BINARY SEARCH TREE (BST)

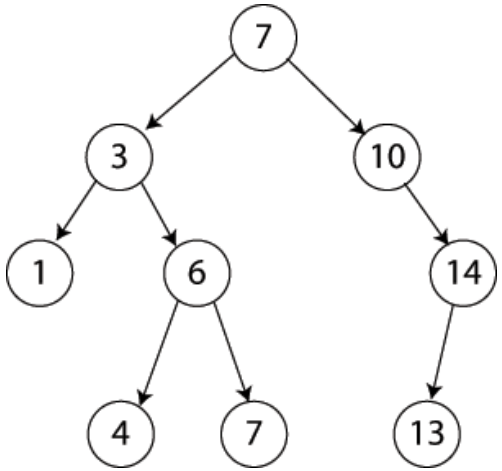
Binary Search Tree (BST)

- **Binary Search Tree** (BST) merupakan tree yang terurut (*ordered Binary Tree*) yang memiliki kelebihan bila dibanding dengan struktur data lain.
- **Kelebihan BST** adalah proses pengurutan (*sorting*) dan pencarian (*searching*) dapat dilakukan bila data sudah tersusun dalam struktur data BST.

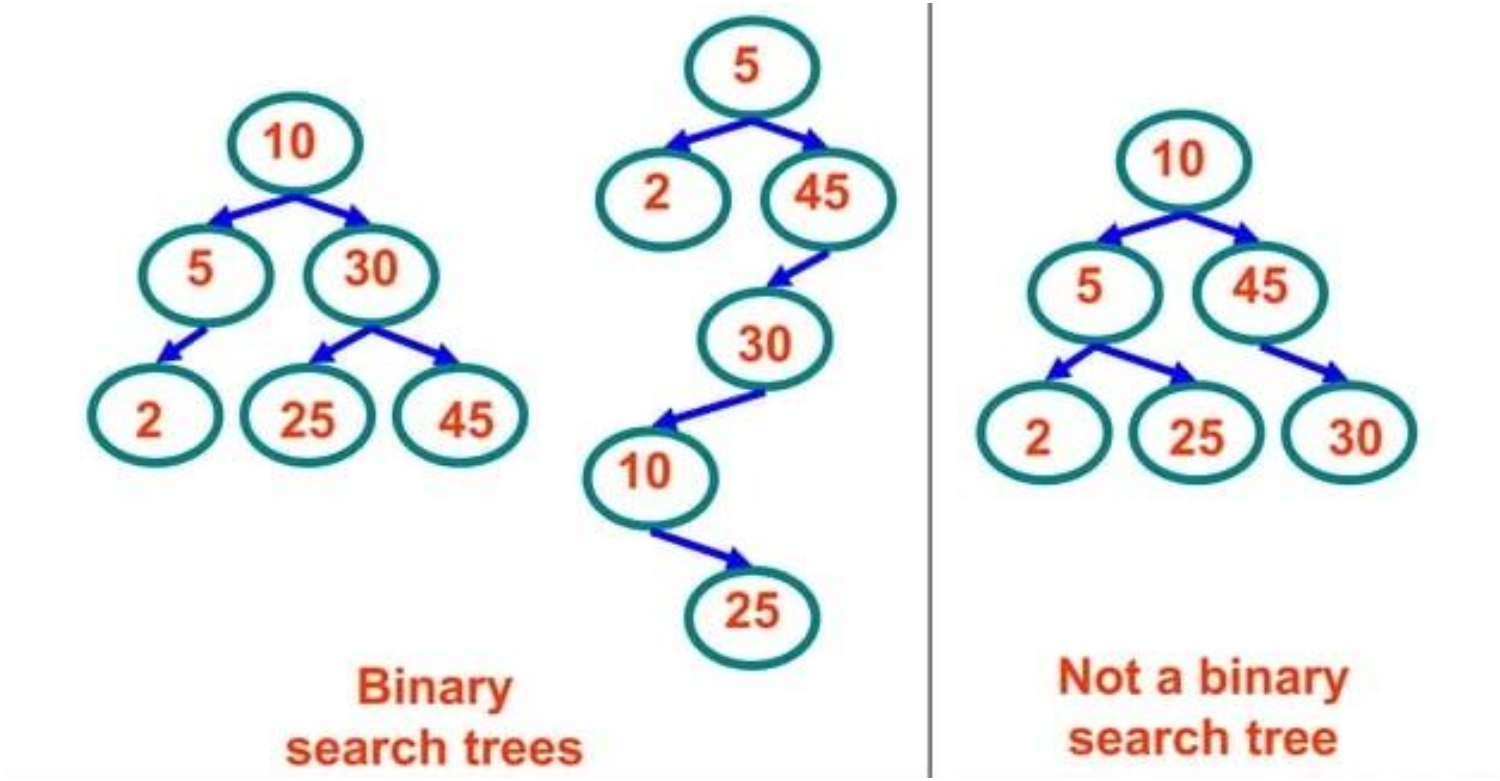
ATURAN MEMBANGUN BST

- Semua data dibagian kiri *sub-tree* dari *node t* selalu lebih kecil daridata dalam *node t* itu sendiri.
- Semua data dibagian kanan *sub-tree* dari *node t* selalu lebih besar atau sama dengan data dalam *node t*.
- Contoh BST:

BST berukuran 9 dengan kedalaman 3 dengan node daun (leaf) adalah 1, 4, 7 dan 13.

