Internship Preparation Guide

- By CodeISM

(**Youtube link for explanation**: https://youtu.be/TCA_4xCEZR4)

The intern season (coding rounds) would take place at the start of August. At this time you should prepare company oriented.

It might be possible that you can't cover all points, try to cover as many as possible. (specially imp/bold one**)*

Preparing for coding problems

0. **Learn any programming language** (C++ / JAVA / Python preferred) and the fundamentals of **time and space complexity**. You must be able to analyze both the space and time complexity of a given code. You can read chapter VI - "*Big O*" from the book "*Cracking the coding interview*". [about 15 pages, but contain a lot of different examples on space and time complexity to have a crystal clear understanding of Big O]

1. Don't miss any **short contests on Codeforces, Atcoder,** and Codechef, as they will certainly help you handle pressure during coding rounds. Also, try to give the **weekly and biweekly contests on leetcode**. [These leetcode contests would also cover topics like binary tree, linked lists, etc. which are never asked on other platforms].

2. Solve the 450 problems in this love babbar's **DSA cracker sheet**: https://drive.google.com/file/d/1FMdN_OCfOI0iAeDlqswCiC2DZzD4nPsb/view

   Do keep these things in mind while solving these problems:

   a. **For theory of the topics, you may also refer to articles from geeksforgeeks (GFG) or some youtube videos.** (Some great youtube channels for DSA are Abdul Bari , Coding Made Simple, Code with Harry)
   b. When solving a problem, **give your best thought to a problem for sufficient time and code what you thought of**, only after this, look at the hints/solution, because directly looking at the solution will hamper your thinking capability.
   c. **Even if you solved the problem, do look at the provided solution as it would give you a better insight.** Till now in competitive coding you might not have to improve space complexity but now start practicing to get both the best space and the best time complexity.
      [Sometimes, your solution may use some extra memory and the interviewer may ask you: "Solve this problem without using any extra memory?".

It may even happen that you solved the problem in O(N) time complexity and the solution also has the same time complexity but your solution performs 2 traversals of the array while the provided solution uses only 1 traversal. Still, read the provided solution once. Sometimes, the interviewer may ask such follow-up questions: “Can you solve this question in only 1 traversal of the array?”.]

d. After solving each problem, ask yourself “Why was I not able to get through the correct approach, at first?“ **Maintain a notebook and take time to note down the things you got stuck on and what helped to figure them out.** Every day, before you start practicing, you can go through those short notes once.
e. **Bookmark the problems for which you think you were not able to solve in 1 attempt** or it took too long to arrive at the correct solution. You can go through all your bookmark problems, once every 15-20 days.

3. After solving the above problems, solve all the problems in the **programming section of interviewbit**:

https://www.interviewbit.com/courses/programming/

[It will help you revise all the important topics. You can also swap the order for 2 and 3, if you have less time left for preparation, but make sure you follow the above instructions while solving any problem.]

4. Topics to be covered in “Data Structures” from GFG:

a. **Arrays**
b. **Linked List**
c. Stacks
d. Queue (Including Priority Queue and Deque)
e. **Binary Tree**
f. **Binary Search Tree**
g. Heap
h. Hashing (Mainly, know how to use hash and the theory)
i. **Graphs** (Only standard algorithms and their applications)
j. Matrix
k. Strings
l. Segment Trees (companies like Codenation, Flipkart, Salesforce may ask)
m. **Trie** (V.Imp. for Arista)
n. N-ary Tree (V.Imp. for Arista)
o. **Disjoint Set Union (DSU) or Union Find**

Also, see **LRU Cache implementation**. It is a very common interview question.

Topics to be covered in Algorithms from GFG:
i. **Searching Algorithms**
ii. **Sorting Algorithms**
iii. Greedy
iv. **DP** (V.Imp.)
v. Pattern Matching (**KMP** and Rabin Karp, specifically)
vi. Backtracking (V.Imp. for Samsung)
vii. **2-pointer** (V.Imp., you may check out the short course on codeforces EDU section too)
viii. Bitwise Algorithms
ix. Divide and conquer (helpful in interviews)

Tips for a good resume

1. **Don't take more than one page, mention dates, and links wherever possible** (shows how genuine you are). For this, first, jot down all the points (even the smallest ones) that you think might be beneficial to you and then decide which one you should keep in the resume.

