HP Forum Archive 21

[Return to Index | Top of Index]

A Sunday Programming Challenge

*Message #1 Posted by **Allen** on 11 Mar 2012, 1:03 a.m.*

Adapted from Project Euler problem 370

> Quote:
>
> Let us define a geometric triangle as an integer sided triangle with sides a <= b <= c so that its sides form a geometric progression, i.e. b^2 = a*c.
>
> An example of such a geometric triangle is the triangle with sides a = 144, b = 156 and c = 169.

There are 42 geometric triangles with perimeter <= 100. Appropriately for this problem F(100)=42.

Write a program (42S or platform of your choice) optimized for speed then for size that can solve F(1000) on a real 42S or F(1e8) on PC emulator in a reasonable time.

Edited: 11 Mar 2012, 1:05 a.m.

Re: A Sunday Programming Challenge

*Message #2 Posted by **Gerson W. Barbosa** on 11 Mar 2012, 1:17 p.m.,*
in response to message #1 by Allen

Unoptimized and very slow, I fear:

```
00 { 71-Byte Prgm }
01>LBL 00
02 STO 00
03 3
04 BASE÷
05 1E-3
06 +
07 STO 01
08 IP
09 STO 02
10>LBL 01
11 RCL 01
12 1
13 BASE-
14 STO 03
15>LBL 02
```

```
16 RCL 01
17 IP
18 X^2
19 RCL÷ 03
20 STO 04
21 IP
22 RCL 03
23 RCL+ 01
24 X<Y?
25 GTO 04
26 RCL 04
27 FP
28 X!=0?
29 GTO 03
30 RCL 00
31 RCL 04
32 RCL 01
33 BASE+
34 RCL+ 03
35 X>Y?
36 GTO 03
37 1
38 STO+ 02
39>LBL 03
40 1
41 STO- 03
42 GTO 02
43>LBL 04
44 DSE 01
45 GTO 01
46 RCL 02
47 .END.


 100 XEQ 00  -->   42     (1 min 19 sec, real HP-42S)
1000 XEQ 00  -->  532     (~1 second (Free42 Decimal)
```

Re: A Sunday Programming Challenge

*Message #3 Posted by **Paul Dale** on 11 Mar 2012, 5:30 p.m.,*
in response to message #2 by Gerson W. Barbosa

What is step "24 X<Y?" testing for? It looks like a+b < c as a loop termination condition. Why is this necessary?

Apart from the trivial cases of a=b=c this program would seem to produce (I converted it to C to play with so could be wrong here) these nine extras:

```
 25  30  36
 20  30  45
 18  24  32
 16  24  36
 16  20  25
 12  18  27
  9  12  16
```

```
 8  12  18
 4   6   9
```

My next question is where are triples like 1 3 9 or 14 28 56? These look to be in geometric progression and satisfy a c = b^2. In fact I get fifty of these which include the nine above:

```
25  30  36
20  30  45
18  30  50
16  28  49
14  28  56
13  26  52
18  24  32
16  24  36
12  24  48
 9  24  64
11  22  44
 9  21  49
 7  21  63
16  20  25
10  20  40
 8  20  50
12  18  27
 9  18  36
 6  18  54
 8  16  32
 4  16  64
 9  15  25
 5  15  45
 3  15  75
 7  14  28
 4  14  49
 9  12  16
 8  12  18
 6  12  24
 4  12  36
 3  12  48
 2  12  72
 5  10  20
 4  10  25
 2  10  50
 3   9  27
 1   9  81
 4   8  16
 2   8  32
 1   8  64
 1   7  49
 4   6   9
 3   6  12
 2   6  18
 1   6  36
 1   5  25
 2   4   8
 1   4  16
 1   3   9
 1   2   4
```

Add to this the 33 a=b=c cases and I'm at 83.

What am I missing or is the problem stated incorrectly?

- Pauli

Re: A Sunday Programming Challenge

*Message #4 Posted by **Gerson W. Barbosa** on 11 Mar 2012, 6:11 p.m.,*
in response to message #3 by Paul Dale

I was getting 83 also, but then I realized we are looking for triangles. In a triangle the length of one side cannot exceed the sum of the lengths of the other two sides, as we know. That's what step 24 is testing for.

Gerson.

Re: A Sunday Programming Challenge

*Message #5 Posted by **Paul Dale** on 11 Mar 2012, 9:30 p.m.,*
in response to message #4 by Gerson W. Barbosa

I knew I was missing something obvious, I feel silly now :-)

I've not been thinking well recently :-(

- Pauli

Re: A Sunday Programming Challenge

*Message #6 Posted by **Gerson W. Barbosa** on 12 Mar 2012, 7:19 p.m.,*
in response to message #5 by Paul Dale

I made the same mistake too, but I prefer to think of it as a temporary trip into a non-Euclidean geometry realm :-)

Gerson.

Re: A Sunday Programming Challenge

*Message #7 Posted by **Paul Dale** on 12 Mar 2012, 8:49 p.m.,*
in response to message #6 by Gerson W. Barbosa

I did almost include a comment about the problem not specifying the triangles live in a Euclidean space. Riemannian spaces are so much more fun and correspond better to the word we live in.

- Pauli

Re: A Sunday Programming Challenge

*Message #8 Posted by **Allen** on 11 Mar 2012, 6:12 p.m.,*
in response to message #3 by Paul Dale

> Quote:
>
> What is step "24 X<Y?" testing for? It looks like a+b < c as a

How do you know this is a triangle:

```
  1   9  81
```

Re: A Sunday Programming Challenge

*Message #9 Posted by **Gerson W. Barbosa** on 12 Mar 2012, 9:39 p.m.,*
in response to message #3 by Paul Dale

> Quote:
>
> Apart from the trivial cases of a=b=c this program would seem to produce (I converted it to C to play with so could be wrong here) these nine extras:
>
> ```
>  25  30  36
>  20  30  45
>  18  24  32
>  16  24  36
>  16  20  25
>  12  18  27
>   9  12  16
>   8  12  18
>   4   6   9
> ```

Your conversion to C is perfect. Here is the BASIC program I wrote to test my second algorithm. The RPN code is just a quick & dirt conversion from it.

```
10 CLS
15 INPUT L: PRINT
20 N = INT(L / 3)
25 B = N
30 A = B - 1
35   C = B * B / A
40   IF INT(C) > (A + B) THEN 65
45   IF C - INT(C) <> 0 THEN 55
50     IF C + A + B <= L THEN N = N + 1:  PRINT USING "###"; A; B; C
55     A = A - 1
60   GOTO 35
65   B = B - 1
70 IF B > 2 THEN 30
75 PRINT : PRINT N
```

```
? 100


 25 30 36
 20 30 45
 18 24 32
 16 24 36
 16 20 25
 12 18 27
  9 12 16
  8 12 18
  4  6  9


 42
```

Would you mind posting your C program? Yesterday I tried to convert it to C by I quit after a while (literally after a while :-). Looks like my BASIC program was not well structured enough. I changed it a bit later but still had trouble with the inner loop when trying to convert it to C:

```
10 CLS
15 INPUT L
20 N = INT(L / 3)
25 FOR B = N TO 2 STEP -1
30   A = B - 1
35     C = B * B / A
40     IF INT(C) > (A + B) THEN 65
45     IF C - INT(C) = 0 THEN IF C + A + B <= L THEN N = N + 1
55     A = A - 1
60   GOTO 35
65   '
70 NEXT B
75 PRINT N
```

Thanks,

Gerson.

Re: A Sunday Programming Challenge

*Message #10 Posted by **Paul Dale** on 12 Mar 2012, 9:47 p.m.,*
in response to message #9 by Gerson W. Barbosa

Seems the code had disappeared, but here is a second stab which is pretty close to the original.

- Pauli

```
#include <stdio.h>


const int p = 100;
```

```
int main() {
    int a, b;
    const int n = p / 3;
    int count = n;


    for (b=n; b>0; b--)
        for (a=b-1; a>0; a--) {
            const int c = b*b/a;
            if (b*b == a*c && c <= a+b && a+b+c <= p)
                count++;
        }
    printf("%d\n", count);
    return 0;
}
```

Re: A Sunday Programming Challenge

*Message #11 Posted by **Gerson W. Barbosa** on 12 Mar 2012, 10:19 p.m.,*
in response to message #10 by Paul Dale

Thank you!

F(100 000) = 75243 in 10 seconds. F(1000 0000) will take a while more * :-)

Gerson.

* this would require int_64, I think.

Edited: 12 Mar 2012, 10:27 p.m.

Re: A Sunday Programming Challenge

*Message #12 Posted by **Paul Dale** on 12 Mar 2012, 10:50 p.m.,*
in response to message #11 by Gerson W. Barbosa

Yes it will. $(10{,}000{,}000 / 3)^2$ exceeds the size of a 32 bit integer. Even 1,000,000 will be too large.

Given that the process is $O(n^2)$, I'd expect 1,000,000 to take 100 times as long and 10,000,000 about 10,000 times as long (very roughly).

I've been trying to think of a way of iterating over c and determining possible values of a & b to scan over. No luck thus far. We know that $a.x^2 = b.x = c$ for some values of x.

- Pauli

Re: A Sunday Programming Challenge

*Message #13 Posted by **Gerson W. Barbosa** on 14 Mar 2012, 7:47 a.m.,*
in response to message #12 by Paul Dale

This is faster, but still an $O(n^2)$ process:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int main()
{
  int a, b, len, n;
  scanf("%d", &len);
  double c, k;
  n = len/3;
  k = (sqrt(5) - 1) / 2;
  for (b = n; b >= 2; b--)
  {
    a = b - 1;
    while (a > b * k )
    {
      c = 1.0 * b * b / a;
      if (c - int(c) == 0)
        if (a + b + c <= len)
          n++;
      a--;
    }
  }
  printf("n = %d\n",n);
  system("pause");
  return 0;
}


