[VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier" Threaded Mode | Linear Mode

04-01-2019, 02:49 AM **Post: #21**

Juan14
Junior Member

Posts: 24
Joined: Jan 2014

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

You are right Albert and I can't find a way around.

EMAIL PM FIND QUOTE REPORT

04-01-2019, 05:25 PM (This post was last modified: 04-03-2019 08:01 PM by Albert Chan.) **Post: #22**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Just figured out how to improve cin(x) accuracy for large x

cin(x) = arcsin(cin(sin(x))) = nest(arcsin, cin(nest(sin, x, n)), n)

Pick enough nested sin's so cin argument is small, say below 0.1 radian

```
cin[x0_] := Block[ {n=0, x=x0+0.0},
   While[Abs[x] ≥ 0.1, x = Sin[x]; n++];
   Nest[ArcSin, x - (1/18) x^3 - (7/1080) x^5 - (51/32285) x^7, n]
]
```

Above cin(x) setup give about 12 digits accuracy:

x	cin(x)	cin(cin(cin(x))) - sin(x)
0.0	0.0	+0.0
0.2	0.199553461081	-1.9e-16
0.4	0.396375366278	+1.8e-14
0.6	0.587446695546	-1.1e-16
0.8	0.769025184826	-9.1e-14
1.0	0.935745970819	+1.4e-13
Pi/2.	1.210368344457	+2.6e-13
-0.71	-0.688778525307	-1.6e-13
2.019	1.026923318694	+6.4e-13

Edit: changed x^7 coefficient from -0.00158 to -51/32285 to get better accuracy

EMAIL PM FIND QUOTE REPORT

04-01-2019, 06:42 PM (This post was last modified: 04-01-2019 08:29 PM by J-F Garnier.) **Post: #23**

J-F Garnier
Senior Member

Posts: 304
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

> **Albert Chan Wrote:** (04-01-2019 05:25 PM)
> Just figured out how to improve cin(x) accuracy for large x
>
> cin(x) = arcsin(cin(sin(x))) = nest(arcsin, cin(nest(sin, x, n)), n)
>
> Pick enough nested sin's so cin argument is small, say below 0.1 radian

```
cin[x0_] := Block[ {n=0, x=Evaluate[x0+0.0]},
   While[Abs[x] ≥ 0.1, x = Sin[x]; n++];
   Nest[ArcSin, x - (1/18) x^3 - (7/1080) x^5 - 0.00158 x^7, n]
]
...
```

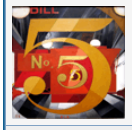
Excellent !
Here is the HP71 version and results, after decipher of your code (not familiar with that language...):

```
10 ! SSMC24
20 A=-1/18 @ B=-7/1080 @ C=-.00158
30 DEF FNC(X)
40 N=0
50 X=SIN(X) @ N=N+1 @ IF ABS(X)>=.1 THEN 50
60 ! X=X+A*X^3+B*X^5+C*X^7
61 X=C*X^7+B*X^5+A*X^3+X ! better
70 FOR I=1 TO N @ X=ASIN(X) @ NEXT I
80 FNC=X
90 END DEF
100 !
110 FOR X=.2 TO 1 STEP .2
120 Y=FNC(FNC(FNC(X)))
130 PRINT X;Y;SIN(X);Y-SIN(X)
140 NEXT X

>RUN
.2 .198669330795 .198669330795 0
.4 .389418342314 .389418342309 .000000000005
.6 .564642473542 .564642473395 .000000000147
.8 .717356091570 .717356090900 .000000000670
1. .841470984040 .841470984808 -.000000000768

>FNC(PI/2);FNC(FNC(FNC(PI/2)))
1.2103683495 .999999998579

>FNC(-0.71)
-.688778525229

>FNC(2.019)
1.02692332142
```

J-F
[Edited: reversed the order of the polynom term evaluation, for slightly better accuracy]

EMAIL PM WWW FIND QUOTE REPORT

04-03-2019, 12:43 AM **Post: #24**



Valentin Albillo
Senior Member

Posts: 376
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hi, all:

Continuing with my original solutions, today it's time for:

Tier 4 - The Challenge:

*Consider the **n**-point dataset (1, 2), (2, 3), (3, 5), (4, 7), (5, 11), (6, 13), ..., (n, p_n) (the prime numbers), and the (n-1)st degree polinomial fit to this dataset of the form:*

$$P(x) = \mathbf{a_0} + \mathbf{a_1}\,(x-1) + \mathbf{a_2}\,(x-1)\,(x-2) + \ldots + \mathbf{a_{n-1}}\,(x-1)\,(x-2)\,(x-3)\,\ldots\,(x-(n-1))$$

