面试小抄：

好代码应该关注的三个核心:

1. 可读性易读的
2. 时间复杂性
3. 空间复杂性

面试官面试的时候到底想要通过算法了解什么问题:

- 逻辑思维能力——你是怎么去思考问题和分析事情的?
- 编码能力与技巧——你能不能写出来好代码？什么是好代码？干净、简单、有条理、可读性高的代码就是好代码
- 基础知识储备——你是不是知道你用的算法或者数据结构，它的基本原理是什么?
- 沟通技巧——通过你对问题和解答的描述，以及期间的沟通，判断沟通能力是否符合公司的认定或者价值观。

面试的时候怎么去逐步的解答这个问题:

1. 当面试官提出问题以后，在顶部写下关键点(数组排序、二分、连通性啥的)。确保你看完这个题所有的细节。体现问题组织能力。
2. 确保自己仔细的检查了：「输入」是什么？「输出」是什么?
3. 这个问题最重要的地方在哪？然后就是时间、空间和内存等，哪个是主要目标?
4. 不要太烦人，不要问太多问题。问一些关键的问题就行
5. **从最简单，最直观的暴力法开始**。首先想到的，能够表明你能够快速和想到解决方案，并且可以批判性地思考（不需要写这段代码，只是说出来）
6. 告诉他们为什么这种方法不是最好的（即 O(n^2) 的复杂度比较高，不可读，有超时风险等...）

7. 仔细的想想解决方案，不断的再心里尝试，看看在哪里可以解决问题。 有没有重复的地方？像O(N^2)这样的瓶颈？或者不必要的逻辑？有没有用面试官给到的所有信息？瓶颈一般是代码中复杂度最高的部分。 有时候，重复的代码或者递归也会发生这种情况。
8. 在开始**写代码之前，再过一遍你的思路，并且写下来每一步要做什么**。
9. **从一开始就把自己的代码分成几个可读性高的小块，并在需要时添加注释。**
10. 写代码时候**准备和理解的代码越多，白板的效果就越好**。 所以**永远不要在不确定事情会如何发展的情况下开始白板面试**。那是灾难的根源。请记住：**很多面试都会提出您无法按时完全回答的问题**。 所以想一想：**我可以拿什么来证明我可以做到这一点，并且我比其他人更好**。

    > 在函数中分解事物（如果你不记得一个方法，就组成一个函数，你至少会在那里拥有它。 写点东西，从简单的部分开始。

11. 考虑错误检查以及如何破解此代码。 永远不要对输入做出假设。不用总是检查你不想要的错误输入。很棒的一个技巧：**在代码中写出你想要做的检查.....编写函数，然后告诉面试官你现在要编写测试以使你的函数失败（但你不需要实际编写测试）**。
12. 不要使用错误/混淆的名称 i 和 j。 编写可读性好的代码。
13. 测试你的代码：检查no params、0、undefined、null、超长数组、async code等...询问面试官我们是否可以对代码做出假设。 你能让答案返回错误吗？在你的解决方案中找出漏洞。 你得不断的去询问自己?
14. 最后告诉面试官，您将在哪里改进代码。 行得通吗？有不同的方法吗？可读性强吗？你会去查询什么来改进？如何提高性能？可能的话：**问下面试官您看到的这个问题最有趣的解决方案是什么?**
15. 如果你的面试官对解决方案感到满意，面试通常会到此结束。 **面试官问你扩展问题也很常见，例如如果整个输入太大而无法放入内存，或者输入是否以流的形式到达，你将如何处理问题。 这是谷歌常见的后续问题，他们非常关心规模。 答案通常是分而治之的方法——执行数据的分布式处理，只将输入的某些块从磁盘读取到内存中，将输出写回磁盘并稍后组合它们。**

有哪些是好的代码

- 有效的，可用的
- 熟练使用数据结构

- 注意代码复用，不要一直去写重复代码
- 模块化，让自己的代码更具有可读性、可维护性和可测试性
- 尽量去避免高复杂度的写法，比如循环嵌套
- 递归的话可能会导致堆栈溢出（比如归并排序时候，什么情况下是On什么情况下是Ologn）

有一些比较好的尝试

- HashMap通常是提高时间复杂度的答案
- 如果是做一个排序数组，尽量去使用分治，不断的去缩小数据范围，但是重复一个相同的操作
- 尝试对输入的数组进行排序
- 哈希表以及预先计算一些信息（比如排序、计算）是优化代码的一部分好方法
- 权衡时间和空间，有时候在空间中做一些存贮可以有效的提升时间
- 如果说面试的时候，面试官给你一些建议，你得去尝试跟着面试官的思路走
- 有时候在内存中存一些额外的状态，可以帮助提升效率，比如做枚举的时候

需要永远记住的，**尽可能多的去交流分享自己的思维过程，不要担心能不能快速完成这个问题。面试的每一个环境都很重要**

总结

当面试官提问完后，只需要先写下来关键点（之后再下面写注释和代码） 看完我的感受就是，**面试只要按照这个来做，成功率蹭蹭提升**

面试技巧

简单题

这里的题目难度比较小， 大多是模拟题，或者是很容易看出解法的题目，另外简单题目一般使用暴力法都是可以解决的。 这个时候只有看一下数据范围，思考下你的算法复杂度就行了。

当然也不排除很多 hard 题目也可以暴力模拟，一定要**多注意数据范围**

中等题

中等题目是力扣比例最大的部分，刷题时候不要太过追求难题，先把中等难度题目做熟了再说。

这部分的题目要不需要我们挖掘题目的内含信息， 将其抽象成简单题目。 要么是一些写起来比较麻烦的题目， 一些人编码能力不行就挂了。因此大家一定要自己做， 即使看了题解”会了“，也要自己码一遍。自己不亲自写一遍，里面的细节永远不知道。

中等题是最容易抽象方法论的题

困难题

困难难度题目合集

困难难度题目从类型上说多是：

- 图
- 设计题
- 游戏场景题目
- 中等题目的 follow up

从解法上来说，多是：

- 图算法
- 动态规划
- 二分法
- DFS & BFS
- 状态压缩
- 剪枝

从逻辑上说， 要么就是非常难想到，要么就是非常难写代码。 这里我总结了几个技巧：

1. 看题目的数据范围， 看能否暴力模拟
2. 暴力枚举所有可能的算法往上套，比如图的题目。
3. 总结和记忆解题模板，减少解题压力

注意事项

不要过于依赖自己的编辑器，面试时候的编辑器大概率不是你常用的，比如力扣、牛客、伯乐、codesandbox、codepen、腾讯云文档