**INDEX**

1. Interview Topics & Timelines - <https://www.facebook.com/groups/1548396065474189/permalink/1750243325289461/>

2. Resume Building Tips - <https://www.facebook.com/groups/1548396065474189/permalink/1749748762005584/>

3. 10 Tips on Online Coding - <https://www.facebook.com/groups/1548396065474189/permalink/1749722412008219/>

4. The art of Problem Solving - <https://www.facebook.com/groups/1548396065474189/permalink/1989131781400613/>

5. Online Coding Platforms - <https://www.facebook.com/groups/1548396065474189/permalink/1723934557920338/>

6. Online Material on "Data Structures & Algorithms" - <https://www.facebook.com/groups/1548396065474189/permalink/1723934687920325/>

7. Importance of Competitive Programming - <https://www.facebook.com/groups/1548396065474189/permalink/1750042841976176/>

8. Crossing the Coding Barrier - <https://www.facebook.com/groups/1548396065474189/permalink/1746305132349947/>

9. Code Debugging Tips - <https://www.facebook.com/groups/1548396065474189/permalink/1730719497241844/>

10. Online Material on "System Design" - <https://www.facebook.com/groups/1548396065474189/permalink/1730706643909796/>

11. Referral policy - <https://www.facebook.com/groups/1548396065474189/permalink/1723935121253615/>

12. Job Application Methods/Sources - <https://www.facebook.com/groups/1548396065474189/permalink/1731701887143605/>

This group is to share and discuss anything related to "Problem Solving using Data Structures & Algorithms" - Concepts, Online Problems, Tips & Tricks, Coding Techniques, Optimisations, Advanced Data Structures & Algorithms, Code Debugging, Interview Experiences among other things.  
Few important posts to check:

1. Interview Topics & Timelines - <https://www.facebook.com/groups/1548396065474189/permalink/1750243325289461/>
2. **Smart Interviews : "Interview Topics and Timelines"**

The topics covered and problems solved in Smart Interviews classes are bound to give you a head start in your interview preparation. But the amount of time spent in class is by no means sufficient. There should be consistent effort from every candidate to cater to their own capabilities and limitations.

The following steps will help you plan your preparation for interviews in the most efficient way. Make sure you go through the details of this post properly and adhere to it as much as possible.

1. Write down the list of categories that you need to work on for your interviews. Common categories include Data Structures/Algorithms (DS/Algo), Database system (DB), Operating Systems (OS), Networks, Object Oriented Design, System Design etc. Most interviews will comprise of 70-90% DS/Algo with limited focus on the other topics, depending on the company. Eg. Oracle asks for DB, Adobe asks for basic OS etc. No matter which company, it is always good to know the basics of these topics so that you are not taken off-guard in any interview.

2. Divide each category into sub-topics and estimate the preparation time for each of those individually. Sum those up to find the total time for each category. This preparation time is not definite but should be used as a strong guideline and will differ from student to student.

3. If any sub-topic is estimated for more than 5 hours, break it down further. No single component should be more than 5 hours.

4. Make sure you spend atleast 4-6 hours in preparing the above list. Find the total hours your preparation will take based on the above analysis.

5. Plan out your personal daily schedule, and analyse how much time you can dedicate per day/week on your preparation.

6. Based on the analysis from 4 and 5 above, find the total number of weeks needed for your preparation. If it is more than 12-16 weeks, you need to adjust your personal schedule to dedicate more time per week on your preparation. The longer your preparation period is, the more the probability of losing momentum. So, it is better to spend as much time as possible everyday so that the overall period is shorter.

7. Here is a detailed excel sheet to help you get started with your planning. Note that you should tweak and add/remove topics according to your background, understanding and the set of companies you are applying for.

