

General

Hi all

Now that your Intern season are about to start in less than 4 months
So we thought of helping you all in our limits

You all we have 4 of us

1. Mohit bhaiya currently at IIT Ropar 4th year CSE and upcoming SWE at Google (1700+ in Codeforces)
2. Kinshuk Bhaiya currently at IIT Dhanbad 4th year Comp.Math. Prev Intern Zepto & upcoming SWE at Zepto (1700+ in Codeforces) + ICPC Regionalist'24 AIR-34 (pre)
3. Aindrela from Nit Agartala CSE (Upcoming summer Intern at ..)

I hope all of you are enjoying the break after mid-sems! :P

The first company that visited during my time was Google (off-campus) in June 2024.
So, you can expect the same around that time this year as well.

That means only three months are left before the real battle begins!

I'll primarily be helping with resumes, referrals, and off-campus opportunities, while Aditi, Kinshuk Bhaiya, and Mohit Bhaiya will guide you through mock interviews, coding, and competitive programming including ICPC.

For now Focus on the preparation part so We have made a Google Doc which will be shared shortly where you can find the preparation materials and strategies!

Contents

1. **Internship Prep Guide By CodeISM 2.**
2. **Kinshuk** will be focusing more on CP i.e., **Competitive Programming** along **ICPC**..
3. **Mohit** will be focusing on **CP/DSA** , **Interview** related preparation..
4. **Aditi** will be focusing on **Machine learning**, **CP**, **DSA** both for **learning & Interview** Prep..
5. **Aindrela** will be focusing on **Resume**, **Referrals** and **Off & On campus** Opportunities!
6. Open Source will be added soon!

Intern Prep Guide- CodeISM

Internship Preparation Guide

- By CodeISM

(**Youtube link for an 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. (especially 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 trees, 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_OCfOI0iAeDIqswCiC2DZzD4nPsb/view

Do keep these things in mind while solving these problems:

- a. **For theory of the topics, you may also refer to [articles from geeks for geeks \(GFG\)](#) 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 \(Including Priority Queue and Dequeue\)](#)
- 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, and 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 have 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!

Kinshuk

COMPETITIVE PROGRAMMING GUIDE -

What is Competitive Programming ?

Competitive programming in simple words is a sport programming in which programmers compete with each other for better results. They need to implement different algorithms to solve a particular problem statement. It involves making logics, reducing complexity of logics and then implementing it.

Who should do CP ?

Competitive programming is for anyone who enjoys problem-solving, logical thinking, and algorithmic challenges. If you are a CS or a related student you should try CP at least once.

If you are aiming for competitions for ICPC , CP is a must.

Why should we start competitive programming ?

Competitive programming helps enhance one's mind and develop analytical and thinking skills. It will improve your problem solving as well as debugging skills. Participating regularly in contests will make you more disciplined, fast as well as efficient.

CP is Like a Strategy Game – It's fun, addictive, and keeps your brain sharp. And also make your resume even more stronger.

When should we start CP?

Now, this is the right time to begin with CP.

Lets begin (Wooooo hoooooo) 🚀

Some of the pre-requisites of CP involves

1. Learning any programming language .(C++ preferred because of more number resources available)
2. Complete knowledge of all the uses of Standard Template Libraries (STL) like Maps, Sets, Priority Queue etc .
3. Please make sure you know the use case of each syntax you are using like using namespace std; // in this there are 3 things using, namespace and std and each of them have different meanings (I was asked this question in one of interview also)

To cover first two points it is recommended to complete at least 5 stars on hackerrank problem solving division.

Which platforms are needed for CP ?

If you want to be good at CP, make codeforces your better half, it has the best set of questions and also gives you an opportunity to compete with CPers all over the world. Once done with basics, start solving problems on CF and at the same time start giving regular contests. It organizes short contests very frequently, and the level and quality of questions are also very good.

Other than this CodeChef, AtCoder, and many more are there.

