Posteriori Analysis of Algorithms Through the Derivations of Growth Rate Based on Frequency Count

Proponent:
Guzman, John Paul

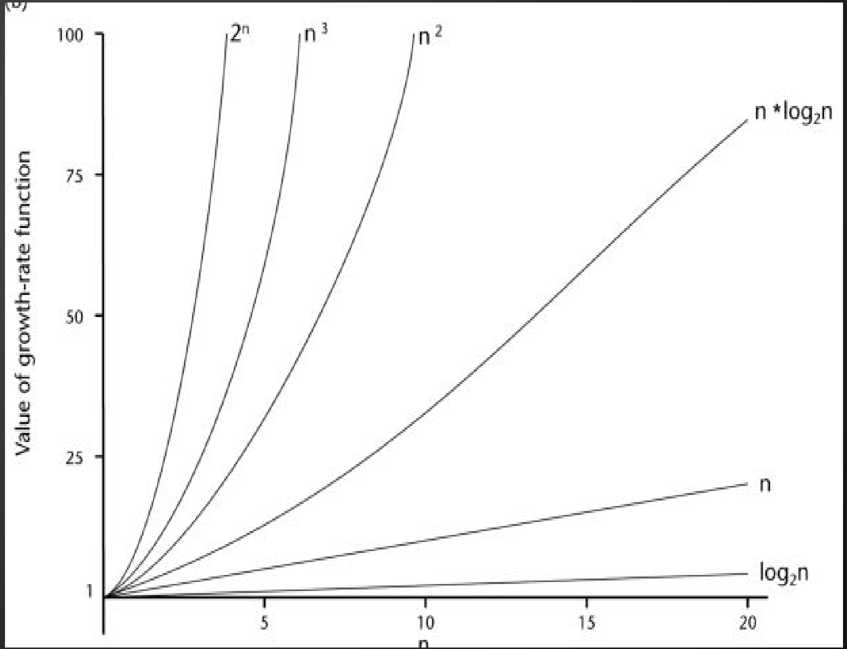
Adviser:
Teresita Limoanco

Outline

I. Overview of the Research
II. Research Problem and Applications
III. Overview of the Proposed System
IV. Testing and Results

Research Description

Developing an "across the board” method of
<u>Algorithm Analysis</u>



Output :

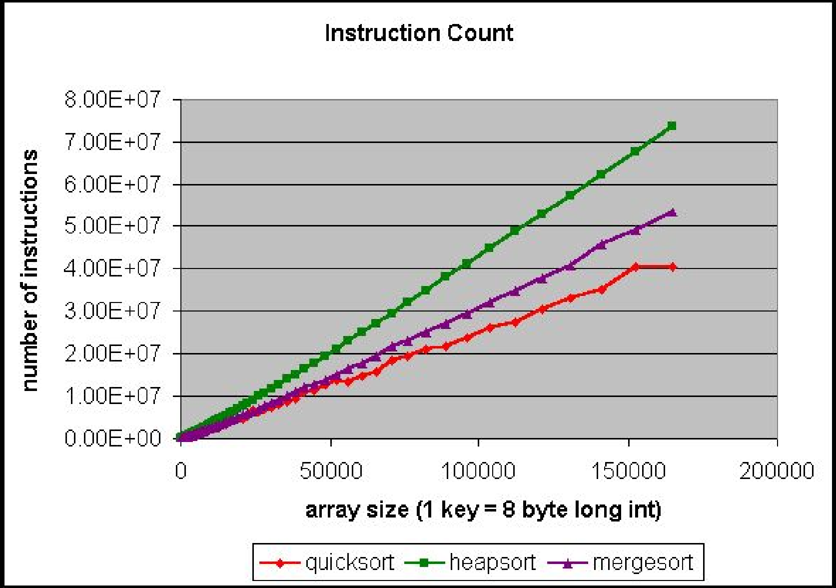
Time complexity

-Asymptotic Behavior

Research Description

Claim / Essence of the Study :

- Performance measurements " → " Asymptotic Behavior



http://www.cpp-home.com/img/164/image9.gif

≅

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Overview of the Current State of Technology

Two ways of analyzing algorithms:

- Posteriori Analysis
- Apriori Analysis

Posteriori Analysis

Idea : Analyze empirical performance of the algorithm

Advantages :

- Can be automated
- Somewhat trivial

Disadvantages :

- Varies with hardware and software
- Does not output asymptotic behavior
    - Necessary for generalizing complexity

Apriori Analysis

Idea : Analyze the logical structure of the algorithm

Advantages :

- Does not depend on external factors
- Does output asymptotic behavior

Disadvantages :

- Done by hand
- Limited by current mathematical methods

Statement of the Research Problem

There are algorithms that are too complex for current apriori and posteriori methods to accurately determine asymptotic behaviors which are necessary for measurement of the general efficiency of algorithms.

Difficult to Analyze Algorithms

- Algorithms with High-Order Linear Recurrence
    - Due to Abel's Impossibility Theorem

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + c_3 a_{n-3} + c_4 a_{n-4} + c_5 a_{n-5}$$

Difficult to Analyze Algorithms

- Algorithms where Master's Method fail
    - Difference between <u>n/log(n) and n*log(2)/log(2)</u> is **not** polynomial

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log(n)}$$

Difficult to Analyze Algorithms

- Lengthy algorithms (hard to keep track of)

http://0.s3.envato.com/files/39890802/_MG_6512.jpg

“Impossible” to Analyze Algorithms

- Algorithms with General Nonlinear Recursion

$$F_n = (n)F_{n-1} + (n^2)c_2F_{n-2} + 3n$$

“Impossible” to Analyze Algorithms

- Algorithms with Irreducible Double Recursion

$$Ack(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ Ack(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ Ack(m-1, Ack(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

A way around the difficulties

- Generating automatic measurements
- Using a method that is numerical
    - Based on asymptotics
    - “Stumbles upon the answer”

Scope and Limitations

- Covers rates of growth from logarithmic to exponential
- Exact answer is not guaranteed
- Convergence test is not completely foolproof
- Results are for one parameter only

Theoretical Framework

The use of asymptotic equivalences

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \longleftrightarrow f(n) \in O(g(n))$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \longleftrightarrow f(n) \in \Theta(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \longleftrightarrow f(n) \in \Omega(g(n))$$

VS.