2. It is very important to have **at least 2 good projects**, mentioned in your resume. And a good project means you have complete background knowledge both application and theoretical and you can speak for around 4-5 minutes on each. And be ready for some questions from the interviewer on your project like, “what problems did you face and how did you overcome those?”, “How can you scale it?”, “How can you further optimize it / improve the accuracy of your model?”, etc.

3. Try to use **short bullets instead of paragraphs**, as resume screeners hardly spend 30-45 seconds on a resume, keeping bullet points of 1-2 lines, have more chances of it being read, rather than that of a big paragraph.

[Note that only a very few companies like Flipkart don’t care about resumes. But almost all other companies do. And for off-campus opportunities, generally, “resume-shortlisting” is the first round, after which only you would get the test link for coding round.]

Questions other than coding problems

1. **OOPS**: Go through the concepts of C++, or any other OOP language that you had written deeply. For C++ OOPS, you can refer "*Balagurusamy "* or can go

through Saurabh Shukla Videos (in Hindi) or this short course by *freecodeacademy* (in English) on Youtube. **Always have a real-life example ready for each concept of OOP.** For revision purposes, you can refer to short notes from balagurusamy or short notes from GFG.
For interviews, apart from direct questions, you may also be asked problems like "Design Snake and ladder game using OOP", "Design a car parking lot using object-oriented principles" etc. to judge your object-oriented skills.

2. **Complete the C quizzes and C++ quiz from GFG**, as some companies ask language MCQs, debugging questions, etc. in the coding test. [And keep on revising the concepts learned from these]
3. You may wish to give the Data structures quiz and Algorithms quiz from GFG, as some companies (GS, Walmart, etc.) also ask MCQs on these.
4. Probability (for GS, Amazon): Solve the book "*Fifty challenging problems in probability*". It will cover almost every concept required for MCQs and interviews (any time you might see the same question as in the book during your MCQs and interview)
5. Puzzles (for GS, Salesforce): Go through the puzzles on GFG or puzzles on interviewbit
[Generally, you would be asked the same puzzles given here]
6. **System Design:** You may go through the system design course by educative.io or system design section on interviewbit . These have 6-12 similar problems. Once you will read all of them you would have sufficient knowledge an interviewer expects from you during your start of the pre-final year.
[You may also view Tushar Roy's system design playlist or read the *System design* portion of "*Cracking the coding interview*" for understanding some of the concepts better, but that it is completely optional]
7. Operating Systems: You may go through the Operating Systems playlist by Gate Smashers and can read some common OS interview questions from GFG.
8. DBMS / SQL (Only if time permits): You may go through the SQL tutorial by freecodecamp (covers all fundamentals of DBMS and SQL,) and Database Design Course by freecodecamp (covers all DBMS concepts in deep, more theoretical course). If you have enough time, you can read some common DBMS interview questions and also try some SQL problems on Leetcode.
[You can skip DBMS if time is not sufficient.]

9. Computer Networking (Only if time permits): Networking questions are rarely asked for internship interviews, but if you have sufficient time, you can cover some important topics as explained in this roadmap video by Love Babbar.

Some tips for coding tests

1. In the coding round do read the instructions very carefully, as some MCQs may have negative markings too.
2. **Keep collecting the questions from other college friends** as some companies will visit their colleges prior to yours. These companies have a high possibility of asking the same question.
3. **Just before a coding round do check out GFG for interview experience**, even there you might get some idea of the problems asked by the company. [We have also shared some interview experiences of our college, in the telegram group.]
4. **Try to wrap up your preparations around 4-5 days before your first internship test.** Set an appropriate deadline for yourself. This will help you avoid panic and give you time to practice according to the company.

How to practice for interviews?

