Intro

These interviews were all for full-time positions. The collected resources below were combined into a study schedule that would be manageable for a full time employee. It's designed to take about 1-1.5 hrs a day on weekdays, and 5-8 hours over the weekend.

Study Schedule

Python is the recommended language for generalist SWE interviews, and the majority of my preparation was done with Elements of Programming Interviews in Python (EPI). For each problem in EPI, I set a timer for 25 minutes, and I attempted to solve the problem with pen and paper in that time (whiteboard is even better). As soon as the timer was up, I would stop what I was doing and look at the answer. As soon as I understood the answer, I would write a code complete solution, again with pen and paper. Afterwards, I checked with the test cases online, and worked out any bugs that came up. I think this is a good way to balance writing code by hand, speed, and actually understanding the solution. Each problem took about 20-60 min overall, although some took longer.

Leetcode premium unlocks all the questions, specifically the company-specific questions.

Mondays: Read the core concepts sections in EPI and did the recommended '3 days' problems (supposedly, the problems you would solve if your interview is in 3 days times).

Tuesdays: Called some friends on Discord, and mock interviewed each other. Usually these were 1 hour sessions, with 30 minute long interviews (we took turns).This was probably the most important part of my study plan - emulating interviews improves communication skills and on-the-spot thinking.

Wednesday-Fri: Bulk EPI problem solving. Aimed for 2-3 each day. The actual schedule is below. Green indicates the entire process was completed (25 min by hand + online checking). Orange problems were only done optimally by hand, and red problems were tried and failed.

Saturday: Leetcode has free online 'contests' where they release 3-4 new problems every Saturday evening. These have a 90 minute hard time limit and are very good practice for phone rounds and Hackerrank challenges. For reference, finishing all 3-4 problems in 90 min is very difficult - personally aimed to finish all the easy and mediums.

Saturdays were also for system design. I mostly followed the Github system design primer, not because it's the most comprehensive resource, but because it offered me the most bang for the buck. More on that below.

Sunday: I tried to solve 1 leetcode Hard problem, with a focus on the more popular ones (e.g. merge k-sorted lists). However, EPI did a great job covering a lot of these.

I also spent some time preparing for machine learning engineering interviews. I took as many ML classes at UCSD as I could, but I've found the best resource for interviews to be Introduction to Statistical Learning w/ R. I followed the associated online Stanford course that the authors created. Personally, I found it to be much more helpful than the Andrew Ng course in introducing statistical concepts in an intuitive way. The course goes into just enough mathematical background to help remember why things work, but the math itself is fairly straightforward. I had actually done the course beforehand, but I did a speedrun through it, taking notes. There are 9 chapters in the course, and I think it's reasonable to do 1 chapter every 2 weeks for someone who hasn't done it before.

Before Phone Rounds:
This is the weekly schedule I followed for 15-ish weeks, until phone rounds started.

Monday - Read EPI, do '3 days' problems
Tuesday - Mock interviews with dedicated interview group (1hr)
Wed,Thurs,Fri - Bulk EPI problem solving
Saturday - Leetcode Contests, System Design
Sun - Leetcode Hard problem, ML

After Phone Rounds

- Complete the 41 ML Questions list.
- More system design case studies
- More leetcode (medium/hard)
- Company specific questions (leetcode premium, stalk blind and glassdoor)
- 75 question list compiled on Blind (listed below)

Wk	Mon (EPI)	Tues	Wed	Thurs	Fri	Sat	Sun
1	4.1 4.7	4.8 4.3	4.2 4.4		4.11 4.9		HARD Q
2	5.1 5.6		5.12 5.18 5.2	5.17 5.5 5.9	5.3 5.10 5.15	TinyURL (tushar)	HARD Q
3	6.1 6.2 6.4	6.5 6.6	LC1029 LC1031	6.7 6.8	6.9 6.11	…	HARD Q 6.13
4	7.1	7.3	7.10	7.5	7.12	...	HARD Q