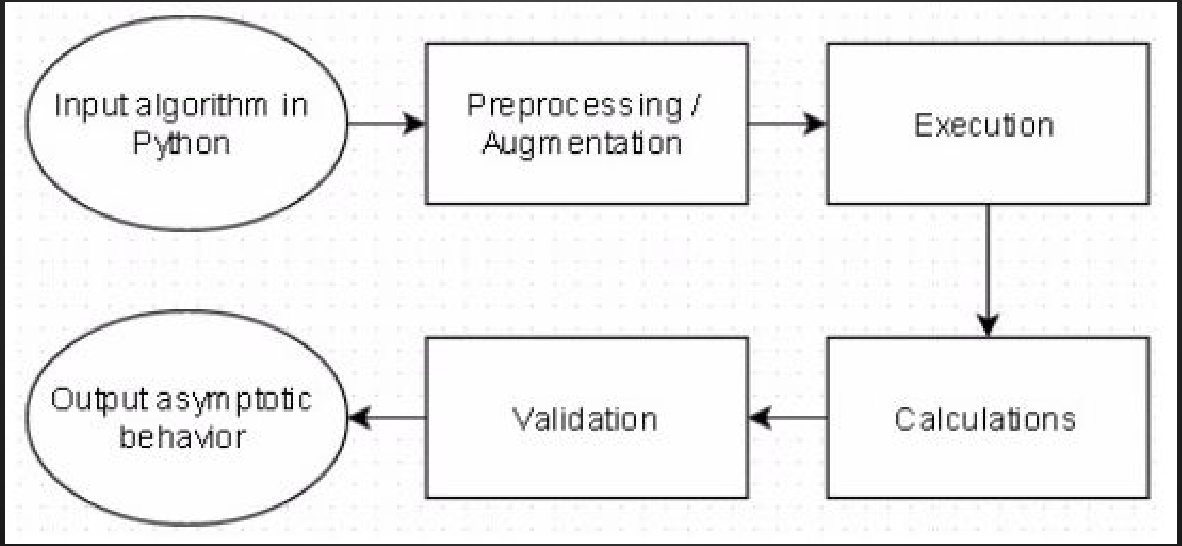
$$f(n) \sim j(n) \text{ as } n \to \infty$$

or

$$(\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1)$$

Overview of the Proposed System

Posteriori method that outputs asymp. behavior



Overview of the Proposed System

Input : Python file of the input algorithm

The Python file must contain:

- Function named “f” taking one parameter
    - “f” will be ran in experiments
- Every function called by “f”

Overview of the Proposed System

Sample input : File containing

```
def f(x):
    if(x<=1):
        return x
    else:
        return fun2(x, “hello world”)

def fun2(x, str):
    print str
    return x/2.0
```

Overview of the Proposed System

Preprocessing : Augmentation of the input file

Inserting freqCount+=1 for lines with:

- Condition Check
- Function call
- Assignment statement
- Return statement

Overview of the Proposed System

Sample augmentation : Resulting file

```
def f(x):
    global freqCount
    freqCount+=1
   if(x<=1):
        freqCount+=1
        return x
     else:
        freqCount+=1
        return fun2(x, "hello world")
```

```
def fun2(x, str):
global freqCount
freqCount+=1
    print str
freqCount+=1
return x/2.0
```

Overview of the Proposed System

Execution : Broken down into five steps

- Running the augmented algorithm for a particular input size
- Storing freqCount measurements
- Setting freqCount variable back to 0
- Increment input size
- Repeat until input size = # of terms

Overview of the Proposed System

Sample execution : Resulting data

Input Size	Measured Frequency Count
0	2
1	2
2	4
3	4
4	4

Overview of the Proposed System

Calculation / Validation :

- Generating asymptotic behavior approx.
- Checking for discontinuities
- Determining the presence of log(n) growth
- Testing for asymptotic equivalence

Generating asymptotic behavior approximations

- Definition used:

$$\left(\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1\right)$$

- Important property (independent):

$$\lim_{n\to\infty}(G_1(n)G_2(n)) = (\lim_{n\to\infty} G_1(n))(\lim_{n\to\infty} G_2(n))$$

- Generalized scope:

$$A^n \cdot n^B \cdot Cn \cdot \sqrt[D]{n} \cdot log_E(n)$$

Generating asymptotic behavior approximations

- Generalized problem

$$a_n = \sum_{i=1}^{m} (e_i^{\,n} \cdot n^{p_i} \cdot c_i \, ln(n,\ hasLog_i))$$

- Simplification (Asymptotics)

$$a_n \sim e_1^{\,n} \, n^{p_1} \, c_1 \, ln(n,\ hasLog_1)$$

Generating asymptotic behavior approximations

- Algebraic Manipulation

$$a_x = e^x x^p c \ln(x)$$

$$\frac{a_x}{e^x x^p \ln(x)} = c = \frac{a_y}{e^y y^p \ln(y)}$$

Generating asymptotic behavior approximations

$$
\begin{aligned}
e = exp(((log(y) - log(z))(log(a_x) - log(a_y) + log(ln(y)) - log(ln(x))) \\
-(log(x) - log(y))(log(a_y) - log(a_z) + log(ln(z)) - log(ln(y)))) \\
\div((log(x) - log(y))(z - y) - (log(y) - log(z))(y - x)))
\end{aligned}
$$

$$
\begin{aligned}
p = ((y - z)(log(a_x) - log(a_y) + log(ln(y)) - log(ln(x))) \\
-(x - y)(log(a_y) - log(a_z) + log(ln(z)) - log(ln(y)))) \\
\div((x - y)(log(z) - log(y)) - (y - z)(log(y) - log(x)))
\end{aligned}
$$

$$
\begin{aligned}
c = exp(((z\, log(y) - ylog(z))(y(log(a_x) - log(ln(x))) - x(log(a_y) - log(ln(y)))) \\
-(ylog(x) - xlog(y))(z(log(a_y) - log(ln(y))) - y(log(a_z) - log(ln(z))))) \\
\div((ylog(x) - xlog(y))(y - z) - (z\, log(y) - ylog(z))(x - y)))
\end{aligned}
$$

Generating asymptotic behavior approximations

Fibo(n): Approximations

enl approx.	pnl approx.	cnl approx.
1.61798561288	0.00087558705	1.78513095706
1.61800288979	0.00058197482	1.78634417355
1.61801402512	0.00038585448	1.78716753511
1.61802119156	0.00025520718	1.78772429388
1.61802579580	0.00016842424	1.78809939417
1.618028749881	0.00011091875	1.78835131160
1.61803064255	0.00007290530	1.78851998432
1.618031853701	0.00004783128	1.78863261076
1.61803262781	0.00003132663	1.78870761822
1.61803312206	0.00002048350	1.78875745226
1.61803343729	0.00001337285	1.78879048638
1.618033638157	0.00000871783	1.78881233782

