Soru : 1'den 2^k-1'e kadar olan sayıları hangi sırada ağaca eklersek tam dengeli bir ikili arama ağacı elde edebilriz?

Dengeli arama ağacı elde edebilmek için önce kök sayı ağaca eklenmeli,daha sonra sol ve sağ çocuklar ağaca eklenmelidir.Bu sebeple fonksiyon her alt ağaç için o alt ağacın kök düğümündeki sayıyı tespit etmektedir(2).Dengeli bir arama ağacında sol alt ağactaki eleman sayısı ile sağ alt ağactaki eleman sayısı aynıdır.o halde kökteki eleman en küçük ve en büyük sayının ortalamasıdır.Fonksiyon daha sonra sırasıyla sol ve sağ alt ağaçlar için kendini çağırmaktadır(5-6).Bu şekilde önce sol alt ağacına ve onunda sol alt ağacına, vs. Eklenecek sayılar belirlenecek,daha sonra fonksiyon geri dönüp sağ alt ağaca eklenecek sayılar belirlenecektir.

```
1. Void dengeli (int sol,int sag){
2. Int orta=(sol+sag)/2;
3. System.out.print(orta);
4. If(sol!=orta){
5. Dengeli(sol,orta-1);
6. Dengeli(orta+1,sağ);
7. }
8. }
```

Soru : Verilen bir ikili arama ağacındaki yaprak sayısını hesaplayan fonksiyonu yazınız.

Agac sınıfının içindeki yaprak_say fonksiyonu bir ağaçtaki yaprak sayısını döndürür.Bunu yapma için Dugum sınıfı içindeki yaprak_say2 fonksiyonu ağacın kök düğümüyle çağırılır(13).Eğer ağaçta hiç düğüm yoksa fonksiyon 0 döndürür(15).

Dugum sınıfı içindeki yaprak_say2 ise kök düğümü verilen düğüm olan alt ağaçtaki yaprak sayısını bulur. Bir alt ağaçtaki yaprak sayısı o alt ağacın sol(5-6) ve sağ (7-8) alt ağaçlarındaki yaprak sayılarının toplamına eşittir.(9).Bir alt ağaç sadece yapraktan oluşuyorsa fonksiyon 1 döndürür(3-4).

```
Int yaprakSay2(){
Int toplam=0;
If(sol==null && sag==null)
Return 1;
If(sol!=null)
Toplam+=sol.yaprakSay2();
If(sag!=null)
Toplam+=sag.yaprakSay2(9;
Return toplam;
}
Int yaprakSay(){
If(kok!=null)
Return kok.yaprakSay2(9;
Else
Return 0;
}
```

Soru : Verilen bir ikili arama ağacında sayıları seviye seviye ekrana yazan fonksiyonu yazınız.Fonksiyon önce kök düğümü,sonra kök düğümün çocuklarını ,daha sonra kök düğümün çocuklarının çocukalrını vs ekrana yazacaktır.

Ağacın kök düğümünden başlamak üzere(6-10) her düğüm kuyruga eklenir.Her aşamada kuyrugun başından bir düğüm silinir(12-13) ve o düğümün varsa sol (15-18) ve sağ çovukları kuyrugun sonuna eklenir.(16-19).Döngü kuyrukta düğüm

kalmayana kadar devam eder(11).Her düğüm için,düğümün çocukları düğümden sonra kuyruga eklendiği için yapılan işlem seviye seviyedir.Önce 1.seviyedeki düğümler sonra 2.seviyedeki düğümler vs . işlenir.

```
Void seviyeYaz(){
 Kuyruk k;
 Dugum d;
 Eleman e;
 K=new Kuyruk(9;
 D=kok;
 If(d!=null){
  E=new Eleman(d);
  k.kuyrugaEkle(e);
 }
 While(!k.kuyrukBos()){
  E=k.kuyrukSil();
  D=e.dugum;
  System.out.print(d.icerik);
  If(d.sol!=null){
   E=new Eleman(d.sol);
   k.kuyrugaEkle(e);
  }
  If(d.sag!=null){
  E=new Eleman(d.sag);
  k.kuyrugaEkle(e);
  }
 }
}
```

Soru : AVL ağacında durum 2'yi çözmek için uygulanan sol çift rotasyon algoritmasını tek rotasyon fonksiyonlarını kullanmadan yazınız.

AVL ağacında durum 2'yi çözmek için uygulanan sol çift rotasyon algoritması aşağıda verilmiştir.4 tane bağlantı değiştirikmektedir.

