An Algorithm and Estimates for the Erdős-Selfridge Function

Brianna Sorenson[1], Jon Sorenson[2], and Jonathan Webster[2]

Butler University, USA
bsorenso@butler.edu, sorenson@butler.edu, jewebste@butler.edu

1. Supported by the Butler Summer Institute
2. Supported by a grant from the Holcomb Awards Committee

The Erdős-Selfridge Function

Definition:
Let $lpf(n)$ be the least prime factor of the integer n.

Then $g(k)$ is the smallest integer, with $g(k) > k+1$, such that

$$lpf\left(\binom{g(k)}{k}\right) > k.$$

Why?

- Paul Erdős posed the $g(k)$ problem in 1969.
- It's in the Online Encyclopedia of Integer Sequences: OEIS A003458

Examples of $g(k)$ using Pascal's Triangle

$k =$	0	1	2	3	4	5	6	7
	1							
	1	1						
	1	2	1					
	1	**3**	3	1				
	1	4	6	4	1			
	1	5	10	10	5	1		
	1	6	15	20	15	6	1	
	1	7	21	35	35	21	7	1

$g(1) = 3$ since $3 > 1 + 1$ and $\binom{3}{1} = 3$ with $lpf(3) = 3 > 1$.

Examples of $g(k)$ using Pascal's Triangle

$k =$	0	1	2	3	4	5	6	7
	1							
	1	1						
	1	2	1					
	1	3	3	1				
	1	4	6	4	1			
	1	5	10	10	5	1		
	1	6	**15**	20	15	6	1	
	1	7	21	35	35	21	7	1

$g(1) = 3$ since $3 > 1 + 1$ and $\binom{3}{1} = 3$ with $lpf(3) = 3 > 1$.

$g(2) = 6$ since $6 > 2 + 1$ and $\binom{6}{2} = 15$ with $lpf(15) = 3 > 2$.

Examples of $g(k)$ using Pascal's Triangle

$k =$	0	1	2	3	4	5	6	7
	1							
	1	1						
	1	2	1					
	1	3	3	1				
	1	4	6	4	1			
	1	5	10	10	5	1		
	1	6	15	20	15	6	1	
	1	7	21	**35**	35	21	7	1

$g(1) = 3$ since $3 > 1 + 1$ and $\binom{3}{1} = 3$ with $lpf(3) = 3 > 1$.

$g(2) = 6$ since $6 > 2 + 1$ and $\binom{6}{2} = 15$ with $lpf(15) = 3 > 2$.

$g(3) = 7$ since $7 > 3 + 1$ and $\binom{7}{3} = 35$ with $lpf(35) = 5 > 3$.

Examples of $g(k)$ using Pascal's Triangle

$k=$	0	1	2	3	4	5	6	7
	1							
	1	1						
	1	2	1					
	1	3	3	1				
	1	4	6	4	1			
	1	5	10	10	5	1		
	1	6	15	20	15	6	1	
	1	7	21	35	**35**	21	7	1

$g(1) = 3$ since $3 > 1 + 1$ and $\binom{3}{1} = 3$ with $lpf(3) = 3 > 1$.

$g(2) = 6$ since $6 > 2 + 1$ and $\binom{6}{2} = 15$ with $lpf(15) = 3 > 2$.

$g(3) = 7$ since $7 > 3 + 1$ and $\binom{7}{3} = 35$ with $lpf(35) = 5 > 3$.

$g(4) = 7$ since $7 > 4 + 1$ and $\binom{7}{4} = 35$ with $lpf(35) = 5 > 4$.

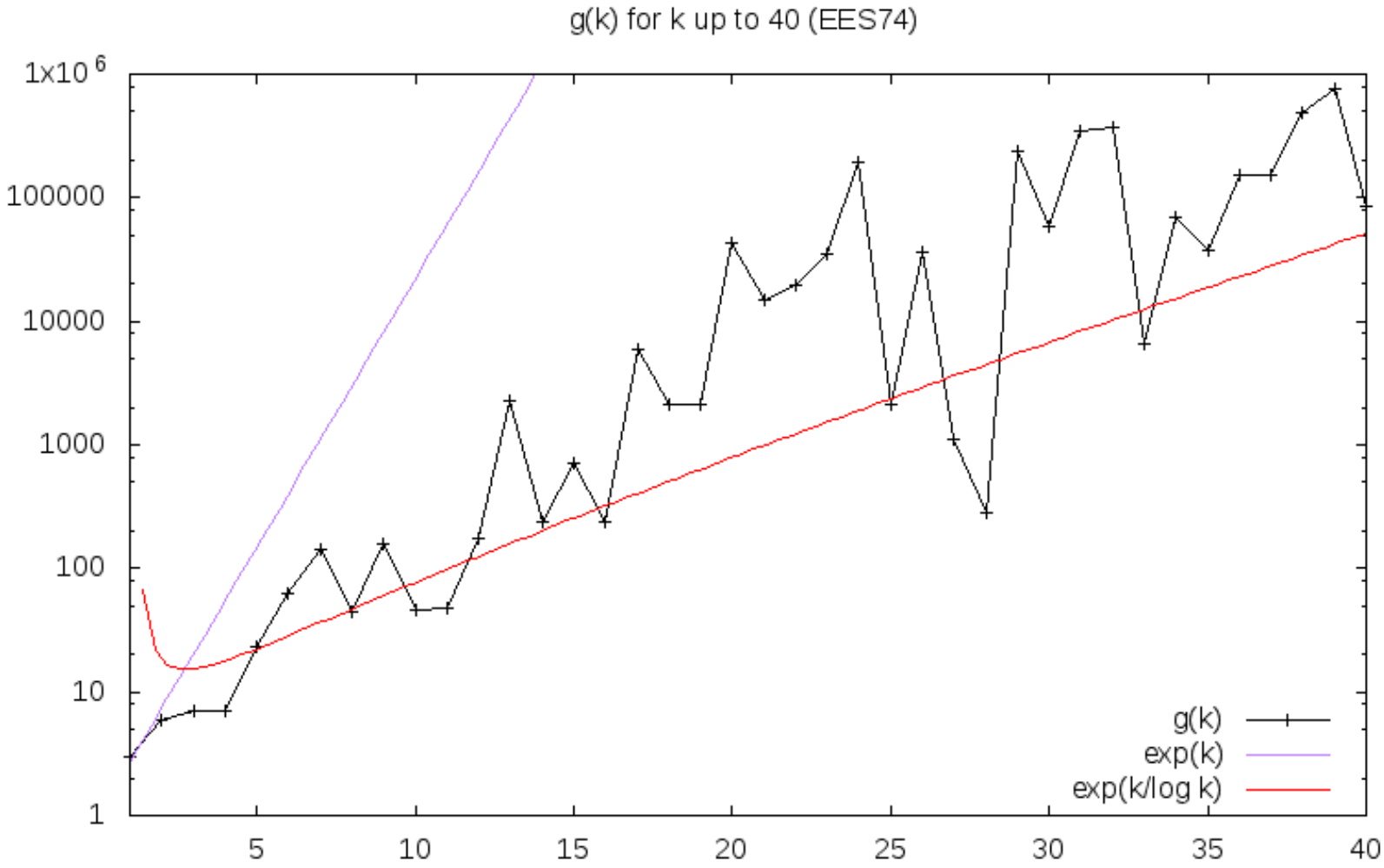
Computational Results: Ecklund, Erdős, Selfridge (1974)

