print('Idle Time of Assembly unit: ' + str(idleTime) + '\n')

**Time complexity of algorithm:**