*Write a program that takes no inputs but computes and outputs the limit of the sum of the coefficients $\mathbf{a_0}$, $\mathbf{a_1}$, ... , $\mathbf{a_{n-1}}$ when **n** tends to infinity.*

My original solution:

My original solution for the **HP-71B** is this *4-liner* (168 bytes):

```
1  DESTROY ALL @ OPTION BASE 0 @ REPEAT @ N=N+1 @ DIM C(N) @ T=S
2  FOR I=1 TO N @ C(I)=FPRIM(C(I-1)+1) @ NEXT I @ S=0
3  FOR I=1 TO N-1 @ FOR J=N TO I+1 STEP -1 @ C(J)=C(J)-C(J-1) @ NEXT J @ NEXT I
4  FOR I=1 TO N @ S=S+C(I)/FACT(I-1) @ NEXT I @ UNTIL S=T @ DISP N;S

>RUN
      20    3.40706916561     { it converged to the limit after fitting the first 20 primes:
2, 3, 5, ..., 71) }
```

Notes:

- *Line* ***1*** initializes and begins the loop to compute the sum of the first *n* coefficients
- *Line* ***2*** fills an array with the first *n* primes
- *Line* ***3*** computes the forward differences in-place (replacing the primes)
- *Line* ***4*** computes the sum of the coefficients (differences / factorials) and loops back until it agrees with the previous sum, then outputs it

That's all for **Tier 4**, thanks a lot to **Albert Chan** for his interest in this particular tier and congratulations for providing a correct solution and some explanation but please, **Albert**, next time **do** provide *actual code for an HP calculator* of your choice, so that people can try your solution for themselves.

In the next days I'll post my solutions for the remaining tiers.

V.

PM FIND EDIT QUOTE REPORT

04-03-2019, 04:05 AM **Post: #25**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

> **Valentin Albillo Wrote:** ➡ (04-03-2019 12:43 AM)
>
> >RUN
> 20 **3.40706916561** *{ it converged to the limit after fitting the first 20 primes: 2, 3, 5, ..., 71) }*

I think you meant sum converged using 19 primes (20 primes to confirm 12-digits convergence)

sum using 19 primes = **414453 270752** 384363 / 19! ≈ 3.40706 916563
sum using 20 primes = **414453 270752 5**80132 / 19! ≈ 3.40706 916563

Also, forward difference tables may be built incrementally.

C(1) = p1
C(2) = p2 - p1
C(3) = p3 - 2 p2 + p1,
C(4) = p4 - 3 p3 + 3 p2 - p1,
C(5) = p5 - 4 p4 + 6 p3 - 4 p2 + p1,
...

Above can be simplified without a prime table:

C(1) = p1
C(2) = p2 - C(1)
C(3) = p3 - C(1) - 2 C(2)
C(4) = p4 - C(1) - 3 C(2) - 3 C(3)
C(5) = p5 - C(1) - 4 C(2) - 6 C(3) - 4 C(4)
...

EMAIL PM FIND QUOTE REPORT

04-03-2019, 04:57 AM (This post was last modified: 04-27-2019 07:02 PM by Albert Chan.) **Post: #26**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

I only have a HP-12C, which is not powerful enough to make primes, build delta tables ...

XCas code:

```
terms(n) := {
local c, s, p, j, k;
c := flatten(matrix(n,0)); s := 0; p := 0;
for(j:=0; j<n; j++) {
    p := nextprime(p);
    c[j] := p;
    for(k:=0; k<j; k++) c[j] := c[j] - comb(j,k) * c[k];
    s += c[j] / float(j!);
    print(p, s);
    }
}
```

terms(20) →

```
02 2.0
03 3.0
05 3.5
07 3.33333333333
11 3.45833333333
13 3.38333333333
17 3.41527777778
19 3.40476190476
23 3.4076140873
29 3.40696097884
31 3.40708691578
37 3.40706684905
41 3.4070693814
43 3.40706915834
47 3.40706916344
53 3.40706916625
59 3.40706916552
61 3.40706916564
67 3.40706916563
71 3.40706916563
```

Edit: replaced Python code to XCas, so HP prime user can try out.

EMAIL PM FIND QUOTE REPORT

04-05-2019, 03:58 AM **Post: #27**

Valentin Albillo
Senior Member

Posts: 376
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hi, all:

At long last, today it's time for my final *original solution*, namely:

Tier 5 - The Challenge:

Consider the function ***cin(x)*** which has the defining property that ***cin(cin(cin(x))) = sin(x)***.

