HP Forum Archive 17

[Return to Index | Top of Index]

HP42S Mini-Challenge: Optimizing !

*Message #1 Posted by **Valentin Albillo** on 1 May 2007, 10:57 p.m.*

Hi all,

A new month's just begun and a little, HP42S-specific Mini-Challenge might prove an adequate welcome for it and a nice ocassion to flex your HP42S programming muscles. So this is

---

The Mini-Challenge

We say that a positive integer is *palindromic* if it reads the same forwards and backwards, such as 1771 or 32823. Some *integer squares* can also be palindromic, such as $121 = 11^2$.

There is *one and only one 12-digit palindromic square* and you must write an HP42S program to try and find it, your choice(s) among three possible flavours:

1. Optimized for *minimum size*, regardless of speed
2. Optimized *primarily for speed*, secondarily for size
3. Optimized for *maximum speed*

Some notes:

- The goal for (1) is to achive the smallest possible program, even if it would take extremely long to find the unique answer.
- The goal for (2) is to find the answer as quick as possible but with some concern as to program size and reasonably "elegant" code.

- The goal for (3) is to find the answer as fast as possible, even at the expense of longer or not-so-elegant code.
- Both (2) and (3) must be able to find the solution in a *physical* HP42S in a few hours at most.

Your program must stop as soon as it finds the unique answer (no need to search for more) and the usual caveats apply, namely:

- Avoid *googling* for the answer, any moron can do that and it would be extremely *lame*. The idea is to work it out and show some *programming muscle* and good, worthwhile *ideas* which might be useful for other tasks. If you can't solve this on your own, you aren't worth your salt as an HP-calc programmer.
- Using reasonable, sound *heuristics* which can be rationalized in a few words is Ok and recommended, but avoid doing all the theoretical work yourself with paper and pen, then have the program simply print the answer after looking at a handful cases. The idea is for *the machine* to do the bulk of the work, not *you*.
- This mini-challenge is HP42S-specific in the sense that I'll post my original solutions which are HP42S RPN programs using some neat tricks that are indeed HP42S-specific. It would be nice if you would write HP42S programs too, you can always use some good emulator (Emu42 or Free42, for instance) if you don't have a physical HP42S.

  That failing, you may nevertheless use any 12-digit HP calc model and see what you come up with.

I'll post my original solutions for the HP42S within a few days, as always, which, as a guide to measure your efforts, have the following characteristics:

1. Trivial but takes ages
2. 72 steps (157-byte), R00-R11 used, finds the unique solution in 2 h 42 min in a physical HP42S, 85 seconds in Emu42 @ 2.4 Ghz, 44 seconds in Free42 @ Palm Z100
3. 129 steps (239-byte), R00-R11 used, finds the unique solution in 1 h 48 min in a physical HP42S, 56 seconds in Emu42 @ 2.4 Ghz, 29 seconds in Free42 @ Palm Z100

Enjoy ! :-)

Best regards from V.

Re: HP42S Mini-Challenge: Optimizing !

*Message #2 Posted by **Egan Ford** on 2 May 2007, 12:31 p.m.,*
in response to message #1 by Valentin Albillo

Thanks for this challenge. My 42S has been collecting dust since I use my 15C, 71B, and 50G for challenges.

Here is my first min size entry. I didn't have a lot of time to work on it. I know there is room for improvement.

This is a brute force search starting at 999999 and working down. Once squared a decimal point is used to separate the 2 halves. The right half is flipped and subtracted from the left half. If the result = 0, then you have found it.

40 steps, 66 bytes. Free42 on my notebook breezed through this in about 40 seconds. ETA for physical 42S: 95-96 hours. I do not have any EMU42 timings. I am too lazy to dump my 42S ROM to my 48GX so that I can use EMU42. I hate to ask this, but can someone please email a ROM dump (I can provide proof that I own a 42S if necessary).

This does not use any unique feature of the 42S so it should be portable to other models (e.g. 15C, however you'll have to settle for a 9 digit palindromic square, there is no 10).

```
 1 LBL "PP"
 2 1E6
 3 STO 00
 4 LBL A
 5 1
 6 STO- 00
 7 6
 8 STO 03
 9 0
10 STO 02
11 RCL 00
12 X^2
13 1E6
14 /
15 FP
16 LASTX
17 IP
18 STO 01
19 LBL B
20 X<>Y
21 10
22 x
23 FP
24 LASTX
25 IP
26 6
```

```
27 RCL 03
28 -
29 10^X
30 x
31 STO+ 02
32 DSE 03
33 GTO B
34 RCL 01
35 RCL 02
36 -
37 X!=0?
38 GTO A
39 RCL 00
40 X^2
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #3 Posted by **Paul Dale** on 2 May 2007, 5:44 p.m.,*
in response to message #1 by Valentin Albillo

[edit: added solution time]

Interesting challenge as usual.

I cannot think of a "clever/short" method to do the palindrome generation on a 42S, however I did some up with this fragment on the HP49g+:

```
->STR
DUP
SREV
==
```

You'll need to attach library 256 for the SREV to be resolved properly (i.e. 256 ATTACH before entering the above fragment. You'll also want everything to stay integer (i.e. not approximate mode).

A final program that performs the search:

```
<<
  6
  ALOG            10^x
  WHILE
    1.
  REPEAT
    1
    -
```

```
    DUP
    SQ              x^2
    ->STR
    DUP
    SREV
    ==
    << HALT >>
    IFT
  END
>>
```

70.5 bytes, checksum #AF30h

Running time is a couple of minutes under an hour for an agumented version that inserted a TICKS before the HALT.

- Pauli

Edited: 2 May 2007, 7:32 p.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #4 Posted by **hugh steers** on 2 May 2007, 6:17 p.m.,*
in response to message #1 by Valentin Albillo

hi valentin,

i started wondering if the so-called "196" algorithm was a good way to search for palindromes of a given size.

196 algorithm: given a number, reverse its digits, add to original number. repeat process until palindromic (or give up after too big). i figure that since your number is unique, unless the 196 can't hit a 12 digit number, it will find it by chance fairly quickly.

sure enough i get it after putting in 899. so it took me 799 trials (started at 100). end of the output looks like this:

```
870 8836886388
871 15851
872 1661
873 2772
874 3883
875 4994
876 23232
877 47674
878 878
879 0
```

```
880 0
881 233332
882 1881
883 2992
884 7117
885 9339
886 1136311
887 0
888 888
889 881188
890 881188
891 79497
892 3113
893 5335
894 7557
895 9779
896 22022
897 46464
898 898
899 133697796331
woot!
```

not sure if this would make a good approach for a 42 program. probably not.

Re: HP42S Mini-Challenge: Optimizing !

*Message #5 Posted by **Paul Dale** on 2 May 2007, 6:26 p.m.,*
in response to message #4 by hugh steers

> Quote:
>
> ```
> 899 133697796331
> ```

I'm not sure you're on the right track here. You've found a 12 digit palindrome but 133697796331 is not a square number.

Palindromes of this size are very easy to find, choose any six digit number not ending with a zero reverse the digits and prepend. Finding the unique one which is also a square is harder.

- Pauli

Re: HP42S Mini-Challenge: Optimizing !

*Message #6 Posted by **Egan Ford** on 2 May 2007, 10:46 p.m.,*
in response to message #1 by Valentin Albillo

My first attempt at fast. I have not tested on real 42S yet, but with Free42 it runs in about 10 sec.

This differs from my brute force attempt in a number of ways.

Instead of calculating squares with X^2, squares are calculated by subtracting odd numbers. (A perfect square N^2 is = to the sum of the first N odds). By using the squares I can cut the numbers of searches by half because perfect squares cannot end in 8,7,3 or 2. I can also skip 0 since palindromic numbers do not start with 0. So I only check numbers starting with 9,6,5,4 and 1.

There is no need to check every square, only check the squares ending with the starting number. I can determine the next square with that matches the lead digit by multiplying the next square deference by 10 and subtracting 90. This has to be done twice because there are two patterns of matching digits spaced 10 squares apart. This reduces the number of searches by 4/5ths. (A simple mod 10 check may be faster).

Overall the total number of iterations required vs. brute force is 1/10.

To further increase performance the check for palindromic exits early if any number does not match (brute force checked all digits). Loop unrolling also helps. This part needs more help.

Like my earlier post this is not 42S specific.

I'll include my Perl prototype, it is easier to read.

