HP Forum Archive 18

[Return to Index | Top of Index]

Valentine's Day 2008 Mini-Challenge !
*Message #1 Posted by **Valentin Albillo** on 14 Feb 2008, 7:45 p.m.*

Hi all,

It's been almost a year since I posted my latest S&SMC (#19), but since today it's my name's day, in order to commemorate the event somewhat and also to mark time before the release of my new ***"S&SMC#20 Spring's Special"*** (to be posted next April 1st), you may want to try this short Mini-Challenge for any HP calc model of your choice, new or vintage (other pocket-sized brands also welcome, as well as faithful emulators/simulators. PC or Mac programs are *expressly disallowed*. Let's see:

Valentine 2008's Mini-Challenge

Imagine two people playing the following "game": one of them selects *five real numbers* of his choice (say x_1, x_2, ..., x_5), and computes the *sums* for all different pairings of those numbers (x_1+x_2, x_1+x_3, ..., x_4+x_5). He then tells the other person these sums (in whichever arbitrary order he chooses), and the other person must then use those sums to try and compute the original five numbers.

Your task is thus:

> ***"To write a program which accepts as input the sums of each pair of numbers, given in arbitrary order, and proceeds to compute and output the five real numbers which originated them, sorted in ascending order ."***

For instance, suppose the five original numbers are:

```
1, 3, -4, 2.1, 3.9
```

and thus the corresponding sums for each pair are:

```
4, -3, 3.1, 4.9, -1, 5.1, 6.9, -1.9, -0.1, 6
```

the program must then accept these sums in whatever order (and *must* work for *any* ordering of the sums) and proceed to compute and output the five numbers like this (example particularized for the HP-71B):

```
>RUN

  S(1)? 5.1,6.9,-1.9,4,-3,6,3.1,4.9,-1,-0.1 (arbitrary order) [ENDLINE]

       -4, 1, 2.1, 3, 3.9 (sorted original numbers)
```

Once you write the program, you must use it to find the solutions for these sample sums:

```
1) 3734, 3768, 284, 3950, 466, 4000, 516, 500, 3966, 3784

2) -0.4233,-1.7274,-2.4485,-0.9055,4.1325,3.4114,4.9544,2.1073,3.6503,2.9292

3) 1, -5, 7, -17, -2, 10, -14, 4, -20, -8

4) -22, -4, 118, 4, 126, 144, -31, -23, -5, 117

5) -34.71,23.992,-25.094,-15.1,-2.696,16.914,4.382,6.92,-32.172,14.376
```

Note:

- The solution, if it exists, is *unique*. Of course, not every set of arbitrary "sums" does have a solution. Your program may assume that the sums have actually been computed from five actual real numbers so there is indeed a unique solution. The behavior for inadequate initial sums which result in no solution may be left undefined as per the challenge specifications they are unacceptable input.
- You must optimize *both* for speed and size, in that order. Next week I'll post my original *5-line* solution for the HP-71B which finds the numbers in negligible time and is trivial to adapt to most any HP model. If you don't have the real HP model of your liking, you may want to consider using any of the excellent freeware emulators/simulators out there such as Emu71, Emu42S, Nonpareil, etc. Just google for the corresponding name.

If you succeed in solving the challenge for five numbers, you might want to try the general case of N numbers and discuss in particular which values of N do indeed have an unique solution (when the problem is solvable at all) so that the N numbers can indeed be retrieved from their sums, and which values of N result in multiple solutions, in which case it'll be impossible to retrieve the original numbers. I won't post any solution for this general case but I'll discuss which N result in unique solutions and which don't.

Turning things up a notch ...

Relying on my experience with these Mini-challenges and keen forum visitors, I knew in advance some among you would eat the first part of this mini-challenge for breakfast, so let's turn things up a notch ...

> ***"Write a program which, from the set of integers 1,2,...,9, selects and uses four of them to fill in a 3x3 matrix so that its determinant is 1 and replacing each element by its square still results in the determinant evaluating to 1."***

For instance, your four selected numbers could be 1,2,3,4, and you would perhaps consider filling up the 3x3 matrix with them like this:

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

where its determinant actually evaluates to 1, as specified. However, squaring each element gives the matrix:

```
1   1   1
1   9   4
9  16  16
```

which has determinant = 35 instead of 1, so regrettably this is not a solution.

Apart from the usual transformations that leave the determinant invariant (swapping of certain rows and columns, etc), the solution is *unique*, and you must optimize your program primarily for *speed* for this particular notch.

The final notch

After adding the previous notch I stated:

"If you manage to solve this with relative ease, perhaps I'll feel "forced" to turn things up still another notch ! Have you got what it takes ? ;-)"

and it seems that some of you do indeed have what it takes, namely an state-of-the art HP model and a version of the C language to run natively on it. This being so, here's the promised "final notch" for your consideration:

"Write a program to find four distinct integers such that the sum of any pair is a perfect square"

For example, the set formed by the three values 6, 58, and 138 is such that their sums in pairs are all perfect squares, namely $6 + 58 = 64 = 8^2$, $6 + 138 = 144 = 12^2$, and $58 + 138 = 196 = 14^2$. Your program must find at least a set of *four* such numbers and I will of course post my original solution to this notch as well.

There will be no further expansions to this mini-challenge, which will hopefully serve as proper training for the incoming ***"S&SMC#20: Spring 2008 Special"*** due next April 1st and which will really, *really* test your programming, math, and resourcefulness to the most, and that's a promise ! :-)

So much for exposition. Now for your results, keep them coming ! :-)

Best regards from V.

Edited to turn things up a notch ! :-)

Edited to add a final notch ! :-)

Edited: 18 Feb 2008, 10:48 a.m. after one or more responses were posted

Re: Valentine's Day 2008 Mini-Challenge !

*Message #2 Posted by **dbatiz** on 14 Feb 2008, 8:54 p.m.,*
in response to message #1 by Valentin Albillo

Happy Valentine's Day!

Thank you for devising yet another interesting challenge. This one had me guessing until I saw you mention your solution was a 5 liner. I thought this must be more of a math problem then a logic problem.

I have solved your examples, but before I post my technique I wanted to ask a question: Are all solutions welcome? I solved it using my 50g, but I didn't need to write any code. I didn't want to be a spoiler.

I've been chewing on the general case of N numbers. My gut tells me that if the number of pairings exceeds the number of original values, there is enough data to find a unique solution. I'm guessing that for N>2 there will be a unique solution, if one exists.

Thanks again,

Very respectfully,

David

PS Edited to add the last paragraph about the general case of N

PPS I jumped the gun. There's much more to this challenge than I originally thought. I thought I had accounted for the random ordering of the sums, but my technique didn't work. This may take a while.... Again, thank you Valentine for an interesting challenge!

Edited: 14 Feb 2008, 10:14 p.m.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #3 Posted by **Steve Perkins** on 15 Feb 2008, 7:32 p.m.,*
in response to message #2 by dbatiz

This is, as usual an interesting problem.

I've been trying to get a feel for the problem using smaller amounts of numbers and sums. So far I've observed:

```
2 numbers 1 sum: infinite solutions
3 numbers 3 sums: unique solution (even I could program this one!)
4 numbers 6 sums: 2 solutions (sometimes identical)
5 numbers 10 sums: unique solution
6 numbers 15 sums: ??
```

I think these are correct so far. More work to do if I have time.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #4 Posted by **Thomas Klemm** on 14 Feb 2008, 10:08 p.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin

Here's my program for an HP-48GX:

```
\<< SORT \-> L
  \<< L 1 2 SUB OBJ\->
DROP L \GSLIST 4 / L
9 10 SUB OBJ\-> DROP
5 \->ARRY
[[ 1 1 0 0 0 ]
 [ 1 0 1 0 0 ]
 [ 1 1 1 1 1 ]
 [ 0 0 1 0 1 ]
 [ 0 0 0 1 1 ]]
/
  \>>
\>>
```

And here are the solutions to your examples:

1. [-1492 1776 1958 1992 2008]
2. [-3.1416 0.6931 1.4142 2.2361 2.7183]
3. [-16 -4 -1 2 8]
4. [-16 -15 -7 11 133]
5. [-32.093 -2.617 -0.079 6.999 16.993]

Thanks for the nice mini-challenge and all the best for your day.

Kind regards
Thomas

Re: Valentine's Day 2008 Mini-Challenge !

*Message #5 Posted by **Paul Dale** on 14 Feb 2008, 11:08 p.m.,*
in response to message #4 by Thomas Klemm

Pipped for the 48/49 solution.