Actual Values

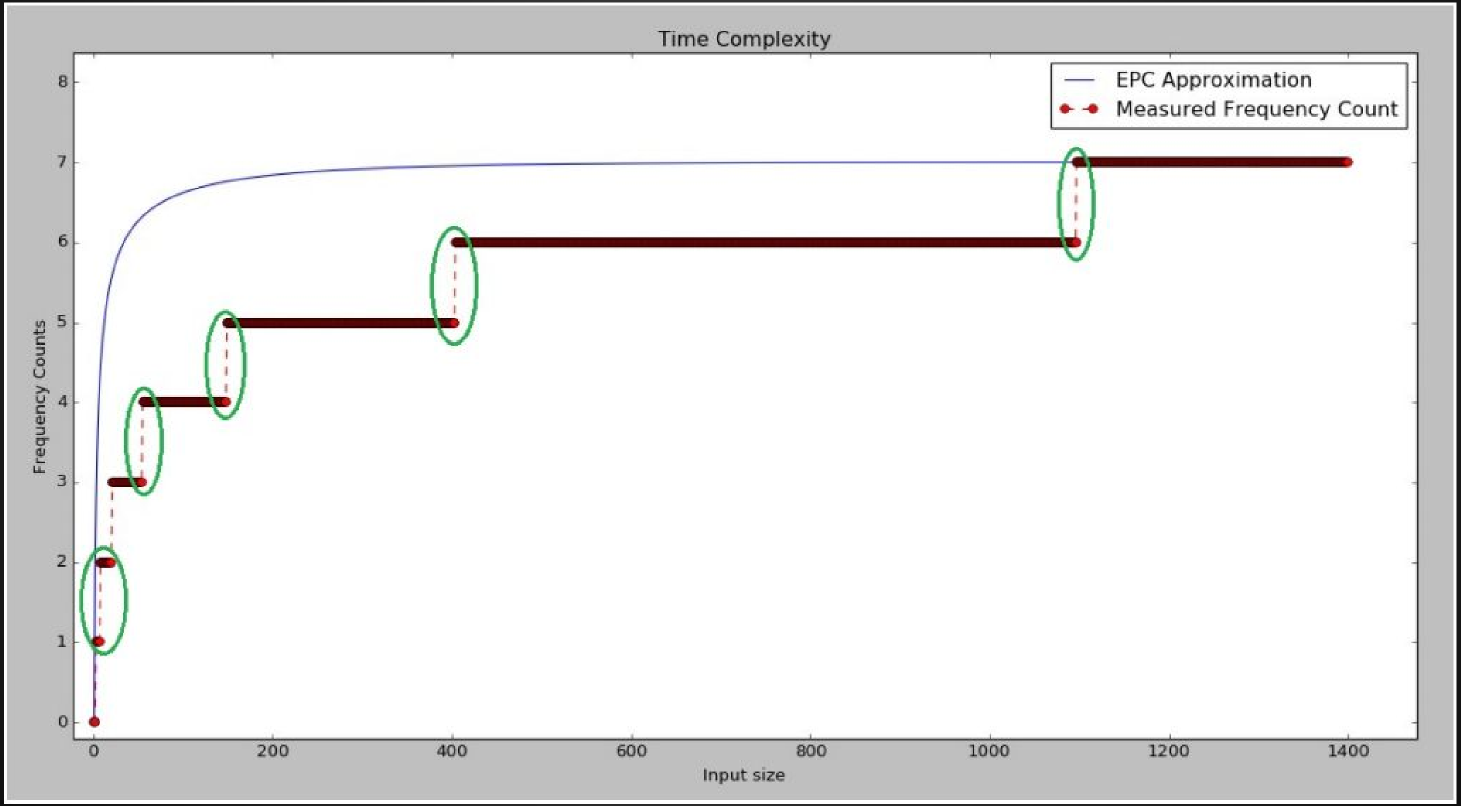
$$e_{Fibo} = \frac{1+\sqrt{5}}{2} \approx 1.618033988750$$

$$p_{Fibo} = 0$$

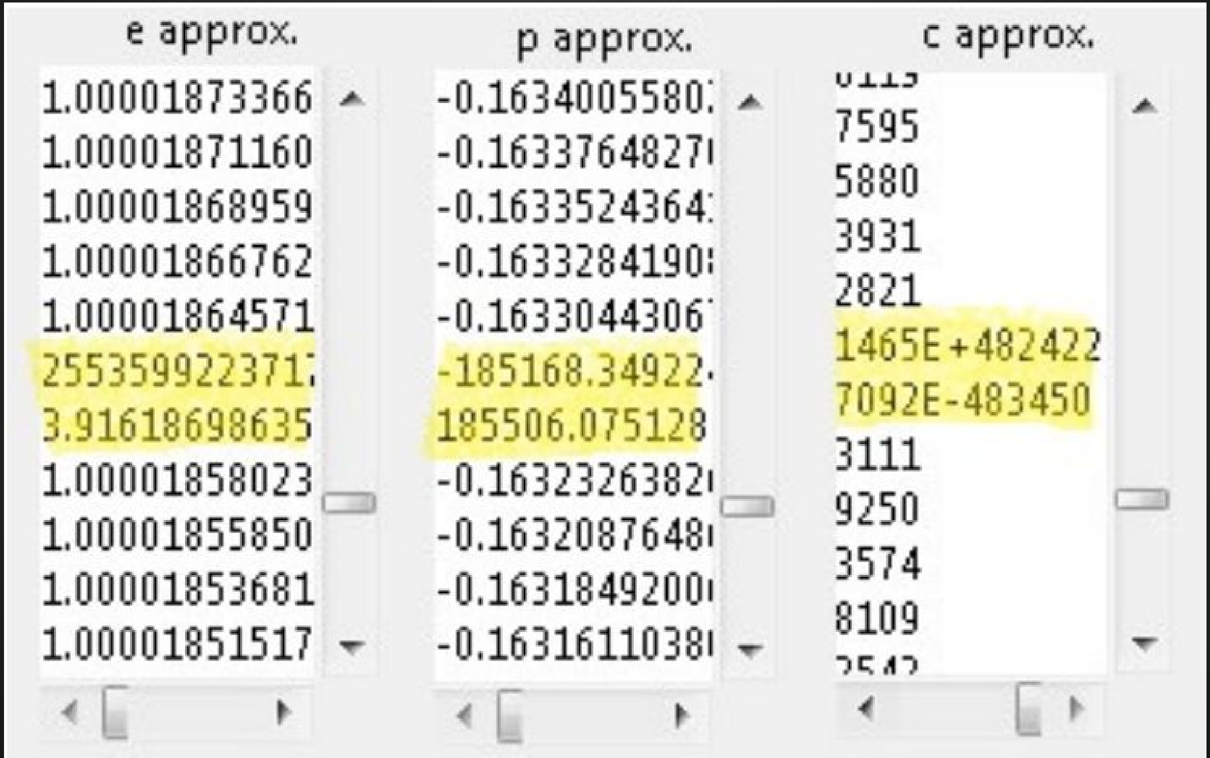
$$c_{Fibo} = \frac{4}{\sqrt{5}} \approx 1.788854382000$$