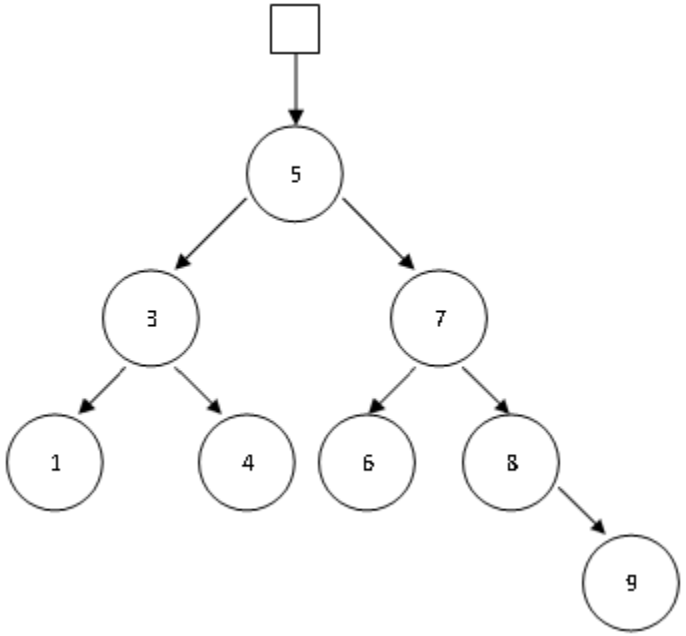


Binary Search Tree (BST)



CONTOH BST

Contoh:
Bila diketahui sederetan data 5, 3, 7, 1, 4, 6, 8, 9 maka pemasukan data tersebut dalam struktur data BST



Perbedaan Binary Tree (BT) dan BST

- BT merupakan bentuk sederhana dari sebuah tree dengan masing-masing node dapat memiliki paling banyak dua anak.
- BST merupakan BT dengan node diberi nilai yang memiliki batasan berikut ini:

a. Tidak ada duplikat nilai

b. Subtree bagian kiri dari node hanya dapat memiliki nilai lebih kecil dari node.

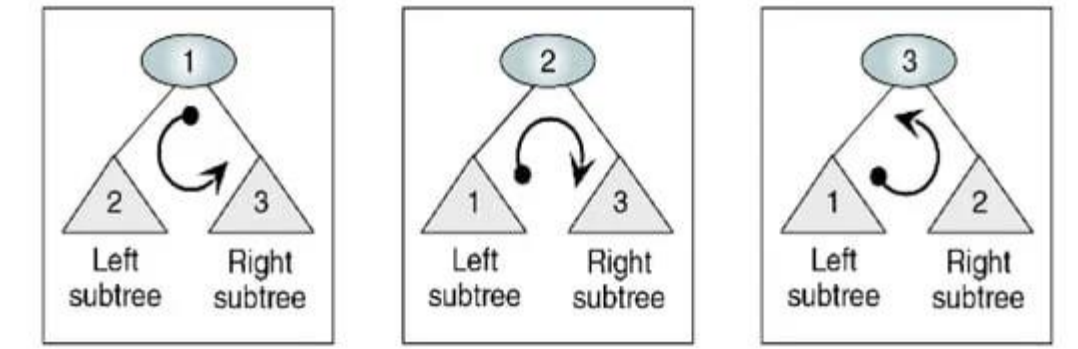
c. Subtree bagian kanan dari node hanya dapat memiliki nilai lebih besar dari node dan didefinisikan secara rekursif.

d. Subtree bagian kiri dari node adalah sebuah BST

e. Subtree bagian kanan dari node adalah sebuah BST.