[https://drive.google.com/open…](https://l.facebook.com/l.php?u=https%3A%2F%2Fdrive.google.com%2Fopen%3Fid%3D1FjiRpXUWKyfjTupNXdxLAtHL46sE444W5-kK0NcQ5CU&h=AT16PbAvFCw14sjrWAxbJnk4wMGHyNmiNFDtmUXlCJ5MQBKg-rUmjYkW0AWhU0dsTF6Uxh4s5QKGS3bxvZ2vJjE09r4Atvfm8wcTQg0U0-RmpadPbZAUVAUB4CepTR3vG3Tg7Oz9HHhQkY5mHMoY9R-E)

8. Last, but probably the most important one, make sure you maintain a cheatsheet of your learnings and important points/tricks/techniques you picked up during your preparation, so that you can revise them quickly before you go for any interviews.

1. **Resume Building Tips** - <https://www.facebook.com/groups/1548396065474189/permalink/1749748762005584/>

Smart Interviews : "Resume Building Tips"

1. Important things to include in your resume:  
-- Name, Email, Phone  
-- Work Experience [including internships, if any]  
-- Education [only the highlights]  
-- Projects [Include 1-liner explanation for your projects]  
-- Technical Skills  
-- Achievements

2. Formatting:  
-- Font: Times New Roman  
-- Font Size: 10-12

3. Resume Length: Ideally 1-page

4. Remove irrelevant information: [Few among many]  
-- Objective/Career Objective  
-- Summary/Personal Summary  
-- Personal Information: Address, Qualities/Traits, Mother/Father Name, DOB etc.  
-- Declaration [Place/Date etc]

5. Properly formatted, with no typos and not too colourful.

6. Strengthen your Resume:  
Your resume should contain few of the following things to be easily shortlisted for interviews:  
-- Open Source Contribution  
-- High CGPA/Grades  
-- Good projects  
-- Good company in your work profile  
-- Graduated from a reputed college  
-- Competitive Programmer  
-- ACM-ICPC Experience  
-- Regularly blogs on Technical Topics  
-- Personal Web Page  
-- Email Address at Own Domain

Note:  
[Only for the students of Smart Interviews]  
1. Please refer my resume "AmitBansal-Resume.pdf" in google drive below for reference.  
2. Please upload your resume in PDF format to [https://drive.google.com/open…](https://drive.google.com/open?id=0B_VPqpjxnC5cSE0welBRVlRudWs). These will be used to provide you with referrals, so please upload your final formatted resumes only.

Smart Interviews - Learn | Evolve | Excel

1. **10 Tips on Online Coding** - <https://www.facebook.com/groups/1548396065474189/permalink/1749722412008219/>

**3. "10 Tips on Online Coding"**

1. Problem Statements: Read the problem statements very very carefully, along with the input/output format, constraints and sample IO.

2. Constraints: It means that the input will be within the mentioned constraints. You need not write code to validate the input.

3. Output Format: Format the output of your code exactly as mentioned in the output format section, with no extra spaces and newlines. Refer to sample IO for better clarity.   
Some problems expect you to print the result % 1e9+7. This is because the result will be really huge (say N!) and cannot be stored in any data type. 1e9+7 is nothing but 10^9 + 7 - a big prime number. You can declare it using "int m = 1e9+7" in your code.

4. Data Types: Look at the constraints carefully and understand the right data types to be used for each of the variables in your code. Using wrong data types can truncate or overflow your results leading to wrong answers.

5. Time Complexity: Analyse the complexity of your code. Use the constraints to find the maximum value of T and M and compute the total runtime O(T\*M) for your algorithm, where M is the complexity for a single test case and T is the number of test cases. T\*M should be less than 1e8, if exceeded, there is a high chance of running into TLE. Do make sure your code doesn't go into infinite loops.

6. Space Complexity: You cannot declare huge memory in your code. In general, you should restrict it to 3-4 arrays of size 1e6 each, and not more. Read about memory declaration limits for your language of choice, both local and global declarations.

7. Coding Practices:

-- Write indented codes, use single TAB to increase/decrease indentations. Indentation increases the readability of your code.