**k1'in sağ çocugu artık k2 değiş B diğer bir deyişle k2'nin sol çocugudur.
**k3'ün sol çocugu artık k1 değil C diğer vir deyişle k2'nin sağ çocugudr.
**k2'nin sol çocugu artık B değil k1'dir.
**k2'nin sağ çocugu artık C değil k3'tür.

```
AvlDugum solCiftRotasyon(AvlDugum k3){
 AvlDugum k1,k2;
 K1=k3.sol;
 K2=k1.sag;
 K1.sag=k2.sol;
 K3.sol=k2.sag;
 K2.sol=k1;
 K2.sag=k3;
 K1.boy=azami(boy(k1.sol),boy(k1.sag))+1;
 K3.boy=azami(boy(k3.sol),boy(k3.sag))+1;
 K2.boy=azami(k1.boy,k3.boy)+1;
 Return k2;
}
```

**1. 9,11,7,13,10,5,6,8 sayılarının boş bir ikili arama ağacına eklenmesiyle ortaya çıkan ağacı gösteriniz

Soru : Birinci çözümlü alıştırmada buldugumuz ikili arama ağacından 9'u siliniz.

İkili arama ağacının özelliğinin bu yerleştırme işlemi sonucunda bozulmamasını sağlayabilcek bir eleman secmemız gerekir.Bu ise, ya silinecek elemanın solundaki alt ağacın en büyük elemanının ya da silinecek elemanın sağındaki alt ağacın en küçük elemanının silinen elemanın yerine yerleştırılmesı ile ollur.Peki bu eleman yerine yerleştırıldıkten sonra o elemandan boşalan yere kim yerleştirilecel? Aynı kural orada da gecerlidir.

Soru : Birinci çözümlü alıştırmada buldugumuz ikili arama ağacında önce,ara ve sonra gezinti algoritmalarıyla geziniz.

Gezintiler : PreOrder: Ebeveyn,sol çocuk,sağ çocuk(8,7,5,6,11,13,10,13)
İnorder: Sol çocuk,Ebeveyn,Sağ çocuk(5,6,7,8,10,11,13)
Postorder: sol çocuk,sağ çocuk,Ebeveyn(6,5,7,10,13,11,8)

Ara gezinti algoritması :

```
Void araGezinti(){
If(sol!=null)
Sol.araGezinti();
System.out.print(icerik);
If(sag!=null)
Sag.araGezinti();
}
```

Sonra gezinti algoritması :

```
Void sonraGezinti(){
If(sol!=null)
Sol.sonraGezinti();
If(sag!=null=
Sag.sonraGezinti();
System.out.print(icerik);
}
```

Önce gezinti algoritması :

```
Void onceGezinti(){
System.out.print(icerik);
If(sol!=null)
Sol.onceGezinti();
If(sag!=null)
Sag.onceGezinti();
}
```

mada buldugumuz ikili arama ağaçlarından hangisi AVL
VL ağacı haline getirmek için hangi rotasyonu
z.

r ikili arama ağacıdır.AVL ağacında her düğümün sol ve
en fazla 1 olabilir.AVL özelliği bozuluş düğüm d olsun.

t ağacına br eeman eklendiğinde
**d'nin sol çocugunun sağ alt ağacına bir eleman eklendiğinde
**d'nin sağ çocugunun sol alt ağacına bir eleman eklendıgında
**d'nin sağ çocugnun sağ alt ağacına bir eleman eklendiğinde
1. Ve 4. duurmlar ile 2. ve 3. duurmlar birbirlerinin siöetriği olup temelde iki değişk duurmu oluştururlar.1.ve 4. durumları tek rotasyonn , 2. ve 3. durumları çift rotasyon uygulayarak çözebilirz.

Tek Rotasyon

Soru : Verilen bir ikili arama ağacındaki sayıları büyükten küçüğe ekrana yazan fonksiyonu yazınız.