100000
n = 75243     (7 seconds @ 1.86 GHz)
```

Gerson.

Re: A Sunday Programming Challenge

*Message #14 Posted by **C.Ret** on 11 Mar 2012, 5:18 p.m.,*
in response to message #1 by Allen

> Quote:
>
> There are 42 geometric triangles with perimeter <= 100.

I get a significant different number of 'goemetric integer traiangle' !

I may have miss samething, or badly interpret the conditions.

I count 83 integer sided triangles (a,b,c) with a <= b <=c and perimeter P less to 100 who satisfed $b^2 = a.c$.

Are you sure you are not counting integer sided triangles with different side's length restrictions (such as 1 < a < b < c) ?

Edited: 11 Mar 2012, 5:45 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #15 Posted by **Allen** on 11 Mar 2012, 5:24 p.m.,*
in response to message #14 by C.Ret

> Quote:
>
> I count 83 integer sided triangles (a,b,c) with a <= b <=c and perimeter P less to 100.

Remember it must also satisfy b^2 = a*c

Are you checking that condition also?

Re: A Sunday Programming Challenge

*Message #16 Posted by **Paul Dale** on 11 Mar 2012, 5:31 p.m.,*
in response to message #15 by Allen

I'm getting 83 too (see above). They all satisfy the condition as stated.

- Pauli

Re: A Sunday Programming Challenge

*Message #17 Posted by **Allen** on 11 Mar 2012, 6:31 p.m.,*
in response to message #16 by Paul Dale

> Quote:
>
> I'm getting 83 too (see above). They all satisfy the condition as stated.

Yes, you're on the right track. Your list satisfies every condition except one very important one... See my question above.

Re: A Sunday Programming Challenge

*Message #18 Posted by **Valentin Albillo** on 11 Mar 2012, 6:50 p.m.,*
in response to message #16 by Paul Dale

> Quote:
>
> I'm getting 83 too (see above). They all satisfy the condition as stated.

> - Pauli

As has already been said, the original statement explicitly says we're looking for triangles, not mere triplets of integers, and in every triangle you have that none of the sides can equal or exceed the sum of the others.

So, if you've got alleged triangle sides A, B, and C, then they must satisfy:

A+B > C

A+C > B

B+C > A

else you can't construct a triangle having those sides. If in doubt, just try to draw a triangle having sides 1, 2, and 4, and see if you succeed. You won't.

With that restriction in place you get exactly 42 integer triplets which can indeed form a triangle whose perimeter doesn't exceed 100.

Best regards from V.

Edited to correct a typo, as kindly pointed out below

Edited: 11 Mar 2012, 7:38 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #19 Posted by **fhub** on 11 Mar 2012, 7:06 p.m.,*
in response to message #18 by Valentin Albillo

> Quote:
>
> So, if you've got alleged triangle sides A, B, and C, then they must satisfy:
>
> A+B < C
>
> A+C < B
>
> B+C < A

If you change all 3 '<' to '>', then you're right. ;-)

Re: A Sunday Programming Challenge

*Message #20 Posted by **Valentin Albillo** on 11 Mar 2012, 7:36 p.m.,*

in response to message #19 by fhub

Yes, of course, thank you.

Unless I incur in some other typo, I get (in reasonable times):

```
F(100)   = 42
F(1000)  = 532
F(10000) = 6427
```

Best regards from V.

Re: A Sunday Programming Challenge

*Message #21 Posted by **Allen** on 11 Mar 2012, 8:41 p.m.,*
in response to message #20 by Valentin Albillo

Valentin, Greetings, What platform are you using? The trusty 71b?

Re: A Sunday Programming Challenge

*Message #22 Posted by **Valentin Albillo** on 12 Mar 2012, 8:20 a.m.,*
in response to message #21 by Allen

> Quote:
>
> Valentin, Greetings, What platform are you using? The trusty 71b?

Hi, Allen.

Yes, I'm using the ole trusty HP-71B in its J-F Garnier's Emu71 incarnation, as usual. Using the naïve approach you can't do better than this:

```
>LIST


10 DESTROY ALL @ INPUT K @ L=K DIV 3 @ N=L @ F=(1+SQR(5))/2
20 SETTIME 0 @ FOR A=1 TO L @ M=(K-A)*A @ FOR B=A+1 TO MIN(A*F,L)
30 IF MOD(B*B,A) THEN 40 ELSE IF B*(A+B)>M THEN 50 ELSE N=N+1
40 NEXT B
50 NEXT A @ DISP N,TIME


>RUN
? 100
          42    .02


>RUN
? 1000
         532    .52
```

```
>RUN
? 10000
         6427   51.57
```

which uses two nested loops and minimizes the number of arithmetic operations and comparisons in the inner one to speed execution up.

Nevertheless, using two loops implies increasing the running time by two orders of magnitude (x100) when the input increases by just one (x10), as was bound to be the case and as the timings above clearly demonstrate thus making this naïve approach utterly hopeless for large inputs.

What's needed is a subtler approach that keeps the running time approximately linear on the input, not quadratic as shown here, but I won't spoil the fun for now ... XD

Nice challenge, Allen, thanks for it and best regards from V.

Edited: 12 Mar 2012, 8:33 a.m.

Re: A Sunday Programming Challenge

*Message #23 Posted by **Marcus von Cube, Germany** on 12 Mar 2012, 10:13 a.m.,*
in response to message #22 by Valentin Albillo

> Quote:
>
> F=(1+SQR(5))/2

I came up with the constraint A > (SQR(5)-1)/2 * B, using a pencil and paper approach. Here are some thoughts:

1. Loop over B (from 1 to N/3, maybe one more and check A+B+C)
2. Add one to the count for the case A=B=C.
3. Loop over A from CEIL((SQR(5)-1)/2 * B) to B-1 and check if B*B/A is an integer.

The last step can be omitted if B is prime or a power of a prime if this can be easily determined.

Edited: 12 Mar 2012, 10:16 a.m.

Re: A Sunday Programming Challenge

*Message #24 Posted by **Allen** on 12 Mar 2012, 9:05 p.m.,*
in response to message #23 by Marcus von Cube, Germany

> Quote:
>
> Here are some thoughts:

1. Loop over B...
2. Loop over A...

Yes, this is in the right direction, Keep in mind, when dealing with Big O Notation reducing the limits may be slightly faster, but not *exponentially* faster, since you have to loop over BOTH A and B to solve F(N).

As N grows, your solve time will still grow quadratically O(N^2)- pronounced "Order N squared".

I have studied this problem for over a month- and I am convinced you could teach a large part of an undergraduate computer science course based on solving this problem. In the initial Problem 370, the writers ask for F(2.5e13). The 14-digit answer is impractical on a pocket calculator. But smaller limits like F(1e8) are well within "reasonable" times for emulated calculators, and F(10,000) within reasonable times on a real calculator.

After a month of study, I've seen/found/broken four different approaches to this problem:

1. Naive approach O(N^2) with two loops (easy to write but slow) - like Valentin's nice 71b solution
2. Reduced O(N^2) approach with two loops, with more mathematical limits and assumptions- See Gerson and C.Ret's short and well posed solution- I like them both for different reasons.
3. An O(N) approach (I will post shortly)
4. The most common approach posted in the Euler Discussion forum, which I will not discuss here.

Except for one program, all of the posted approaches in the Project Euler discussion forum for Problem 370 are in category 4 (due to the large size of 2e13). I believe category 4 approach is quite impractical on a pocked calculator for reasons I won't discuss here. But there is at least 1 category 3 approach that is worth discussing in more detail....

Edited: 12 Mar 2012, 9:09 p.m.

Re: A Sunday Programming Challenge

*Message #25 Posted by **Valentin Albillo** on 13 Mar 2012, 7:02 a.m.,*
in response to message #24 by Allen

Hi, Allen:

> Quote:
>
> After a month of study, I've seen/found/broken four different approaches to this problem:

After a couple' hours of study I came up with a O(n) solution (or $O(n)^{1+e}$), or O(n*ln(n)), difficult to assess exactly ...) which can find the solution for reasonably large values of N in reasonable times on Emu71, about 1,000 times faster in a compiled high-level language.

I suggest we all post and comment our solutions next Friday so as not to spoil the fun for people working right now on their very own approach. Deal ? :D

Best regards from V.

Re: A Sunday Programming Challenge

*Message #26 Posted by **Valentin Albillo** on 16 Mar 2012, 3:48 p.m.,*
in response to message #25 by Valentin Albillo

Hi, all

We're already on Friday so time to post some non-naïve solutions to this challenge.

When I created my above naïve solution I fully realized that having two nested loops, though simple and immediately coded, was not the way to go as you'll inevitably get times of order $O(n^2)$. Some clever tricks with the nested loop limits did significantly improve the timing but it still was $O(n^2)$.

Regrettably I couldn't dedicate more than a few hours to this but I decided to make the most of the available time and last Tuesday I concocted this much improved 6-line, 234-byte solution for the HP-71B which I'll presently comment:

```
1 DESTROY ALL @ INPUT K @ P=4*K @ N=0 @ F=(1+SQR(5))/2 @ SETTIME 0
2 FOR A=1 TO K/3 @ N=N+FNT(FNF(A)) @ NEXT A @ DISP N;TIME


3 DEF FNT(N)=MIN(A*F,A*((SQR(P/A-3)-1)/2)) DIV N-A DIV N+NOT MOD(A,N)


4 DEF FNF(N) @ M=N @ E=0
5 D=PRIM(N) @ IF D<2 THEN FNF=M/(E*(N=E)+(N#E)) @ END ELSE N=N/D
6 IF E=D THEN M=M/E @ E=0 @ GOTO 5 ELSE E=D @ GOTO 5
```

First, some timings:

Perimeter	Triangles	Time (sec).	Time increase
100	42	0.03	-
1000	532	0.16	5.2x
10000	6427	1.8	11.3x
100000	75243	19.52	10.8x
1000000	861805	219.62	11.3x
10000000	9712598	2589.75	11.8x

Now for the comments:

- As you can see, the main program is just 2-line long and all it does is to initialize some needed constants so that the ensuing loop will go as fast as possible by removing invariant operations within the loop, then the single loop on the shortest side (A) is executed, which simply calls a user-defined function FNT which returns the number of valid triangles for that value of A and adds this up to the tally, eventually returning the total and the timing when the loop is done for. Simple as can be.

- Line 3 is a nifty user-defined single-line function, **FNT**, that duly returns the number of valid triangles within limits given by the maximum and minimum possible values for side B, which are cleverly computed so that no tests about triangularity and/or perimeter are necessary within the loop at all, this way:

  - the **A*F** term, where F is the golden ratio, ensures that A, B, and C form inded a triangle, as C is guaranteed to be less than A+B.

  - the A*((SQR(P/A-3)-1)/2)) term ensures that the sum of A+B+C is guaranteed to be less than or equal to the given perimeter.

- Lines 4-5 constitute another user-defined function, **FNF**, that given the value of A computes the periodicity with which B*B is divisible by A. This way there's no need for another nested loop that examines each B in turn to check for divisibility but we can simply tell at once just how many values of B do meet the divisibility condition by simply using this computed periodicity as input for the triangle-counting function FNT.

Thus, this approach does allow us to get over the dreaded $O(n^2)$ to achive something much more like O(n*log(n)), as the results above clearly show (~ 11.3x increase in time for every 10x increase in the input value).

Note:

FNF uses the PRIM function from the HP-71B JPC ROM for faster execution. If this function were unavailable the functionality can easily be achieved using plain-vanilla HP-71B BASIC code, which would affect speed by a constant factor (~3x slower) but would *not* change the order O(n*log(n)) so timing would ultimately be much faster than any $O(n^2)$ approach for sufficiently large input values of the perimeter.

That's all on my side, thanks to Allen for an interesting challenge.

Best regards from V.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #27 Posted by **Allen** on 16 Mar 2012, 9:35 p.m.,*
in response to message #26 by Valentin Albillo

Valentin, C.Ret, Gerson- thank you for your interesting solutions. I hope I was clear in saying there is nothing wrong with the naive approach- only an observation that it is the easiest solution to develop and program. Your solutions are certainly well thought out and not "naive" as in a bad.

I admit my approach is not the fastest I've seen (or even close) but I propose a method that neither counts *a* nor *b*, but only focuses on counting similar triangles with ratio of b/a. In so doing, There are no perimeter checks, no triangle inequality, and no loops over a or b- in fact there is only 1 loop to generate a series of fractions, and 1 check to make sure that fraction b/a does not pass the golden ratio. This allows execution to proceed in O(N) time - requiring roughly N/16 cycles for F(N).

How?

F(100) can be calculated in 7 iterations of fraction loop that creates coprime numerator and denominator pairs called a Farey Sequence (A close cousin of the Stern-Brocot tree). Using Integer Division, these coprime pairs can be used to quickly sum how many similar triangles are less than the perimeter. A more lengthy description can be found in this PDF file: Solving Project Euler.net Problem 370 on an HP calculator. This PDF also includes Bar codes for HP41, .RAW file if you want to use it in an Free42S emulator, commented HP42S Code, and a C program if you wish to compile on a PC.

Some example Run Times:

F(N)	41C	32sii	42s	Free42(D)	Result	Iterations	Time increase
F(100)	9s	2s	6s	0s	42	7	-
F(1000)	72s	15s	45s	0s	532	64	9.142857143
F(1e4)	651s	145s	426s	0s	6427	619	9.671875
F(1e5)	-	-	-	0s	75243	6262	10.11631664
F(1e6)	-	-	-	1.5s	861805	62710	10.0143724
F(1e7)	-	-	-	5s	9712598	626180	9.985329294
F(1e8)	-	-	-	60s	108073540	6262634	10.00133189

Note: in MATLAB or C the sum of times for F(2) to F(8)) is around 0.35 seconds

The time increase approaches 10X for a 10X increase in F(N) so it's almost exactly O(N) all the way up to higher orders.

Here is the commented HP2S code. It uses 10 registers: (1 for the Total, 1 for the order of the sequence, 1 for Phi, the golden ratio, 1 temporary register to simplify calculations, and 6 registers to iterate the Farey Sequence.)

1. Lines 1-22 set up 10 registers for Main loop
2. Lines 24-34 find the smallest similar Triangle with sides a and b
3. Lines 36-62 Creates the next Farey sequence.
4. Lines 63-66 Check to exit if m2/n2> golden ratio