Mine solution (neither smallest nor fastest) is:

```
<< 10 ->LIST SORT -> A
  << A 1 GET A 2 GET A 9 GET A 10 GET A sigmaLIST 5 ->ARRY >>
"[[1 1 0 0 0[1 0 1 0 0[0 0 1 0 1[0 0 0 1 1[4 4 4 4 4" OBJ-> / >>
```

Assuming I've not made any typos. Input is the 10 sums on the stack, output is an array containing the 5 unknowns in ascending order. Execution time is essentially instant on my 49g+.

One possible improvement would be to invert the matrix ahead of time and include that inline instead of calculating the inverse every time.

- Pauli

Re: Valentine's Day 2008 Mini-Challenge !

*Message #6 Posted by **Thomas Klemm** on 15 Feb 2008, 7:53 a.m.,*
in response to message #4 by Thomas Klemm

A version without local variable:

```
\<< SORT DUP
1 GET SWAP DUP
2 GET SWAP DUP
\GSLIST SWAP DUP
9 GET SWAP
10 GET
5 \->ARRY
[[ 1 1 0 0 0 ]
 [ 1 0 1 0 0 ]
 [ 4 4 4 4 4 ]
 [ 0 0 1 0 1 ]
 [ 0 0 0 1 1 ]]
/
\>>
```

Seems to be a little slower though? I'm not sure about the influence of the determinant of the matrix. It might be better if it is 1.
However Egan will probably post a 50g/HPGCC3 version which is way faster and Raymond might beat us all with a native assembler program.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #7 Posted by **Thomas Klemm** on 15 Feb 2008, 4:13 p.m.,*
in response to message #4 by Thomas Klemm

Here's the listing for an HP-11C:

```
001  LBL A      021  STO 2      041  GTO 3      061  4          081  -
002  1          022  ISG        042  STO 3      062  STO / 2    082  STO 4
003  .          023  LBL 0      043  x<>y       063  RCL 0      083  RDN
```

```
004  0          024  RCL I      044  STO 4      064  RCL 0      084  STO - 2
005  1          025  R/S        045  GTO 9      065  RCL 2      085  RCL 0
006  STO I      026  STO + 2    046  LBL 1      066  RCL 4      086  R/S
007  R/S        027  RCL 0      047  STO 1      067  -          087  RCL 1
008  STO 0      028  x>y        048  x<>y       068  RCL 1      088  R/S
009  ISG        029  GTO 1      049  STO 0      069  -          089  RCL 2
010  RCL I      030  x<>y       050  GTO 9      070  STO 1      090  R/S
011  R/S        031  RCL 1      051  LBL 2      071  -          091  RCL 3
012  RCL 0      032  x>y        052  x<>y       072  STO 0      092  R/S
013  x>y        033  GTO 2      053  STO 1      073  RDN        093  RCL 4
014  x<>y       034  x<>y       054  GTO 9      074  STO - 2    094  RTN
015  STO 0      035  RCL 3      055  LBL 3      075  RCL 4
016  STO 3      036  x>y        056  x<>y       076  RCL 4
017  x<>y       037  GTO 9      057  STO 3      077  RCL 2
018  STO 1      038  x<>y       058  LBL 9      078  RCL 3
019  STO 4      039  RCL 4      059  ISG        079  -
020  +          040  x>y        060  GTO 0      080  STO 3
```

Example 1:

```
Display         Command


0.0000          f FIX 3
0.000           f A
1.010           3734 R/S
2.010           3768 R/S


(...)


9.010           3966 R/S
10.010          3784 R/S
running
-1,492.000      R/S
1,776.000       R/S
1,958.000       R/S
1,992.000       R/S
2,008.000
```

For those too lazy to type that program into nonpareil emulator
I've base64-encoded the file after saving the state of the calculator:

```
H4sIAAAAAAAAA5WXTU/jMBCG7/srsr5D7XyQRmrLATistLusBFqJE3KccYnIR0kDC/9+J2lrx07apEV
Ro/jJzHjmnXFZXH/mmfMB1TYtiyVhl5Q4UIgySYv1kvx4uL+Yz4PogpHr1bfF99Yv7m8enP3fOtuY1OA
```

9PD493vxxSlMWGV5Bml0mdkNVit9q1iUbzMoEMHbAb4mwyXsuyypfko/zia6iIwyvxsiTFe43vv6ez1
UK8pBun4Dksye/2aQXrdFtjoKsF3u6XOHESXvMlocaHoAENxQYk2w/iJiQMaGcNOJhQPgD1LBVToLWC
PNPF5rAgzOdvh+cW//a8heywZu1JlmrBWtmqhbmdiI2RCTMIrKx4JU5aJPCJOVdG6Dw6BbKpoKtBOyz
TtXccTECkOT+WEsGr6ku9bAbN//FXOLK2kc88SVCoB7mZr1Y8t9ZDE+CiTj/gOebFUP7GWZ3CcVZncZ
zViRxnfbX7cTY4g706gw3PYOdnsNEZrB4643nQs2ec1Y03ziZnxHtM00OalIbdWWfozpqJbMzlv7vh7
fy8ucWp3UH16IWCx5nyz8x9xVnadMO+o5q1AX855GX15fDtkmCToZ+sFE7Tiu3MP7wdMT9yD0dAOyG7
mG6eA9J+Nx67mO6bk5humZOY7pYGM06drlPdKF1rczs23SNdrD0jutZ0e2hsf4p1Md0ZGsM726lqClf
O6TwGGYOLWbYx3Q+nrEHnuOhyljWYViyYViyYViwwi6XCs2MbLpadEBguVg8bKla/CjCtWKCKpaJvb+
wtTCyWnluGOduaHlknMT2tTmJqUBhUL296Ru252At8YWNS6S3wBA1cvBhePUzpLRCChb6MQ0/GgT1Dp
NJb4AY+/jUwlbGVEKn0FjK00lw+vtBzqvQW0EhIn1G4olHYc6r0JlzqsShk+O2y9rdWp52l0ltIhRsy
yhD1hWvHpvQWeswDgTlDW2DPEKn0hnlzmYsgumXt7O46VXpDgHpCUE9S3IbtVOktdCUP/WYLoRf2EqL
0JhqHtLEGV31M6U3QSEomm7nKKLedTtObHNKbB9K21tPbTnYNNtsdT3jT/tOz+vYfHmnWslANAAA=

Edited: 15 Feb 2008, 5:07 p.m.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #8 Posted by **Rodger Rosenbaum** on 16 Feb 2008, 9:20 a.m.,*
in response to message #4 by Thomas Klemm

Can you solve the case of 6 numbers whose pairwise sums are:

4,12,4,5,7,8,2,6,10,5,9,13,6,10,14

and the case of 7 numbers whose pairwise sums are:

9,3,10,8,9,3,4,15,10,15,8,9,3,9,2,10,16,9,10,3,9

Also, your program will return a solution even when that solution when used to recreate the pairwise sums doesn't give the sums that you started with. That would be one way to determine if the original pairwise sums have a unique solution.

Can you find a way to determine if some given sums have a unique solution without first solving with your program as it stands?

Re: Valentine's Day 2008 Mini-Challenge !

*Message #9 Posted by **Thomas Klemm** on 18 Feb 2008, 4:53 p.m.,*
in response to message #8 by Rodger Rosenbaum

The solutions are:

- [1, 1, 3, 4, 5, 9]

- [1, 1, 2, 2, 7, 8, 8]

Let's assume the solution is [a, b, c, d, ...]
It is assured that a+b <= a+c <= ...
However you can not tell which of the sums a+d or b+c belongs to the third smallest number.
Therefore I tried both possibilities and checked the results.

In the case of 6 numbers I got the solution with b+c=4:

```
| 1 1 0 0 0 0 | | a |   |  2 |
| 1 0 1 0 0 0 | | b |   |  4 |
| 0 1 1 0 0 0 | | c |   |  4 |
| 1 1 1 1 1 1 | | d | = | 23 |
| 0 0 0 1 0 1 | | e |   | 13 |
| 0 0 0 0 1 1 | | f |   | 14 |
```

And here's the equation in the case of 7 numbers:

```
| 1 1 0 0 0 0 0 | | a |   |  2 |
| 1 0 1 0 0 0 0 | | b |   |  3 |
| 0 1 1 0 0 0 0 | | c |   |  3 |
| 1 1 1 1 1 1 1 | | d | = | 29 |
| 0 0 0 0 1 1 0 | | e |   | 15 |
| 0 0 0 0 1 0 1 | | f |   | 15 |
| 0 0 0 0 0 1 1 | | g |   | 16 |
```

> Quote:
>
> Also, your program will return a solution even when that solution when used to recreate the pairwise sums doesn't give the sums that you started with.

