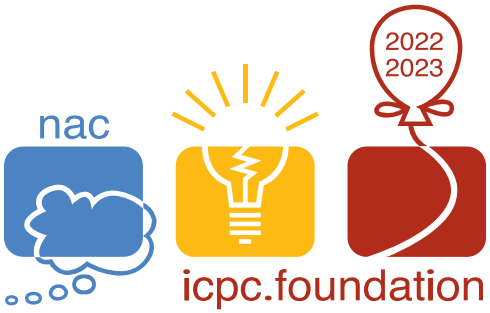
icpc international collegiate programming contest

ICPC North America Contests

Southern California Regional Contest

Official Problem Set



icpc global sponsor
programming tools

upsilon pi epsilon
honor society

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
I Could Have Won

"We will be closing in about 5 minutes. Thank you for visiting the ICPC gym today."

With the announcement, Alice and Bob stopped playing their table tennis match in the middle of the 10th game. Each game was played by the *first-to-11* rule, meaning that whoever scores 11 points first wins the game. Today, Bob narrowly defeated Alice by a single game and won the match.

After carefully inspecting how each game was played, however, Alice realized that she could have won the match if they played under slightly different rules, such as first-to-5 or first-to-8, instead of the regular first-to-11.

Given a sequence of points won by Alice and Bob, determine all values of k such that Alice would have won more games than Bob under the *first-to-k* rule.

Both Alice and Bob start with zero points in the beginning of a game. As soon as a player reaches k points, that player wins the game, and a new game starts. Neither player wins a game that is interrupted by the gym closing before either player reaches k points. Alice wins the match if she wins more games than Bob.

The input is a single line containing a string consisting of uppercase letters 'A' (for Alice) or 'B' (for Bob), denoting the winner of each point from the beginning of the match. The length of the string is between 1 and 2000 letters, inclusive.

Print a line with the number of values of k such that first-to-k games would have made Alice win more games than Bob. If this number is non-zero, print a second line with all such values of k in increasing order, separated from each other by spaces.

Sample Input 1

```
BBAAABABBAAABB
```

Output for Sample Input 1

```
3
3 6 7
```

Sample Input 2

```
AABBBAAB
```

Output for Sample Input 2

```
2
2 4
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Triangle Split

Your team is to write a program that, given a triangle on the 2-dimensional plane, finds the horizontal line of the form $y = a$ (*i.e.*, parallel to the x-axis) that splits the triangle into two equal-area polygons.

The input is a series of 1 to 100 lines, terminated by end-of-file. Each line is a test case, with three pairs of space-separated integers, denoting the (x, y) coordinates of the vertices. The coordinates are between -5000 and 5000, inclusive. The triangles are guaranteed to have positive area. In other words, no three vertices will be collinear.

For each test case, print one line with the value of a that would divide the triangle in two equal-area polygons. Values within 10^{-5} (relative or absolute) of the judges' reference values are considered correct.

Sample Input

