EAST WEST UNIVERSITY

Assignment 02

Department of CSE
Course: CSE207 Data Structures
Course code: CSE207

Submitted by:

Name: Izazul Hoq Imran
ID : 2019-3-60-043

Submitted to:

Dr. Maheen Islam
Associate Professor
Department of Computer
Science & Engineering

Ans: to the Q no: 1

A) Ans: If the list is sorted at the point using linked list the required element can be searched simply by moving forward using next pointer.

Since binary search performs random access it can be done only on an array. Linked list can not be used for Binary search operation.

B) Ans: An the case list is not sorted its mean unsorted then using arrays the specified element can be searched. The arrays encourages the access to random element.

searching element are scan in a sequntial fashion. This is done using the linear search algorithm and it can be done on an Array or a linked list.

Ans: to the Q no: 2

Solution;

Pseudocode for the function.

```
squareRoot (num, ans, tol)
    if (|ans^2 - num|) ≤ tol
     return ans;
    else
     return squareRoot (num, (ans^2 + num) /
                (2*ans), tol)
```

Now we test it to manually.

* Case 1: squareRoot (5, 2, 0.01)

$|ans^2 - num|$

$= 4 - 5 = 1$

$1 > 0.01$ thus squareRoot (5, 2.25, 0.01)

$|ans^2 - num| = |5.0625 - 5| = 0.0625$

$0.0625 > 0.01$ thus squareRoot (5, 2.2361, 0.01)

$|ans^2 - num| = |5.0002 - 5| = 0.0002$

which is less than 0.01

Here ans is 2.23611.

test another squareRoot (4, 2, 0.01)

⇒ squareRoot (4, 2, 0.01)

$|ans^2 - num|$

$\Rightarrow 4 - 4 = 0$

$0 = 0$

thus here ans is 2

Ans: to the Q no: 3

⇒ //code for find length

```
int findExprLen (char * expr)
{
    int len1, len2;

    if(strcspn (expr, OPERATORS)== 0)
    {
        len 1 = findExprLen (exp+1)

        len 2= findExprLen(expr+1+len)
    }
    else

        len1 = len 2=0;

        return len1+ len2+1;

}
```

⇒ Here is the find length code.

⇒ Now discribe How recurssion function work;

Solution,

Here expression given that *+AB/CD.

Here 1st character is an operator (*), we call recursive after stripping off the into operator leaving "+AB/CD". And again 1st character is (+) so we call recursive again. This time with "AB/CD". NOW 1st character is operand (A) and we return 1;. we call again recursive this time with the expression is "B/CD". which return length of 1.

we now return from the call having expression +AB/CD. The return value is 2.

The second recursive call "*+AB/CD", expression +AB, which is "/CD" 1st case is operator so we call again recursion. This complete isolation of the first expression *+AB by returning length 4.

Ans: to the Q no: 9

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

Void recursion ( int k ,char string[], int n)
    {
      if(n==k)
        {string [n] = NULL;
         Printf ("%s \n", string);

      if( string [n-1] == 1)
        {
          string [n] = '0';
          recursion( k, string , n+1)

        if ( string [n-1] = '0')
          {string [n] = '0';
           recursion ( k, string, n+1):
           string [n] = '1';
           recursion ( k, string, n+1);
          }
        }
void  print ( int K)
    {
      if ( K<=0)
      Char string [K];
        String [0] = '0';
        recursion (K,s
```

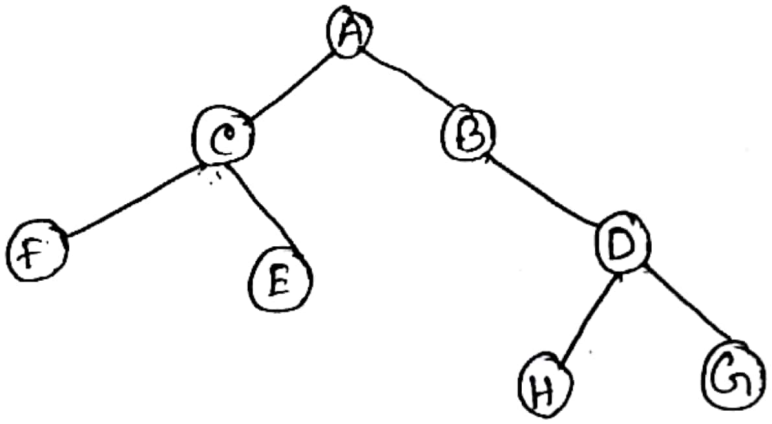
```
    s[0] = 1;
    recursion(k, string, 1)
}