Soru : Verilen bir ikili arama ağacındaki dolu düğüm sayısını hesaplayan fonksiyonu yazınız.Bir düğümün hem sol hem de sağ çocugu varsa o düğüme dolu düğüm denir.

```
// Java implementation of the
```

```
// above approach
import java.util.*;
class GFG{

// Node of the binary tree
static class node
{
int data;
node left;
node right;

node(int data)
{
      this.data = data;
      left = null;
      right = null;
}
};

// Function to print flattened
// binary tree
static void print(node parent)
{
node curr = parent;
while (curr != null)
{
      System.out.print(curr.data + " ");
      curr = curr.right;
}
}

static node prev;

// Function to perform reverse
// in-order traversal
static void revInorder(node curr)
{
// Base case
if (curr == null)
      return;
revInorder(curr.right);
prev.left = null;
prev.right = curr;
prev = curr;
revInorder(curr.left);
}

// Function to flatten binary
// tree using level order
// traversal
static node flatten(node parent)
{
// Dummy node
node dummy = new node(-1);

// Pointer to previous
// element
prev = dummy;
```

```
// Calling in-order
// traversal
revInorder(parent);

prev.left = null;
prev.right = null;
node ret = dummy.right;

// Delete dummy node
//delete dummy;
return ret;
}

// Driver code
public static void main(String[] args)
{
node root = new node(5);
root.left = new node(3);
root.right = new node(7);
root.left.left = new node(2);
root.left.right = new node(4);
root.right.left = new node(6);
root.right.right = new node(8);

// Calling required function
print(flatten(root));
}
}

// This code is contributed by Amit Katiyar
```

Soru : Verilen bir ikili arama ağacındaki tek düğüm sayısını hesaplayan foksiyonu yazınız.Bir düğümün değeri tek sayıysa o duğüme tek düğüm denir.

```
// Java program to print all odd node of BST
class GfG {

// create Tree
static class Node {
      int key;
      Node left, right;
}

// A utility function to create a new BST node
static Node newNode(int item)
{
      Node temp = new Node();
      temp.key = item;
      temp.left = null;
      temp.right = null;
      return temp;
}

// A utility function to do inorder traversal of BST
static void inorder(Node root)
{
      if (root != null) {
            inorder(root.left);
            System.out.print(root.key + " ");
```

```
            inorder(root.right);
        }
    }

    /* A utility function to insert a new node
    with given key in BST */
    static Node insert(Node node, int key)
    {
        /* If the tree is empty, return a new node */
        if (node == null)
            return newNode(key);

        /* Otherwise, recur down the tree */
        if (key < node.key)
            node.left = insert(node.left, key);
        else
            node.right = insert(node.right, key);

        /* return the (unchanged) node pointer */
        return node;
    }

    // Function to print all odd nodes
    static void oddNode(Node root)
    {
        if (root != null) {
            oddNode(root.left);

            // if node is odd then print it
            if (root.key % 2 != 0)
                System.out.print(root.key + " ");

            oddNode(root.right);
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        /* Let us create following BST
              5
        / \
        3 7
        / \ / \
        2 4 6 8 */
        Node root = null;
        root = insert(root, 5);
        root = insert(root, 3);
        root = insert(root, 2);
        root = insert(root, 4);
        root = insert(root, 7);
        root = insert(root, 6);
        root = insert(root, 8);

        oddNode(root);

    }
    }
```

Soru : Verilen bir ikili arama ağacındaki yapraklardan rastgele bir tanesini döndüren fonksiyonu yazınız.

```
  // Java program to Select a Random Node from a tree
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
static class Node
{
    int data;
    int children;
    Node left, right;
}

static Node newNode(int data)
{
    Node temp = new Node();
    temp.data = data;
    temp.left = temp.right = null;
    temp.children = 0;
    return temp;
}

// This is used to fill children counts.
static int getElements(Node root)
{
    if (root == null)
        return 0;
    return getElements(root.left) +
        getElements(root.right) + 1;
}

// Inserts Children count for each node
static Node insertChildrenCount(Node root)
{
    if (root == null)
        return null;

    root.children = getElements(root) - 1;
    root.left = insertChildrenCount(root.left);
    root.right = insertChildrenCount(root.right);
    return root;
}

// returns number of children for root
static int children(Node root)
{
    if (root == null)
        return 0;
    return root.children + 1;
}

// Helper Function to return a random node
static int randomNodeUtil(Node root, int count)
{
```

```
        if (root == null)
            return 0;

        if (count == children(root.left))
            return root.data;

        if (count < children(root.left))
            return randomNodeUtil(root.left, count);

        return randomNodeUtil(root.right,
            count - children(root.left) - 1);
}

// Returns Random node
static int randomNode(Node root)
{

        int count = (int) Math.random() *
                            (root.children + 1);
        return randomNodeUtil(root, count);
}

// Driver Code
public static void main(String args[])
{

        // Creating Above Tree
        Node root = newNode(10);
        root.left = newNode(20);
        root.right = newNode(30);
        root.left.right = newNode(40);
        root.left.right = newNode(50);
        root.right.left = newNode(60);
        root.right.right = newNode(70);

        insertChildrenCount(root);

        System.out.println( "A Random Node From Tree : " +
                                                randomNode(root));
}
}

// This code is contributed by Arnab Kundu
```

Soru :Verilen bir ikili arama ağacındaki düğümlerden değeri X'ten küçük olanların sayısını döndüren fonksiyonu yazınız.

