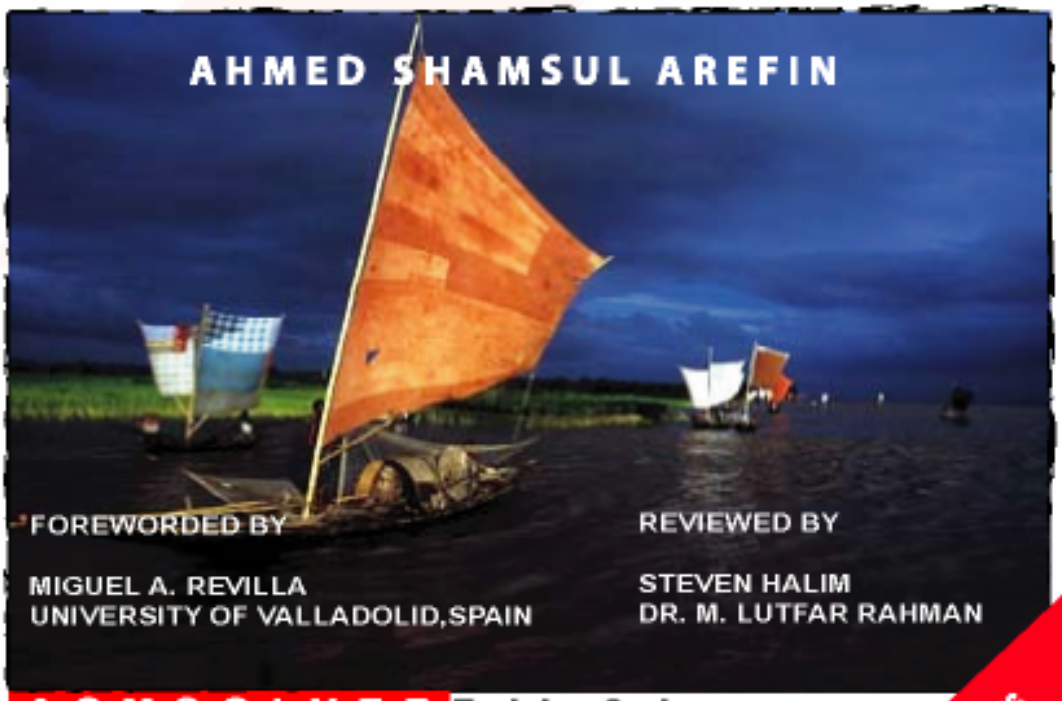
Special Online Edition for UVa Online Judge Users

Art of Programming Contest

C Programming | Data Structures | Algorithms

2nd Edition



AHMED SHAMSUL AREFIN

FOREWORDED BY

MIGUEL A. REVILLA
UNIVERSITY OF VALLADOLID,SPAIN

REVIEWED BY

STEVEN HALIM
DR. M. LUTFAR RAHMAN

ACMSOLVER Training Series

acm International Collegiate Programming Contest

An Essential Book for Programmers

ART OF PROGRAMMING CONTEST

C Programming Tutorials | Data Structures | Algorithms

Compiled by

Ahmed Shamsul Arefin
Graduate Student,
Institute of Information and Comunicaion Technology
Bangladesh University of Engineering and Technology (BUET)
BSc. in Computer Science and Engineering, CUET

Reviewed By
Steven Halim
School of Computing, National University of Singapore
Singapore.
Dr. M. Lutfar Rahman
Professor, Departent of Computer Science and Engineering
University of Dhaka.

Foreworded By
Professor Miguel A. Revilla
ACM-ICPC International Steering Committee Member and Problem Archivist
University of Valladolid,
Spain.
http://acmicpc-live-archive.uva.es
http://online-judge.uva.es

Gyankosh Prokashoni, Bangladesh

ISBN 984-32-3382-4

DEDICATED TO

Shahriar Manzoor
Judge ACM/ICPC World Finals 2003-2006
(Whose mails, posts and problems are invaluable to all programmers)

And
My loving parents and colleagues

ACKNOWLEDGEMENTS

I would like to thank following people for supporting me and helping me for the significant improvement of my humble works. Infact, this list is still incomplete.