$$hasLog_{Fibo} = False$$

Checking for Discontinuities



Checking for Discontinuities



Determining hasLog & Convergence

"Voting" heuristics

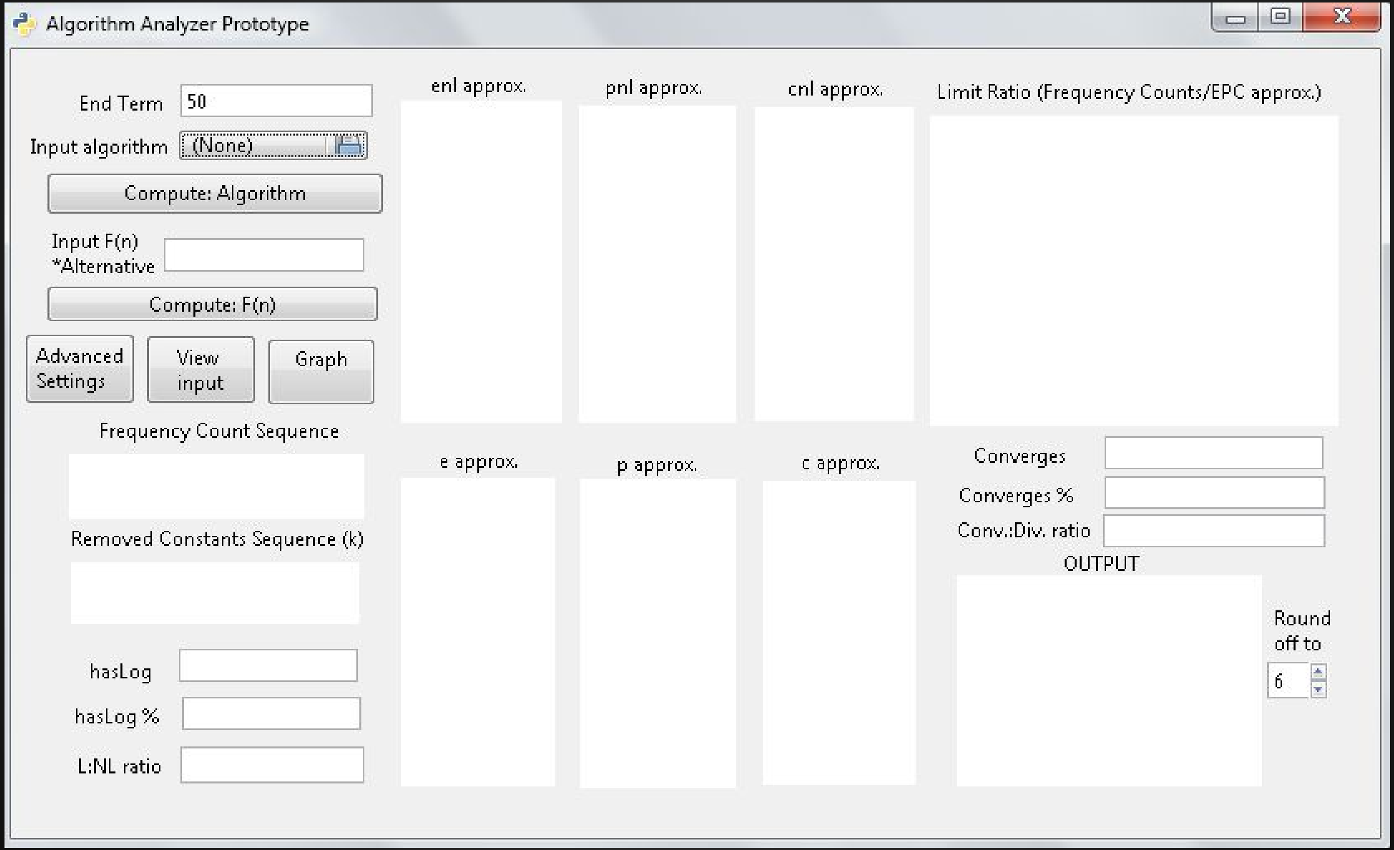
If $\left|a_n - {e_{nl}}^n * n^{p_{nl}} * c_{nl}\right| > \left|a_n - e^n * n^p * c\ln(n)\right|$

then "No Log" case gains 1 vote

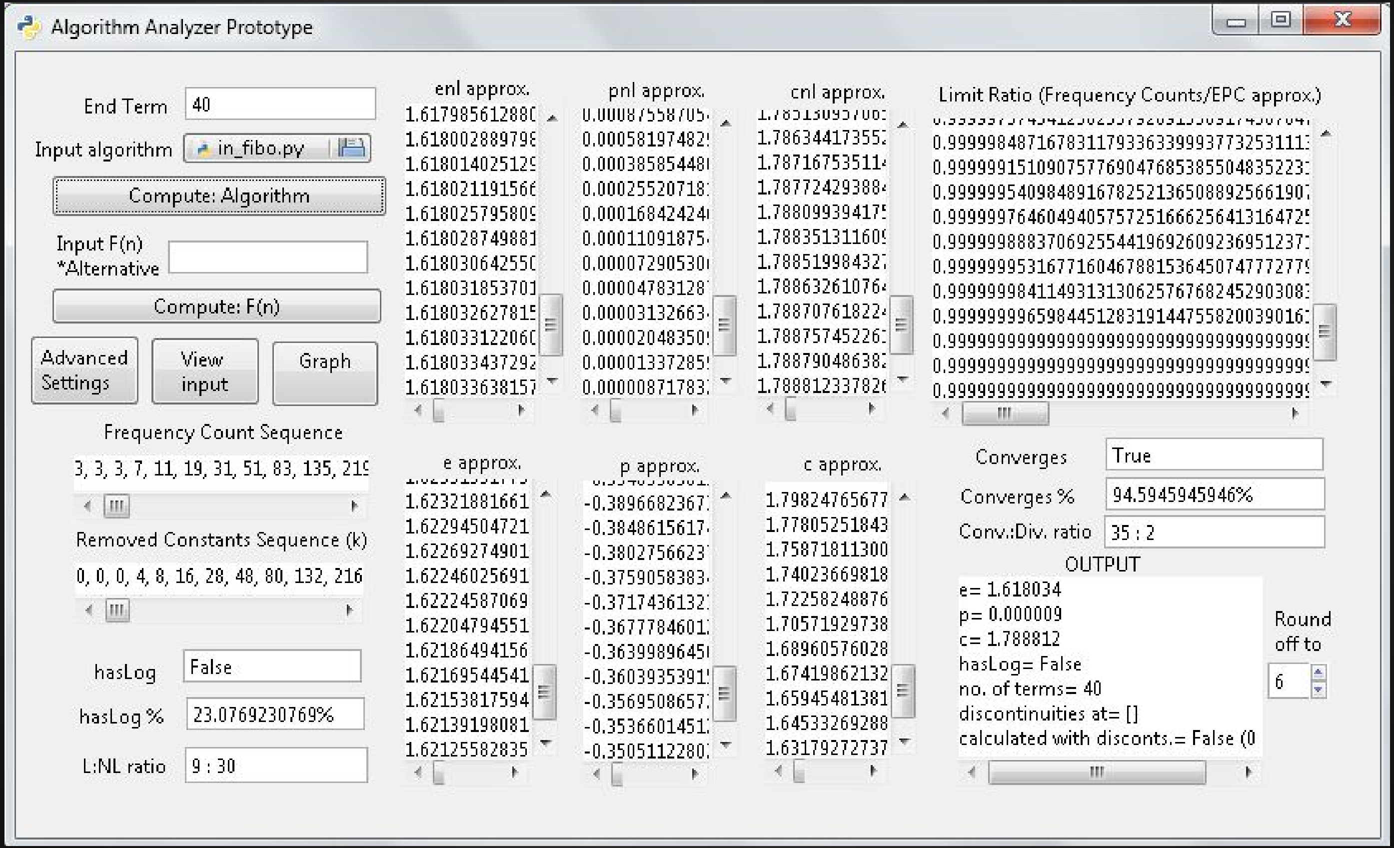
If $|(n+1) * (F_{n+2} - F_{n+1})| \leq |n * (F_{n+1} - F_n)|$