	7.2	7.4 7.7	7.11 7.6	7.8 7.9	7.13 6.13		etc..
5	8.1 8.7	8.2 8.8 8.3	8.9 8.4	8.5 8.6 8.10		...	
6	9.1 9.4 LC 3	9.2 9.12 9.11	9.13 9.16 9.3	9.5 9.6 9.7	9.8 9.9 9.10	9.14 9.15 9.17	
7	10.1	10.4 10.3	10.5 10.7	10.2 10.6		...	
8	11.1 11.4	11.8 11.3	11.9 11.5	11.10 11.6	11.2 11.7	...	
9	12.1 12.2 12.3	12.6 12.4 12.7*	12.10 12.5 12.8	12.9 12.11 12.12	12.13 12.14	Youtube	
10	13.1	13.2 13.5 13.7	13.10 13.8 13.3	13.4 13.6 13.9	13.11 13.12	Typeahe ad	
11	14.1 14.2 14.3	14.4 14.8 14.5	14.9 14.7 14.10	14.6	14.12	API ratelim	
12	15.1 15.2	15.3 15.4 15.9	15.6 15.10 15.5	15.7 15.8 15.11		Twttr Search	
13	16.1		16.2 16.3	16.6 16.7 16.12	etc	Web Crawler	
14		17.4 17.6 17.5	17.7 17.8 17.1	17.3		Newsfee d	
15	18.1 18.7 18.2	18.3 18.9 18.8 topsort	18.4 18.2			Nearby Friends	
16						Uber backend	

17						Ticketmaster	
18							

Interview Formula

In order to emphasize communication skills, I followed the following formula for every phone and onsite round. This helped me understand the problem, figure out edge cases, and get myself on the right track without really thinking too hard and tripping myself up.

1. Literally write down “Assumptions” and underline it. Write down the following things you clarify from the interviewer.
2. Define Input constraints
    a. e.g. if string, Write Input **Type** and input **length**
3. Understand the problem
    a. Discuss **Brute Force** method that you know will solve the problem 100%. Discuss time and space complexity.
    b. e.g. for 2 sum, write down “brute force. time: $o(n^2)$, space o(1)”.
    c. This shows the interviewer that you have a starting point and you understand the question. Before moving on, ask the interviewer if you’re on the right track.
4. Run through 1-2 examples with expected **return output**
    a. Write down “Test case 1:” and “Input:” and the correct output: “output”
    b. If the interviewer gave you an example, write those down.
    c. Think of all the possible edge test cases that could be a problem, and write those down.
    d. Always add an example test case that you made up. Try to make it more difficult.
5. Discuss and draw out **improved** algorithm.
    a. Write down “Ideas” and start listing things you think might help. It’s helpful to refer back to this list, and helps your interviewer be on the same page.
    b. draw out - draw diagrams and **write pseudocode**
6. Run through exactly 1 example test case in #3 as soon as possible.
    a. I tried to do this as formally as possible, like a stack trace, by using variables and writing their values, crossing them out when necessary.
    b. By writing down things like “index: ~~0 1 2 3~~ 4” and “a: ~~0 1 2 4~~ 9”, I found myself able to correct any incorrect ideas and really understand my solution, before writing any code.
    c. This step usually took me the longest.
7. Write code complete algorithm
    a. This step should not take that long, b/c I already did all the hard thinking.

    b. Refer to pseudocode and diagrams in step 5 - this step is really just to please the interviewer and be able to write code that can pass a compiler/interpreter
8. Discuss tradeoffs with interviewer

Study Resources

Data Structures & Algorithms

- 1 Hour Study Session (github)
- Leetcode Patterns
- Structured List of Concepts
- Interview Cake
- The Algorithm Design Manual
- Arden Dertat's Interview Questions List
- Top 100 Questions
- Cram Score Questions
- Collaborative Tech Interview Handbook
- For Finance companies (2sig, js, citadel, hrt, etc..)
    - C++
    - OS (flags and shit, deadlock)

Something to review basics/cool algos:

- https://www.coursera.org/learn/algorithms-part1
- https://www.coursera.org/learn/algorithms-part2
    - Good for review of union find, string matching/sorts (KMP, radix sort), and tries

If you're hardcore AF:

- https://codeforces.com/problemset
- https://www.topcoder.com/community/competitive-programming/

System Design

System design was more difficult to study for, although there are plenty of resources online. I've collected a few below. I managed to pass my system design onsite rounds by studying as many of the examples on the Github system design primer, and watching some videos on youtube when I didn't understand a concept.

What I did to start was by watching this Harvard lecture on scalability on Youtube. It's 2 hours long, but it's very good at introducing the concepts in an intuitive and natural way. I would recommend taking notes and watching it on 1x speed. I actually watched this video twice - once

at the beginning when I understood nothing, and once after I studied all the terms and concepts on the System Design primer to tie it all together.