Professor Miguel A. Revilla	*University of Valladolid, Spain.*
Dr. M Kaykobad	*North South University, Bangladesh*
Dr. M. Zafar Iqbal	*Shahjalal University of Science and Technology, Bangladesh*
Dr. M. Lutfar Rahman	*University of Dhaka, Bangladesh*
Dr. Abu Taher	*Daffodil International University*
Howard Cheng	*University of Lethbridge, Canada*
Steven Halim	*National University of Singapore, Singapore*
Shahriar Manzoor	*South East University, Bangladesh*
Carlos Marcelino Casas Cuadrado	*University of Valladolid, Spain*
Mahbub Murshed Suman	*Arizona State University, USA*
Salahuddin Mohammad Masum	*Daffodil International University*
Samiran Mahmud	*Dhaka University of Engineering and Technology*
M H Rasel	*Chittagong University of Engineering and Technology*
Sadiq M. Alam	*National University of Singapore, Singapore*
Mehedi Bakht	*Bangladesh University of Engineering and Technology*
Ahsan Raja Chowdhury	*University of Dhaka*
Mohammad Rubaiyat Ferdous Jewel	*University of Toronto, Canada*
KM Hasan	*North South University*
Monirul Islam Sharif	*Georgia Institute of Technology,USA*
Gahangir Hossain	*Chittagong University of Engineering and Technology*
S.M Saif Shams	*Shahjalal University of Science and Technology*
Shah Md. Shamsul Alam	*Daffodil International University*

Author's Biography: *Ahmed Shamsul Arefin is completing his Masters from* ***Bangladesh University of Engineering & Technology (BUET)*** *and has completed BSc. in Coputer Science and Eningeering from* ***CUET****. In Computer Science and Engineering . He participated in the 2001 ACM Regional Contest in Dhaka, and his team was ranked 10th. He became contest organizer at* ***Valladolid online judge*** *by arranging "Rockford Programming Contest 2001" and local Contest at several universities. His Programming Contest Training Website* ***"ACMSolver.org"*** *has been linked with ACM UVa , USU and Polish Online Judge – Sphere.*

His research interests are ***Contests, Algorithms, Graph Theory*** *and* ***Web-based applications****. His Contact E-mail* ***: asarefin@yahoo.com*** *Web:* ***http://www.daffodilvarsity.edu.bd/acmsolver/asarefin/***

Preface to 2nd Edition

I am happy to be able to introduce the 2nd Edition of this book to the readers. The objective of this edition is not only to assist the contestants during the contest hours but also describing the core subjects of **Computer Science** such as **C Programming**, **Data Structures** and **Algorithms**. This edition is an improvement to the previous edition. Few more programming techniques like **STL (Standard Template Library**), manipulating strings and handling mathematical functions are introduced here.

It is hoped that the new edition will be welcomed by all those for whom it is meant and this will become an **essential book for Computer Science students**.

Preface to 1st Edition

Why do programmers love Programming Contest? Because young computer programmers like to battle for fame, money, and they love algorithms. The first ACM-ICPC (International Collegiate Programming Contest) Asia Regional Contest Bangladesh was held at North South University in the year 1997. Except the year 2000, our country hosted this contest each year and our invaluable programmers have participated the world final every year from 1997.

Our performance in ACM/ICPC is boosting up day by day. The attention and time we are spending on solving moderate and difficult problems is noticeable. BUET, University of Dhaka, NSU and AIUB has produced many programmers who fought for World Finals. Institutions looking for boosting the performance of their teams in the programming contests may consider them as prospective coaches/trainers. Some universities have recently adopted another strategy. They are offering 1-credit courses for students interested in improving their problem-solving and programming skills.

I am very much grateful to our mentors, **Dr. M Kaykobad** who was honored with the "Best Coach" award in the World Finals in Honolulu. Under his dynamic presence our country teams became champion several times in the ACM/ICPC Asia Regional. **Dr. M. Zafar Iqbal**, Chief Judge of our ACM/ICPC Regional Contests. **Dr. Abul L Haque**, who first contacted **Dr. C.J. Hwang** (Asia Contests Director and Professor at Texas State University, San Marcos, USA) and wanted to have a n ACM/ICPC regional site at Dhaka back in 1997. Also a big thank should go to **Mr. Shahriar Manzoor**, our renown Problem Setter, Judging Director for ACM/ICPC Regional (Dhaka Site) and World Final Judge and Problem Setter. I would like to thank him personally because, he showed me the right way several times when I was setting problems for Valladolid Online Judge in "Rockford Programming Contest 2001" and while developing my Programming Contest Training Site "ACMSolver.org".