```
1       LBL "PP2"
2       9
3       0
4       XEQ B
5       9
6       1
7       XEQ B
8       6
9       0
10      XEQ B
11      6
12      1
13      XEQ B
14      5
15      0
16      XEQ B
17      5
18      1
```

```
19      XEQ B
20      4
21      0
22      XEQ B
23      4
24      1
25      XEQ B
26      1
27      0
28      XEQ B
29      1
30      1
31      XEQ B
32      LBL B
33      STO 04
34      X<>Y
35      STO 03
36      1
37      +
38      11
39      10^X
40      x
41      1
42      -
43      SQRT
44      IP
45      X^2
46      STO 05
47      LASTX
48      1
49      -
50      X^2
51      -
52      STO 06
53      LBL 00
54      RCL 05
55      10
56      MOD
57      RCL 03
58      X=Y?
59      GTO 01
60      RCL 06
61      STO- 05
62      2
63      STO- 06
64      GTO 00
65      LBL 01
```

```
66      RCL 04
67      X=0?
68      GTO 02
69      0
70      STO 04
71      RCL 06
72      STO- 05
73      2
74      STO- 06
75      GTO 00
76      LBL 02
77      RCL 03
78      11
79      10^X
80      *
81      STO 07
82      LBL 04
83      RCL 07
84      RCL 05
85      X<Y?
86      RTN
87      1E6
88      /
89      IP
90      LASTX
91      FP
92      STO 02
93      X<>Y
94      STO 01
95      10
96      MOD
97      10
98      RCLx 02
99      IP
100     X!=Y?
101     GTO 03
102     1E-1
103     RCLx 01
104     IP
105     10
106     MOD
107     10
108     RCLx 02
109     FP
110     10
111     x
112     IP
```

```
113     X!=Y?
114     GTO 03
115     1E-2
116     RCLx 01
117     IP
118     10
119     MOD
120     1E2
121     RCLx 02
122     FP
123     10
124     x
125     IP
126     X!=Y?
127     GTO 03
128     1E-3
129     RCLx 01
130     IP
131     10
132     MOD
133     1E3
134     RCLx 02
135     FP
136     10
137     x
138     IP
139     X!=Y?
140     GTO 03
141     1E-4
142     RCLx 01
143     IP
144     10
145     MOD
146     1E4
147     RCLx 02
148     FP
149     10
150     x
151     IP
152     X!=Y?
153     GTO 03
154     1E-5
155     RCLx 01
156     IP
157     10
158     MOD
159     1E5
```

```
160     RCLx 02
161     FP
162     10
163     x
164     IP
165     X!=Y?
166     GTO 03
167     GTO E
168     LBL 03
169     10
170     RCLx 06
171     90
172     -
173     STO- 05
174     20
175     STO- 06
176     GTO 02
177     LBL E
178     RCL 05
179     STOP
```

```
#!/usr/bin/perl
$lc=0;
print "9...\n";
a(9,0);
a(9,1);
print "6...\n";
a(6,0);
a(6,1);
#will not get here;
a(5,0);
a(5,1);
a(4,0);
a(4,1);
a(1,0);
a(1,1);
sub a
{
        my $leaddigit = shift;
        my $nexthigh = shift;
        my $highnum = ($leaddigit + 1) * 10**11 - 1;
        my $highsqrt = int(sqrt($highnum));
        my $nexthighsqrt = $highsqrt - 1;
        my $highsqr = $highsqrt**2;
        my $odd = $highsqr - $nexthighsqrt**2;
        while($highsqr % 10 != $leaddigit) {
                $highsqr -= $odd;
```

```
                $odd -= 2;
        }
        if($nexthigh) {
                $highsqr -= $odd;
                $odd -= 2;
                while($highsqr % 10 != $leaddigit) {
                        $highsqr -= $odd;
                        $odd -= 2;
                }
        }
        while($highsqr > $leaddigit*10**11) {
                $lc++;
                my $a = $highsqr;
                my $left = int($a / 1e6);
                my $right = $a - int($a / 1e6) * 1e6;
                if($left - reverse($right) == 0) {
                        $s = $a**.5;
                        print $s . "\t" . $a . "\n";
                        print "loops:\t" . $lc . "\n";
                        exit(0);
                }
                $highsqr -= $odd * 10 - 90;
                $odd -= 20;
        }
}
```

Edited: 3 May 2007, 2:44 a.m.

Re: HP42S Mini-Challenge: Optimizing ! (Doh! Bug!)

*Message #7 Posted by **Egan Ford** on 4 May 2007, 12:55 a.m.,*
in response to message #6 by Egan Ford

Line 80 above:

```
80      *
```

Should be

```
80      x
```

txt2raw.pl will ignore the *. The above takes 10 times longer than it should. Below is a fixed version with a few other cleanups. Still not 42S specific.

```
Run times:
```

```
Free42 @ 1.7 GHz Pentium M:   1 sec
Emu42  @ 1.7 GHz Pentium M:  60 sec
Free42 @ 471 MHz PXA255:     70 sec
```

I'll key this in my 42S and report the time this weekend.

```
00 { 343-Byte Prgm }
01>LBL "PP4"
02 9
03 0
04 XEQ B
05 9
06 1
07 XEQ B
08 6
09 0
10 XEQ B
11 6
12 1
13 XEQ B
14 5
15 0
16 XEQ B
17 5
18 1
19 XEQ B
20 4
21 0
22 XEQ B
23 4
24 1
25 XEQ B
26 1
27 0
28 XEQ B
29 1
30 1
31>LBL B
32 STO 04
33 X<>Y
34 STO 03
35 1
36 +
37 11
38 10^X
39 ×
```

```
40 1
41 -
42 SQRT
43 IP
44 X^2
45 STO 05
46 LASTX
47 1
48 -
49 X^2
50 -
51 STO 06
52>LBL 00
53 RCL 05
54 10
55 MOD
56 RCL 03
57 X=Y?
58 GTO 01
59 RCL 06
60 STO- 05
61 2
62 STO- 06
63 GTO 00
64>LBL 01
65 RCL 04
66 X=0?
67 GTO 02
68 0
69 STO 04
70 RCL 06
71 STO- 05
72 2
73 STO- 06
74 GTO 00
75>LBL 02
76 11
77 10^X
78 RCL× 03
79 STO 07
80 GTO 04
81>LBL 03
82 10
83 RCL× 06
84 90
85 -
86 STO- 05
```

```
87 20
88 STO- 06
89>LBL 04
90 RCL 07
91 RCL 05
92 X<Y?
93 RTN
94 1E6
95 ÷
96 IP
97 LASTX
98 FP
99 STO 02
100 X<>Y
101 STO 01
102 10
103 MOD
104 10
105 RCL× 02
106 IP
107 X!=Y?
108 GTO 03
109 0.1
110 RCL× 01
111 IP
112 10
113 MOD
114 10
115 RCL× 02
116 FP
117 10
118 ×
119 IP
120 X!=Y?
121 GTO 03
122 0.01
123 RCL× 01
124 IP
125 10
126 MOD
127 100
128 RCL× 02
129 FP
130 10
131 ×
132 IP
133 X!=Y?
```

```
134 GTO 03
135 1E-3
136 RCL× 01
137 IP
138 10
139 MOD
140 1E3
141 RCL× 02
142 FP
143 10
144 ×
145 IP
146 X!=Y?
147 GTO 03
148 1E-4
149 RCL× 01
150 IP
151 10
152 MOD
153 1E4
154 RCL× 02
155 FP
156 10
157 ×
158 IP
159 X!=Y?
160 GTO 03
161 1E-5
162 RCL× 01
163 IP
164 10
165 MOD
166 1E5
167 RCL× 02
168 FP
169 10
170 ×
171 IP
172 X!=Y?
173 GTO 03
174 RCL 05
175 ENTER
176 SQRT
177 STOP
178 .END.
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #8 Posted by **J-F Garnier** on 3 May 2007, 3:11 p.m.,*
in response to message #1 by Valentin Albillo

Well, here is my solution for the HP-42S:

```
00 { 156-Byte Prgm }
01>LBL "PAL12"
02 FIX 00
03 CF 29
04 1E11
05 STO 00
06 1
07 1
08 XEQ A
09 9
10 1
11 XEQ A
12 2
13 4
14 XEQ A
15 8
16 4
17 XEQ A
18 5
19 5
20 XEQ A
21 4
22 6
23 XEQ A
24 6
25 6
26 XEQ A
27 3
28 9
29 XEQ A
30 7
31 9
32 XEQ A
33 RTN


34>LBL A
35 RCL ST X
36 1
37 +
```

```
38 RCL× 00
39 SQRT
40 STO 02
41 Rv
42 RCL× 00
43 SQRT
44 IP
45 RCL ST X
46 10
47 MOD
48 -
49 +
50 STO 01
51>LBL 00
52 X^2
53 XEQ 01
54 RCL 02
55 RCL 01
56 10
57 +
58 STO 01
59 X<Y?
60 GTO 00
61 RTN


62>LBL 01
63 CLA
64 ARCL ST X
65 6
66 STO ST L
67>LBL 02
68 -1
69 AROT
70 ATOX
71 ATOX
72 X!=Y?
73 RTN
74 DSE ST L
75 GTO 02
76 RCL 01
77 X^2
78 VIEW ST X
79 STOP
80 RTN
```