then "Converges" case gains 1 vote

System UI


Algorithm Analyzer Prototype
End Term
50
Input algorithm
(None)
Compute: Algorithm
Input F(n)
*Alternative
Compute: F(n)
Advanced Settings
View input
Graph
Frequency Count Sequence
Removed Constants Sequence (k)
hasLog
hasLog %
L:NL ratio
enl approx.
pnl approx.
cnl approx.
Limit Ratio (Frequency Counts/EPC approx.)
e approx.
p approx.
c approx.
Converges
Converges %
Conv.:Div. ratio
OUTPUT
Round off to
6

System UI



Algorithm Analyzer Prototype
End Term
40
Input algorithm
in_fibo.py
Compute: Algorithm
Input F(n)
*Alternative
Compute: F(n)
Advanced Settings
View input
Graph
Frequency Count Sequence
3, 3, 3, 7, 11, 19, 31, 51, 83, 135, 21
Removed Constants Sequence (k)
0, 0, 0, 4, 8, 16, 28, 48, 80, 132, 216
hasLog
False
hasLog %
23.0769230769%
L:NL ratio
9 : 30
enl approx.
1.617985612880
1.618002889798
1.618014025129
1.618021191566
1.618025795809
1.618028749881
1.618030642550
1.618031853701
1.618032627815
1.618033122060
1.618033437292
1.618033638157
pnl approx.
0.00058197482
0.00038585448
0.00025520718
0.00016842424
0.00011091875
0.00007290530
0.00004783128
0.00003132663
0.00002048350
0.00001337285
0.00000871783
cnl approx.
1.78634417355
1.78716753511
1.78772429388
1.78809939417
1.78835131160
1.78851998432
1.78863261076
1.78870761822
1.78875745226
1.78879048638
1.78881233782
Limit Ratio (Frequency Counts/EPC approx.)
0.99999848716783117933633999377325311
0.99999915109075776904768538550483522
0.99999954098489167825213650889256619
0.99999976460494057572516662564131647
0.99999988837069255441969260923695123
0.99999995316771604678815364507477727
0.99999998411493131306257676824529030
0.99999999659844512831914475582003901
0.9999999999999999999999999999999999
0.9999999999999999999999999999999999
0.9999999999999999999999999999999999
Converges
True
Converges %
94.5945945946%
Conv.:Div. ratio
35 : 2
e approx.
1.62321881661
1.62294504721
1.62269274901
1.62246025691
1.62224587069
1.62204794551
1.62186494156
1.62169544541
1.62153817594
1.62139198081
1.62125582835
p approx.
-0.3896682367
-0.3848615617
-0.3802756623
-0.3759058383
-0.3717436132
-0.3677784601
-0.3639989645
-0.3603935391
-0.3569508657
-0.3536601451
-0.3505112280
c approx.
1.79824765677
1.77805251843
1.75871811300
1.74023669818
1.72258248876
1.70571929738
1.68960576028
1.67419862132
1.65945481381
1.64533269288
1.63179272737
OUTPUT
e= 1.618034
p= 0.000009
c= 1.788812
hasLog= False
no. of terms= 40
discontinuities at= []
calculated with disconts.= False (0
Round off to
6

Expected vs Actual Results

```
def f(n):
    if(n<=2):
        return 1
    else:
        return f(n-1) + f(n-2)
```

Algorithm	Expected Algorithm Output	Actual Algorithm Output
Fibo(n)	e = 1.618034 p = 0 c = 1.788854 hasLog = False	e = 1.618034 p = 0.000000 c = 1.788854 hasLog = False (Using 50 terms)

Expected vs Actual Results

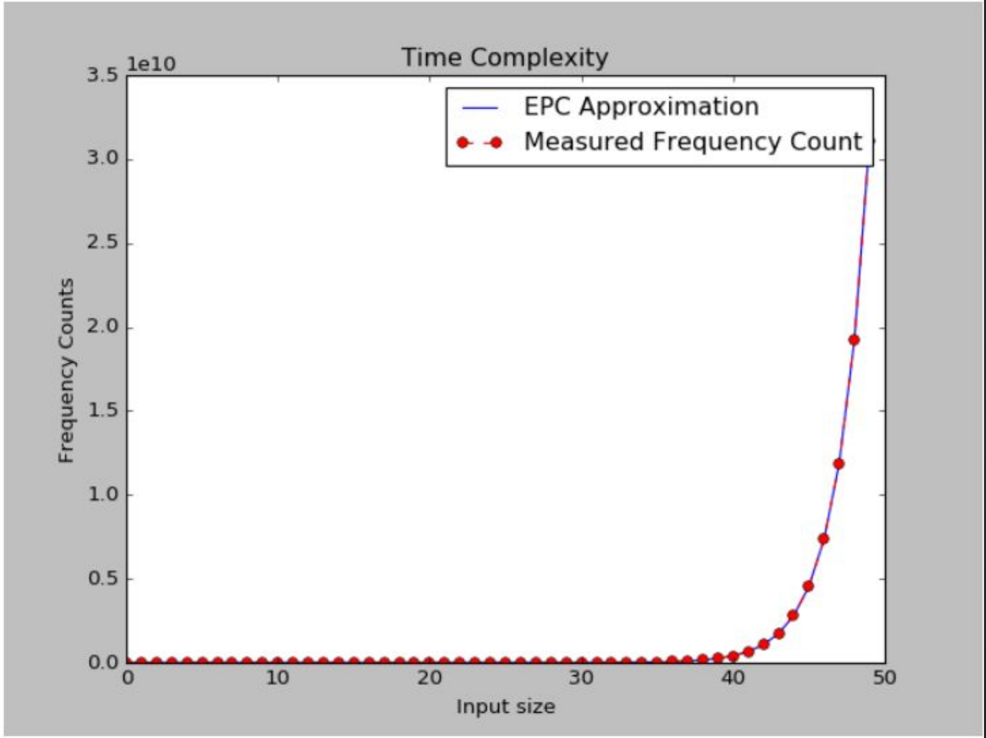


Figure F.11: Graph of Fibo(n)

Expected vs Actual Results

```
def f(n):
    if(n<=2):
        return 1
    else:
        return 1+f(n/2.0)
```

