**Graph Algorithms**

1. [Breadth First Search (BFS)](http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/)
2. [Depth First Search (DFS)](http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/)
3. [Shortest Path from source to all vertices \*\*Dijkstra\*\*](http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/)
4. [Shortest Path from every vertex to every other vertex \*\*Floyd Warshall\*\*](http://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/)
5. [Minimum Spanning tree \*\*Prim\*\*](http://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/)
6. [Minimum Spanning tree \*\*Kruskal\*\*](http://www.geeksforgeeks.org/greedy-algorithms-set-2-kruskals-minimum-spanning-tree-mst/)
7. [Topological Sort](http://www.geeksforgeeks.org/topological-sorting/)
8. [Johnson’s algorithm](http://www.geeksforgeeks.org/johnsons-algorithm/)
9. [Articulation Points (or Cut Vertices) in a Graph](http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/)
10. [Bridges in a graph](http://www.geeksforgeeks.org/bridge-in-a-graph/)

[All Graph Algorithms](http://www.geeksforgeeks.org/category/graph/)

**Dynamic Programming**

1. [Longest Common Subsequence](http://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/)
2. [Longest Increasing Subsequence](http://www.geeksforgeeks.org/dynamic-programming-set-3-longest-increasing-subsequence/)
3. [Edit Distance](http://www.geeksforgeeks.org/dynamic-programming-set-5-edit-distance/)
4. [Minimum Partition](http://www.geeksforgeeks.org/partition-a-set-into-two-subsets-such-that-the-difference-of-subset-sums-is-minimum/)
5. [Ways to Cover a Distance](http://www.geeksforgeeks.org/count-number-of-ways-to-cover-a-distance/)
6. [Longest Path In Matrix](http://www.geeksforgeeks.org/find-the-longest-path-in-a-matrix-with-given-constraints/)
7. [Subset Sum Problem](http://www.geeksforgeeks.org/dynamic-programming-subset-sum-problem/)
8. [Optimal Strategy for a Game](http://www.geeksforgeeks.org/dynamic-programming-set-31-optimal-strategy-for-a-game/)
9. [0-1 Knapsack Problem](http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/)
10. [Assembly Line Scheduling](http://www.geeksforgeeks.org/dynamic-programming-set-34-assembly-line-scheduling/)

**Searching And Sorting**

1. [Binary Search](http://geeksquiz.com/binary-search/)
2. [Quick Sort](http://geeksquiz.com/quick-sort/)
3. [Merge Sort](http://geeksquiz.com/merge-sort/)
4. [Order Statistics](http://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-2-expected-linear-time/)
5. [KMP algorithm](http://www.geeksforgeeks.org/searching-for-patterns-set-2-kmp-algorithm/)
6. [Rabin karp](http://www.geeksforgeeks.org/searching-for-patterns-set-3-rabin-karp-algorithm/)
7. [Z’s algorithm](http://www.geeksforgeeks.org/z-algorithm-linear-time-pattern-searching-algorithm/)
8. [Aho Corasick String Matching](http://www.geeksforgeeks.org/aho-corasick-algorithm-pattern-searching/)
9. [Counting Sort](http://www.geeksforgeeks.org/counting-sort/)
10. Manacher’s algorithm: [Part 1](http://www.geeksforgeeks.org/manachers-algorithm-linear-time-longest-palindromic-substring-part-1/), [Part 2](http://www.geeksforgeeks.org/manachers-algorithm-linear-time-longest-palindromic-substring-part-2/) and [Part 3](http://www.geeksforgeeks.org/manachers-algorithm-linear-time-longest-palindromic-substring-part-3-2/)

**Number theory and Other Mathematical**

**Prime Numbers and Prime Factorization**

1. [Primality Test | Set 1 (Introduction and School Method)](http://www.geeksforgeeks.org/primality-test-set-1-introduction-and-school-method/)
2. [Primality Test | Set 2 (Fermat Method)](http://www.geeksforgeeks.org/primality-test-set-2-fermet-method/)
3. [Primality Test | Set 3 (Miller–Rabin)](http://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/)
4. [Sieve of Eratosthenes](http://www.geeksforgeeks.org/sieve-of-eratosthenes/)
5. [Segmented Sieve](http://www.geeksforgeeks.org/segmented-sieve/)
6. [Wilson’s Theorem](http://www.geeksforgeeks.org/wilsons-theorem/)
7. [Prime Factorisation](http://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/)
8. [Pollard’s rho algorithm](http://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/)