Write a program or function which accepts an argument ***x*** in the range *[-Pi, Pi]* and outputs the corresponding value of ***cin(x)*** correct to at least 8-10 digits in the *whole* range. Use it to tabulate ***cin(x)*** for *x = 0.0, 0.2, 0.4, ..., 1.0* and also to compute *cin(Pi/2), cin(-0.71), cin(2.019)*.

My original solution:

My original solution for the **HP-71B** is the following *user-defined function* (plus initialization code):

```
1  DESTROY ALL @ OPTION BASE 1 @ DIM C(7) @ READ C
2  DATA 1,-1/18,-7/1080,-643/408240,-13583/29393280,-29957/215550720,-24277937/648499737600

3  DEF FNC(X) @ L=0 @ M=1/3 @ REPEAT @ X=SIN(X) @ L=L+1 @ UNTIL ABS(X)<M
```

```
4  S=0 @ FOR Z=1 TO 7 @ S=S+C(Z)*X^(2*Z-1) @ NEXT Z
5  FOR Z=1 TO L @ S=ASIN(S) @ NEXT Z @ FNC=S @ END DEF
```

Instead of tabulating it for *0.0, 0.2, ..., 1.0* as I originally asked, let's better tabulate it for *x* from *0* to *Pi/2* in steps of *Pi/10*:

```
6  FOR X=0 TO PI/2 STEP PI/10
7  Y=FNC(FNC(FNC(X))) @ DISP X;FNC(X);Y;SIN(X);Y-SIN(X) @ NEXT X

>FIX 10
>RUN
             x            cin(x)         cin(cin(cin(x)))     sin(x)        Error
-------------------------------------------------------------------------------
0.0000000000   0.0000000000      0.0000000000   0.0000000000     0
0.3141592654   0.3124163699      0.3090169944   0.3090169944  -1.0E-12
0.6283185307   0.6138343796      0.5877852523   0.5877852523   2.2E-11
0.9424777961   0.8897456012      0.8090169944   0.8090169944   4.1E-11
1.2566370614   1.1122980783      0.9510565164   0.9510565163   1.0E-10
1.5707963268   1.2103683445      1.0000000000   1.0000000000   1.0E-11
```

So we've got 10 correct decimals or better, as the error in *cin(x)* is even smaller than the error in *cin(cin(cin(x)))-sin(x)* which doesn't exceed 10^{-10}. As for the discrete values asked in the challenge:

```
>FIX 10 @ FNC(PI/2); FNC(-0.71); FNC(2.019)

       1.2103683445   -0.6887785253   1.0269233188
```

Notes:

- ***Line 4*** evaluates the formal series in a simple loop but that's not optimal. I know of several better ways to evaluate the series but I don't want to digress from the main subject, which is the computation of ***cin(x)***.

- **Albert Chan** found the correct *conjugation (sin/arcsin)* procedure to increase accuracy and *almost* duplicated my original solution but there's an important difference which affects both accuracy and running time. He used up to the x^7 term in his formal series expansion:

  $x - (1/18)\ x^3 - (7/1080)\ x^5 - 0.00158\ x^7$

  and then iterated the sine of the argument till it got **< 0.1**, while my original solution uses up to the x^{13} term:

  $x - 1/18\ x^3 - 7/1080\ x^5 - 643/408240\ x^7 - 13583/29393280\ x^9 - 29957/215550720\ x^{11} - 24277937/648499737600\ x^{13}$

  and iterates until the *sin* gets **< 1/3**. This way significantly *fewer sin/arcsin* iterations are needed and the computation is both more accurate and faster. For instance, to see how many iterations my function performs when computing ***cin(Pi/2)*** just execute this:

  ```
  >FNC(PI/2);L
                  1.2103683445    24
  ```

  thus **24** *sin/arcsin* were necessary for this argument while in **J-F Garnier**'s *HP-71B* version of Albert Chan's code several hundred sines/arcsines are necessary to bring this argument below 0.1, which explains why it takes much longer and worse, several decimal places are *lost* in the process.

- My solution will also work for ***tin(x)*** , defined as ***tin(tin(x)) = sin(x)*** , by simply replacing the coefficients in the `DATA` statement at line 2 by those of its own formal series, namely:

  $x - x^3/12 - x^5/160 - 53/40320\ x^7 - 23/71680\ x^9 - 92713/1277337600\ x^{11} + ...$

  and of course it will also work for any other such functions as well.

- The coefficients of the formal series for ***cin(x)*** and ***tin(x)*** can be obtained in a number of ways (even manually for the first 4 or so !), most easily by using some *CAS* which can deal with formal series (even a version of *Newton's method* for solving *f(x) = 0* can be put to the task), but it's important to be aware that both formal series *do* ***not*** *converge*.