While I recommend absorbing as much of the System design primer as possible, I found this one problem VERY VERY USEFUL for the actual system design interview. This one is unique because it introduces the problem in a gradual way, starting with a very simple system and adding on improvements for each identified bottleneck. The system design interviews that I had reflected the iterative nature one problem very closely. I would recommend understanding and memorizing this problem.

Resources:

- High Scalability
- Github System Design Primer
- HiredInTech Course
- Designing Data-Intensive Applications
    - On libgen
- Mining Massive Datasets
    - I started this course a while ago, and would highly recommend it.
- Cool course in distributed systems : https://pdos.csail.mit.edu/6.824/
    - https://www.youtube.com/playlist?list=PLpl804R-ZwjKCOwWpTZ21eeaBS3uBrMfV
- https://www.youtube.com/channel/UCZLJf_R2sWyUtXSKiKlyvAw
    - Has a bunch of tutorials taking you through system design interview problems in context of interviews

Machine Learning

Machine Learning was definitely the hardest to prepare for, as someone who hasn't really studied ML in depth. Like I mentioned before, going through ISLR and taking notes was the best resource for me.

The UCSD recommender systems course CSE 158 is very useful and applicable to ML design interviews. I took this course, but I went through all the slides to review concepts such as the cold start problem (very relevant to industry) and basic things like cosine similarity.

One thing I'm really glad I prepared for is the derivation of gradient descent for linear regression. Specifically, the derivation for MSE as a loss function. I memorized this, because it came up so many times.

I also read as many examples of ML interview problems as I could find, which are below.

Resources

- ISLR

- Sci-Kit Learn Cheat Sheet
- 41 Essential ML Questions
- Cracking the ML Interview
- ML System Design
- 5 Lessons in building ML systems
- ML System Design - Spam Classifier
    - Talks about Precision vs. Recall

SQL & Data Modeling

I never actually studied this, and it never came up in my interviews.

- Mode Analytics SQL tutorial
- Kimball's The Data Warehouse Toolkit

Interview Heuristics

I did my research on why I wanted to join the company. I looked up each company's values, and read a tech blog article or some relevant news story so I would have something to talk about.

Algorithmic Heuristics

- Recursion Template from Leetcode
- Sliding window problems
- Formally define the recurrence relation for DP problems
    - T(n) = T(n/2)
    - T(1) = 1

Interesting (hard) Problems

- Median of Medians (cool)
- Merge k sorted lists
    - pop heap, store tuples/class in heap
- Cherry Pickup (memorize optimal answer)
- Bidirectional BFS
- Lexicographic order
- LRU Cache
- Russian Doll Envelopes [redacted]
- Serialize + Deserialize BST
- Random Integer with Blacklist
- KMP/Rabin Karp (some sort of efficient string matching)
- Topological sort

- Number of Islands II [redacted]
- Line sweep algo
  [redacted]
  - https://leetcode.com/tag/line-sweep/
- Solve sudoku [redacted]
- Sliding window problems
- Random pick integer with blacklist
- Need to add “classic” leetcode hard problems like LRU Cache
- Complex data structures
  - Fenwick Tree
  - Bloom Filter
  - Suffix Tree
  - Segment/Interval tree

75 questions Blind Curated List

Array

- - Two Sum - https://leetcode.com/problems/two-sum/
- - Best Time to Buy and Sell Stock - https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
- - Contains Duplicate - https://leetcode.com/problems/contains-duplicate/
- - Product of Array Except Self - https://leetcode.com/problems/product-of-array-except-self/
- - Maximum Subarray - https://leetcode.com/problems/maximum-subarray/
  - Linear
  - Divide and Conquer
- - Maximum Product Subarray - https://leetcode.com/problems/maximum-product-subarray/
  - DP solution
    - M = max(m*a,M*a,a)
    - m = min(m*a,M*a,a)
    - return final M
- - Find Minimum in Rotated Sorted Array - https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/
  - Binary search with edge cases and handle length < 3
- - Search in Rotated Sorted Array - https://leetcode.com/problems/search-in-rotated-sorted-array/
  - Find index of minimum in rotated sorted array
  - then binary search on either half
- - 3Sum - https://leetcode.com/problems/3sum/
  - Because it will be $O(N^2)$ anyway, we can sort the array first.
  - This gives us the ability to search for a pair of numbers using a Left and Right pointer instead of using a linear pass + dict