Thanks to **Professor Miguel A. Revilla**, University of Valladolid, Spain for linking my ACMSolver (http://www.acmsolver.org) site with his world famous Valladolid Online Judge (http://acm.uva.es/p) and making me ACM Valladolid Online Judge Algorithmic Team Member for helping them to add some problems at live archive.

And also invaluable thanks to **Steven Halim**, a PhD Student of NUS, Singapore for the permission of using his website (http://www.comp.nus.edu.sg/~stevenha/) contents. A major part of this book is compiled from his renowned website. Of course, it is mentionable that his website is based upon USACO Training page located at (http://ace.delos.com/)

I am grateful to **Daffodil International University**, especially to honorable Vice-Chancellor **Professor Aminul Islam** and Dean, Faculty of Science and Informaion Technology **Dr. M. Lutfar Rahman** and all my colleagues at **Department of Computer Science and Engineering** here, for providing me the golden opportunity of doing something on ACM Programming Contest and other researches.

Furthermore, since this project is a collection of tutorials from several sources so all the authors of tutorials are acknowledged in the **Reference** section of this book. Tracking down the original authors of some of these tutorials is much difficult. I have tried to identify case by case and in each case asked permission. I apologize in advance if there are any oversights. If so, please let me know so that I can mention the name in future edition.

Finally I would like to add a line at the end of this preface, for last few years while making and maintaining my site on ACM Programming Contest, I have got few experiences. I felt that there should be some guideline for beginners to enter into the world of programming. So, I started collecting tutorials and compiling them to my site. Furthermore, this is another attempt to make Programming Contest in our country, as I have tried to put all my collections in a printed form. Your suggestions will be cordially accepted.

Best regards,

Ahmed Shamsul Arefin.

UNIVERSIDAD DE VALLADOLID

Foreword Note

As the main resposible of the University of Valladolid Online Judge I has the feeling that this book is not only a recollection of tutorials as the author says in the preface, but also will be an essential part of the help sections of the UVa site, as it put together a lot of scattered information of the Online Judge, that may help to many programmers around the world, mainly to the newcomers, what is very important for us. The author proves a special interest in guiding the reader, and his tips must be considered almost as orders, as they are a result of a great experience as solver of problems as well as a problemsetter. Of course, the book is much more that an Online Judge user manual and contains very important information missing in our web, as the very interesting clasification of a lot of problems by categories, that analyze in detail and with examples. I think it is a book all our users should be allowed to access to, as is a perfect complement to our Online Judge.

Miguel A. Revilla
ACM-ICPC International Steering Committee Member and Problem Archivist
University of Valladolid, Spain.
http://acmicpc-live-archive.uva.es
http://online-judge.uva.es

University Of Dhaka 

Review Note

A Computer programming contest is a pleasurable event for the budding programmers, but only a few books are available as a training manual for programming competitions.

This book is designed to serve as a textbook for an algorithm course focusing on programming as well as a programming course focusing on algorithms. The book is specially designed to train students to participate in competitions such as the ACM International Collegiate Programming Contest.

The book covers several important topics related to the development of programming skills such as, fundamental concepts of contest, game plan for a contest, essential data structures for contest, Input/output techniques, brute force method, mathematics, sorting, searching, greedy algorithms, dynamic programming, graphs, computational geometry, Valladolid Online Judge problem category, selected ACM programming problems, common codes/routines for programming, Standard Template Library (STL), PC^2 contest administration and team guide.The book also lists some important websites/books for ACM/ICPC Programmers.

I believe that the book will be book will be of immense use for young programmers interested in taking part in programming competitions.

Dr. M. Lutfar Rahman
Professor, Department of Computer Science and Engineering (CSE)
University of Dhaka.
Bangladesh.

Notes from Steven Halim

When I created my own website World of Seven few years back (http://www.comp.nus.edu.sg/~stevenha), my aim was to promote understanding of data structures and algorithms especially in the context of programming contest and to motivate more programmers to be more competitive by giving a lot of hints for many University of Valladolid (UVa) Online Judge problems. However, due to my busyness, I never managed to set aside a time to properly publicize the content of my website in a book format. Thus, I am glad that Ahmed compiled this book and he got my permission to do so. Hopefully, this book will be beneficial for the programmers in general, but especially to the Bangladeshi programmers where this book will be sold.