[If you don't know what a coding interview is like, watch this short interview video by Google. We also recommend going through the interview prep crash course by Uber]

1. If you are not good at **English Conversation**(at least you can speak so that person in front of you can understand), do practice as it might leave an impact on the interviewer.
2. **Try to write neat and clean code/ pseudo-codes/ approaches using pen and paper** [In offline interviews, you won't have your system to press backspace].
3. During the last month of your preparation, try to give as many **mock interviews** as possible among your friends. [You may also use websites like https://www.pramp.com/ or https://interviewing.io/ for this purpose]

Interview Specific Tips

Trivial Interview Questions

You should remember at least one:

- of the problems that you faced while working on your project, along with how did you overcome them?
- example of an interesting technical problem you solved
- example of a conflict you overcame, while working in a team
- example of leadership

To get an idea of, how to structure your answers for such type of questions, do read chapter V - “Behavioural Questions” from the book *“Cracking the coding interview”* [It is not more than 6 pages, but is really very helpful because a few companies like Walmart

don’t ask any technical question in the interview and how you answer such questions is only important]

Keep your questions ready

In most interviews, in the end, the interviewer would ask you if you have a question for him.
You should have at least one question about:

- Any of the company's product/business
- Something related to the company's engineering strategy (like how they perform testing, deployment, etc.)

For this, take some time before the interview to go through the company’s website once.

Understand the problem clearly

When the interviewer is explaining a problem statement, listen very carefully. Each and every tiny detail of the problem would prove useful in arriving at the most optimal solution. Take a moment to repeat the question back at the interviewer and make sure that you understand exactly what they are asking.

Do ask clarifying queries to get the complete meaning. One should ask about the unmentioned cases, like “Can elements be negative?”, “Are repeated values allowed?”, “What will be output for — an edge case?”, etc.

Think out loud

After understanding the problem, don’t jump directly to code. Explaining your thought process to the interviewer is more valuable than writing the correct code. While you are building up a solution, don’t be quiet for too long. Try to think louder, whatever you are thinking!

Note that interviewers choose hard problems on purpose. They want to see how you approach a problem you don’t immediately know how to solve. If you don't get every-or any-answer immediately, that's okay! But make sure you are able to communicate your thought process to the interviewer.

First, state a brute force approach. Despite being possibly slow, a brute force algorithm is valuable to discuss because it is a starting point for further optimizations required to build an optimal solution. You don't want your interviewer to think that you're struggling to see even the easy solution.

Then, you can optimize your solution gradually. Maybe, you may look for repeated work and try to optimize your solution by potentially pre-computing some result and use it

later, rather than having to compute it all over again. Maybe, using some data structure helps [Do think if you can use Hashmaps (unordered_map), Treemaps (map), or Heaps (priority_queue) once. Try to think, if you can model the problem as a graph problem]. Maybe, you can write some more test cases and find a pattern.

Once your interviewer is satisfied with your approach, you may start writing the code.

Writing a readable code

Use descriptive variable names. This may take a bit of time, but it will prevent you from losing track of what your code is doing. Functions returning booleans should start with "is_*". Names of arrays, vectors, etc. should end with "s.".

If you are using C++, try to use structure instead of pair.

If you are copying and pasting a large chunk of code, do think once, if instead you could create a function for executing those lines and call that function, multiple times with just different values of some parameters. [This is also called *Don't Repeat Yourself (DRY)*]

Always be explaining what you are currently writing/typing to the interviewer.

[Many companies would ask you to write your code on google docs, so make sure you have **set up google docs for coding** and done some practice of writing code over there.]

Test edge cases and state complexity

After writing the code, you must check whether your code works for all edge cases. These might include empty sets, single-item sets, or negative numbers.

Then, you can state the space/time complexity of the code and why it is so.

Extra Tip

Sometimes, the interviewer may ask some follow-up questions. A common follow-up question is how would you handle the problem if the input is too large to fit in memory or if the input is in the form of a stream. An answer for this is usually a divide-and-conquer approach— only read certain chunks of the input from disk into memory, write the output back to disk, and combine them later on.

PREPARE WELL, DON'T PANIC, AND BE CONFIDENT!

CodeISM, IIT(ISM) Dhanbad