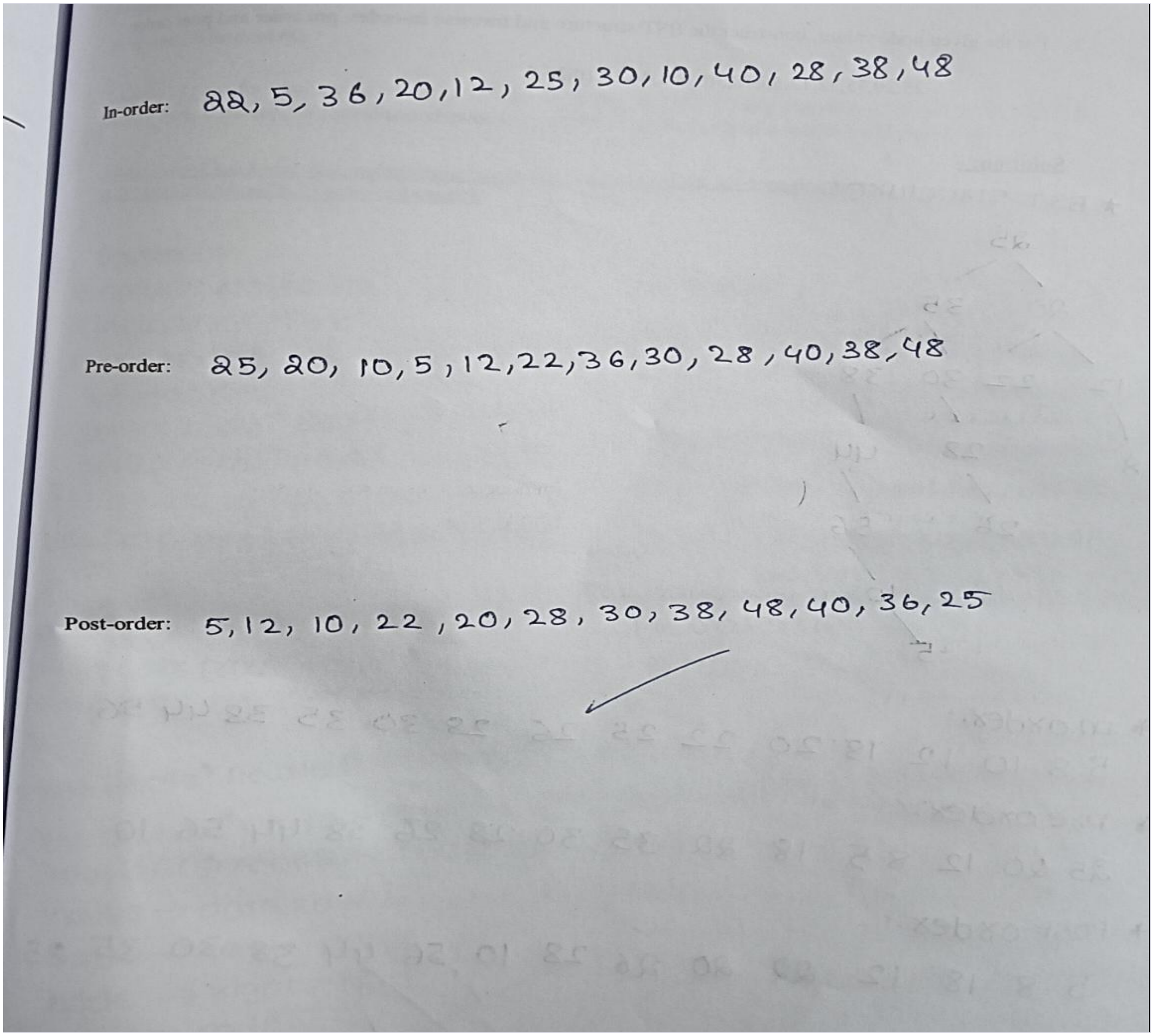
Ds week 10

pg no 162…….