-- Write modular codes. Modular codes are re-usable, has increased readability and are easy to debug. You can delay the implementation of the helper methods and focus on the core logic of your solution - A very important point for your interviews.

-- Declare iterators in the for loops itself; use braces in the same line to reduce code length. Ex: for(int i=0; i<n; i++) {

-- Iterate over test cases one by one, ie, read a single test case, process, print the desired output and repeat. Code Snippet:  
int T;  
Read T  
while(T--) {  
Read Input  
Process  
Produce Output  
}

-- Use the right variable names, like T/t/tc/testCases for testcases, n/m for array sizes, i/j/k/l for iterations, etc. It increases the readability of your code.

-- Write comments, whenever feasible, so that the next time you visit your code, its easier for you to understand the same.

8. Declaring arrays: Declare proper array sizes.

-- Read the array size N and then declare ar[N].

-- Understand the constraints for max array size and declare your array accordingly.

9. Print Statements: Don’t write irrelevant print statements. Ex: printf(“Enter number of test cases”); printf(“Enter the number”); etc. Of course, you can include print statements for debugging purpose and remove them before your submission.

10. Jump into Code: Once you have analysed all the above points for your algorithm and your code - time/space complexity analysis, using right data types, formatting your output correctly, THEN AND ONLY THEN jump into writing code on paper and then in a system.

Smart Interviews - Learn | Evolve | Excel

**4. The art of Problem Solving** - <https://www.facebook.com/groups/1548396065474189/permalink/1989131781400613/>

Smart Interviews : "The art of Problem Solving"

"Why does it has to be so hard?"   
- If this is the question that pops up every now and then to you, then here you go!

Interview panel ask hard questions because most of the problems in real life are challenging and difficult to solve. They look at your ability to approach new and difficult problems and come up with a solution or multiple solutions. The first step to approach a difficult problem is to accept the fact that it can be solved and to understand its significance. Don’t give up on them easily. Practicing handful of few hard problems will make you learn much more than a lot of easy problems.

Here are a few strategies for dealing with hard problems, and the frustration that comes with them:

\*Do something\*  
- We understand that the problem is difficult and it will require lot of efforts. But stop staring and start working on them. If nothing clicks, start with brute force solution.   
You should have always at least one working solution to any problem.

\*Reflect on successes\*  
- Think of all tricks and techniques used previously to solve any problem of similar type. You may find some constructive solution out of it and reflect back on already used DS/Algo tricks and techniques. At times, you may surprise yourself with a way better solution.

\*Work backwards\*  
- Try figuring out what is expected out of the problem instead of what the problem expects from you and there you go. Apply reverse engineering. Make assumptions, note down observation, take different examples and figure out if the output is valid for all scenarios as per your assumptions.

\*Ask for help\*  
- A person who asks a question is a fool for a minute; person who does not ask a question remains a fool forever.   
Don’t hesitate, ask foolish questions and lame doubts. You will get stuck. You will need help. And if you ask for it, you’ll get much farther than if you don't.

\*Start early\*  
- Last moment preparation are never helpful. Hard problems require complex idea and takes more time to understand fully. This is the first step for building foundation so better start early and give more time to assignments to accomplish it within the given time frames.

\*Take a break\*  
- If you are stuck at a problem struggling and burnt out, you don’t need more productivity. You need a break.   
You may find that you haven't entirely gotten away from the problem, but the background processes of your brain have continued looking for solutions, without you actually knowing about it. Wehen you revisit the problem after few days, you'll find yourself a lot closer to the solution.

\*Start over\*  
- Failure is the opportunity to start over again. If nothing strikes, question your assumptions. Take a fresh paper and start thinking from scratch.

\*Give up\*  
- Some problems are too hard to solve. It's okay to give up, but this applies if and only if you have completed about 70-80% of the tasks. A single difficult problem is usually going to teach you more in the first hour or two, than it will in the next six, and there are a lot more problems to learn from. So, set yourself a time limit, and if you’re still hopelessly stuck at the end of it, then read the solutions and move on.

