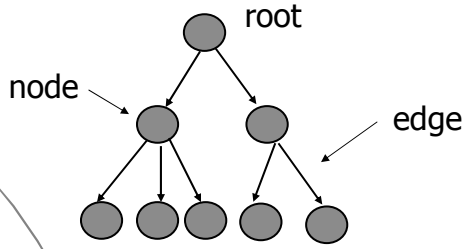
Ağaçlar (Trees)

- Giriş
- Binary Trees (İkilik Ağaçlar)
- Full Binary Trees
- Proper Binary Trees
- Complete Binary Trees
- Heap Binary Trees
- Balanced Binary Trees
- Binary Search Trees (İkilik Arama Ağaçları)

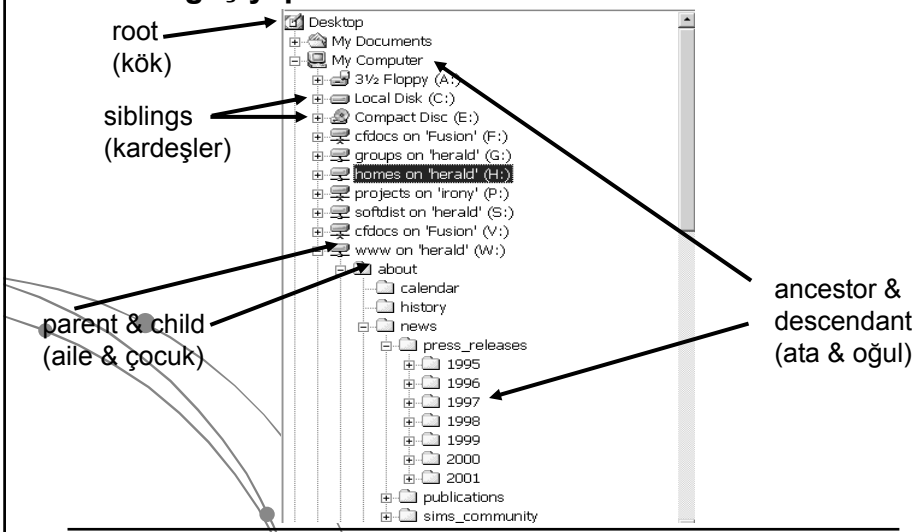
Ağaçlar (Trees)

- Ağaçlar hiyerarşik ilişkileri göstermek için kullanılır.
- Her ağaç node'lar ve kenarlardan (edge) oluşur.
- Herbir node bir nesneyi gösterir.
- Herbir kenar (bağlantı) iki node arasındaki ilişkiyi gösterir.
- Arama işlemi bağlı dizilere göre çok hızlı yapılır.



Ağaçlar (Trees)

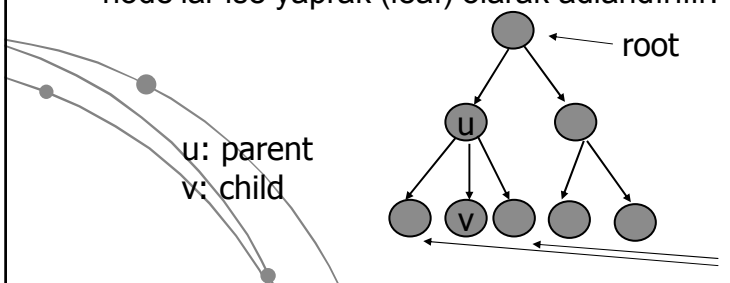
Örnek ağaç yapısı



Ağaçlar (Trees)

Terminoloji

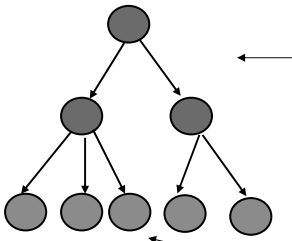
- u ve v gibi aralarında bir kenar olan iki node olsun. u parent node ve v ise child node olarak adlandırılır. Bu kenar (u, v) şeklinde gösterilir.
- Bir ağaçta parent'ı olmayan tek bir node vardır ve kök (root) diye adlandırılır. Child node'a sahip olmayan node'lar ise yaprak (leaf) olarak adlandırılır.



