算法

1.树的遍历

- **树的前中后的非递归遍历**

```
Stack<TreeNode> nodeStack = new Stack<TreeNode>();
        nodeStack.add(root);
        TreeNode thisNode;

        /**如果栈不空
         * 1.如果有右孩子就压入栈
         * 2.如果有左孩子压入栈
         * 3.访问栈顶
         */
        while (!nodeStack.empty()){
             thisNode =  nodeStack.pop();
            if(thisNode.right!=null){
                nodeStack.add(thisNode.right);
            }
            if(thisNode.left!=null){
                nodeStack.add(thisNode.left);
            }
            ans.add(thisNode.val);
        }
        return ans;

 Stack<TreeNode> nodeStack = new Stack<>();
        nodeStack.add(root);
        TreeNode thisNode;

        /**
         * 如果栈非空
         *1.如果有左孩子，一直压栈
         * 2.访问栈顶元素，将右孩子压入栈
         * 重复1
         */
        while (!nodeStack.empty()){
            thisNode = nodeStack.pop();
            while (thisNode.left!=null){
                nodeStack.add(thisNode.left);
                thisNode = thisNode.left;
            }
            ans.add(thisNode.val);
            if(thisNode.right!=null){
                thisNode = thisNode.right;
                nodeStack.add(thisNode);
            }
        }
        return ans;

Stack<TreeNode> nodeStack = new Stack<>();
        nodeStack.add(root);
        TreeNode thisNode = root;
```

```
        TreeNode preNode = null;
        /**1.一直访问左孩子
         * 2.如果栈顶元素有右节点且没被访问过的则不访问，将右节点入栈
         */
        while (!nodeStack.empty()){
            while (thisNode.left!=null){
                thisNode = thisNode.left;
                nodeStack.add(thisNode);
            }
            thisNode = nodeStack.peek();
            if(thisNode.right!=null&& preNode!=thisNode.right){
                thisNode = thisNode.right;
                nodeStack.add(thisNode);
            }else{
                thisNode = nodeStack.pop();
                preNode = thisNode;
                ans.add(thisNode.val);
                thisNode.left = null;
            }
        }
        return ans;
```

2.排序算法

- 快排

```
public static void quickSort(int[] arr){
    quickSort(arr,0, arr.length-1);
}
private static void quickSort(int[] arr,int left,int right){
    if(left>=right){
        return;
    }
    int startIndex = left;
    int endIndex = right;
    int pviot = arr[left];
    int tmp = 0;
    while (left<right){
        while (arr[right]>=pviot&&right>left){
            right--;
        }
        while (arr[left]<=pviot&&left<right){
            left++;
        }
        if(left>=right) {
            break;
        }
        tmp = arr[left];
        arr[left] = arr[right];
        arr[right] = tmp;
    }
    tmp = arr[left];
    arr[left] = arr[startIndex];
    arr[startIndex] = tmp;
    quickSort(arr,startIndex,left-1);
    quickSort(arr, left+1, endIndex);
}
```

- **归并**

```
public static void mergerSort(int arr[]){
    mergerSort(arr,0,arr.length-1);
}
private  static  void mergerSort(int[] arr,int left,int right){
    if(left>=right) {
        return;
    }
    int mid = (left+right)/2;
    mergerSort(arr, left, mid);
    mergerSort(arr, mid+1,right);
    merger(arr,left,mid,right);
}
private static void merger(int[] arr,int left,int mid ,int right){
    int i = left;
    int m = mid+1;
    int k = 0;
    int[] tmp = new int[right-left+1];
    while (true){
        if(i>mid||m>right) {
            break;
        }
        if(arr[i]<arr[m]){
            tmp[k] = arr[i];
            i++;
            k++;
        }else{
            tmp[k] = arr[m];
            m++;
            k++;
        }
    }
    if(i<=mid){
        System.arraycopy(arr,i,tmp,k,mid-i+1);
    }
    if(m<=right){
        System.arraycopy(arr,m,tmp,k,right-m+1);
    }
    System.arraycopy(tmp,0,arr,left,tmp.length);
}
```