```
00 { 72-Byte Prgm }     Comment
01>LBL 01       Lines 1-22 set up 10 registers for Main loop
02 STO 00       p=Perimeter
03 3
04 ÷
05 SQRT         Set Order for Recursive Farey Series
06 IP
07 STO 01       =floor(sqrt(P/3))
08 SIGN         save a byte to get 1 on the stack
09 STO 03       mt=1 - Initial Farey upper bound mt/nt=1/1
10 STO 04       nt=1
11 STO 05       m2=current side a=1
12 STO 06       n2=current side b=1
13 STO 07       n1=1
14 5
15 SQRT
16 +
17 2
18 ÷
19 STO 02       Phi Golden Ratio (the upper limit for the ratio of mt/nt)
20 CLX
```

```
21 STO 08       m1=0  Initial Farey lower bound m1/n1=0/1
22 STO 09       Total triangles= 0
23>LBL 02       Main loop
24 RCL 00       Lines 24-34 find the smallest similar Triangle with sides a and b
25 RCL 05
26 RCL 06
27 +
28 X^2          The smallest perimeter is ps=(a+b)^2-ab
29 RCL 05
30 RCL 06
31 ×
32 -
33 ÷
34 IP
35 STO+ 09      Total=total+floor(p/ps) [if ps>p this adds 0- no branching test needed]
36 RCL 07       Lines 36-62 Create the next Farey sequence.
37 RCL 01         This guarantees the numbers mt and nt are coprime and thus similar triangles are not
38 +              counted twice.   Example: For the trivial case where the perimeter p=100,
39 RCL 06         the order 5 Farey series is: 0/1, 1/1, 6/5, 5/4, 4/3, 7/5, 3/2, 8/5…
40 ÷             Contributing 0+33+1+1+2+0+5+0= 42 respectively.
41 IP            h=floor((n1+d)/n2)
42 ENTER
43 ENTER
44 RCL 05
45 ×
46 RCL 08
47 -
48 STO 03       mt=h*m2-m1
49 X<>Y
50 RCL 06
51 ×
52 RCL 07
53 -
54 STO 04       nt=h*n2-n1
55 RCL 06
56 STO 07       n1=n2
57 RCL 05
58 STO 08       m1=m2
59 RCL 03
60 STO 05       m2=mt
61 RCL 04
62 STO 06       n2=nt
63 ÷
64 RCL 02
65 X>Y?         Lines 63-66 Check to exit if m2/n2> golden ratio
66 GTO 02       Next Farey sequence
67 RCL 09       Otherwise recall total (END)
68 .END.
```

For F(2.5e13) the C program will get the solution to Euler Problem 370 eventually, but since it's O(N) this takes a long time. Nonetheless, because of the relatively small size program and minimal data requirements, finding Similar triangles using the Farey Sequence is reasonable for smaller orders of N. For these reasons, I believe this approach is is moderately suited for many values of N using a real or emulated HP calculator.

Many thanks to those who contributed solutions, you have some clever and compact solutions, all of which I studied and learned from.

Edited: 16 Mar 2012, 9:37 p.m.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #28 Posted by **Oliver Unter Ecker** on 17 Mar 2012, 8:02 a.m.,*
in response to message #27 by Allen

Congrats, Allen, this is a really beautiful solution!

For fun I coded it up in RPL+ and ended up with this:

```
\<< =n
   0 =:m1 =sum
   1 =:m2 =:n1 =n2
   floor(sqrt(n/3)) =d
   goldenRatio EVAL =gr


   WHILE m2/n2<=gr
   REPEAT
      floor(n/((m2+n2)*(m2+n2)-m2*n2)) =+sum
      floor((n1+d)/n2) =h
      h*m2-m1 =mt
      h*n2-n1 =nt
      n2 =n1 nt =n2
      m2 =m1 mt =m2
   END
   sum
\>>
```

goldenRatio is a built-in continued fraction. It's EVAL'ed in order to yield a real.

Run-times:
ND1 (iPhone): F(1000): <0.1s; F(1e6): 35s
CalcPad (Mac): F(1000): 0.007s; F(1e6): 1.8s

I used your C code to arrive at this in JavaScript:

```
function (n) { /*as is*/
   var sum=0, maxp = n;
   var mt=1, nt=1, m2=1, n1=1, n2=1, h, m1=0;
   var d = Math.floor(Math.sqrt(maxp/3));
   var gr = 1.6180339887498948482;


   while (m2/n2 <= gr ) {
     sum += Math.floor(maxp/((m2+n2)*(m2+n2)-m2*n2));
     h = Math.floor((n1+d)/n2);
     mt = h * m2 - m1;
     nt = h * n2 - n1;
     n1 = n2;
     n2 = nt;
```

```
    m1 = m2;
    m2 = mt;
  }


  return sum;
}
```

Run-times:
ND1 (iPhone): F(1000): 0.003s; F(1e8): 32s
CalcPad (Mac): F(1000): instant; F(1e8): 3.6s

My goal for RPL+ is to eventually achieve roughly half of JavaScript speed.

Edited: 17 Mar 2012, 8:03 a.m.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #29 Posted by **Valentin Albillo** on 17 Mar 2012, 12:01 p.m.,*
in response to message #27 by Allen

Hi again, Allen:

> Quote:
>
> Valentin, C.Ret, Gerson- thank you for your interesting solutions. I hope I was clear in saying there is nothing wrong with the naive approach- only an observation that it is the easiest solution to develop and program. Your solutions are certainly well thought out and not "naive" as in a bad.

Well, I wouldn't say my 6-line HP-71B solution above is that naïve in any sense good or bad but I get your point. As kinda proof-of-concept, this is the exact same solution directly converted to just 5 lines of interpreted UBASIC code almost *verbatim*:

```
10  word -3 : point -2 : for I=2 to 8 : K=10^I : P=4*K : S=0 : F=(1+sqrt(5))/2 : clr time
20  for A=1 to K\3 : N=A : M=N : E=0
30  D=prmdiv(N) : if D>1 then N\=D : if E=D then M\=D : E=0 : goto 30 else E=D : goto 30
40  S+=int(min(A*F,A*((sqrt(P/A-3)-1)/2))/M)-A\M+(res=0) : next
50  print I,S,time : next : print "Yasta"
```

Upon running it, the results and execution times are, in the same hardware:

```
run

 2   42          0:00:00
 3   532         0:00:00
 4   6427        0:00:00
```

```
   5   75243       0:00:00
   6   861805      0:00:01
   7   9712598     0:00:23
   8   108073540   0:04:47


Yasta
```

which produces all the results you asked for in reasonable times, about two full orders of magnitude (~100x) faster than Emu71, and of course still O(n*log(n)) [4:47 / 00:23 = 12.5x]. A conversion to compiled C# code produces almost two additional orders of magnitude of speed increase.

> Quote:
>
> Many thanks to those who contributed solutions, you have some clever and compact solutions, all of which I studied and learned from.

You're welcome and again, thanks for both your interesting challenge and your detailed, well-documented solution.

Best regads from V.

Edited: 17 Mar 2012, 12:08 p.m.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #30 Posted by **Gerson W. Barbosa** on 17 Mar 2012, 5:22 p.m.,*
in response to message #27 by Allen

Nice chALLENge and even nicer solution!
While I was watching a BBC documentary about Henry the Meek, I decided to try F(100000) on the real HP-42S. The calculator bravely accomplished the task just before the second episode was over.
Let me present an alternative G(N) function for those of you who are in a hurry and can tolerate some error:

```
00 { 119-Byte Prgm }
01>LBL "T"
02 STO 00
03 3
04 BASE÷
05 RCL 00
06 2
07 ÷
08 SQRT
09 BASE+
10 RCL 00
11 LN
12 RCL× ST L
13 4.76115738459E-2
```

```
14 ×
15 1.29705734564E-1
16 RCL× 00
17 -
18 47.8815
19 -
20 +
21 RCL 00
22 SQRT
23 LASTX
24 LN
25 8.60090977995E-2
26 ×
27 1.50150179296
28 -
29 ×
30 53.456
31 +
32 BASE+
33 .END.
```

N	F(N)	G(N)	absolute error	percent error
1E+02	42	43	+1	2.381
1E+03	532	530	-2	0.376
1E+04	6427	6426	-1	0.016
1E+05	75243	75244	+1	0.001
1E+06	861805	861805	0	0.000
1E+07	9712598	9712232	-366	0.004
1E+08	108073540	108074423	+883	0.001
1E+09	1190212172	1190326147	+113975	0.010
1E+10	12996874312	12999364614	+2490302	0.019

Less than one second on the real HP-42S :-)

Gerson.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #31 Posted by **Valentin Albillo** on 17 Mar 2012, 8:57 p.m.,*
in response to message #30 by Gerson W. Barbosa

Hi, Gerson:

> Quote:
>
> Let me present an alternative G(N) function for those of you who are in a hurry and can tolerate some error:

Good idea, here's my take on the subject for the HP-71B:

```
1  DESTROY ALL @ OPTION BASE 1 @ DIM C(8) @ READ C @ INPUT K @ FIX 0
2  DATA 2.23467244246E-8,-1.76140742602E-6,5.78605330267E-5,-1.04128697469E-3
3  DATA 1.14617819796E-2,-8.37891596549E-2,2.79672680182,-1.59739297094
4  P=LGT(K) @ N=C(1) @ FOR I=2 TO 8 @ N=N*P+C(I) @ NEXT I @ DISP EXP(N)
```

The results are:

```
 P      Exact value  Computed val. Error   % error
-------------------------------------------------
 1E02           42           42        0   -0.001
 1E03          532          532        0   +0.001
 1E04         6427         6427        0   -0.001
 1E05        75243        75242       +1   +0.001
 1E06       861805       861813       -8   -0.001
 1E07      9712598      9712504      +94   +0.001
 1E08    108073540    108074589    -1049   -0.001
 1E09   1190212172   1190200618   +11554   +0.001
 1E10  12996874312  12997000479  -126167   -0.001
```

and of course it runs almost instantaneously.

Best regards from V.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #32 Posted by **Gerson W. Barbosa** on 17 Mar 2012, 11:02 p.m.,*
in response to message #31 by Valentin Albillo

Hi, Valentin!

Great fit! What about fitting H(N) = F(N) - (N DIV 3 + INT(SQR(N/2))) and then making G(N) = N DIV 3 + INT(SQR(N/2)) + H(N)? Would this result in an even better fit?

Best regards,

Gerson.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #33 Posted by **Vassilis Prevelakis** on 17 Mar 2012, 11:11 p.m.,*
in response to message #27 by Allen

Not only a 42S program! It runs with almost no change on my HP-67.

The only thing I needed to change was to replace SIGN on line 8 with 1.

It calculated F(1000) in 2 minutes 26 sec!

**vp

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #34 Posted by **C.Ret** on 18 Mar 2012, 7:25 p.m.,*
in response to message #27 by Allen

Thank you for this nice challenge.

And felicitations for your brilliant solution. Using Farey series is a much more elegant solution that the crude and brut way to determine coprimes factors from a or b the way I made it!

I am studying your solution with great interest. The use of the coprime factors use from Farey the fractions is really stricky. But I still miss why or how to select range of the serie. Why are FLOOR(SQRT(P)) appropriate ? Is it a kind of magic ? lol I can't figure out how to relay this with the problem!

I can't resist to present here an old-style user-RPL version using your clever Farey's tricks: It is close-looking to RPL+ version above. Except that I have use more stack manipulations to spare local variables (and speed up runs on my poor old HP-28S).

42 (6s) 532 (20s) 6427 (1min30s)

```
« DUP                          @ Preserve P
  3 / SQRT FLOOR               @ Set Order for Recursive Farey Series
  1 5 SQRT + 2/                @ golden ratio
  -> P d gr
  « 0                          @ Initiate Sum
    0 1 1 1                    @ m1 n1 m2 n2 leave and treated in stack
    WHILE DUP2 / gr <          @ test  m2/n2 < gr
    REPEAT
       DUP2 * LAST + SQ SWAP -
       P SWAP / FLOOR          @ FLOOR(P/((a+b)²-a*b))
       6 ROLL + 5 ROLLD        @ add to sum and ROLLD back to top
       3 PICK d + OVER / FLOOR @  -> h
       3 PICK OVER * 6 ROLL -  @ compute new mt (and remove m1 from stack)
       SWAP 3 PICK * 5 ROLL -  @ compute new nt (and remove n1 from stack)
    END
    4 DROPN                    @ remove m1 n1 m2 n2 from stack (Sum remain on top)
   »
»
```

And again, thank to you for this nice challenge and for other forumers to participate. Great Time !

EDIT: remove swapped >> from code listing.

Edited: 19 Mar 2012, 12:00 p.m. after one or more responses were posted

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #35 Posted by **Allen** on 18 Mar 2012, 8:40 p.m.,*
in response to message #34 by C.Ret

You are most welcome- I enjoyed working on this problem with what time I had.

You raise a good point about how to get the order of the Farey sequence. I had two challenges, first the Sequence normally covers {0..1},but I needed {1..phi} for reasons already discussed.

The second challenge, determining the order = floor(sqrt(p/3)) was not as trivial. In examining the data for the smaller O(N^2) runs, the irreducible coprime factors came up. I noticed that in every case F(N), the boundary closest to unity: k+1/k always stopped at a certain value of k_max (and that no greater denominator existed anywhere in the set):

```
  N         k_max
------------------
  100         5
  1000       18
  10000      57
```

And so on. Since I already came to the same conclusion many others had about the importance of "P/3" in determining the upper limit for side A, It did not take much guess work to identify SQRT(P/3) as an interesting number in the Farey sequence, and the FLOOR function simply ensured the remainder of the denominators were integers.

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #36 Posted by **Werner** on 21 Mar 2012, 4:33 a.m.,*
in response to message #35 by Allen

```
If a <= b <= c
and n/d is the irreducible fraction of b/a,
then the smallest triangle you can construct (subject to b^2 = a*c) is
(a,b,c) = (d^2,n*d,n^2)


because


 b = k*n
 a = k*d


for k a positive integer, then


 b^2 = a*c
 k^2 * n^2 = k*d*c


 c = k*n^2/d
```

```
 Since n and d are coprime, k >= d


Since a <= P/3, it follows that
      d <= SQRT(P/3)
```

Werner

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #37 Posted by **Allen** on 21 Mar 2012, 6:38 a.m.,*
in response to message #36 by Werner

Werner, A very nice derivation, thank you!

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #38 Posted by **Reth** on 19 Mar 2012, 6:22 a.m.,*
in response to message #34 by C.Ret

Quote:

```
« DUP                          @ Preserve P
  3 / SQRT FLOOR               @ Set Order for Recursive Farey Series
  1 5 SQRT + 2/                @ golden ratio
  -> P d gr
  « 0                          @ Initiate Sum
    0 1 1 1                    @ m1 n1 m2 n2 leave and treated in stack
    WHILE DUP2 / gr <          @ test  m2/n2 < gr
    REPEAT
       DUP2 * LAST + SQ SWAP -
       P SWAP / FLOOR          @ FLOOR(P/((a+b)²-a*b))
       6 ROLL + 5 ROLLD        @ add to sum and ROLLD back to top
       3 PICK d + OVER/ FLOOR  @  -> h
       3 PICK OVER * 6 ROLL -  @ compute new mt (and remove m1 from stack)
       SWAP 3 PICK * 5 ROLL -  @ compute new nt (and remove n1 from stack)
       »
    END
    4 DROPN                    @ remove m1 n1 m2 n2 from stack (Sum remain on top)
»
```

Are you sure this code will work? Aren't >> and END swapped over?

Re: O(N) solution for "A Sunday Programming Challenge"

*Message #39 Posted by **C.Ret** on 19 Mar 2012, 11:54 a.m.,*
in response to message #38 by Reth

You are right, the \>> swap here from a bad copy-paste operation!

Corrected listing:

```
« DUP                              @ Preserve P
  3 / SQRT FLOOR                   @ Set Order for Recursive Farey Series
  1 5 SQRT + 2/                    @ golden ratio
  -> P d gr
  « 0                              @ Initiate Sum
    0 1 1 1                        @ m1 n1 m2 n2 leave and treated in stack
    WHILE DUP2 / gr <              @ test  m2/n2 < gr
    REPEAT
       DUP2 * LAST + SQ SWAP -
       P SWAP / FLOOR              @ FLOOR(P/((a+b)²-a*b))
       6 ROLL + 5 ROLLD            @ add to sum and ROLLD back to top
       3 PICK d + OVER / FLOOR     @  -> h
       3 PICK OVER * 6 ROLL -      @ compute new mt (and remove m1 from stack)
       SWAP 3 PICK * 5 ROLL -      @ compute new nt (and remove n1 from stack)
    END
    4 DROPN                        @ remove m1 n1 m2 n2 from stack (Sum remain on top)
   »
»
```

Sorry.

Edited: 19 Mar 2012, 12:03 p.m.

Re: A Sunday Programming Challenge

*Message #40 Posted by **C.Ret** on 15 Mar 2012, 10:19 a.m.,*
in response to message #24 by Allen

I am really glad to learn that my first approach doesn't below to the first naïve category and is consider as 'reduce naïve'. Because, in my point of view, my first code is really a "brute force scan", since very few optimization in the search strategy is done. Two loops imbedded. By luck the outer one scan step by step on *a* which is fortunately the shortest side of the triangle, make it walk trough it as short as possible because limited to (P/3) ! How Lucky I am here! On the other hand, the inner loop (which will weight much more of the time to find the solution) is far for a good optimization. Despite the two exit tests which appropriately short-cut the long b walk through, the worst in my code is that at b is scan step by step (only increase by one). Here I am bad. This bad strategy is perhaps not a shame for small perimeter, but will be a disaster for larger perimeter.

As a lot of Euler problems, here a more tricky strategy have to be take into account, to avoid this very long systematic scans of "brute force like" algorithm.

There must be a way to scan over *a* only, not over *b* !

What are the constraints:
(1) Sorted Sides: $a <= b <= c$
(2) Being a triangle $c <= a + b$
Since other inequalities are both verified:
Due to $a < b$ we get $a <= b + n$ whatever is the natural integer n. Due to initial condition (1) $b <= c$ so we get b <= c + n for any natural integer n, thus

$b <= c + a$. As long as initial condition (1) is respected, we only have to care about $c <= a + b$
(3) Limited perimeter $a+b+c <= P$ As Marcus Von Cube pointed it out, this constraint has to be tested. But generally it is less constringent than (2). (4) Specific condition: $b^2 = a.c$ Here is the key to reduce

Suppose that b decompose into the following product of primary factors : b_1, b_2, b_3…

With $b = b_1. b_2. b_3...$ then (4) implies that a and c are both a subset of the primary factors of b.

To illustrate that, investigate all the triplets *(a,b,c)* that verify (4). Only a fraction of this list of triplets corresponds to constraints (1) (2) and (3):

```
Looking for triangle P<=100
b = 2 x 2 x 3 = 12 scanning all (a,c) pairs built up table 1:


TABLE 1:
========
    a    b    c   P=a+b+c Violate constraints
----- ---- ---- --------- ----------------------------------------
    1   12  144     157   (2)(3)      Not a triangle + P Too large
    2   12   72      86   (2)         Not a triangle
    3   12   48      63   (2)         Not a triangle
    4   12   36      52   (2)         Not a triangle
    6   12   24      42   (2)         Not a triangle
    8   12   18      38               Ok n°1
    9   12   16      37               Ok n°2
   12   12   12      36               Ok n°3
   16   12    9      37   (1)         a b c not sorted
   18   12    8      38   (1)         a b c not sorted
   24   12    6      42   (1)         a b c not sorted
   36   12    4      52   (1)         a b c not sorted
   48   12    3      63   (1)         a b c not sorted
   72   12    2      86   (1)         a b c not sorted
  144   12    1     157   (1)(3)      a b c not sorted + Too large P
----- ---- ---- --------- ----------------------------------------
```

We can observe : - the symmetry role of a and c, scanning over a or c is equivalent. - half of the triplet have $a>b$ (i.e. $c<b$), scan can be reduce considerably (for example by only scanning over a from 1 to b) - not every $a$value have to be scan. In fact, only value of a composed using a subset of the primary factor from b^2 :