The program behaves as specified in Valentin's note:

> Quote:
>
> The behavior for inadequate initial sums which result in no solution may be left undefined as per the challenge specifications they are unacceptable input.

As for your last question: I don't know of a simple way to determine whether the sums have actually been computed from five real numbers.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #10 Posted by **Rodger Rosenbaum** on 18 Feb 2008, 9:10 p.m.,*
in response to message #9 by Thomas Klemm

> Quote:
>
> The program behaves as specified in Valentin's note:
>
> > Quote:
> >
> > The behavior for inadequate initial sums which result in no solution may be left undefined as per the challenge specifications they are unacceptable input.

Yes, I realized that. I wasn't complaining that your program didn't meet Valentin's specifications. I was just mentioning it as a prelude to asking for a method to determine if a true solution exists.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #11 Posted by **Steve Perkins** on 20 Feb 2008, 12:22 p.m.,*
in response to message #10 by Rodger Rosenbaum

Consider the following sets of pairwise sums from 6 numbers:

```
4 8 8 10 10 14 14 16 16 18 20 20 22 22 28
4 8 8 10 10 14 16 16 16 16 20 20 22 22 28


The 6 numbers generating the first set are: 1 3 7 7 13 15
The 6 numbers generating the other set are: 2 2 6 8 14 14
```

Clearly you need more than the 3 lowest and 3 highest sums to determine which solution works. My simple early attempts at a solver for 6 numbers gave both sets as possible solutions.

Do we have to go back and compare the entire set of sums, or is there an easier way?

Can we always resolve the 2 candidate solutions by comparing with the complete set of sums, or are there cases where 2 sets of original numbers generate identical sums? I don't believe identical sums can come from 2 sets, except back in the simple case of 6 sums from 4 numbers.

With 7 numbers and 21 sums it looks like we would have 4 guesses based on the 3 greatest and 3 least sums. It gets more complicated quickly. It feels like I'm getting closer to the general solution, but there may be cases that limit the possibilities. I wish I could be more rigorous, but time is (as always) limited.

Edited: 20 Feb 2008, 1:13 p.m.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #12 Posted by **Steve Perkins** on 20 Feb 2008, 2:30 p.m.,*
in response to message #11 by Steve Perkins

This has been, as usual, an interesting problem.

I think I have the general case for up to 11 numbers:

```
2 numbers 1 sum: infinite solutions
3 numbers 3 sums: unique solution (even I could program this one!)
4 numbers 6 sums: 2 solutions (sometimes identical, and thus unique)
5 numbers 10 sums: unique solution
6 numbers 15 sums: 2*1 guesses, resolvable to a unique solution
7 numbers 21 sums: 2*2 guesses, resolvable to a unique solution
8 numbers 28 sums: 7*2 guesses, resolvable to a unique solution
9 numbers 36 sums: 7*7 guesses, resolvable to a unique solution
10 numbers 45 sums: 50*7 guesses, resolvable to a unique solution
11 numbers 55 sums: 50*50 guesses, resolvable to a unique solution
....
```

In each case of 6+ original numbers, you can always resolve the solution by calculating the partial sums of the candidate guesses and comparing with the original entries.

I haven't found a reasonable equation to get a handle on the number of guesses. I wouldn't even be surprised to have made a mistake. The basic idea, is to guess which partial sum corresponds to each row of the sparse matrix:

```
1 1 0 0 0 ...
1 0 1 0 0 ...
1 0 0 1 0 ...
1 0 0 0 1 ...
...
1 1 1 1 1 ...
...
... 1 0 0 0 1
... 0 1 0 0 1
```

```
... 0 0 1 0 1
... 0 0 0 1 1
```

The first 2 and last two rows are determined (2 least and 2 greatest sums). The others require guessing.

If the sorted solution is [a b c d e ...] and the sorted partial sums are [s1 s2 s3 s4 s5 ...] then row 3 of our sparse matrix can be s3 or s4. Row 4 can be s4-s7. Row 5 can be s5-s11. A similar guess is made for the last rows. The possibilities can be reduced if you consider which sum you chose for previous rows.

I hope there are more simplified ways to tackle this. Hopefully someone can enlighten me further.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #13 Posted by **Rodger Rosenbaum** on 21 Feb 2008, 12:23 a.m.,*
in response to message #12 by Steve Perkins

Can you give an example of two distinct (meaning you can't derive one from the other by just reordering) sets of 4 numbers which give the same pairwise sums?

I can (Re: Valentine's Day 2008 Mini-Challenge !)

*Message #14 Posted by **Valentin Albillo** on 21 Feb 2008, 2:45 a.m.,*
in response to message #13 by Rodger Rosenbaum

(1,4,6,7) and (2,3,5,8)

Best regards from V.

Re: I can (Re: Valentine's Day 2008 Mini-Challenge !)

*Message #15 Posted by **Rodger Rosenbaum** on 21 Feb 2008, 8:00 a.m.,*
in response to message #14 by Valentin Albillo

I was wondering if he had discovered how to know if the sums have only one solution or two.

Given 4 numbers {a, b, c, d}, if d = (b + c - a) then the pairwise sums generated from these 4 will have only one (exact) solution, otherwise there are two solutions. This means that almost any 4 numbers you type will also have a companion solution. But, notice that the set (1, 2, 3, 4), which would be a set a person might type generates a set of pairwise sums with only one solution!

If there *are* two solutions, then if one of them is {a, b, c, d}, where d = (b + c - a - n), n = some number <> zero, the other solution is:

{ (2a + n)/2, (2b - n)/2, (2c - n)/2, (2d + n)/2)

It appears that a necessary condition that any 6 numbers that actually have an exact solution will have a pair of solutions (which may not be distinct) is that at least two of the pairwise sums are the same, but this is not a sufficient condition.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #16 Posted by **Rodger Rosenbaum** on 21 Feb 2008, 9:46 a.m.,*
in response to message #12 by Steve Perkins

The 4 numbers 6 sums case can have a solution that appears to be unique, but isn't. It is a solution of multiplicity 2; for example the sums {3 4 3 5 4 5} which are generated from the 4 numbers {1 2 3 2}.

It can be shown that:

Given 4 numbers {a, b, c, d}, if d = (b + c - a) then the pairwise sums generated from these 4 will have only one (exact) solution, otherwise there are two solutions.

If there are two solutions, then if one of them is {a, b, c, d}, where d = (b + c - a - n), n = some number <> zero, the other solution is:

{ (2a + n)/2, (2b - n)/2, (2c - n)/2, (2d + n)/2)

If you start with the numbers {1 2 3 2} and let n = 2, then the expression above evaluates to {2 1 2 3}, which appears to be the same as what we started with, rearranged. The fact that using the espression with n=2 gives a set which generates the same sums, shows that it's really a solution of multiplicity 2.

On the other hand, if you use the set {1 2 3 4} to generate the sums, there is really only one exact solution.

The case where we start with 5 numbers has the solution using the matrix:

```
[ 1 1 0 0 0 ] [a]   [smallest pairwise sum       ]
[ 1 0 1 0 0 ] [b]   [next smallest pairwise sum  ]
[ 4 4 4 4 4 ] [c] = [sum of all the pairwise sums]
[ 0 0 1 0 1 ] [d]   [next to largest pairwise sum]
[ 0 0 0 1 1 ] [e]   [largest pairwise sum        ]
```

I use 4's in the 3rd row instead of 1's so that the element in the column vector is the sum of the pairwise sums, rather than that sum divided by 4.

With this technique the only way determine if a solution is good is to regenerate the pairwise sums and see if you get what you started with.

However, if you add a row to the matrix:

```
[ 1 1 0 0 0 ] [a]   [smallest pairwise sum       ]
[ 1 0 1 0 0 ] [b]   [next smallest pairwise sum  ]
[ 4 4 4 4 4 ] [c] = [sum of all the pairwise sums]
[ 0 0 0 1 1 ] [d]   [next to largest pairwise sum]
[ 0 0 1 0 1 ] [e]   [largest pairwise sum        ]
[ 2 3 2 3 2 ]       [sum of sums - smallest two sums - largest two sums]
```

Now you can augment the 6x5 overdetermined matrix with the column matrix and see if the rank of the augmented matrix is larger than the rank of the 6x5 matrix alone. If the rank increases to 6, then there is no exact solution. If the rank of the augmented matrix is still 5, then the exact solution is the least squares solution to the overdetermined system.