**Modulo Arithmetic Algorithms**

1. [Basic and Extended Euclidean algorithms](http://www.geeksforgeeks.org/basic-and-extended-euclidean-algorithms/)
2. [Euler’s Totient Function](http://www.geeksforgeeks.org/eulers-totient-function/)
3. [Modular Exponentiation](http://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/)
4. [Modular Multiplicative Inverse](http://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/)
5. [Chinese remainder theorem Introduction](http://www.geeksforgeeks.org/chinese-remainder-theorem-set-1-introduction/)
6. [Chinese remainder theorem and Modulo Inverse Implementation](http://www.geeksforgeeks.org/chinese-remainder-theorem-set-2-implementation/)
7. [nCr%m](http://www.geeksforgeeks.org/compute-ncr-p-set-2-lucas-theorem/)

**Miscellaneous:**

1. [Counting Inversions](http://www.geeksforgeeks.org/counting-inversions/)
2. [Counting Inversions using BIT](http://www.geeksforgeeks.org/count-inversions-array-set-3-using-bit/)
3. [logarithmic exponentiation](http://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/)
4. [Square root of an integer](http://www.geeksforgeeks.org/square-root-of-an-integer/)
5. [Heavy light Decomposition](http://www.geeksforgeeks.org/heavy-light-decomposition-set-1-introduction/) , [this](http://e-maxx.ru/algo/heavy_light) and [this](http://blog.anudeep2011.com/heavy-light-decomposition/)
6. [Matrix Rank](http://www.geeksforgeeks.org/program-for-rank-of-matrix/)
7. [Gaussian Elimination to Solve Linear Equations](http://www.geeksforgeeks.org/gaussian-elimination/)
8. [Hungarian algorithm](https://en.wikipedia.org/wiki/Hungarian_algorithm)
9. [Link cut](http://www.cs.cmu.edu/~avrim/451f12/lectures/lect1009-linkcut.txt)
10. [Mo’s algorithm](http://www.geeksforgeeks.org/mos-algorithm-query-square-root-decomposition-set-1-introduction/) and [this](http://blog.anudeep2011.com/mos-algorithm/)
11. [Factorial of a large number in C++](http://www.geeksforgeeks.org/factorial-large-number/)
12. [Factorial of a large number in Java+](http://www.geeksforgeeks.org/biginteger-class-in-java/)
13. [Russian Peasant Multiplication](http://www.geeksforgeeks.org/fast-multiplication-method-without-using-multiplication-operator-russian-peasants-algorithm/)
14. [Catalan Number](http://www.geeksforgeeks.org/program-nth-catalan-number/)

**Geometrical and Network Flow Algorithms**

1. [Convex Hull](http://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/)
2. [Graham Scan](http://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/)
3. [Line Intersection](http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/)
4. [Interval Tree](http://www.geeksforgeeks.org/interval-tree/)
5. [Matrix Exponentiation](http://www.geeksforgeeks.org/matrix-exponentiation/) and [this](http://zobayer.blogspot.in/2010/11/matrix-exponentiation.html)
6. [Maxflow Ford Furkerson Algo and Edmond Karp Implementation](http://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/)
7. [Min cut](http://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/)
8. [Stable Marriage Problem](http://www.geeksforgeeks.org/stable-marriage-problem/)
9. [Hopcroft–Karp Algorithm for Maximum Matching](http://www.geeksforgeeks.org/hopcroft-karp-algorithm-for-maximum-matching-set-1-introduction/)
10. [Dinic’s algo](http://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/) and [e-maxx](http://e-maxx.ru/algo/dinic)