```
-10 10 -10 -10 25 0
-2 5 0 10 10 3
```

Output for the Sample Input

```
0.0
5.81670
```

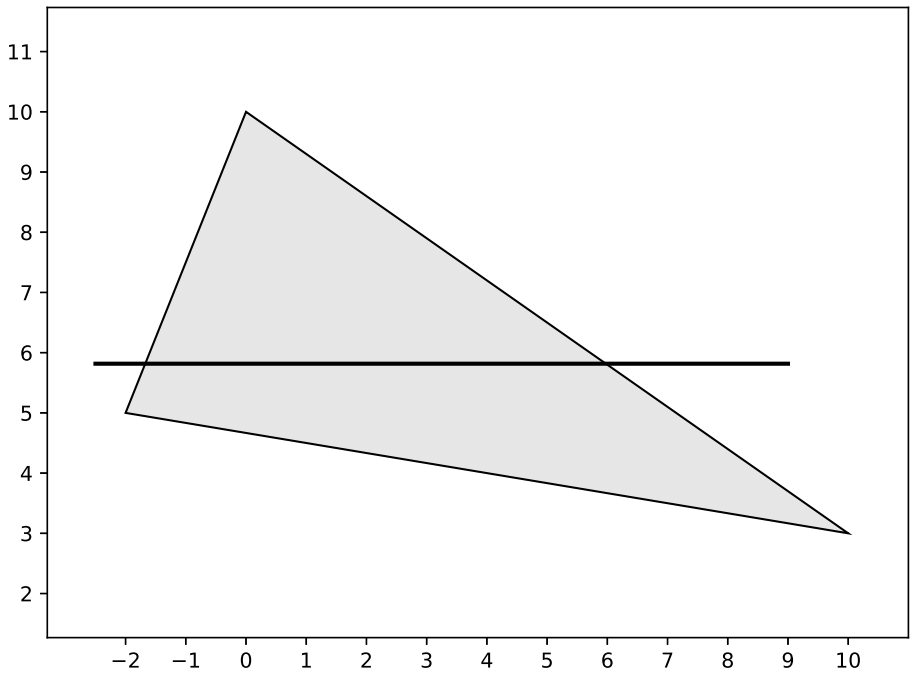


Figure 1. Diagram for the second line of the sample input.

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Stacks

Your team's job is to implement the Stacks processing language. See the attached manual page.

The input consists of a Stacks program. The Stacks program itself has no input.

The program consists of a series of lines of at most 80 characters terminated by end of file. There will be at most 100 lines.

The judged data are guaranteed to be valid with no run-time errors.

Sample Input 1

```
[1].print foo.clear foo.push [1 2 3] foo.add foo.print [1 5].print
[2].print foo.clear foo.push [1 2 3] foo.sub foo.print [1 -1].print
[3].print foo.clear foo.push [1 2 3] foo.addall foo.print [6].print
[4].print foo.clear foo.push [1 2 3 4] foo.len [1 2 3 4 5 6]  foo.print [1 2 3 4 6].print
  [ 5   ] .  print foo.clear foo.push [1 2 3 4] foo.len []  foo.print [1 2 3 4 0].print

[1].while{ [1].print [1].if { [2].print break [3].print} [4].print } [5].print
[1 2 5].print [].print

bar.push [3] bar.while { bar.print bar.sub1}
[3 2 1].print goodby.print
```

Problem 3
Stacks (continued)

Output for Sample Input 1

```
[ 1 ]
[ 1 5 ]
[ 1 5 ]
[ 2 ]
[ 1 -1 ]
[ 1 -1 ]
[ 3 ]
[ 6 ]
[ 6 ]
[ 4 ]
[ 1 2 3 4 6 ]
[ 1 2 3 4 6 ]
[ 5 ]
[ 1 2 3 4 0 ]
[ 1 2 3 4 0 ]
[ 1 ]
[ 2 ]
[ 5 ]
[ 1 2 5 ]
[ ]
[ 3 ]
[ 2 ]
[ 1 ]
[ 3 2 1 ]
[ ]
```

Sample Input 2

```
[1].if { [1].print [2].print else [3].print [4].print } [1 2] . print
[0].if { [1].print [2].print else [3].print [4].print } [ 3 4 ] . print
```

Output for Sample Input 2

```
[ 1 ]
[ 2 ]
[ 1 2 ]
[ 3 ]
[ 4 ]
[ 3 4 ]
```

STACKS

Stacks is a language where all variables are stacks and they are operated on by methods. Stack and method names are alphanumeric strings of at most 30 characters, starting with a letter. Case is significant. A stack name and a method name are separated by a '.':

```
stack1.method1
```

calls method1 with stack1. A method may also have another stack as a parameter:

```
stack1.method1 stack2
```

Where stack1 and stack2 are stack names and method1 is a method name.

Invocations and stack parameters are separated by whitespace (spaces, tabs, newlines). The '.' may also be preceded and followed by whitespace.

The contents of a stack are integers. -2147483648 <= i <= +2147483647

A stack can also be a literal: a list of whitepsace separated integers enclosed in '[]'

```
[1 2 3 4 5 -29 +23]
```

The rightmost integer is at the top of the stack. The brackets may be preceded and followed by whitespace.

There are two special methods: 'if' and 'while'