TABLE 1. *Values of* $g(k) \leqq 2500000$ *for* $2 \leqq k \leqq 100$

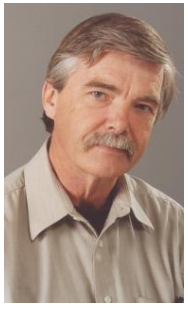
k	$g(k)$	k	$g(k)$	k	$g(k)$	k	$g(k)$	k	$g(k)$
		11	47	21	14871	31	341087	41	*B*
2	6	12	174	22	19574	32	371942	42	96622
3	7	13	2239	23	35423	33	6459	43/	*B*
4	7	14	239	24	193049	34	69614	45	
5	23	15	719	25	2105	35	37619	46	692222
6	62	16	241	26	36287	36	152188	47/	*B*
7	143	17	5849	27	1119	37	152189	51	
8	44	18	2098	28	284	38	487343	52	366847
9	159	19	2099	29	240479	39	767919	53/	*B*
10	46	20	43196	30	58782	40	85741	100	

B: $g(k)$ *exceeds the search bound of* 2500000

$g(k)$ is troublesome



More Computations

Scheidler and Williams (1992) used Kummer's theorem to construct a sieve algorithm to compute $g(k)$ for all $k \leq 140$.

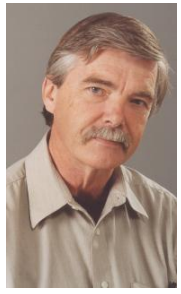
Typo:
$g(114) = 59819\ 90286\ 02614$.

TABLE 1

k	$g(k)$	k	$g(k)$	k	$g(k)$
2	6	49	38074099	96	5589371247
3	7	50	4302206	97	104141995747
4	7	51	13927679	98	10628330723
5	23	52	366847	99	5675499
6	62	53	79221239	100	3935600486
7	143	54	7638454	101	2128236159983
8	44	55	53583095	102	175209712494
9	159	56	17868986	103	5092910127863
10	46	57	34296443	104	6003175578749
11	47	58	4703099	105	4753399456493
12	174	59	108178559	106	488898352367
13	2239	60	93851196	107	6260627365739
14	239	61	2237874623	108	9746385386989
15	719	62	254322494	109	73245091349869
16	241	63	157776319	110	94794806842238
17	5849	64	266194499	111	222261611307119
18	2098	65	174133871	112	90200708362489
19	2099	66	25013442	113	517968108138869
20	43196	67	673750867	114	517968108138869
21	14871	68	643364693	115	12714356616655615
22	19574	69	237484869	116	4112143718554871
23	35423	70	549177974	117	10584753118053749
24	193049	71	3184709471	118	3781786358757119
25	2105	72	4179979724	119	598228285941119
26	36287	73	15780276223	120	260509131365372
27	1119	74	19942847999	121	404087677322873
28	284	75	48899668971	122	115598852533247
29	240479	76	16360062718	123	71406652074623
30	58782	77	2198202863	124	28204866143999
31	341087	78	950337359	125	3988617067133
32	371942	79	29154401359	126	5614007242751
33	6459	80	43228410965	127	60503616486143
34	69614	81	6599930719	128	14320632355808
35	37619	82	1101163607	129	38423911578259
36	152188	83	797012560343	130	7984603413422
37	152189	84	95695473244	131	3249072073157063
38	487343	85	449488751711	132	96965971239157
39	767919	86	328151678711	133	1558724612351669
40	85741	87	39419852119	134	621248003653094
41	3017321	88	94923115999	135	3157756005623
42	96622	89	3524996442239	136	4138898693368
43	24041599	90	2487760912090	137	951598054985213
44	45043199	91	739416801247	138	745504491090939
45	9484095	92	2380889434844	139	25972027636644319
46	692222	93	577593151999	140	9089854222866845
47	232906799	94	107706126974		
48	45375224	95	71573860223		

