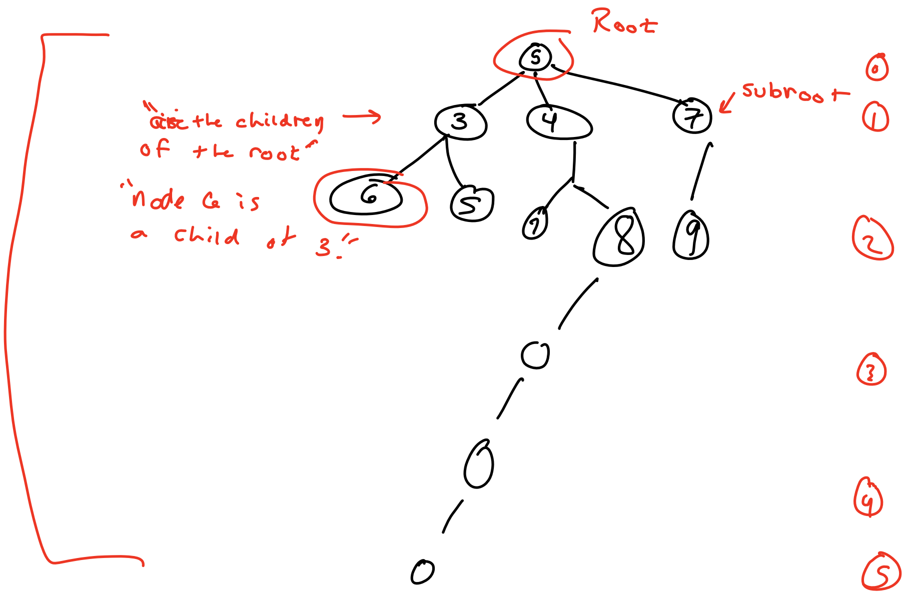
Tree

height of a tree is 6.

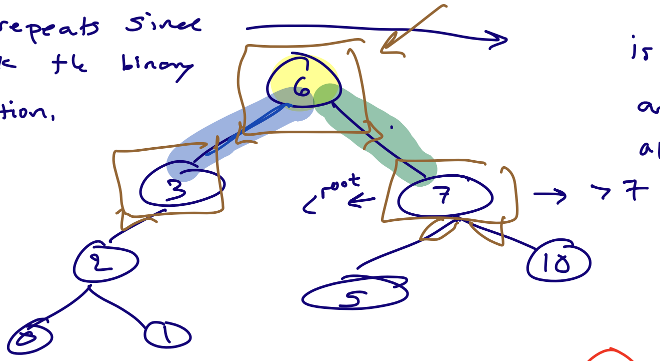


example non-binary tree since the root of the tree has more than 2 children.

Def A binary is such that every node has at most 2 children.

Binary Search Tree to hold data in sorted order,

→ Does not allow repeats since we'd then break the binary tree condition.



is the left child always < root and the right is always > root.

how do we represent btrees?

```
typedef struct binary_tree_node bstnode;
struct binary-tree node {
    void *item;
    bstnode * lsub;
    bstnode * rsub;
};
```

```
typedef struct {
    bstnode * root;
} bst;
```

→ typedef (bstnode*) bst;

2.

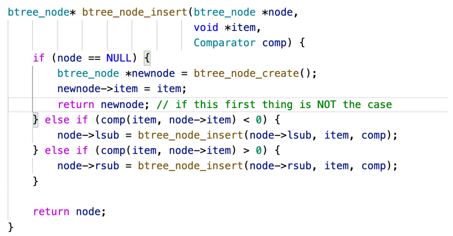
1. create/Destroy
2. Iterate
3. Insert / remove
4. how many elements
5. height of tree.

```
typedef struct llist_node_ llist_node;
Struct llist_node_ {
        void *item;
        llist_node *next;
};
```