While this is theoretically satisfying, it's just about as easy to provisionally solve the system with the smaller matrix and test the solution.

One might think this scheme could be applied to the larger N systems, but it doesn't work because there aren't enough known relationships to provide more rows in the matrix to increase the rank to N.

For N starting numbers, consider the matrix of order (PERM(N,2) X N) whose rows have only two unity elements in each row. The PERM(N,2) rows have all possible combinations of columns containing 1's taken two at a time. For example, the N=5 case:

```
[ 1 1 0 0 0 ]
[ 1 0 1 0 0 ]
[ 1 0 0 1 0 ]
[ 1 0 0 0 1 ]
[ 0 1 1 0 0 ]
[ 0 1 0 1 0 ]
[ 0 1 0 0 1 ]
[ 0 0 1 1 0 ]
[ 0 0 1 0 1 ]
[ 0 0 0 1 1 ]
```

I'll call this the MOC (matrix of combinations). The MOC can be used to generate the pairwise sums by postmultiplying the MOC by a column vector of the starting numbers [a b c d e]T. For example, MOC * [1 2 3 4 5] gives [3 4 5 6 5 6 7 7 8 9]T. And, the MOC can be used to solve the problem.

Using an HP50, put [3 4 5 6 5 6 7 7 8 9]T on level 2 of the stack, and the 5th order MOC on level 1 and execute LSQ. See [1 2 3 4 5]T, the correct answer. In theory, this will solve the problem for any size set of pairwise sums.

However in practice, a problem is that if the arrangement of the sums is not right, it won't work. But, in theory the solution can be found in a finite, though large, number of steps. The procedure is to augment the MOC with all possible permutations of the pairwise sums as a column vector. If the rank of the augmented MOC is increased, then there is no exact solution for that particular permutation of the sums. If the rank is not increased, then the least squares solution of the system is the desired solution. If all of the permutations of the column vector of sums increase the rank of the augmented matrix, then there is no exact solution.

This is just feasible to do on a PC for the N=5 case. The procedure will find a maximum of 5! solutions with 10! permutations of the sums tested. The solutions are all the same set of numbers, in all possible permutations. Of course, if some of the sums are the same, then the number of (distinct) permutations is reduced.

You can see that the number of permutations to be considered rapidly gets out of hand as the order of the problem increases!

Re: Valentine's Day 2008 Mini-Challenge !

*Message #17 Posted by **Egan Ford** on 21 Feb 2008, 1:15 p.m.,*
in response to message #16 by Rodger Rosenbaum

> Quote:
>
> But, in theory the solution can be found in a finite, though large, number of steps.

Hi Rodger,

I experimented with this last weekend as I worked on my general solution for any N, but got sidetracked with parts 2 and 3. Here is what I discovered so far:

For any N there will be

$$S = \frac{N*(N-1)}{2}$$

pairwise sums.

If N = 1, then there is one solution, if N = 2, there are infinite solutions. If N > 2 there is at least one solution. With random numbers I found that with N > 2 there was only one solution except for N = 4 where there was always 2 solutions. I looks like you found a special case for N = 4 that will give only one solution. That makes be wonder about N > 4 having multiple solutions.

When brute force searching for solutions, if your NxN matrix A contains all ones in the first row, the first column and the diagonal, and the first value set in the vector b is the sum of all pairs/(N-1), then there are PERM(S,N-1) permutations to test. N! of the permutations will return the same correct answer. If any of the original numbers (x_1, x_2, ...) have duplicates then the number of correct identical solutions is > N!.

My brute force program has the option to find all solutions or quit after finding one. It is fast for N=3..7, slows a bit at 8, takes forever at 9. I can speed it up by using the two largest and smallest values reducing the number of permutations to test to PERM(S-4,N-5). For the original problem there is no permutations, just one way to solve it. For N = 6 there are only 2 tests as Thomas has already pointed out. If I have time I'll finish and post it this weekend.

Edited: 21 Feb 2008, 1:20 p.m.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #18 Posted by **Rodger Rosenbaum** on 21 Feb 2008, 7:19 p.m.,*
in response to message #17 by Egan Ford

> Quote:
>
> For any N there will be
>
> N*(N-1) S = ------- 2
>
> pairwise sums.

This is, of course, equal to COMB(N,2). Makes one wonder about the N numbers case with sums taken 3 at a time, or 4 at a time, etc.

The canonical method I gave in another post is a big time waster because we are only looking for N numbers, but we have COMB(N,2) equations. So a lot of the permutations are guaranteed not to have a solution. Having only N equations in N unknowns is better.

> Quote:

> That makes be wonder about N > 4 having multiple solutions.

I also wonder, but the details of my derivation of the special case for N=4 makes me lean toward the notion that there are no multiple solutions for *most* of the higher order cases, and maybe not any.

> Quote:
>
> If any of the original numbers (x1, x2, ...) have duplicates then the number of correct identical solutions is > N!.

Don't you mean: "...the number of correct identical solutions is < N!."? I assume you're referring to the fact that when there are duplicate original numbers, the sums include duplicates, and therefore the number of (distinct) permutations is reduced.

> Quote:
>
> I can speed it up by using the two largest and smallest values reducing the number of permutations to test to PERM(S-4,N-5).

You must be doing this to get a solution for the N=9 case in fewer than the 10^12 permutations it would take otherwise!

We need to take advantage of any sums or combinations thereof that we can use to reduce the order of the problem. I think the 5 we know, the 2 smallest and 2 largest sums and the sum of the sums, are all there are in the case of pairwise sums.

I think the major reductions in time have been found. There may be a few small tweaks left.

Now, on to the sums of 3 at a time case!

Premature retreat (Re: Valentine's Day 2008 Mini-Challenge !)

*Message #19 Posted by **Valentin Albillo** on 21 Feb 2008, 7:41 p.m.,*
in response to message #18 by Rodger Rosenbaum

Hi, Rodger:

Rodger posted:

> *"Now, on to the sums of 3 at a time case!"*

Fine with me but before seeking worthier pastures may I remind you that the question of the *uniqueness* or not of the solution for *general* N hasn't been settled yet.

You now know that for N=1,3,5 the solution is unique and for N=2,4 it's not. But what about arbitrary N ? Find that out and prove your savvy to the highly knowledgeable audience following this thread; it isn't trivial but that's why I call it a "mini-challenge" (full-fledged challenge, next April 1st)

After all, this is intended as a mere *training* for the incoming "Spring Special". If the difficulties here make you go for a premature retreat, the ones there will make you run screaming for the hills ! ... :-)

Best regards from V.

Re: Premature retreat (Re: Valentine's Day 2008 Mini-Challenge !)

Message #20 Posted by ***Egan Ford*** *on 22 Feb 2008, 2:29 p.m.,*
in response to message #19 by Valentin Albillo

Valentin, Rodger,

Here are two sets of 8 with the same pairwise sums.

```
1 5 7 9  9 11 13 17
2 4 6 8 10 12 14 16
```

So, I'm going to make a guess. If N can be expressed as 2^x where x is an integer, then the reversal of the pairwise sums may have multiple solutions.

Edited: 22 Feb 2008, 7:19 p.m. after one or more responses were posted

Re: Premature retreat (Re: Valentine's Day 2008 Mini-Challenge !)

Message #21 Posted by ***Steve Perkins*** *on 25 Feb 2008, 12:54 p.m.,*
in response to message #20 by Egan Ford

An interesting counter example to my best guess. I supposed that solutions were unique beyond N=5, and I felt fairly confident that this was true.

Guesses, feelings and hunches are not mathematical proofs. I clearly should have done a lot more investigating.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #22 Posted by **Egan Ford** on 22 Feb 2008, 1:34 p.m.,*
in response to message #18 by Rodger Rosenbaum

> Quote:
>
> Don't you mean: "...the number of correct identical solutions is < N!."? I assume you're referring to the fact that when there are duplicate original numbers, the sums include duplicates, and therefore the number of (distinct) permutations is reduced.

Yes and no. If you consider the permutations of unique values then yes < N!, but if you consider all permutations based on position and allow duplicate pair sums, then > N!.

I forgot to mention that there was an error in my post, PERM(S-4,N-5) should be COMB(S-4,N-5).