\*Be introspective\*  
- Read solutions, take clues and build on the hints. Read actively, not passively. As you read it, think about what clues in the problem could have led you to this solution. Never take entire logic and code from somewhere or someone. If the solution involves math facts that you don't understand, then go investigate and read more about it. Learning in this way will serve you well later.

\*Come back\*  
- If you give up on certain problems, keep them alive and maintain a list of doubts and unsolved problems. Come back and try them again in a few weeks. Whenever you come across a new strategy or technique, quickly run it through your pending problems and see if can use it to solve atleast one of them.

**5. Online Coding Platforms** - <https://www.facebook.com/groups/1548396065474189/permalink/1723934557920338/>

Online Coding Platforms:  
1. Smart Interviews - [https://www.hackerrank.com/con…/smart-interviews/challenges/](https://www.hackerrank.com/contests/smart-interviews/challenges/)  
2. Topcoder - [https://www.topcoder.com/community/events/](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.topcoder.com%2Fcommunity%2Fevents%2F&h=AT0nw_7Leebw5ODT1fSvCrHTlMSxsh2nvrTa6EUfQUkehBCzFh-qBFw9bP7icPxzsFGI7kB2xQsMW3qd8UgP6gAsl9ytlLrMwgKqpZ-m0FUyWTaDebSdlSM2XXxOdc7fh61vm1C9ceH-QbboQM5Vs_gx) [SRMs]  
3. Codechef - [https://www.codechef.com/contests/](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.codechef.com%2Fcontests%2F&h=AT2DepHBMJWS40DjVqKNFcnpaQ6xxkdE9FEDiB7wajRxw-kSRdT-eB7hhcyw56jzP2Qf7Sip3R-G8o1SEdJcOUpCZtME2f2VgST9TI0jIvlnj6pZiKP6unlVB55cvY-y1_MhHy1q9P36fUo024EHyLPG)  
4. Codeforces - [http://codeforces.com/contests/](https://l.facebook.com/l.php?u=http%3A%2F%2Fcodeforces.com%2Fcontests%2F&h=AT30k4fI6LbeVbw8UrpeAKyVHudEgVzLfVl-MaDQMPC16PtjJw01SOc6VoGWEvWBUWObWZf-obcb6RDuYPW6-7AVU0aBYV5EPNdD1NZoDV_nj0XeTj-wB1Rxz0Nu_qbuL-tmDgmctwwnFKLzhtCIljCJ)  
5. Spoj - [http://www.spoj.com/problems/classical/](https://l.facebook.com/l.php?u=http%3A%2F%2Fwww.spoj.com%2Fproblems%2Fclassical%2F&h=AT1JjUvI0avK_Pkw3A6dKf4USShfKmob9fQcgzfo2wx7SVbzOWy_fOia5b_Q308KP1X_ORjxU7dXTFWE3vLOQ5D7Eff04Fcxyte43Xo6JoXk3C9KRG1y-khiLuREACgk_Y1IVyB-KXJYyEjHPwAKtcro)  
6. Interviewbit - [https://www.interviewbit.com/courses/programming/](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.interviewbit.com%2Fcourses%2Fprogramming%2F&h=AT1jr7XnemrsaycSViUtdOrCaFdYT-c2TO2PhQWIUbD6cbmmfFPIa8W94sSUyTqTivJqIDbPCNKVuV94fgS0ufus5MYEbPjDIlVmfy6D4eJttAnSFJzW3Ayk1CBEA0_LQWRlGLQVLIpqvYo_9UwAG5rz)

Pick any 2 platforms of your choice and try achieving the following targets:  
1. Smart Interviews: Solve 80% of the problems  
2. Topcoder: Blue Coder [>=1200 points]  
3. Codeforces: Candidate Master [>=1900 points]  
4. Codechef: Ratings - Long(>=4000), Short/Ltime(>=1500)  
5. Spoj: World Rank <= 3000  
6. Interviewbit - Rank <= 5000

There are many other platforms to practice on: Hackerrank, UVA, Hackerearth, ProjectEuler, etc.