Even More Computations

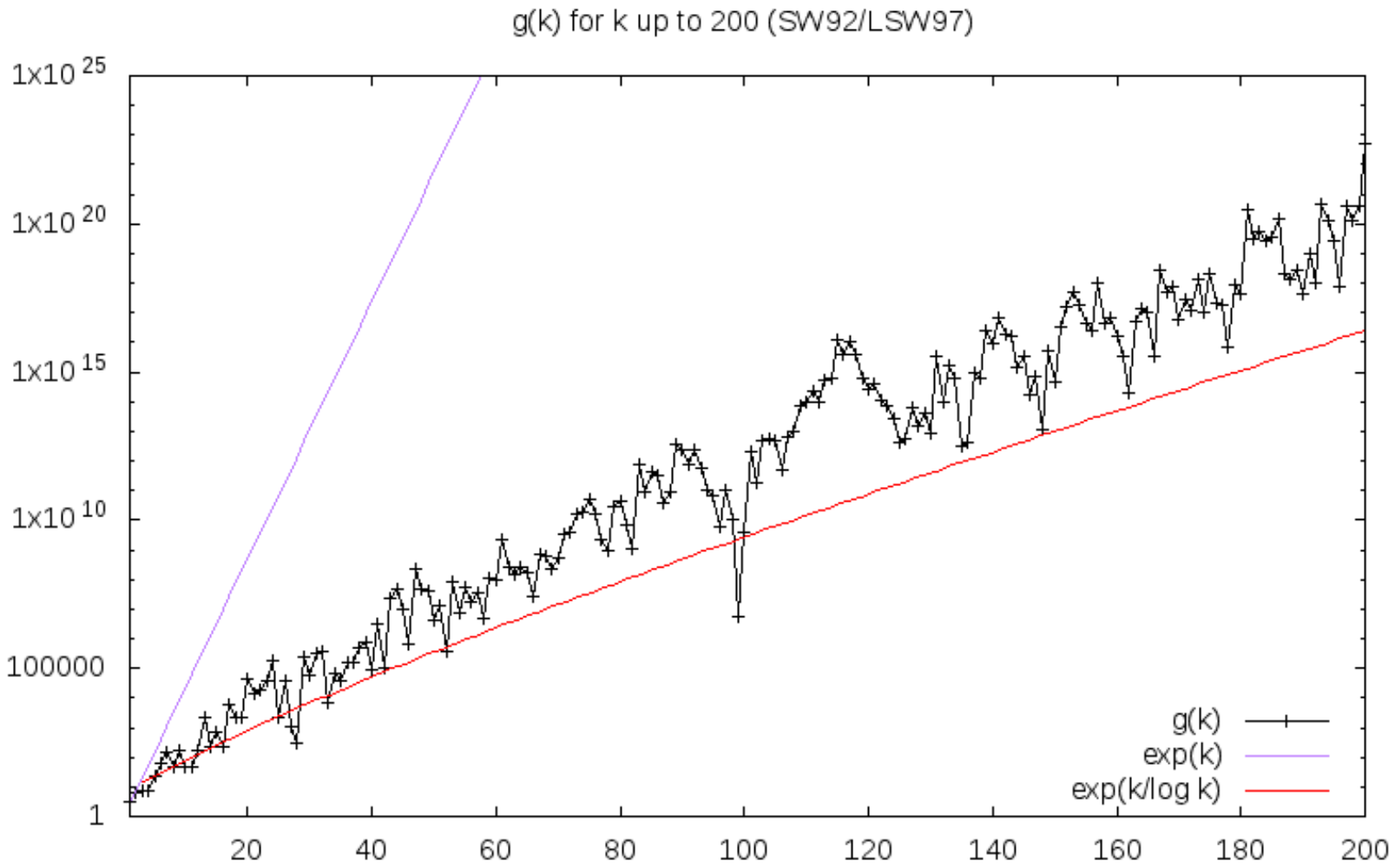




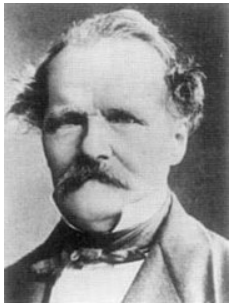
Lukes, Scheidler, and Williams (1997) improved their sieve, used special-purpose hardware, and computed $g(k)$ for all $k \leq 200$.

k	$g(k)$
135	315 7756005623
136	413 8898693368
137	95159 8054985213
138	60124 2167764223
139	2597202 7636644319
140	908985 4222866845
141	6333152 3816662671
142	1990465 6320115423
143	1542289 5461804543
144	139719 3586455769
145	339515 6674599871
146	17509 5016485374
147	72531 1731192223
148	1180 8400809148
149	542394 5342959799
150	47313 8520098551
151	3258989 9217872863
152	15549796 7465547419
153	53518499 5256751839
154	17864690 7528990874
155	4129798 4000013467
156	2527233 4970944959
157	104130829 7102375167
158	4702566 0758882783
159	6485551 8266246559
160	1664745 6280932287
161	342159 0108339941
162	20212 9337635322
163	5188127 2225707439
164	14664726 1829992439
165	9865227 4401898671
166	347584 7868933047
167	277308556 4165092343
168	48669223 2365306798
169	72698097 9380669099

k	$g(k)$
170	5942841 5007516671
171	28496594 9074228671
172	11223206 5794463997
173	138175311 6390427373
174	11057733 6695616174
175	19440921 9 4361247743
176	2262553 0 3912072703
177	1924652 3 8561441207
178	68448 0 9280136434
179	9078741 9 7930300859
180	4397630 1 6255983614
181	2 8336150170 1232528573
182	289888386 3 4918997183
183	53516247 0 5 6143575999
184	26626878 4 4 8827721469
185	37136036 5 5 0263266493
186	1 4274157 9 9 4 6200597438
187	2208849 9 1 2824359867
188	1271981 0 6 5611178943
189	2956298 0 5 3153332989
190	4556522 3 2192890367
191	9396883 2 1 4719852991
192	1063650 7 2 4436901873
193	4 3525141 9 7 2 8230720249
194	1 2638743 5 8 8 3753706219
195	2632591 2 1 6 1870817495
196	7815166 6 4 2 1536537 3
197	4 2796097 7 1 2 6350089949
198	1 3539364 6 0 3 6546861 9 8
199	4 0316886 8 8 6 7096129999
200	520 8783889 2 7 1 0191382732


g(k) for k up to 200 (SW92/LSW97)
1x10 25
1x10 20
1x10 15
1x10 10
100000
1
20
40
60
80
100
120
140
160
180
200
g(k)
exp(k)
exp(k/log k)

Kummer's Theorem



Theorem (Kummer).

Let $k < n$ be positive integers, and let p be a prime $\leq k$.
Let $t \geq \lfloor \log_p k \rfloor + 1$ be an integer and write

$$k = \sum_{i=0}^{t} a_i p^i \quad \text{and} \quad n = \sum_{i=0}^{t} b_i p^i$$

as the base-p representations of k and n, respectively.
Then p does not divide $\binom{n}{k}$ iff $b_i \geq a_i$ for all $i \leq t$.

Kummer's Theorem - Examples

$k = 10, n = 12, p = 5$:

$k = 20_5$, $n = 22_5$ so 5 does not divide $\binom{12}{10} = 66$.