Re: Valentine's Day 2008 Mini-Challenge !

*Message #23 Posted by **Rodger Rosenbaum** on 21 Feb 2008, 11:49 p.m.,*
in response to message #17 by Egan Ford

You said:

> Quote:
>
> For the original problem there is no permutations, just one way to solve it. For N = 6 there are only 2 tests as Thomas has already pointed out.

And I said in a responding post, at the end:

> Quote:
>
> There may be a few small tweaks left.

It looks like we can continue taking advantage of special properties for a while longer.

For the case N = 6, we need the matrix (using Thomas' style):

```
[ 1 1 0 0 0 0 ] [ a ]   [ s1 ]
[ 1 0 1 0 0 0 ] [ b ]   [ s2 ]
[ 0 1 1 0 0 0 ] [ c ] = [ s3 ]
[ 1 1 1 1 1 1 ] [ d ]   [ s1+s2+s3+s4+s5+s6 ]
[ 0 0 0 1 0 1 ] [ e ]   [ s5 ]
[ 0 0 0 0 1 1 ] [ f ]   [ s6 ]
```

where the 3rd row has two possibilities, both of which must be tried: [0 1 1 0 0 0] and [1 0 0 1 0 0 0]; one of these equals the 3rd smallest sum. So two tests will solve the N=6 case.

For the N=7 case, we need:

```
[ 1 1 0 0 0 0 0 ] [ a ]   [ s1 ]
[ 1 0 1 0 0 0 0 ] [ b ]   [ s2 ]
[ 0 1 1 0 0 0 0 ] [ c ]   [ s3 ]
[ 1 0 0 1 0 0 0 ] [ d ] = [ s4 ]
[ 1 1 1 1 1 1 1 ] [ e ]   [ s1+s2+s3+s4+s5+s6+s7 ]
[ 0 0 0 0 1 0 1 ] [ f ]   [ s6 ]
[ 0 0 0 0 0 1 1 ] [ g ]   [ s7 ]
```

Here we know that the 3rd (element in the set of ordered sums) sum is either a+d or b+c. If the 3rd is a+d, then the 4th is b+c OR a+e; if the 3rd is b+c then the 4th is a+d. So we try the solution with the 3rd and 4th rows as shown just above. If that doesn't work, then make row 3 a+d and try row 4 = b+c OR row 4 = a+e, leaving the 3rd and 4th sums in the same positions in the column vector. One of the three will give the solution. So we need three tests for the N=7 case.

Thomas appears to have added a row both before and after the row of all ones and tried the 4 possible combinations. He didn't say exactly how he did it; he just showed the two extra rows. But I don't think you need 4 tests for the N=7 case; I think 3 will do it.

For the case N=8, consider:

```
[ 1 1 0 0 0 0 0 0 ] [ a ]   [ s1 ]
[ 1 0 1 0 0 0 0 0 ] [ b ]   [ s2 ]
[ 0 1 1 0 0 0 0 0 ] [ c ]   [ s3 ]
[ 1 0 0 1 0 0 0 0 ] [ d ]   [ s4 ]
[ 1 1 1 1 1 1 1 0 ] [ e ] = [ s1+s2+s3+s4+s5+s6+s7+s8 ]
[ 0 0 0 0 0 1 1 0 ] [ f ]   [ s6 ]
[ 0 0 0 0 0 1 0 1 ] [ g ]   [ s7 ]
[ 0 0 0 0 0 0 1 1 ] [ h ]   [ s8 ]
```

I've added another row after the row of all ones. It can take on two possibilities, [0 0 0 0 1 1 0] and [0 0 0 1 0 0 1], and the corresponding quantity in the column vector is the 3rd sum from the end. For each of the three possibilities for rows 3 and 4, try

each of the two possibilities for row 6. This will give the solution in 6 tries.

For the N=9 case, add another row (it becomes row 6) after the all ones row. Then go through the three possibilities for rows 3 and 4, and rows 6 and 7. So we can solve the N=9 case with 9 tries.

I hope I did this without too many stupid errors.

This is much better than the PERM(S-4,N-5) tries that would be needed if we gave up using special cases earlier.

I suppose this kind of thing can be carried even further, but it is less systematic than the all permutations method (although much more efficient), and it becomes more difficult to decipher the possibilities.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #24 Posted by **Egan Ford** on 22 Feb 2008, 2:44 p.m.,*
in response to message #23 by Rodger Rosenbaum

My early PERM(S-4,N-5) should have been COMB(S-4,N-5).

I had some time late last night to work on this again. My current solution firsts sets up the matrix:

```
| 1 1 0 ...           |
| 1 0 1 0 ...         |
| 1 0 0 1 0 ...       |
| ...                 |
| 1 1 1 1 1 ...       |
| 0 ...         1 0 1 |
| 0 ...           1 1 |
```

The first 2 rows are the smallest values, the last two the largest values, and the 3rd from last the sum of all pairs/(N-1).

The remaining rows follow the pattern x_1+x_4, x_1+x_5, ...

All that is left is to compute the COMB(S-4,N-5) combinations and test them in the remaining slots. There is no need to run permutations with each combination set since they should only be tested in increasing order to best match the pattern in A starting from the 3rd row down. There is no need to sort each combination array since the array is already sorted.

This solution is reasonably quick to find the first solution with N < 13. Reasonably quick being less time than to eat lunch. Finding all the solutions for N = 8, was very fast.

Edited: 22 Feb 2008, 4:03 p.m.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #25 Posted by **Bram** on 16 Feb 2008, 5:46 a.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin,

I found the time to do some thinking and programming.

For the first part of your again inspiring challenge I was considering the following steps.
1. the total of all the ten sums appears to be 4 times the total of the five unknown numbers, so sum_of_ABCDE=sum_of_10_sums/4.
2. I write some equations like this:
a. A+(B+C)+(D+E)=sum5
b. A+(B+D)+(C+E)=sum5
c. A+(B+E)+(C+D)=sum5
And so, if I compute from the ten sums another 45 sums of each couple of sums, I should get three times the same value, being (B+C)+(D+E) as well as the other combinations. Although I cannot tell which sum is which, I still can compute A by subtracting this value from sum5.
3. the same reasoning goes for all of the 5 numbers, so in the array of 45 sums I should see 5 times 3 equal numbers, from which each separate original number can be computed.

I've written a program for my HP-41CV in which the ten given sums must be in the regs 01 upto 10. It computes sum5 first. It then generates the 45 new sums and puts them in regs 40 upto 84 and finally hunts for the triplets in this array. It displays one by one the 5 original numbers. It does so in a reasonable time.
I guess that's things get complicated when the 5 numbers aren't all different, so I have to assume that for the program to work.

(listing of program will come later)

Re: Valentine's Day 2008 Mini-Challenge ! part 2

*Message #26 Posted by **Egan Ford** on 16 Feb 2008, 12:33 p.m.,*
in response to message #1 by Valentin Albillo

> Quote:
>
> ***"Write a program which, from the set of integers 1,2,...,9, selects and uses four of them to fill in a 3x3 matrix so that its determinant is 1 and replacing each element by its square still results in the determinant evaluating to 1."***

...and you must optimize your program primarily for speed for this particular notch.

Solution:

```
    | 3 3 5 |       |  9   9  25 |
det | 4 3 4 | = det | 16   9  16 | = 1
    | 4 5 9 |       | 16  25  81 |
```

I use an unimaginative bruce force attack on the problem and cracked it in 14 seconds with 50g/HPGGC3.

Output:

```
Valentine's Day 2008
  Mini-Challenge!
  Brute Force Attack!


Trying 1: 1 2 3 4
Trying 2: 1 2 3 5
Trying 3: 1 2 3 6
...
Trying 93: 3 4 5 7
Trying 94: 3 4 5 8
Trying 95: 3 4 5 9


Solution:


3 3 5 4 3 4 4 5 9
9 9 25 16 9 16 16 25 81


Time to Soltuion: 14 seconds
```

Algorithm: (for complete HPGCC3 code goto http://sense.net/~egan/v08det.c)

```
int det(int *a) {
    return(a[0]*a[4]*a[8] - a[0]*a[5]*a[7] - a[1]*a[3]*a[8] +
        a[1]*a[5]*a[6] + a[2]*a[3]*a[7] - a[2]*a[4]*a[6]);
}


void brute(int n) {
    int i;
```

```
    int static c = 0;
    for(i = c; i < 6+n ; i++) {
        if(done)
            return;
        A[n] = i + 1;
        if(n < 3) {
            c = i + 1;
            brute(n+1);
        }
        else
            force(0);
    }
}


void force(int n)
{
    int i, j;
    for(i = 3; i >= 0; i--) {
        if(done)
            return;
        B[n] = A[3-i];
        if(n < 8)
            force(n+1);
        else {
            if(det(B) == 1) {
                for(j=0;j<9;j++)
                    C[j] = B[j]*B[j];
                if(det(C) == 1) {
                    done = 1;
                    return;
                }
            }
        }
    }
}
```

> Quote:
>
> If you manage to solve this with relative ease, perhaps I'll feel "forced" to turn things up still *another* notch ! Have you got what it takes ? ;-)

Easy. :-) IMHO, The general solution to N in part one is the harder problem. I may have a UserRPL solution for that today.