**Data Structures**

1. [Binary Indexed Tree or Fenwick tree](http://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/)
2. [Segment Tree](http://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/) ([RMQ](http://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/), [Range Sum](http://www.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/) and [Lazy Propagation](http://www.geeksforgeeks.org/lazy-propagation-in-segment-tree/))
3. [K-D tree](http://www.geeksforgeeks.org/k-dimensional-tree/) (See [insert](http://www.geeksforgeeks.org/k-dimensional-tree/), [minimum](http://www.geeksforgeeks.org/k-dimensional-tree-set-2-find-minimum/) and [delete](http://www.geeksforgeeks.org/k-dimensional-tree-set-3-delete/))
4. [Union Find Disjoint Set](http://www.geeksforgeeks.org/union-find/) ([Cycle Detection](http://www.geeksforgeeks.org/union-find-algorithm-set-2-union-by-rank/) and [By Rank and Path Compression](http://www.geeksforgeeks.org/union-find-algorithm-set-2-union-by-rank/))
5. [Tries](http://www.geeksforgeeks.org/trie-insert-and-search/)
6. Suffix array ([this](http://web.stanford.edu/class/cs97si/suffix-array.pdf), [this](http://www.geeksforgeeks.org/suffix-array-set-1-introduction/) and [this](http://www.geeksforgeeks.org/suffix-array-set-2-a-nlognlogn-algorithm/))
7. [Sparse table](http://www.geeksforgeeks.org/range-minimum-query-for-static-array/)
8. [Suffix automata](http://www.geeksforgeeks.org/searching-for-patterns-set-5-finite-automata/)
9. [Suffix automata II](http://www.geeksforgeeks.org/pattern-searching-set-5-efficient-constructtion-of-finite-automata/)
10. [LCA and RMQ](http://www.geeksforgeeks.org/find-lca-in-binary-tree-using-rmq/)

* Sieve of Eratosthenes, or another prime number sieve
* Depth-first search
* Breadth-first search
* Dijkstra's algorithm
* Floyd--Warshall algorithm
* Either Kruskal's or Prim's algorithm
* Some implementation of topological sorting, such as by using DFS
* Convex hull (I recommend the Monotone Chains algorithm)
* Coordinate compression
* Edmonds--Karp, or another implementation of the Ford--Fulkerson method; or a preflow-push algorithm; or, if you are preparing an ACM codebook, Dinic's algorithm. (Note: Max flow is not allowed to appear on the IOI, but may nevertheless appear on national team-selection contests)

1) Graph algorithms: Breadth first search(BFS), Depth first search(DFS), Strongly connected components(SCC), Dijkstra, Floyd-Warshall, Minimum spanning tree(MST), Topological sort.

2) Dynamic programming: Standard dynamic programming problems such as Rod Cutting, Knapsack, Matrix chain multiplication etc.

3) Number theory: Modular arithmetic, Fermat’s theorem, Chinese remainder theorem(CRT), Euclidian method for GCD, Logarithmic Exponentiation, Sieve of Eratosthenes, Euler’s totient function.

3) Greedy: Standard problems such as Activity selection.

4) Search techniques: Binary search, Ternary search and Meet in the middle.

5) Data structures (Basic): Stacks, Queues, Trees and Heaps.

6) Data structures (Advanced): Trie, Segment trees, Fenwick tree or Binary indexed tree(BIT), Disjoint data structures.

7) Strings: Knuth Morris Pratt(KMP), Z algorithm, Suffix arrays/Suffix trees. These are bit advanced algorithms.

8) Computational geometry: Graham-Scan for convex hull, Line sweep.

9) Game theory: Basic principles of Nim game, Grundy numbers, Sprague-Grundy theorem. The list is not complete but these are the ones that you encounter very frequently in the contests. There are other algorithms but are required very rarely in the contests.

Once you have sufficient knowledge of popular algorithms, you can start solving the medium level problems. That is Div 2 all problems in Topcoder and Codeforces. It is advisable not to go for Div 1 500 at this point.

Learning to code is all about practicing. Participate regularly in the programming contests. Solve the ones that you cannot solve in the contest, after the contest. Apart from Topcoder and Codeforces you can also look at HackerEarth Challenges or Codechef contests.

Read the codes of high rated programmers. Compare your solution with them. You can observe that it is simple and shorter than your solution. Analyse how they have approached and improve your implementation skills.