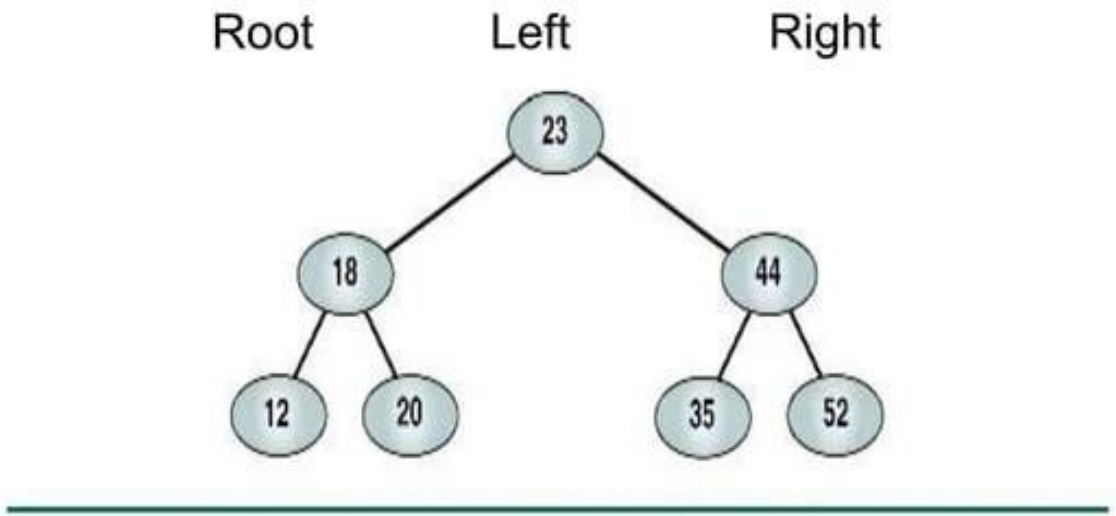
BASIC OPERASI BST

BST TRAVERSAL



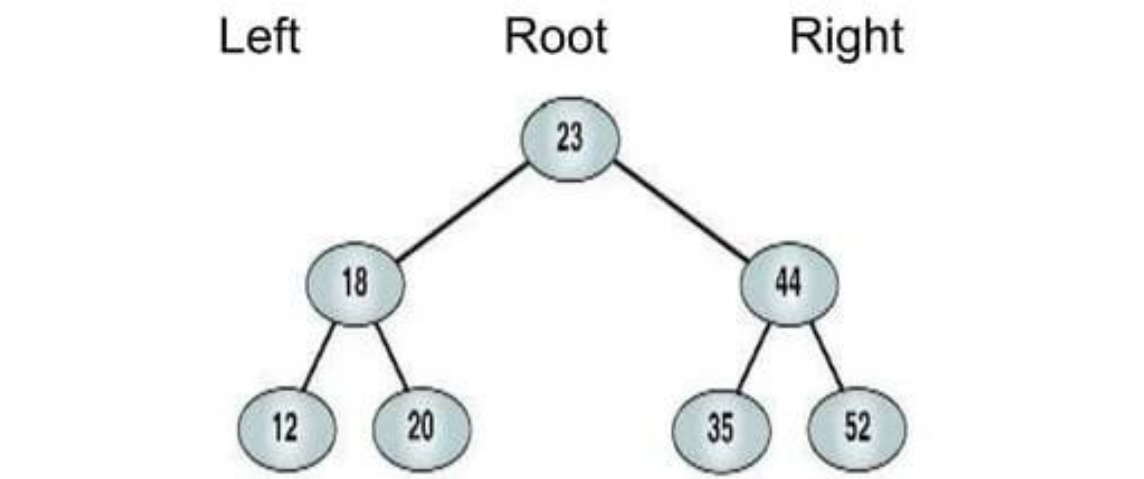
(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