The Final Notch (Re: Valentine's Day 2008 Mini-Challenge !)

*Message #27 Posted by **Valentin Albillo** on 18 Feb 2008, 10:55 a.m.,*
in response to message #26 by Egan Ford

Hi, Egan:

Egan boasted ..., er, *posted*:

"Easy. :-)"

so I've kept my promise and have added a final notch to the mini-challenge, you'll find it edited-in directly in my original post.

See whether you find it to your liking and considerable programming abilities.

Best regards from V.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #28 Posted by **Alex L** on 16 Feb 2008, 7:29 p.m.,*
in response to message #1 by Valentin Albillo

I took on part one. A five-liner for the 71B, eh? Well, I got it down to four, if you don't mind entering the sums one at a time. I'm looking forward to learning more about input processing.

This runs on a bare-bones, no-ROMs 71B. It's instant, as much of the work happens in the input loop.

```
10 T=0 @ L=MAXREAL @ M=L @ N=-M @ P=N
20 FOR C=1 TO 10 @ INPUT "Pairsum ";R @ T=T+R @ IF R>=P THEN N=P @ P=R ELSE N=MAX(N,R)
30 IF R<=L THEN M=L @ L=R ELSE M=MIN(M,R)
40 NEXT C @ S=T/4 @ C=S-L-P @ E=N-C @ D=P-E @ A=M-C @ B=L-A @ DISP A;B;C;D;E
```

Re: Valentine's Day 2008 Mini-Challenge !

*Message #29 Posted by **Steve Perkins** on 19 Feb 2008, 3:43 p.m.,*
in response to message #28 by Alex L

Very nice Alex!

This seems to head in the direction I wanted to go (before the holiday weekend distracted me).

Your solution is more elegant than I envisioned. I especially like how you didn't bother storing the sums at all, just kept the total sum and the 2 smallest and 2 largest values. Those are all it takes to solve the problem in the case of 5 original numbers.

Good job.

Re: Valentine's Day 2008 Mini-Challenge !

*Message #30 Posted by **Alex L** on 20 Feb 2008, 3:30 p.m.,*
in response to message #29 by Steve Perkins

Thanks! Of course, my solution's only valid because Valentin originally constrained the problem such that we could assume valid input.

And equally of course, my comment about why the solution is fast is a red herring. :)

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #31 Posted by **Egan Ford** on 18 Feb 2008, 9:32 p.m.,*
in response to message #1 by Valentin Albillo

> Quote:
>
> ***"Write a program to find four distinct integers such that the sum of any pair is a perfect square"***

I could have written this in UserRPL or BASIC, possibly RPN, but I selected C because I needed the speed for the *five distinct integers* variant of this problem. (And, HPGCC is my latest toy :-)

The program below takes an argument of 3, 4, or 5 and searches for that number of *distinct integers such that the sum of any pair is a perfect square.*

Output for 3, 4, and 5:

```
3:
1 24 120
Time to solution: 0 seconds


4:
Threes: 218
2 167 674 6722
Time to solution: 0 seconds
```

```
5:
Threes: 2124549
Fours:  6711
7442 28658 148583 177458 763442
Time to solution: 293 seconds
```

The "Threes" and "Fours" are the number of n-1 and/or n-2 sets found along the way.

Searching for 5 integers takes a long time at 192Mhz and 2.5x longer at 75Mhz (HPGCC2).

Thanks again Valentin for the challenges, and thanks to Mr. Glasbey for the algorithm. For an entertaining read pick up the March 1978 edition of *The Mathematical Gazette*.

Code:

```
#include <hpgcc49.h>


int find(int);
int checkit(int, int);
int isqrt(int);


//#define USE_TABLE
//#define USE_MOD_FILTER


#ifndef HPAPINE
#ifdef HPGCC2
extern unsigned int __heap_ptr, _heap_base_addr;
int freemem();
#endif
#endif


#ifdef USE_TABLE
char *ps;
int max, maxmax;
#endif


int main()
{
    int i, n, start;
#ifdef USE_TABLE
```

```
#ifndef HPAPINE
    max = sqrt(freemem() - 20000);
#else
    max = 617;
#endif
    maxmax = max * max;
    if ((ps = malloc(maxmax * sizeof(char))) == NULL)
        return (0);
    memset(ps, 1, maxmax * sizeof(char));
    for (i = 1; i < max; i++)
        ps[i * i] = 0;
#endif
    clear_screen();
#ifdef HPGCC2
    sys_slowOff();                    // 75Mhz, bummer
    n = sat_pop_zint_llong();
#ifdef HPAPINE
    n = sat_pop_real();
#endif
#else
    cpu_setspeed(192 * 1000000);    // 192Mhz
    n = sat3_pop_int(4);
#endif
    if (n > 5)
        n = 5;
    if (n < 3)
        n = 3;
    start = sys_RTC_seconds();
    find(n);
#ifdef USE_TABLE
    free(ps);
#endif
    printf("Time to solution: %d seconds", sys_RTC_seconds() - start);
#ifdef USE_TABLE
    printf("\nTable size: %d^2=%d bytes", max, maxmax);
#endif
#ifdef HPGCC2
    sys_slowOn();
    WAIT_CANCEL;
#else
    SLOW_WAIT_CANCEL;
#endif
    return (0);
}


int find(int n)
```

```
{
    register int a = 0, b, c, d, e, n1, n2, n3, n4, n5, a2, b2, c2;
    int threes = 0, fours = 0;
    for (;;) {
        a++;
        if (n == 5) {
            printf(" %3d", a);
            if (a % 8 == 0)
                printf("\n");
        }
        a2 = a * a;
        for (b = (a2 - 1) / 2 - 1; b > a; b--) {
            b2 = b * b;
            for (c = sqrt(a2 + b2); c > b; c--) {
                c2 = c * c;
                n1 = (a2 + b2 - c2) / 2;
                n2 = (a2 + c2 - b2) / 2;
                n3 = (b2 + c2 - a2) / 2;
                if (checkit(n1, n2) || checkit(n2, n3) || checkit(n1, n3))
                    continue;
                threes++;
                if (n == 3) {
                    printf("%d %d %d\n\n", n1, n2, n3);
                    return (1);
                }
                for (e = sqrt(n2 - n1); e > 0; e--) {
                    d = (n2 - n1 - e * e) / (2 * e);
                    n4 = d * d - n1;
                    if (n4 < 1 || n4 < n3 || n4 == n1 || n4 == n2
                        || n4 == n3)
                        continue;
                    if (checkit(n2, n4) || checkit(n3, n4))
                        continue;
                    fours++;
                    if (n == 4) {
                        printf("Threes: %d\n\n", threes);
                        printf("%d %d %d %d\n\n", n1, n2, n3, n4);
                        return (1);
                    }
                    break;
                }
                for (e = sqrt(n2 - n1); e > 0; e--) {
                    d = (n2 - n1 - e * e) / (2 * e);
                    n5 = d * d - n1;
                    if (n5 < 1 || n5 < n4 || n5 == n1 || n5 == n2
                        || n5 == n3 || n5 == n4)
                        continue;
```

```
                    if (checkit(n2, n5) || checkit(n3, n5)
                        || checkit(n4, n5))
                        continue;
                    if (n == 5) {
                        printf("\n\nThrees: %d\n", threes);
                        printf("Fours:  %d\n\n", fours);
                        printf("%d %d %d %d %d\n\n", n1, n2, n3, n4, n5);
                        return (1);
                    }
                }
            }
        }
    }
    return (0);
}


int checkit(int x, int y)
{
    register int a, b = x + y;
#ifdef USE_TABLE
    if (b < maxmax)
        return (ps[b]);
#endif
#ifdef USE_MOD_FILTER
    a = b % 9;
    if (a != 0 && a != 1 && a != 4 && a != 7)
        return (1);
    a = b % 5;
    if (a != 0 && a != 1 && a != 4)
        return (1);
    a = b % 7;
    if (a != 0 && a != 1 && a != 2 && a != 4)
        return (1);
    a = b % 13;
    if (a != 0 && a != 1 && a != 3 && a != 4 && a != 9 && a != 10 && a != 12)
        return (1);
    a = b % 17;
    if (a != 0 && a != 1 && a != 2 && a != 4 && a != 8 && a != 9 && a != 13 && a != 15 && a != 16)
        return (1);
#endif
    a = isqrt(b);
    if (b == a * a)
        return (0);
    return (1);
}
```

```
int isqrt(int x)
{
    int squaredbit, remainder, root;
    if (x < 1)
        return 0;
    squaredbit = (long) ((((unsigned long) ~0L) >> 1) & ~(((unsigned long) ~0L) >> 2));
    remainder = x;
    root = 0;
    while (squaredbit > 0) {
        if (remainder >= (squaredbit | root)) {
            remainder -= (squaredbit | root);
            root >>= 1;
            root |= squaredbit;
        } else
            root >>= 1;
        squaredbit >>= 2;
    }
    return root;
}


#ifndef HPAPINE
#ifdef HPGCC2
int freemem()
{
    register unsigned int stack_ptr asm("sp");
    unsigned int base;
    base = (__heap_ptr == 0) ? _heap_base_addr : __heap_ptr;
    return stack_ptr - base;
}
#endif
#endif
```