Kummer's Theorem - Examples

$k = 10, n = 12, p = 5$:
$k = 20_5$, $n = 22_5$ so 5 does not divide $\binom{12}{10} = 66$.

$k = 10, n = 12, p = 3$:
$k = 101_3$, $n = 110_3$ so 3 does divide $\binom{12}{10} = 66$.

Kummer's Theorem - Examples

$k = 10, n = 12, p = 5$:
$k = 20_5$, $n = 22_5$ so 5 does not divide $\binom{12}{10} = 66$.

$k = 10, n = 12, p = 3$:
$k = 101_3$, $n = 110_3$ so 3 does divide $\binom{12}{10} = 66$.

Idea:
For each $p \leq k$, we get a list of *acceptable residues* modulo $p^{\lfloor \log_p k \rfloor + 1}$.
Combine these residues modulo

$$M_k := \prod_{p \leq k} p^{\lfloor \log_p k \rfloor + 1}$$

using the Chinese remainder theorem to search for the solution.

Kummer's Theorem - Examples

$k = 10, n = 12, p = 5$:
$k = 20_5$, $n = 22_5$ so 5 does not divide $\binom{12}{10} = 66$.

$k = 10, n = 12, p = 3$:
$k = 101_3$, $n = 110_3$ so 3 does divide $\binom{12}{10} = 66$.

Idea:
For each $p \leq k$, we get a list of *acceptable residues* modulo $p^{\lfloor \log_p k \rfloor + 1}$.
Combine these residues modulo

$$M_k := \prod_{p \leq k} p^{\lfloor \log_p k \rfloor + 1}$$

using the Chinese remainder theorem to search for the solution.
$g(k)$ is the smallest acceptable residue modulo M_k with $g(k) > k + 1$.

Example with $k = 10$

$p = 2$, $k = 1010_2$:
1010_2, 1011_2, 1110_2, 1111_2
(4 residues mod $2^4 = 16$)

$p = 3$, $k = 101_3$:
101_3, 102_3, 111_3, 112_3, 121_3, 122_3, 201_3, 202_3, 211_3, 212_3, 221_3, 222_3
(12 residues mod $3^3 = 27$)

$p = 5$, $k = 20_5$:
20_5, 21_5, 22_5, 23_5, 24_5, 30_5, 31_5, 32_5, 33_5, 34_5, 40_5, 41_5, 42_5, 43_5, 44_5
(15 residues mod $5^2 = 25$)

$p = 7$, $k = 13_7$:
13_7, 14_7, 15_7, 16_7, 23_7, 24_7, 25_7, 26_7, 33_7, 34_7, 35_7, 36_7, 43_7, 44_7, 45_7,
46_7, 53_7, 54_7, 55_7, 56_7, 63_7, 64_7, 65_7, 66_7
(24 residues mod $7^2 = 49$)

Algorithm Example with $k = 10$

$M_{10} = 16 \cdot 27 \cdot 25 \cdot 49 = 529200$.
$R_{10} := 4 \cdot 12 \cdot 15 \cdot 24 = 17280$ is *too many* residues to check.

Algorithm Example with $k = 10$

$M_{10} = 16 \cdot 27 \cdot 25 \cdot 49 = 529200$.
$R_{10} := 4 \cdot 12 \cdot 15 \cdot 24 = 17280$ is *too many* residues to check.
We estimate that $g(10) \leq 306.27$. (roughly $k \cdot M_k / R_k$)

Algorithm Example with $k = 10$

$M_{10} = 16 \cdot 27 \cdot 25 \cdot 49 = 529200$.
$R_{10} := 4 \cdot 12 \cdot 15 \cdot 24 = 17280$ is *too many* residues to check.
We estimate that $g(10) \leq 306.27$. (roughly $k \cdot M_k / R_k$)
We will look modulo $N = 2^4 \cdot 3 \cdot 7 = 336$, which divides M_k, and search among the residues modulo 336 instead:

10, 11, 14, 15 modulo 16
1, 2 modulo 3
3, 4, 5, 6 modulo 7

For a total of only 32 residues to check.

Algorithm Example with $k = 10$

$M_{10} = 16 \cdot 27 \cdot 25 \cdot 49 = 529200$.
$R_{10} := 4 \cdot 12 \cdot 15 \cdot 24 = 17280$ is *too many* residues to check.
We estimate that $g(10) \leq 306.27$. (roughly $k \cdot M_k / R_k$)
We will look modulo $N = 2^4 \cdot 3 \cdot 7 = 336$, which divides M_k, and search among the residues modulo 336 instead:

10, 11, 14, 15 modulo 16
1, 2 modulo 3
3, 4, 5, 6 modulo 7

For a total of only 32 residues to check.
We use a wheel data structure to do this efficiently.

Algorithm Example with $k = 10$

$M_{10} = 16 \cdot 27 \cdot 25 \cdot 49 = 529200$.
$R_{10} := 4 \cdot 12 \cdot 15 \cdot 24 = 17280$ is *too many* residues to check.
We estimate that $g(10) \leq 306.27$. (roughly $k \cdot M_k / R_k$)
We will look modulo $N = 2^4 \cdot 3 \cdot 7 = 336$, which divides M_k, and search among the residues modulo 336 instead:

10, 11, 14, 15 modulo 16
1, 2 modulo 3
3, 4, 5, 6 modulo 7

For a total of only 32 residues to check.
We use a wheel data structure to do this efficiently.

How did we pick $N = 336$?

Knapsack

- Choosing prime rings to include in N is a variation of the *knapsack problem*, which is NP-complete.

Knapsack

- Choosing prime rings to include in N is a variation of the *knapsack problem*, which is NP-complete.
- For a ring with modulus p^e and $r(p, e)$ residues,

$$\textit{Ring Size} := \log(p^e) = e \log p.$$
$$\textit{Ring Value} := \log(p^e / r(p, e)) = e \log p - \log r(p, e).$$

Knapsack

- Choosing prime rings to include in N is a variation of the *knapsack problem*, which is NP-complete.
- For a ring with modulus p^e and $r(p, e)$ residues,

$$\begin{aligned} \textit{Ring Size} &:= \log(p^e) = e \log p. \\ \textit{Ring Value} &:= \log(p^e / r(p, e)) = e \log p - \log r(p, e). \end{aligned}$$

- These definitions lead to minimizing the number of residues to check modulo N.