Topics to be covered :-

The topics i am mentioning below should be completed in same order as mentioned

1) Standard template libraries (again recap)

- a) Vectors
- b) Pairs
- c) Set
- d) Map
- e) Stack and Queue
- f) Priority Queue

2) Maths and Number Theory

- a) Prefix Sum
- b) Modulus
- c) Binary Exponentiation
- d) Checking primes
- e) GCD (greatest common divisor)
- f) Sieve of Erasthones
- g) Smallest Prime Factor (SPF)
- h) Number of divisors
- i) Euler Totient function (ETF)
- j) Modulo Inverse
- k) Combinatorics

3) Bits Manipulation

- a) Bitwise operations
- b) Bit Masking

4) Binary Search

5) 2 Pointers

(Above mentioned topics involve the basic implementation part of CP, these are really very important so make sure you have a good hold over these topics. If you are done till now you can reach a specialist easily with practice.)

After this there are advance topics

6) Dynamic Programming

- a) Recursive Approach
- b) Iterative Approach
- c) DP with bitmask

7) Graphs and Trees

- a) DFS and BFS
- b) Shortest path algorithms
- c) Disjoint set union (DSU)
- d) Minimum Spanning tree (MST)
- e) Strongly Connected Components (SCC)
- f) Diameter of tree
- g) Binary Lifting
- h) Least Common Ancestor (LCA)
- i) ReRooting
- j) DP on trees

8) String algorithms and Tries

9) Sparse table and Fenwick tree

10) Segment Tree

How to practice problem solving ?

Practice that I used to follow was learning any new topics only during weekends and then full week of practice of all standard and random questions of that particular topic.

Giving regular contests on all platforms and resolving them again with atleast 1 problem more than one's solved during the contest.

Once you learn or read about some topic, try making logic for standard problems. If you are able to do so then good otherwise simply try to get logic from editorial or any videos.

And make sure you write code on your own rather than simply copying from solution, this will improve your debugging skills also.

Read blogs and articles from websites like CPAlgorithms, Codeforces etc this will give you some more info on each topic.

How to approach a problem ?

- 1) Try to understand the problem completely like what the author wants you to answer, what are the constraints on variables etc. and identify the expected time complexity.
- 2) Try to make the simplest possible logic or a brute force on your solving sheet.
- 3) Try to solve questions using making functions and make sure to use call by reference.
- 4) Now comes the optimization part :-
 - a) Check whether you can use any STL to optimize it.
 - b) Check whether any part of logic resembles with algorithms use studied like binary search, prefix sum etc
 - c) Try storing some reusable things in maps, or vectors.
 - d) Now check for some advance algo like DP, segment tree etc.

Don't miss these Contests :-

- 1) Codeforces - All contest
- 2) Atcoder - Beginners contest every saturday 17:30
- 3) Codechef - Starters every wednesday 20:00

Also try to give a virtual contest on codeforces in your free time.

Giving 3-4 contests every week will help you improve problem solving, write code more faster and efficiently.

How to deal with situations when you are not able to solve questions ?

There are mainly 2 reasons for this —

- 1) You haven't studied or read about some particular algorithm that you need for that question. Just simply search about that also and read related articles and again come back after sometime and solve that problem.
Identify -> Learn -> Solve Again
- 2) The second case may be that you are familiar with the topic of the question but still, you are not able to think of the approach. In this case, try to divide the question into smaller tasks if possible and then try to solve, take hints and discuss it with peers. If none of this works, then see the editorial and try to analyze why you were not able to think of it.

Some important sheets for practise —

- 1) **Codeforces EDU course** :- good for topic wise questions
<https://codeforces.com/edu/course/2>
- 2) **TLE CP-31 sheet** :- Try to solve YourRating + 100/200 rated sheets at least
<https://www.tle-eliminators.com/cp-sheet>
- 3) **CSES Problem set**
<https://cses.fi/problemset/>
- 4) **AtCoder DP tasks**
<https://atcoder.jp/contests/dp/tasks>

Some other resources where you can read and study —

- 1) **Competitive programmer's Handbook:**
<https://cses.fi/book/book.pdf>
- 2) **CPAlgorithm** :- <https://cp-algorithms.com/>
- 3)

Mohit

How to Practice Problems in the Most Effective Way

When solving problems, we usually go through three main steps:

1. **Understanding**
2. **Approaching and Coding**
3. **Reviewing the Solution**

Let's break these down in a simple and practical way:

1. Understanding the Problem

This is the most important part, and it takes practice to get good at it.