Ağaçlar (Trees)

Terminoloji (devam)

- En az bir child'a sahip olan node'lar internal node olarak adlandırlır.



internal node'lar

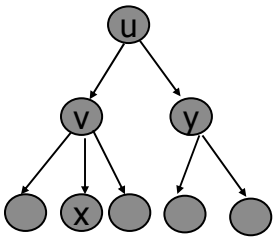
yapraklar (leaves - external)

Ağaçlar (Trees)

Terminoloji (devam)

- Aynı parent'a sahip iki node kardeş (siblings) olarak adlandırılır.
- Bir node parent'ı olduğu veya parent'ının parent'ı olduğu tüm node'ların atasıdır (ancestor).
- Bir node çocuğu olduğu veya çocuğunun çocuğu olduğu node'un oğuludur (descendant).

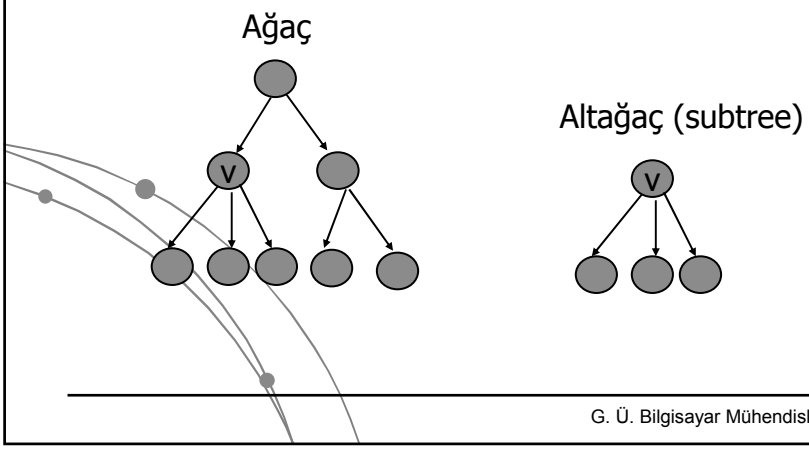
  - v ve y kardeş node'lar
  - u ve v node'ları x'in ataları
  - v ve x node'ları u'nun oğullarıdır



Ağaçlar (Trees)

Terminoloji (devam)

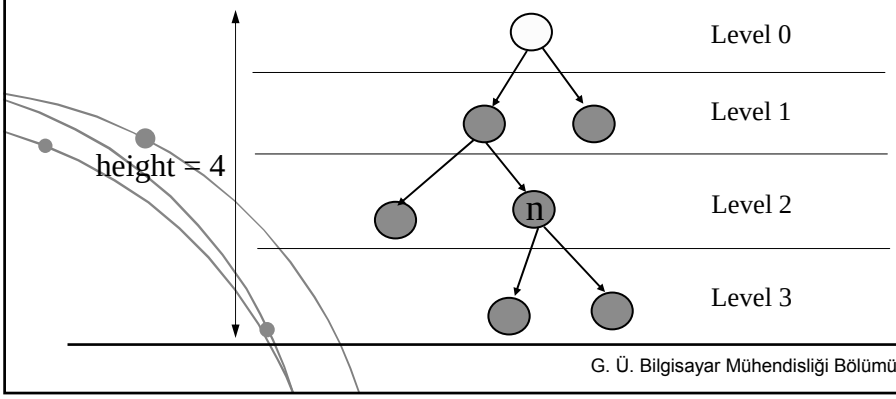
- Bir altağaç (subtree) bütün oğullarıyla birlikte herhangi bir node'dur.



Ağaçlar (Trees)

Terminoloji (devam)

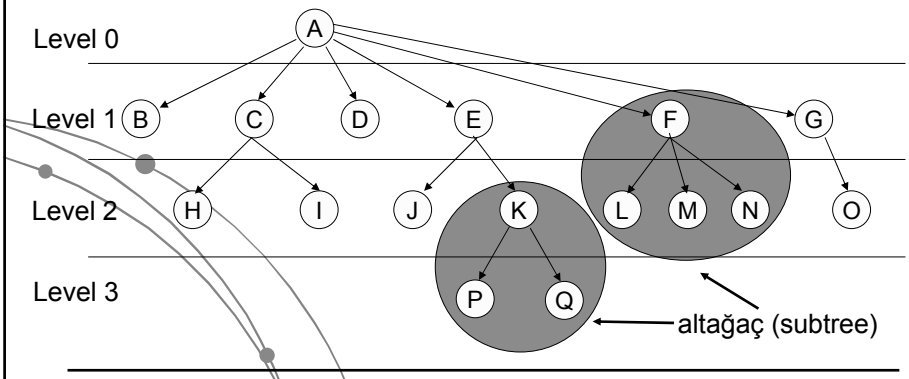
- Seviye (Level - Depth) : root node'undan herhangi bir n node'una giden yolda geçilen node sayısıdır.
- Yükseklik (Height) : root ile yaprak node'ları arasındaki en yüksek seviyedir.