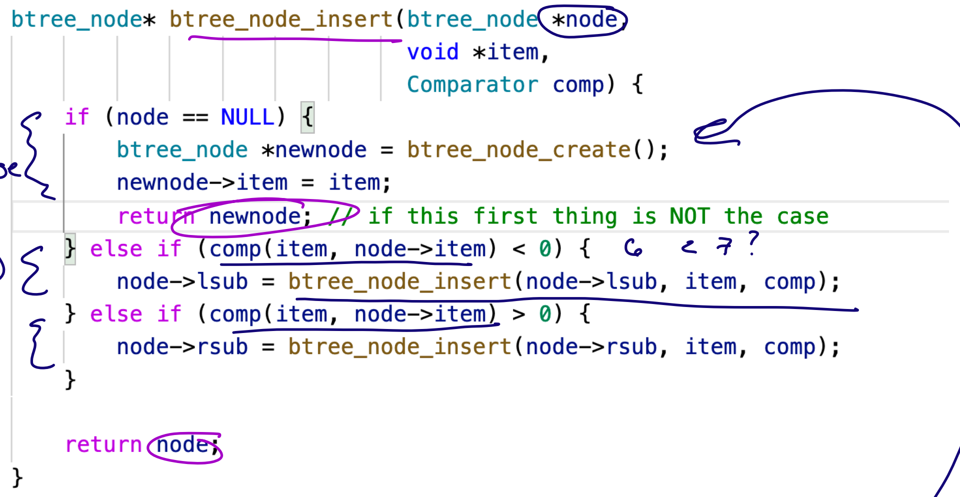
```
typedef struct {
        llist_node *head;
} llist;
```

Encapsulation → linked List

```
llist → head

llist → head = NULL;

llist_node *myList = NULL;

list *myList = malloc(sizeof(llist));
    myList → head = NULL;
```



```
btree_node* btree_node_insert(btree_node *node,
                              void *item,
                              Comparator comp) {
    if (node == NULL) {
        btree_node *newnode = btree_node_create();
        newnode->item = item;
        return newnode; // if this first thing is NOT the case
    } else if (comp(item, node->item) < 0) {
        node->lsub = btree_node_insert(node->lsub, item, comp);
    } else if (comp(item, node->item) > 0) {
        node->rsub = btree_node_insert(node->rsub, item, comp);
    }

    return node;
}
```



```
btree_node* btree_node_insert(btree_node *node,
                              void *item,
                              Comparator comp) {
    if (node == NULL) {
        btree_node *newnode = btree_node_create();
        newnode->item = item;
        return newnode; // if this first thing is NOT the case
    } else if (comp(item, node->item) < 0) {   6 < 7 ?
        node->lsub = btree_node_insert(node->lsub, item, comp);
    } else if (comp(item, node->item) > 0) {
        node->rsub = btree_node_insert(node->rsub, item, comp);
    }

    return node;
}
```