And the HP-71B version, for free:

```
10 ! find 12-digit square palindrome
20 T=TIME
30 N=100000000000 ! 1E11
40 ! squares ending with '1', try numbers ending with '1' or '9'
50 K=1 @ R=1 @ GOSUB 150 @ R=9 @ GOSUB 150
60 ! squares ending with '4', try numbers ending with '2' or '8'
70 K=4 @ R=2 @ GOSUB 150 @ R=8 @ GOSUB 150
80 ! squares ending with '5', try numbers ending with '5'
90 K=5 @ R=5 @ GOSUB 150
100 ! squares ending with '6', try numbers ending with '4' or '6'
110 K=6 @ R=4 @ GOSUB 150 @ R=6 @ GOSUB 150
120 ! squares ending with '9', try numbers ending with '3' or '7'
130 K=9 @ R=3 @ GOSUB 150 @ R=7 @ GOSUB 150
140 END
150 A=(SQR(N*K) DIV 10)*10+R @ B=SQR(N*(K+1))
160 FOR I=A TO B STEP 10
170 A$=STR$(I*I)
180 IF A$=REV$(A$) THEN DISP I;A$;TIME-T @ PAUSE
190 NEXT I
200 RETURN
```

May I add that these programs can be slightly modified to find the only one 8-digit hexadecimal palindromic square?

J-F

Re: HP42S Mini-Challenge: Optimizing !

*Message #9 Posted by **Alex L** on 3 May 2007, 3:23 p.m.,*
in response to message #1 by Valentin Albillo

This is my first-ever submission to one of these challenges. I was aiming at part (1), but mostly just learning how to program my 42S.

41-byte, 25-step program, at the end displays both the base number in Y and the sought square in X. Uses only R01 and ALPHA. Fairly 42S-specific, I think, as it depends on the existence of a 12-digit integer representation to move into the ALPHA register with AIP.

I had started counting up from 316228, but then realized that I could save 3 bytes in step 01 by counting down from 1E6. Turns out to save execution time, too.

18s on Free42 @ 2.1GHz, and based on two partial runs, an estimated 11h25m on a physical 42S. Since I'm an HP calculator programming rookie, I'd guess someone on this forum can make this algorithm even shorter.

```
01  1E6
02  STO 01
03  LBL 01
```

```
04  1
05  STO- 01
06  RCL 01
07  X^2
08  CLA
09  AIP
10  LBL 02
11  ALENG
12  2
13  X>Y?
14  GTO 03
15  -1
16  AROT
17  ATOX
18  ATOX
19  X=Y?
20  GTO 02
21  GTO 01
22  LBL 03
23  RCL 01
24  ENTER
25  X^2
```

Edited: 3 May 2007, 3:28 p.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #10 Posted by **Karl Schneider** on 4 May 2007, 12:18 a.m.,*
in response to message #9 by Alex L

Hello, Alex --

> Quote:
>
> This is my first-ever submission to one of these challenges. I was aiming at part (1), but mostly just learning how to program my 42S.

Well, I'm very impressed with your elegant program. I believe that you've taken the approach that Valentin had in mind, and which I didn't quite see how to do, having not researched all available HP-42S string commands.

> Quote:
>
> Fairly 42S-specific, I think, as it depends on the existence of a 12-digit integer representation to move into the ALPHA register with AIP.

AIP (or J-F Garnier's "ARCL ST X") is indeed the key to making this approach work. If I were to tackle this problem with a C-language program, I would use "sprintf" or a non-standard library routine "itoa" to convert an integer to a string, then compared pairs of digits, working from the outside inward. I thought that the HP-42S "ATOX" and "XTOA" would be analogous, but they weren't. However, ATOX *in conjunction with* AIP and AROT was just the ticket.

I've taken a similar approach, using numerical calculations instead of string operations to extract and remove the first and last digits of N^2. It's slower and more cumbersome.

> Quote:
>
> I'd guess someone on this forum can make this algorithm even shorter.

I don't see any particular way to make your algorithm much more concise, but the execution speed could be improved by using some basic heuristics that restrict, within appropriate ranges of N, which last-digit values of N^2 to even check for.

Good job!

-- KS

Edited: 4 May 2007, 12:52 a.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #11 Posted by **Alex L** on 4 May 2007, 11:16 a.m.,*
in response to message #10 by Karl Schneider

Karl,

> Quote:
>
> Well, I'm very impressed with your elegant program. ... Good job!

Thank you! I'm especially happy with the generality of this algorithm. If you append 4 steps/6 bytes:

```
26 STOP
27 1
28 X!=Y?
29 GTO 01
```

then the program will display each palindromic square of 12 digits or fewer (36 in all, counting the trivially palindromic squares of 3, 2, and 1, only one other of which has an even number of digits). But it will take days on a physical 42S.

> Quote:
>
> the execution speed could be improved by using some basic heuristics that restrict, within appropriate ranges of N, which last-digit values of N^2 to even check for

I'm enjoying studying the other submissions to this challenge to see slick ways to do exactly that.

-A

Re: HP42S Mini-Challenge: Optimizing !

*Message #12 Posted by **Allen** on 9 May 2007, 9:22 p.m.,*
in response to message #10 by Karl Schneider

> Quote:
>
> I don't see any particular way to make your algorithm much more concise

Karl, I am surprised that you made no comment RE the 32 byte solution or the X/11 heuristic I added below, the former being 25% smaller than the previous post. I really enjoy the 'write a small program' challenges. Also enjoy the heuristics you added. In both case 1 and 3 (but not Valentin's case #2-speed/size challenge) The creation of these algorithms is mathematical poetry! You and Alex are both great writers, among others here. Hats off to all.

Some comments: HP42S Mini-Challenge

*Message #13 Posted by **Karl Schneider** on 10 May 2007, 12:58 a.m.,*
in response to message #12 by Allen

> Quote:
>
> Karl, I am surprised that you made no comment RE the 32 byte solution or the X/11 heuristic I added below,...

Hi, Allen --

I did incorporate the X/11 heuristic into my own solution, which reduced the execution time by more than 60%. Paul Brogger had included it in his submission, prior to yours. I hadn't even known of the theorem until then.

Your 32-byte program seemed to have the same basis as Alex' program. His algorithm was already taut, but there's always a way to save a few bytes...

> Quote:
>
> I really enjoy the 'write a small program' challenges.

I don't, in particular. It seems to me that reduction of program size to its absolute minimum defeats the true purpose of a programmable calculator: To allow the user to quickly create functional programmed solutions to a problem without the need to remember syntactical rules, spellings, formalities, and compilation procedures.

"Paring down" of a program may have been necessary for implementing anything non-trivial on the earliest programmables, but not on a HP-42S with 7 kB of RAM.

-- KS

Edited: 10 May 2007, 1:18 a.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #14 Posted by **Werner** on 3 May 2007, 4:57 p.m.,*
in response to message #1 by Valentin Albillo

I took the following approach: I generate the 12-digit palindromes, generating the first (and last) two digits separately making use of the knowledge that the number is a square (thus dividing the work by four, roughly), and just blind force for the inner loop. It is not HP-42 specific.

timing: 1m13s @ 1.4Ghz size: 115 Bytes, regs R1-R5 used

```
00 { 115-Byte Prgm }
01*LBL "P12"
02 99
03 STO 05
04*LBL 05      generate xy00000000yx, with yx = R05^2 MOD 100
05 RCL 05
06 X^2
07 100
08 MOD
```

```
09 X=0?        can't start/end with zeroes
10 GTO 99
11 10
12 STO 04
13 %
14 ENTER
15 FP
16 100
17 *
18 +
19 IP
20 1e10
21 *
22 +
23*LBL 04
24 10
25 STO 03
26 RDN
27*LBL 03
28 10
29 STO 02
30 RDN
31*LBL 02
32 10
33 STO 01
34 RDN
35*LBL 01       inner loop
36 ENTER        test for square
37 SQRT
38 FP
39 X=0?
40 RTN
41 RDN
42 11e5         add 1 to innermost digits (6 and 7)
43 +
44 DSE 01
45 GTO 01
46 99e4         adjust for digits 5 and 8
47 -
48 DSE 02
49 GTO 02
50 99e3         adjust for digits 4 and 9
51 -
52 DSE 03
53 GTO 03
54 99e2
55 -
```

```
56 DSE 04
57 GTO 04
58*LBL 99
59 DSE 05
60 GTO 05
61 END
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #15 Posted by **Paul Brogger** on 3 May 2007, 7:24 p.m.,*
in response to message #1 by Valentin Albillo

I wrote mine originally on the HP-33s, then copied it to Free42 on Windows. As such, it doesn't make use of any particular HP-42s features, and so isn't optimized for that machine.

It uses two registers, R00 and R01.

