***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

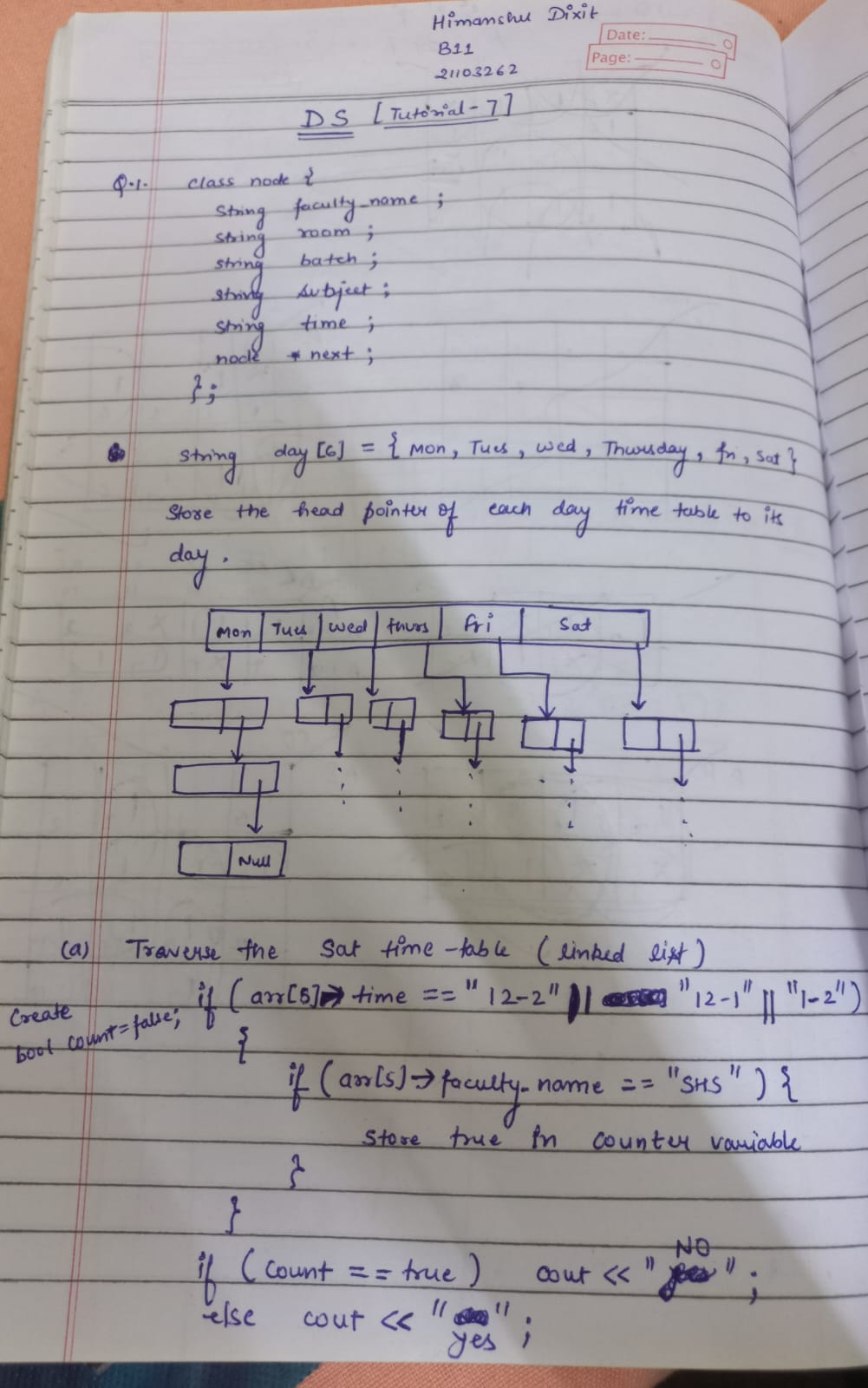
***BATCH : B11***

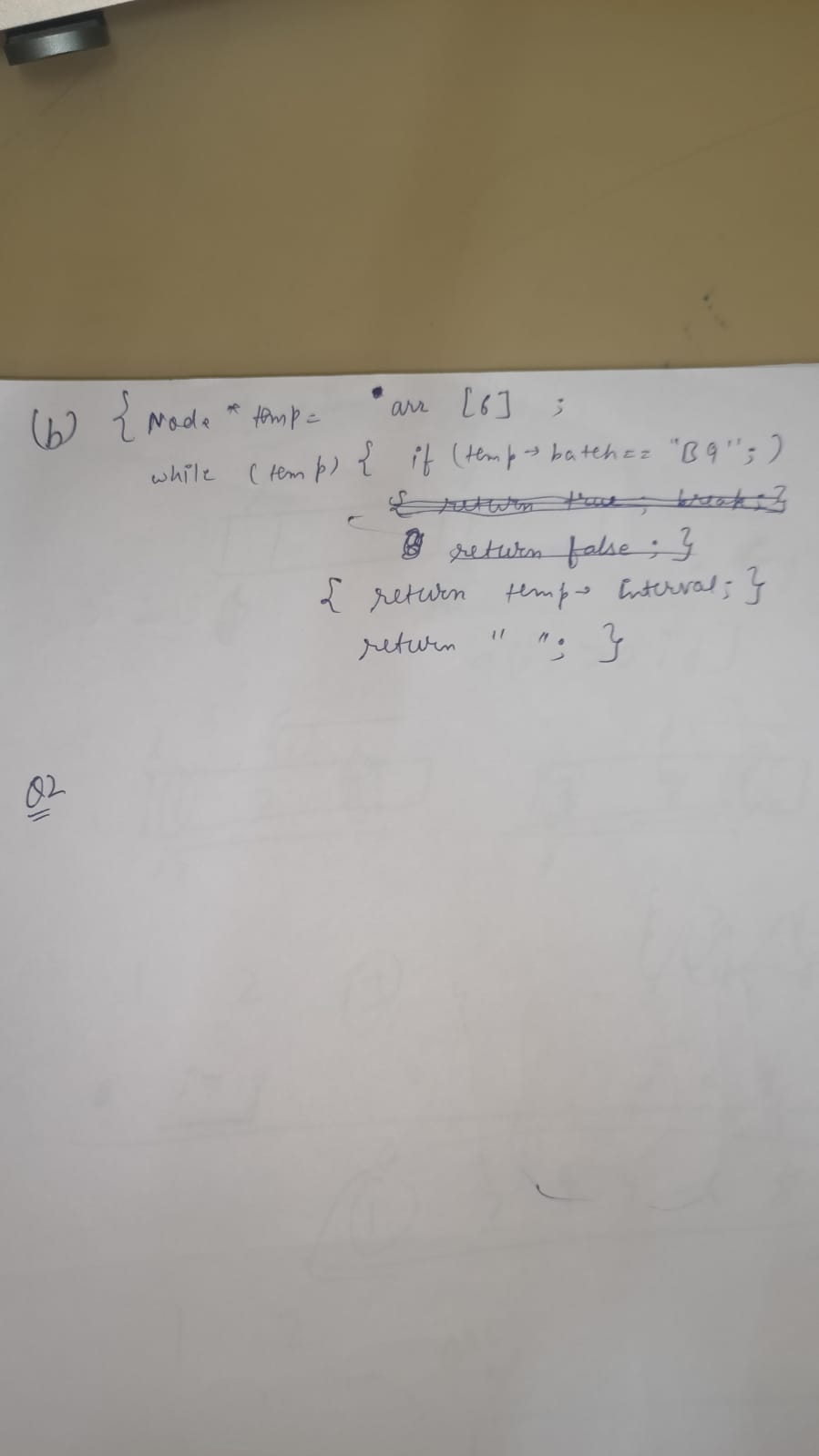
***Data Structure [15B11CI311]***

***Tutorial Sheet***

***Week 7***

***Solution 1:***

**

**

***Solution 2:***

*1) Take "cur" pointer, which will point to head of the first level of the list*

*2) Take "tail" pointer, which will point to end of the first level of the list*

*3) Repeat the below procedure while "curr" is not NULL.*

*I) if current node has a child then*

*a) append this new child list to the "tail"*

*tail->next = cur->child*

*b) find the last node of new child list and update "tail"*

*tmp = cur->child;*

*while (tmp->next != NULL)*

*tmp = tmp->next;*

*tail = tmp;*

*II) move to the next node. i.e. cur = cur->next*

***Solution 3:***

*1) If the node is null return 0, this is also the base case of our recursive algorithm  
2) If a leaf node is encountered then return 1  
3) Repeat the process with left and right subtree  
4) Return the sum of leaf nodes from both left and right subtree*

***Solution 4:***

*int isSumProperty(Node \*root)*

*{*

*if(root == NULL)*

*{*

*return 1;*

*}*

*if(root->left == NULL && root->right == NULL)*

*{*

*return 1;*

*}*

*int sum=0;*

*if(root->left != NULL)*

*{*

*sum+=root->left->data;*

*}*

*if(root->right != NULL)*

*{*

*sum+=root->right->data;*

*}*

*return (root->data == sum && isSumProperty(root->left) && isSumProperty(root->right));*