```
stack1.if{code1}
stack1.if{code1 else code2}
stack1.while{code3}
```

The braces may be preceded and followed by whitespace.

code1, code2, and code3 are sequences of stack method calls.

code1 is executed if the top of stack1 is non-zero and code2 is executed if the top of the stack is 0.

code3 is executed as long as the top of the stack is non-zero.

'if's and 'while's may be nested in the usual way.

Within a 'while' method there are two keywords: 'break' and 'continue'. 'break' passes control to code after the end of the innermost 'while' and 'continue' passes control back to the start of the innermost 'while' code.

'break', 'continue', 'else', 'if', and 'while' may not be stack names.

A literal stack can be called by a method:

```
    [1].while{code}
```

and be a parameter to a method:

```
   stack1.method1 [1 2 3]
```

It is an error to modify a literal stack.

In the method list below the contents of the stack, or stacks, before the execution of the method is shown on the left of '->' and contents after are shown on the right.

It is a run-time error if the necessary operands are not on a stack.

example:

```
stack.add
[... b a] -> [... b+a]
```

the top two integers on the stack are replaced by their sum. The top of the stack is on the right of the list [... top].

```
stack1.op stack2
[stack1-before] [stack2-before] -> [stack1-after] [stack2-after]
```

[] is an empty stack.

During execution the first time a stack is referenced it is created as an empty stack.

methods:

```
stack.add
[... b a] -> [... b+a]

stack.sub
[... b a] -> [... b-a]

stack.div
[... b a] -> [... b/a]
this is integer division
it is a run-time error if a==0
it is a run-time error if a or b is negative

stack.mod
[... b a] -> [... (b mod a)]
it is a run-time error if a==0
it is a run-time error if a or b is negative

stack.divmod
[... b a] -> [... (b mod a) (b/a)]
it is a run-time error if a==0
it is a run-time error if a or b is negative
```

```
stack.mul
[... b a] -> [... b*a]

stack.chs
[... a] -> [... -a]

stack.pop
[... b a] -> [... b]

stack.swap
[... b a] -> [... a b]

stack.add1
[... a] -> [... a+1]

stack.sub1
[... a] -> [... a-1]

stack.roll
[z ... b a] -> [a z ... b]

stack.top
[... b a] -> [a]

stack.dup1
[... b a] -> [... b a a]

stack.dup2
[... b a] -> [... b a b a]

stack.rev
[z ... b a] -> [a b ... z]

stack.addall
[z ... b a] -> [ (z+...+b+a)]

stack.mulall
[z ... b a] -> [ (z*...*b*a)]

stack.clear
[... b a] -> []

stack1.push stack2
[... a] [z ... b] -> [... a z ... b] [z ... b]

stack1.len stack2
[... a] [z ... b] -> [... a len(stack2)] [z ... b]
an empty stack has length 0

stack.print
[z ... b a] -> [z ... b a]
side effect: print the stack

Print a '[', a space, then the contents of the stack, if any,
```

from the bottom to the top, with a single space after each integer,
a ']', then a newline.

Integers are to be printed with no unnecessary leading '0's, a '-' if
necessary with no space between it and the integer. Do not print '+' signs.

example of print:
[].print
[-1 0 +1 2 3].print

output from example:
[ ]
[-1 0 1 2 3]

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Profitable Trip

You are planning a road trip. You have carefully mapped out all potential waypoints that you may stop at. From each waypoint, there are roads that allow you to drive to other waypoints, but roads can only be used in one direction. In order to alleviate traffic jams, the road planners have decided to require tolls on the more popular roads. The less popular roads may even pay you to drive on them.

As a result, it may be possible to make a profit from the road trip. But there is a catch—your wallet can only hold a maximum of w dollars more than what you have started with. You are allowed to get paid even when your wallet is full, but you will not be able to keep more than w dollars over what you started with. You may assume that whenever you need to pay a toll, you will have money to pay.

What is the maximum profit you can make on your trip?

The first line of input contains three space-separated integers n, m, and w ($1 \leq n, m \leq 2\,000$, $1 \leq w \leq 100$), where n is the number of waypoints, m is the number of roads, and w is the additional capacity of your wallet. The waypoints are numbered 1 through n. The first waypoint (1) is the start of your road trip, and the last waypoint (n) is the destination.

Each of the next m lines contains three space-separated integers u, v, and t ($1 \leq u, v \leq n$, $-100 \leq t \leq 100$), indicating that there is a road from waypoint u to waypoint v, with a gain of t dollars. If $t > 0$, then you are paid t dollars to use the road. If $t < 0$, you must pay a toll of $|t|$ dollars to use the road, resulting in a change of t to your profit. It is guaranteed that $u \neq v$, and there is at most one road from u to v (but there can also be a road from v to u). You may assume that it is possible to reach waypoint n from waypoint 1.

Output a single integer, which is the maximum profit you can make during the road trip. If there is a loss, output the loss as a negative number.

Sample Input 1

