How to practice Leetcode for 2200+ rating

I have done Lettcode for more than 4 years. As more and more practice, my LC rating gradually increased to 2400+ in one month ago (though sadly dropped to 2350+ recently). It seems like although many people ranked much higher than me, the blogs related to self-practice are still very limited. So I think it's maybe a good time to share some experience. I hope the blog could motivate anyone interested in LC and make self-practice a little easier. The post is very limited to my own experience and must not be the most effective strategy fitting for everyone. If you think it's helpful, I appreciate it. If you think it's very biased or nonsense, then definitely do practice in your way.

The Article will consist of 6 parts below:

1. What is LC rating
2. Does the LC rating matter to the job opportunity
3. How to practice as a beginner
4. How to practice for 1800 ~ 2200
5. How to practice for 2200+
6. Some really good readings and resources.

Let's go over them one by one.

What is LC rating

Like many of the contest websites, LC follows the ELO system. LC rating is a good way to measure the ability to solve problems, the higher the rating, the higher chance to solve a problem in a contest.

Technically, problem rating is the rating of a hypothetical contestant such that his/her expected rank equals the number of participants who solved the problem. In practice, if you meet a problem whose rating is equal to your rating, you are expected to solve it in half of your contests (Reference).

Thus rating is a good way to make self-practice more effective. Try to start with problems close to your LC rating ([your rating - 200, your rating + 200]), these problems could be good fits for you. Otherwise, they're either too easy or too hard.

Unfortunately, LC doesn't provide the official problem rating like some other websites, for example, codeforce. However, the legendary LC player @zerotrac does provide a LC rating list computed from each contest. Here is the github link. The rating list is much more precise than LC official tags (easy, medium, hard). As mentioned before, practice with problems close to your rating may be a good approach.

Does LC rating matter to job opportunities?

I hope the answer could be "Yes". However, unfortunately, from my personal experience, ***the LC rating is not related to job opportunities very much***.

There're too many factors needed to find a good offer, for example, good luck, algorithm skill, presentation skill, system design skill, education background, interviewer, or even the weather. Algorithm skill is only one of them, though it may be an important one. There's no guarantee or relationship like if I can make my rating

higher than X, then I can get an offer from company Y. You can get an offer from company Y no matter what your LC rating is.

But the good news is the higher your LC rating, the higher chance you can impress your interviewer (If they asked for an algorithm solving session, of course). The statistical data from another legendary player @wisdompeak shows that the median rating from SDEs who got an offer from FAANG company varies a lot. The list below shows more details.source

1. Meta: 2100+
2. Microsoft: 1900+
3. Amazon: 1900+
4. Google: 2100+
5. Bytedance(Tiktok): 2200+

Please note, that the data above may be very biased. The statistical data are collected from people who are very actively practicing in LC and the number of samples is very limited. I believe the rating mentioned above should be higher than practice. For example, if we can ask every SDEs in Amazon to do leetcode contest, I believe the median rating should be below 1900. Because as far as I know, most of the SDEs in FAANG company are not practicing in LC very actively :)

From my point of view, the data above are kinds like a good indicator or the expectation from an interviewer. If you can reach the rating when you are doing the interview, you can likely impress your interviewer. But again, ***it's only a chance, and algorithm skill is only one of many factors to get a good offer***.

How to practice as a beginner

Although leetcode is much easier than some other OJ, for example, codeforces. Most of the problems could be still tricky for beginners.

I think for a beginner, the most important part is to be confident. Every gain comes after a great amount of practice. If you want to improve your skill, you have to spend some time. Even the most genius person still needs to practice and he/she still unavoidably makes mistakes.

I think keeping a good mindset is the key for beginners. When I first start leetcode, I often doubt my ability. Am I too stupid to solve the problem? Is it something done wrong by my side? But after some practice, I gradually become more and more comfortable with the problems. After I solved about 400 problems (I often need to find solutions from discussion of course), I can solve 60% ~ 70% of medium problems without looking at the solution. I think my IQ doesn't change over the period, the key point is to practice and learn from others' solutions.

It's totally fine to admit that some problem is a little too hard at the current moment. If you cannot solve a problem at this time, it doesn't mean you're not good at coding or you're stupid. It only shows that maybe you're not familiar with some concepts or patterns. I think it will be a good time to look at the solution, learn the concept, and practice with some similar problems. ***The list*** maintained by @wisdompeak is a good reference. You can always find problems related to some specific topics from this list.

And I think the 3 topics below are most crucial for beginners.