Ağaçlar (Trees)

Terminoloji (örnek)

- A tüm node'ların atasıdır (ancestor).
- B…G node'ları kardeştir (siblings).
- H ve I node'ları C node'unun çocuklarıdır (children).
- P ve Q node'ları A, E, K node'larının oğullarıdır (descendants).



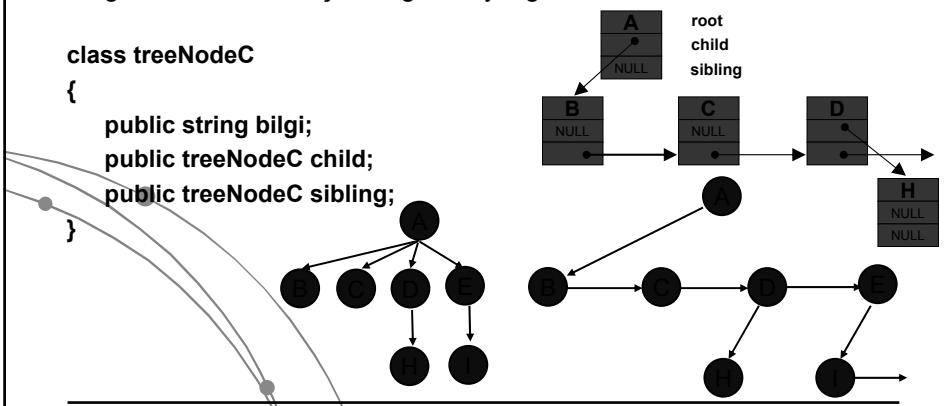
Ağaçlar (Trees)

Ağaç oluşturma

1- Herbir node kendisine ait olan tüm child node'ları gösterebilir. Ancak child sayısı değişken olduğu için infeasible'dir.

2- Herbir node kendisine ait olan child ile kardeşlerinin tümü gösterebilir. Kardeşler bağlı listeyle gösterilir.

```
class treeNodeC
{
    public string bilgi;
    public treeNodeC child;
    public treeNodeC sibling;
}
```



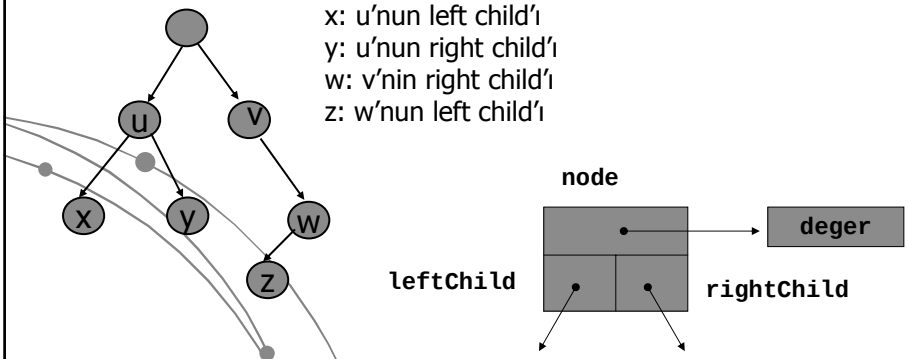
Ağaçlar (Trees)

Binary Trees (İkilik Ağaçlar)

1- Herbir node en fazla iki tane child node'a sahip olabilir.

2- Left child : Bir node'un sol işaretçisine bağlıdır.

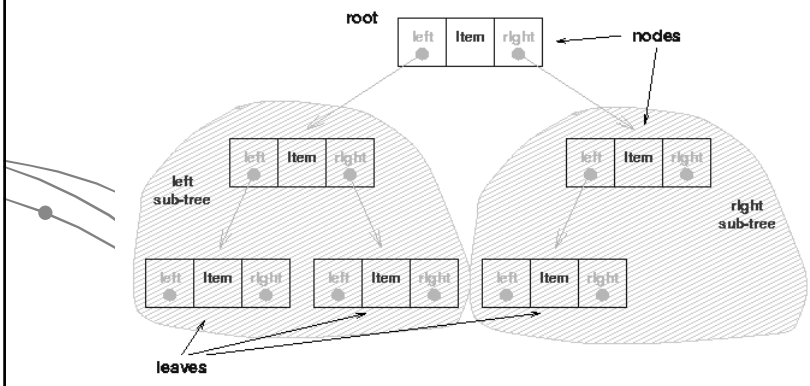
3- Right child : Bir node'un sağ işaretçisine bağlıdır.