PREORDER TRAVERSAL



23 18 12 20 44 35 52

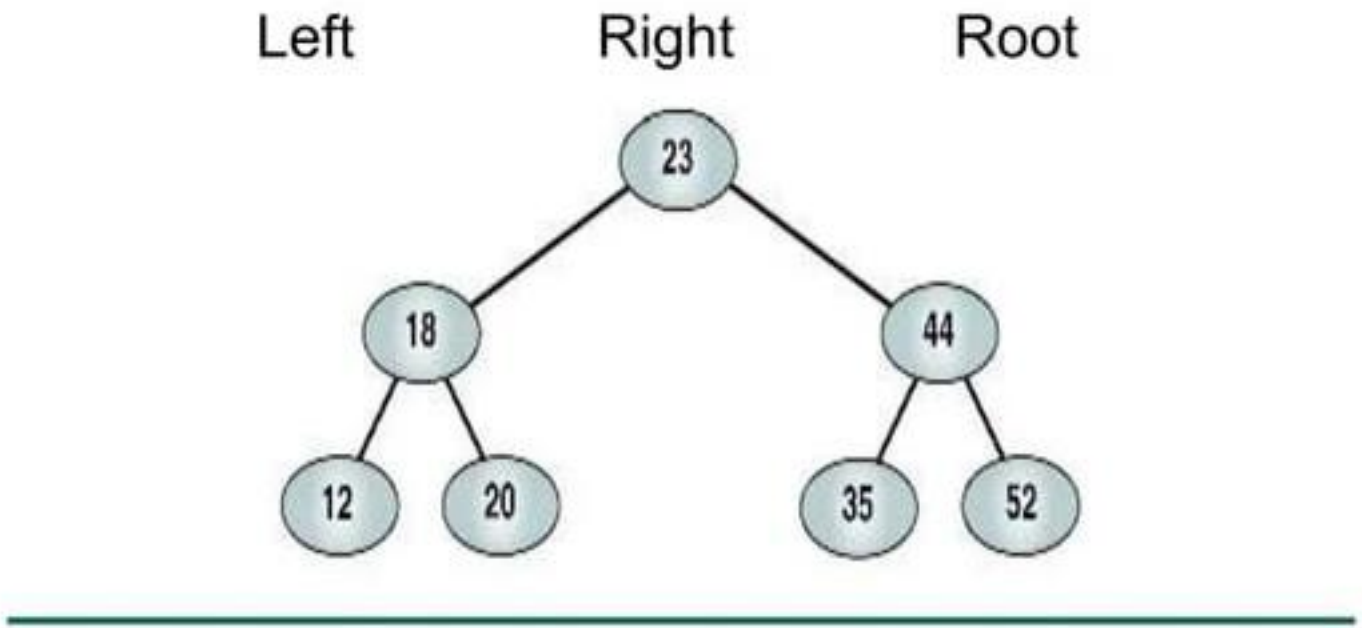
INORDER TRAVERSAL



12 18 20 23 35 44 52

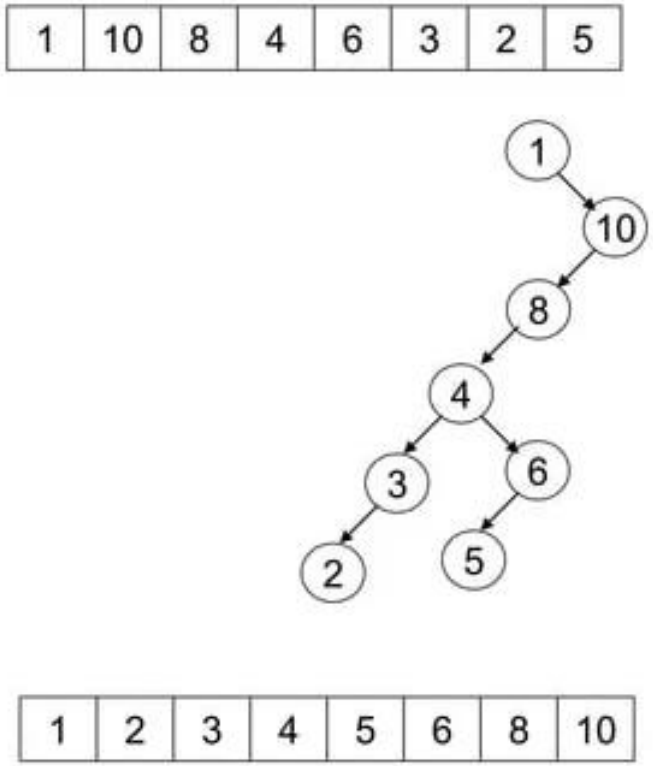
Produces a sequenced list

POSTORDER TRAVERSAL



12 20 18 35 52 44 23

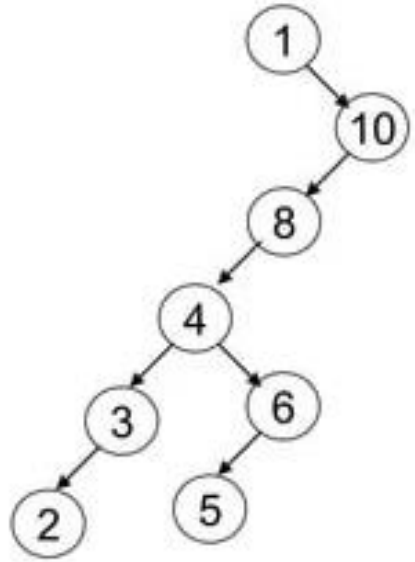
INORDER TRAVERSAL



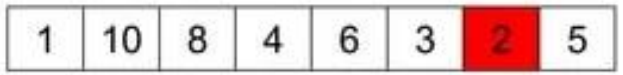
BINARY TREE INSERTION

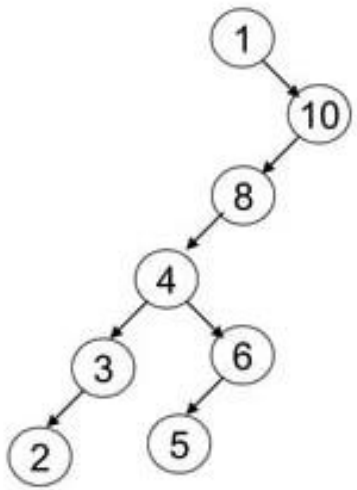
1	10	8	4	6	3	2	5

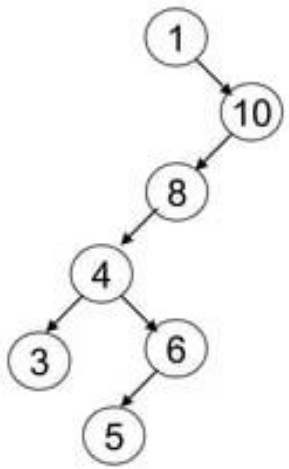
1	10	8	4	6	3	2	5



BINARY TREE DELETION







CONTOH PROGRAM SEARCH PADA BST

```
bool BST::search(int x){
    node *tmp = root;
    while(tmp!=NULL){
        if(x == tmp->item)
            return true;
        if(x < tmp->item)
            tmp = tmp->left;
        else
            tmp = tmp->right;
    }
    return false;
}
```

CONTOH PROGRAM INSERTION PADA BST

```
void BST::insert(int x){
    node *n = new node(x);
    node *tmp = root;
    if(tmp = NULL)
        root = n;
    else{
        node *tmp2;
        while(tmp!=NULL){
            tmp2 = tmp;
            if(x < tmp->item)
                tmp = tmp->left;
            else
                tmp = tmp->right;
        }
        if(x < tmp2->item)
            tmp2->left = n;
        else
            tmp2->right = n;
    }
}
```

CONTOH PROGRAM DELETE PADA BST

DELETE NODE

```
bool BST::deleteNode(int x){
    node *del = searchNode(x);
    if(del->left == NULL && del->right==NULL)
        delete del; //leaf
    else{

    }
}
```

CONTOH PROGRAM DELETE PADA BST

ONE CHILD

```
if(del->left==NULL)
   del = del->right;
else
if(del->right==NULL)
   del = del->left;
```

CONTOH PROGRAM DELETE PADA BST

TWO CHILDREN

```
else{
   node *ptr = minimum(del->right);
   int x = ptr->item;
   deleteNode(x);
   del->item = x;
}
```