    - Time = O(NlogN + N^2)
- - Container With Most Water - https://leetcode.com/problems/container-with-most-water/

Bitwise Bullshit

- - Sum of Two Integers - https://leetcode.com/problems/sum-of-two-integers/
- - Number of 1 Bits - https://leetcode.com/problems/number-of-1-bits/
- - Counting Bits - https://leetcode.com/problems/counting-bits/
- - Missing Number - https://leetcode.com/problems/missing-number/
- - Reverse Bits - https://leetcode.com/problems/reverse-bits/

Dynamic Programming

- - Climbing Stairs - https://leetcode.com/problems/climbing-stairs/
    - Recurrence Relation:
    - T(n) = T(n-1)+T(n-2)
    - T(0) = 0, T(1) = 1
- - Coin Change - https://leetcode.com/problems/coin-change/
- - Longest Increasing Subsequence - https://leetcode.com/problems/longest-increasing-subsequence/
- - Longest Common Subsequence -
- - Word Break Problem - https://leetcode.com/problems/word-break/
- - Combination Sum - https://leetcode.com/problems/combination-sum-iv/
- - House Robber - https://leetcode.com/problems/house-robber/
- - House Robber II - https://leetcode.com/problems/house-robber-ii/
- - Decode Ways - https://leetcode.com/problems/decode-ways/
- - Unique Paths - https://leetcode.com/problems/unique-paths/
- - Jump Game - https://leetcode.com/problems/jump-game/

Graph

- - Clone Graph - https://leetcode.com/problems/clone-graph/
- - Course Schedule - https://leetcode.com/problems/course-schedule/
- - Pacific Atlantic Water Flow - https://leetcode.com/problems/pacific-atlantic-water-flow/
- - Number of Islands - https://leetcode.com/problems/number-of-islands/
- - Longest Consecutive Sequence - https://leetcode.com/problems/longest-consecutive-sequence/
- - Alien Dictionary (Leetcode Premium) - https://leetcode.com/problems/alien-dictionary/
- - Graph Valid Tree (Leetcode Premium) - https://leetcode.com/problems/graph-valid-tree/
- - Number of Connected Components in an Undirected Graph (Leetcode Premium) - https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/

Interval

- - Insert Interval - https://leetcode.com/problems/insert-interval/
- - Merge Intervals - https://leetcode.com/problems/merge-intervals/
- - Non-overlapping Intervals - https://leetcode.com/problems/non-overlapping-intervals/
- - Meeting Rooms (Leetcode Premium) - https://leetcode.com/problems/meeting-rooms/
- - Meeting Rooms II (Leetcode Premium) - https://leetcode.com/problems/meeting-rooms-ii/

Linked List

- - Reverse a Linked List - https://leetcode.com/problems/reverse-linked-list/
- - Detect Cycle in a Linked List - https://leetcode.com/problems/linked-list-cycle/
- - Merge Two Sorted Lists - https://leetcode.com/problems/merge-two-sorted-lists/
- - Merge K Sorted Lists - https://leetcode.com/problems/merge-k-sorted-lists/
- - Remove Nth Node From End Of List - https://leetcode.com/problems/remove-nth-node-from-end-of-list/
- - Reorder List - https://leetcode.com/problems/reorder-list/

Matrix

- - Set Matrix Zeroes - https://leetcode.com/problems/set-matrix-zeroes/
- - Spiral Matrix - https://leetcode.com/problems/spiral-matrix/
- - Rotate Image - https://leetcode.com/problems/rotate-image/
- - Word Search - https://leetcode.com/problems/word-search/

String

- - Longest Substring Without Repeating Characters - https://leetcode.com/problems/longest-substring-without-repeating-characters/
- - Longest Repeating Character Replacement - https://leetcode.com/problems/longest-repeating-character-replacement/
    - Sliding window
- - Minimum Window Substring - https://leetcode.com/problems/minimum-window-substring/
- - Valid Anagram - https://leetcode.com/problems/valid-anagram/
- - Group Anagrams - https://leetcode.com/problems/group-anagrams/
- - Valid Parentheses - https://leetcode.com/problems/valid-parentheses/
- - Valid Palindrome - https://leetcode.com/problems/valid-palindrome/
- - Longest Palindromic Substring - https://leetcode.com/problems/longest-palindromic-substring/
- - Palindromic Substrings - https://leetcode.com/problems/palindromic-substrings/
- - Encode and Decode Strings (Leetcode Premium) - https://leetcode.com/problems/encode-and-decode-strings/