**6. Online Material on "Data Structures & Algorithms"** - <https://www.facebook.com/groups/1548396065474189/permalink/1723934687920325/>

Online Material on "Problem Solving using Data Structures & Algorithms": [Few among many]  
1. [https://www.topcoder.com/…/data-sci…/data-science-tutorials/](https://www.topcoder.com/community/data-science/data-science-tutorials/)  
2. <http://web.stanford.edu/class/cs97si/>  
3. <https://people.cs.clemson.edu/~bcdean/dp_practice/>  
4. [http://www.geeksforgeeks.org](http://www.geeksforgeeks.org/)  
5. Introduction to Algorithms - CLRS [Book]  
6. Data Structures and Algorithms Made Easy by Narasimha Karumanchi [Book]  
7. <http://codeforces.com/blog/entry/23054>

Problem Classifiers: [Few among many]  
1. [https://a2oj.com/categories](https://l.facebook.com/l.php?u=https%3A%2F%2Fa2oj.com%2Fcategories&h=AT1JmnbK00mopMpmnjl0hX7uBAalr9iYZ6MxklVs937Opoxv5wXTD63LwkSEWWYocDIGjUbX8gJruietWeqDcNWEDfBPbZK-FogfrLNOOKbPw2Q9-LTiavoY6WtG7dLR8836FYCmlfM00S_ygyMaei8d)  
2. [http://problemclassifier.appspot.com](https://problemclassifier.appspot.com/)  
3. <https://www.codechef.com/tags/problems>

**7. Importance of Competitive Programming** - <https://www.facebook.com/groups/1548396065474189/permalink/1750042841976176/>

Smart Interviews : "Importance of Competitive Programming and how to get started"

I had never done competitive programming before December 2016. I vaguely remember trying the TEST problem on SPOJ about 4 years ago back in college. As someone who did not know how to take continuous stream of input back then, that problem was enough to make me stay away from SPOJ for the years to come. It did not affect my coding skills or campus placement in anyway.

Or so I thought, until 3 months ago.

Before we jump into things, let me make this very clear: Competitive programming is not an absolute requisite for clearing interviews. There are many people who have never done competitive programming in their life but manage to clear the interviews of top-notch product-based companies. Nevertheless, I suggest that anyone who is looking to code for a living should try competitive programming at least once, for a period of time consistently. Why?

1. Competitive programming hones your skills on sorting, searching, hashing, number theory, graphs and DP: that's basically your entire DS/Algo excluding trees and linked lists. The more practice you have with competitive programming, the more comfortable you are with problem solving using DS/Algo.

2. Programming contests are time bound, the exact same way your interviews are. In programming contest, you'll mostly have 6 questions to be solved in 3 hours: in interviews you'll have maximum 2 questions for every one hour of interview round. Also, programming contests generally have their questions arranged from easy to difficult, the same way interviews proceed.

3. Some programming contest portals like Codeforces and TopCoder give you the option to look into others' code during the contest to find errors. This gives you a higher motivation to improve your own debugging skills as well as pick up good coding practices.

4. Most importantly, having a good rank on a coding portal or achieving a good position in ACM-ICPC gives your resume a definite edge. Companies like Facebook and Google are known for going directly to ICPC World finals to recruit (Fact courtesy: Amit Bansal) If you are the kind of person whose resume does not get past the initial screening because you do not have a good company in your work profile or good projects/GPA/college in your academic profile, competitive programming is a sure-shot way of getting past this barrier.

You can find a good coder who is not a competitive programmer. But you will not find a good competitive programmer who is not a good coder.

GENERAL TIPS ON GETTING STARTED:

-- Non-live contest practice:

1. Start with basic/easy problems with highest submissions/acceptance rate. (Most portals will have options to filter/sort based on this) This will keep you motivated and help remove the mental block around it.