```
4 4 9
1 2 5
1 3 -2
2 4 1
3 4 10
```

Output for Sample Input 1

```
8
```

Sample Input 2

```
4 4 7
1 2 5
1 3 -2
2 4 1
3 4 10
```

Problem 4
Profitable Trip (continued)

Output for Sample Input 2

```
7
```

Sample Input 3

```
3 3 5
1 3 -10
3 2 2
2 3 -1
```

Output for Sample Input 3

```
4
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Lone Knight

In the game of chess, a knight moves as shown in Figure 1 below; each move is one square horizontally and two squares vertically or two squares horizontally and one square vertically. A rook can move any number of squares horizontally or vertically, but not both in the same move. If a square can be reached by a rook in one move, that square is said to be attacked by the rook.

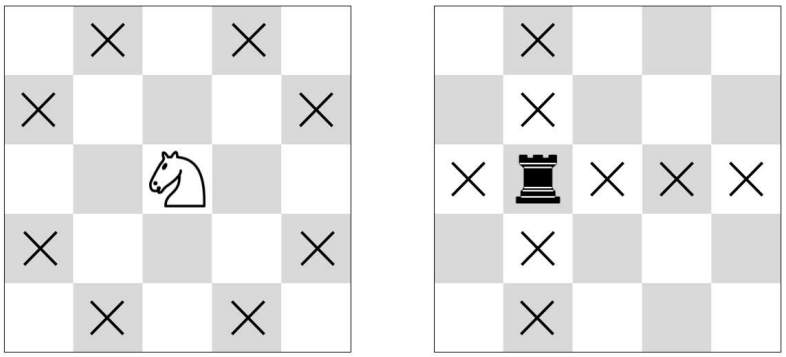


Figure 1. Moves of a chess knight and rook.

Consider an infinite chess board, with squares that can be indexed by integer coordinates. There is a white knight on the board on a square, and it wants to go to another square. However, there are also a number of black rooks on the board. The knight can make as many moves as it needs to get to its target square, but it cannot stop on a square that is attacked by or occupied by a rook. The rooks don't move.

Can the white knight reach its target square? You are to answer that question many times!

The first line of input contains two space-separated integers n and q ($1 \le n, q \le 1\,000$), where n is the number of black rooks and q is the number of queries.

Each of the next n lines contains two space-separated integers x and y. This indicates that there is a black rook at (x, y). No two rooks share the same square.

Each of the next q lines contains four space-separated integers x_s, y_s, x_t and y_t. This is a query, where the white knight starts at square (x_s, y_s) and wants to move to square (x_t, y_t).

All square coordinates in the input are no larger than 10^9 in absolute value. It is guaranteed that in every query the knight's initial and target squares are not attacked by or occupied by any rook, and the target square is not the same as the initial square.

For each query, output on a single line `1` if the knight can reach the target square, or `0` otherwise.

Problem 5
Lone Knight (continued)

Sample Input 1

```
6 6
10 14
1 0
0 1
4 9
9 13
5 9
2 2 3 4
2 2 2 4
2 2 6 4
2 2 2 10
7 11 6 2
6 2 8 12
```

Output for Sample Input 1

```
1
0
0
1
0
1
```

Sample Input 2

```
8 10
0 0
1 1
5 5
8 8
11 11
14 14
18 18
19 19
17 10 13 9
15 15 15 9
7 3 17 4
15 15 12 3
9 17 3 3
4 4 9 4
12 12 2 6
10 15 6 6
15 17 4 16
-1000000000 -999999999 15 7
```

Problem 5
Lone Knight (still continued)