- Always read the full problem statement **twice** before doing anything else.
 - If you're still not sure, use **1-2 sample test cases** to help you understand the problem (but don't overdo it).
 - Save the other test cases for **local testing** later.
 - Misunderstanding the problem at this stage will lead to wasted effort, so take your time here.
-

2. Approaching and Coding

Approaching the Problem

- **Write things down:** Start by making observations or conclusions on paper. If the constraints are tricky, make some simplifying assumptions for special cases and generalize later.
- **Set a time limit:** Spend **20-50 minutes** (depending on difficulty) exploring ideas. If you're not making progress, move on to the next step—there's no point wasting hours.
- **Brute force it:** If you're stuck, write a brute-force solution for small cases and look for patterns or corner cases.
- If you get an idea, double-check it. Think through all possible scenarios—are you confident it'll work?

Coding the Solution

- Before you write a single line of code, **plan your logic**. Break the solution into small parts and figure out what you're going to write.
- **Write and test incrementally:** Implement one part at a time, and test it to make sure it works before moving on.

- Use the remaining test cases you saved earlier for debugging.

Pro Tip: Multitask Your Thinking

- You don't always need to sit in front of a computer to solve problems. Pick **1-2 problem statements** and think about them while doing other activities—attending classes, etc.
- Once you figure out a solution, **add it to your queue** to code later and verify if it gets an AC.

If it doesn't work

- **Wrong Answer (WA)**: Check if it's because of a bug in the code, a flawed approach, or a misunderstanding of the problem.
 - Depending on the issue, debug, revise your logic, or reread the problem statement.
-

3. Reviewing the Solution

Even if you get an **Accepted (AC)**, **don't skip** this step. It's a huge opportunity to learn.

- **Read editorials**: They often reference useful articles that can help you improve.
 - **Use video solutions sparingly**: Watch them only if the problem statement or editorial isn't clear.
 - If you couldn't solve it:
 - Figure out if it's because you're unfamiliar with the concept or just missed the idea.
 - If it's a knowledge gap, lower the difficulty of problems or study the required algorithms.
-

Tips:

1. **Write contests regularly**: Participate in contests as often as you can and focus on **upsolving** problems afterward. This helps reinforce what you've learned and exposes you to new patterns and techniques.
2. **Practice daily**: Consistency is key. The effort you put in today might not show immediate results, but you'll see noticeable improvement in **1-2 months** if you stay consistent.
3. **Be patient**: Improvement doesn't happen overnight. Keep pushing yourself and trust the process.
4. **Multitask your thinking**:
 - You don't always need to be at your desk to work on problems. Pick **1-2 problem statements** and think about them during other activities like attending classes, commuting, or even taking breaks.

- Once you figure out a solution, **add it to your queue** to code later and verify if it gets an AC. This way, you make productive use of your downtime while keeping your mind engaged.

AI/ML

Here, I will mostly be mentioning stuff related to CP, DSA, and Machine Learning, both for learning and interview preparation.

AI Engineer = Data Scientist + Software Engineer

Machine Learning Journey

Machine learning requires you to master both **core programming skills** and specific **ML tools**.

1. Python

Topics to Cover:

- Variables, Numbers, Strings
- Lists, Dictionaries, Sets, Tuples
- If Conditions, For Loops
- Functions, Lambda Functions
- Modules (`pip install`)
- Reading and Writing Files
- Exception Handling
- Classes, Objects
- Inheritance, Generators, Iterators
- List Comprehensions, Decorators
- Multithreading, Multiprocessing

Resources: [Choose one]

1. [Programming with Mosh](#) – Free YouTube Course
2. [Udemy Course](#)

2. DSA in Python

If you already have a good understanding of DSA, you can skip this or binge-watch the content to refresh your knowledge.

Resources:

- [Complete DSA in Python](#) – Free YouTube Playlist
-

3. SQL

Topics to Cover:

- Basics of relational databases
- **Basic Queries:** `SELECT`, `WHERE`, `LIKE`, `DISTINCT`, `BETWEEN`, `GROUP BY`, `ORDER BY`
- **Advanced Queries:** CTE, Subqueries, Window Functions
- **Joins:** Left, Right, Inner, Full
- Database creation, indexes, stored procedures

Resources:

- [W3Schools - SQL](#)
-

4. Numpy, Pandas, Matplotlib, Seaborn

These tools are essential for working with data in ML. You only need to understand how to use them, not memorize the syntax.

