

CS 91 USACO

Bronze Division

Unit 1: Introduction to Competitive Programming



LECTURE 1: INTRODUCTION

DR. ERIC CHOU

IEEE SENIOR MEMBER



Objectives

- Creation of USACO Gateway and USACO competition account.
- Tools and Textbooks.
- Sample Submission for Gateway Training Site.
- Introduction to Competitive Programming.

Creation of Accounts

SECTION 1



Accounts

- CS 25 Java Data Structure and Algorithms
- CS 91 USACO Bronze
- Coding Bat Completion
- USACO Training site: gateway
- USACO web-site
- USACO Guide



WELCOME TO THE USACO TRAINING PROGRAM GATEWAY

2022.10.24 10:54:29

Please enter your correct UserName and Password in order to see your USACO Training Program curriculum.

[Register here for a username/password](#) if you do not already have one.

USACO
UserName

Forgot your USACO UserName? Enter your e-mail address and click FORGOT PASSWORD, below

Password

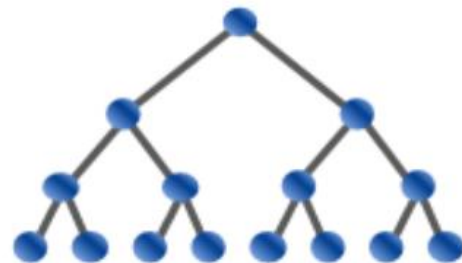
Forgot your password? Enter your USACO username or e-mail address above and click below to have it e-mailed to you.

[Listen to this amusing MP3 file that explains it all for computer geeks.](#)

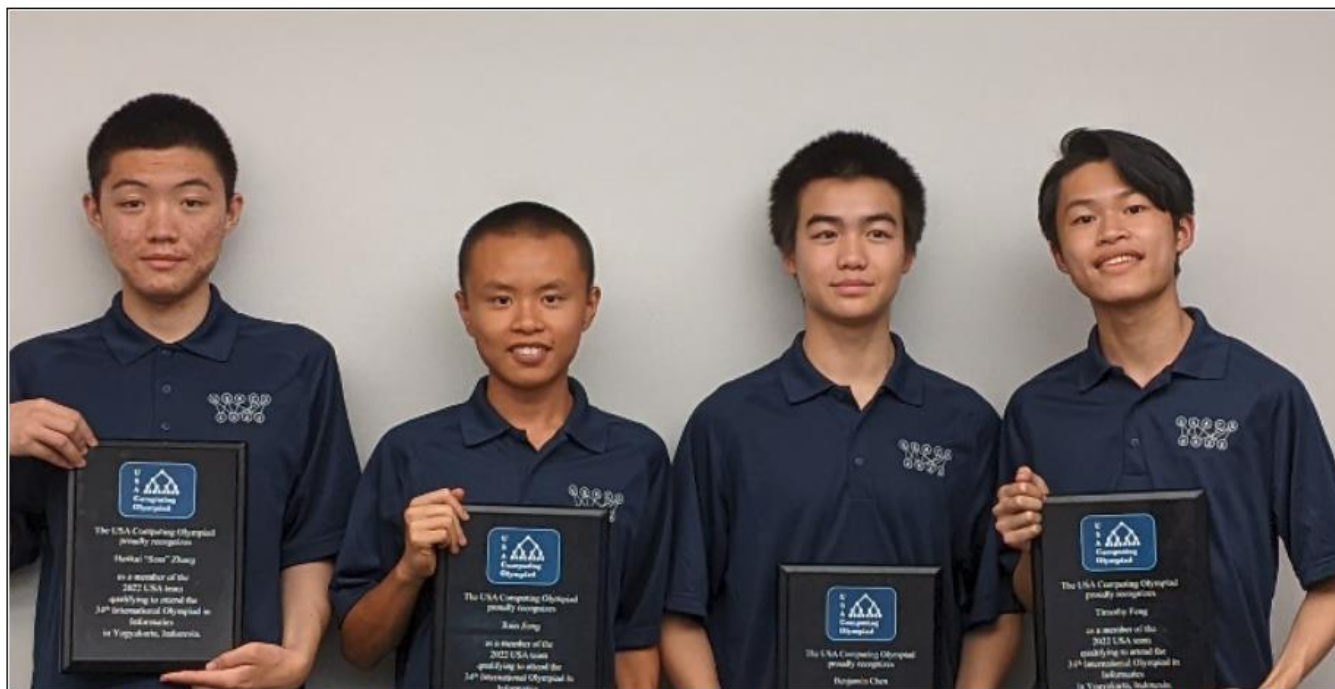
[USACO Home](#) : Web Contact rob.kolstad@gmail.com : Phone 719-481-6542 : [Privacy Policy](#)