2. Exclusively for Smart Interviews: Solve problems based on topics that are covered that week/covered so far. Every portal has the option to filter problems by topics/tags. This will help you get a grasp of the concepts much faster. There is no such thing as too much practice.

3. Learn to use the discussion board that accompanies the problem to get you moving when stuck.

4. Do not try to keep track of too many portals. Try all of them in the beginning (maybe), see which is to your liking and choose a maximum of two among them to keep consistent ranking.

--Live contest:

1. Be on time for the contest. Be serious about solving the problems.

2. Start from the first question and proceed in ascending order. There will almost NEVER be a time when 6th question is easier than the 2nd.

3. Read the editorial and solution after the contest. This is what will contribute to most of your learning.

4. Try to utilize the features on Topcoder and Codeforces to read others' code. I find this to be one of the most enlightening experiences personally.

Compiled by Sivaranjani Nadar, Class of November 2016 - January 2017

**8. Crossing the Coding Barrier** - <https://www.facebook.com/groups/1548396065474189/permalink/1746305132349947/>

[**A Mohammed Junaid**](https://www.facebook.com/amdjunaid?fref=gc&dti=1548396065474189) Situation: There's a problem hosted on Smart Interviews - Hackerrank," Smaller Elements". I tried so hard to solve it and I was just not able to solve it even after like spending hours on it. I felt irritated and sad.   
  
Action:   
I messaged Amit and he asked me to calm down and he told me that's it completely fine and to leave the problem aside for sometime and to comeback to it later with a fresh mindset. So, I did as per instructed.  
  
Response:   
It was about 6+ hours since I left the problem and couldn't come back any quicker as I was working in office. While heading back home, that very night, I tried solving it again and all of a sudden, an intuitive approach struck my mind. I was so pretty happy about it. It did help me and I find it works out pretty well! Give a try! https://static.xx.fbcdn.net/images/emoji.php/v9/f55/1.5/16/1f44d.png👍   
  
Logic behind the idea :  
when we leave the brainstorming problems aside for a while and get involved in something else, our brains get into Diffused mode and all the neural connections related to it are formed. Also, our minds unconsciously work at the background level to find patterns / logic etc. So, when you revisit the same problem again, you're usually steps closer to it than the previous state.

[**Sivaranjani Sathya-Subramanian**](https://www.facebook.com/sranjani.s?fref=gc&dti=1548396065474189) The first thing you need to understand is, not being able to solve a problem is a situation that can happen to both beginners and seasoned coders. Sure, solving a lot of problems give you the practice required to reduce the probability of getting into this situation. But depending on the problem even an experienced coder sits with a problem for days together trying to solve it (trust me on this one) So, it's alright that you did not get that problem right the first time around or the 100th time around. Do not give up or feel disappointed just yet. Remember that at the end of the day, it is the learning that matters.  
  
Some of the reasons why you'd get frustrated when you don't solve the problem:  
  
-- You think you'd spent too much time on a single problem: It's perfectly alright unless it is some time bound scenario like an interview. Even then it is alright, because it'll be a learning curve that'll catapult you to a better next interview.  
  
-- You think you're behind people in class: Please do not compare yourself with someone else. Compare what you know today with what you knew yesterday. That graph should always be increasing https://static.xx.fbcdn.net/images/emoji.php/v9/fa5/1.5/16/1f642.png:)   
  
-- You feel stupid: It is a good sign. You'll have the drive to not feel this way: you will absolutely cherish the moment you see all the green ticks on solution submission, as a result of your long hard work. Every point on the Leader board will be hard-earned and highly deserved.  
  
-- You think you'll never make it: This is just a mental block (Again,trust me on this) There's nothing in the world you can't learn when you're ready to work for it. When so many people, batch after batch, in Smart Interviews can do it, you can do it too. My batch had people from all background: people who are professional coders to people who did not study CS back in college. But they were equally competent only because they never told themselves that they can't do it. In short, stop feeling sorry for yourself and work.  
  