*}*

***Solution 5:***

int height(Node \*root)

{

    if (root == NULL)

      return 0;

     else

         return 1 + max(height(root->left), height(root->right));

}

int diameter(Node \*root)

{

     if (root == NULL)

         return 0;

     int d1 = 1 + height(root->left) + height(root->right);

     int d2 = diameter(root->left);

     int d3 = diameter(root->right);

     return max(d1, max(d2, d3));

}

***Solution 6:***

**void** helper(Node\* root, **int** d)

{

**if** (!root)

**return**;

**if** (d == res.size())

        cout<<root->val;

**else**

        res[d] = max(res[d], root->val);

    helper(res, root->left, d + 1);

    helper(res, root->right, d + 1);

}

void largestValues(Node\* root)

{

    helper(root, 0);

**return** res;

}

***Solution 7:***

**void** getVerticalOrder(Node\* root, **int** hd, map<**int**, vector<**int**>> &m)

{

**if** (root == NULL)

**return**;

    m[hd].push\_back(root->key);

    getVerticalOrder(root->left, hd-1, m);

    getVerticalOrder(root->right, hd+1, m);

}

**void** printVerticalOrder(Node\* root)

{

    map < **int**,vector<**int**> > m;

**int** hd = 0;

    getVerticalOrder(root, hd,m);

    map< **int**,vector<**int**> > :: iterator it;

**for** (it=m.begin(); it!=m.end(); it++)

    {

**for** (**int** i=0; i<it->second.size(); ++i)

            cout << it->second[i] << " ";

        cout << endl;

    }

}