Ağaçlar (Trees)

Binary Trees (İkilik Ağaçlar)

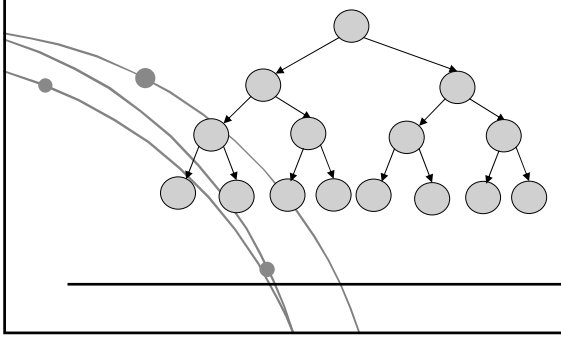
1- rootNode'un sağ ve sol işaretçilerinin gösterdiği her node bir altağacı gösterir.



Ağaçlar (Trees)

Full Binary Trees

- Eğer T ağacı boş ise, T yüksekliği 0 olan bir full binary ağaçtır.
- T ağacının yüksekliği h ise ve yüksekliği h'den küçük olan tüm node'lar iki child node'a sahipse, T full binary tree'dir.
- Full binary tree'de her node aynı yüksekliğe eşit sağ ve sol altağaçlara sahiptir.

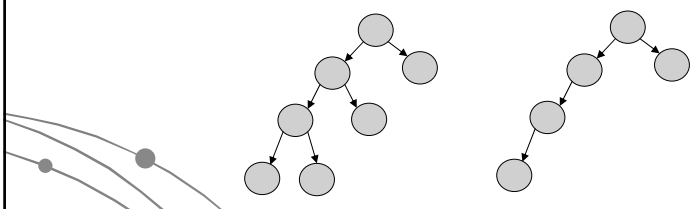


Yüksekliği 4 olan full binary tree

Ağaçlar (Trees)

Proper Binary Trees

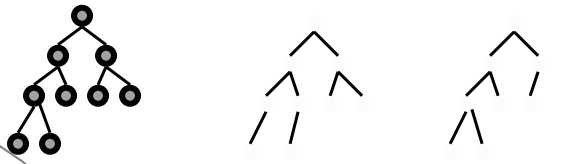
- Eğer T ağacının her node'u ya hiç child node'a sahip değilse veya iki node'a sahipse T ağacı proper binary tree olarak adlandırılır.



Ağaçlar (Trees)

Complete Binary Trees

- T, n yükseklikte complete binary tree ise, tüm yaprak node'ları n veya n-1 derinliğindedir ve yeni bir derinliğe soldan sağa doğru ekleme başlanır.
- Her node iki tane child node'a sahip olmayabilir.

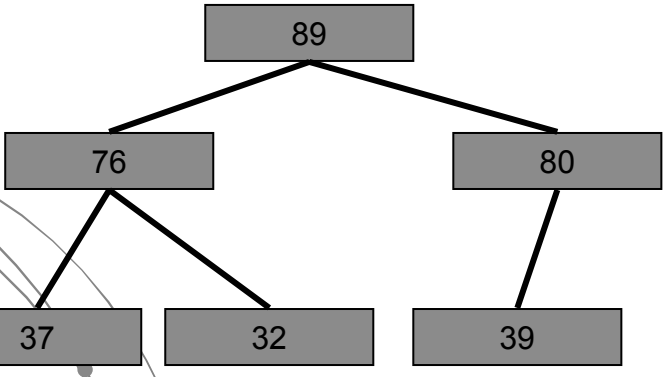


complete incomplete incomplete

Ağaçlar (Trees)

Heap

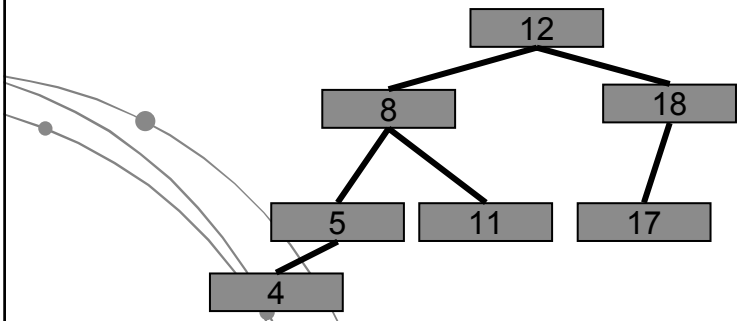
- Herbir node kendi child node'larından büyük veya eşittir.
- Sıralama işlemlerinde kullanılır (Heap Sort).