-- Unsolved problems keep you awake at night: Again a good sign. But try to calm down a bit because it is not particularly healthy both physically and mentally. Monitored breaks and change of perspectives often help.  
  
Once you understand why you are getting frustrated, understand that the frustration is actually keeping you from solving your problem at hand. So, take away this unnecessary emotional burden and focus on methodically getting to your solution.  
  
Some suggestions on what you can do that'll actually help you to solve your problem:  
  
-- Write your logic down using pen/paper before you type it on the screen: very often than not, the major source of bugs is lack of clarity of thought. It is very difficult to follow your thought process and write your code simultaneously. So, write down your logic/pseudo-code first before you jump into coding.  
  
-- Get your syntax right: The most annoying form of error. From using the wrong placeholder in C to missing a \n to confusing = and == to wrong datatype, these tiny errors can cost you your time. Make sure you read your constraints and expected output and program accordingly.  
  
-- Explain your code out loud to a person (or a toy: the "Rubber ducky" method): When you say out loud what you are doing, half the time, you'll realise where you're going wrong in the middle of your sentence. This is one of the major reasons why during interviews, you're expected to "think out loud".  
  
-- Check the edge cases: Second most common source of error, not handling edge cases. Anytime you use an array, think about scenarios when you might access out of bound index. Anytime you use a pointer, think about situations when it might point to NULL and what you should do about it. Always check if your code is working for the extremes of your constraints.  
  
-- Look into the debugging tips posted in this group: Having print statements/break points at multiple places will help you see if the program is executing in the way you wanted it to. Comparing your "efficient" algorithm output with brute force output will help you check the validity of your logic.Read through your code and see if you have translated your logic correctly into code.  
  
-- Take a break: and come back to solve the problem. Change of mood and environment for a brief period can help you take a fresh look at the problem. But beware of this "break" prolonging for days.  
  
-- And if nothing works, bug [Amit Bansal](https://www.facebook.com/amit.bansal.1276?fref=gc&dti=1548396065474189&hc_location=ufi) https://static.xx.fbcdn.net/images/emoji.php/v9/ff8/1.5/16/1f61b.png:P He'll only nudge you in the right direction. You need to pick yourself up again, decode the clues and work for your solution: the EXACT same way it happens in interviews.  
  
Remember that it is consistent hard work that'll take you anywhere. You cannot expect results if you give up before you reach the finishing line. The only difference between winners and the rest is that winners never quit.  
  
Hope that helps https://static.xx.fbcdn.net/images/emoji.php/v9/fa5/1.5/16/1f642.png:)

**9. Code Debugging Tips** - <https://www.facebook.com/groups/1548396065474189/permalink/1730719497241844/>

Smart Interviews : "Code Debugging Tips"  
1. Dry-run[[https://en.wikipedia.org/wiki/Dry\_run\_(testing)](https://en.wikipedia.org/wiki/Dry_run_(testing)" \t "_blank)]: Take an example and execute the code "mentally" or on "paper" against it, there is a good chance of finding the bugs quickly.  
2. Question the assumptions: Don't assume that a piece of code is working fine. Dry-run against few sample examples to verify whether it's actually working fine or not.  
3. Print Statements: Insert print statements to inspect the data values.  
4. Debugger: Step through the code execution line by line and inspect everything starting with the value of variables.  
[gdb for c++, pdb for python etc]

More importantly, we should avoid introducing bugs in our code. Few Tips:  
1. Plan Your Code: Sketch the code outline on paper before writing it on machine.  
2. Copy-Paste: Minimize potential problems by avoiding Copy-Paste syndrome.  
3. Modularize the code: Poorly architected code is a major source of hard-to-fix bugs. Modularize and write helper functions to assist the main code.