```
00 { 66-Byte Prgm }
01>LBL 00
02 99999      Iterate through all 6-digit first-halves,
03 STO 00        starting at 100,000


04>LBL 01
05 1
06 STO+ 00    Increment high-order six-digit section
07 0
08 RCL 00     Put it in x
09 XEQ 02
10 XEQ 02           Reverse
11 XEQ 02
12 XEQ 02                that
13 XEQ 02
14 XEQ 02                    in y
15 Rv
16 RCL 00     and
17 1E6           splice
18 *                  them
19 +                      together
20 STO 01     (save 12-digit value, in case I need it later)
21 SQRT
22 FP
23 X!=0?      Is it the square of some integer?
24 GTO 01        no -- try next 6-digit value
```

```
25 RCL 01       yes! -- stop with 12-digit palindromic square in x
26 STOP


27>LBL 02
28 10         Take low-order digit
29 ÷                       in x,
30 FP
31 LASTX
32 IP
33 Rv
34 +           and append it
35 10               to y, shifting that left
36 *
37 R^
38 RTN
```

Pretty fast (~53 sec.) on Free42Decimal (on 2.8GHz Win2000) and very slow (still running) on HP-33s.

(To use only one register, don't save in R01, and replace line 25 with a LASTx, x^2.)

Edited: 3 May 2007, 10:44 p.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #16 Posted by **Paul Brogger** on 4 May 2007, 2:30 a.m.,*
in response to message #15 by Paul Brogger

This one takes about a third the time of my previous attempt. It has (if I do say so myself) a pretty slick test for palindromicity (?) in section labeled LBL 02.

```
00 { 58-Byte Prgm }
01>LBL 00
02 316227
03 STO 00


04>LBL 01     Increment integer
05 1
06 STO+ 00
07 RCL 00
08 X^2            and square it
09 1E6
10 <div>                  prior to palindrome test.
```

```
11>LBL 02     Assumes decimal point in center of
12 X=0?           even-length palindromic sequence in x.
13 GTO 03     If zero, we've found our result.
14 10
15 <mult>          Shift one digit left
16 FP
17 LASTX
18 IP
19 100        Then two digits right
20 <div>
21 IP
22 LASTX
23 FP
24 0.11       Divide middle two digits by .11
25 <div>
26 FP
27 X!=0?
28 GTO 01     Don't match?  Try square of next integer.
29 Rv
30 +          Concatenate remaining fragments
31 GTO 02       and test next middle two digits


32>LBL 03
33 RCL 00
34 X^2
35 RTN
```

It uses only register R00.

Edited: 9 May 2007, 5:33 p.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #17 Posted by **Gerson W. Barbosa** on 3 May 2007, 9:18 p.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin,

> Quote:
>
> 1. Optimized for *minimum size*, regardless of speed

It was not my intention to participate on this one, although it's very interesting as always. I would try goal one at most. Looks like I was able to optimize for *minimum* speed only :-)

About 5 minutes (Free42 @ 500 MHz), which means about 46 hours on the real 42S.

```
00 { 64-Byte Prgm }
01>LBL "SP"
02 1E6
03 STO 09
04>LBL 01
05 1
06 STO- 09
07 6
08 STO 00
09 RCL 09
10 0
11 STO 08
12>LBL 00
13 X<>Y
14 IP
15 10
16 ÷
17 ENTER
18 FP
19 RCL 00
20 10^X
21 ×
22 STO+ 08
23 DSE 00
24 GTO 00
25 RCL 09
26 1E6
27 ×
28 RCL 08
29 +
30 SQRT
31 FP
32 X!=0?
33 GTO 01
34 1E6
35 RCL× 09
36 RCL+ 08
37 .END.
```

Best regards,

Gerson.

Re: HP42S Mini-Challenge: Optimizing ! (final submission)

*Message #18 Posted by **Egan Ford** on 5 May 2007, 1:41 p.m.,*
in response to message #1 by Valentin Albillo

> Quote:
>
> Enjoy ! :-)

I did. This is my final submission for part 3 (speed) of this mini challenge. One of my objectives from the start was to NOT be 42S specific (personal challenge). As for, "*Using reasonable, sound heuristics which can be rationalized in a few words is Ok*". I'll let you be the judge.

```
Answer: 798,644 * 798,644 = 637,832,238,736
```

Runtimes:

```
Free42, 1.7 GHz Pentium M:    <1 sec
Emu42,  1.7 GHz Pentium M:    23 sec
Physical 42S:                 1h, 2min
```

Code suitable for import into Free42 or Emu42: http://sense.net/~egan/pp9.txt.raw.

I broke this problem down into two separate problems: iteration reduction and verification optimization.

Iteration reduction:

My initial brute force approach started at 999999 and counted down until a solution was found. I selected count down vs. count up because I was planning on using DSE (easier than ISE). This approach will make 201,355 attempts to find a solution. Heuristical methods will need to be used to reduce this.

Knowing that perfect squares must end in 0, 1, 4, 5, 6, or 9, I opted to count down from the largest 12 digit square (999999^2) and ignore any that didn't start with 9, 6, 5, 4, or 1. This will cut the number of iterations roughly in half. 89,333 iterations to be exact. Since N*N = the sum of the first N odd numbers, counting down is easy. Just find the last odd: 999999*2 - 999998*2. Then subtract 2 from the odd in a loop to get the difference to the next perfect square.

Of the perfect squares only a fraction will end in with a digit that matches the first. Is there a way to determine the next first/last matching digit perfect square? Yes, there is. Just sum 10 sequential odd numbers to get a result that ends in 0. Simply find the first perfect square with matching first/last digit by subtracting the odds, then to get 10 odds down, take the next odd, multiple by 10, subtract 90. This approach does have one gotcha, there are 2 series of first/last matching squares, and both have to be searched, e.g.:

```
999999^2 =  999998000001  <= Start at top. End in 9? No, then subtract odd difference.
999998^2 =  999996000004  <= End in 9? No, then subtract odd difference.
999997^2 =  999994000009  <= End in 9? Yes, 10*odd diff - 90 -------.
999996^2 =  999992000016                                           |
999995^2 =  999990000025                                           |
999994^2 =  999988000036                                           |
999993^2 =  999986000049  <= Opps, missed this one. 10 down-----.  |
999992^2 =  999984000064                                       |   |
999991^2 =  999982000081                                       |   |
999990^2 =  999980000100                                       |   |
999989^2 =  999978000121                                       |   |
999988^2 =  999976000144                                       |   |
999987^2 =  999974000169  <------------------------------------|---'
999986^2 =  999972000196                                       |
999985^2 =  999970000225                                       |
999984^2 =  999968000256                                       |
999983^2 =  999966000289  <------------------------------------'
999982^2 =  999964000324
```

This happens because there is no guarantee that a downward count of the next odd will always end at 9. If ending at 7, 5, 3, or 1, then the next matching first/last digit will be lest than 10 odds away.

```
(9 + 7 + 5 + 3 + 1 + 9 + 7 + 5 + 3 + 1) mod 10 = 0
(7 + 5 + 3 + 1 + 9 + 7 + 5 + 3) mod 10 = 0
(5 + 3 + 1 + 9 + 7 + 5) mod 10 = 0
(3 + 1 + 9 + 7) mod 10 = 0
(1 + 9) mod 10 = 0
```

To get around this problem without too much work I just did two passes. Pass 1 starts with the highest first/last matching square, pass 2 starts with the 2nd largest. Each pass jumps 10 squares at a time. This reduces the number of iterations by ~1/5. 20,271 iterations to be exact.

Lastly I started looking at the 2nd digits. Without much thought you can prove that the 2nd to last digit of all perfect squares is even, unless the last digit is 6.

E.g. take squares ending in 9. Only 3*3 or 7*7 end in 9, so...