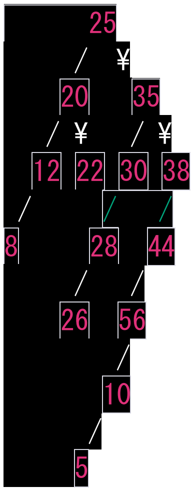


In-order: 22, 5, 36, 20, 12, 25, 30, 10, 40, 28, 38, 48

Pre-order: 25, 20, 10, 5, 12, 22, 36, 30, 28, 40, 38, 48

Post-order: 5, 12, 10, 22, 20, 28, 30, 38, 48, 40, 36, 25

Pg no···163



```
       25
      /  ¥
    20    35
   / ¥    / ¥
 12  22  30  38
 /       /   /
8      28   44
      /    /
    26   56
        /
      10
      /
     5
```

In order······

5 8 10 12 18 20 22 25 26 28 30 35 38 44 56

Pre order……

25 20 12 8 5 18 22 35 30 28 26 38 44 56 10

Post order…..

5 8 18 12 22 20 26 28 10 56 44 38 30 35 25

Pg no 164

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* left;

   struct Node* right;
```

```
};

void preOrder(struct Node* root) {
    if(root == NULL)
        return;
    printf("%d ", root->data);
    preOrder(root->left);
    preOrder(root->right);
}

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

int main() {
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Preorder traversal of binary tree is: ");
    preOrder(root);

    return 0;
```

```
}
```

Pg no 165…

```
#include <stdio.h>
#include <stdlib.h>

// Definition for a binary tree node.
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

// Helper function to create a new node with the given value.
struct TreeNode* newNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = NULL;
    node->right = NULL;
    return node;
}

// Recursive function to insert a value into the binary search tree.
struct TreeNode* insertIntoBST(struct TreeNode* root, int val) {
    // If the tree is empty, create a new node with the given value.
    if (root == NULL) {
        return newNode(val);
    }
    // If the value is smaller than the root's value, insert it into the left subtree.
    if (val < root->val) {
        root->left = insertIntoBST(root->left, val);
    }
    // If the value is larger than the root's value, insert it into the right subtree.
    else {
        root->right = insertIntoBST(root->right, val);
    }
    return root;
}

int main() {
    // Example usage.
    struct TreeNode* root = NULL;
    root = insertIntoBST(root, 4);
    root = insertIntoBST(root, 2);
    root = insertIntoBST(root, 7);
    root = insertIntoBST(root, 1);
    root = insertIntoBST(root, 3);
    return 0;
}
```

Pg no 166…

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
```

```
int max(int a, int b) {

    return (a > b) ? a : b;

}


int getHeight(struct Node* root) {

    if (root == NULL) {

        return -1;

    } else {

        int leftHeight = getHeight(root->left);

        int rightHeight = getHeight(root->right);

        return 1 + max(leftHeight, rightHeight);

    }

}


struct Node* createNode(int data) {

    struct Node* node = (struct Node*)malloc(sizeof(struct Node));

    node->data = data;

    node->left = NULL;

    node->right = NULL;

    return node;

}


int main() {

    struct Node* root = createNode(10);

    root->left = createNode(5);

    root->right = createNode(20);

    root->right->left = createNode(15);

    root->right->right = createNode(25);
    printf("Height of the binary search tree is %d\n", getHeight(root));


    return 0;
}
```

Pg no 167….

Here's a complete binary tree with exactly six nodes and different values in each node...

```
    5
  /  \
 2    3
 / \
1    4
```

To represent this tree in an array, we can use the following indices:..
0 1 2 3 4 5

| 5 | 2 | 3 | 1 | 4 | null |

Note that because this is a complete binary tree, any missing nodes are represented with `null` in the array.