Edited: Optimizations to increase speed 4x.

Edited: More optimizations.

Edited: 20 Feb 2008, 8:52 p.m. after one or more responses were posted

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #32 Posted by **Paul Dale** on 19 Feb 2008, 3:19 p.m.,*
in response to message #31 by Egan Ford

I had a solution to this problem last night but was still working on getting it faster. Interestingly, it is different (smaller?):

```
    3362 482 359 2
```

There are many others of course.

Also, one major possibility for speeding up your program is to avoid a lot of the sqrt calls thus:

```
        #define MAX 100
        static unsigned char tbl[MAX*MAX];


        static inline int checkit(int x, int y) {
                return !tbl[x + y];
        }
```

and initialise the array via:

```
        for (i=0; i<MAX; i++)
                tbl[i*i] = 1;
```

On my desktop system this change provides a reasonable speed up even with hardware floating point support. For the paranoid, check if x+y >= MAX*MAX in checkit() and do the old code in this case.

- Pauli

edit: added a not in the code

Edited: 19 Feb 2008, 5:41 p.m.

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #33 Posted by **Paul Dale** on 19 Feb 2008, 7:36 p.m.,*
in response to message #32 by Paul Dale

Another speed up is to precalculate the sqrt calls:

```
        static int sqrttbl[MAX*MAX];
        static inline int isqrt(int x) {
                if (x > MAX*MAX)
                        return sqrt(x);
                return sqrttbl[x];
        }
```

Initialise this array via:

```
for (i=0, n=1; n<MAX*MAX; n++) {
        if (tbl[n])
                i++;
        sqrttbl[n] = i;
}
```

and change the existing calls to sqrt() to isqrt().

- Pauli

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #34 Posted by **Egan Ford** on 20 Feb 2008, 8:08 p.m.,*
in response to message #32 by Paul Dale

Hi Paul,

In the past I have used static tables (e.g. in the 80s I used cos/sin tables for faster real time 3d graphics), and dynamic tables (e.g. SSMC19 A+ 71B solution--more 80s tech) to increase critical parts of applications. I thought about something similar for this problem but quickly (too quickly) dismissed it without testing. Post 1980s processor performance has quickly outpaced memory performance. The number of clock cycles needed to access memory, especially a large table (i.e. cache miss), is often longer than calculating basic functions like sin, cos, sqrt. I assumed that this would also have been the case. But, I forgot that the ARM processor has no floating point processor. The same tricks I used in the 80s should apply here as well. Lets see...

The baseline for 3 and 4 integers is 0 seconds. The baseline for 5 integers is 451 seconds (initially it was 2000+ seconds, but a few small algorithm optimizations dropped this down to 451). BTW, none of the optimization changes noticeably increased or decreased the performance of the 3 and 4 (original problem) integer cases. They run so fast that the results are present before you can remove your finger from the button that launched it.

Late last night I was able to work on checkit optimization. My first test was to reduce the impact of sqrt with isqrt. Unfortunately HPGCC3 does not have an isqrt function, so I wrote one, a poor one, and performance was a bit worse (2 seconds slower for the 5 integer case). So I went shopping and took the first Google hit for "isqrt". The faster isqrt reduced the time from 451 to 293 seconds.

For my 2nd test I wanted to find a way to call sqrt as little as possible. The following code should identify 99.25% of non-square numbers. I measured a 61% hit rate. Of the 39% of the numbers that slipped through, 11% were non-square and 89% were perfect squares. The "mod filter" caught ~96% of the non-square numbers.

```
register int a, b = x + y;
a = b % 9;
if (a != 0 && a != 1 && a != 4 && a != 7)
```

```
        return (1);
    a = b % 5;
    if (a != 0 && a != 1 && a != 4)
        return (1);
    a = b % 7;
    if (a != 0 && a != 1 && a != 2 && a != 4)
        return (1);
    a = b % 13;
    if (a != 0 && a != 1 && a != 3 && a != 4 && a != 9 && a != 10 && a != 12)
        return (1);
    a = b % 17;
    if (a != 0 && a != 1 && a != 2 && a != 4 && a != 8 && a != 9 && a != 13 && a != 15 && a != 16)
        return (1);
```

This code reduced the time from 451 to 351 seconds.

My 3rd test used your suggestion of a true/false table for perfect squares. I collected the following data:

MAX	Bytes	Time(s)	Hit Rate
100	10000	451	2%
200	40000	423	18%
300	90000	413	25%
447	199809	393	36%
617	380689	385	45%

The table method works well, but my 50g runs out of RAM with MAX=617. Even if I had the ~1MB of RAM required to hold a table large enough for a 100% hit rate the estimated time to solution is ~312 seconds. Faster than my 2nd test (mod filter), but slower than isqrt. Evidence that computing basic functions like sqrt on modern machines will be faster than RAM lookup tables.

Next I tried to combine methods. isqrt is so far ahead of the other methods that I expect that they may have little or no impact:

Uni	Time(s)
baseline	451
isqrt	293
mod filter	351
table	385

Combo	Time(s)
table + mod filter	329
table + isqrt	295

```
mod filter + isqrt              292
table + mod filter + isqrt      294
```

Table(45%) + isqrt(55%) is a bit slower that just isqrt. I expect that more table hits vs. isqrt hits will be even slower.

Mod filter (61%) + isqrt(39%) only improved by 1 second.

The combination of the three (table(45%) + mod filter(34%) + isqrt(21%)) gained 1 second.

IMHO, the most elegant solution is isqrt. It also has a small memory footprint.

I've updated my original code above with all three optimizations, but left the table lookup and mod filter commented out.

P.S. I forgot to add that I ran all of the above at 192MHz. Running the 50g at the default of 75MHz, the table lookup may have performed the fastest. Even faster than isqrt. It is not uncommon for memory access performance to be independent of processor MHz.

Edited: 20 Feb 2008, 8:39 p.m. after one or more responses were posted

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #35 Posted by **Paul Dale** on 20 Feb 2008, 8:35 p.m.,*
in response to message #34 by Egan Ford

Great analysis!

I tried isqrt on my desktop machine and it was significantly slower than the table approach. Of course, the tables are relatively small compared to modern CPU caches so lookups will be fast.

The ARM has a tiny cache and is very good at the kind of operations involved in the sqrt calculation.

- Pauli

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #36 Posted by **Egan Ford** on 20 Feb 2008, 8:55 p.m.,*
in response to message #35 by Paul Dale

You may want to try the isqrt that I used in my amended post. On my PC isqrt and table lookup was the same speed. Perhaps I have smaller L1/L2 caches.

I think table lookup will be faster at 75Mhz on the 50g. If I have time I will try it tonight.