Resources:

- **Numpy:** [Learn Numpy](#)
 - **Pandas:** [Learn Pandas](#)
 - **Matplotlib:** [Learn Matplotlib](#)
-

5. Exploratory Data Analysis (EDA)

EDA helps you understand data patterns and distributions. Very important for a data scientist.

[Try doing EDA of any dataset on Kaggle]

Resources:

- [EDA with Python](#)
-

6. Mathematics

You don't need deep math knowledge at the beginning — just enough to understand how ML works practically. You can go deeper later.

Topics to Cover:

1. Linear Algebra

- Vectors and Vector Spaces
- Matrices and Matrix Operations
- Matrix Factorizations
- Linear Transformations
- Systems of Linear Equations
- Eigenvalues and Eigenvectors
- Norms and Distances

2. Calculus *(Skip if you had a good understanding in 12th)*

3. Probability and Statistics

- Basic Plots: Histograms, Pie Charts, Bar Charts, Scatter Plots
- Central Tendency: Mean, Median, Mode
- Dispersion: Variance, Standard Deviation
- Probability Basics
- Distributions: Normal Distribution
- Correlation and Covariance
- Central Limit Theorem
- Hypothesis Testing: p-value, Confidence Interval, Type 1 vs Type 2 Error, Z-Test

Resources:

- [Mathematics for Machine Learning – Coursera](#) (Audit it for free)
 - [Linear Algebra by Gilbert Strang](#) (Only if you love Linear Algebra)
-

7. Machine Learning Concepts

1. Data Preprocessing

Data preprocessing is the foundation of any ML pipeline.

- Handling Missing Values
 - Outlier Treatment
 - Data Normalization and Scaling
 - Encoding Categorical Variables
 - Feature Engineering
 - Train-Test Split
 - Cross-Validation
-

2. Types of Machine Learning

- **Supervised Learning (most common)**
 - Regression – Predict continuous values
 - Classification – Predict discrete values
 - **Unsupervised Learning**
 - Clustering (K-Means)
 - Dimensionality Reduction (PCA)
 - **Semi-Supervised Learning**
 - **Reinforcement Learning**
-

3. Tree-Based Models

- Decision Tree
 - Random Forest
 - XGBoost
 - LightGBM
-

4. Unsupervised Learning Models

- K-Means Clustering
 - Hierarchical Clustering
 - Gaussian Mixture Models (GMM)
 - DBSCAN
 - Principal Component Analysis (PCA)
-

5. Model Evaluation

Regression Metrics

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R-Squared (R^2)

Classification Metrics (Very Important for interviews)

- Accuracy
 - Precision
 - Recall
 - F1 Score
 - ROC Curve
 - Confusion Matrix
-

6. Hyperparameter Tuning

- GridSearchCV
 - RandomizedSearchCV
 - Bayesian Optimization
-

All Learning Resources for ML Concepts (not DL right now)

Courses:

1. Andrew NG – Machine Learning Specialization (*Audit for free*)
2. [ML Course by Kaggle Master](#)
3. [Google's ML Crash Course](#)
4. CS50's Introduction to AI with Python
5. [Machine Learning by Siddhardhan](#)

Books:

- *Hands-On Machine Learning* – Aurélien Géron
- *Pattern Recognition and Machine Learning* – Christopher Bishop
- *Introduction to Statistical Learning* – Gareth James

YouTube Channels:

- 3Blue1Brown
- StatQuest with Josh Starmer
- ZedStatistics
- Codebasics
- Krish Naik
- Stanford Online
- Alexander Amini (MIT)

Projects to Begin With (Without deep learning)

1. Iris Flower Classification (*Beginner*)
 2. Build Your Own Linear Regression (*Intermediate*)
 3. Housing Price Predictor (*Beginner*)
 4. Sentiment Analysis System (*Intermediate*)
 5. Customer Churn Predictor (*Beginner*)
 6. Recommendation System (*Intermediate*)
-

Things You Must Keep in Mind:

1. **Data is a critical part of the entire AI arena.**

All existing models (ChatGPT, Grok, etc.) are limited in intelligence due to a lack of sufficient high-quality data. Gathering, cleaning, and refining data is an essential job for data scientists and AI engineers. You can't skip this (it's kinda boring but necessary)

2. **AI engineering is more like being a scientist than an engineer.**

ML engineering involves much more than just coding: statistical thinking, understanding data, problem formulation, model selection, business intuition, and the ability to handle real-world complexities. You need to balance both the technical and research aspects while working on ML. Understanding how models work, how they improve, and how the ML pipeline functions is essential. Even if you aren't into AI research, you should still think like a researcher.