Steven Halim
National University of Singapore (NUS)
Singapore.

Contents

acm International Collegiate Programming Contest

IBM event sponsor

What is the ACM Programming Contest?

The Association for Computing Machinery (ACM) sponsors a yearly programming contest, recently with the sponsorship of IBM. The contest is both well-known and highly regarded: last year 2400 teams competed from more than 100 nations competed at the regional levels. Sixty of these went on to the international finals. This contest is known as **ACM International Collegiate Programming Contest (ICPC).**

The regional contest itself is typically held in November, with the finals in March. Teams of three students use C, C++, or Java to solve six to eight problems within five hours. One machine is provided to each team, leaving one or two team members free to work out an approach. Often, deciding which problems to attack first is the most important skill in the contest. The problems test the identification of underlying algorithms as much as programming savvy and speed.

CHAPTER 1 FUNDAMENTAL CONCEPTS

Programming Contest is a delightful playground for the exploration of intelligence of programmers. To start solving problems in contests, first of all, you have to fix your aim. Some contestants want to increase the number of problems solved by them and the other contestants want to solve less problems but with more efficiency. Choose any of the two categories and then start. A contestant without any aim can never prosper in 24 hours online judge contests. So, think about your aim.[1]

If you are a beginner, first try to find the easier problems.Try to solve them within short time. At first, you may need more and more time to solve even simple problems. But do not be pessimistic. It is for your lack of practice. Try to solve easier problems as they increase your programming ability. Many beginners spend a lot of time for coding the program in a particular language but to be a great programmer you should not spend more times for coding, rather you should spend more time for debugging and thinking about the algorithm for the particular problem. A good programmer spends 10% time for coding and 45% time for thinking and the rest of the time for debugging. So to decrease the time for coding you should practice to solve easier problems first.

Do not try to use input file for input and even any output file for output when sending the program to online judges. All input and output parts should be done using standard input and outputs. If you are a C or C++ programmer try this, while coding and debugging for errors add the lines at the first line of the main procedure i.e.

```
#include <stdio.h>
main ()
{
freopen("FILE_NAME_FOR_INPUT","r",stdin);
freopen("FILE_NAME_FOR OUTPUT","w",stdout);
Rest of the codes…
return 0;}
```

But while sending to online judges remove the two lines with freopen to avoid restricted function errors. If you use the first freopen above, it will cause your program to take input from the file "FILE_NAME_FOR_INPUT". Write down the inputs in the file to avoid entering input several times for debugging. It saves a lot of time. But as the function opens input file which can be a cause of hacking the websites of online judges they don't allow using the function and if you use it they will give compilation error (Restricted Function). The second freopen is for generating the output of your program in a specified file named "FILE_NAME_FOR_OUTPUT" on the machine. It is very helpful when the output can't be justified just viewing the output window (Especially for String Manipulation Problem where even a single space character can be a cause of

Wrong answer). To learn about the function more check Microsoft Developer Network (MSDN Collection) and C programming helps.

Programming languages and dirty debugging

Most of the time a beginner faces this problem of deciding which programming language to be used to solve the problems. So, sometimes he uses such a programming language which he doesn't know deeply. That is why; he debugs for finding the faults for hour after hour and at last can understand that his problem is not in the algorithm, rather it is in the code written in that particular language. To avoid this, try to learn only one programming language very deeply and then to explore other flexible programming languages. The most commonly used languages are C, C++, PASCAL and JAVA. Java is the least used programming language among the other languages. Avoid dirty debugging.

Avoid Compilation Errors

The most common reply to the beginner from 24 hours online judge is COMPILATION ERROR (CE). The advices are,

1) When you use a function check the help and see whether it is available in Standard Form of the language. For example, do not use strrev function of string.h header file of C and C++ as it is not ANSI C, C++ standard. You should make the function manually if you need it. Code manually or avoid those functions that are available in your particular compiler but not in Standard Form of the languages.

2) Don't use input and output file for your program. Take all the inputs for standard input and write all the outputs on standard output (normally on the console window). Check my previous topics.