  In fact, their *radius of convergence* is **0** and thus they behave like *asymptotic* series, so you can't get arbitrarily accurate results by taking more and more terms, you must instead truncate the series after a certain number of terms to get the most accurate results. Using further terms only *worsens* the accuracy.

- Although at first sight the coefficients of the formal series for ***cin(x)*** and ***tin(x)*** seem to (slowly) get smaller and smaller, matter of fact they tend to grow ever bigger after a while, tending to infinity. For instance, for ***tin(x)*** we find that the smallest coefficient in absolute value is:

  $Coeff_{37}$ = *-0.000000000594338574503*

  but afterwards we have, e.g.:

  $Coeff_{101}$ = *0.0833756228055*

  $Coeff_{151}$ = *388536047335.239*

  $Coeff_{201}$ = *655542387465125662381118699 1.51*

  $Coeff_{251}$ = *-353652204927082961403770877488044401702544920 09.57*

That's all for **Tier 5**, I could say a *whole* lot more about this topic and post additional code and results but this post is long enough as it is so I'll stop right now.

Thank you very much to **Albert Chan, J-F Garnier**, **Oulan** and **Gerson W. Barbosa** for your valuable contributions and to **Werner** for your interest, I hope you enjoyed it all !

V.

PM FIND EDIT QUOTE REPORT

04-05-2019, 09:40 PM (This post was last modified: 04-07-2019 06:12 PM by Albert Chan.) **Post: #28**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Below Lua code scale cin argument to [sin(0.5), 0.5], do cin, then undo asin/sin's

```
local sin, asin = math.sin, math.asin

function cin(x)
   local y, n = x*x, 0
   while y > 0.25 do x=sin(x); y=x*x; n=n+1 end
   if y < 0.0324 then   -- |x| < 0.18
      local z = y*(0.00013898 + y*0.00003744) + 13583/29393280
      x = x - x*y*(1/18 + y*(7/1080 + y*(643/408240 + y*z)))
      return n==0 and x or asin(x)
   end
   while y < 0.229848847 do x=asin(x); y=x*x; n=n-1 end

   y = y - 0.2399       -- |x| = [sin(0.5), 0.5]
   y = 0.013724194890539722 + y*(
        0.058965322546572385 + y*(
        0.007795773378183463 + y*(
        0.002109528417736682 + y*(
        0.000663984666232017 + y*(
        0.000199482968029459 )))))

   x = x - x*y          -- x = cin(x)
   for i=1,n do x = asin(x) end
   for i=1,-n do x = sin(x) end
   return x
end

Result *very* accurate. Example:

x = 2.019
cin(x) = 1.02692 331869 35764
cin(cin(x)) = 0.956628 929996 1186
cin(cin(cin(x))) = 0.90122 698939 98129
math.sin(x)      = 0.90122 698939 98126
```

EMAIL PM FIND QUOTE REPORT

04-07-2019, 06:58 PM **Post: #29**

John Keith
Senior Member
Posts: 408
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Though I did not participate in this challenge, I have taken the liberty of adapting Valantin's Albert's programs into RPL with a twist- unlimited precision.

This program does not run until convergence but a fixed number of iterations, which is the number n that is input into the program. The program requires the external libraries ListExt, GoferLists, and Long Float.

```
%%HP: T(3)A(R)F(.);
@ Generate list of primes:
\<< \-> n
\<< 2 2 n
START DUP NEXTPRIME
NEXT n \->LIST
@ Inverse binomial transform of above list:
DUP HEAD SWAP 2 n
START \GDLIST DUP HEAD SWAP
NEXT DROP n \->LIST
@ List of factorials 0 through n:
1 n 1 - LSEQ
\<< *
\>> Scanl1 +
@ Divide to form list of ratios:
/
@ Accumulate above list:
\<< + EVAL
\>> Scanl1
@ Convert to list of LongFloats:
1.
\<< \->FNUM
\>> DOLIST
\>>
\>>
```

To begin, store a number into the variable DIGITS which sets the precision that LongFloat uses. In this example, I used 50. for DIGITS and 60 for the number of iterations.

I then used the following simple program to turn the resulting list into a string suitable for display or printing:

```
\<< \->STR 3. OVER SIZE 2. - SUB " " 13. CHR 10. CHR + SREPL DROP
\>>
```