```
      x + 3              x  + 7
      x + 3              x  + 7
------------    -----------------
     3x + 9           (7x+4) + 9
x*2 + 3x        x*2 +  (7x+0)
------------    -----------------
x*2 + 6x + 9    x*2 + (14x+4) + 9
```

2nd digit will be 6x, and even*anything is even. This is true for 14x+4 as well (note the even carry digit). You can do the rest of this exercise in your head (i.e. avoid *doing all the theoretical work yourself with paper and pen*), just look at the carry digit, e.g. to end in 9 you must have 3*3 or 7*7, the carry digit is even (i.e. 0 or 4), to end in 6 you must have 4*4 or 6*6, the carry digit is 1 or 3 ending with a even + odd for the 2nd digit. IANS, implementing this will reduce the number of iterations by ~1/2. 8,889 iterations to be exact.

3rd digit? Too much brain power was needed, further optimization will need to be done in the verification.

Verification optimization:

The first check is the most important check. Taking a hint from Paul Brogger I cut out the center, divided by 11 and checked for FP = 0:

```
1E-5
×
IP
100
MOD
11
÷
FP
X!=0?
```

Other attempts that set up an efficient way to check the rest of the digits is a waste. The first digit check will happen at each iteration. The probability of a match is only 1:10 making the 2nd check take place ~1/10 of the iterations. Make the first check fast, make the rest source efficient.

More optimization can be done, but I made my ~1 hour time target.

Finally, I must state how awesome the 42S is. I did all my development using the VI editor and then using txt2raw.pl to convert to raw for import into Free42/Emu42. I was not looking forward to keying this into my 42S for a formal benchmark. After doing so, I must state that there is an attention to detail that I do not think I have seen with any other vertically oriented HP (48GX is close). The orientation of the menus and keys made for very accurate and quick work of entering this code. I applaud the 42S architects.

Source with comments:

```
00 { 279-Byte Prgm }


Main loop.  DSE from 9 to 1.  If 8,7,3,2 skip
since perfect squares only end in 9,6,5,4,1,0.


If '6' then note the 2nd digits will be odd (start with 9
```

```
count down by 2).  Evens start with 8 and count down by 2.
This is stored in register 08.


01>LBL "PP9"
02 9
03 STO 09
04>LBL 09
05 8
06 STO 08
07 RCL 09
08 X=Y?
09 GTO 10
10 7
11 RCL 09
12 X=Y?
13 GTO 10
14 3
15 RCL 09
16 X=Y?
17 GTO 10
18 2
19 RCL 09
20 X=Y?
21 GTO 10
22 6
23 RCL 09
24 X!=Y?
25 GTO 11
26 1
27 STO+ 08
28>LBL 11
29 XEQ A
30>LBL 10
31 DSE 09
32 GTO 09
33 STOP


This is the first called subroutine.  It DSE counts from
5 to 1 creating a sequence for LBL B with the first 2
digits to start searching form, (e.g. 98, 96, 94, 92, 90,
69, 67, ..., 10).  This will also store the mirror image
of the first to digits (REG 13).  REG 04 (0 or 1) is used to
select the 2 different series of matching first/last
squares.
```

```
34>LBL A
35 5
36 STO 12
37>LBL 12
38 10
39 RCL× 09
40 RCL+ 08
41 STO 03
42 10
43 ÷
44 IP
45 RCL 03
46 10
47 MOD
48 10
49 ×
50 +
51 STO 13
52 0
53 STO 04
54 XEQ B
55 1
56 STO 04
57 XEQ B
58 2
59 STO- 08
60 DSE 12
61 GTO 12
62 RTN


This is where all the work is done.  Starting from
the largest two digit leading 12 digit number (e.g.
989999999999) find the first or 2nd (check REG 04)
square with matching reversed last 2 digits.


63>LBL B
64 RCL 03
65 1
66 +
67 10
68 10^X
69 ×
70 1
71 -
72 SQRT
```

```
73 IP
74 X^2
75 STO 05
76 LASTX
77 1
78 -
79 X^2
80 -
81 STO 06
82>LBL 00
83 RCL 05
84 100
85 MOD
86 RCL 13
87 X=Y?
88 GTO 01
89 RCL 06
90 STO- 05
91 2
92 STO- 06
93 GTO 00
94>LBL 01
95 RCL 04
96 X=0?
97 GTO 02
98 0
99 STO 04
100 RCL 06
101 STO- 05
102 2
103 STO- 06
104 GTO 00
105>LBL 02
106 10
107 10^X
108 RCL× 03
109 STO 07
110 GTO 04


This is where ~90% of all the computation takes place.
The first iteration starts at line 119.  Center digits
are checked, if no match GTO 03, 10*next odd - 90 to get
next square, subtract 20 from next odd to get next odd.
If the two leading digits change, exit loop.


111>LBL 03
```

```
112 10
113 RCL× 06
114 90
115 -
116 STO- 05
117 20
118 STO- 06
119>LBL 04
120 RCL 07
121 RCL 05
122 X<Y?
123 RTN
124 1E-5
125 ×
126 IP
127 100
128 MOD
129 11
130 ÷
131 FP
132 X!=0?
133 GTO 03


Check last 4 pairs.  If the centers match, start
from the outside and work in.  Since squares are
deterministically calculated with matching first/last
digits there is no need to check the outside.
At most there are 4 checks.


134 4
135 STO 15
136>LBL 15
137 -2
138 RCL× 15
139 1
140 -
141 10^X
142 RCL 15
143 5
144 -
145 10^X
146 RCL× 05
147 IP
148 2
149 RCL× 15
150 2
```

```
151 +
152 10^X
153 MOD
154 ×
155 LASTX
156 10
157 MOD
158 X<>Y
159 IP
160 X!=Y?
161 GTO 03
162 DSE 15
163 GTO 15
164 RCL 05
165 ENTER
166 SQRT
167 STOP
168 .END.
```

Edited: 5 May 2007, 2:11 p.m.

Re: HP42S Mini-Challenge: Optimizing !

*Message #19 Posted by **Karl Schneider** on 6 May 2007, 1:57 a.m.,*
in response to message #1 by Valentin Albillo

(NOTE: This post was edited to implement the heuristic regarding divisibility by 11, which was identified by others. Only two instructions in my program were changed, reducing execution time by more than 60%.)

To get a working and fast program for this challenge, I finally installed Thomas Okken's magnificent Free42 software that I had downloaded quite a while ago. It was quite easy to do; I ought to have done it a long time ago.

My intended approach was similar to the one elegantly implemented by "Adam L", which was to convert N^2 to a string, then compare the outermost "digit-characters", working toward the middle from each end each time a match was found, and immediately jumping to the next N upon getting a mismatch. Not immediately seeing how to do that with HP-42S string functions, I implemented the same logic with mathematical calculations based on INT and MOD. This is slower and more cumbersome on a physical HP-42S, but it worked.

My "ideal" optimized program would incorporate heuristics 3 and 4 into Adam L's tidy code.

The heuristics:

1. It is faster and simpler to generate a square and iteratively check for its "palindomicity" than to generate palindromes and check for integer square roots. There are 900,000 possible 12-digit palindromes, which can also be cumbersome to generate.

2. The lowest integer N that produces a 12-digit square is 316,228; the largest is 999,999. The overall search should be restricted to this range of 683,772 possiblities.

3. A squared integer not divisible by 10 can end only with 1, 4, 5, 6, or 9. Since the last digit of N^2 must match the first, there is no need to try values of N for which N^2 is within the inclusive ranges {200,000 to 399,999} or {700,000 to 899,999}. Therefore, N = 447,214 through 632,457 and N = 836,661 through 948,683 can be bypassed *en masse*.

4. A palindrome having an even number of digits will be evenly divisible by 11. Therefore, all values of N that are *not* as such can be skipped over, eliminating 91% of all remaining possibilities.

5. N ending in 0 will yield an N^2 ending in 00. If the first two digits were to match, N^2 would be only a 10-digit number. Therefore, each N that is a multiple of 10 can be skipped over. In view of the preceding heuristic, however, this may not yield a significant reduction in processing time for the programming effort.

6. For a given value of the leading digit of N^2, only one or two ending digits of N will match it. However, checks for this heuristic are a bit cumbersome to implement, and may not save much execution time.

So, here's my program, which finds the unique solution *working upward from N = 316,228*. It runs in only 1.8 seconds on Free42 Binary and 7.2 seconds on Free42 Decimal, on a PC with twin Pentium IV 3.0-GHz CPU's.

The program could be easily modified to search from N = 999,999 downward. This would find the answer even faster, as it turns out.

Lines 52 and 53 (FIX 00 and RND) were necessary to eliminate floating-point arithmetic errors. Line 57 (ALL display format) is for debugging purposes.