Copyright ©2020 Rob Kolstad, All rights reserved. : 2022.10.24 10:54:29

USA Computing Olympiad

[OVERVIEW](#)[TRAINING](#)[CONTESTS](#)[HISTORY](#)[STAFF](#)[RESOURCES](#)

2022 IOI TEAM ANNOUNCED



Congratulations to Hankai (Sam) Zhang, Rain Jiang, Benjamin Chen, and Timothy Feng, who have been selected to represent the USA at the 2022 International Olympiad in Informatics!

YOUR ACCOUNT

Not currently logged in.

Username:

Password:

[Forgot password?](#)

Login

Register for New Account

2021-2022 SCHEDULE

Dec 17-20: First Contest

Jan 28-31: Second Contest

Feb 25-28: Third Contest

Mar 25-28: US Open

May 26-June 4: Training Camp

Aug 7-14: IOI 2022 in Indonesia



USACO Guide

A free collection of [curated, high-quality resources](#) to take you from Bronze to Platinum and beyond.

Get Started

Textbook and Tools

SECTION 2



Visual Studio Code



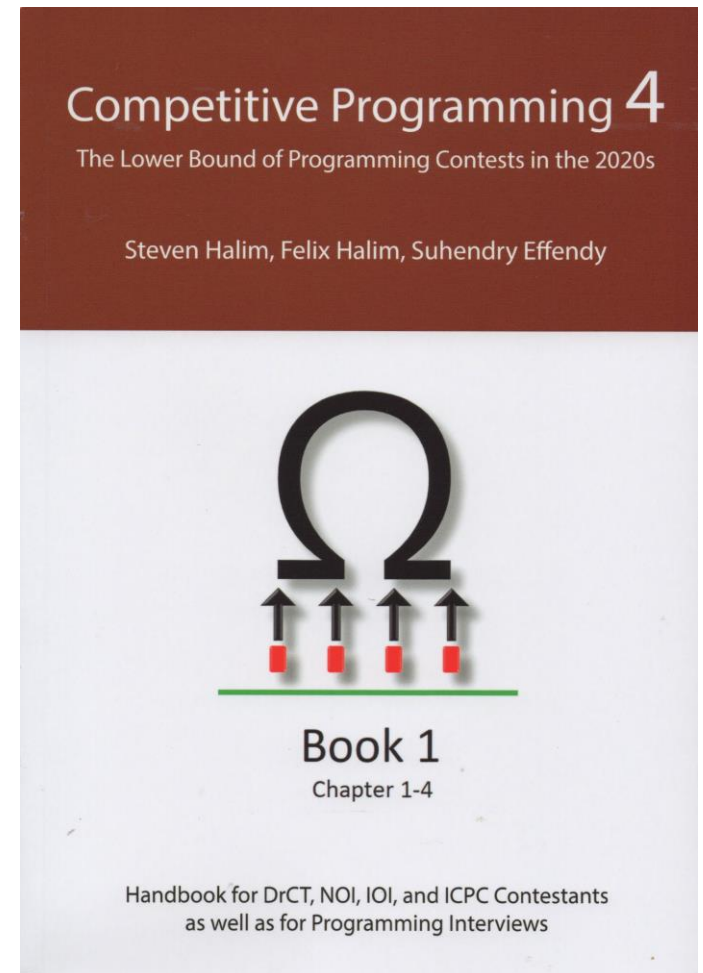
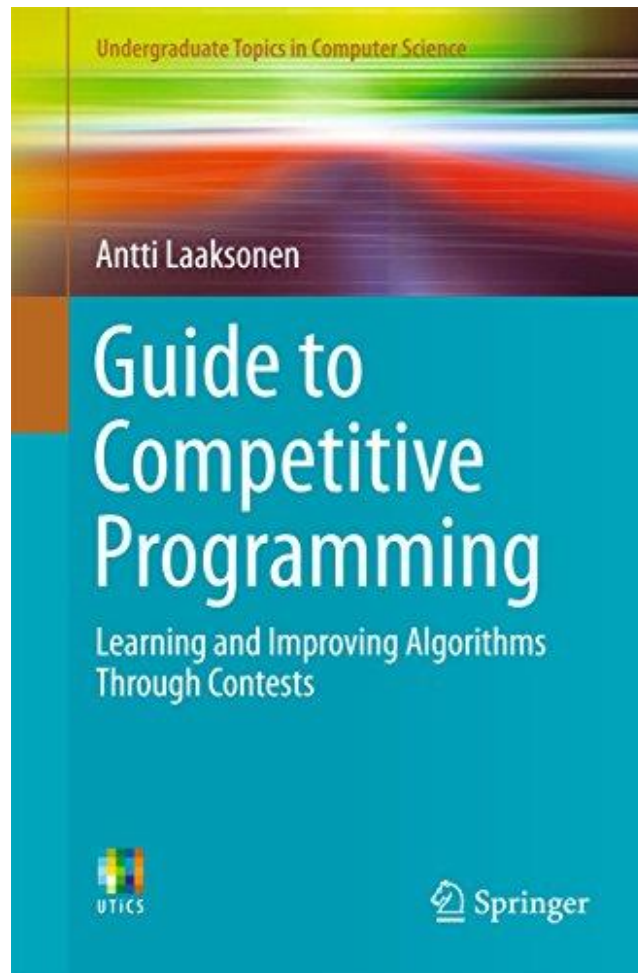
Bluej