Re: Valentine's Day 2008 Mini-Challenge ! part 3

*Message #37 Posted by **Egan Ford** on 21 Feb 2008, 2:07 a.m.,*
in response to message #34 by Egan Ford

> Quote:
>
> P.S. I forgot to add that I ran all of the above at 192MHz. Running the 50g at the default of 75MHz, the table lookup may have performed the fastest. Even faster than isqrt. It is not uncommon for memory access performance to be independent of processor MHz.

Here are some 75Mhz/HPGCC2/50g numbers:

```
isqrt                   837s
table + isqrt           825s
mod filter + isqrt      971s
```

When running at slower processor speeds, table lookup adds the most benefit.

Re: Valentine's Day Mini-Challenges !: My original solutions & comments

*Message #38 Posted by **Valentin Albillo** on 25 Feb 2008, 7:45 p.m.,*
in response to message #1 by Valentin Albillo

Hi all,

Thanks to all of you for your extremely interesting inputs to this VD 2008 Mini-challenge, these are my original solutions plus assorted interspersed comments:

First notch

For the specific case of five numbers, there's no need to construct a system of linear equations, overdetermined or not. As some of you guessed, we can identify both the sum of the two smallest values and that of the two largest ones. This, plus the fact that the sum of all five values equals 4 times the total of all pairwise sums, is enough to find all values via a few simple arithmetic operations.

My original program for the HP-71B does exactly that. I used arrays to input and output all values at once, plus sorting the input. However, as some of you did, it's actually possible to avoid storing and sorting the input values and simply keep a tally of them on the fly instead, for a shorter and simpler program. Mine was the following 5-liner:

```
1 DESTROY ALL @ OPTION BASE 1 @ N=10 @ DIM X(5),A(N) @ MAT INPUT A
2 FOR I=1 TO N @ FOR J=I+1 TO N @ IF A(I)>A(J) THEN K=A(J) @ A(J)=A(I) @ A(I)=K
3 NEXT J @ NEXT I @ S=0 @ FOR I=1 TO N @ S=S+A(I) @ NEXT I
4 X(3)=S/4-A(1)-A(N) @ X(1)=A(2)-X(3) @ X(5)=A(9)-X(3)
5 X(2)=A(1)-X(1) @ X(4)=A(N)-X(5) @ MAT DISP X;

 >RUN

    A(1)?  3734,3768,284,3950,466,4000,516,500,3966,3784


      -1492    1776    1958    1992    2008

 >RUN

    A(1)?  -0.4233,-1.7274,-2.4485,-0.9055,4.1325,3.4114,4.9544,2.1073,3.6503,2.9292


      -3.1416   .6931   1.4142   2.2361   2.7183

 >RUN

    A(1)?  1, -5, 7, -17, -2, 10, -14, 4, -20, -8

      -16  -4  -1   2   8

 >RUN

    A(1)?  -22, -4, 118, 4, 126, 144, -31, -23, -5, 117

      -16  -15  -7   11   133
```

```
>RUN

    A(1)?  -34.71,23.992,-25.094,-15.1,-2.696,16.914,4.382,6.92,-32.172,14.376


      -32.093   -2.617   -0.079    6.999    16.993
```

For arbitrary N, the solution, when it exists, is unique *except when N is a power of 2*, where distinct sets of N numbers having the exact same pairwise sums can be found. Thus, this is the case for N=2, N=4, N=8, N=16, ..., N=1048576, ... , N=33554432, ... you get the point.

For N not a power of 2, there's no efficient method known to compute the unique solution, where *efficient* means it runs in polynomial or near-polynomial time. Solving a number of systems covering all necessary combinations is an exponentially bounded procedure, thus inefficient and ultimately unfeasible for sufficiently large values of N.

Intermediate notch

This yields to brute force and I took that approach, doing very little thinking and letting the machine do all the work instead for a change. My original program for the HP-71B is thus simply the following no-brainer 10-liner affair:

```
 1  DESTROY ALL @ OPTION BASE 1 @ DIM T(9)
 2  FOR A=1 TO 9 @ DISP A @ FOR B=1 TO 9 @ FOR C=1 TO 9 @ FOR D=1 TO 9 @ FOR E=1 TO 9
 3  MAT T=ZER @ T(A)=1 @ T(B)=1 @ T(C)=1 @ T(D)=1 @ T(E)=1 @ IF CNORM(T)=5 THEN 10
 4  FOR F=1 TO 9 @ FOR G=1 TO 9 @ FOR H=1 TO 9 @ FOR I=1 TO 9
 5  MAT T=ZER @ T(A)=1 @ T(B)=1 @ T(C)=1 @ T(D)=1 @ T(E)=1 @ T(F)=1 @ T(G)=1 @ T(H)=1 @ T(I)=1
 6  IF CNORM(T)#4 THEN 9 ELSE IF A*(E*I-F*H)-B*(D*I-F*G)+C*(D*H-E*G)#1 THEN 9
 7  IF A*A*(E*E*I*I-F*F*H*H)-B*B*(D*D*I*I-F*F*G*G)+C*C*(D*D*H*H-E*E*G*G)#1 THEN 9
 8  DISP A;B;C;D;E;F;G;H;I @ END
 9  NEXT I @ NEXT H @ NEXT G @ NEXT F
10  NEXT E @ NEXT D @ NEXT C @ NEXT B @ NEXT A @ DISP "OK" @ END
```

which is far, far from optimal but does the job if you simply let it run for a while (though a fast emulator such as Emu71 is recommended unless you want to flatten out the batteries):

```
>RUN
          1
```

```
          2
          3
              3  3  5  4  3  4  4  5  9
```

$$\text{and thus we have: } \det\begin{pmatrix} 3 & 3 & 5 \\ 4 & 3 & 4 \\ 4 & 5 & 9 \end{pmatrix} = \det\begin{pmatrix} 3^2 & 3^2 & 5^2 \\ 4^2 & 3^2 & 4^2 \\ 4^2 & 5^2 & 9^2 \end{pmatrix} = 1$$

The only "trick" worth mentioning in the above program is the use of CNORM to count how many different values are we dealing with, which is used as a cutoff at two places to save unnecessary looping or computing the determinants. This is still very far from optimal but I was feeling pretty lazy at the time ... The problem doesn't extend to cubes but I find it nice that the solution is unique.

The final notch

Another lazy effort on my part but at least this 10-liner for the HP-71B runs *fast*: less than one second in Emu71 (a few minutes in a physical HP-71B) to produce the three solutions shown (and the corresponding square sums; there are infinite solutions):

```
 1  DESTROY ALL @ OPTION BASE 1 @ DIM Y(3) @ FOR N=0 TO 1000
 2  M=0 @ FOR A=IP(SQR(N/2))+1 TO IP(SQR(N)) @ B=SQR(N-A*A)
 3  IF NOT B OR FP(B) THEN 4 ELSE M=M+1 @ Y(M)=B @ IF M=3 THEN 5
 4  NEXT A @ IF M#3 THEN 10
 5  P=Y(1) @ Q=Y(2) @ R=Y(3) @ U=(Q*Q+R*R-P*P)/2 @ IF FP(U) THEN 10
 6  V=(R*R+P*P-Q*Q)/2 @ W=(P*P+Q*Q-R*R)/2 @ FOR X=-U TO N @ IF X=U OR X=V OR X=W THEN 9
 7  IF FP(SQR(U+X)) OR FP(SQR(V+X)) OR FP(SQR(W+X)) THEN 9
 8  DISP U;V;W;X,U+V;U+W;U+X;V+W;V+X;W+X
 9  NEXT X
10  NEXT N

>RUN

   -40   65  104  296      25   64  256  169  361  400
   -94   98  263  578       4  169  484  361  676  841
  -110  135  306  594      25  196  484  441  729  900
```

as you can see, all sets of six sums are perfect squares, as required. Extensive optimization or a different algorithm are possible, but for a mini-challenge on my nameday it's fine by me as it goes.

Thanks for your interest in this humble mini-challenge, you've shown tremendous insight and programming muscle as opposed to my admitted laziness this one time.

However, this was but a *training* for the incoming ***S&SMC#20*** *full-fledged challenge*, where you'll see some of the most *devious* math teasers known to man despite extremely simple wording, and further I've gone to serious lengths to provide quality, amazing original solutions to each and everyone of them for assorted HP models, despite it seeming utterly impossible at times.

Let's hope that momentous event doesn't find *you* affected with the lazy bug ! See you next April 1st ! :-)

Best regards from V.

Edited to correct a typo

Edited: 25 Feb 2008, 7:48 p.m.

[Return to Index | Top of Index]

Go back to the main exhibit hall