int main()
{
    int n;
    printf("Enter the value for binary");
    scanf("%d", &n);
    print(n);
}
```

Ans: to the Q no: 5

Here given post order is : FECHGDBA

Inorder is : FCEABHDG

Tree is;



Here Preorder is : ACFEBDHG

Ans: to the Q no: 6

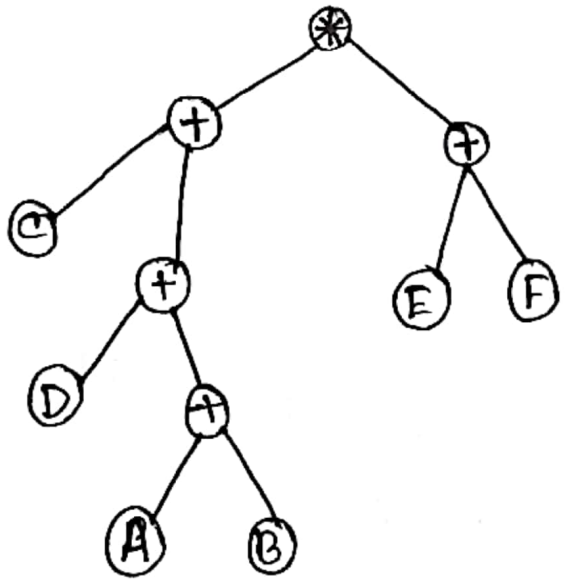
Here given expression is (C+D+A+B) x (E+F)

solution;

Postfix expression is: CDAB+++*EF+

Prefix expression is: +++CDAB*+EF

Tree is:



Draw the expression tree.

Ans: to the Q no:7

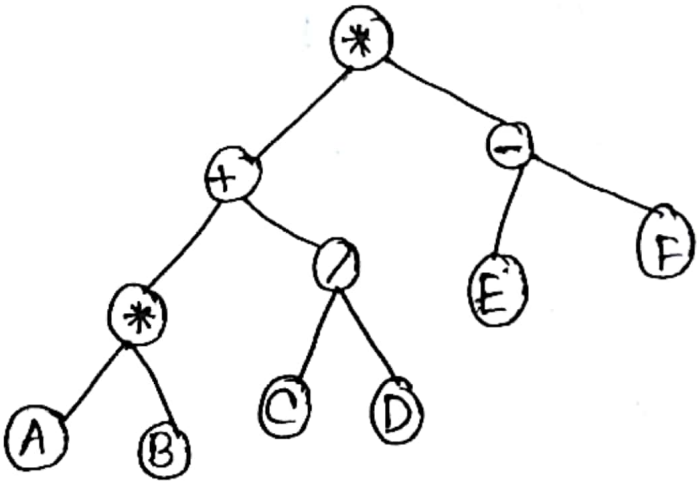
Here given postfix expression is : A B x C D / + E F - x

Solution;

infix expression is : (A * B + C / D) x (E - F)

prefix expression is: * + * AB / CD - EF

⇒ The tree is;



Expression the tree:

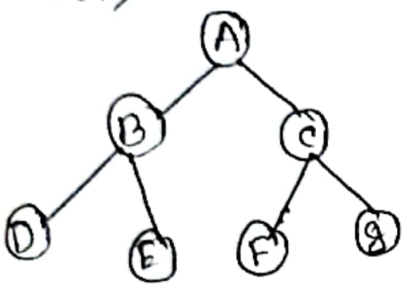
Ans: to the Q no: 8

solution

In the quesation mark missing factor is : 2^{L-1}

for this solution

Let,



Level 0 = 1

Level 1 = 2

Level 2 = 4

Level (1) For,

$n(1) = n(0) + x$

$x = 1$

$2^{L-1} = 1$

For, Level 2

$n(2) = n(1) + x$

$= 4 = 2 + x$

$x = 2$

For, level 3,

$n(3) = n(2) + x$

$8 = 4 + x$

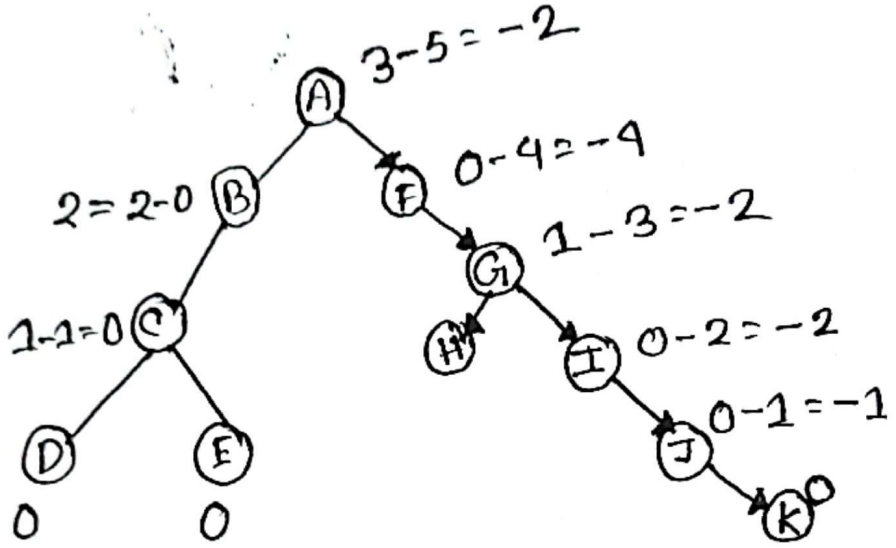
$x = 4$

(Should)

Ans: to the Q no: 9 (A)

(A) Ans:

Given tree



We know and given that $B = H_L - H_R$

(Ans:)

Ans: to the Q;9 (B)

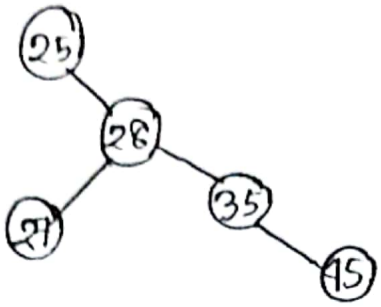
solution,
write a function for balance factor is