```
00 { 148-Byte Prgm }
01>LBL "SQ12P"
02 316227
03 STO 00
04>LBL 00
05 1
06 STO+ 00
07 RCL 00
08 11
09 MOD
10 X!=0?
11 GTO 00
12 RCL 00
```

```
13 447214
14 X=Y?
15 GTO 06
16 CLX
17 836661
18 X=Y?
19 GTO 07
20 CLX
21 999999
22 X<>Y
23 X>Y?
24 STOP
25 X^2
26 1E11
27 STO 03
28 ÷
29 STO 02
30 XEQ 01
31>LBL 06
32 632455
33 STO 00
34 GTO 00
35>LBL 07
36 948683
37 STO 00
38 GTO 00
39>LBL 05
40 RCL 00
41 ENTER
42 X^2
43 STOP
44>LBL 01
45 RCL 02
46 ENTER
47 IP
48 STO 04
49 -
50 RCL 03
51 ×
52 FIX 00
53 RND
54 ENTER
55 ENTER
56 10
57 ALL
58 MOD
59 STO 05
```

```
60 -
61 STO 02
62 RCL 04
63 RCL 05
64 X!=Y?
65 GTO 00
66 RCL 02
67 X=0?
68 GTO 05
69 10
70 STO÷ 03
71 RCL 03
72 STO÷ 02
73 10
74 STO÷ 03
75 GTO 01
```

Edited: 13 May 2007, 4:20 p.m.

32 byte, (20 steps in 55 sec.) solution for free42 Mini-Challenge

*Message #20 Posted by **allen** on 6 May 2007, 9:04 a.m.,*
in response to message #1 by Valentin Albillo

This was also my first occasion to write a 42s program, though the title of the challenge could be best titled "free42 challenge" since the minimum size contest is likely not practical on a REAL 42S. Accordingly, like Karl, I had to dust off the free42 installation from last year and learn how to use it. What a GREAT program!!!

I propose a 32 byte solution using the STAT register as the counter. (mainly because the \sigma+ command automatically puts the increment result on the stack eliminating the need for a 1 STO+ XX RCL XX to bring the value back to the stack.) Runtime is <1min on free42.

```
00 { 32-Byte Prgm }
01 CL\Sigma            ' Clear stat register
02>LBL 00              ‘ Begin counting loop
03 \Sigma+             ‘ Increment STAT register
04 X^2
05 CLA                 ‘ Clear Alpha register
06 AIP                 ' Pull result to Alpha reg
07 6
08 STO 01              ' Store counter for length loop
09>LBL 02              ' Begin palindromic check / length loop
10 -1
11 AROT                ' Rotate last char to front of ALPHA reg
```

```
12 ATOX                ' Pull last character code to X
13 ATOX                ' Pull first character code to X
14 X!=Y?               ' Are the first and last characters equal?
15 GTO 00              ' if not save time by ending loop early and go to couterloop
16 DSE 01              ' if they are, decrement and check for length
17 GTO 02              ' if too short, go to the length loop
18 X=0?                ' otherwise was the last result from an empty ALPHA reg?
19 GTO 00              ‘ if yes, return to counter
20 RCL 16              ‘ otherwise SUCCESS!!! – recall the stat N value
```

Edited: 6 May 2007, 6:33 p.m. after one or more responses were posted

Re: 41 byte, 7 second solution for free42 Mini-Challenge

*Message #21 Posted by **allen** on 6 May 2007, 6:13 p.m.,*
in response to message #20 by allen

Using the heuristic that palindromes are divisible by ll, one can add 4 steps to the program and finish 90% faster:

```
00 {41-Byte Prgm }
01 CL\Sigma            ' Clear stat register
02>LBL 00              ‘ Begin counting loop
03 \Sigma+             ‘ Increment STAT register
04 X^2
05 121                 '  ADDED TO CHECK ONLY MULTIPLES OF 11
06 *                   '
07 CLA                 ‘ Clear Alpha register
08 AIP                 ' Pull result to Alpha reg
09 6
10 STO 01              ' Store counter for length loop
11>LBL 02              ' Begin palindromic check / length loop
12 -1
13 AROT                ' Rotate last char to front of ALPHA reg
14 ATOX                ' Pull last character code to X
15 ATOX                ' Pull first character code to X
16 X!=Y?               ' Are the first and last characters equal?
17 GTO 00              ' if not save time by ending loop early and go to couterloop
18 DSE 01              ' if the are, decrement and check for length
19 GTO 02              ' if too short, go to the length loop
20 X=0?                ' otherwise was the last result from an empty ALPHA reg?
21 GTO 00              ‘ if yes, return to counter
22 RCL 16              ‘ otherwise SUCCESS!!! recall stat N value
23 11
24 *                ' ADDED TO CONVERT couter to X^2 format where X is couter*11
```

This differs slightly from my initial 44 byte program where I used an (ATOX POSA x=0?) loop similar to Alex's above (ATOX ATOX X=Y?). I was hoping to use the ATOX and POSA commands to create an AND gate so I would not have to process both the length loop and the empty ALPHA loop.

Re: 41 byte, 7 second solution for free42 Mini-Challenge

*Message #22 Posted by **Alex L** on 8 May 2007, 12:34 p.m.,*
in response to message #21 by allen

Add me to the "why didn't I think of that" category with the 11 heuristic.

That can be incorporated into my submission at a cost of 4 bytes and 0 steps (plus manually checking 999999), by changing step 01 to 999999, and step 04 to 11. This finishes in under 2 sec on Free42, which leads me to believe it should be just over an hour on a physical 42S, which puts it into the range of respectability. Perhaps I'll check. :)

Of course it's no longer necessarily going to work for the general case.

-- complete updated program --

```
01  999999
02  STO 01
03  LBL 01
04  11
05  STO- 01
06  RCL 01
07  X^2
08  CLA
09  AIP
10  LBL 02
11  ALENG
12  2
13  X>Y?
14  GTO 03
15  -1
16  AROT
17  ATOX
18  ATOX
19  X=Y?
20  GTO 02
21  GTO 01
22  LBL 03
23  RCL 01
24  ENTER
25  X^2
```

Re: 41 byte, 7 second solution for free42 Mini-Challenge

*Message #23 Posted by **Gerson W. Barbosa** on 8 May 2007, 9:31 p.m.,*
in response to message #22 by Alex L

Hello Alex,

Congratulations for your tiny versions. The MOD(11) heuristics others have found is a killer one indeed. I added it to the simple 71B program I had written yesterday (too lazy to write a 42S or 33S version; besides, just a variation of what has already been written).

I've just run your latest version on my HP-42S: 1 hour 3 minutes.

Here is the HP-71 program:

```
10 DESTROY ALL @ T=TIME
20 FOR N=999999 TO 316228 STEP -11
30 X$=STR$(N*N) @ I=0 @ S=0
40 I=I+1
50 IF X$[I,I]#X$[13-I,13-I] THEN 80
60 S=S+1 @ IF I<6 THEN 40
70 IF S=6 THEN DISP X$;TIME-T @ END
80 NEXT N
>run
637832238736 15.02
```

15 seconds @ 500 MHz, about 26 minutes on the real 71B. 144 bytes.

By the way, the latest version of J-F Garnier's Emu71 (2006) is a pleasure to use: copy and paste to and from the DOS window works nicely. I have to remember to upgrade to the registered version, even though I still don't have the serial interface :-)

Regards,

Gerson.

Re: HP42S Mini-Challenge: Optimizing !

*Message #24 Posted by **Werner** on 7 May 2007, 7:49 a.m.,*
in response to message #1 by Valentin Albillo

Second try..

```
heuristics:
- palindromes with an even nr of digits are divisible by 11, so x has to be divisible by 11 as well.
- if x ends with digit i, y starts/ends with digit i^2 MOD 10 and
x is between SQRT(i*10^11) and SQRT((i+1)*10^11)


We'll loop over the 'ending digits' and determine the range to check.
Within that range, only multiples of 11 ending in the same digit have to be checked.


timing: Emu42 @ 3Ghz: 9s


It should be noted that since the result is obtained with ending digit 4, counting upwards would've been faster still.