University of
Kent

Supported by
ORACLE®

MinGW-w64



Test Submission

SECTION 3



Submitting Solutions

- The USACO Training Program features an automatic grading system for your homework problems. You submit your programs from the problem page itself; they are compiled and graded; the results are conveyed back to you -- all within a few seconds.
- C/C++/C++11/C++14, PASCAL, Python2, Python3, and Java are available. This system uses the GNU GCC compilation suite for C/C++/C++11 programs and the Free Pascal system for Pascal programs. The Java compiler is Oracle's most recent release.
- The grading system compilers are those formerly used at the IOI.
- These newer compilers use 32 bit int's; the Borland compilers use 16 bit int's.
DO NOT GET IN TROUBLE BECAUSE OF THIS!



Submitting Solutions

- Submit solutions via the web by typing the name of the file containing the source code into the 'Submit a file:' box at the bottom of problem description pages..
- Program submissions require simple **Header comments**: your ID (i.e., your USACO login name), the name of the program (which will be given in each programming assignment, and the language used. See the examples below to get the idea.
- Every training page problem has input and output. Currently, the input appears in a file named 'probname.in' (e.g., if the problem name is 'ride', then the input filename is 'ride.in'). Output must be written to a file named 'probname.out' (i.e., 'ride.out' for the 'ride' problem).



The First Challenge

ID: your_id_here

LANG: JAVA

TASK: test

*/

```
import java.io.*;
```

```
import java.util.*;
```

```
class test {
```

```
    public static void main (String [] args) throws IOException {
```

```
        // Use BufferedReader rather than RandomAccessFile; it's much faster
```

```
        BufferedReader f = new BufferedReader(new FileReader("test.in"));
```

```
                                // input file name goes above
```

```
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("test.out")));
```

```
        // Use StringTokenizer vs. readLine/split -- lots faster
```

```
        StringTokenizer st = new StringTokenizer(f.readLine());
```

```
                                // Get line, break into tokens
```

```
        int i1 = Integer.parseInt(st.nextToken());    // first integer
```

```
        int i2 = Integer.parseInt(st.nextToken());    // second integer
```

```
        out.println(i1+i2);                            // output result
```

```
        out.close();                                    // close the output file
```

```
    }
```

```
}
```




Submitting Solutions

- Guide every students to submit the test program.

USACO Grader Results x +

train.usaco.org/upload3



USACO Training Grader Results

9 users online
CHN/1 IND/2 USA/6

Upgrades in grading system: Please report problems to rob.kolstad@gmail.com

USER: Rob Kolstad [kolstad001]
TASK: test
LANG: C

Compiling...
Compile: OK

Executing...
Test 1: TEST OK [0.000 secs, 540 KB]
Test 2: TEST OK [0.000 secs, 540 KB]

All tests OK.