Algorithm	Expected Algorithm Output	Actual Algorithm Output
Log2(n)	e = 1 p = 0 c = 2.885390 hasLog = True	e = 1.000000 p = 0.000035 c = 2.884487 hasLog = True (Using 20000 terms)

Expected vs Actual Results

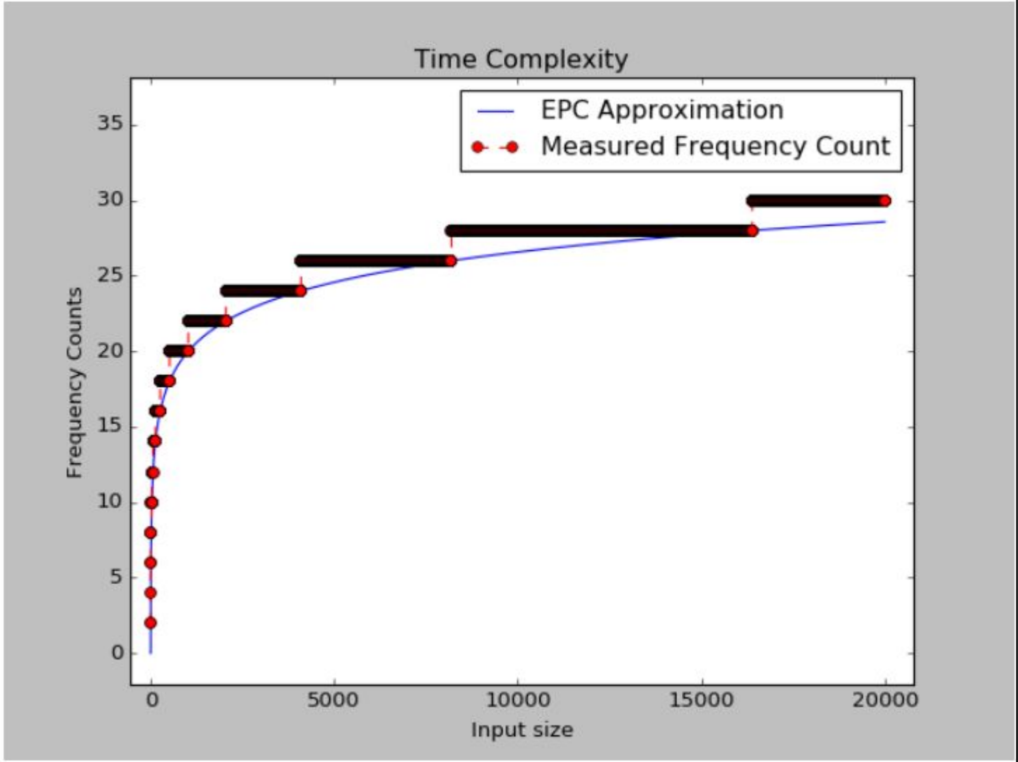


Figure F.10: Graph of Log2(n)

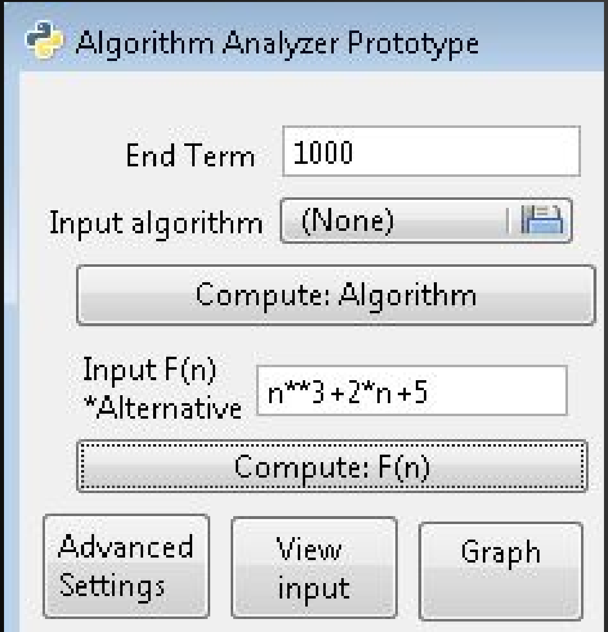
Expected vs Actual Results

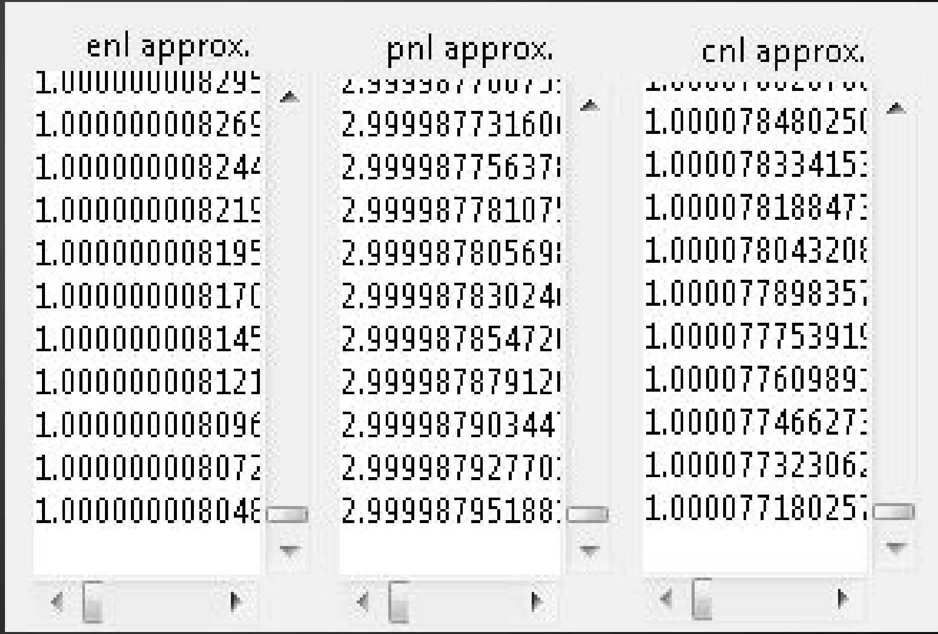
```
def f(n):
    if n==0:
        return 1
    else:
        for i in range(0, n):
            f(n-1)
```

Algorithm	Expected Algorithm Output	Actual Algorithm Output
FactLike(n)	e = divergent p = divergent c = divergent hasLog = False	Converges = False (Using 30 terms)