Tree

- - Maximum Depth of Binary Tree - https://leetcode.com/problems/maximum-depth-of-binary-tree/
- - Same Tree - https://leetcode.com/problems/same-tree/
- - Invert/Flip Binary Tree - https://leetcode.com/problems/invert-binary-tree/
- - Binary Tree Maximum Path Sum - https://leetcode.com/problems/binary-tree-maximum-path-sum/
- - Binary Tree Level Order Traversal - https://leetcode.com/problems/binary-tree-level-order-traversal/
- - Serialize and Deserialize Binary Tree - https://leetcode.com/problems/serialize-and-deserialize-binary-tree/
- - Subtree of Another Tree - https://leetcode.com/problems/subtree-of-another-tree/

- - Construct Binary Tree from Preorder and Inorder Traversal - https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/
- - Validate Binary Search Tree - https://leetcode.com/problems/validate-binary-search-tree/
- - Kth Smallest Element in a BST - https://leetcode.com/problems/kth-smallest-element-in-a-bst/
- - Lowest Common Ancestor of BST - https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/
- - Implement Trie (Prefix Tree) - https://leetcode.com/problems/implement-trie-prefix-tree/
- - Add and Search Word - https://leetcode.com/problems/add-and-search-word-data-structure-design/
- - Word Search II - https://leetcode.com/problems/word-search-ii/

Heap

- - Merge K Sorted Lists - https://leetcode.com/problems/merge-k-sorted-lists/
- - Top K Frequent Elements - https://leetcode.com/problems/top-k-frequent-elements/
- - Find Median from Data Stream - https://leetcode.com/problems/find-median-from-data-stream/

Miscellaneous Resources

- FB Success Story
- Career Planning

Application Process

Getting interviews was by far the hardest part. Other than referrals, something that worked well was applying online, finding 2-3 company recruiters on LinkedIn, and pestering them with requests to look at my application. I emphasized my interest in the company, my relevant qualifications, and the fact that I was messaging them so that my application doesn't get lost in a sea of other applicants.

- Github Easy Application List
- AngelList
- Breakout Companies

Phone interviews

Please PM for initial rounds at Quip, UnifyID, Scale.ai, Bolt, Front.

Onsite interviews

I'll list my experiences with the onsite rounds, because I think it'll be helpful to some. The opinions are just my opinions - things like culture fit are always subjective. The following describes the mindset I had when choosing companies.

Affirm

- SF fintech startup (500~ engineers)
- Excited vibe in the office
- Interesting engineering problems
- ML Engineering position
- 2 practical algo+ml rounds, 1 stats round (rip)

The interviews were challenging because they were non-traditional but very fair. I was pleasantly surprised by the interview questions, because they asked things that are reasonable for every ML engineer to know, and were things that might actually be used on the job. There was a statistical intuition round which was very difficult for me (wasn't prepared for in depth stats). Ended up getting rejected, but I had a great experience overall.

Uber ATG

- Newly public large company (1000+ engineers)
- Strong mission statement
- Potential for growth
- Very interesting engineering problems
- SWE position on Perception team
- 1 bar raiser round, 1 algo, 1 design-focused algo, 1 system design+architecture, 1 hiring manager

I would say Uber's interviews were on the medium side of difficulty. There was a system design round which the Github primer helped with to an extent, until the interview got too detailed and I told the interviewer that was the extent of my knowledge. He seemed satisfied, and then transitioned to projects. The algo rounds weren't too difficult, but I prepared well enough for those.

Gusto

- Growing payroll services startup (150~ engineers)
- Strong mission statement
- High growth potential that seems very stable
- Opportunity to work on rapidly growing teams
- SWE position
- Strong culture, passionate people

- 2 algo rounds, 1 pair programming, 1 culture fit, 1 manager

Gusto's interview was by far my favorite, and I highly recommend everyone to apply there. The algo rounds were around Leetcode medium, but the interviews were different in that they were much more cooperative than most. I particularly enjoyed the pair programming round, and I think I learned quite a bit on good coding practices from that session. The environment is also very cooperative and I could tell everyone enjoyed being there, which really stood out from the other companies. Culture fit is particularly important at Gusto, so I made sure to read up on their values and picked a few that I identified with the most.