TIPE BST

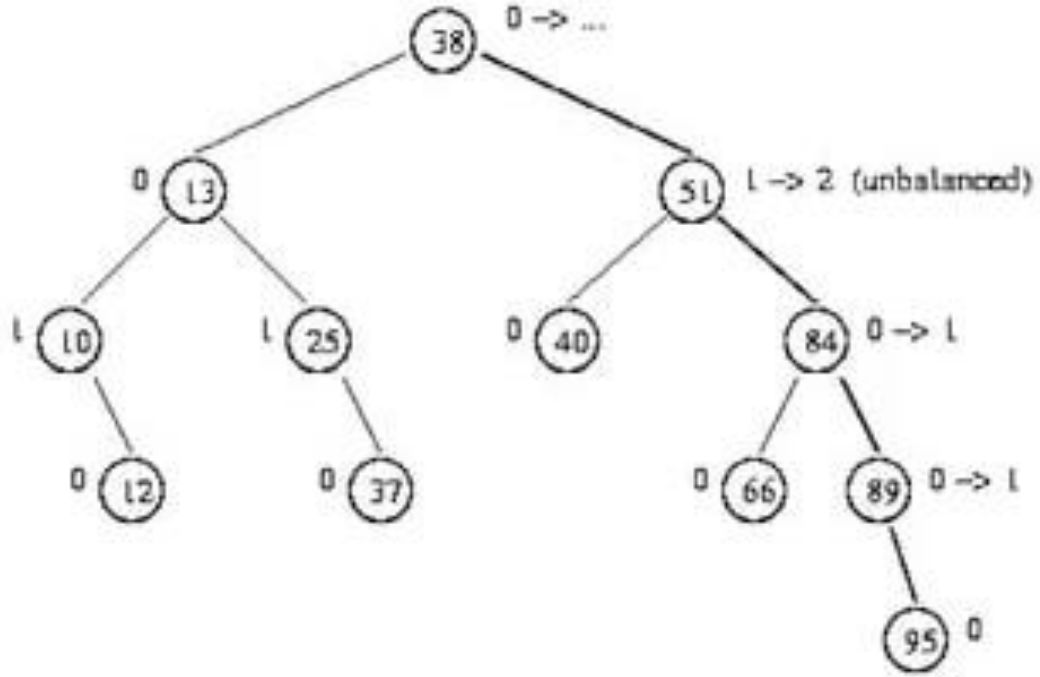
- AVL Tree
- Red-black Tree
- Splay Tree
- B-tree
- 2-4 tree

ADELSON-VELSKI LANDIS (AVL) TREE

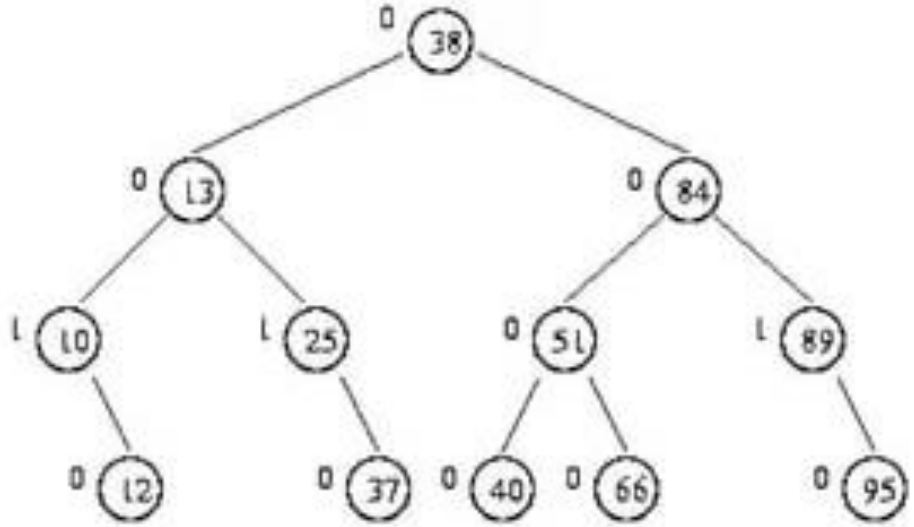
- AVL Tree adalah Binary Search Tree yang dimodifikasi untuk meningkatkan kinerjanya.
- AVL tree merupakan sebuah self-balancing BST yang perbedaan antara height dari subtree kiri dan kanan maksimal hanya 1 untuk semua node.
- operasi yang dapat dilakukan terhadap sebuah AVL Tree adalah sama dengan operasi yang ada pada Binary Search Tree dengan sedikit modifikasi setelah melakukan insert maupun delete.
- Tambahan modifikasi AVL tree adalah proses rebalancing (penyeimbangan kembali) dari bentuk tree agar tetap memenuhi kaidah dan sifat-sifat sebuah binary tree.
- Proses delete node dapat dilakukan pada node internal ataupun node eksternal (node leaf).

AVL TREE

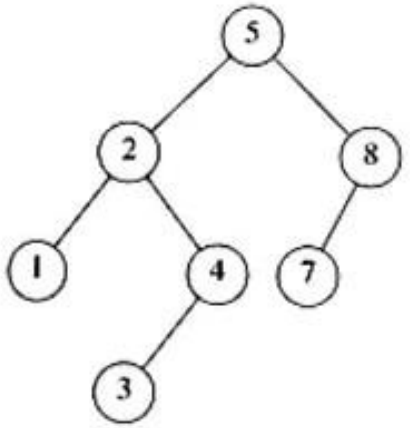
The tree became unballanced after inserting key 95.



After the tree is rebalanced using rotation we have:

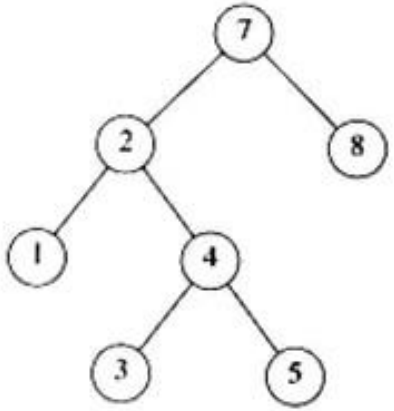


AVL TREE



YES

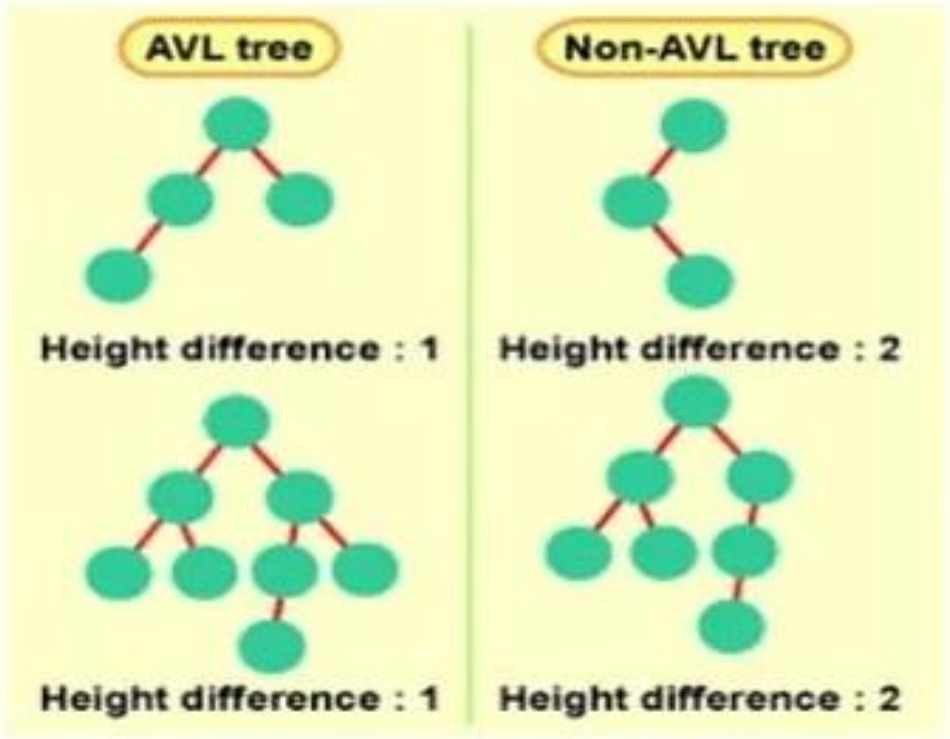
Each left sub-tree has height 1 greater than each right sub-tree



NO

Left sub-tree has height 3, but right sub-tree has height 1

AVL TREE


AVL tree
Non-AVL tree
Height difference : 1
Height difference : 2
Height difference : 1
Height difference : 2

