Algorithm learning links

https://github.com/andreis/interview

https://www.topcoder.com/community/data-science/data-science-tutorials/

https://github.com/prakhar1989/awesome-courses/blob/master/README.md

http://contest-wiki.csc.kth.se/index.php/How_to_get_better%3F

http://web.stanford.edu/~liszt90/acm/notebook.html

http://www.cs.princeton.edu/courses/archive/fall14/cos521/

http://www.problemclassifier.appspot.com/

http://www.quora.com/What-was-Anudeep-Nekkantis-Competitive-Programming-strategy-to-become-35th-in-Global-ranking-in-just-6-7-months

Top coder problem archive:http://community.topcoder.com/tc?module=ProblemArchive

Codechef: http://discuss.codechef.com/questions/48877/data-structures-and-algorithms-- good one

http://www.quora.com/How-should-I-practice-so-that-I-will-be-at-a-level-where-I-can-approach-TopCoders-Div1-500-problems-with-confidence

Read this:
http://norvig.com/21-days.html

System design interview questions
http://www.quora.com/How-should-I-prepare-system-design-questions-for-Google-Facebook-Interview

https://github.com/shashank88/system_design/blob/master/README.md

http://www.quora.com/Scalability/How-to-design-a-system-to-buffer-cache-frequent-write-operation-to-database

http://www.quora.com/How-will-you-design-a-cache-system

Distributed Systems
http://www.quora.com/What-are-the-good-resources-to-learn-about-distributed-scalable-robust-software-architecture-infrastructure-building

http://perspectives.mvdirona.com/2014/07/challenges-in-designing-at-scale-formal-methods-in-building-robust-distributed-systems/

http://the-paper-trail.org/blog/

http://merbist.com/2011/01/18/causality-of-scalability/
http://www.aosabook.org/en/distsys.html

http://book.mixu.net/distsys/single-page.html

https://courses.cs.washington.edu/courses/cse490h/07wi/

http://lpd.epfl.ch/site/education/da

Google interview preparation
http://courses.csail.mit.edu/iap/interview/materials.php
http://grouplens.org/blog/preparing-for-a-google-technical-interview/
https://www.google.com/about/careers/lifeatgoogle/hiringprocess/

http://matt.might.net/articles/what-cs-majors-should-know/
http://www.profshonle.com/2010/08/ten-things-every-computer-science-major.html

Algorithms solution
https://github.com/aistrate/AlgorithmsSedgewick
http://algs4.cs.princeton.edu/code/

Userids:

Hackerrank: subnr01@gmail.com
Spoj: subnr01
topcoder: subnr
codechef: subnr/subnr@gmail.com
codeforces:
project Euler:
http://www.daqwest.com/ --Facbook login
interviewbit - facebook login
Project Euler - subnr

If I am to start programming now, I would do it this way

1. Solve 200 most solved problems on SPOJ, Problem by problem. In 2 months.
2. *(This will teach all standard problems, algorithms and implementation skills)*
3.
4. Solve problems from CodeChef and CodeForces for 2 months.
5. *(This will teach variations, we can read others solutions and learn better ways. Skip easy problems)*
6.
7. Solve problems from TopCoder for 2 months.
8. *(This will teach Dynamic Programming. Div 1 500p)*
9. Check past ACM ICPC Regional's Problems
10. *(Great quality problems)*

https://michaelochurch.wordpress.com/

GOLANG
http://www.quora.com/What-are-the-best-free-sources-to-learn-Go-programming-language

Algorithm notes
http://www.ugrad.cs.ubc.ca/~cs490/sec202/notes.html

Programming channel
https://www.youtube.com/user/gtprogrammingteam/videos

===advice===

https://www.quora.com/What-is-the-best-strategy-to-improve-my-skills-in-competitive-programming-in-2-3-months?redirected_qid=4281068

Knowledge in theory of algorithms and data structures

I started practicing for competitive programming with my teacher in school, the first few things we learnt were:

- Data Structures: Topcoder tutorial
- Binary Search: Topcoder tutorial
- Sorting algorithms: Wikipedia list of sorting algorithms (They usually teach a few of them like these in this order: Bubble sort, Insertion sort, Merge sort, Heapsort, Quicksort and Bucket sort, a bit different. Look at visualizations too, like at sorting.at they are cool.)
- Greedy algorithms: Quora: What is an intuitive explanation of greedy algorithms?
- Backtracking: GeeksforGeeks: The Knight's tour problem, Sada Kurapati: N Queens problem
- Dynamic programming: Function Space: Fibonacci series and Dynamic programming (And I really like Jonathan Paulson's answer here.)
- Graph theory and some algorithms: Computer Science Source: Depth/Breadth First Search and Youtube video: Dijkstra's algorithmwere the first for me.

I think that's basically all I knew when I first competed. Probably other people will tell you a lot of other things, this is just how I started out.

Later, you can find a book that works for you and you can read it or watch an online course to get into the topic more deeply.

Here are some examples:

Quora: Computer Programming: What are the best books on Competitive programming out there?

Quora: What is the best set of algorithms books to read?

Topcoder Data Science Tutorials

Youtube playlist: MIT 6.006 Introduction to Algorithms, Fall 2011

Coursera: Princeton Algorithms course

Coursera: Stanford Algorithms course

Khan Academy course with Darthmouth college

A programming language

I like C++. It is fast, it has its Standard Template Library with plenty of cool stuff. For example if you need a good sorting algorithm you can just include the algorithm library and use one function. It is really useful because you don't want to waste your time on a competition to implement basic things like data structures and basic algorithms.

Some tutorials on STL:

Topcoder: Power up C++ with the Standard Template Library: Part 1

YoLinux: C++ STL Tutorial and Books at the end

And the reference site I use: cplusplus.com

Look up some Containers like vector, list, stack, queue and the sortalgorithm I talked about.

Some argue that C/C++ are the only reasonable programming languages for competitive programming. I would say if you don't really like C, go for a high level language. It's better to be comfortable with the platform than to be miserable with C. A lot of good competitors use Java, and they say it is not a drawback for them. Look up similar libraries for your choice of language and try using them!