Soru :Verilen bir ikili arama ağacındaki tüm yaprakları silen fonksiyonu yazınız.

```
// Java program to delete leaf Node from
// binary search tree.
class GfG {

        static class Node {
            int data;
            Node left;
```

```
    Node right;
}

// Create a newNode in binary search tree.
static Node newNode(int data)
{
    Node temp = new Node();
    temp.data = data;
    temp.left = null;
    temp.right = null;
    return temp;
}

// Insert a Node in binary search tree.
static Node insert(Node root, int data)
{
    if (root == null)
        return newNode(data);
    if (data < root.data)
        root.left = insert(root.left, data);
    else if (data > root.data)
        root.right = insert(root.right, data);
    return root;
}

// Function for inorder traversal in a BST.
static void inorder(Node root)
{
    if (root != null) {
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
    }
}

// Delete leaf nodes from binary search tree.
static Node leafDelete(Node root)
{
    if (root == null) {
        return null;
    }
    if (root.left == null && root.right == null) {
        return null;
    }

    // Else recursively delete in left and right
    // subtrees.
    root.left = leafDelete(root.left);
    root.right = leafDelete(root.right);

    return root;
}

// Driver code
public static void main(String[] args)
{
    Node root = null;
    root = insert(root, 20);
    insert(root, 10);
```

```
            insert(root, 5);
            insert(root, 15);
            insert(root, 30);
            insert(root, 25);
            insert(root, 35);
            System.out.println("Inorder before Deleting the leaf Node. ");
            inorder(root);
            System.out.println();
            leafDelete(root);
            System.out.println("INorder after Deleting the leaf Node. ");
            inorder(root);
      }
}
// This code is contributed by Prerna saini
```

Soru: Verilen bir ikili arama ağacındaki tüm sayıların toplamını bulan fonksiyonu yazınız.

```
// Java Program to print sum of
// all the elements of a binary tree
class GFG
{
static class Node
{
      int key;
      Node left, right;
}

/* utility that allocates a new
Node with the given key */
static Node newNode(int key)
{
      Node node = new Node();
      node.key = key;
      node.left = node.right = null;
      return (node);
}

/* Function to find sum
of all the elements*/
static int addBT(Node root)
{
      if (root == null)
            return 0;
      return (root.key + addBT(root.left) +
                        addBT(root.right));
}

// Driver Code
public static void main(String args[])
{
      Node root = newNode(1);
      root.left = newNode(2);
      root.right = newNode(3);
      root.left.left = newNode(4);
      root.left.right = newNode(5);
      root.right.left = newNode(6);
```

```
        root.right.right = newNode(7);
        root.right.left.right = newNode(8);

        int sum = addBT(root);
        System.out.println("Sum of all the elements is: " + sum);
}
}

// This code is contributed by Arnab Kundu
```

Soru :Verilen bir ağacın ikili arama ağacı koşullarını sağlayıp sağlamadıgını bulan fonksiyonu yazınız.

```
// Java program to check if a given tree is BST.
class Sol
{

// A binary tree node has data, pointer to
//left child && a pointer to right child /
static class Node
{
    int data;
    Node left, right;
};

// Returns true if given tree is BST.
static boolean isBST(Node root, Node l, Node r)
{
    // Base condition
    if (root == null)
        return true;

    // if left node exist then check it has
    // correct data or not i.e. left node's data
    // should be less than root's data
    if (l != null && root.data <= l.data)
        return false;

    // if right node exist then check it has
    // correct data or not i.e. right node's data
    // should be greater than root's data
    if (r != null && root.data >= r.data)
        return false;

    // check recursively for every node.
    return isBST(root.left, l, root) &&
        isBST(root.right, root, r);
}

// Helper function that allocates a new node with the
//given data && null left && right pointers. /
static Node newNode(int data)
{
    Node node = new Node();
    node.data = data;
    node.left = node.right = null;
    return (node);
}
```

```
// Driver code
public static void main(String args[])
{
	Node root = newNode(3);
	root.left = newNode(2);
	root.right = newNode(5);
	root.left.left = newNode(1);
	root.left.right = newNode(4);

	if (isBST(root,null,null))
		System.out.print("Is BST");
	else
		System.out.print("Not a BST");
}
}

// This code is contributed by Arnab Kundu
```