Your program ('test') produced all correct answers! This is your submission #336 for this problem. **Congratulations!**

Here are the test data inputs:

```
----- test 1 [length 4 bytes] ----  
1 1  
----- test 2 [length 4 bytes] ----  
3 9
```

Keep up the good work!
Thanks for your submission!

[Problem Statement for task test](#) | [USACO Gateway](#) | [Comment or Question](#)

Submit a file: No file chosen

Gateway: Your ride is Here (ride)

SECTION 4



Problem Statement

- It is a well-known fact that behind every good comet is a UFO. These UFOs often come to collect loyal supporters from here on Earth. Unfortunately, they only have room to pick up one group of followers on each trip. They do, however, let the groups know ahead of time which will be picked up for each comet by a clever scheme: they pick a name for the comet which, along with the name of the group, can be used to determine if it is a particular group's turn to go (who do you think names the comets?).
- The details of the matching scheme are given below; your job is to write a program which takes the names of a group and a comet and then determines whether the group should go with the UFO behind that comet.



Problem Statement

- Both the name of the group and the name of the comet are converted into a number in the following manner: the final number is just the product of all the letters in the name, where "A" is 1 and "Z" is 26. For instance, the group "USACO" would be $21 * 19 * 1 * 3 * 15 = 17955$. If the group's number mod 47 is the same as the comet's number mod 47, then you need to tell the group to get ready! (Remember that "a mod b" is the remainder left over after dividing a by b; $34 \bmod 10$ is 4.)
- Write a program which reads in the name of the comet and the name of the group and figures out whether according to the above scheme the names are a match, printing "GO" if they match and "STAY" if not. The names of the groups and the comets will be a string of capital letters with no spaces or punctuation, up to 6 characters long.



Examples:

Input	Output
COMETQ HVNGAT	GO
ABSTAR USACO	STAY



INPUT FORMAT (file ride.in):

- Line 1: An upper case character string of length 1..6 that is the name of the comet.
- Line 2: An upper case character string of length 1..6 that is the name of the group.
- NOTE: The input file has a newline at the end of each line but does not have a "return". Sometimes, programmers code for the Windows paradigm of "return" followed by "newline"; don't do that! Use simple input routines like "readln" (for Pascal) and, for C/C++, "fscanf" and "fid>>string".
- NOTE 2: Because of the extra characters, be sure to leave enough room for a 'newline' (also notated as '\n') and an end of string character ('\0') if your language uses it (as C and C++ do). This means you need eight characters of room instead of six.

SAMPLE INPUT:

```
COMETQ  
HVNGAT
```




OUTPUT FORMAT (ride.out):

- A single line containing either the word "GO" or the word "STAY".

SAMPLE OUTPUT:

GO

Converting the letters to numbers:

C	O	M	E	T	Q
3	15	13	5	20	17
H	V	N	G	A	T
8	22	14	7	1	20

then calculate the product mod 47:

$$3 * 15 * 13 * 5 * 20 * 17 = 994500 \text{ mod } 47 = 27$$
$$8 * 22 * 14 * 7 * 1 * 20 = 344960 \text{ mod } 47 = 27$$

Because both products evaluate to 27 (when modded by 47), the mission is 'GO'.



Nature of the Problem

- Traversal with Filter
- Accumulative Product
- Modulo Arithmetic (Avoid BigInteger)



Submitting Solutions

- Guide every students to submit the ride program.

Introduction to Competitive Programming

SECTION 5



Problem Solving/Practice Guidelines

- Anatomy of a Programming Contest Problem
- Typical Input/Output Routines
- The Journey of Competitive Programming (Start from Easy to Hard)
 - Super Easy
 - Easy
 - Medium
 - Hard
 - Super Hard
- Ad hoc Problems



What are different from AP CS and Competitive Programming?

- No limitation on the AP Subset.
- Any function, library, API are allowed. (`Arrays.sort`)
- Time limits.
- Problem size is way bigger.
- Corner cases. (more important) Need to be complete and pass all test data.

The Journey of Competitive Programming

SECTION 6



Uncompetitive programmer A

The Blurry One

Step 1: Reads the problem and becomes confused. (This problem is new for him).

Step 2: Tries to code something: Reading the non-trivial input and output.