The result:

```
2
3
3500000000000000000000000000000000000000000000000.E-49
3333333333333333333333333333333333333333333333333.E-49
3458333333333333333333333333333333333333333333333.E-49
3383333333333333333333333333333333333333333333333.E-49
3415277777777777777777777777777777777777777777778.E-49
3404761904761904761904761904761904761904761904761.E-49
3407614087301587301587301587301587301587301587301.E-49
3406960978835978835978835978835978835978835978836.E-49
3407086915784832451499118165784832451499118165784.E-49
3407066849046015712682379349046015712682379349046.E-49
3407069381396638341082785527229971674416118860563.E-49
3407069158336519447630558741669852780963892075003.E-49
3407069163441001238620286239333858381477429096476.E-49
3407069166245216179078612941046803480665914528348.E-49
3407069165524423202581271364340147408930477713546.E-49
3407069165640634788189643465086975824604209291417.E-49
3407069165625787326237375084074257570816688290838.E-49
3407069165627396671778971482478616326307449540368.E-49
3407069165627244259968403780412014504871920742752.E-49
3407069165627257030588248823864490967372839203226.E-49
3407069165627256084539928995379575004970755647027.E-49
3407069165627256145278130423131107299403592641395.E-49
3407069165627256142116296184345441647565534072135.E-49
3407069165627256142216885951078104426779609546510.E-49
```

340706916562725614222036232272277922379545747668 70.E-49
340706916562725614221934999366050210282681659329 35.E-49
340706916562725614221946807107354932122040285443 06.E-49
340706916562725614221945750661534900586956862706 06.E-49
340706916562725614221945831443221911130493015429 36.E-49
340706916562725614221945825966110053035472581965 12.E-49
340706916562725614221945826300261117671642068580 21.E-49
340706916562725614221945826281843667024261278444 81.E-49
340706916562725614221945826282756663488667801107 05.E-49
340706916562725614221945826282716616928583099013 67.E-49
340706916562725614221945826282718106009362800349 71.E-49
340706916562725614221945826282718065043360812169 50.E-49
340706916562725614221945826282718065291516976292 34.E-49
340706916562725614221945826282718065361704656297 60.E-49
340706916562725614221945826282718065354993007459 12.E-49
340706916562725614221945826282718065355425486888 40.E-49
340706916562725614221945826282718065355402415287 38.E-49
340706916562725614221945826282718065355403485570 90.E-49
340706916562725614221945826282718065355403442395 72.E-49
340706916562725614221945826282718065355403443832 89.E-49
340706916562725614221945826282718065355403443801 87.E-49
340706916562725614221945826282718065355403443801 40.E-49
340706916562725614221945826282718065355403443801 51.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 49.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 49.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 51.E-49
340706916562725614221945826282718065355403443801 50.E-49
340706916562725614221945826282718065355403443801 51.E-49

It can be observed that:

-- LongFloat numbers are not very user-friendly. 🙂

-- There is noise in the last digit, so really 49-digit accuracy in this case.

-- Rate of converge increases, only about 56 iterations required to confirm 49 digits.

EMAIL PM FIND QUOTE REPORT

04-08-2019, 05:36 PM (This post was last modified: 04-10-2019 04:41 AM by Albert Chan.) **Post: #30**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

I posted cin(x) puzzle to the Lua mailing list, and got an elegant solution from Egor Skriptunoff.
Taylor coefficients built on the fly, without any need for CAS. 🙂
http://lua-users.org/lists/lua-l/2019-04/msg00063.html

Below code modified a bit for speed, accuracy, and extended cin(x) for tin(x):

Quote:

```
local sin, asin = math.sin, math.asin

local function g(k, m, c, a)     -- assume c[0] = 1, m = [0,999]
   if k < 2 then return c[m] end
   local i = 1000 * k + m
   local r = a[i]
   if r then return r end
   r = g(k-1, m, c, a) + c[m] -- case for j=0 and j=m
   for j = 1, m-1 do
      r = r + c[j] * g(k-1, m-j, c, a)
   end
   a[i] = r
   return r
end
```

```
local function f(d, c, a)
    local r = 0
    for j = 1, #c do
        r = r + d[j] * g(2*j+1, #c+1-j, c, a)
    end
    return r
end

function maclaurin_of_cin()
    local c, c2, s = {}, {}, 1
    return function(k)
        for n = #c + 1, k do
            s = -(2*n)*(2*n+1)*s
            local a = {}
            local r, r2 = f(c, c, a), f(c2, c, a)
            local t = (1/s-r-r2)/3
            c[n], c2[n] = t, r + 2*t
        end
        return c[k]
    end
end

function maclaurin_of_tin()
    local c, s = {}, 1
    return function(k)
        for n = #c + 1, k do
            s = -(2*n)*(2*n+1)*s
            c[n] = (1/s - f(c, c, {})) / 2
        end
        return c[k]
    end
end

function egor(x)
    if x*x > 0.25 then return asin(egor(sin(x))) end
    local r, p, s, n, R = 0, x, x*x, 0
    repeat
        R, n, p = r, n+1, p*s
        r = r + maclaurin_coefs(n) * p
    until r == R
    return r + x
end
```

```
lua> maclaurin_coefs = maclaurin_of_tin()
lua> for i=50,125,25 do    -- match post #28 Coefs
:      print(2*i+1, maclaurin_coefs(i))
:      end
101  0.08337562280550574
151  388536047335.2163
201  6.555423874650777e+027
251  -3.536522049267692e+046

lua> function nest(f,x,n) for i=1,n do x=f(x);print(i, x) end end
lua> nest(egor, 2.019, 2) -- egor = tin
1      0.9894569770589354
2      0.9012269893998129

lua> maclaurin_coefs = maclaurin_of_cin()
lua> nest(egor, 2.019, 3) -- egor = cin
1      1.0269233186935764
2      0.9566289299961186
3      0.9012269893998129

lua> math.sin(2.019)
0.9012269893998126
```