basecase

2

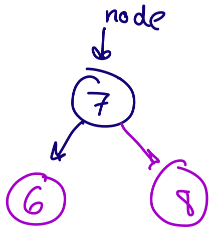
3

1 Tree empty

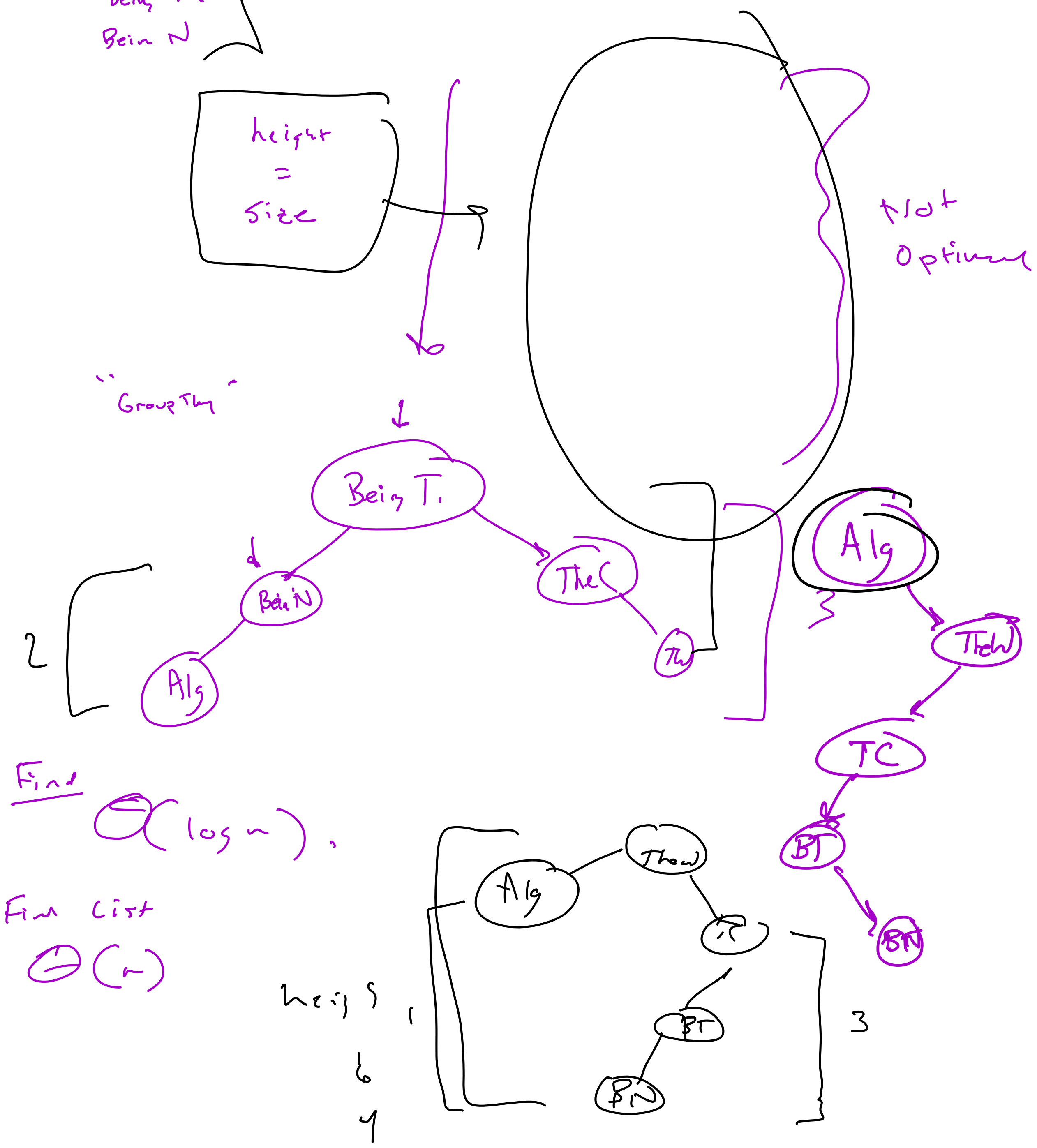
tree→root = new node

2 insert 6

3 Insert 8



Alg
The W
The C
Being Tim
Bein N

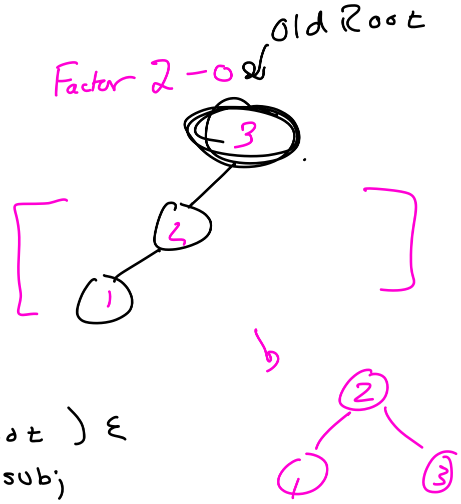


Balance it?

We write a balanced insert function.

We check if when inserting we cause the tree to become unbalanced.

```
factor = height(node→lsub) - height(node→rsub);

  if ( factor == 0 ){ // our tree

  } else if(factor > 0) {
        //rotater right

  } else {
        // rotate left

  }
```



```
btree_node * rsubRotate ( btree_node *old_root ) {
      btree_node *newRoot = oldRoot→lsub;
      btree_node *newGuyl = newRoot→rsub;
      new_Root → rsub = old_root
      Old_root → lsub = newGuyl;
      return newRoot;

}
```

// to do other rotations

// return the newRoot in insert.