```
Table 2 :
=========
b  =  2 x 2 x 3           b.b =  2 x 2 x 2 x 2 x 3 x 3
------------------------ ------------------------------
a  =  1                     c =  2 x 2 x 2 x 2 x 3 x 3 = 144
a  =  2                     c =      2 x 2 x 2 x 3 x 3 =  72
a  =  3                     c =  2 x 2 x 2 x 2 x 3     =  48
a  =  2 x 2                 c =          2 x 2 x 3 x 3 =  36
a  =  2 x 3                 c =      2 x 2 x 2 x 3     =  24
a  =  2 x 2 x 2             c =              2 x 3 x 3 =  18
a  =  3 x 3                 c =  2 x 2 x 2 x 2         =  16
```

```
a  =  2 x 2 x 3             c =          2 x 2 x 3     =  12
...
```

This prove that scanning step by step over *a* or *b* is not the trickiest way to look for such triangle.

This suggest that an algorithm based on combination of a prime factors decomposition of *b*can be more adapt to this Week-End challenge.

On the other hands, having to scan over the smallest side *a* will be much simpler and allow the uses of short-cut tests. I am on the way thinking that investigating how to built the correct *b* side from a given *a* under constrains (1) to (4) will be an elegant and surely efficient method. Of course, the over-head need to analyze *a* and built up *b* will only be benefic for scanning over large Perimeter and large side. But the great interest would be that no more inner loop over *b* will be needed.

To investigate how to construct *b* from a given *a*, please consider a first example: Looking for geometric triangle responding to constraints (1) to (4) with perimeter less or egal to P = 100. Considering *a* prime :

```
Table 3:   prime  a = 7  , P <=100
=======
    a    b    c   P=a+b+c Violate constraints
----- ---- ---- --------- ----------------------------------------
    7    7    7      21   none                OK
    7   14   28      49   (2)                 not a triangle (c>a+b)
    7   21   63      91   (2)                 not a triangle
    7   28  112     147   (2)(3)              not a triangle & too large
    7   35  175     217   (2)(3)              not a triangle & too large
    and so on...
----- ---- ---- --------- ----------------------------------------
```

Only first triplet (7,7,7) correspond to the researched geometric triangle. As soon as the second line, triplets violate one or more constrains, so I may have stop the scan as soon. Putting more values in the table shows how *b* value are build. As *a* is prim, there is no combination of factors is possible, and we observe that *b* is multiple of *a* The triplets are of the general formulae *(a , k.a , a.k^2)* with *k=1, 2, 3, ...*

Now, what if a is a simple composite: *a* = 8

```
Table 4:   simple composite a = 8 , P <=100
=======
    a    b    c   P=a+b+c Violate constraints
----- ---- ---- --------- ----------------------------------------
    8    8    8      24   none                OK
    8   12   18      38   none                OK
    8   16   32      56   (2)                 not a triangle
    8   20   50      78   (2)                 not a triangle
    8   24   72     104   (2)(3)              not a triangle & too large
    8   28   98     134   (2)(3)              not a triangle & too large
    and so on...
----- ---- ---- --------- ----------------------------------------
```

Here, two triangles of small side 8 exist (8,8,8) and (8,12,18). Both candidates respect all the constraints.

If we consider the *b* set , we now observe an arithmetic suite *a*, *a+4*, *a+2x4*, *a+3x4*,… At each rwo, we have to increase *b* by 4. Where come this value from? Is there any chance that 4 is built from half of the prime factors of *a* ? *a = 1x2x2x2 = 8* , the delta D to use to built the set of *b* is *D = 2x2 = 4*

The triplets are of the form *(a, a+D.k, c).*, where *k = 0, 1, 2, ...*

Now, what if a is *a[italic] square: [italic]a* = 9

```
Table 5:   simple composite a = 25 , P <= 100
=======
    a    b    c   P=a+b+c Violate constraints
----- ---- ---- --------- ----------------------------------------
   25   25   25       75   none               OK
   25   30   36       91   none               OK
   25   35   49      109   (3)                perimeter too long
   25   40   64      129   (3)                perimeter too long
   25   45   81      151   (2)(3)             not a triangle & too long
   25   50  100      175   (2)(3)             not a triangle & too long
   25   55  121      201   (2)(3)             not a triangle & too long
    and so on...
----- ---- ---- --------- ----------------------------------------
```

At each row, *b* have to be increase by *D = 5* (not 25). So we have to construct *D* as small as possible with "half" of the factors of *a*. Triplets are of the form $(a^2, a^2+k.a, (a+k)^2)$.

In fact, *D* has to be built from each factor of *a*, but using only half of the multiple factors (i.e; with half to power of each factor).

```
Table 6:   simple composite a = 42 , P <= 500
=======
    a    b    c   P=a+b+c Violate constraints
----- ---- ---- --------- ----------------------------------------
   42   42   25      126   none               OK
   42   84  168      294   (2)                not a triangle
   42  126  378      546   (2)(3)             nat & perimeter too long
   42  168  672      882   (2)(3)             nat & perimeter too long
----- ---- ---- --------- ----------------------------------------
```

Here *a = 1x2x3x7 = 42* , and *b = a + 42.k* with *k = 0, 1, 2, ...* This confirming that *D* has to be built from each factor of *a*, but using only half of the multiple factors (i.e; with half to power of each factor).

```
Table 7:   Samples of D from factor decomposition of a
=======
    a                   D              |   a                   D              | a                       D
-----   ------------ ----- -------------|-----   ------------ ----- -------------|-----   ------------ ----- -------------
    1   1x1              1              |   31   1x31             31             |   61   1x61            61
    2   1x2              2              |   32   1x2x2x2x2x2       8  = 2x2x2    |   62   1x2x31          62  = 2x31
    3   1x3              3              |   33   1x33             33             |   63   1x63            63
    4   1x2x2            2  =  2        |   34   1x2x17           34  = 2x17     |   64   1x2x2x2x2x2x2    8  = 2x2x2
    5   1x5              5              |   35   1x5x7            35  = 5x7      |
    6   1x2x3            6  = 2x3       |   36   1x2x13           36  = 2x2x2    |
    7   1x7              7              |   37   1x37             37             |
    8   1x2x2x2          4  = 2x2       |   38   1x2x19           38  = 2x19     |
    9   1x3x3            3  =  3        |   39   1x3x13           39  = 3x13     |
   10   1x2x5           10  = 2x5       |   40   1x2x2x2x5        20  = 2x2x5    |
   11   1x11            11             |   41   1x41             41             |
   12   1x2x2x3          6  = 2x3       |   42   1x2x3x7          42  = 2x3x7    |
   13   1x13            13             |   43   1x42             43             |
```

14	1x2x7	14	= 2x7	44	1x2x2x11	22	= 2x11
15	1x3x5	15	= 3x5	45	1x3x3x5	15	= 3x5
16	1x2x2x2x2	4	= 2x2	46	1x2x23	46	= 2x23
17	1x17	17		47	1x47	47	
18	1x2x3x3	6	= 2x3	48	1x2x2x2x2x3	12	= 2x2x3
19	1x19	19		49	1x7x7	7	= 7
20	1x2x2x5	10	= 2x5	50	1x2x5x5	10	= 2x25
21	1x3x7	21	= 3x7	51	1x3x17	51	= 3x17
22	1x2x11	22	= 2x11	52	1x2x2x13	26	= 2x13
23	1x23	23		53	1x53	53	
24	1x2x2x2x3	12	= 2x2x3	54	1x2x3x3x3	18	= 2x3x3
25	1x5x5	5	= 5	55	1x5x11	55	= 5x11
26	1x2x13	26	= 2x13	56	1x2x2x2x7	28	= 2x2x7
27	1x3x3x3	9	= 3x3	57	1x3x19	57	= 3x19
28	1x2x2x7	14	= 2x7	58	1x2x29	58	= 2x29
29	1x29	29		59	1x59	59	
30	1x2x3x5	30	= 2x3x5	60	1x2x2x3x5	30	= 2x3x5

The larger is *D*, the more rapid is the scan. As we can see, the process (involving the two embedded loops) has a great chance to be speed-up, since most of the *a* value leads to $D = a$. Especially when *a* is prime.

The inner loop (the one over *b*) can be suppress if we found a way of forecasting at which *k* we have to stop because the triplet violate one (or more) constraints.

To test for constraint (2) : *(a, b, c)* being a triangle, we have to test $c >= a + b$

From the given *a* value, we may get *D* (see table 7 above) or find a way to built *D* from *a* factors. With such a D, we know that the triplet (a, a+k.D, c) respect (4) $b^2 = a.c$ when $b = a + k.D$, therefore $c = b^2 / a$

To test for constrinat (2), we have to express $c >= a + b$ as a function of variable *k* and parameters *a* and *D*.

The inequality $c >= a + b$ can be transform into $k^2.D^2 + k.D.(2-a) - a^2 <= 0$. Resolving the corresponding quadratic equation, allow to indicate the domain of validity:
The variable have to be set from 0 up to $k_2 = -(a/2D).(1-V5)$

To test for constraint (3) : $a+b+c <= P$, we may found the limit for k the same way, by expression the inequality as a function of variable k and parameters *a*, *P* and *D* $a+b+c <= P$ is transform into $k^2.D^2 + k.D.(2+a) + 3a^2-aP <= 0$. Solving the corresponding quadratic equation for *k* indicate that *k* may varie from 0 upto $k_3 = -(a/2D).(3-V(4P/a - 3))$

This give you the way to code for a more efficient algorithm using only one loop over *a* :

```
Input   P  upper limit of perimeter
Initiate Sum = 0
For each integer a from 1 to P/3
        Determine D  ( 1 <= D <= a ) from the prime factor decomposition of a.
        In most of the cases D will be equal to a
          or less than a in specific cases
```

```
        IF P > (3+SQR(5))*a
        THEN   ' Limited by triangle (2)
                        kl = -(a/2D)*(1-SQR(5))
        ELSE   ' Limited by perimeter (3)
                        kl = -(a/2D)*(3-SQR(4P/a - 3))
        End if
        There is n = INT( 1 + kl ) geometric triangles of small side [italic]a[/italic] since k = 0,1, 2, ..., kl
        Sum = Sum + n         Add integer n into the Sum


At end, display Sum that contains number of researched triangle of perimeter less or equal to P.
```

In conclusion, I leave you a few to compose you own code on your favorite calculator or system. I curious to know how much time this new algorithm loose on shot perimeter (P = 100, 1000), and what it is win for larger size (P> 10^4).

Edited: 15 Mar 2012, 7:26 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

Message #41 Posted by ***Gerson W. Barbosa*** *on 15 Mar 2012, 2:25 p.m.,*
in response to message #40 by C.Ret

> Quote:
>
> I am really glad to learn that my first approach doesn’t below to the first naïve category and is consider as ‘reduce naïve’. Because, in my point of view, my first code is really a “brute force scan”, since very few optimization in the search strategy is done.

That is exactly my feelings, regarding mine. I haven't timed your first approach, but mine isn't any better than Valentin's, quite the contrary:

```
10 DESTROY ALL @ INPUT L @ N= L DIV 3 @ K=(SQR(5)-1)/2
20 SETTIME 0 @ FOR B = N TO 2 STEP -1 @ A = B - 1
30 IF A < B * K THEN 90
40 C = B * B / A
50 IF FP(C) <>  0 THEN 70
60 IF A + B + C <= L THEN N = N + 1
70 A = A - 1
80 GOTO 30
90 NEXT B @ PRINT N,TIME


>RUN
? 100
 42                   .05
>RUN
? 1000
 532                  2.51
>RUN
? 10000
 6427                 255.22
```

Valentin's times in my notebook running EMU71 @ 1.86 GHz are:

```
>RUN
? 100
 42                   .05
>RUN
? 1000
 532                  1.2
>RUN
? 10000
 6427                 114.67
```

Gerson.

Re: A Sunday Programming Challenge

*Message #42 Posted by **C.Ret** on 15 Mar 2012, 3:06 p.m.,*
in response to message #41 by Gerson W. Barbosa

You may try this last version :

```
(Microsoft Q-BASIC)
DEFLNG A-Z          '  ---- All variables long integer except k and t (single precision)
DIM k AS SINGLE
DIM t AS SINGLE


PRINT
INPUT "ENTER perimeter"; P
IF P < 1 THEN END
'
'   Initiate Sum and timer
'
Sum = 0
t0 = TIMER
'
'  --------------------- MAIN LOOP OVER a
'
FOR a = 1 TO P / 3
    '
    ' ------------------------ Compute step D
    '
    x = a: i = 2: j = 1: D = 1
    '
    '  - - - - - - - - - - Isprime? Test Loop
    DO UNTIL i * i > x
        n = 0
        DO WHILE x MOD i = 0: n = n + 1: x = x / i: LOOP
        IF n > 0 THEN D = D * i ^ INT(.5 + n / 2)
        i = i + j: j = 2
    LOOP
    D = D * x
    '
    ' ----------------------- test Limit for k
    '
    IF P < (3 + SQR(5)) * a THEN
```

```
          k = SQR(4 * P / a - 3) - 3   ' Limited by perimeter
    ELSE
          k = SQR(5) - 1               ' Limited by triangle
    END IF
    '
    ' ----------------------- Add current number to sum
    '
    Sum = Sum + 1 + INT(a / 2 / D * k)
NEXT a
'
'   ------------------------ Display result and time
'
PRINT : PRINT
PRINT "There is "; Sum; "triangles of perimeter <="; P; " with b²=a.c"
PRINT "("; INT((TIMER - t0) * 10) / 10; "s )"
PRINT
```

Approximetely translate into HP-71 BASIC : not sure about correct syntax of MOD CEIL and power (^).

```
 10 DESTROY ALL @ INPUT P @ SETTIME 0


   REM -------------  MAIN LOOP OVER A


 20 S = 0 @ FOR A = 1 TO P DIV 3


       REM -------------  DETERMINE STEP D


 30    X = A @ I = 2 @ D = 1 @ J = 1


       REM - - - - - - - ISPRIME TEST LOOP


 40    N = 0 @ IF I * I > X THEN 90
 50       IF X MOD I = 0 THEN N = N + 1 @ X = X DIV I @ GOTO 50
 60       IF N > 0 THEN D = D * I ^ CEIL( N / 2 )
 70       I = I + J @ J = 2
 80    GOTO 40


      REM -------------  COUNT SOLUTIONS


 90    D = D * X @ K = 1 - SQRT 5
100    IF P < (3 + SQRT 5) * A THEN K = 3 - SQRT(4 * P / A - 3)
110    S = S + 1 + INT( - A / 2 / D * K)
120 NEXT A @ PRINT S,TIME
```

Edit : Replace N by S at line 120 in HP71 listing.

Edited: 15 Mar 2012, 6:53 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #43 Posted by **Gerson W. Barbosa** on 15 Mar 2012, 6:13 p.m.,*
in response to message #42 by C.Ret

It looks like there is an error in your conversion from QBASIC to HP-71 BASIC or I have introduced one myself:

```
10 DESTROY ALL @ INPUT P @ SETTIME 0
20 S=0 @ FOR A=1 TO P DIV 3
23 REM
25 REM -------------  determine step d
27 REM
30 X=A @ I=2 @ D=1 @ J=1
33 REM
35 REM - - - - - - - isprime test loop
37 REM
40 N=0 @ IF I*I>X THEN 90
45 N=0
50 IF MOD(X,I)=0 THEN N=N+1 @ X=X DIV I @ GOTO 50
60 IF N>0 THEN D=D*I^CEIL(N/2)
70 I=I+J @ J=2
80 GOTO 40
83 REM
85 REM -------------  count solutions
87 REM
90 D=D*X @ K=1-SQR(5)
100 IF P<(3+SQR(5))*A THEN K=3-SQR(4*P/A-3)
110 S=S+1+INT(-A/2/D*K)
120 NEXT A @ PRINT N,TIME


>RUN
? 1000
 0                    .32
>RUN
? 10000
 0                    5.1
>RUN
? 100000
 0                    102.21
```

The QBASIC times are:

```
    F(100000) -->   1.4 s
   F(1000000) -->  30.6 s
  F(10000000) --> 567.8 s
```

Re: A Sunday Programming Challenge

*Message #44 Posted by **C.Ret** on 15 Mar 2012, 6:40 p.m.,*
in response to message #43 by Gerson W. Barbosa

Yes, I made a mistake in the last line where N stand for S. Sorry.

```
10 DESTROY ALL @ INPUT P @ SETTIME 0 @ S=0 @ FOR A=1 TO P DIV 3 @  X=A @ I=2 @ D=1 @ J=1
40 N=0 @ IF I*I>X THEN 90
50 IF MOD(X,I)=0 THEN N=N+1 @ X=X DIV I @ GOTO 50
60 IF N>0 THEN D=D*I^CEIL(N/2)
70 I=I+J @ J=2 @ GOTO 40
90 D=D*X @ K=SQR(5)-1 @ IF P<(3+SQR(5))*A THEN K=SQR(4*P/A-3)-3
110 S=S+1+INT(A/2/D*K) @ NEXT A @ PRINT S,TIME
```

Here the same code for "old" RPL aka HP-28C/S. "New PRL" may take advantage of ISPRIME? and NEXT-PRIME instructions !

```
« -> P
   « 0                                    @ Initiate Sum=0
     1 P 3 / FOR a
         1 SF
         2                                @ Initiate D = 2
         a                                @          x = a
         2                                @          i = 2
         WHILE DUP2 SQ >=                 @ while i²<=x
         REPEAT
            0 3 ROLLD                     @    Initate  n = 0
            WHILE DUP2 MOD NOT            @    while x MOD i = 0
            REPEAT
               SWAP OVER / SWAP           @       x = x/i
               ROT 1 + 3 ROLLD            @       n = n+1
            END
            4 ROLL OVER 5 ROLL 2 /
            CEIL ^ * 3 ROLLD              @       D = D*i^CEIL(n/2)
            1 +                           @   Inc i
            IF 1 FC?C THEN 1 + END        @   Inc i when i>3
         END
         DROP *                           @   D = D*x
         a SWAP /
         IF P a 3 5 SQRT + * <
            THEN 4 P * a / 3 - SQRT 3 -   @ limited by perimeter
            ELSE 5 SQRT 1 -               @ limited by triangle side
         END
         * IP 1 + +                       @   Sum = Sum + 1 + INT(...)
      NEXT
   »
»  'B2AC' STO
```

Execution Times HP28S 100 B2AC ---> 42 (12s.) 1000 B2AC ---> 532 (3min45s)

Edited: 15 Mar 2012, 8:04 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #45 Posted by **Gerson W. Barbosa** on 15 Mar 2012, 7:26 p.m.,*
in response to message #44 by C.Ret

Here are your times up to 10^5:

```
  Emu71: HP-71B & HP-IL system emulator         J-F GARNIER 1996, 2006
>list
10 DESTROY ALL @ FOR E=2 TO 5 @ P=10^E @ SETTIME 0 @ S=0
20 FOR A=1 TO P DIV 3 @ X=A @ I=2 @ D=1 @ J=1
40 N=0 @ IF I*I>X THEN 90
50 IF MOD(X,I)=0 THEN N=N+1 @ X=X DIV I @ GOTO 50
60 IF N>0 THEN D=D*I^CEIL(N/2)
70 I=I+J @ J=2 @ GOTO 40
90 D=D*X @ K=SQR(5)-1 @ IF P<(3+SQR(5))*A THEN K=SQR(4*P/A-3)-3
110 S=S+1+INT(A/2/D*K) @ NEXT A @ PRINT P,S,TIME @ NEXT E


>run
 100                42                .05
 1000               532               .3
 10000              6427              4.81
 100000             75243             94.45
```

Re: A Sunday Programming Challenge

*Message #46 Posted by **C.Ret** on 15 Mar 2012, 8:05 p.m.,*
in response to message #45 by Gerson W. Barbosa

Thank you.

It's look like this version is not as fast as Valentin Albillo's code (see post #023) who is using machine ROM PRIM function.

My code loose much of his time to test for primility, where the PRIM function spare a lot of time.

As as explain above, most od the time, *D* equal *a*. In most of the case my code test all factors of *a* where a simple *D* = *a* will do the job.

Following a transcription of the above code into HP-32S RPN program :

```
00 { 141-Byte Prgm } :                    :
01>LBL "B2AC"        : 31 1               : 61 SQRT
02 STO 01            : 32 +               : 62 +
03 3                 : 33 GTO 02          : 63 ×
04 ÷                 : 34>LBL 03<         : 64 X<=Y?
05 IP                : 35 Rv              : 65 GTO 06
06 STO 02            : 36 X=0?            : 66 Rv
07 0                 : 37 GTO 04          : 67 4
08 STO 00            : 38 2               : 68 ×
09>LBL 00<-----------: 39 1/X             : 69 RCL÷ 02
10 2                 : 40 STO× ST Y       : 70 3
11 STO 03            : 41 +               : 71 STO- ST Y
12 RCL 02            : 42 IP              : 72 X<>Y
13 2                 : 43 RCL ST Y        : 73 GTO 07
14 SF 00             : 44 X<>Y            : 74>LBL 06<-------
15>LBL 01            : 45 Y^X             : 75 1
16 X^2               : 46 STO× 03         : 76 5
17 X>Y?              : 47>LBL 04<         : 77>LBL 07<-------
18 GTO 05            : 48 CLX             : 78 SQRT
```

```
19 X<> ST L          : 49 1                : 79 X<>Y
20 0                 : 50 FC?C 00          : 80 -
21>LBL 02<           : 51 STO+ ST Y        : 81 RCL× 02
22 RCL ST Z          : 52 +                : 82 RCL÷ 03
23 RCL÷ ST Z         : 53 GTO 01           : 83 IP
24 FP                : 54>LBL 05<----------: 84 1
25 X>0?              : 55 Rv               : 85 +
26 GTO 03            : 56 STO× 03          : 86 STO+ 00
27 Rv                : 57 RCL 01           : 87 DSE 02
28 X<>Y              : 58 RCL 02           : 88 GTO 00
29 STO÷ ST Z         : 59 3                : 89 RCL 01
30 X<>Y              : 60 5                : 90 RCL 00
                     :                     : 91 END


REgisters
R00 : Sum ,number of triangles
R01 :  P  ,perimeter
R02 :  a  ,small side
R03 :  D  ,


>LBL 00 : Main loop (For a = INT(P/3) downto 1


          Factors analysis of a


>LBL 01 : Look for factors of a
>LBL 02 : search for multiple factor of a
>LBL 03 : Built D by adding each factor of a (but half of each multiple factor number).
>LBL 04 : next factor f=2,3,5,...