Output for Sample Input 2

```
1
0
0
0
0
0
0
0
0
0
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Programming Contest

Alice, Bob, and Charlie are competing in the 2022/2023 Southern California Regional ICPC as a team. The team's goal is to place as high as possible. Teams in the contest are ranked by the total number of problems solved in decreasing order. Teams with the same number of solved problems are ranked in increasing order of the *time penalty*.

The *time penalty* is defined as the sum of the time it took for a team to solve each problem, measured from the beginning of the contest. Note that any unsolved problem does not contribute to the time penalty. For example, suppose that the contest started at 11:00 AM, and Alice's team solved 3 problems during the contest: first at 11:25 AM, second at 12:10 PM, and third at 1:30 PM. The team's time penalty is $25 + 70 + 150 = 245$.

There are n problems in the contest, numbered from 1 to n. The contest is T minutes long. As soon as the contest started, Alice's team quickly read through the problem statements to gauge each problem's difficulty. They then wrote down the estimated time it would take for each person to solve each problem. Alice estimated that problem i would take a_i minutes to solve. Similarly, Bob and Charlie estimated that they would each take b_i and c_i minutes to solve problem i. They trained hard for this moment, so we may assume that their estimates are exact, and that they only spent a negligible amount of time reading through the problems.

The team plans to come up with an optimal sequence of problems and tackle the problems in that order. Since they are given just one computer, only one person can be a *coder* and actually work on a problem at any given time. Unfortunately, they all prefer using different text editors, so it takes s minutes whenever they switch to a different coder. However, the first coder can start solving a problem immediately without spending s minutes to set up. A problem solved at the end of the Tth minute counts (see Sample Input 1).

Write a program to determine the number of problems and the time penalty they would get if they followed an optimal sequence of problems to tackle.

The first line of input contains three space-separated integers n, s, and T ($n \leq 18$; $s, T \leq 300$). The following lines represent problems 1 to n in order. Each line i contains three space-separated integers a_i, b_i, and c_i, which describe the team members' estimated time to solve problem i. Each of these numbers is non-negative.

Your program must output two space-separated integers on a line: the total number of problems solved, followed by the total time penalty, when an optimal strategy is employed.

Sample Input 1

```
2 1 10
8 3 7
6 8 10
```

Output for Sample Input 1

```
2 13
```

Problem 6
Programming Contest (continued)

Sample Input 2

```
6 10 300
0 5 2
70 20 45
38 90 29
119 118 120
102 999 999
999 173 999
```

Output for Sample Input 2

```
5 559
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Color Tubes

There is a new puzzle generating buzz on social media—Color Tubes. The rules are relatively simple: you are given $n + 1$ tubes filled with $3n$ colored balls. Each tube can hold at most 3 balls, and each color appears on exactly 3 balls (so there are n colors).

Using a series of moves, you are supposed to reach a Color Tubes state—each tube should either hold balls of a single color or it should be empty. The only move allowed is to take the top ball from one tube and place it into a different tube that has room for it (i.e. holds at most two balls before the move).

You want to write a program to solve this puzzle for you. Initially, you are not interested in an optimal solution, but you want your program to be good enough to solve any puzzle configuration using at most $20n$ moves.

The first line of input contains a single integer n ($1 \leq n \leq 1000$), which is the number of colors.

Each of the next $n + 1$ lines contains three space-separated integers b, m and t ($0 \leq b, m, t \leq n$), which are the descriptions of each tube, where b is the color of the ball on the bottom, m is the color of the ball in the middle, and t is the color of the ball on the top. The tubes are numbered from 1 to $n + 1$ and are listed in order. The colors are numbered from 1 to n. The number 0 describes an empty space. It is guaranteed that no empty space will be below a colored ball.

On the first line output an integer m, the number of moves that your program will use to solve the puzzle. Remember, m has to be at most $20n$.

On the next m lines, output two space-separated integers u and v that describe a move ($1 \leq u, v \leq n+1$). In each move, you are taking the uppermost ball out of tube u and placing it in tube v, where it will fall until it hits the uppermost ball already in that tube, or the bottom of the tube if the tube is empty.

Your solution will be deemed incorrect if it uses more than $20n$ moves, or any of the moves are not allowed, or the final configuration is not a Color Tubes state.