EMAIL PM FIND QUOTE REPORT

04-09-2019, 08:53 PM **Post: #31**

Gerson W. Barbosa

Posts: 1,155
Joined: Dec 2013

Senior Member

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

> **John Keith Wrote:** ⇨ (04-07-2019 06:58 PM)
> It can be observed that:
>
> -- LongFloat numbers are not very user-friendly.

They needn't be so.

3407069165627256142219458262827180653554034438015 1.E-49

\<< ZZ\<-\->F -51 FC? { "." } { "," } IFTE SWAP ROT \->STR DUP SIZE ROT + OVER 1 ROT SUB ROT + " " ROT + 1 ROT REPL
\>>

EVAL

-->

3.4070691656272561422194582628271806535540344380151

EE7Dh
100 bytes,

which can be optimized, of course.

EMAIL PM FIND QUOTE REPORT

04-10-2019, 01:37 AM **Post: #32**

Valentin Albillo
Senior Member

Posts: 376
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

.
Hi again, all

> **Albert Chan Wrote:** ⇨ (04-05-2019 09:40 PM)
> Below Lua code scale cin argument to [sin(0.5), 0.5], do cin, then undo asin/sin's [...] **Result *very* accurate**.
> Example:
> x = 2.019
> cin(x) = 1.02692 331869 35764
> cin(cin(x)) = 0.956628 929996 1186
> cin(cin(cin(x))) = 0.90122 698939 98129
> math.sin(x) = 0.90122 698939 98126

Indeed, *impressive* accuracy ! Thanks a lot for your Lua code, **Albert Chan**, I hope you'll adapt it to some HP calc's native programming language when you eventually get your hands on one (apart from the **HP-12C**, that is).

> **John Keith Wrote:** ⇨ (04-07-2019 06:58 PM)
> Though I did not participate in this challenge, I have taken the liberty of adapting Valantin's Albert's programs into RPL with a twist- unlimited precision.[...] The result:
> [...]
> 3407069165627256142219458262827180653554034438015 1.E-49
> [...]
> -- Rate of converge increases, only about 56 iterations required to confirm 49 digits.

Yes, it does converge very fast and I *love* multiprecision computations and results. In fact, I don't understand why *HP* didn't ever include it in some of its advanced models right from the box (at least *double precision* as in some *SHARP* models which would do 20 digits without batting an eyelid.)

Thanks a lot for your interest and your *RPL high-precision* results, much appreciated.

> **Albert Chan Wrote:** ⇒ (04-08-2019 05:36 PM)
> I posted cin(x) puzzle to the Lua mailing list, and got an elegant solution from Egor Skriptunoff. Taylor coefficients built on the fly, without any need for CAS.
> http://lua-users.org/lists/lua-l/2019-04/msg00063.html
> [...]
> Below code modified a bit for speed, accuracy, and extended **cin(x)** for **tin(x)**:
> [...]
> lua> function nest(f,x,n) for i=1,n do x=f(x);print(i, x) end end
> lua> nest(egor, 2.019, 2) -- egor = **tin**
> 1 0.9894569770589354
> 2 0.9012269893998129
>
> lua> maclaurin_coefs = maclaurin_of_cin()
> lua> nest(egor, 2.019, 3) -- egor = **cin**
> 1 1.0269233186935764
> 2 0.9566289299961186
> 3 0.9012269893998129
>
> lua> math.**sin**(2.019)
> 0.9012269893998126

As I said before, *truly excellent* accuracy. Also thank you very much for posting my challenge to the **Lua** forums, for giving me credit for it, and for your outstandingly clear code which also includes an implementation and high-precision results fot the **tin(x)** function I mentioned in the challenge. Again, really appreciated.