Resulting Approximations

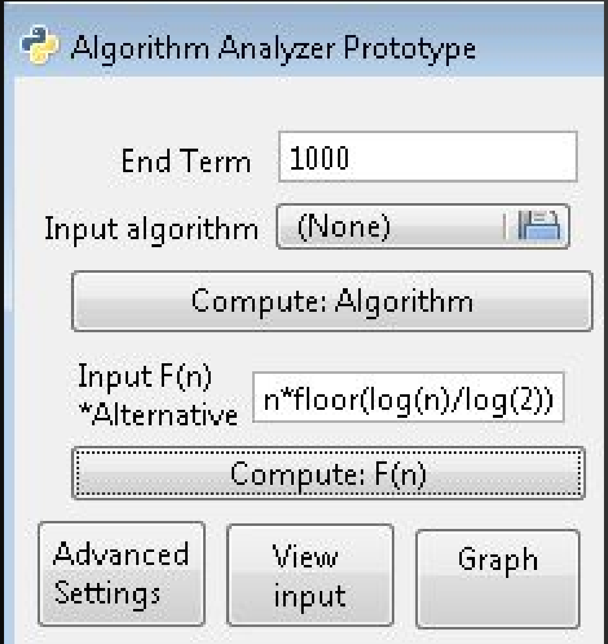
- Convergent Continuous Approximation

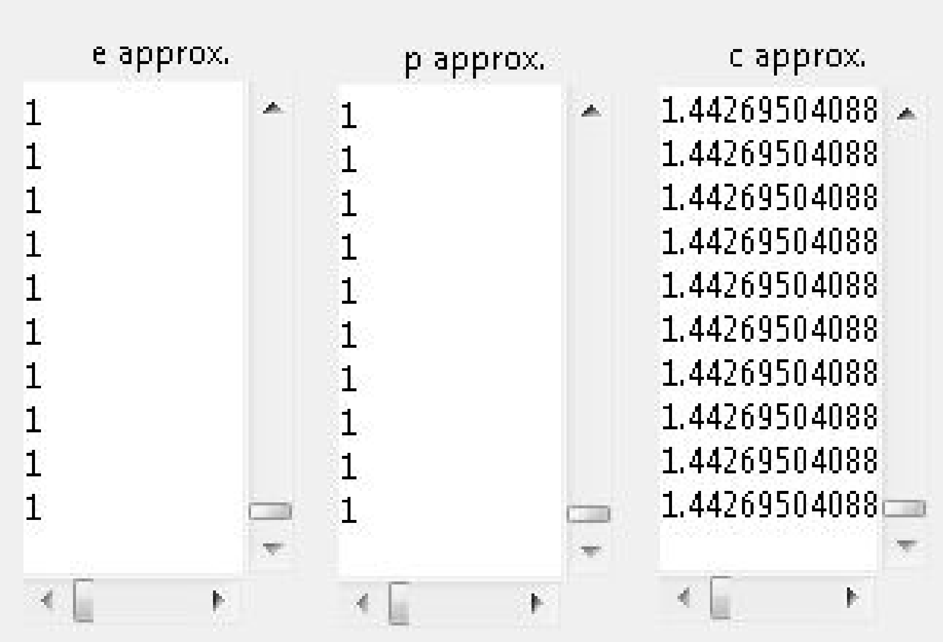




Resulting Approximations

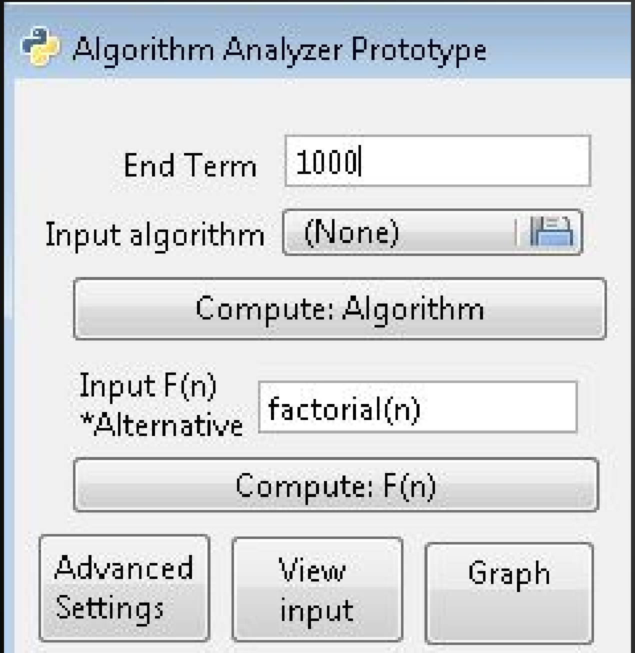
- Convergent Discontinuous Approximation