Sample Input 1

```
3
2 2 0
1 3 1
3 1 2
3 0 0
```

Output for Sample Input 1

```
6
3 1
2 3
2 4
3 2
3 2
3 4
```

Problem 7
Color Tubes (continued)

Sample Input 2

```
1
0 0 0
1 1 1
```

Output for Sample Input 2

```
0
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Distinct Parity Excess

A property of any positive integer is its *prime parity*, which is derived from the count of its distinct prime factors. If this count is even, the prime parity is even; if the count is odd, the prime parity is odd.

You are given a sequence of ranges to test. Each range is given as two numbers a and b, defining the range from a to b inclusive. You want to compute the excess of even parity integers over odd parity integers over this range. If there are more odd parity integers, the computed difference will be negative.

Input is a series of from 1 to 100 lines, ending with end-of-file. Each line contains two integers a and b ($2 \leq a \leq b \leq 10^7$), which is a range to test.

Output n lines, one for each range in the input. For each range, print a line with a single integer giving the excess of even parity integers over odd parity integers with no unnecessary leading zeroes.

Sample Input

```
2 100
2 50
50 100
2 1000
100 143
1234567 1234567
1000000 10000000
2 1000000
80000 90000
1000000 1000000
```

Output for the Sample Input

```
13
-1
15
63
0
1
259
-1909
-31
1
```

2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Workout Anonymizer

A large technology company has decided to produce a fitness tracking app, *AnonyFit*. AnonyFit permits users to compare their workouts with other random users. To minimize the likelihood that workout data can reveal the location and exercise habits—hence identity—of each user, the collected workout data is run through an anonymizer prior to uploading to AnonyFit servers for comparison. Your team must write the anonymizer. Your program must take time-stamped geospatial data as *absolute* time and location data and covert it into anonymized elapsed and cumulative *relative* metrics.

For displacements within a few kilometers, location data can use a locally flat earth model, meaning that longitude and latitude can be viewed as a rectangular (x, y) coordinate, and altitude as perpendicular displacement above or below the local plane. To translate a non-planar earth into a flat model, the input specifies conversion factors to convert latitudes and longitudes into tangential (locally flat) distances. Positive latitude denotes degrees north of the equator, negative denotes degrees south of the equator. Positive longitude denotes degrees east of the prime meridian, negative denotes degrees west of the prime meridian.

The first input line contains the local degrees latitude to meters (y-axis) conversion factor. The second line contains the local degrees longitude to meters (x-axis) conversion factor. The remaining N lines until end-of-file ($2 \leq N \leq 100$) each contain four values separated by whitespace: the time of the sample in *hh:mm:ss* format, the latitude in degrees, longitude in degrees, and altitude in meters for the sample. Latitude, longitude, and altitude are floating point values.

Successive sample times always differ by at least one second. Times appear in sequence and will not cross midnight. Longitudes never cross the international date line. Latitudes never get anywhere near the north or south poles.

Produce $N - 1$ lines that transform the input to elapsed time from the first time stamp, cumulative 3-D distance traveled, and altitude offset from the starting point. Separate the values by spaces. Display elapsed time as seconds, and both cumulative distance and altitude offset as meters. Distance and altitude values must be accurate to within 0.001 meters.

Figure 1 shows a cumulative distance graph vs. elapsed time plotted from the sample output. Figure 2 shows a relative altitude graph vs. elapsed time plotted from the sample output.

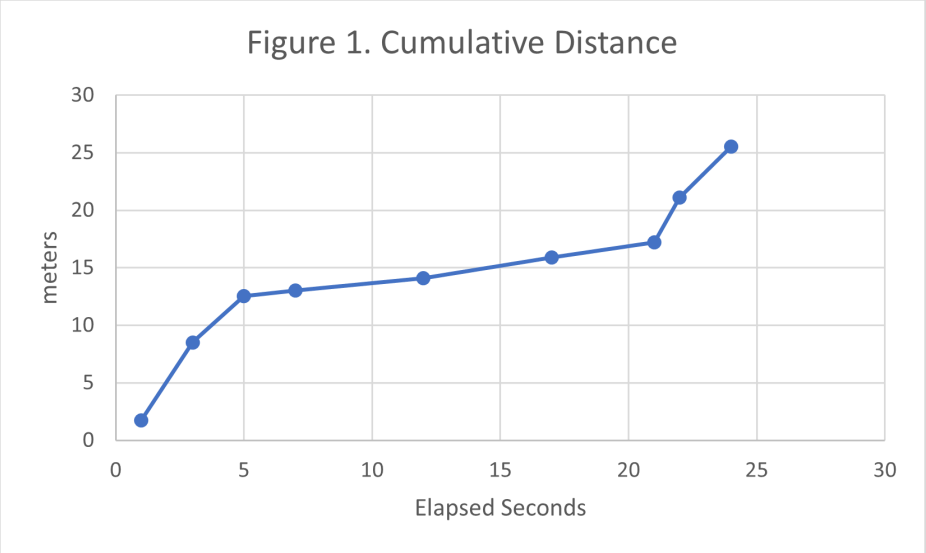
Sample Input

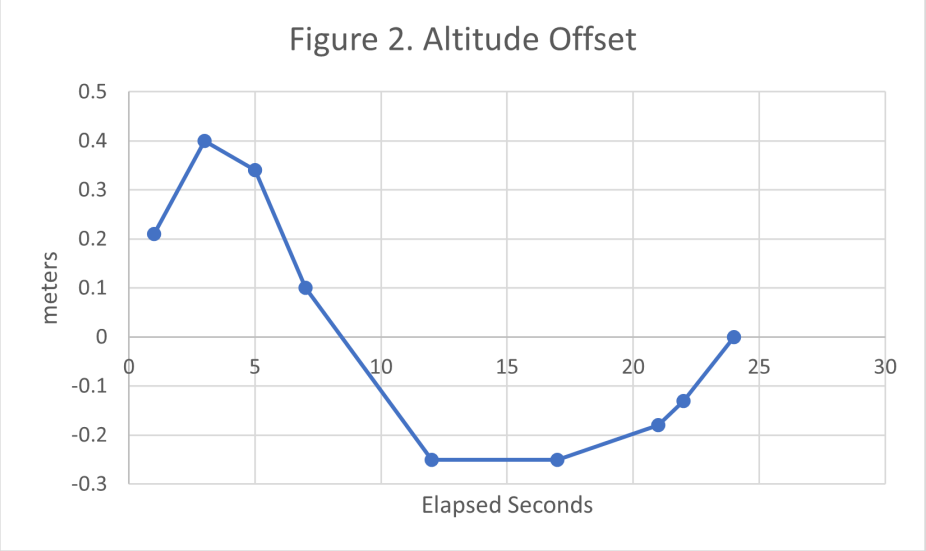
```
111111.1
50000.0
10:01:01 34.500000 -118.060000 800.1
10:01:02 34.500012 -118.060022 800.31
10:01:04 34.500072 -118.059998 800.5
10:01:06 34.500108 -118.059990 800.44
10:01:08 34.500110 -118.059983 800.2
10:01:13 34.500101 -118.059980 799.85
10:01:18 34.500085 -118.059984 799.85
10:01:22 34.500075 -118.059998 799.92
10:01:23 34.500040 -118.059998 799.97
10:01:25 34.500000 -118.060000 800.1
```

Problem 9
Workout Anonymizer (continued)

Output for the Sample Input