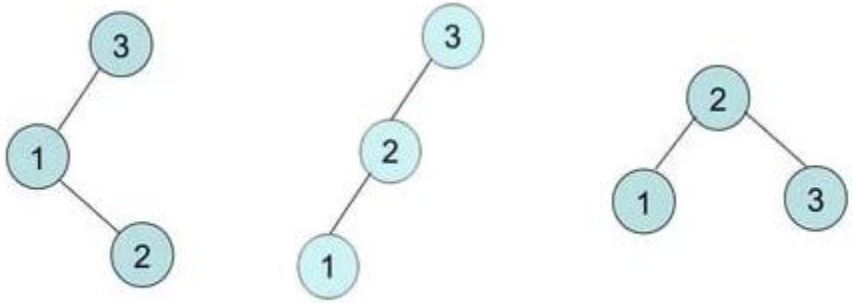
AVL TREE

Right Rotate

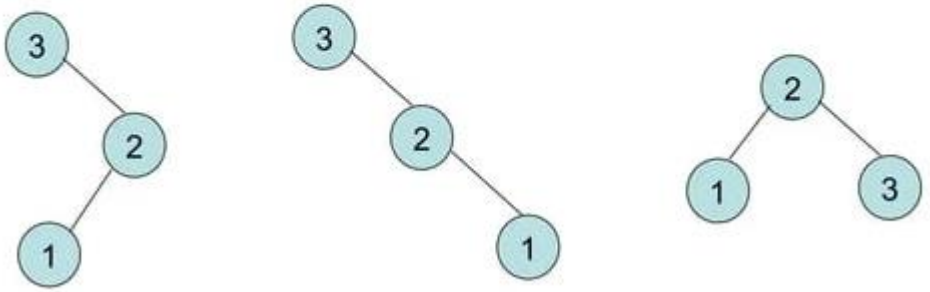
Left-Rotate

AVL TREE

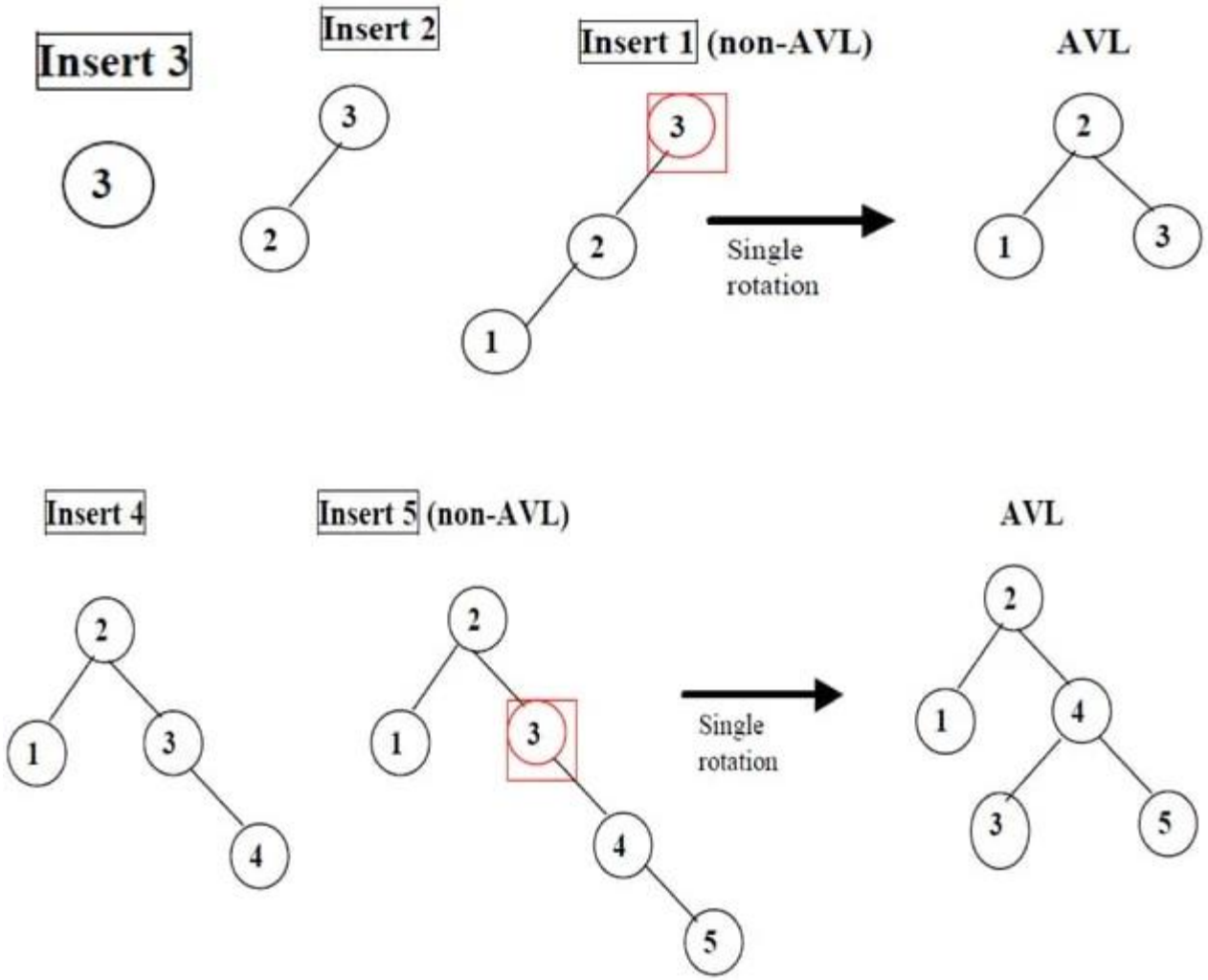
Left-Right Rotate



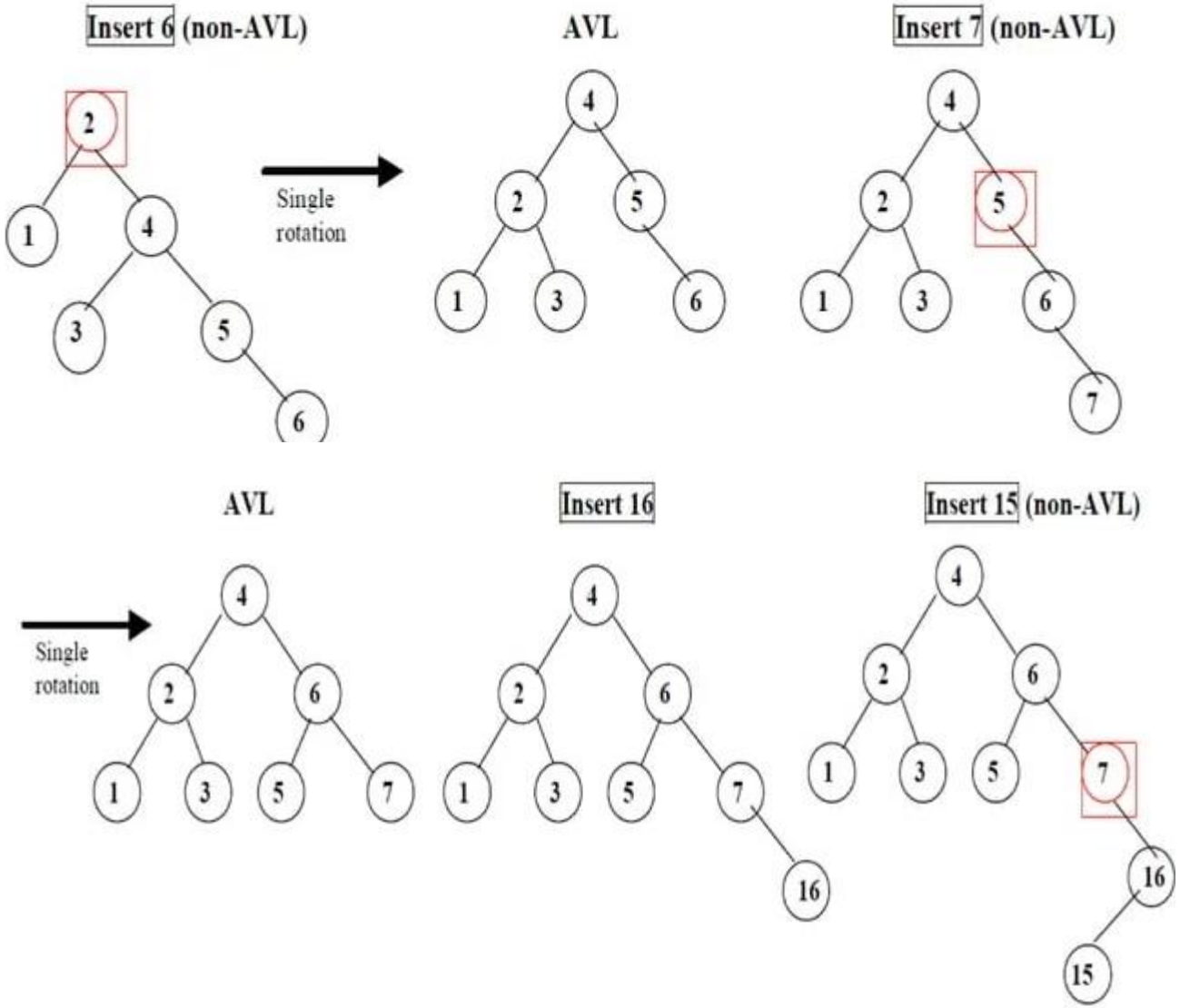
Right-Left Rotate



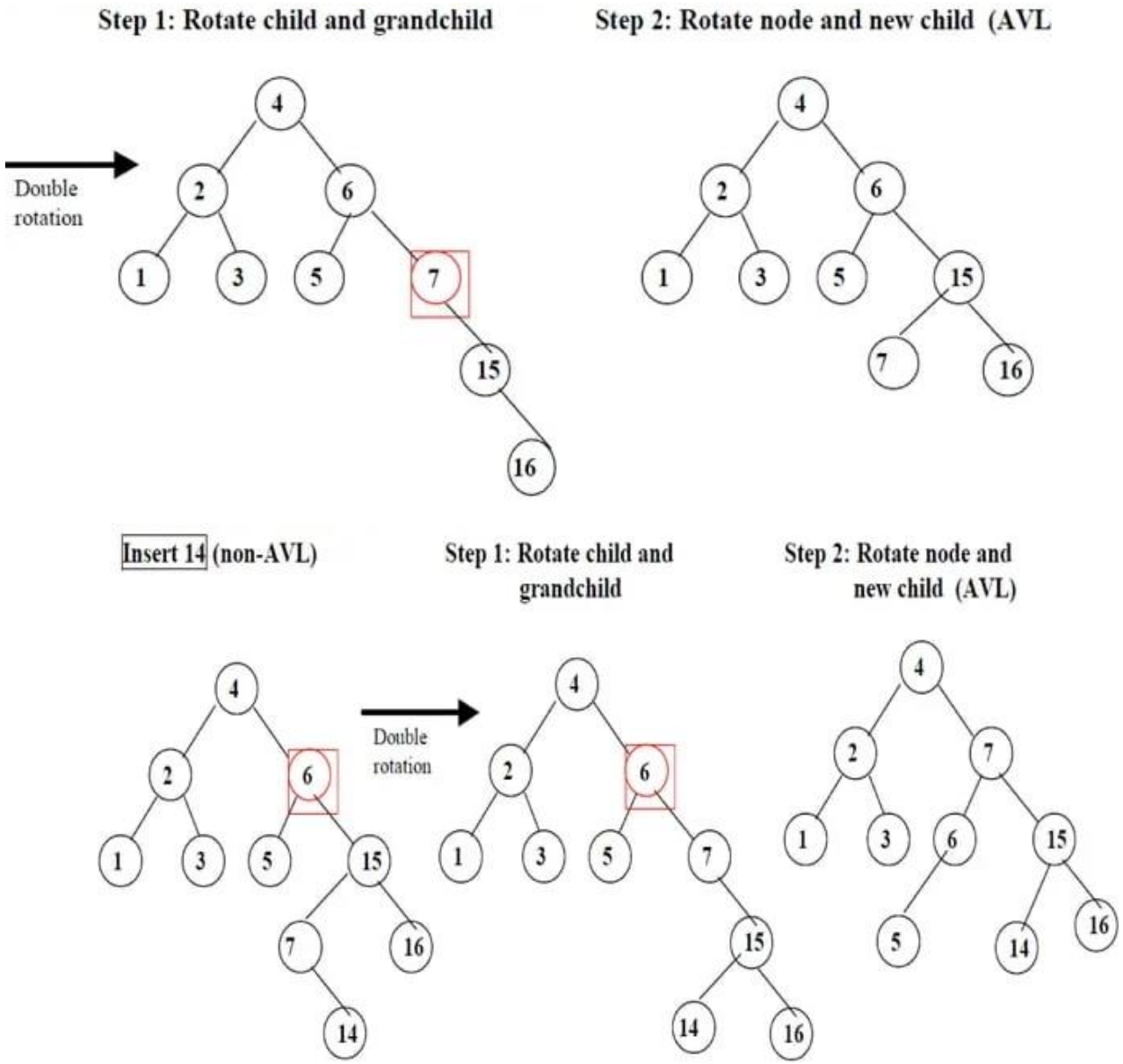
CONTOH AVL TREE



CONTOH AVL TREE



CONTOH AVL TREE



RED BLACK TREE

- Setiap node memiliki kedua warna, merah dan hitam
- Root dari tree adalah selalu hitam.
- Tidak ada 2 node merah yang berdekatan (sebuah node merah tidak dapat memiliki sebuah red parent atau red child.
- Setiap jalur dari root ke sebuah node Null memiliki jumlah node hitam yang sama.

RED BLACK TREE

A chain of 3 nodes is nodes is not possible in Red-Black Trees.

Following are NOT Red-Black Trees

Following are different possible Red-Black Trees with above 3 keys

SPLAY TREE

- Secara otomatis memindahkan elemen yang sering diakses lebih dekat ke root untuk akses cepat.

TERIMA KASIH