Knapsack

- Choosing prime rings to include in N is a variation of the *knapsack problem*, which is NP-complete.
- For a ring with modulus p^e and $r(p, e)$ residues,

$$\textit{Ring Size} := \log(p^e) = e \log p.$$
$$\textit{Ring Value} := \log(p^e / r(p, e)) = e \log p - \log r(p, e).$$

- These definitions lead to minimizing the number of residues to check modulo N.
- *Prime splitting* is choosing the best e for each p to maximize the Value/Size ratio.

Knapsack

- Choosing prime rings to include in N is a variation of the *knapsack problem*, which is NP-complete.
- For a ring with modulus p^e and $r(p, e)$ residues,

$$\begin{aligned} \textit{Ring Size} &:= \log(p^e) = e \log p. \\ \textit{Ring Value} &:= \log(p^e / r(p, e)) = e \log p - \log r(p, e). \end{aligned}$$

- These definitions lead to minimizing the number of residues to check modulo N.
- *Prime splitting* is choosing the best e for each p to maximize the Value/Size ratio.
- Our greedy knapsack algorithm picks the rings of highest value such that the sum of the sizes is roughly $\log N$. For $k = 10$ we'd aim for $\log N$ near $\log 307$.

Knapsack Example with $k = 10$

p	e	$\lfloor \log_p k \rfloor + 1$	$r(p, e)$	*filter rate*	*k in base p*
2	*4*	*4*	*4*	*0.25*	1010_2
3	*1*	*3*	*2*	*0.666667*	101_3
7	*1*	*2*	*4*	*0.571429*	13_7
3	*3*	*3*	*6(12)*	*0.666667*	
5	*2*	*2*	*15*	*0.6*	20_5
7	*2*	*2*	*6(24)*	*0.857143*	

We ran an algorithm to choose an optimal splitting point (e) for each prime. Both rings are included as options for the knapsack algorithm.

Knapsack Example with $k = 280$

p	e	$\lfloor \log_p k \rfloor + 1$	$r(p, e)$	*filter rate*	*k in base p*
47	*1*	*2*	*2*	*0.0425532*	$5 : 45_{47}$
71	*1*	*2*	*4*	*0.056338*	$3 : 67_{71}$
17	*2*	*2*	*9*	*0.0311419*	$16 : 8_{17}$
41	*1*	*2*	*7*	*0.170732*	$6 : 34_{41}$
97	*1*	*2*	*11*	*0.113402*	$2 : 86_{97}$
19	*2*	*2*	*25*	*0.0692521*	$14 : 14_{19}$
7	*3*	*3*	*28*	*0.0816327*	550_7
149	*1*	*2*	*18*	*0.120805*	$1 : 131_{149}$
73	*1*	*2*	*12*	*0.164384*	$3 : 61_{73}$
2	*5*	*9*	*8*	*0.25*	100011000_2
151	*1*	*2*	*22*	*0.145695*	$1 : 129_{151}$
3	*1*	*6*	*2*	*0.666667*	101101_3
13	*2*	*3*	*30*	*0.177515*	187_{13}
59	*1*	*2*	*15*	*0.254237*	$4 : 44_{59}$

The Wheel Data Structure, $k = 10$

Ring 16:

residue	0	1	2	3	4	5	6	7
admissible	0	0	0	0	0	0	0	0
jump	+10	+9	+8	+7	+6	+5	+4	+3

residue	8	9	10	11	12	13	14	15
admissible	0	0	1	1	0	0	1	1
jump	+2	+1	+1	+3	+2	+1	+1	+11

Ring 3:

residue	0	1	2
admissible	0	1	1
jump	+16	+16	+32

Ring 7:

residue	0	1	2	3	4	5	6
admissible	0	0	0	1	1	1	1
jump	+48	+96	+144	+192	+48	+48	+48

The Wheel Data Structure Example, $k = 10$

16	3	7	Filters
$k + 2 = 12 \to 14$	$14 \to 14$	$14 \to 62$	Fails mod 27
		$+48 = 110$	Fails mod 27
		$+48 = 158$	Fails mod 25
		$+48 = 206$	Fails mod 25
	$+32 = 46$	$46 \to 46$	Passes! $g(10) \leq 46$
		$+48 = 94$	Don't bother
		$+192 = 286$	Don't bother
		$+48 = 334$	Don't bother
$+1 = 15$	$15 \to 31$	$31 \to 31$	...

Algorithm Running Time

Theorem.

Under the assumption of the *Uniform Distribution Heuristic* (see the next slide) our new algorithm has a running time of

$$g(k) \exp\left[-\frac{c \cdot k \log\log k}{(\log k)^2} \cdot (1 + o(1))\right]$$

arithmetic operations, for a constant $c > 0$.

It's possible to prove a sublinear running time with no assumptions.

Approximating $g(k)$

We have $M_k = \prod_{p \leq k} p^{\lceil \log_p k \rceil + 1}$.
R_k counts the acceptable residues, modulo M_k.
$\hat{g}(k) := M_k / R_k$.

Uniform Distribution Heuristic (UDH)

The R_k acceptable residues are uniformly distributed modulo M_k.

Theorem.