```
// Balanced function
    int Balanced (struct node *node);
    {
      int lh; // lh is for sub tree height
      int rh; // rh is for sub tree height
      if(root == NULL)
        return 1;
        lh = height (root -> left)
        rh = right (root -> right)
if (abs (lh-rh) <= 1 && Balanced (root -> left) && Balanced
  (root -> right))
        return 1;
// compute height
  int height ( struct node * node)
    {
      if (node == NULL)
        return 0;
      else
        return 1 + max (height (node -> left), height
        (node -> right));
    }
```

Ans: to the Q no: 10

Solution,

given tree is



are in here see all data entered a sequance Here i write a. sequance 25, 28, 27, 35, 45. where data firstly chacked with root if it is bigger than root go to right and it's smaller than root go to left.

Here is possible more than one sequance

: 25, 28, 27, 35, 45.

⇒ or: 25, 28, 35, 27, 45.

⇒ or; 25, 28, 35, 45, 27.

Ans: to the Q no: 11

(B)

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

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

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

int height_BST (struct node *node)
   {
      if (node == NULL)
         return 0;
      else
         {
            int LS;
```

```
int RS;
LS= height_BST (node -> left);
RS= height_BST (node -> Right);
if (LS > RS)
  {
   return LS+1;
  }
else
   return RS+1;

void insert (struct node *root, int n1, int n2, char lr)
  {
   if (root == NULL)
      return;
   if (root -> data == n1)
     {
      switch (lr)
        {
         case 'l':
            root -> left = newnode (n2);
            break;
         case 'r':
            root -> right = newnode (n2);
            break;
        }
     }
```

```
else
    { insert (root->left, n1, n2, lr);
      insert ( root->right, n1, n2, lr);
    }
}
void inorder (struct node *root)
    { if (root == NULL)
        return;
      else
      { inorder (root->left);
        printf ("%d ", root->data);
        inorder (root->right);
      }
    }

int main ( )
    {
      struct node *root = NULL;
      int n;
      scanf ("%d", &n);
      while (n)
       {
        char lr;
        int n1, n2;

        scanf ("%d", &n1);
        scanf ("%d", &n2);
```

```
scanf("%c", &lr);
if(root == NULL)
  {
    root = newnode(n1)
      switch(lr)
        {
          case 'l':
              root->left = newnode(n2);
              break;
           case 'r':
              root 'r';
                root->right = newnode(n2);
                 break;
              }
    }
  else
     { insert (root, n1, n2, lr);
     }
inorde (root);
printf("Height is: %d", height_BST(root));
}
```