Pg no 168…..

```
struct Node* search(Node* node, int key) {
  if (node == NULL || node->data == key) {
    return node;
  }
  if (key < node->data) {
    return search(node->left, key);
  }
  else {
    return search(node->right, key);
  }
}
```

Pg no 169…

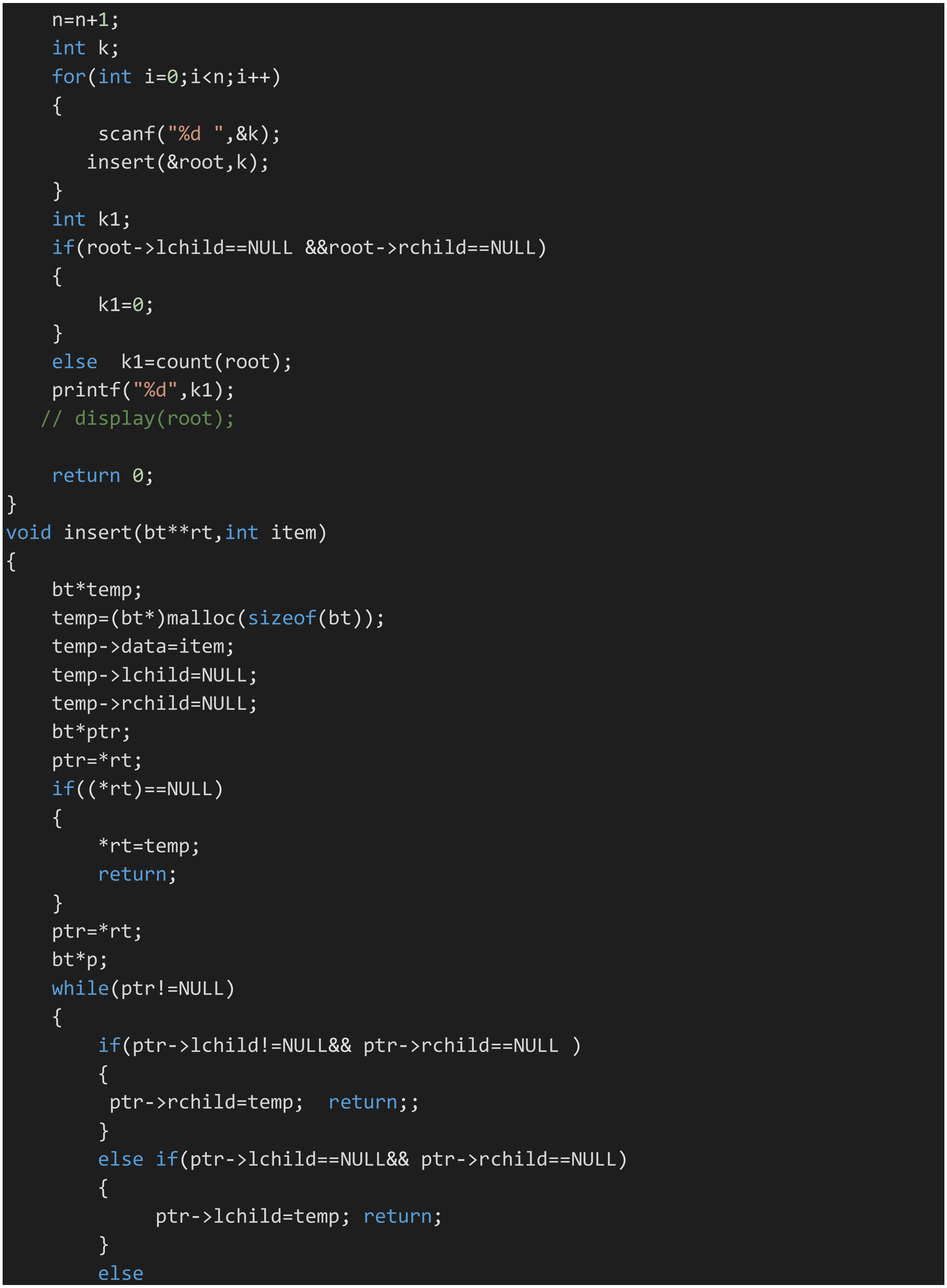
```
#include<stdio.h>
#include<ctype.h>
char stack[100];
```

```
int top=-1;
void push(char ch)
{
    stack[++top]=ch;
}
char pop()
{
    return (stack[top--]);
}
int priority(char ch)
{
    if(ch=='(')
    return -1;
    if(ch=='+'||ch=='-')
    return 1;
    if(ch=='*'||ch=='/')
    return 2;
}

int main()
{
    char exp[400],ch;
    int t;
    scanf("%d",&t);
    while(t--)
    {
    //printf("\nneter expression:");
    scanf("%s",exp);
    char *e=exp;
    while(*e!='\0')
    {
              if(isalpha(*e))
                        printf("%c",*e);
              else if(*e=='(')
                        push(*e);
              else if(*e==')')
              {
                        ch=pop();
                        while(ch!='(')
                        {
                                  printf("%c",ch);
```

```
                ch=pop();
            }
        }
        else
        {

            if(priority(stack[top])>=priority(*e))
            {
                printf("%c",pop());
            }


            push(*e);
        }

        e++;
    }
    while(top!=-1)
    printf("%c",pop());
    printf("\n");
}
    return 0;
}
```