Step 3: Realizes that all his attempts are not Accepted (AC):

- Greedy (Section 3.4): Repeatedly pairing the two remaining students with the shortest separating distances gives the Wrong Answer (WA).
- Naive Complete Search: Using recursive backtracking (Section 3.2) and trying all possible pairings yields Time Limit Exceeded (TLE).



Uncompetitive programmer B

Give Up

Step 1: Reads the problem and realizes that he has seen this problem before.

But also remembers that he has not learned how to solve this kind of problem...

He is not aware of the Dynamic Programming (DP) solution (Section 3.5)...

Step 2: Skips the problem and reads another problem in the problem set.



Uncompetitive programmer C

Slow

Step 1: Reads the problem and realizes that it is a hard problem: 'minimum weight perfect matching on a small general weighted graph'. However, since the input size is small, this problem is solvable using DP. The DP state is a bitmask that describes a matching status, and matching unmatched students i and j will turn on two bits i and j in the bitmask (Section 8.3.1).

Step 2: Codes I/O routine, writes recursive top-down DP, tests, debugs $>.<...$

Step 3: After 3 hours, his solution obtains AC (passed all the secret test data).



Competitive programmer D

- Completes all the steps taken by uncompetitive programmer C in ≤ 30 minutes.



Very competitive programmer E

- A very competitive programmer (e.g. the red 'target' coders in TopCoder [32]) would solve this 'well known' problem ≤ 15 minutes...



Training Goals

- Start from simple problems.
- Move from A/B type programmer to C, D, then E.
One problem type after a problem type.
- Study standard design styles.

Design Example and Tips

SECTION 7



First Design Pattern: Big Factorial

Demo Program: `BigFactorial.java`

Try to calculate $n!$ (when n is large).

Design Pattern: Linear Traversal.

Design Issues:

- (1) Library support.
- (2) Easiness for debugging.
- (3) Short learning curve.



BigFactorial.java

```
1 import java.util.Scanner;
2 import java.math.BigInteger;
3
4 class BigFactorial{                                // :
5     public static void main(String[] args) {
6         BigInteger fac = BigInteger.ONE;
7         for (int i = 2; i <= 25; i++)
8             fac = fac.multiply(BigInteger.valueOf(i));
9         System.out.println(fac);
10    }
11 }
```




Knowledge

- Constant: `BigInteger.ONE`
- `BigInteger` Class extends `Number` // Java Abstract Class, similar to `Integer`, `BigInteger` is sortable.
- Java is slower, but Java has powerful built-in libraries and APIs such as **`BigInteger`**/**`BigDecimal`**, **`GregorianCalendar`**, **`Regex`**, etc.
- Java programs are easier to debug with the virtual machine's ability to provide a stack trace when it crashes (as opposed to core dumps or segmentation faults in C/C++). On the other hand, C/C++ has its own merits as well. Depending on the problem at hand, either language may be the better choice for implementing a solution in the shortest time.
- BlueJ, Eclipse, or JetBrains



Tips 1: Type Faster

- No kidding! Although this tip may not mean much as ICPC and (especially) IOI are not typing contests, we have seen Rank i and Rank $i + 1$ ICPC teams separated only by a few minutes and frustrated IOI contestants who miss out on salvaging important marks by not being able to code a last-minute brute force solution properly.
- When you can solve the same number of problems as your competitor, it will then be down to coding skill (your ability to produce concise and robust code) and ... typing speed.



Golden Ratio

Diagnostics Problem

$$1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}}$$

- Write an iterative function that can reach the n division levels. This function returns the golden ratio to the n division level.

static BigDecimal goldenRatio(int n);

- Write a recursive function that can also reach the n division level.

static BigDecimal goldenRatio(int n, BigDecimal g);

- Compare the two algorithms.
- **Hint:** use BigDecimal class.