3. **Don't get stuck in an endless loop of tutorials and courses.**

Focus on understanding the foundations and then apply that knowledge by building projects. Participate in Kaggle competitions, study notebooks from Kaggle masters, and read research papers. Try to implement research papers on your own — that's the best way to deepen your understanding.

4. **Build machine learning projects yourself — avoid copying code.**

When building ML projects, write the code from scratch as much as possible. Start by building linear and logistic regression models on your own before moving to more complex models.

Upcoming things i will be adding: -

1. I will provide detailed information about deep learning and other recent AI models, along with recommended resources and research papers to explore.
2. I will list some additional books and good projects.
3. I will write in detail about DSA interview rounds as well as ML interview preparation, including relevant resources.

Feel free to reach out if you have any questions. Happy learning :)

- Deep Learning
- Why are Data Science/AI Engineering jobs both uncommon and common?

Aindrela

I will be helping with building the resume during mid of April therefore no Qs related to Resume building will be entertained later (Because I will be busy during summer)

So utilize your time wisely

For now all of you focus on only & only on Preparation!

Right now beside preparation you have to focus on few other things as well for standing out in off campus opportunities

1. Referrals
2. Small Internship
3. Open source
4. Participate in hackathons and contests to put in your Resume as Achievements!

1. Referrals! How to approach getting a referral right?

- At first send requests to more than 20+ employees per company, mostly SDE 2 & 3 and some famous Linkedin people. If they refer you chances are really high 10K followers on Linkedin is really great with SDE 3 or SDE 2 employees!
- Now as soon as they accepts your request don't write too many msg

Scene 1: If you are sending the msg when the opportunity has come. Everyone mostly does this but I would prefer scene 2. So in that case just sent this msg →

Here's a short and crisp referral format:

Good Afternoon Sir/Ma'am

I hope You are doing well!

Recently, I have come across a job opportunity with Google for Summer Intern 2025→ [Opportunity Link]

I would greatly appreciate your support in providing a referral for me.
Let me know if there's any information you need from my side.

My resume Link → [Google Drive Link]

Thank you for considering this request.

Best regards,

Aindrela Saha

Scene 2: Just start a casual conversation with the employee asking how to prepare for the coming opportunity and slowly ask him/her to provide you a referral in future. You can ask their mail/Contact no. so that you can get in touch with them when you will actually need a referral.

This way is better than seeking a referral just the day when the opportunity has come. Scene 2 is better than scene 1 because scene 1 will be more tough as thousands of students will be seeking a referral at that time so it's always better to have it beforehand.

- Last but not the least, be active on Twitter. A lot of people give referrals in twitter and it's easier to get a referral in twitter than LinkedIn nowadays!

2. Small Paid Internships/ Freelancing : The First thing that comes while applying in any opportunity is RESUME which is equally important just like Preparation! Therefore to have the best resume you should always have something unique than other applicants! Try getting small Internships done now so that you can put the certificates in your resume and also RESUME plays a crucial & major role when it comes to Off campus opportunities!

Few Channels/Sites where you can find such opportunities are given below →

<https://whatsapp.com/channel/0029Va9sXNw3AzNa6mBs060w>

<https://whatsapp.com/channel/0029Va6I79K60eBfQ92DwH0W>

Wellfound


3. Open source: Do open source whether in Gsoc or in Gssoc(Girlscript summer of code) but do it so that you can have something different in your resume!

4. Participate in hackathons and regularly give contests in Codeforce : Participating in hackathons will teach you many things just like giving contests reduces your pressure during an OA! And both of their achievements can be showcased in your resume! Look into UNSTOP for hackathons and various other sites such as <https://www.talentd.in/index.php> for hackathons!

Open Source soon to be added

Question Answer


Q/A & Feedback

If You have any Question always put it here on this Doc (link Given )

No one is going to reply on WhatsApp msg so use this Doc For your Questions + Feedback

Feedback

Q/A & Feedback

If You have any Question always put it here on this Doc (link Given )

No one is going to reply on WhatsApp msg so use this Doc For your Questions + Feedback