LinkedIn

- Large established public company (5000+)
- Interesting and important ML work
- ML Engineering position
- Strong employee-centric culture
- 2 algo rounds, 1 ML design, 1 ML engineering, 1 manager round

The LinkedIn interviews were pretty difficult in terms of length. The rest covered general ML knowledge, ML product design, etc. This post sums up the interviews pretty well. I found my ML prep to be sufficient for this role.

The phone rounds were an interesting mix of rapid-fire ML knowledge questions (30 min) and a leetcode medium level algo question (30 min). Not much thinking involved in either section, mostly recall.

Waymo

- Large private company (1000+?)
- Very talented engineers
- Very interesting engineering problems
- Strong mission statement
- Very high growth opportunity in terms of learning
- Software engineering role in Simulation
- 4 algo rounds, 45 min each

The Waymo interviews were brutal. Leetcode hard doesn't really do the difficulty justice. However, I would say the problems were fair, and mostly they were difficult because of the length and scope of the problem, and not because of an obscure algorithm that I needed to know. What really helped with Waymo was that I practiced my communication skills, and was able to work with the interviewers extensively to reach a solution by the end of the interview. The interviews were also shorter than most (45 min) and back-to-back, which was grueling. Speed was of the essence for these interviews.

Compensation / Negotiating

Resources

Candor - very good article to read regarding mindset and wording
Angellist

Below are the questions that were important to me – after receiving an offer, recruiters generally called me on the phone for 30 minutes. I took that as an opportunity to ask these questions.

What to do on the phone call giving the Offer

- **What level is the offer?**
  - What are the requirements for this level vs. the level above it?
- **What is the salary band for this level?**
  - This is 100% completely reasonable thing to ask. In California, an employer must legally provide this if asked.
- **How much is the equity worth currently?**
  - You can also ask: what percentage of the company does the equity represent? What is the valuation of the company?
- **What is the vesting schedule?**
  - Is there a 1 year cliff? Are there quarterly vesting deadlines I should know about? Confirm whether the equity was quoted to you on a yearly basis or over 4 years.
- **For options: what is the strike price?**
  - (the price you'll pay for the shares) How long after leaving do I have to exercise the options?
- **“How are raises calculated and awarded?”**
  - What about promotions? (less relevant for startups)
  - You want to make sure raises aren't uncommon in this organization.
- **“How do bonuses work?”**
  - You want to see generous bonuses that are tied to plausible goals.

Startups

Growth

- **“What progression do you envision for someone in this role?”**
  - You want a company that plans for your growth.
- **“Does this role contribute to higher-level decisions?”**
  - The more responsibility you get out of the gate, the better.
- **“Will I be able to learn new skills and technologies in this role?”**
  - You don't want to get stuck maintaining legacy code.

Money

- **“Do you have product-market fit?”**
    - If not, they don't have real money, and there's no guarantee they ever will.
- **"What is your current growth rate?”**
    - Your earning potential grows with revenue. Slow growth means less income.
- **“What is your runway?”**
    - The longer the runway, the more financially stable the company is.
- **Extended Exercise Window?**
    - https://github.com/holman/extended-exercise-windows

Exploding Offers

- Sometimes a tight deadline is legitimate but often it’s a mechanism to prevent you from looking around too much. “Exploding offers” are a bad industry practice and unfortunately very common with new grad offers, where companies are trying to lock down candidates before everyone else. Read the rest

Compensation data points:

Large tech companies (FAANG, Microsoft, LinkedIn, etc..) generally pay very well. Google is a sort of benchmark, but the best way to know your market worth is to get multiple offers, and ask your friends what they got.

Returning Interns at UCSD: I believe that companies that recruit interns through UCSD have to give you until the end of October before you have to commit to a full time conversion. Double check this date with the career center.

I’ve found levels.fyi to be more or less accurate.

Google

Note: In my personal opinion, the numbers below are on the higher side. If you take anything away from this though, it should be that you should always negotiate upwards. Generally, base salary is the hardest to negotiate, and signing bonus is the easiest.

Source (2018)
Format is the following:
Level
1% - 10% - 25% - 50% - 75% - 90% - 99% - compensation percentile

Percentile	1%	10%	25%	50%	75%	90%	99%

L3	120	125	150	200	220	225	250
L4	200	225	240	260	280	300	330
L5	250	285	315	330	365	390	450
L6	375	400	425	450	480	550	575
L7	550	575	625	675	725	750	800