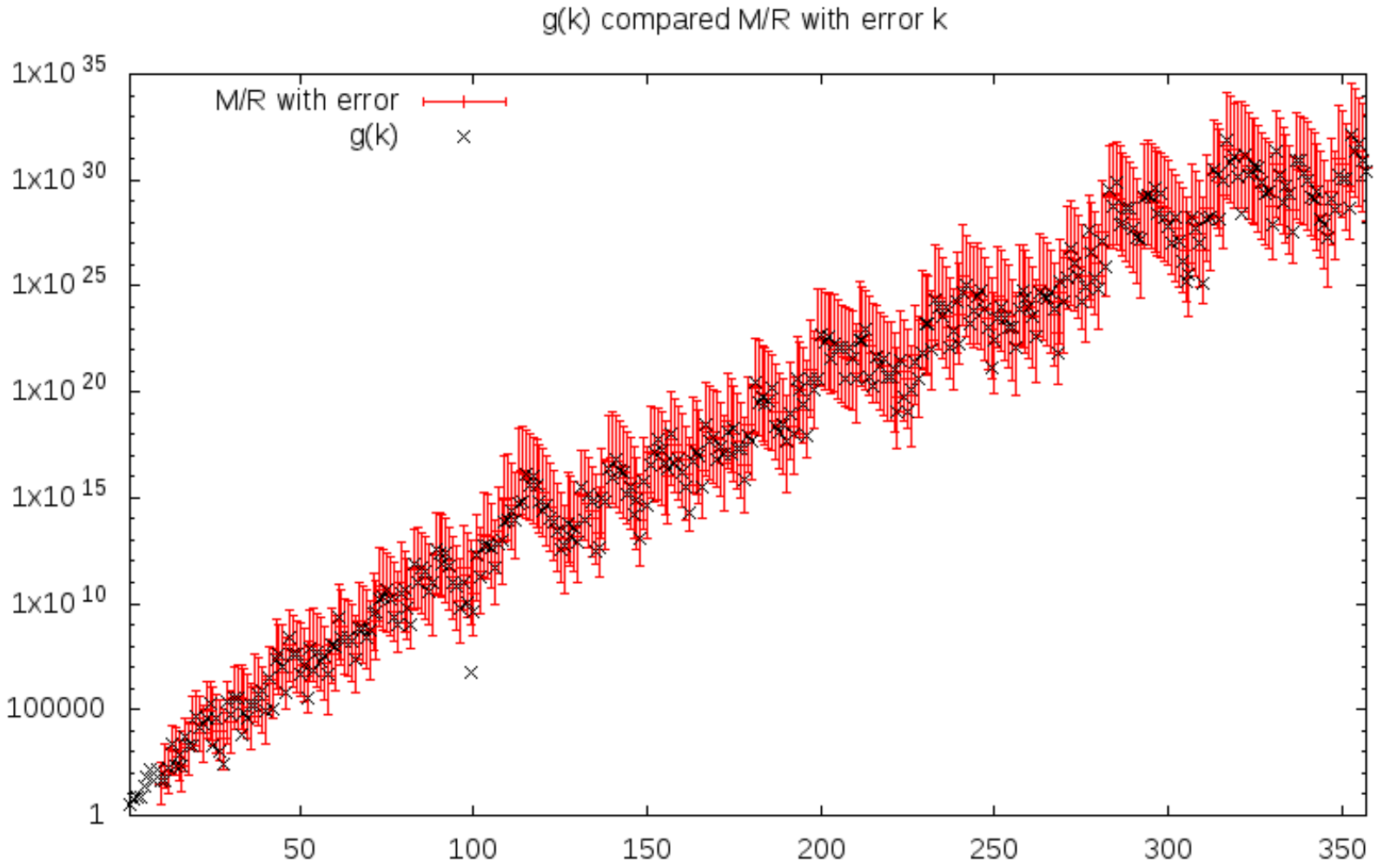
The UDH implies that, with high "probability",

