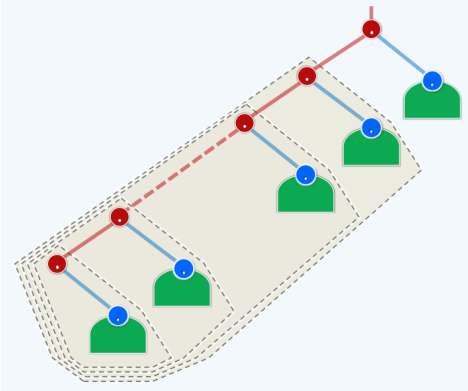
归并排序JS实现

```
function sort(n){
    if(n.length<2){return n}
    let mid=n.length>>1
    let A=sort(n.slice(0,mid))
    let B=sort(n.slice(mid))
    return merge([...A,...B],mid)
}
function merge(n,mid){
    let C=[]
    let A=n.slice(0,mid)
    let B=n.slice(mid)
    for(let i=0,j=0,k=0;(i<A.length)||(j<B.length);){
        if((i<A.length)&&(B.length<=j || A[i]<=B[j])){C[k++]=A[i++]}
        if((j<B.length)&&(A.length<=i || B[j]<A[i])){C[k++]=B[j++]}
    }
    return C
}
```

二叉树后序遍历的另一种迭代算法

将二叉树中的每一个树都看成是（左侧链+右子树）的结构，如下图所示：



因此对于每一颗树来说，其遍历过程如下：沿左侧链扫描将沿途节点按序入栈，然后反向对每一个左侧链上的节点进行以下操作：① 如果有右子树，对其右子树进行相同的扫描操作，然后将该右子树删除，说明该右子树已经被访问过了；① 如果没有右子树，则将其出栈并访问。

```
class BinTree{
    ......
    traverse(){
        let stack=new Stack()
        let x=this.rootBinNode // 根节点
        this.goAlongLeftBranch(x,stack)
        while(!stack.empty()){
            x=stack.top()
            if(x.rChild){
```

```
                this.goAlongLeftBranch(x.rChild,stack)
                x.rChild=null
            }else{
                console.log(stack.pop()) // 模拟访问操作
            }
        }
    }
    goAlongLeftBranch(bNode,stack){
        stack.push(bNode)
        let x=bNode.lChild
        while (x){
            stack.push(x)
            x=x.lChild
        }
    }
}
```