Ağaçlar (Trees)

Balanced Binary Trees

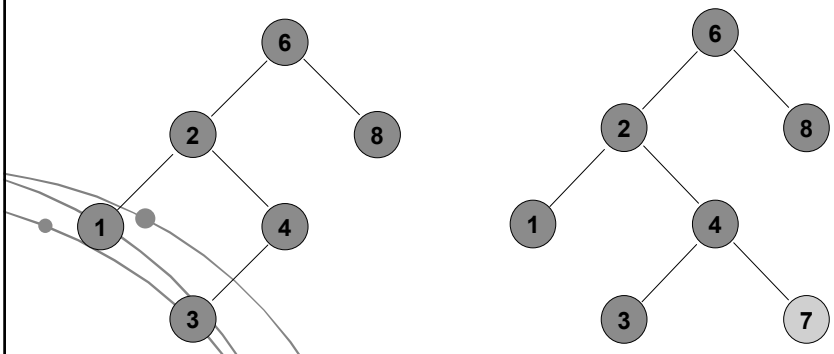
- Yüksekliği ayalanmış ağaçlardır.
- Bütün node'lar için sağ altağacın yüksekliği ile sol altağacın yüksekliği arasında en fazla bir fark varsa balanced binary tree olarak adlandırılır.
- Complete binary tree'ler aynı zamanda balanced binary tree'dir.
- Her balanced binary tree, complete binary tree olmayabilir.



Ağaçlar (Trees)

Binary Search Trees (İkilik Arama Ağaçları)

- Herbir node'un sağındaki tüm node'lar kendisinden büyük ve solundaki tüm node'lar kendisinden küçüktür.



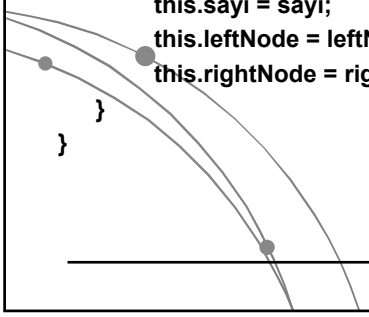
Binary search tree

Binary tree

Ağaçlar (Trees)

İkilik Arama Ağacı Oluşturmak

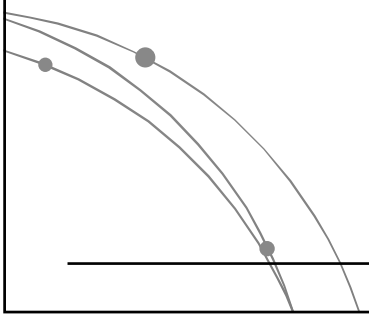
```
class bstNodeC
{
    public int sayi;
    public bstNodeC leftNode, rightNode;
    public bstNodeC(int sayi, bstNodeC leftNode, bstNodeC rightNode)
    {
        this.sayi = sayi;
        this.leftNode = leftNode;
        this.rightNode = rightNode;
    }
}
```



Ağaçlar (Trees)

İkilik Arama Ağacında İşlemler

- Dolaşma (Traverse)
- Boşaltma (Make Empty)
- Arama (Find)
- Ekleme (Append)
- Silme (Delete)

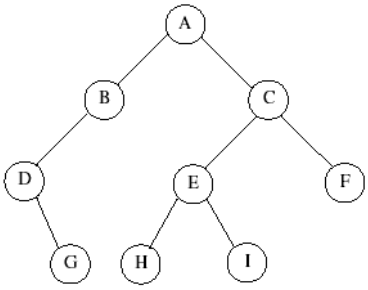


Ağaçlar (Trees)

İkilik Arama Ağacında Dolaşma (Traversal)