3) Do not use conio.h header file in C or C++ as it is not available in Standard C and C++. Usually don't use any functions available in <conio.h> header file. It is the great cause of Compilation Error for the programmers that use Turbo C++ type compiler.

4) built-in functions and packages are not allowed for using in online judge.

5) Don't mail your program i.e. don't use yahoo, hotmail etc. for sending your program to judge as it is a complex method—write judge id, problems number etc. Rather use submit-system of the online judge for example, Submit page of Valladolid. Using the former will give you CE most of the time as they include there advertisements at the beginning and end of your program. So the judge can't recognize the extra characters concatenated in your sent code and gives you CE. About 90% CE we ever got is for this

reason. The mail system also breaks your programs into several lines causing Wrong Answer or Other Errors though your program was correct indeed.

There are many famous online judges that can judge your solution codes 24 hours. Some of them are:

- Valladolid OJ (http://acm.uva.es/p)
- Ural OJ (http://acm.timus.ru)
- Saratov OJ (http://acm.sgu.ru)
- ZJU OJ (http://acm.zju.edu.cn)
- Official ACM Live Archive (http://cii-judge.baylor.edu/)
- Peking University Online Judge (http://acm.pku.edu.cn/JudgeOnline/)
- Programming Challenges (http://www.programming-challenges.com)

Forget Efficiency and start solving easier problems

Sometimes, you may notice that many programmers solved many problems but they made very few submissions (they are geniuses!). At first, you may think that I should try to solve the problems as less try as possible. So, after solving a problem, you will not want to try it again with other algorithm (may be far far better than the previous algorithm you used to solve that problem) to update your rank in the rank lists. But my opinion is that if you think so you are in a wrong track. You should try other ways as in that and only that way you can know that which of the algorithms is better. Again in that way you will be able to know about various errors than can occur. If you don't submit, you can't know it. Perhaps a problem that you solved may be solved with less time in other way. So, my opinion is to try all the ways you know. In a word, if you are a beginner forget about efficiency.

Find the easier problems.Those problems are called ADHOC problems. You can find the list of those problems available in 24 OJ in S. Halim's, acmbeginner's, acmsolver's websites. Try to solve these problems and in that way you can increase your programming capability.

Learn algorithms

Most of the problems of Online Judges are dependent on various algorithms. An algorithm is a definite way to solve a particular problem. If you are now skilled in coding and solving easier problems, read the books of algorithms next. Of course, you should have a very good mathematical skill to understand various algorithms. Otherwise, there is no other way but just to skip the topics of the books. If you have skill in math, read the algorithms one by one, try to understand. Aft er understanding the algorithms, try to write it in the programming language you have learnt (This is because, most of the

algorithms are described in Pseudocode). If you can write it without any errors, try to find the problems related to the algorithm, try to solve them. There are many famous books of algorithms. Try to make modified algorithm from the given algorithms in the book to solve the problems.

Use simple algorithms, that are guaranteed to solve the problem in question, even if they are not the optimum or the most elegant solution. Use them even if they are the most stupid solution, provided they work and they are not exponential. You are not competing for algorithm elegance or efficiency. You just need a correct algorithm, and you need it now. The simplest the algorithm, the more the chances are that you will code it correctly with your first shot at it.

This is the most important tip to follow in a programming contest. You don't have the time to design complex algorithms once you have an algorithm that will do your job. Judging on the size of your input you can implement the stupidest of algorithms and have a working solution in no time. Don't underestimate today's CPUs. A for loop of 10 million repetitions will execute in no time. And even if it takes 2 or 3 seconds you needn't bother. You just want a program that will finish in a couple of seconds. Usually the timeout for solutions is set to 30 seconds or more. Experience shows that if your algorithm takes more than 10 seconds to finish then it is probably exponential and you should do something better.

Obviously this tip should not be followed when writing critical code that needs to be as optimized as possible. However in my few years of experience we have only come to meet such critical code in device drivers. We are talking about routines that will execute thousands of time per second. In such a case every instruction counts. Otherwise it is not worth the while spending 5 hours for a 50% improvement on a routine that takes 10 milliseconds to complete and is called whenever the user presses a button. Nobody will ever notice the difference. Only you will know.

Simple Coding

1. Avoid the usage of the ++ or -- operators inside expressions or function calls. Always use them in a separate instruction. If you do this there is no chance that you introduce an error due to post-increment or pre-increment. Remember it makes no difference to the output code produced.