Ans: to the Q no: 12

```
#include <stdio.h>
#include<stdlib.h>
  Struct tree
        {
          String data;
          tree * left;
          tree * right;
        }
tree * createTree (String item)
   {
     tree *newnode = new tree;
          newNode -> data = item;
          newnode -> left= NULL;
          newnode -> right = NULL;
       return newnode;
     }
void  Add_left (tree * parent, tree, *chidn  )
     {
       parent -> left = childNode;
     {
void Add_Right (tree * parent, tree, * child  )
      {
        parent -> Right = childNode;
      {
 tree * createTree ()
      {
          tree *lab = createTree ("Labradon");
          tree *Gen = createTree ("German Shphend");
```

```
tree  *Rot = createTree ("Rottweiler");
    Addleft (lab, Ger);
    Add right (lab, Rot);
tree * coc  = createtree ("cocker spaniel");
tree * Gol  = createtree ("Golden Retriever");
Addleft (Ger, coc);
Add_right (Ger Gol);
tree  *poo = createtree ("poodle");
tree  * she = createtree ("shetland sheepdog");
Add left (Rot, poo);
Add_Right (Rot, she);
tree  *bea = createtree ("Beagle");
tree * dac = createtree ("Dachshund");
Add left (coc, bea);
Add_Right (coc, dac);
tree *dal = createtree ("Dalmtian");
    Addleft (dac, dal);
    return lab;
}

Void print (tree * node, int level)
    {
      for (i= 0; i< level-1; i++)
        {
```

```
        Printf("\t ");
        printf( "%5 %5 ", level, node->data);
        if (node->lef != NULL)
            Print(node->left, level+1)
            if (node->right != NULL)
                Print(node->right, level+1)
    }

int main()
    {
        tree *root = createTree();
        Print(root, 1);
    }
```

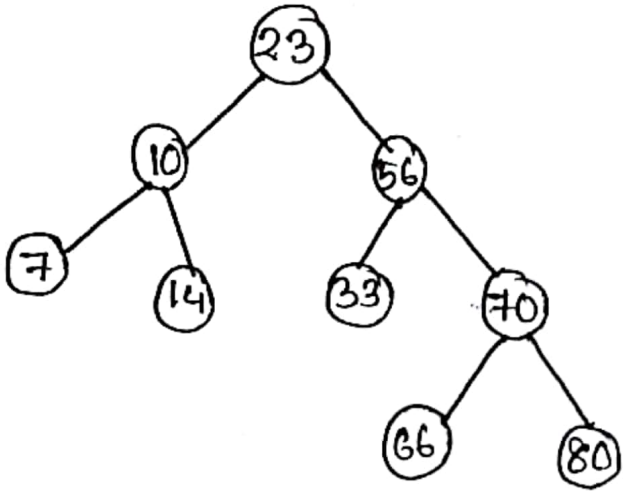
—— o ——

Ans: to the Q no: 13 (a)

a) Given that Balancing factor is;

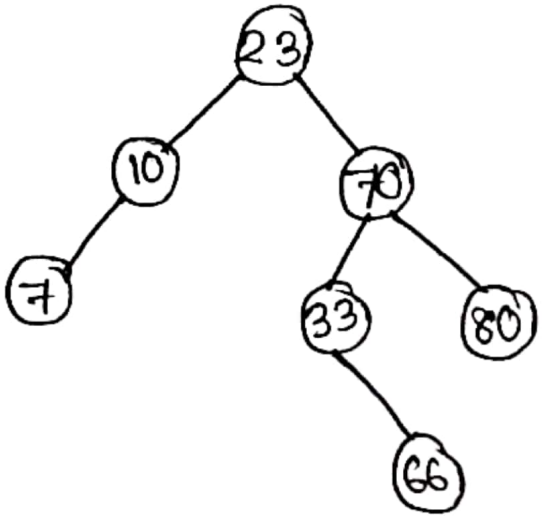
7 10 14 23 33 56 66 70 80

Now create a AVL Tree;



13

After delete node 56 and M tree are



Ans