```
1 1.74123 0.210000
3 8.51770 0.400000
5 12.5381 0.340000
7 13.0171 0.100000
12 14.0872 -0.250000
17 15.8762 -0.250000
21 17.1913 -0.180000
22 21.0805 -0.130000
24 25.5279 0.00000
```





2022/2023 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
Xylophone

A *xylophone* is a musical instrument made of wooden bars, each of which makes a specific pitch when struck with a mallet. The wooden bars must have contiguous integer lengths from the longest to the shortest, without duplicates. In other words, every bar except for the rightmost one must have a length exactly 1 longer than the one immediately to its right. For example, a xylophone may have bars of lengths [7, 6, 5, 4] or [10, 9, 8], but not [7, 5, 4] nor [3, 3, 2, 1].

You already have 3 wooden bars of different lengths, and want to create a xylophone using all of them. You may not cut the bars or alter them in any way, but you may buy additional wooden bars as necessary. The cost of buying a wooden bar is equal to its length. Find the minimum cost to build a xylophone.

The input contains a single line with three space-separated integers, denoting the lengths of the wooden bars you already have. Each integer is between 1 and 5000, inclusive. You are guaranteed that the three integers are distinct.

Your program must output a single integer that represents the minimum cost to make a xylophone using all three given wooden bars.

Sample Input

```
10 3 7
```

Output for the Sample Input

```
32
```