> **Gerson W. Barbosa Wrote:** ⇒ (04-09-2019 08:53 PM)
> > **John Keith Wrote:** ⇒ (04-07-2019 06:58 PM)
> > It can be observed that: [...] LongFloat numbers are not very user-friendly.
>
> They needn't be so.
>
> 3407069165627256142219458262827180653554034438015 1.E-49
> [...]
> 3.407069165627256142219458262827180653554034438015 1

Very good effort to increase usability. As you know RPL is not my thing but I can appreciate your ingenuity. Thanks, **Gerson**.

Best regards to all of you.
V.
.

PM FIND EDIT QUOTE REPORT

04-10-2019, 06:11 PM (This post was last modified: 04-10-2019 06:11 PM by John Keith.) **Post: #33**

John Keith
Senior Member

Posts: 408
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

> **Valentin Albillo Wrote:** ⇒ (04-10-2019 01:37 AM)
> Yes, it does converge very fast and I *love* multiprecision computations and results. In fact, I don't understand why *HP* didn't ever include it in some of its advanced models right from the box (at least *double precision* as in some *SHARP* models which would do 20 digits without batting an eyelid.)
>
> Thanks a lot for your interest and your *RPL high-precision* results, much appreciated.

Thanks for your kind words, Valentin. The HP 49 and 50 do have exact integers whose size is limited only by memory. Though LongFloat is an external library and is a bit rough around the edges, its precision can be set up to 9999 digits. At that point, I think formatting becomes moot.

EMAIL PM FIND QUOTE REPORT

04-13-2019, 10:01 PM **Post: #34**

Bernd Grubert

Posts: 70
Joined: Dec 2013

Member

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hello Valentin,
I don't understand the term composite in the context of Tier 2. I first thought, that the result of S_B must have at least 2 digits, but that can't be the point. Please explain what's meant by composite.

Best regards
Bernd

EMAIL PM FIND QUOTE REPORT

04-14-2019, 12:35 AM (This post was last modified: 04-14-2019 10:53 PM by Albert Chan.) **Post: #35**

Albert Chan
Senior Member

Posts: 627
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

I recently created nextprime.lua, which is needed for solving Tier 2 puzzle.
My Lua code available in https://github.com/achan001/PrimePi

Quote:

```
p = require 'nextprime'

function sb(base, n)
   local t, d = 0
   while n > 0 do
      d = n % base; t = t + d; n = (n-d)/base
   end
   return t
end

function sb_find(base, n)
   if not n then n=1 end
   return function()
      repeat n = p.nextprime(n) until not p.isprime(sb(base, n))
      return n
   end
end
```

lua> function loop(n,f) for i=1,n do io.write(f(),' ') end print() end

lua> **seq=sb_find(7)**
lua> loop(10,seq)
7 4801 9547 9601 11311 11317 11941 11953 13033 13327

lua> **seq=sb_find(31)**
lua> loop(10,seq)
31 619 709 739 769 829 859 919 1549 1579

EMAIL PM FIND QUOTE REPORT

04-14-2019, 01:27 AM **Post: #36**

Valentin Albillo
Senior Member

Posts: 376
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hi, **Bernd Grubert** and **Albert Chan**:

> **Bernd Grubert Wrote:** ➡ (04-13-2019 10:01 PM)
> I don't understand the term ***composite*** in the context of ***Tier 2***. [...] Please explain what's meant by *composite*.

With pleasure. In this context ***composite*** simply means ***not prime***, i.e., if a number is not prime (thus it can be factored as the product of at least two not necessarily distinct prime factors) then it is considered ***composite***. For instance:

25 is ***composite*** because it's ***not*** *a* ***prime***, as it can be factored as **5 * 5** (two identical *prime* factors).

23 isn't ***composite*** because it's *a* ***prime***, as its prime factoring is just itself, **23** (a single prime).

Thanks for your interest. Should you have any further doubts, just tell me.

> **Albert Chan Wrote:** ⇒ (04-14-2019 12:35 AM)
> I recently created nextprime.lua, which is needed for solving Tier 2 puzzle.
> [...]
> lua> seq=sb_find(***7***)
> lua> loop(10,seq)
> ***7 4801 9547 9601 11311 11317 11941 11953 13033 13327***

Nope, this computed sequence for base 7 and all others that follow are *incorrect* and thus not valid solutions for ***Tier 2***. I think you misunderstood what's actually being asked, which I repeat here with some relevant highlighting for your convenience:

- *"Write a program that accepts a base B (2 to 36) and outputs in order those* ***prime*** *numbers N such that* $S_B(N)$ *is* ***composite*** *and* ***distinct*** *from the previous ones, where* $S_B(N)$ *is a function which returns the sum of the base-B digits of a given integer N."*

Best regards to all.
V.

