# 数据结构

* 时间复杂度和空间复杂度（详解版）

[时间复杂度和空间复杂度（详解版） (biancheng.net)](http://data.biancheng.net/view/272.html)

|  |
| --- |
| 如下列举了常用的几种时间复杂度，以及它们之间的大小关系：  O(1)常数阶 < O(logn)对数阶 < O(n)线性阶 < O(n2)平方阶 < O(n3)(立方阶) < O(2n) (指数阶) |
| 如果程序所占用的存储空间和输入值无关，则该程序的空间复杂度就为 O(1)；反之，如果有关，则需要进一步判断它们之间的关系：   * 如果随着输入值 n 的增大，程序申请的临时空间成线性增长，则程序的空间复杂度用 O(n) 表示; * 如果随着输入值 n 的增大，程序申请的临时空间成 n2 关系增长，则程序的空间复杂度用 O(n2) 表示； * 如果随着输入值 n 的增大，程序申请的临时空间成 n3 关系增长，则程序的空间复杂度用 O(n3) 表示； * 等等。   在多数场景中，一个好的算法往往更注重的是时间复杂度的比较，而空间复杂度只要在一个合理的范围内就可以。 |

* 栈、队列 互相模拟

[如何用栈来模拟队列，如何用队列模拟栈？？\_x.yao的博客-CSDN博客\_用栈模拟队列](https://blog.csdn.net/lilililililiki/article/details/104439469)

* 二叉树前序遍历

[二叉树多种遍历的时间复杂度和空间复杂度\_chen270的博客-CSDN博客\_二叉树遍历时间复杂度](https://blog.csdn.net/weixin_42513339/article/details/102705428)

|  |
| --- |
| 对于遍历方式只是打印顺序而已，所以四种遍历复杂度均相同。 [时间复杂度](https://so.csdn.net/so/search?q=%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_42513339/article/details/_blank)**：O(N)**[空间复杂度](https://so.csdn.net/so/search?q=%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_42513339/article/details/_blank)**：O(N)** |

* 二叉树前序遍历

[二叉树的后序遍历\_牛客题霸\_牛客网 (nowcoder.com)](https://www.nowcoder.com/practice/1291064f4d5d4bdeaefbf0dd47d78541?tpId=295&tags=&title=&difficulty=0&judgeStatus=0&rp=0&sourceUrl=/exam/oj?tab=%E7%AE%97%E6%B3%95%E7%AF%87&topicId=295)

[(2条消息) 详细介绍js实现二叉树非递归遍历算法(前序、中序、后序、层序、路径）\_coocochen的博客-CSDN博客](https://blog.csdn.net/qq_36368991/article/details/102652086)

|  |
| --- |
|  |
| 递归  function Preorder(root) {  let res = [];  const pre = (\_root) => {  if (\_root){  res.push(\_root.val)  pre(\_root.left)  pre(\_root.right)  }  }  pre(root)  return res } |
| 迭代  function PreOrder(root) {  let res = [];  if(root===null){  return [];  }  let tempStack = [root];  while (tempStack.length!==0){  //每次pop的都是一个左节点，然后下面再将右节点和左节点push进栈  //这样的话，每一次pop完之后，还没push进子节点时，stack栈里面都是右节点哦。  let node = tempStack.pop();  res.push(node.val);  //每次都把右节点先压进栈，之后把左节点叠上去  if(node.right){  tempStack.push(node.right)  }  if(node.left){  tempStack.push(node.left)  }  }  return res; } |

* 二叉树后序遍历

|  |
| --- |
| function postorder(root) {  let res = [];  const post = (\_root)=>{  if(\_root){  post(\_root.left)  post(\_root.right)  res.push(\_root.val)  }  }  return res; } |
| function postOrder(root) {  let res = [];  if (root === null){  return [];  }  let tempStack = [root];  while (tempStack.length > 0){  let node = tempStack.pop();  res.unshift(node.val); //unshift  if(node.left){  tempStack.push(node.left) //左右顺序跟前序相反  }  if(node.right){  tempStack.push(node.right)  }  }  return res; } |

* 二叉树中序遍历

|  |
| --- |
| function inorder(root) {  let res = [];  const ino = (\_root) => {  if (\_root){  ino(\_root.left);  res.push(\_root.val);  ino(\_root.right);  }  }  return res; } |
| function inOrder(root) {  let res = [];  if (root === null){  return [];  }  let tempStack = [];  while (root !== null || tempStack.length > 0){  while (root !== null){ //将根节点顺着左节点放进栈  tempStack.push(root);  root = root.left;  }    root = tempStack.pop();  res.push(root.val);   root = root.right; //换成右节点，之后再顺着这个右节点一直  }  return res; } |