$$\hat{g}(k)/k < g(k) < \hat{g}(k) \cdot k,$$

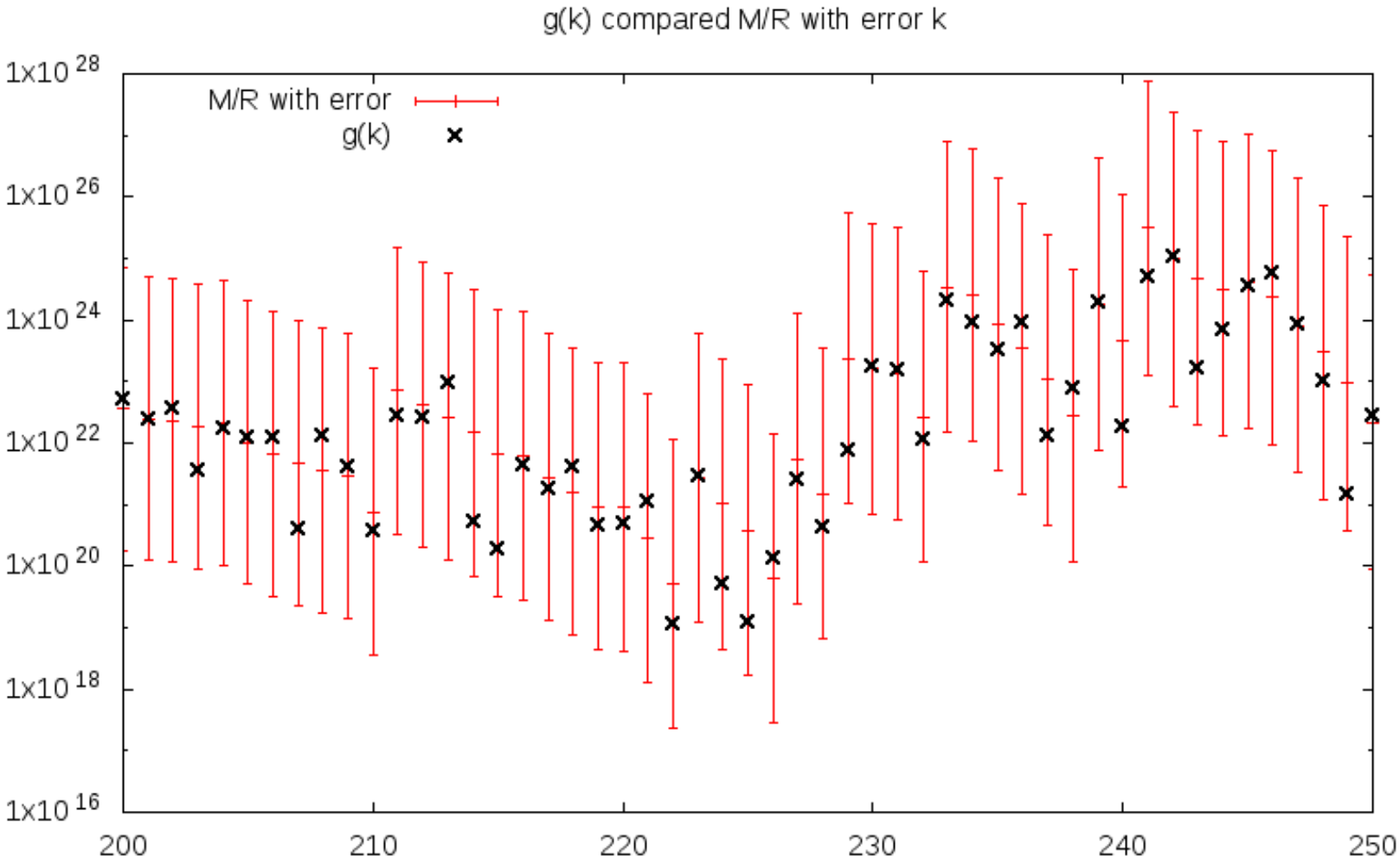
or

$$\log g(k) = \log \hat{g}(k) + O(\log k).$$

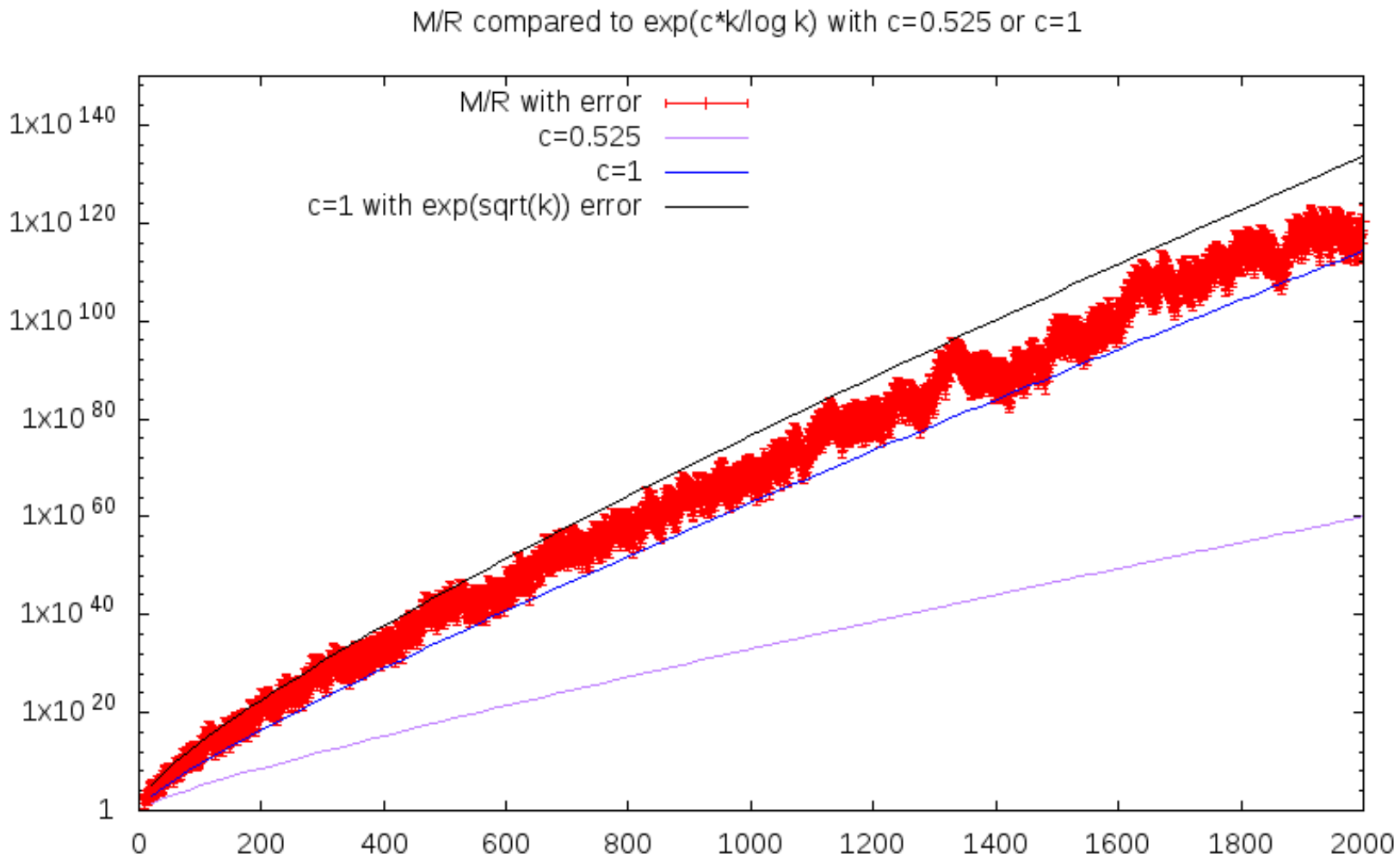
$g(k)$ versus $\hat{g}(k)$ (logscale)



Zooming in: `for(k=200; k<250; k++)`



Upper and lower bounds



Main Theorem Proof: Three Amigos

For prime p, write k in base p:

$$k = \sum_{i=0}^{\lfloor \log_p k \rfloor} a_{ip} p^i.$$

Main Theorem Proof: Three Amigos

For prime p, write k in base p:

$$k = \sum_{i=0}^{\lfloor \log_p k \rfloor} a_{ip} p^i.$$

$$\hat{g}(k) = \frac{M_k}{R_k} = \frac{\prod_{p \leq k} p^{\lfloor \log_p k \rfloor + 1}}{\prod_{p \leq k} \prod_{i=0}^{\lfloor \log_p k \rfloor} (p - a_{ip})}$$

Main Theorem Proof: Three Amigos

For prime p, write k in base p:

$$k = \sum_{i=0}^{\lfloor \log_p k \rfloor} a_{ip} p^i.$$

$$\hat{g}(k) = \frac{M_k}{R_k} = \frac{\prod_{p \leq k} p^{\lfloor \log_p k \rfloor + 1}}{\prod_{p \leq k} \prod_{i=0}^{\lfloor \log_p k \rfloor} (p - a_{ip})}$$

$$= \prod_{p \leq k} \prod_{i=0}^{\lfloor \log_p k \rfloor} \frac{p}{p - a_{ip}}$$