Soru : Verilen bir ikili arama ağacının tüm düğümlerinin sol ve sağ çocuklarını yer değiştiren fonksiyonu yazınız.

```
// Java program swap nodes
class GFG
{

// A Binary Tree Node
static class Node
{
	int data;
	Node left, right;
};

// function to create a new tree node
static Node newNode(int data)
{
	Node temp = new Node();
	temp.data = data;
	temp.left = temp.right = null;
	return temp;
}



// A utility function swap left- node & right node of tree
// of every k'th level
static void swapEveryKLevelUtil( Node root, int level, int k)
{
	// base case
	if (root== null ||
			(root.left==null && root.right==null) )
		return ;

	//if current level + 1 is present in swap vector
	//then we swap left & right node
	if ( (level + 1) % k == 0)
		{
			Node temp=root.left;
			root.left=root.right;
```

```
                root.right=temp;
            }

        // Recur for left and right subtrees
        swapEveryKLevelUtil(root.left, level+1, k);
        swapEveryKLevelUtil(root.right, level+1, k);
}

// This function mainly calls recursive function
// swapEveryKLevelUtil()
static void swapEveryKLevel(Node root, int k)
{
        // call swapEveryKLevelUtil function with
        // initial level as 1.
        swapEveryKLevelUtil(root, 1, k);
}

// Utility method for inorder tree traversal
static void inorder(Node root)
{
        if (root == null)
                return;
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
}

// Driver Code
public static void main(String args[])
{
        /* 1
                / \
        2 3
        / / \
        4 7 8 */
        Node root = newNode(1);
        root.left = newNode(2);
        root.right = newNode(3);
        root.left.left = newNode(4);
        root.right.right = newNode(8);
        root.right.left = newNode(7);

        int k = 2;
        System.out.println("Before swap node :");
        inorder(root);

        swapEveryKLevel(root, k);

        System.out.println("\nAfter swap Node :" );
        inorder(root);
}
}

// This code is contributed by Arnab Kundu
```

Soru : Verilen bir ikili arama ağacının derinliğini hesaplayan fonksiyonu yazınız.Bir ağacın derinliği o ağacın kök düğümünden yaprak düğümlerine giden yolların en uzunu olarak tanımlanır.

```
// Java program to find height of tree

// A binary tree node
class Node
{
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    Node root;

    /* Compute the "maxDepth" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
    int maxDepth(Node node)
    {
        if (node == null)
            return 0;
        else
        {
            /* compute the depth of each subtree */
            int lDepth = maxDepth(node.left);
            int rDepth = maxDepth(node.right);

            /* use the larger one */
            if (lDepth > rDepth)
                return (lDepth + 1);
            else
                return (rDepth + 1);
        }
    }

    /* Driver program to test above functions */
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();

        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("Height of tree is : " +
                                        tree.maxDepth(tree.root));
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Soru : Verilen bir ikili arama ağacındaki sayılar k1 ve k2 arasında olanları ekrana yazan fonksiyonu yazınız.

```
// Java program to print BST in given range

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int d) {
        data = d;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /* The functions prints all the keys which in
    the given range [k1..k2]. The function assumes than k1 < k2 */
    void Print(Node node, int k1, int k2) {

        /* base case */
        if (node == null) {
            return;
        }

        /* Since the desired o/p is sorted, recurse for left subtree first
        If root->data is greater than k1, then only we can get o/p keys
        in left subtree */
        if (k1 < node.data) {
            Print(node.left, k1, k2);
        }

        /* if root's data lies in range, then prints root's data */
        if (k1 <= node.data && k2 >= node.data) {
            System.out.print(node.data + " ");
        }

        /* If root->data is smaller than k2, then only we
        can get o/p keys in right subtree */
        if (k2 > node.data) {
            Print(node.right, k1, k2);
        }
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        int k1 = 10, k2 = 25;
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.right = new Node(22);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(12);

        tree.Print(root, k1, k2);
```

```
    }
}
```

// This code has been contributed by Mayank Jaiswal

Soru : Verilen bir ikili arama ağacındaki tüm düğümlerin toplam derinliğini hesaplayan fonksiyonu yazınız.

Soru :AVL ağacında durum 3’ü çözmek için uygulanan sağ çift rotasyon algoritmasını tek rotasyon fonksiyonlarını kullanmadan yazınız.