Pg no 170….

```
#include <stdio.h>
#include<malloc.h>
struct node
{
    int data;
    struct node*lchild;
    struct node*rchild;
};
typedef struct node bt;
void insert(bt**,int);
int count(bt*);
void display(bt*);
int main()
{
    bt*root=NULL;
    int n;
    //enter no of nodes//
    scanf("%d\n",&n);
```



```
    n=n+1;
    int k;
    for(int i=0;i<n;i++)
    {
        scanf("%d ",&k);
       insert(&root,k);
    }
    int k1;
    if(root->lchild==NULL &&root->rchild==NULL)
    {
        k1=0;
    }
    else  k1=count(root);
    printf("%d",k1);
   // display(root);

    return 0;
}
void insert(bt**rt,int item)
{
    bt*temp;
    temp=(bt*)malloc(sizeof(bt));
    temp->data=item;
    temp->lchild=NULL;
    temp->rchild=NULL;
    bt*ptr;
    ptr=*rt;
    if((*rt)==NULL)
    {
        *rt=temp;
        return;
    }
    ptr=*rt;
    bt*p;
    while(ptr!=NULL)
    {
        if(ptr->lchild!=NULL&& ptr->rchild==NULL )
        {
         ptr->rchild=temp;  return;;
        }
        else if(ptr->lchild==NULL&& ptr->rchild==NULL)
        {
             ptr->lchild=temp; return;
        }
        else
        {
            ptr=ptr->lchild;
        }

    }

}
int count(bt*rt)
{
    if(rt==NULL) return 0;
    if(rt->lchild==NULL && rt->rchild==NULL)
    {
        return 1;
    }
    if(rt->lchild!=NULL && rt->rchild!=NULL)
       return count(rt->lchild)+count(rt->rchild);
    /*else if(rt->lchild!=NULL && rt->rchild==NULL)
       return count(rt->lchild);
    else
       return count(rt->rchild);*/

}
```

Pg no 171….