PM FIND EDIT QUOTE REPORT

04-14-2019, 06:39 PM **Post: #37**

Bernd Grubert
Member

Posts: 70
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hello Valentin,
Thanks for the explanation. Now everything is clear.

Best regards
Bernd

EMAIL PM FIND QUOTE REPORT

04-14-2019, 09:57 PM **Post: #38**

John Keith
Senior Member

Posts: 408
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Somehow I had completely missed Tier 2 until I saw Bernd's post #35. Then I thought I had a good program until I saw Albert's reply and realized the uniqueness requirement, so back to the drawing board.

This problem turns out to be a good fit for the 50g and the Prime, both of which have **NEXTPRIME** and **ISPRIME?** as built-in functions.

My program also uses the **I->BL** command plus a couple of other commands from ListExt. I have tried to keep *stackrobatics* to a minimum in the interest of readability.

```
%%HP: T(3)A(R)F(.);
\<< I\->R \-> b n
\<< { } 1 1. n
START NEXTPRIME DUP b I\->BL LSUM DUP
IF ISPRIME?
THEN DROP
ELSE ROT SWAP DUP2
IF POS
THEN DROP SWAP
ELSE + OVER + SWAP
END
END
NEXT DROP DUP SIZE 2. / LDIST EVAL
\>>
\>>
```

Inputs are the base on level 2 and the number of primes to check on level 1. Output are two separate lists, the composites and the primes.

I would classify the size (163 bytes) and speed as reasonable if not exactly prize-winning, and it is sort of cheating as it uses so many pre-existing commands. I shudder to think of writing such a program on a "classic" era machine.

I have checked the first 100000 primes for 7 and 31, which take over 5 minutes each on the emulator, so my results are nowhere near as extensive as Albert's. Still a neat problem, I only wish I had noticed it earlier.

EMAIL PM FIND QUOTE REPORT

04-20-2019, 12:47 PM (This post was last modified: 04-20-2019 12:48 PM by Bernd Grubert.) **Post: #39**

Bernd Grubert
Member

Posts: 70
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Hello Valentin,
here is my solution to Tier 2. It is 192 bytes long, due to the lack of prime number checking and the remainder function on the HP-15C.
I have done the test runs on the HP-15C emulator on a PC, since the processing time on my DM-15L is far too long...

Since the largest integer number the HP-15C can exactly represent is 9,999,999,999. , this implementation of the Miller-Rabin algorithm can check only number up to 99,999.
Due to memory limitations, on the real HP-15C and the DM 15L the longest sequence is 26 values.

For base 31 I got the sequence: 619, 18257, ...,(I stopped at 34139 after ~90 min., because I didn't want to wait any longer)

For base 7 I got the sequence: 4801, ...,(I stopped at 23451 after ~60 min.)

I have attached an HTML-documentation and a txt-file, that can be read into the emulator after changing the extension back to ".15c":
Tier_2.htm (Size: 49.27 KB / Downloads: 0) and
Tier_2.txt (Size: 6.5 KB / Downloads: 1) .

Best regards
Bernd

EMAIL PM FIND QUOTE REPORT

04-21-2019, 09:06 AM (This post was last modified: 04-22-2019 12:47 AM by Gilles.) **Post: #40**

Gilles
Member

Posts: 162
Joined: Oct 2014

RE: [VA] Short & Sweet Math Challenge #24: "2019 Spring Special 5-tier"

Tier 1 :

Here is my solution without reading others responses. I image that there exists better way. This one is "bestial" ;D
Always impressed how fast NewRPL is.

Brutal force :

1/ HP50g NewRPL or RPL

Code:

```
«
 0
 1000001111 1E10 FOR 'n'
  n ->STR
  IF "0" "" SREPL 1 == THEN
   IF "1" "" SREPL 1 == THEN
    IF "2" "" SREPL 1 == THEN
     IF "3" "" SREPL 1 == THEN
      IF "4" "" SREPL 1 == THEN
       IF "5" "" SREPL 1 == THEN
        IF "6" "" SREPL 1 == THEN
```

Solved in **only 1.3s in newRPL** (on my PC) , **116s with HP50g hdw**, much much slower **in 779s in RPL** (on my PC with Emu48). NewRPL 600 times faster in this case on a PC.
2/ HP50g RPL with ListExt, shorter but slower

Code:

```
« 0 1000001111 1E10 FOR n  n I->NL LDDUP SIZE 10 == { 1 + } IFT  11111 STEP »
```

EMAIL PM FIND QUOTE REPORT

« Next Oldest | Next Newest »

Enter Keywords Search Thread

Pages (3): « Previous 1 **2** 3 Next »

NEW REPLY

View a Printable Version
Send this Thread to a Friend
Subscribe to this thread

User(s) browsing this thread: Valentin Albillo*