2. Avoid expressions of the form *p++.

3. Avoid pointer arithmetic. Instead of (p+5) use p[5].

4. Never code like :

```
return (x*y)+Func(t)/(1-s);
```

but like :

```
temp = func(t);
RetVal = (x*y) + temp/(1-s);
return RetVal;
```

This way you can check with your debugger what was the return value of Func(t) and what will be the return code of your function.

5. Avoid using the = operator. Instead of :

```
return (((x*8-111)%7)>5) ? y : 8-x;
```

Rather use :

```
Temp = ((x*8-111)%7);     if (5<Temp) return y; else  return 8-x;
```

If you follow those rules then you eliminate all chances for trivial errors, and if you need to debug the code it will be much easier to do so.

· NAMING 1 : Don't use small and similar names for your variables. If you have three pointer variables don't name them p1, p2 and p3. Use descriptive names. Remember that your task is to write code that when you read it it says what it does. Using names like Index, RightMost, and Retries is much better than i, rm and rt. The time you waste by the extra typing is nothing compared to the gain of having a code that speaks for itself.

· NAMING 2 : Use hungarian naming, but to a certain extent. Even if you oppose it (which, whether you like it or not, is a sign of immaturity) it is of immense help.

· NAMING 3 : Don't use names like {i,j,k} for loop control variables. Use {I,K,M}. It is very easy to mistake a j for an i when you read code or "copy, paste & change" code, but there is no chance that you mistake I for K or M.

Last words

Practice makes a man perfect. So, try to solve more and more problems. A genius can't be built in a day. It is you who may be one of the first ten of the rank lists after someday. So, get a pc, install a programming language and start solving problem at once.[1]

CHAPTER 2 GAME PLAN FOR A CONTEST

During a real time contest, teams consisting of three students and one computer are to solve as many of the given problems as possible within 5 hours. The team with the most problems solved wins, where ``solved'' means producing the right outputs for a set of (secret) test inputs. Though the individual skills of the team members are important, in order to be a top team it is necessary to make use of synergy within the team.[2]

However, to make full use of a strategy, it is also important that your individual skills are as honed as possible. You do not have to be a genius as practicing can take you quite far. In our philosophy, there are three factors crucial for being a good programming team:

- Knowledge of standard algorithms and the ability to find an appropriate algorithm for every problem in the set;
- Ability to code an algorithm into a working program; and
- Having a strategy of cooperation with your teammates.

What is an Algorithm?

"A good algorithm is like a sharp knife - it does exactly what it is supposed to do with a minimum amount of applied effort. Using the wrong algorithm to solve a problem is trying to cut a steak with a screwdriver: you may eventually get a digestible result, but you will expend considerable more effort than necessary, and the result is unlikely to be aesthetically pleasing."

Algorithm is a step-by-step sequence of instructions for the computer to follow.

To be able to do something competitive in programming contests, you need to know a lot of well-known algorithms and ability to identify which algorithms is suitable for a particular problem (if the problem is straightforward), or which combinations or variants of algorithms (if the problem is a bit more complex).

A good and correct algorithm according to the judges in programming contests:

1. Must terminate
   [otherwise: Time Limit/Memory Limit/Output Limit Exceeded will be given]
2. When it terminate, it must produce a correct output
   [otherwise: the famous Wrong Answer reply will be given]
3. It should be as efficient as possible
   [otherwise: Time Limit Exceeded will be given]

Ability to quickly identify problem types

In all programming contests, there are only three types of problems:
1. I haven't see this one before
2. I have seen this type before, but haven't or can't solve it
3. I have solve this type before

In programming contests, you will be dealing with a set of problems, not only one problem. The ability to quickly identify problems into the above mentioned contest-classifications (haven't see, have seen, have solved) will be one of key factor to do well in programming contests.