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

struct node {

    int data;
    struct node *left;
    struct node *right;

};
```

```
struct node* insert( struct node* root, int data ) {

    if(root == NULL) {

        struct node* node = (struct node*)malloc(sizeof(struct node));

        node->data = data;

        node->left = NULL;
        node->right = NULL;
        return node;

    } else {

        struct node* cur;

        if(data <= root->data) {
            cur = insert(root->left, data);
            root->left = cur;
        } else {
            cur = insert(root->right, data);
            root->right = cur;
        }

        return root;
    }
}
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

void inOrder(struct Node *root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}

int main() {
```

```c
    struct node* root = NULL;

    int t;
    int data;

    scanf("%d", &t);

    while(t-- > 0) {
        scanf("%d", &data);
        root = insert(root, data);
    }

    inOrder(root);
    return 0;
}
```

Pg no 172….

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define pcx putchar_unlocked
#define gcx getchar_unlocked
#define MAXNODES    200000
#define HBKSIZE     (1<<18)
typedef long int lint;

    lint getl () {
        lint n =0;
        register int c = gcx();
        while(c<'0' || c>'9') c = gcx();
        while(c>='0' && c<='9') {
            n = n * 10 + c-'0';
            c = gcx();
        }
        return n;
    }

    void putsx (char *s, lint l) {
        for (lint ci=0; ci<l; ++ci) pcx(s[ci]);
    }

```

```c
typedef struct htnd {
    lint val;
    struct htnd* nxt;
} HT_t;

lint gNPIdx =0;
HT_t *gNodePool;
HT_t *hBkt[HBKSIZE];

lint insHTval (lint X) {
    lint bid = X & (HBKSIZE -1);
    HT_t *node = hBkt[bid];
    for (; node; node = node->nxt )
        if (node->val == X) return -1; // no insert
    node = gNodePool + gNPIdx ++;
    node->val = X;
    node->nxt = hBkt[bid];  // NULL or Hash-Bucket-Head
    hBkt[bid] = node;
    return X;
}

int main () {
    lint T = getl() +1;
    gNodePool = (HT_t*) malloc (MAXNODES * sizeof(HT_t));
    while(--T) {
        lint N = getl();
        lint M = getl();
        for(lint ni=0; ni<N; ++ni) insHTval (getl());
        for(lint mi=0; mi<M; ++mi)
            if (insHTval (getl()) < 0) putsx("YES\n", 4);
            else putsx("NO\n", 3);
        gNPIdx = 0;
        memset(hBkt, 0, sizeof(hBkt));
    }
    return 0;
}
```

Pg no 173…

```
#include <stdio.h>
#include <stdlib.h>

struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

void postorderTraversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->val);
}

int main() {
    // create a binary tree
    struct TreeNode *root = (struct TreeNode*) malloc(sizeof(struct TreeNode));
    root->val = 1;
    root->left = (struct TreeNode*) malloc(sizeof(struct TreeNode));
    root->left->val = 2;
    root->left->left = NULL;
    root->left->right = NULL;
    root->right = (struct TreeNode*) malloc(sizeof(struct TreeNode));
    root->right->val = 3;
    root->right->left = NULL;
    root->right->right = NULL;

    // perform postorder traversal
    printf("Postorder Traversal: ");
    postorderTraversal(root);

    return 0;
}
```

Pg no 174...

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) {
        struct TreeNode* new_node = (struct TreeNode*) malloc(sizeof(struct TreeNode));
        new_node->val = val;
        new_node->left = NULL;
        new_node->right = NULL;
        return new_node;
    }
    if (val <= root->val) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

void preOrderTraversal(struct TreeNode* root, int Q) {
    if (root == NULL) {
        return;
    }
    if (root->val == Q) {
        printf("%d ", root->val);
        preOrderTraversal(root->left, Q);
        preOrderTraversal(root->right, Q);
    } else if (root->val < Q) {
        preOrderTraversal(root->right, Q);
    } else {
        preOrderTraversal(root->left, Q);
        if (root->val == Q) {
            printf("%d ", root->val);
        }
        preOrderTraversal(root->right, Q);
    }
}
```

```
int main() {
    int N, Q;
    scanf("%d %d", &N, &Q);
    struct TreeNode* root = NULL;
    int val;
    for (int i = 0; i < N; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    printf("Preorder Traversal of Subtree with Root Node Data Equal to %d: ", Q);
    preOrderTraversal(root, Q);
    return 0;
}
```