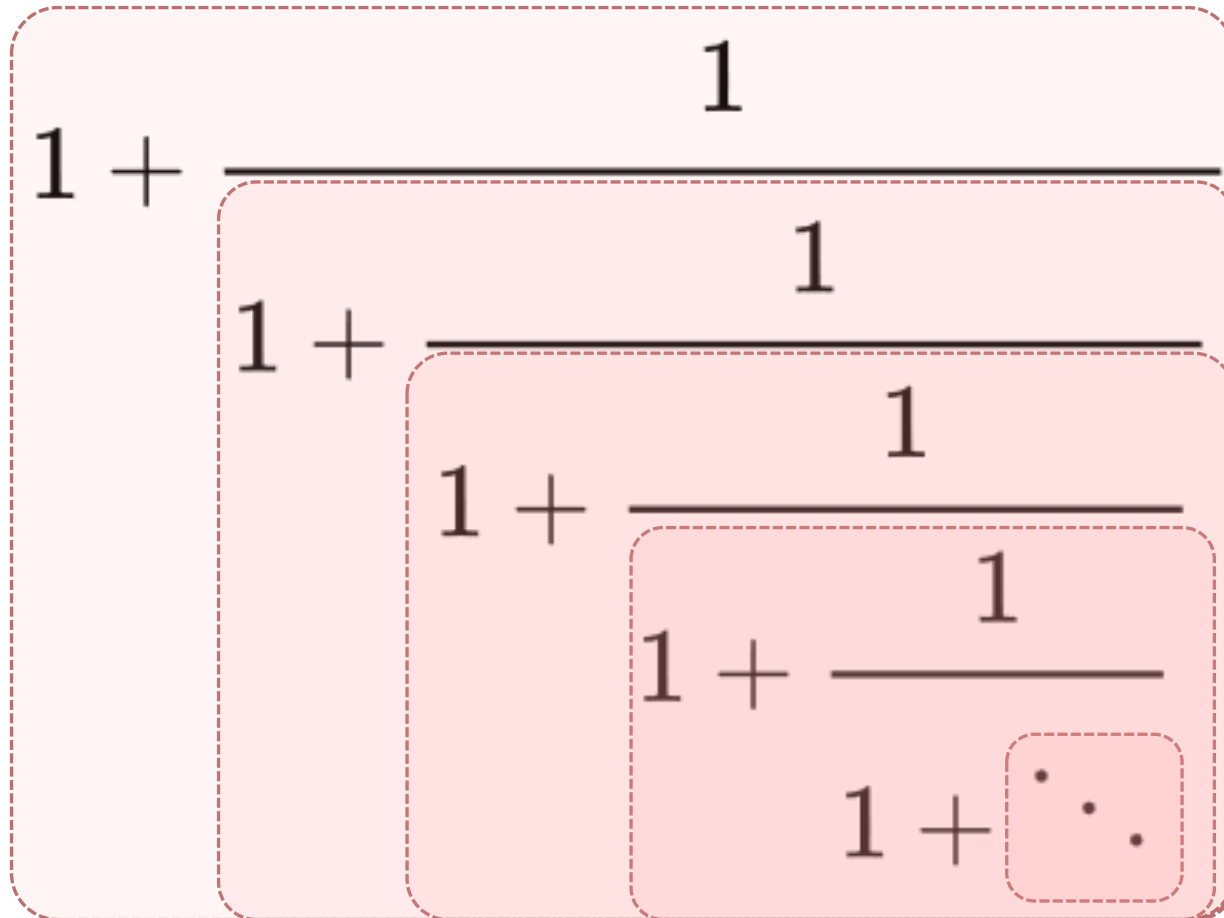


Tip 2: Quickly Identify Problem Types

Recursion



Step 1: Read the Recursive Structure



$$g_0 = 1.0$$

$$g_1 = 1.0 + \frac{1.0}{g_0}$$

$$g_2 = 1.0 + \frac{1.0}{g_1}$$

...

$$g_n = 1.0 + \frac{1.0}{g_{n-1}}$$



Setup Recursive Formula

Base Case: $g_0 = 1.0$

Recursive Formula: $g_n = 1.0 + \frac{1.0}{g_{n-1}}$



Tip 3: Do Algorithm Analysis

Tail Recursive Version:

```
double gold(int n, double g){  
    if (n==0) return g; // base case  
    return gold(n-1, 1.0+1.0/g);  
}
```

Calling Method:

```
gold(N, 1.0);
```



Tip 3: Do Algorithm Analysis

Iterative Version:

```
double gold(int n) {  
    double g;  
    for (int i=0; i<n; i++) g=1.0+1.0/g;  
    return g;  
}
```

Calling Method:

```
gold(N) ;
```




Pick Algorithm

- Iterative Version should be faster.