{ 115-Byte Prgm }
*LBL "PSX"
 9
 STO 01
*LBL 01         loop over i=9..1 step -1
 RCL 01
 X^2
 10
 MOD
 1e11
 STO* ST Y
 X<>Y
 STO+ ST Y
 XEQ 90         determine x boundaries
 STO 02
 RDN
 XEQ 90
 STO 03
*LBL 02         inner loop
 X^2
 1e6
 /
 ENTER
*LBL 03         palindrome test
 RDN
 X=0?
 RTN            R03 contains x
 10             xy,yx
 /              x,yyx
 IP
 LASTX
 FP             0,yyx   x
```

```
 100
 *
 +
 LASTX
 IP
 STO- ST Y      yy      x,x
 11
 MOD
 X=0?
 GTO 03
 RCL 02
 RCL 03
 110            steps of 110 ensure that x remains divisble by 11 and
 -              retains the same ending digit
 STO 03
 X#Y?
 GTO 02
 DSE 01
 GTO 01
 RTN
*LBL 90         determine x so that [11*(i + 10*x)]^2 < X
 SQRT
 11
 /
 RCL 01
 -
 10
 /
 IP
 10
 *
 RCL 01
 +
 11
 *
 END
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #25 Posted by **Werner** on 7 May 2007, 3:36 p.m.,*
in response to message #24 by Werner

a few corrections: the intro should read:

- palindromes with an even nr of digits are divisible by 11, so x has to be divisible by 11 as well.

```
- if x ends with digit i, y starts/ends with digit j = i^2 MOD 10 and
  x is between SQRT(j*10^11) and SQRT((j+1)*10^11)
  Since x also has to be a multiple of 11, we have:
  SQRT(j*10^11) < x             < SQRT((j+1)*10^11)
                < 11*(10*z + i) <
We'll loop over the 'ending digits' i=9..1 and determine the range to check.
Within that range, only multiples of 11 ending in the same digit have to be checked.


And the length of the program is 112 Bytes, not 115
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #26 Posted by **Werner** on 8 May 2007, 2:17 a.m.,*
in response to message #25 by Werner

```
Third, and final, try.
p = x^2
Basically the same program as in #2, but counting the last two digits of x, and determining the corresponding range of p.
Runs in a true HP42S in under 3 minutes, uses R01-R03


{ 137-Byte Prgm }
*LBL "PS3"
 99
 STO 01
*LBL 01         loop over ij=99..1 step -1
 RCL 01
 X^2
 100
 MOD            yx = ending digits of palindrome
 10
 /
 IP
 LASTX
 FP
 X=0?
 GTO 00
 100
 *
 +              xy = starting digits
 1e10
 STO* ST Y
 X<>Y
 STO+ ST Y
 XEQ 90         determine x boundaries
 STO 02
```

```
 RDN
 XEQ 90
 STO 03
*LBL 02         inner loop
 X^2
 1e6
 /
 ENTER
*LBL 03         palindrome test
 RDN
 X=0?
 RTN            R03 contains x
 10             xy,yx
 /              x,yyx
 IP
 LASTX
 FP             0,yyx   x
 100
 *
 +
 LASTX
 IP
 STO- ST Y      yy      x,x
 11
 MOD
 X=0?
 GTO 03
 RCL 02
 RCL 03
 1100           steps of 1100 ensure that x remains divisible by 11 and
 -              retains the same ending digits
 STO 03
 X#Y?
 GTO 02
*LBL 00
 DSE 01
 GTO 01
 RTN
*LBL 90         determine x so that [11*(ij - j*10 + 100*x)]^2 < X
 SQRT           endures x ends in ij and is divisible by 11
 1100
 /
 IP
 100
 *
 RCL 01
 10
```

```
MOD
LASTX
*
RCL 01
-
-
11
*
END
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #27 Posted by **Egan Ford** on 7 May 2007, 3:50 p.m.,*
in response to message #24 by Werner

> Quote:
>
> - palindromes with an even nr of digits are divisible by 11...

Ugggg! How did I miss this? It was trival to prove too.

I applied the following patch to my last submission and get 9s with Emu42 @ 1.7 GHz Pentium M.

Thanks.

```
1c1
< LBL "PP9"
---
> LBL "PP11"
87a88,94
> GTO 21
> GTO 22
> LBL 21
> RCL 05
> 11
> MOD
> X=0?
88a96
> LBL 22
112c120
< 10
---
> 110
114c122
```

```
< 90
---
> 11990
117c125
< 20
---
> 220
```

Re: HP42S Mini-Challenge: Optimizing !

*Message #28 Posted by **Howard Owen** on 7 May 2007, 6:16 p.m.,*
in response to message #27 by Egan Ford

Great, Egan!

But where am I going to get a *patch(1)* binary for my 42S?

8)

Regards,
Howard

Re: HP42S Mini-Challenge: Optimizing !

*Message #29 Posted by **Egan Ford** on 7 May 2007, 6:39 p.m.,*
in response to message #28 by Howard Owen

> Quote:
>
> But where am I going to get a *patch(1)* binary for my 42S?

Sorry, source http://sense.net/~egan/pp11.txt and binary http://sense.net/~egan/pp11.txt.raw .

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #30 Posted by **Valentin Albillo** on 9 May 2007, 7:21 p.m.,*
in response to message #1 by Valentin Albillo

Hi all,

First of all, thank you very much for the overwhelming response to this 42S mini-challenge. Awesome solutions have been produced and though I should pretty much be used to it by now, I'm continually amazed at the ingenuity displayed by so many contributors, both frequent posters and newcomers alike. I'm sure we all have learned a lot from these excellent solutions and added a lot of fine techniques and brilliant ideas to our arsenal of programming knowledge, not only HP42S-related but general in nature.

For the record, I'll give and briefly discuss my original solutions. I initially wrote the (b) case solution, then suitably modified it to also cover cases (a) and (c).

My rationale was as follows: the task consisted of finding a 12-digit palindromic square as easily and fast as possible. Two approaches came to mind immediately:

- Generate all squares in the required range (316228^2 to 999999^2, in steps of 11 (because a palindromic even-digit number is necessarily a multiple of 11, thus its square root must be as well as 11 is a prime number), and test them for palindromicity. This amounted to some 62000 candidates, thus some 31000 on average.

- Generate all 12-digit palindromes from 100000000001 to 980000000089 subject to some simple heuristics, and test them for squaredness.

Both approaches seemed feasible but I decided that it would probably be a lot easier and faster to test for squaredness (which can be done in just 3 fast steps) than testing for palindromicity, which is more involved and slow, so the second option seemed easier as long as I could devise a fast, simple procedure for directly generating 12-digit palindromic numbers and further, to only generate those which had a chance of being perfect squares, thus reducing the number of candidates to a reasonable minimum.

This I could do, so I opted for the second approach. My program for the case (b) below directly generates 12-digit palindromes in a very fast way, using pre-stored constants which are quickly initialized once at the beginning of the program, and taking all loop invariants out of the innermost loops, so that redundant arithmetic operations are avoided.

The following extremely simple heuristics are used:

- We only need to loop from 100000 to 999999, the first six digits of the palindrome, as the remaining 6 are simply the mirror image of these, thus a maximum of some 900000 loops would be needed

- Further, a perfect square can only end in the following digits:

  01,04,09,16,21,24,25,29,36,41,44,49,56,61,64,69,76,81,84,89,96

  the termination 00 being discarded as otherwise the palindromic number would begin by 00 and thus be just a 10-digit number. These ending sequences must of course also appear (reversed) at the beginning of any candidate, thus limiting the maximum number of cases to try to some 210000

in the worst case, an average of about only 105000 statistically probable. This is about 3 times as many candidates as the first option, but requiring a much simpler and faster test per candidate so it seemed plausible to go this way.

Assuming I could generate the palindromic candidates very fast subject to these constraints, and as the squaredness test was just 3 fast steps, I considered that 105,000 cases (average) to try was perfectly feasible even on a physical HP42S. This is the resulting 72-step (157-byte) program:

```
01 LBL "P"      19 x          37 LBL 02     55 GTO 04
02 (see *1)     20 LASTX      38 RCL 04     56 Rdown
03 (see *1)     21 RCLx 04    39 STO 02     57 RCL+ 08
04 1000000100   22 LASTX      40 RCL 10     58 DSE 02
05 STO 06       23 RCL/ 04    41 LBL 03     59 GTO 03
06 100001E3     24 IP         42 RCL 04     60 RCL 07
07 STO 07       25 RCLx 05    43 STO 03     61 STO+ 10
08 1001E4       26 -          44 Rdown      62 DSE 01
09 STO 08       27 +          45 ENTER      63 GTO 02
10 11E5         28 VIEW ST X  46 LBL 04     64 RCL 06
11 STO 09       29 STO 11     47 ENTER      65 STO+ 11
12 10           30 RCL 04     48 SQRT       66 DSE 00
13 STO 04       31 STO 00     49 FP         67 GTO 01
14 99           32 LBL 01     50 X=0?       68 GTO 00
15 STO 05       33 RCL 04     51 GTO 05     69 LBL 05
16 LBL 00       34 STO 01     52 Rdown      70 LASTX
17 1E10         35 RCL 11     53 RCL+ 09    71 X^2
18 ATOX         36 STO 10     54 DSE 03     72 END
```

*1:

- line 2 is alpha text formed by the following ASCII characters:

  10,40,90,61,12,42,52,92,63,14,44,94,65,16,46

- line 3 is *appended* alpha text formed by the following ASCII characters:

  96,67,18,48,98,69

All of them are keyable from the ALPHA entry menus. This is an image of how they should look like as program lines:

As you can see, an added *finesse* is to have all 21 possible beginning (i.e., reversed ending) sequences stored as ALPHA characters in the ALPHA register, from which they are easily taken out, one at a time, with a simple ATOX instruction. This is fast and saves many program steps/bytes and/or registers.