Mathematics	Prime Number
	Big Integer
	Permutation
	Number Theory
	Factorial
	Fibonacci
	Sequences
	Modulus
Dynmic Programming	Longest Common Subsequence
	Longest Increasing Subsequence
	Edit Distance
	0/1 Knapsack
	Coin Change
	Matrix Chain Multiplication
	Max Interval Sum
Graph	Traversal
	Flood Fill
	Floyed Warshal
	MST
	Max Bipertite Matching
	Network Flow
	Aritculation Point
Sorting	Bubble Sort
	Quick Sort
	Merge Sort (DAndC)
	Selection Sort
	Radix Sort
	Bucket Sort
Searching	Complete Search, Brute Force
	Binary Search (DAndC)
	BST
Simulation	Josephus
String Processing	String Matching
	Pattern Matching
Computational Geometry	Convex Hull
AdHoc	Trivial Problems

Sometimes, the algorithm may be 'nested' inside a loop of another algorithm. Such as binary search inside a DP algorithm, making the problem type identification not so trivial.

If you want to be able to compete well in programming contests, you must be able to know all that we listed above, with some precautions to ad-hoc problems.

'Ad Hoc' problems are those whose algorithms do not fall into standard categories with well-studied solutions. Each Ad Hoc problem is different... No specific or general techniques exist to solve them. This makes the problems the 'fun' ones (and sometimes frustrating), since each one presents a new challenge.

The solutions might require a novel data structure or an unusual set of loops or conditionals. Sometimes they require special combinations that are rare or at least rarely encountered. It usually requires careful problem description reading and usually yield to an attack that revolves around carefully sequencing the instructions given in the problem. Ad Hoc problems can still require reasonable optimizations and at least a degree of analysis that enables one to avoid loops nested five deep, for example.

Ability to analyze your algorithm

You have identified your problem. You think you know how to solve it. The question that you must ask now is simple: Given the maximum input bound (usually given in problem description), can my algorithm, with the complexity that I can compute, pass the time limit given in the programming contest.

Usually, there are more than one way to solve a problem. However, some of them may be incorrect and some of them is not fast enough. However, the rule of thumb is: **Brainstorm many possible algorithms - then pick the stupidest that works!**

Things to learn in algorithm

1. Proof of algorithm correctness (especially for Greedy algorithms)
2. Time/Space complexity analysis for non recursive algorithms.
3. For recursive algorithms, the knowledge of computing recurrence relations and analyze them: iterative method, substitution method, recursion tree method and finally, Master Theorem
4. Analysis of randomized algorithm which involves probabilistic knowledge, e.g: Random variable, Expectation, etc.
5. Amortized analysis.
6. Output-sensitive analysis, to analyze algorithm which depends on output size, example: O(n log k) LIS algorithm, which depends on k, which is output size not input size.

Table 1: Time comparison of different order of growth
We assume our computer can compute 1000 elements in 1 seconds (1000 ms)

Order of Growth	n	Time (ms)	Comment
O(1)	1000	1	Excellent, almost impossible for most cases
O(log n)	1000	9.96	Very good, example: Binary Search
O(n)	1000	1000	Normal, Linear time algorithm
O(n log n)	1000	9960	Average, this is usually found in sorting algorithm such as Quick sort
O(n^2)	1000	1000000	Slow
O(n^3)	1000	10^9	Slow, btw, All Pairs Shortest Paths algorithm: Floyd Warshall, is O(N^3)
O(2^n)	1000	2^1000	Poor, exponential growth... try to avoid this. Use Dynamic Programming if you can.
O(n!)	1000	uncountable	Typical NP-Complete problems.

Table 2: Limit of maximum input size under 60 seconds time limit (with assumptions)

Order of Growth	Time (ms)	Max Possible n	Comment
O(1)	60.000 ms	Virtually infinite	Best
O(log n)	60.000 ms	6^18.000	A very very large number
O(n)	60.000 ms	60.000	Still a very big number
O(n log n)	60.000 ms	~ 5.000	Moderate, average real life size
O(n^2)	60.000 ms	244	small
O(n^3)	60.000 ms	39	very small
O(2^n)	60.000 ms	16	avoid if you can
O(n!)	60.000 ms	8	extremely too small.

It may be useful to memorize the following ordering for quickly determine which algorithm perform better asymptotically: **constant < log n < n < n log n < n^2 < n^3 < 2^n < n!**

Some rules of thumb

1. Biggest built in data structure "long long" is 2^63-1: 9*10^18 (up to 18 digits)
2. If you have k nested loops running about n iterations each, the program has O(n*k) complexity
3. If your program is recursive with b recursive calls per level and has l levels, the program O(b*l) complexity
4. Bear in mind that there are n! permutations and 2^n subsets or combinations of n elements when dealing with those kinds of algorithms