1. Basic data structure

> *Array/Vector, Queue, Stack, Linked List, Sorted Map/TreeMap, UnSorted Map/HashMap, Priority Queue/Heap, Tree, Graph.*
>
> *You may need to know the implementation details of Linked List, And you may need to know how to build a stack/queue from linked list.*
>
> *For Sorted Map/TreeMap, UnSorted Map/HashMap, Priority Queue/Heap. You need to be familiar with API provided by your programming language.*
>
> *Most of the languages don't support Tree and Graph natively. You need to know how to represent Tree and Graph by yourself.*

2. Recursion

> *You may need to know how recursion works. And what is the relationship between recursion and stack. How to emulate a recursion call by stack.*
>
> *You may need to know some classical Recursion algorithms. For example, Quick Sort, Preorder Vist, Inorder Visit, Postorder Vist Euclidean Algorithm, and so on.*
>
> *You may need to know how to use the memoization technique to implement Dynamic Programming in a top-down fashion. From my point of view, I think coming up with a top-down DP solution is much easier and more intuitive than traditional bottom-up DP.*

3. Basic Algorithm Concept

> ***Sort*** *You may need to be familiar with several sort algorithms. For example, Bubble Sort, Insert Sort, Heap Sort, Merge Sort, and Quick Sort of course.*
>
> ***Breadth First Search*** *You may need to know how to write a general BFS algorithm by Queue. And you may need to know some basic applications of BFS, for example, finding the shortest path in an unweighted graph.*
>
> ***Depth First Search*** *You may need to know how to write a general DFS algorithm by Recursion. And you may need to know some basic applications of DFS, for example, Preorder visit.*
>
> ***Divide and Conquer*** *This one is not very common in Leetcode. If you're not interested in. I think you can skip this topic. But you may want to know how to implement quicksort.*
>
> ***Dynamic Programming*** *This is a notoriously well-known hard topic. The most difficult part is recognizing the problem is a DP problem. You have to do a lot of practice to get the intuition. I think we can start from the well know one, for example. Longest Common Subsequence, Longest Increasing Subsequence.*
>
> ***Greedy*** *From my point of view. I think the greedy problem is the toughest one. You have to have good intuition to resolve the greedy problem. And it's commonly super hard to prove if the greedy is correct. I am not good at greedy problems at all. Luckily enough, there're not very many greedy problems in LC. We can start from the classical ones, for example, Dijkstra and course schedule (The first example in Introduction to Algorithms third edition, Chapter 16).*

> ***Binary search*** *You need to find a good binary search template. And then the hard part becomes how to identify the problem is actually can be solved by binary search.*

I know it's a lot of concepts here. Luckily, most of them are related to each other. For example, you don't need to learn Divide and Conquer independently. You can almost always learn Divide and Conquer with Recursion together. And most of the algorithm concepts are related to data structures very much, you can try to learn them together. For example, how to visit a tree by DFS (Preorder visit) or in a BFS way. (Level order visit)

I believe if you can be familiar with all the concepts mentioned above. You can at least reach a 1800+ rating. And it's may enough for Amazon interview from the statistical data shown before!

How to practice for 1800 ~ 2200

If the main reason to do leetcode is for seeking new opportunities, I think it's already a good time to start. At this moment, you should already be familiar with most of the concepts mentioned above and should be comfortable with most of the medium problems. From my point of view, I think your algorithm skill should be good enough for most of the normal SED roles.

If you still want to pursue a breakthrough, then you may have to learn some "advanced" topics which may rarely be used in routine development. I think the 11 topics below may be helpful.

1. Difference Array
2. BIT Tree (Fenwick Tree)
3. Basic Math, such as the Sieve algorithm and inverse modular
4. Trie
5. Advance String Algorithms, such as KMP and Rolling Hash
6. Monotonic Stack
7. Sliding Window
8. Bit Manipulation
9. Union Set
10. Bitmask DP
11. Graph Algorithm, such as Floyd Algorithm, and Minimal Spanning Tree.

And of course, you have to practice more and be comfortable with most (if not any) of medium problems. At this point, you should be able to solve maybe half of the hard problems without looking into solutions. (I am not sure, I may be wrong here. And the difficulties of hard problems vary a lot). And you need to consider improving your typing speed if that's your bottleneck, I am not kidding.

How to practice for 2200+

Congratulation! If you can reach 2200+ in leetcode, you're already the top 2% players! It's a big achievement, isn't it?

To be honest, I am struggling at this range as well. I have stayed at the 2200+ rating for about 1 year and just reached 2350+ in very recent months. I don't think I can

provide useful suggestions for people already in this rating. But from my experience, I think if you still want a breakthrough. Then maybe practice in Leetcode is not enough. Currently, I start practice problems from Google Kickstart and Code Jam. I do know some people with 2700+ ratings. Most of them had some professional competitive programing training before. They're genius and focused, and some of them are top players in ICPC. I think if we want to improve our rating. We have to be passionate, we have to be focused, and we have to be confident. I hope I can reach 2600+ at the end of this year. I hope it's not too ambitious for me.

Some really good readings and resources

I think the below resources are helpful for anyone who treats Leetcode seriously. Some of the readings are actually for professional-level contests, don't be frightened if you found some materials are very hard to understand. I am struggling with these materials as well. LOL

1. Algorithms by Jeff Erickson
2. Competitive Programming The book covers most of the advanced topics in professional-level contests. I highly recommend this book.
3. The OI wiki Chinese only, but you can refer to #6 for English version.
4. Category list of each LC problem by wisdompeak
5. Rating list of each LC problem by zerotrac
6. Algorithms for Competitive Programming
7. Code force It's hard, but some blogs are super helpful. I didn't practice it very much.
8. Google KickStart I just started this one.

It's a long blog than I originally expected. Keep practicing, and coding is fun!!!