- Bir ikilik ağaçta tüm node'lar için aynı işlem yapılmak istendiğinde kullanılır. (Örn: Tüm değerleri bir sayıyla çarpmak veya belirli bir kriteri sağlayan değerleri bulmak v.b.)
- Tüm node'ları dolaşmak için bir dolaşma algoritmasına ihtiyaç duyulur.
- Dolaşma algortimalarında ve recursive yapı kullanılır.
- Dolaşma algoritmaları genel olarak 4 çeşittir:
  - **Preorder (Önce kök)**
  - **Inorder (Ortada kök)**
  - **Postorder (Sonra kök)**
  - **Level order**

Preorder = ABDGCEHIF
Inorder = DGBAHEICF
Postorder = GDBHIEFCA
Level order = ABCDEFGHI



Ağaçlar (Trees)

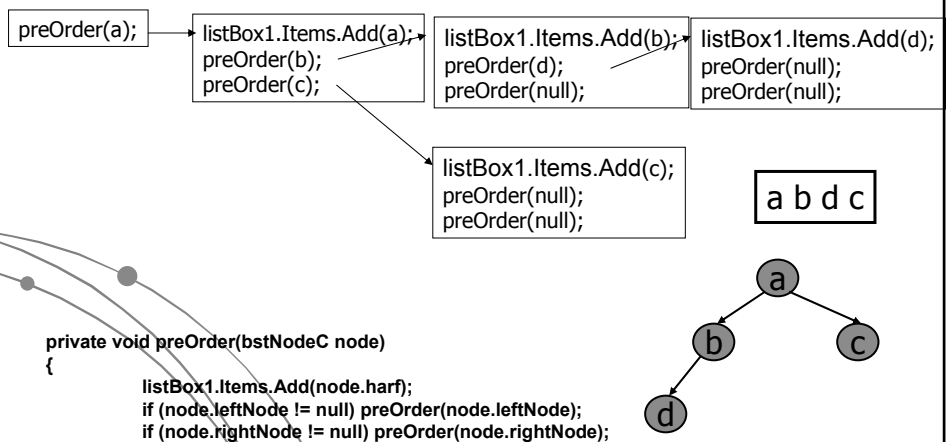
Preorder dolaşma

- Köke uğra (V)
- Sol alt ağacı preorder olarak dolaş (L)
- Sağ alt ağacı preorder olarak dolaş (R)

```
private void preOrder(bstNodeC node)
{
    listBox1.Items.Add(node.sayi);
    if (node.leftNode != null) preOrder(node.leftNode);
    if (node.rightNode != null) preOrder(node.rightNode);
}
```

Ağaçlar (Trees)

Preorder dolaşma (örnek)




```
private void preOrder(bstNodeC node)
{
        listBox1.Items.Add(node.harf);
        if (node.leftNode != null) preOrder(node.leftNode);
        if (node.rightNode != null) preOrder(node.rightNode);
}
```


Ağaçlar (Trees)

Inorder dolaşma

- Sol alt ağacı inorder olarak dolaş (L)
- Köke uğra (V)
- Sağ alt ağacı inorder olarak dolaş (R)

```
private void inOrder(bstNodeC node)
{
    if (node.leftNode != null) inOrder(node.leftNode);
    listBox1.Items.Add(node.sayi);
    if (node.rightNode != null) inOrder(node.rightNode);
}
```

Ağaçlar (Trees)

Postorder dolaşma

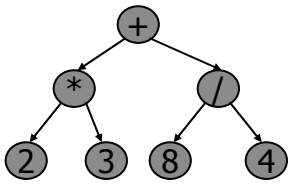
- Sol alt ağacı postorder olarak dolaş (L)
- Sağ alt ağacı postorder olarak dolaş (R)
- Köke uğra (V)

```
private void postOrder(bstNodeC node)
{
    if (node.leftNode != null) postOrder(node.leftNode);
    if (node.rightNode != null) postOrder(node.rightNode);
    listBox1.Items.Add(node.sayi);
}
```


Ağaçlar (Trees)

Dolaşma algoritmaları (örnek expression tree)

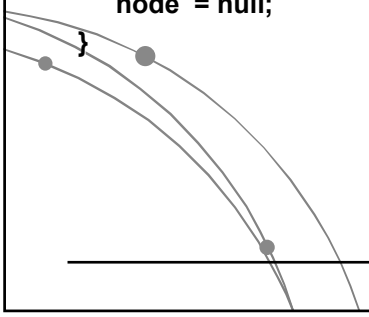
- preOrder, (prefix)
  - + * 2 3 / 8 4
- inOrder, (infix)
  - 2 * 3 + 8 / 4
- postOrder, (postfix)
  - 2 3 * 8 4 / +
- levelOrder,
  - + * / 2 3 8 4