          Compute number of triangle of small side a


>LBL 05 : Nbr of triangle limited by perimeter a+b+c < P
>LBL 06 : Nbr of triangle limited by side c < a+b
>LBL 07 : Add number of traingle of side a to global sum (R00)
```

Edited: 16 Mar 2012, 5:56 p.m.

Re: A Sunday Programming Challenge

*Message #47 Posted by **Oliver Unter Ecker** on 17 Mar 2012, 8:30 a.m.,*
in response to message #44 by C.Ret

Nice code as always, C.Ret!

> Quote:
>
> Execution Times HP28S 100 B2AC ---> 42 (12s.) 1000 B2AC ---> 532 (3min45s)

I ran it (unchanged) in ND1, yielding the following run-times: B2AC(100): 0.2s; B2AC(1000): 2.8s

It's not obvious to me how to change the code to use NEXTPRIME and/or ISPRIME?. Can you show me?

The 50g or ND1 also have FACTORS, which can be used to derive your "D". It's straightforward but not terribly elegant, as you'd still need a loop to do an integer divide on every second element in the factors array (which interleaves factors and their exponents).

Re: A Sunday Programming Challenge

*Message #48 Posted by **C.Ret** on 21 Mar 2012, 7:11 a.m.,*
in response to message #47 by Oliver Unter Ecker

Here is what the code using ISPRIME? and NEXTPRIME instruction may look at.

where ISPRIME? returns true (1) for prime numbers only. and NEXTPRIME converts its argument to the next prime number (ex. 1 NEXTPRIME is 2, 2 NEXTPRIME is 3, 3 NEXTPRIME is 5, 5 NEXTPRIME is 7 and so on...)

```
PROGRAM:                                  @ Comments                  @ Stack   5: 4: 3:  2:  1:
                                          @                           @                           P
« -> P
   « 0                                    @ Initiate Sum=0            @                 0 : 1 : P/3
     1 P 3 / FOR a                        @  Main Loop                @                           Sum
        2                                 @ Initiate D <- 2
        a                                 @          x <- a
        1                                 @          i <- 1           @           Sum : D : x :   i
        WHILE OVER ISPRIME? NOT           @ loop until x is prime
        REPEAT
           NEXTPRIME                      @    i' <- nextprime ( i )  @           Sum : D : x :   i
           0 3 ROLLD                      @    Initiate  n <- 0       @        Sum : D : n : x :   i
           WHILE DUP2 MOD NOT             @    while x MOD i = 0
           REPEAT
              SWAP OVER / SWAP            @       x <- x/i            @    Sum : D :  n  : x/i :   i
              ROT 1 + 3 ROLLD             @       n <- n+1            @    Sum : D : n+1 : x/i :   i
           END
           4 ROLL OVER 5 ROLL 2 /
           CEIL ^ * 3 ROLLD               @       D = D*i^CEIL(n/2)   @         Sum : D' : x   :   i
        END
        DROP *                            @   D = D*x                 @                  Sum : D*x
        a SWAP /                                                      @                 Sum : a/2D
        IF P a 3 5 SQRT + * <
           THEN 4 P * a / 3 - SQRT 3 -    @ limited by perimeter      @  Sum : a/2D : SQRT(4P/a-3)-3
           ELSE 5 SQRT 1 -                @ limited by triangle side  @  Sum : a/2D :      SQRT(5)-1
        END
        * IP 1 + +                        @  Sum' = Sum + 1 + INT(...)  @                       Sum'
     NEXT
   »
»                                                                     @                        Sum
```

The code is not far from the original one without ISPRIME? and NEXTPRIME. Time is spare since for large arguments, fewer tests or loops are needed to detemine all cofactor. Especely, no more loop are waste to test for composite (ex. i=9 is always

negative because i=3 was alreday tested, but no way !) Most of the time, D is close to a. With a large a great number of loops are no more waste. And when a is large and prim, no loop at all, immediately D=a is detected.

Time may be lost when ISPRIME? and NEXTPRIME are long time runs. This would be the case with my HP-28S if I have to programm ISPRIME? or NEXTPRIME as USER-RPL. Without built-in instruction using internal speed machine code, no gain.

But still, as demonstrate by **Allen**, finding cofactor using Farey Series is much more tricky and faster than determining D for each a value, since there is no more loops for cofactors determination or prime testing. The Farey Series directly give coprime b / a pairs and a few scale and end-test is only need :

```
« DUP                          @ Preserve P
  3 / SQRT FLOOR               @ Set Order for Recursive Farey Series
  1 5 SQRT + 2/                @ golden ratio
  -> P d gr
  « 0                          @ Initiate Sum
    0 1 1 1                    @ m1 n1 m2 n2 leave and treated in stack
    WHILE DUP2 / gr <          @ test  m2/n2 < gr
    REPEAT
       DUP2 * LAST + SQ SWAP -
       P SWAP / FLOOR          @ FLOOR(P/((a+b)²-a*b))
       6 ROLL + 5 ROLLD        @ add to sum and ROLLD back to top
       3 PICK d + OVER/ FLOOR  @  -> h
       3 PICK OVER * 6 ROLL -  @ compute new mt (and remove m1 from stack)
       SWAP 3 PICK * 5 ROLL -  @ compute new nt (and remove n1 from stack)
    END
    4 DROPN                    @ remove m1 n1 m2 n2 from stack (Sum remain on top)
  »
»
```

Edited: 21 Mar 2012, 8:03 a.m.

Re: A Sunday Programming Challenge

*Message #49 Posted by **Oliver Unter Ecker** on 21 Mar 2012, 8:24 a.m.,*
in response to message #48 by C.Ret

Thanks for this, C.Ret!

Your points about there being no speed gain if the functions aren't accelerated is understood. I was really just curious if I missed something obvious and the code would simplify significantly.

I tried running it but it gets stuck in an endless loop, with ISPRIME? getting 1 as arguments apparently indefinitely.
I'll try to follow your stack diagram and see where the problem is.

Re: A Sunday Programming Challenge

*Message #50 Posted by **C.Ret** on 21 Mar 2012, 9:28 a.m.,*
in response to message #49 by Oliver Unter Ecker

You have to check, 1 ISPRIME? my have return 1 as expected. You have to check also for 2: 2 ISPRIME? may have return 1 (2 is prime).

Perhaps, an inifinite loop result from NEXTPRIME fonction. I have no idea of how it is working.

My assumption is that : 1 NEXTPRIME returns 2, 2 NEXTPRIME returns 3 etc.

But, depending of impletation, maybe NEXTPRIME returns the closest prime number, resulting in the catastrophic chain reaction:
1 NEXTPRIME returns 1 (as 1 is the closest prime number),
4 NEXTPRIME return 5 (as 5 is the closest prime number - greatest or equal to the argument).

In this case, the modified version:

```
PROGRAM:                                  @ Comments                  @ Stack   5: 4: 3:  2:  1:
                                          @                           @                           P
« -> P
   « 0                                    @ Initiate Sum=0            @                 0 : 1 : P/3
     1 P 3 / FOR a
         2                                @ Initiate D <- 2
         a                                @          x <- a
         2                                @          i <- 2           @            Sum : D : x :   i
         WHILE OVER ISPRIME? NOT          @ loop until x is prime
         REPEAT
            NEXTPRIME                     @    i' <- nextprime ( i )  @            Sum : D : x :   i
            0 3 ROLLD                     @    Initiate  n <- 0       @        Sum : D : n : x :   i
            WHILE DUP2 MOD NOT            @    while x MOD i = 0
            REPEAT
               SWAP OVER / SWAP           @       x <- x/i            @    Sum : D :  n  : x/i :   i
               ROT 1 + 3 ROLLD            @       n <- n+1            @    Sum : D : n+1 : x/i :   i
            END
            4 ROLL OVER 5 ROLL 2 /
            CEIL ^ * 3 ROLLD              @       D = D*i^CEIL(n/2)   @         Sum : D' : x   :   i
            1 +                           @       inc i               @
         END
         DROP *                           @   D = D*x                 @                  Sum : D*x
         a SWAP /                                                     @                 Sum : a/2D
         IF P a 3 5 SQRT + * <
            THEN 4 P * a / 3 - SQRT 3 -   @ limited by perimeter      @  Sum : a/2D : SQRT(4P/a-3)-3
            ELSE 5 SQRT 1 -               @ limited by triangle side  @  Sum : a/2D :      SQRT(5)-1
         END
         * IP 1 + +                       @  Sum' = Sum + 1 + INT(...)  @                       Sum'
      NEXT
   »
»                                                                     @                       Sum
```

Re: A Sunday Programming Challenge

*Message #51 Posted by **Oliver Unter Ecker** on 21 Mar 2012, 11:23 a.m.,*
in response to message #50 by C.Ret

Hi C.Ret.

Believe it or not, 1 is not a prime... ;-)

You're not alone in thinking that it should be one. Here's what the Wikipedia page about primes has to say about the matter:

> Quote:
>
> Most early Greeks did not even consider 1 to be a number,[4] so did not consider it a prime. In the 19th century however, many mathematicians did consider the number 1 a prime. For example, Derrick Norman Lehmer's list of primes up to 10,006,721, reprinted as late as 1956,[5] started with 1 as its first prime.[6] Henri Lebesgue is said to be the last professional mathematician to call 1 prime.[7] Although a large body of mathematical work is also valid when calling 1 a prime, the above fundamental theorem of arithmetic does not hold as stated. For example, the number 15 can be factored as 3 · 5 or 1 · 3 · 5. If 1 were admitted as a prime, these two presentations would be considered different factorizations of 15 into prime numbers, so the statement of that theorem would have to be modified. Furthermore, the prime numbers have several properties that the number 1 lacks, such as the relationship of the number to its corresponding value of Euler's totient function or the sum of divisors function.[8][9]

Anyway, I had suspected that and changed my ISPRIME? function to return true for "1" but that led to stack underrun. I also tried starting the FOR loop with 2.

I shall debug this a little later when I have time.

The 50g (and ND1) behavior for these two functions is:
ISPRIME?: the first int for which 1 is returned is 2
NEXTPRIME: returns 2 for any int smaller than 2; any int is valid input, the next closest prime will be returned

This is obviously hard to develop if you don't actually have the functions built-in.

Re: A Sunday Programming Challenge

*Message #52 Posted by **Gerson W. Barbosa** on 14 Mar 2012, 7:39 a.m.,*
in response to message #23 by Marcus von Cube, Germany

> Quote:
>
> I came up with the constraint A > (SQR(5)-1)/2 * B, using a pencil and paper approach.

I'll try to follow your steps:

Since