The resulting program find the only solution,

637,832,238,736 = $798{,}644^2$

in just 2 h 42 min in a physical HP42S, 85 seconds in Emu42 @ 2.4 Ghz, and 44 seconds in Free42 @ a very modest Palm Z100.

Once I had this solution, I created the one for case (c) by simply unrolling the innermost loop, like this:

```
  ...
41 LBL 03
42 ENTER
43 ENTER      50 ENTER            113 Rdown    (line 56 of (b))
44 SQRT       51 SQRT             114 RCL+ 08  (line 57 of (b))
45 FP         52 FP          ...      ...      (etc)
46 X=0?       53 X=0?        ...   129 END
47 GTO 05     54 GTO 05      ...
48 Rdown      55 Rdown
49 RCL+ 09    56 RCL+ 09
```

which produces a 129-step (239-byte) program which, as compensation for these extra steps, runs much faster, finding the solution in 1 h 48 min in a physical HP42S, 56 seconds in Emu42 @ 2.4 Ghz, and 29 seconds in Free42 @ Palm Z100. Thus it's quite clear that when speed does matter a lot, and in certain architectures, unrolling the innermost loop or loops actually pays handsomely.

As for the case (a) solution, it was just the one for case (b) with all the heuristics and finesses removed, which made it much shorter but unbearably slow. We've seen much better case (a) solutions posted here so it doesn't bear posting it.

By the way, I'm including here my case (a) solution for the HP-71B as a bonus, which is the following 2-line (58-byte) program (STD display mode is assumed):

```
1 FOR I=999999 TO 316228 STEP -11 @ S$=STR$(I*I) @ IF S$=REV$(S$) THEN DISP I,S$ @ END
2 NEXT I
```

It does find the unique solution in just 3 seconds under Emu71 @ 2.4 Ghz, about 12 min. in a physical HP-71B. Apart form reversing the loop, I don't think it can be made much simpler :-)

Again, thanks a lots for your valuable and superb inputs and get ready for the next one, it will be tougher and that's a promise.

Best regards from V.

Edited: 9 May 2007, 7:32 p.m.

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #31 Posted by **Gerson W. Barbosa** on 9 May 2007, 7:56 p.m.,*
in response to message #30 by Valentin Albillo

Hi Valentin,

> Quote:
>
> By the way, I'm including here my case (a) solution for the HP-71B as a bonus, which is the following 2-line (58-byte) program:
>
> ```
> 1 FOR I=999999 TO 316228 STEP -11 @ S$=STR$(I*I) @ IF S$=REV$(S$) THEN DISP I,S$ @ END
> 2 NEXT I
> ```

I have printed out both the HP-71 and the Math Pac Owner's Manual. I should have printed out the one with REV$ keyword :-)

Thanks for another MO&C, especially the extra bonus :-)

Best regards,

Gerson.

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #32 Posted by **Valentin Albillo** on 10 May 2007, 9:00 a.m.,*
in response to message #31 by Gerson W. Barbosa

Hi, Gerson:

Gerson posted:

"I have printed out both the HP-71 and the Math Pac Owner's Manual. I should have printed out the one with REV$ keyword :-)"

Thanks for your kind words of appreciation and for your posted contributions to this mini-challenge.

I'm sure you are already aware but just in case, the REV$ keyword is not available in a bare-bones HP-71B. It resides in a number of LEX (Language EXtension) files, such as STRUTIL, for instance, which can be copied to RAM or used from some ROM or EPROM. It's also simple enough that it can be very easily POKEd in as well with a fairly small program or even manually.

For people using Emu71, the basic freeware install does include the STRUTIL LEX file already resident at :PORT(5), as you can check by using CAT, so REV$ is already immediately available.

As I use Emu71 to develop my programs and create my challenges, mini-challenges, and articles, I usually include in them the functionality afforder by Emu71 right from installation, which means:

- all Math ROM keywords (e.g.: MAT A=INV(B))

- all HP-IL keywords (e.g.: BIT, etc)

- all STRUTIL keywords (e.g.: REV$, RPT$, etc)

Also, the physical HP-71B is typically available with an HP-IL ROM already plugged in, though getting a physical Math ROM is certainly much more difficult, but its incredible usefulness more that justifies any effort to get one and, IMHO, you don't have a full HP-71B unless you have one plugged in.

Best regards from V.

71B...

*Message #33 Posted by **Gene** on 10 May 2007, 3:20 p.m.,*
in response to message #32 by Valentin Albillo

I'm fortunate enough to have a physical 71B with the math rom and extra ram installed *internally*.

Leaves front ports open for the 41 translator, JPC X, etc. :-)

Re: 71B...

*Message #34 Posted by **Valentin Albillo** on 10 May 2007, 5:41 p.m.,*
in response to message #33 by Gene

Yes, very interesting, good for you ...

What about the determinants ? :-) :-)

Best regards from V.

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #35 Posted by **Gerson W. Barbosa** on 10 May 2007, 9:17 p.m.,*
in response to message #32 by Valentin Albillo

Hi Valentin,

> Quote:
>
> I'm sure you are already aware but just in case, the REV$ keyword is not available in a bare-bones HP-71B.

Yes, I knew REV$ was not a standard keyword when I noticed it wouldn't run on my physical HP-71B. I thought it was because mine is 1BBBB version, so your explanation has been much appreciated.

I was not lucky enough to have owned one when it was released. It was simply out of my reach! But I am lucky enough to have one now with the Math ROM as per your advice. I have also a card reader, though I believe an RS-232 interface would be more useful. I'm about to add a 32KB RAM module, which shoud be enough.

Since I am not familiar with the HP-71B yet, my programs tend to be longer than they ought to be. For instance, to get the following output formatted the way I wanted

```
R(4)   50º 07' 06" NW
```

I had to use almost two full lines:

```
280 M$=CHR$(48*(2-LEN(STR$(M))))&STR$(M) @ S$=CHR$(48*(2-LEN(STR$(S))))&STR$(S)
285 DISP "R(";STR$(I);")";TAB(T);D$;CHR$(167);" ";M$;"' ";S$;CHR$(34);" ";X$
```

Perhaps this could have been done with format strings, but I haven't figured a way to use them in this case. Reading the manuals might help :-)

Best regards,

Gerson.

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #36 Posted by **Valentin Albillo** on 11 May 2007, 7:27 a.m.,*
in response to message #35 by Gerson W. Barbosa

Hi, Gerson:

Gerson posted:

"Perhaps this could have been done with format strings, but I haven't figured a way to use them in this case."

Try this:

```
  1 I=4 @ M=7 @ S=6 @ T=10 @ X$="NW" @ D$="50"
  2 !
280 M$=CHR$(48*(2-LEN(STR$(M))))&STR$(M) @ S$=CHR$(48*(2-LEN(STR$(S))))&STR$(S)
285 DISP "R(";STR$(I);")";TAB(T);D$;CHR$(167);" ";M$;"' ";S$;CHR$(34);" ";X$
290 !
300 DISP USING "'R(',D,')',5X,2A,B,2(X,2Z,B),X,2A";I,D$,167,M,39,S,34,X$


 >RUN

    R(4)     50º 07' 06" NW

    R(4)     50º 07' 06" NW
```

where lines 1 and 2 simply set up some variables to mimic the ones you're using in your code, lines 280 and 285 are your own posted coded, and line 300 is my suggested version, which seems to do what you want in a simpler way.

As you can see in the sample RUN, both your code and my line 300 code do produce exactly the same output for this particular example of yours. You may want to fine-tune it as you need to cover other more general cases in your program.

If the exact same format is to be used more than once at several different places, put it in their own IMAGE statement (say line 900 IMAGE ..., and then simply DISP USING 900; .... This will be simpler and more easy to maintain, as you would need to change just the image at line 900 to have the changes propagate to all DISP statements using it, instead of having to change them one by one, individually.

Best regards from V.

Re: HP42S Mini-Challenge: My Original Solution & Comments

*Message #37 Posted by **Gerson W. Barbosa** on 11 May 2007, 10:28 a.m.,*
in response to message #36 by Valentin Albillo

Hi Valentin,

Considering the HP-71B is relatively new to me, I was pleased with my HP-71B port of a CASIO PB-700 program I wrote years ago, except for those two lines. I will follow your suggestion in the definitive version.

The Owner's Manual doesn't cover formatting strings properly. A glance at the Reference Manual seemed more promising to me. I will print it out as well.

Thank you very much for the free lesson! :-)

Best regards,

Gerson.

[Return to Index | Top of Index]

Go back to the main exhibit hall