Ağaçlar (Trees)

Boşaltma (makeEmpty)

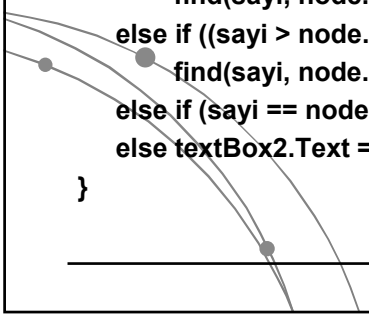
```
private void makeEmpty(bstNodeC node)
{
    if (node.leftNode != null) makeEmpty(node.leftNode);
    if (node.rightNode != null) makeEmpty(node.rightNode);
    node  = null;
}
```



Ağaçlar (Trees)

Arama (find)

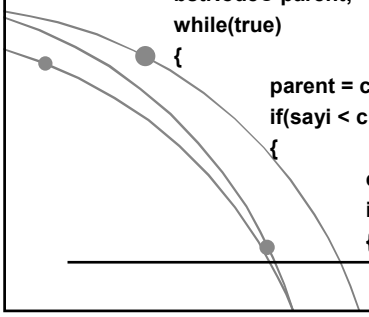
```
private void find(int sayi, bstNodeC node)
{
    if (node.sayi == 0) textBox2.Text = "Boş ağaç";
    else if ((sayi < node.sayi)&&(node.leftNode!=null))
          find(sayi, node.leftNode);
    else if ((sayi > node.sayi)&&(node.rightNode!=null))
          find(sayi, node.rightNode);
    else if (sayi == node.sayi) textBox2.Text = “Sayı bulundu";
    else textBox2.Text = “Sayı yok";
}
```



Ağaçlar (Trees)

Ekleme (append)

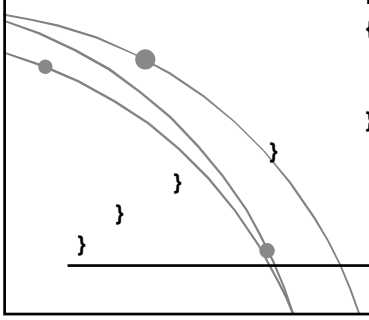
```
private void append(int sayi, bstNodeC node)
{
    bstNodeC yeniNode = new bstNodeC(sayi, null, null);
    if (node.sayi == 0) node.sayi = sayi;
    else
    {
        bstNodeC current = node;
        bstNodeC parent;
        while(true)
        {
            parent = current;
            if(sayi < current.sayi)
            {
                current = current.leftNode;
                if(current == null)
                {
```



Ağaçlar (Trees)

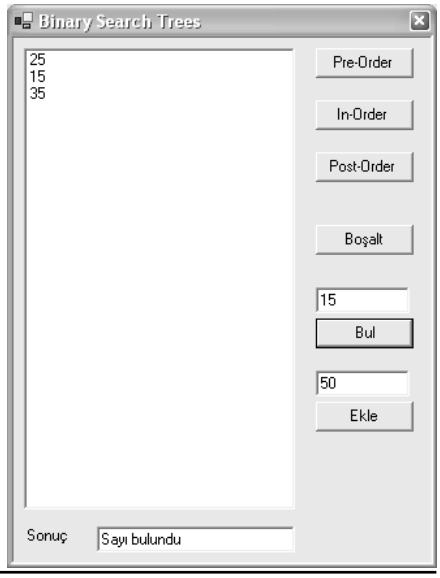
Ekleme (append) - devam

```
                    parent.leftNode = yeniNode;
                        break;
                    }
                }
                else
                {
                    current = current.rightNode;
                    if(current==null)
                    {
                        parent.rightNode = yeniNode;
                        return;
                    }
                }
            }
        }
    }
}
```



Ağaçlar (Trees)

Örnek program:



Ağaçlar (Trees)

Haftalık Ödev:

- **İkilik binary ağacında minimum ve maksimum değeri bulan metodları yazınız.**
- **İkilik binary ağacında lever-order dolaşmayı yapınız. (Kuyruk yapısı kullanılacak)**
- **İkilik binary arama ağacında silme işlemini yapınız.**