```
b^2 = a*c
```

and

```
c <= a + b
```

then

```
b^2 <= a*(a + b)


b^2 <= a^2 + a*b


b^2/a^2 <= 1 + b/a


(b/a)^2 - b/a - 1 <= 0
```

Solving for b/a:

```
b/a <= (sqrt(5) + 1)/2


a/b > (sqrt(5) - 1)/2


a > b*(sqrt(5) - 1)/2
```

I've modified my algorithm to use this constraint. I thought there was no need to check the sum of the sides of the triangles because that's what this constraint was supposed to, but it had to stay. Here are the corresponding C program and RPN code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int main()
{
  int a, b, len, n;
  scanf("%d", &len);
  double c, k;
  n = len/3;
  k = (sqrt(5) - 1) / 2;
  for (b = n; b >= 2; b--)
  {
    a = b - 1;
    while (a > b * k )
    {
      c = 1.0 * b * b / a;
      if (c - int(c) == 0)
        if (a + b + c <= len)
          n++;
      a--;
    }
  }
```

```
  printf("n = %d\n",n);
  system("pause");
  return 0;
}
```

```
00 { 85-Byte Prgm }
01>LBL "GT"
02 STO 00                       R00 <- len
03 3
04 BASE÷
05 1E-3
06 +
07 STO 01                       R01 <- b
08 IP
09 STO 02                       R02 <- n = int(len/3)
10 5
11 SQRT
12 1
13 -
14 2
15 ÷
16 STO 04                       R04 <- k = (sqrt(5) - 1)/2
17>LBL 01                       for b = n down to 2
18 RCL 01
19 1
20 BASE-
21 STO 03                       R03 <- a = b - 1
22>LBL 02
23 RCL 01
24 IP
25 RCL× 04
26 RCL 03
27 X<=Y?                         a <= b*k ?  (while a > b*k)
28 GTO 04
29 RCL 01
30 IP
31 X^2
32 RCL÷ 03                      c = b^2/a
33 FP
34 X!=0?                        frac(c) != 0 ?
35 GTO 03
36 LASTX                          goto 03
37 RCL 01                        else
38 BASE+
39 RCL+ 03
40 RCL 00
41 X<=Y?                         c + b + a >= len?
42 GTO 03                          goto 03
43 1                              else
44 STO+ 02                          n = n + 1
45>LBL 03
46 1
47 STO- 03                      a = a - 1
48 GTO 02                       endwhile
49>LBL 04
```

```
50 DSE 01
51 GTO 01                       endfor
52 RCL 02                       disp n
53 .END.


   100 XEQ GT  -->    42     (1 min 10 sec, real HP-42S)
  1000 XEQ GT  -->   532     (~ 0.5 seconds, Free42 Decimal)
 10000 XEQ GT  -->  6427     ( 20.8 seconds, Free42 Decimal)
100000 XEQ GT  --> 75243     (~ 41 minutes,  Free42 Decimal)
```

Still an $O(n^2)$ process. I've been busy these days and haven't given it much thought, if this can be an excuse. Anyway, no progress after three days. Perhaps I have to upgrade my hardware (brainware) :-)

Gerson.

Edited to fix a typo as per Valentin's observation below.

Edited: 14 Mar 2012, 12:14 p.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #53 Posted by **Valentin Albillo** on 14 Mar 2012, 8:27 a.m.,*
in response to message #52 by Gerson W. Barbosa

Hi, Gerson:

Sorry but methinks you've got a typo (unless it's a program bug, hopefully not):

> Quote:
>
> ```
> 10000 XEQ GT  -->  6437     ( 20.8 seconds, Free42 Decimal)
> ```

The correct value is **6427**, not 6437.

Best regards from V.

Re: A Sunday Programming Challenge

*Message #54 Posted by **Gerson W. Barbosa** on 14 Mar 2012, 12:10 p.m.,*
in response to message #53 by Valentin Albillo

That was indeed a typo. The program does return 6427. I will fix it presently. Thank you for pointing it out.

Best regards,

Gerson.

Edited: 14 Mar 2012, 12:14 p.m.

Re: A Sunday Programming Challenge

*Message #55 Posted by **Marcus von Cube, Germany** on 14 Mar 2012, 8:56 a.m.,*
in response to message #52 by Gerson W. Barbosa

My paper and pencil approach was just what you found out: find the zero of a+b>c with ac=b^2. This doesn't make sure that a+b+c is bound in any way, that's just another inequality to check for.

Re: A Sunday Programming Challenge

*Message #56 Posted by **C.Ret** on 15 Mar 2012, 4:49 a.m.,*
in response to message #55 by Marcus von Cube, Germany

That's right !

Re: A Sunday Programming Challenge

*Message #57 Posted by **Peter Murphy (Livermore)** on 11 Mar 2012, 9:37 p.m.,*
in response to message #18 by Valentin Albillo

An alternative to the usual triangle inequality arises from Heron's formula, which makes it clear that the semiperimeter of an actual triangle is greater than all three side lengths.

Re: A Sunday Programming Challenge

*Message #58 Posted by **C.Ret** on 11 Mar 2012, 6:31 p.m.,*
in response to message #15 by Allen

Yes, all the 83 I count satisfy b^2=ac condition.

I also get 50 by considering only a < b < c.

a	b	c	P=a+b+c		a	b	c	P=a+b+c
1	2	4	7	nat	1	1	1	3
1	3	9	13	nat	2	2	2	6
2	4	8	14	nat	3	3	3	9
4	6	9	19		4	4	4	12
1	4	16	21	nat	5	5	5	15
3	6	12	21	nat	6	6	6	18
2	6	18	26	nat	7	7	7	21
4	8	16	28	nat	8	8	8	24
1	5	25	31	nat	9	9	9	27
5	10	20	35	nat	10	10	10	30
9	12	16	37		11	11	11	33

8	12	18	38		12	12	12	36
3	9	27	39	nat	13	13	13	39
4	10	25	39	nat	14	14	14	42
2	8	32	42	nat	15	15	15	45
6	12	24	42	nat	16	16	16	48
1	6	36	43	nat	17	17	17	51
7	14	28	49	nat	18	18	18	54
9	15	25	49	nat	19	19	19	57
4	12	36	52	nat	20	20	20	60
8	16	32	56	nat	21	21	21	63
1	7	49	57	nat	22	22	22	66
12	18	27	57		23	23	23	69
16	20	25	61		24	24	24	72
2	10	50	62	nat	25	25	25	75
3	12	48	63	nat	26	26	26	78
9	18	36	63	nat	27	27	27	81
5	15	45	65	nat	28	28	28	84
4	14	49	67	nat	29	29	29	87
10	20	40	70	nat	30	30	30	90
1	8	64	73	nat	31	31	31	93
18	24	32	74		32	32	32	96
16	24	36	76		33	33	33	99
11	22	44	77	nat				
6	18	54	78	nat				
8	20	50	78	nat				
9	21	49	79	nat				
4	16	64	84	nat				
12	24	48	84	nat				
2	12	72	86	nat				
1	9	81	91	nat				
7	21	63	91	nat				
13	26	52	91	nat				
25	30	36	91					
3	15	75	93	nat				
16	28	49	93	nat				
20	30	45	95					
9	24	64	97	nat				
14	28	56	98	nat				
18	30	50	98	nat				

Edit : nat = not a triangle

Edited: 12 Mar 2012, 11:46 a.m. after one or more responses were posted

Re: A Sunday Programming Challenge

*Message #59 Posted by **Allen** on 11 Mar 2012, 6:35 p.m.,*
in response to message #58 by C.Ret

> Quote:
>
> Yes, all the 83 I count satisfy b^2=ac condition. I also get 50 by considering only a < b < c.

Ok, I assume you also get the 1,2,4 shape as above? What is the largest angle in a triangle with sides 1,2 and 4?

Re: A Sunday Programming Challenge

*Message #60 Posted by **C.Ret** on 11 Mar 2012, 6:52 p.m.,*
in response to message #59 by Allen

OK.

You ask the good question.

I found 83 triplets (a,b,c), but I have not check at all if there are triangles !

In fact (1,2,4) is not a triangle, I found no way to draw it. One of the edge is too short.

That what I miss in the problem; a triplet (a,b,c) is not necessery a triangle.

Thanks.

I have to found a way to check for triangularity and to adjust my algorithm.

```
'---------------------------------

00 { 59-Byte Prgm }
01>LBL "FTRI"
02 STO 01              @  Store perimeter in R01
03 3
04 ÷
05 IP                  @ Initate a to INT(P/3)
06 0                   @ Initiate counter R00 to zero
07 STO 00
08 Rv
09>LBL 00              @ ---------- Main loop (over a in stack x:)
10 RCL ST X
11>LBL 01              @ ---------- Loop over b from a to ...
12 RCL ST X
13 RCL+ ST Z           @    place a+b in stack
14 RCL ST Y
15 X^2
16 RCL÷ ST T           @   estimate c = b.b/a
17 X>Y?
18 GTO 02              @   test triangle if ( c > a+b ) exit
19 +
20 RCL 01
21 X<Y?
22 GTO 02              @  test perimeter if ( a+b+c > P ) exit
23 +
24 FP
25 X=0?                @  test c integer if c interger count triangle
26 ISG 00
27 NEG
```

```
28 Rv
29 1                  @ increase b
30 +
31 GTO 01
32>LBL 02             @ -------  exit
33 R^
34 DSE ST X
35 GTO 00             @     loop while a>0
36 RCL 00             @ ------ recall counter
37 END
```

Register:
R00 store n (number of geometric triangles)
R01 store perimeter P

usage:
100 EXQ FTRI ---> 42
1000 EXQ FTRI ---> 532
10000 EXQ FTRI ----> 6427

Edited: 11 Mar 2012, 9:01 p.m.

Re: A Sunday Programming Challenge

*Message #61 Posted by **Allen** on 11 Mar 2012, 10:34 p.m.,*
in response to message #60 by C.Ret

> Quote:
>
> 10000 EXQ FTRI ----> 6427

A very compact and small size solution. (Thank you for commenting the various loops and code) Very Nice!

Now as to the speed... How long does it take your emulator to calculate F(1e6)? What about F(1e8)?

Is there a clever way to speed up your program by a factor of 1000 or more?

Here are some target times for different machines:

F(N)	41C	32sii	42s	Free42(D)	Result
F(100)	9s	2s	6s	0s	42
F(1000)	72s	15s	45s	0s	532
F(1e4)	651s	145s	426s	0s	6,427
F(1e5)	-	-	-	0s	DD,DDD
F(1e6)	-	-	-	1.5s	CCC,CCC
F(1e7)	-	-	-	5s	B,BBB,BBB
F(1e8)	-	-	-	60s	AAA,AAA,AAA