Tips 4: Master the Programming Language

- `BigDecimal` is needed.
- `No BigDecimal.One`
- `toString()` method
- `add()`, `divide()`



java.math.BigDecimal.divide()

- import java.math.BigDecimal;
- BigDecimal(BigDecimal **divisor**, int **scale**, RoundingMode **roundingMode**)
- **Parameters:**
 - **divisor** – Value by which this BigDecimal is to be divided.
 - **scale** – Scale of the BigDecimal quotient to be returned.
 - **roundingMode** – Rounding mode to apply.
- **Return value:**
 - This method returns a BigDecimal object whose value is this / divisor rounded as specified to given scale.



Other Tips (5, 6, and 7)

- Master the Art of Testing Code
- Practice and More Practice
- Team Work

Solution

SECTION 8



GoldRatio (Recursive Version)

```
1 import java.math.BigDecimal;
2 public class GoldRatio
3 {
4     final static int PRECISION = 100;
5     final static int LEVELS    = 200;
6     static BigDecimal goldenRatio(int n, BigDecimal g){
7         BigDecimal one = new BigDecimal("1.0");
8         if (n == 0) return g;
9         return goldenRatio(n-1, one.add(one.divide(g, PRECISION, BigDecimal.ROUND_HALF_UP)));
10    }
11    public static void main(){
12        System.out.print("\n");
13        for (int i=0; i<=LEVELS; i++){
14            System.out.printf("Golden Ratio at Division Level[%d] = %s\n", i, goldenRatio(i, new BigDecimal("1.0")).toString());
15        }
16    }
17 }
```



GoldenRatio (Iterative Version)

```
1 import java.math.BigDecimal;
2 public class GoldenRatio
3 {
4     final static int PRECISION = 100;
5     final static int LEVELS    = 200;
6     static BigDecimal goldenRatio(int n){
7         BigDecimal g = new BigDecimal("1.0");
8         BigDecimal one = new BigDecimal("1.0");
9         for (int i=0; i<n; i++){
10             g = one.add(one.divide(g, PRECISION, BigDecimal.ROUND_HALF_UP));
11         }
12         return g;
13     }
14     public static void main(){
15         System.out.print("\f");
16         for (int i=0; i<=LEVELS; i++){
17             System.out.printf("Golden Ratio at Division Level[%d] = %s\n", i, goldenRatio(i).toString());
18         }
19     }
20 }
```

```
Golden Ratio at Division Level[0] = 1.0
Golden Ratio at Division Level[1] = 2.0000000000000000
Golden Ratio at Division Level[2] = 1.5000000000000000
Golden Ratio at Division Level[3] = 1.6666666666666667
Golden Ratio at Division Level[4] = 1.6000000000000000
Golden Ratio at Division Level[5] = 1.6250000000000000
Golden Ratio at Division Level[6] = 1.615384615384615
Golden Ratio at Division Level[7] = 1.619047619047619
Golden Ratio at Division Level[8] = 1.617647058823529
Golden Ratio at Division Level[9] = 1.618181818181818
Golden Ratio at Division Level[10] = 1.617977528089888
Golden Ratio at Division Level[11] = 1.6180555555555555
Golden Ratio at Division Level[12] = 1.618025751072962
Golden Ratio at Division Level[13] = 1.618037135278514
Golden Ratio at Division Level[14] = 1.618032786885246
Golden Ratio at Division Level[15] = 1.618034447821682
Golden Ratio at Division Level[16] = 1.618033813400125
Golden Ratio at Division Level[17] = 1.618034055727554
Golden Ratio at Division Level[18] = 1.618033963166707
Golden Ratio at Division Level[19] = 1.618033998521803
Golden Ratio at Division Level[20] = 1.618033985017358
Golden Ratio at Division Level[21] = 1.618033990175597
Golden Ratio at Division Level[22] = 1.618033988205325
Golden Ratio at Division Level[23] = 1.618033988957902
Golden Ratio at Division Level[24] = 1.618033988670443
Golden Ratio at Division Level[25] = 1.618033988780243
Golden Ratio at Division Level[26] = 1.618033988738303
Golden Ratio at Division Level[27] = 1.618033988754323
Golden Ratio at Division Level[28] = 1.618033988748203
Golden Ratio at Division Level[29] = 1.618033988750541
Golden Ratio at Division Level[30] = 1.618033988749648
```

```
final static int PRECISION = 15;
final static int LEVELS    = 30;
```