Read the editorials after the contest. You can learn how to solve the problems that you were not able to solve in the contest and learn alternative ways to solve the problems which you could solve.

Always practice the problems that you could solve in the contest. Suppose if you are able to solve Div 2 250 and 500 in the contest but not Div 2 1000 then practice as many Div 2 1000 problems as as you can. . Do not spend too much time if you are not getting the solution or are stuck somewhere. After you feel that you have spent enough time, look at the editorials. Understand the algorithm and code it. Do not look at the actual solution before you have attempted to write the code on your own.

Programming is a very practical and hands on skill. You have to continuously do it to be good at it. It’s not enough to solve the problem theoretically, you have to code it and get the solution accepted. Knowing which algorithm/logic to use and implementing it are two different things. It takes both to be good at programming.

Programming learning phase is going to take a lot of time and the key is practicing regularly. It takes some time before you can attempt Div 1 500 and other tough problems. Do not give up on reading the editorials and implementing them, even if it takes many hours/days. Remember everything requires practice to master it. It takes considerable amount of time before you get good at it. You have to keep yourself motivated throughout. Forming a team and practicing is a good choice. Not giving up is the key here.

In the 20 days period of the month when there is no Long Challenge at Codechef, solve the problems at SPOJ according to most users solved. (2-3 hours per day). Few days before the Cook-off attempt any running contest(s) at Codeforces, or previous un-attempted contests just like you would when the contest were live.

In the Long Challenge at Codechef, you'll be able to solve 2 problems at ease. Then think and keep attempting the next problems. There might be cases when you get TLE, in those cases try to analyze your algorithm and think how can you further improve on that. If you're fed up, move on to next problem (next most solved problem) and attempt it as well. The idea is to give breaks to your mind, thinking strategy and solve more problems to open it up.

Within 2 months you'll find tremendous improvement and you'll be able to solve ~4-5 problems in the Long Challenge at Codechef.

Few tips though:

1. Don't think *"let's first study all the algorithms, then I will attempt the problems."*
2. Don't think *"let's study this algorithm and do few problems related to it."*

**NO!!** This is not going to improve your problem solving skill. In the first case, just because you are all the time into theory, you didn't practice code and you kept studying. In the second case, you already know what algorithm to apply for the problems, so you didn't trigger that part of mind which thinks *How to approach this problem!*

Solution: **Study problems wise and write code!**

That's why I suggested to do the SPOJ classical problems in the free time of every month, according to most users solved. Almost everyone in this phase might have gone through a basic instinct of the algorithms, so to exploit it to the fullest...just attempt problems, later when you study the algorithm again because you needed it to solve the problem then you will automatically be learning more and there will be an increase in *"let's implement it my way"* motive. I would suggest Stanford video lectures, for they provide a good basic idea about algorithms and running time analysis and also about the applications. The explanation is clear and even people with average math skills can understand the proofs and all. Otherwise there are many other sources as well.

Conclusion: Practice! Attempt a problem, study things(algorithms) that are required for that. Move on! Don't aim to be a top ranker in few days. Focus on learning things, and be patient and carry on hard work!

*Disclaimer:* This shouldn't be taken as a generalized fact. But I have experienced and observed that problem wise strategy does work indeed! So, you're free to accept/deny this strategy. For each his own, after all. But I will be glad if this write-up could help few people improve their skills.

1. Depth-First Search (Graph Search)  
2. Breadth-First Search (Graph Search)(Including Flood-Fill)  
3. Dijkstra's Algorithm (Shortest Path)  
4. Floyd-Warshall's Algorithm (Shortest Path)  
5. Bellman-Ford Algorithm (Shortest Path)  
6. Greedy Algorithms  
7. Dynamic Programming (including Knapsack problems)  
8. Recursion  
9. Minimum Spanning Trees  
10. Binary and Linear Search  
11. Some basic combinatorics and number theory.  
  
You'd also need some data structures:  
  
1. Stack  
2. Queue  
3. Hash Table  
4. Heap  
5. Bag (maybe. I haven't seen this used in programming competitions often)