**10. Online Material on "System Design"** - <https://www.facebook.com/groups/1548396065474189/permalink/1730706643909796/>

Online Material on "System Design":   
[Few among many]

1. <https://www.youtube.com/watch?v=-W9F__D3oY4>  
2. [http://www.lecloud.net/…/scalability-for-dummies-part-1-clo…](http://www.lecloud.net/post/7295452622/scalability-for-dummies-part-1-clones)  
3. <https://www.hiredintech.com/system-design>  
4. [http://highscalability.com](http://highscalability.com/)  
5. [https://www.quora.com/How-do-I-prepare-to-answer-design-que…](https://www.quora.com/How-do-I-prepare-to-answer-design-questions-in-a-technical-interview)

**11. Referral policy -**

<https://www.facebook.com/groups/1548396065474189/permalink/1723935121253615/>

1) If you need referrals in over 20+ Tier-1[10-25LPA]\* companies including Google, Facebook, Amazon, Microsoft, Adobe, Walmart among others, please meet the following requirements:

A. [MUST] Solve 80% or more problems on <https://www.hackerrank.com/contests/smart-interviews/> .  
B. [OPTIONAL] Provide links to the rankings of atleast 5 coding contests[in total] from below:

-- Upload 3 or more problems in Codeforces Regular Contests.  
-- Upload 4 or more problems in Codechef Long contests.  
-- Upload 2 or more problems in Topcoder SRM Contests.

2) If you need referrals in over 50+ Tier-2[5-15LPA]\* companies including Oracle, JP Morgan, Cisco, Groupon, Practo, Nvidia among others, please try to meet one of the following requirements:

A. [MUST] Solve 60% or more problems on <https://www.hackerrank.com/contests/smart-interviews/> .  
B. [OPTIONAL] Provide links to the rankings of atleast 5 coding contests[in total] from below:

-- Upload 2 or more problems in Codeforces Regular Contests.  
-- Upload 3 or more problems in Codechef Long contests.  
-- Upload 1 or more problems in Topcoder SRM Contests.

\* The package varies depending on relevant openings, candidate's background/experience/skillset among other factors.

Note:  
1. Please refer my resume "AmitBansal.pdf" in google drive below for reference.  
2. Please upload your resume [<YourFullName.pdf> format only] to [https://drive.google.com/open…](https://l.facebook.com/l.php?u=https%3A%2F%2Fdrive.google.com%2Fopen%3Fid%3D0B_VPqpjxnC5cSE0welBRVlRudWs&h=AT1GMWeDuE6IdMaR67gmfIoRDJ5Bn19YhN5BWLuGFxJRrk2P8w5xGSCIgYIXjRZLI7hxQ4erOR6LhQxNxS207a-vyhZHKKxjXsrOzJGEY2PErYEHMUUFTihD6HnmrIyKkF3_HeLVOoKrzkbLN5wbTp3I). These will be used to provide you with referrals, so please upload your final formatted resumes only.

**12. Job Application Methods/Sources** - <https://www.facebook.com/groups/1548396065474189/permalink/1731701887143605/>

Smart Interviews : "Job Application Methods/Sources"

The simplest and basic principle is to apply in around 50 companies through "Referrals", you will get call from around 10 of them, out of which you will clear 2-3 interviews.  
Finally, decide which one you want to join. Use the following sources for application [Change the search criteria as per your needs]:

[Few among Many]  
1. <https://www.indeed.co.in/jobs?q=Developer&l=>  
2. <http://www.jobsaggregator.com/US/+/Developer#jobresults>  
3. <http://www.careerjet.com/search/jobs?s=developer&l=>  
4. <http://www.simplyhired.com/search?q=developer&l=>  
5. [https://www.instahyre.com](https://www.instahyre.com/)  
6. Monster, Naukri, Shine  
7. <https://www.interviewbit.com/referrals>  
8. Career page of the company you are applying for.  
9. Lastly, the most important one, find people on linkedin and facebook, and request them for referrals.

Top of Form