* dfs 和bfs

[求二叉树的层序遍历\_牛客题霸\_牛客网 (nowcoder.com)](https://www.nowcoder.com/practice/04a5560e43e24e9db4595865dc9c63a3?tpId=295&tags=&title=&difficulty=0&judgeStatus=0&rp=0&sourceUrl=/exam/oj?tab=%E7%AE%97%E6%B3%95%E7%AF%87&topicId=295)

[二叉树的最大深度\_牛客题霸\_牛客网 (nowcoder.com)](https://www.nowcoder.com/practice/8a2b2bf6c19b4f23a9bdb9b233eefa73?tpId=295&tags=&title=&difficulty=0&judgeStatus=0&rp=0&sourceUrl=/exam/oj?tab=%E7%AE%97%E6%B3%95%E7%AF%87&topicId=295)

|  |
| --- |
| dfs  function maxDepth( root ) {  // write code here  const compute = (root) =>{  if(!root){return 0;}  let left = compute(root.left);  let right = compute(root.right);  return Math.max(left+1,right+1);  }    const dep = compute(root);  return dep;  } |
| bfs  function levelOrder( root ) {  // write code here  let res = [];  const pre = (root,level)=>{  if(root==null){return ;}  if(level>=res.length){  res.push([]);  }  res[level].push(root.val);  pre(root.left,level+1);  pre(root.right,level+1);  }  pre(root,0);  return res;  }  function levelOrderQue( root ){  let res = [];  let que = [root];  while(que.length!==0){  let tempRes = [];  let tempQue = [];  for(let i = 0; i < que.length; i++){  tempRes.push(que[i].val);  if(que[i].left){tempQue.push(que[i].left)}  if(que[i].right){tempQue.push(que[i].right)}  }  res.push(tempRes);  que = tempQue;  }  que = null;  return res;  } |

* 对称的二叉树

[剑指 Offer 28. 对称的二叉树 镜像，递归的思维非常简单！Javascript版 - 对称的二叉树 - 力扣（LeetCode） (leetcode-cn.com)](https://leetcode-cn.com/problems/dui-cheng-de-er-cha-shu-lcof/solution/jing-xiang-di-gui-de-si-wei-fei-chang-ji-gsrf/)

|  |
| --- |
| function mirrorBST(root) {  if (root === null){  return true;  }   const judge = (left,right) => {  if (!left && !right){  return true;  }  if (!left || !right){  return false;  }  return left.val === right.val && judge(left.left,right.right) && judge(left.right,right.left);  }   return judge(root.left,root.right); } |

* morris遍历

[【动画模拟】二叉树神级遍历！（Morris） - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/384818393)

|  |
| --- |
| Morris 遍历利用树的左右孩子为空（大量空闲指针），实现空间开销的极限缩减。 |

* 快排时间复杂度

[如何理解快速排序的时间复杂度是O(nlogn)\_sun123704的博客-CSDN博客\_快速排序时间复杂度为什么是nlogn](https://blog.csdn.net/u011947630/article/details/104691611)

|  |
| --- |
| ****快速排序的情况比较棘手，在最糟情况下，其运行时间为O(n2)。。在平均情况下，快速排序的运行时间为O(nlogn)。**** |
| [快速排序](https://so.csdn.net/so/search?q=%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u011947630/article/details/_blank)的性能高度依赖于你选择的基准值。   * 最糟情况 假设你总是将第一个元素用作基准值，且要处理的数组是有序的。由于快速排序算法不检查输入数组是否有序，因此它依然尝试对其进行排序。注意，数组并没有被分成两半，相反，其中一个子数组始终为空，这导致****调用栈非常长****。IMG_256 |
| IMG_256平均情况 假设你总是将中间的元素用作基准值，在这种情况下，调用栈如下。 调用栈短得多！因为你每次都将数组分成两半，所以不需要那么多递归调用。你很快就到达 了基线条件，因此调用栈短得多。 |

* 快排

[面试官：说说你对快速排序的理解？如何实现？应用场景？ · Issue #275 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/275)

|  |
| --- |
| var list = [0,1,15,9,4,2,26,8,3,12,20,6]  function quick( list )  {  const res = (arr) =>{  if(arr.length < 1){  return arr;  }  const left = [];  const right = [];  const mid = arr[0];  for (let i = 1; i < arr.length; i++){  if(arr[i] < mid){  left.push(arr[i]);  }  else{  right.push(arr[i]);  }  }  return [...res(left),mid,...res(right)];  }  return res(list);  }  quick(list); |

* 链表反转

[反转链表\_牛客题霸\_牛客网 (nowcoder.com)](https://www.nowcoder.com/practice/75e878df47f24fdc9dc3e400ec6058ca?tpId=295&tqId=23286&ru=/exam/oj&qru=/ta/format-top101/question-ranking&sourceUrl=/exam/oj?tab=%E7%AE%97%E6%B3%95%E7%AF%87&topicId=295)