排序方式	时间复杂度			空间复杂度	稳定性	复杂性
	平均情况	最坏情况	最好情况			
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定	简单
希尔排序	$O(n^{1.3})$			$O(1)$	不稳定	较复杂
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定	简单
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定	较复杂
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定	简单
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定	较复杂
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定	较复杂
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(r)$	稳定	较复杂

关于寻找中位数：

```
关于寻找中位数：
/*
0,1,2
0,1,2,3
*/
n=num.size(); //n=3 or 4
mid=n/2; //1 or 2
mid_index=(n-1)/2; //1 or 1
//奇数个数，都是返回中间索引；
 //偶数个数，n/2 返回中间值的后一个，(n-1)/2中间值的前一个

 left=0,right=2;
 (left+right)/2; //1
 left=0,right=3;
 (left+right)/2; //1
 //二分维护的两个索引，(left+right)/2要么在中间（奇数个），要么在中间两个的前一个（偶数个）
```

二叉树遍历的非递归方法

前序遍历：对每个结点按照 根->右->左 的顺序入栈，出栈的顺序就是前序遍历的结果。

```
Stack<TreeNode*> s;
s.push(pRoot);
while(s.empty()==false){
 TreeNode *temp=s.top();
 visit(temp);
```

```
 s.pop();
 if(temp->left)
 s.push(temp->left);
 if(temp->right)
  s.push(temp->right);
 }
```

中序遍历：

（1）树先一直向左走到叶节点并将沿途的结点入栈；

（2）然后向右走一步，重复第一步操作。

```
Stack<TreeNode*> s;
s.push(pRoot);
TreeNode* p=pRoot->left;
while(p!=NULL && s.empty()==false){
 while(p!=NULL){
 s.push(p);
 p=p->left;
 }
 if(s.empty()==false){
  TreeNode *temp=s.top();
  visit(temp);
  s.push(temp->right);
  }
}
```