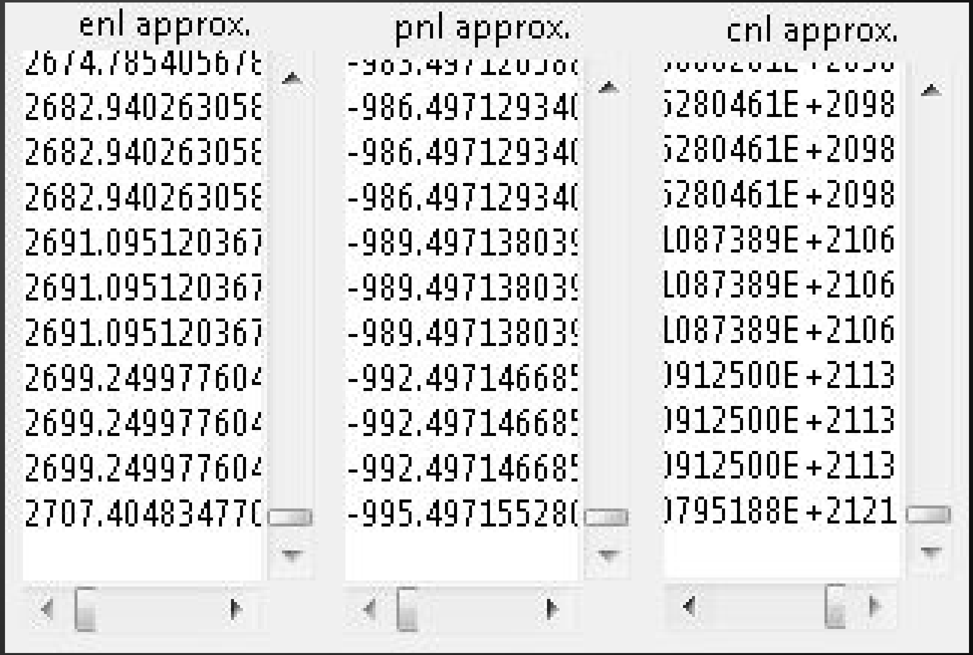




Resulting Approximations

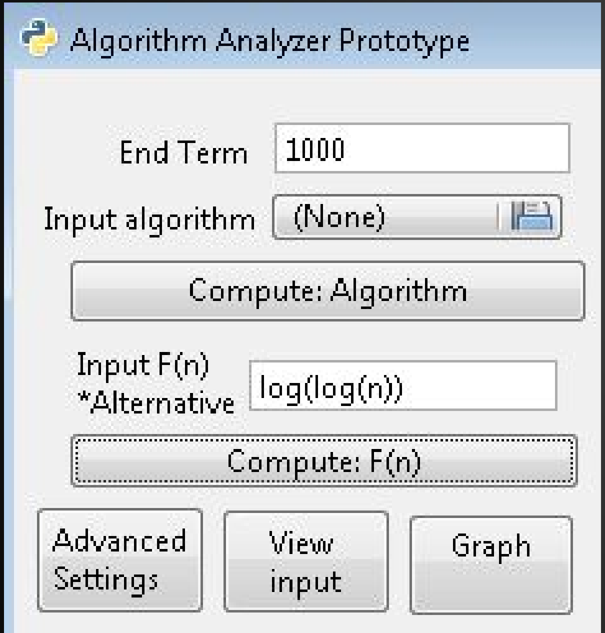
- Divergent Approximation

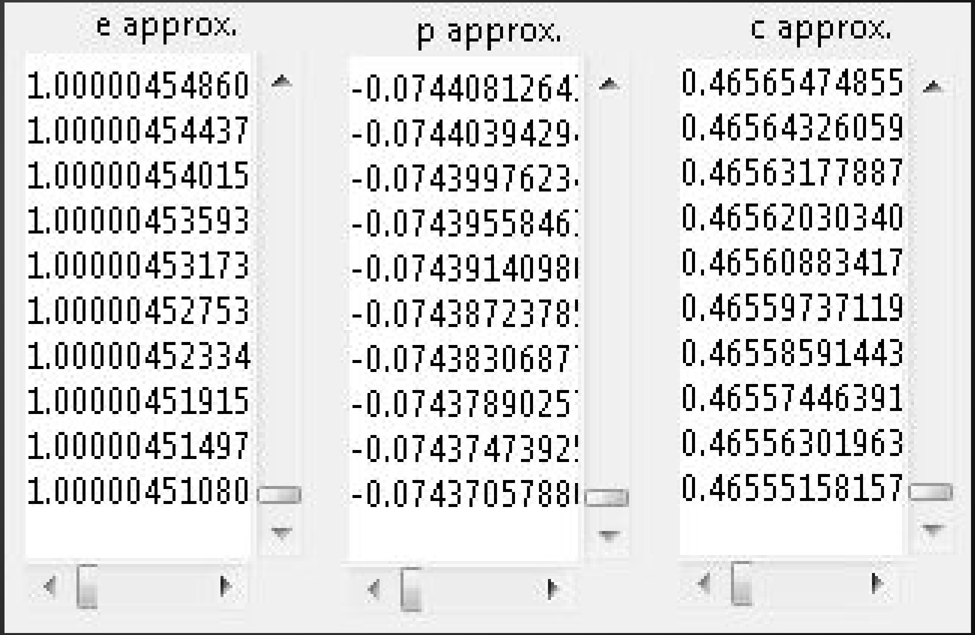




Resulting Approximations

- Slowly Divergent Approximation (false positive)





Questions & Answers

References

[1] Abel, N. H. (1824). Mmoire sur les quations algbriques, ou l'on dmontre l'impossibilit de la rsolution de l'quation gnrale du cinquime degr.

[2] Ackermann, W. (1928). Zum hilbertschen aufbau der reellen zahlen. Mathematische Annalen, 99, 118-133.

[3] Cormen, T., et al. (2009). Introduction to Algorithms. 3rd edition. The MIT Press.

[4] Fradkin, L. (2013). Elementary Algebra and Calculus: The Whys and Hows. Bookboon.

[5] Gossett, E. (2009). Discrete mathematics with Proof. 2nd edition. A John Wiley and Sons, Inc. Publication

[6] Greene, D., et al. (1982). Mathematics for the analysis of algorithms. 2nd edition. Birkhuser.

[7] Knuth, D. (1976). Mathematics and computer science: Coping with finiteness. Science, 194(17) , 1235-1242.

[8] McConnell, J. (2008). Analysis of algorithms. Jones and Bartlett Publishers Inc.

[9] McGeoch, C., et al. (2002). Using Finite experiments to study asymptotic performance. In From algorithm design to robust and efficient software. Springer Berlin Heidelberg

[10] Montesinos, V., et al. (2015). An introduction to modern analysis. Springer.

[11] Pai, G. (2008). Data structures and algorithms:concepts, techniques and applications. Tata McGraw-Hill Education Pvt. Ltd.

[12] Profiling of Algorithms. (n.d.). Retrieved from https://www8.cs.umu.se/ kurser/TDBC91/H99/Slides/profiling.pdf (Accessed on August 2015)

[13] Rabinovich, S., et al. (1996). Solving nonlinear recursions. Journal of Mathematical Physics, 37(11), 5828-5836.

[14] Sedgewick, R., etal (2013). Introduction to the Analysis of Algorithms, 2nd edition. Addison-Wesley Professional.

[15] Wolfram alpha. (n.d.). Retrieved from http://www.wolframalpha.com/input/ ?i=x\%5E5-2x\%5E3-2x\%5E2-2x\%5E1-1\%3D0 (Accessed on February 2015)

Posteriori Sample - Profilers

1. Pre-Processing (Augmentation)
2. Processing (Running)
3. Post-Processing (Analysis)

main		935 ms (100%)	1
Main.main (String[])		935 ms (100%)	1
Main.parseNextTestCase ()		924 ms (98.9%)	2002
Main.buildStrings (boolean[], int)		393 ms (42.1%)	867266
Self time		283 ms (30.4%)	2002
Main.write (Object)		66.7 ms (7.1%)	17763
Main.writeln (Object)		62.4 ms (6.7%)	3998
Main.buildBits (int, int)		44.9 ms (4.8%)	4
Main.checkParams ()		40.1 ms (4.3%)	2001
Main.writeln ()		13.0 ms (1.4%)	837
Main.ready ()		7.94 ms (0.8%)	2002
Main.isAnagram ()		4.81 ms (0.5%)	2000
java.util.HashMap.get (Object)		3.41 ms (0.4%)	1999
java.lang.String.toCharArray ()		2.13 ms (0.2%)	4000
java.lang.Boolean.valueOf (boolean)		0.655 ms (0.1%)	2002
java.lang.Integer.valueOf (int)		0.419 ms (0%)	2003
java.lang.ClassLoader.loadClass (String)		0.135 ms (0%)	2
java.util.HashMap.put (Object, Object)		0.076 ms (0%)	6
java.lang.System.getSecurityManager ()		0.000 ms (0%)	2

http://www.redgreencode.com/wp-content/uploads/2015/07/Optimized.png

Apriori Sample - Iterative Method

1. Expanding any iterations or recursions present
2. Finding a pattern or a series
3. Generalization of the pattern

$$T(n) = T(n-1) + n \text{ , } T(0) = c$$

$$T(n) = T(n-2) + n-1 + n$$

$$T(n) = T(n-3) + n-2 + n-1 +n$$

$$T(n) = T(n-i) + i*n - (0+1+2+...+i-1)$$

$$\therefore \; T(n) = c + n^2 - (n-1)*n / 2$$