Main Theorem Proof: Three Amigos

For prime p, write k in base p:

$$k = \sum_{i=0}^{\lfloor \log_p k \rfloor} a_{ip} p^i.$$

$$\begin{aligned}
\hat{g}(k) = \frac{M_k}{R_k} &= \frac{\prod_{p \le k} p^{\lfloor \log_p k \rfloor + 1}}{\prod_{p \le k} \prod_{i=0}^{\lfloor \log_p k \rfloor} (p - a_{ip})} \\
&= \prod_{p \le k} \prod_{i=0}^{\lfloor \log_p k \rfloor} \frac{p}{p - a_{ip}} \\
&= \prod_{p \le \sqrt{k}} \prod_{i=0}^{\lfloor \log_p k \rfloor} \frac{p}{p - a_{ip}} \cdot \prod_{\sqrt{k} < p \le k} \frac{p}{p - a_{1p}} \frac{p}{p - a_{0p}}.
\end{aligned}$$

What we did: $g(k)$ (single core)

k	$g(k)$
201	235 54612 35023 12966 07453
202	371 93707 68876 94169 93998
203	36 66628 18040 77694 67119
204	178 22243 70804 75634 88989
205	119 23364 21369 70734 19215
206	118 94994 19601 54916 70238
207	4 14102 11738 06206 56623
208	128 63517 97975 53174 93464
209	40 80254 70430 94462 56859
210	3 81063 47274 59626 93595
211	277 19087 51211 86811 93467
212	254 11430 46501 91044 33623
213	941 02942 94951 13843 10999
214	5 42943 43587 62853 77239
215	1 94050 01839 78664 31743
216	43 71951 29369 55065 01119
217	17 60181 71551 23707 20217
218	40 46933 47457 90358 45374
219	4 60119 05176 06932 47999
220	5 07302 74025 33237 33471
221	11 24738 59029 35409 05471
222	11720 13806 06713 22847
223	29 34696 47028 07658 76223
224	51633 58728 02682 87224
225	12369 18109 50028 52853

k	$g(k)$
226	1 33170 49136 16068 80243
227	25 43371 29078 24284 53367
228	4 42953 79137 83327 73614
229	74 31339 46454 40891 68359
230	1795 17836 21533 83405 06863
231	1535 32995 48871 64662 39991
232	111 43965 49911 64968 95483
233	20200 73550 49977 05129 39243
234	9141 02029 72226 64023 95374
235	3353 56843 86952 45592 15615
236	9004 68924 26010 08758 44863
237	128 10339 84890 50088 80623
238	797 67177 19809 53861 33999
239	18991 37758 35752 30838 29999
240	179 81118 13875 42559 25240
241	50194 81877 83204 04927 52119
242	1 05794 00205 01218 84737 54618
243	1675 50917 78080 00171 78623
244	7032 91964 49292 36074 24244
245	35619 19278 93870 30042 55997
246	57819 80943 94360 21432 63998
247	8791 54436 07103 73768 64247
248	1028 52788 08587 24965 75999
249	14 56775 25795 65720 74749
250	276 46926 37489 69206 77374

k	$g(k)$
251	3056 66797 58148 26766 11579
252	10505 00162 90998 95371 30494
253	3103 09358 30344 20590 94269
254	1166 62737 17826 44531 56094
255	1021 56121 82556 87267 01055
256	128 22340 15164 11349 38548
257	7712 29340 10480 52695 50483
258	58587 79034 00801 08562 54858
259	28954 56510 29429 20300 85999
260	9052 78792 60680 37520 93549
261	3752 17161 28291 37355 29917
262	370 44716 59526 93861 95399
263	52789 04430 06789 43127 33639
264	34644 90142 64935 39757 27919
265	29014 53224 87882 19896 83691
266	45629 29239 07110 01927 14698
267	8235 39060 08758 91988 86219
268	68 39739 60096 00722 01118
269	1 61558 09307 54284 34696 01199
270	17012 60056 85638 85052 85598
271	2 37245 88062 88508 66946 32223
272	57 61284 34192 78614 55093 37498

What we did: $g(k)$ (192 Cores)

k	$g(k)$
272	57 61284 34192 78614 55093 37498
273	11 93755 72096 07235 88168 84023
274	2 88454 13176 35913 24169 68574
275	17152 34131 63802 94572 85911
276	88030 17168 24411 10341 86038
277	453 93397 13957 10829 21927 70333
278	39 50539 72728 23202 00849 01718
279	2 66648 13175 24792 99862 36799
280	70874 78896 73459 57906 03609
281	123 66293 54022 28562 17374 11069
282	7 38297 70384 67082 65425 71838
283	37813 55429 48519 12235 53898 37243
284	6100 10364 48359 18395 19770 39199
285	78766 18312 18052 31134 42561 68735
286	747 51565 01679 14418 20981 52223
287	992 72191 61150 70855 53665 86719
288	5090 76951 48442 32227 73921 76099
289	4834 99245 99858 90424 83401 35289
290	580 10024 22391 77582 79250 69666
291	209 69391 29197 03178 94977 03719
292	174 18908 52958 77197 48733 28493
293	16639 09980 87532 46018 20569 16799
294	20223 01592 35223 93093 61644 12799
295	14858 57580 15296 66376 70445 68447
296	41418 90259 64755 93533 87671 33096
297	2412 51951 98121 56990 65688 86073
298	25619 63627 54642 94279 56273 35598
299	1832 43102 56640 25079 58634 93499
300	701 85519 63812 11947 39815 22430

k	$g(k)$
301	113 92964 05228 07857 10715 23117
302	1742 88530 64455 07964 88047 54943
303	129 26741 47619 33558 63300 61679
304	13 26053 17393 60472 36038 80314
305	1 72946 53384 73935 85567 11859
306	3 51841 28928 12034 40626 05307
307	1779 34819 88869 76850 45198 63743
308	563 71964 39859 00813 40202 10998
309	98 07021 15457 23811 10525 81749
310	1 24437 81505 29347 17696 51070
311	1560 53896 22680 68278 05256 06711
312	1796 99278 95512 29968 42460 24124
313	3 00996 54176 68374 47827 87101 84313
314	1 87014 93014 72478 06122 67573 88094
315	1361 11485 02742 01184 89157 03743
316	1 03374 01931 39808 86145 47639 65949
317	68 85447 25707 42253 40215 24113 79199
318	6 86881 00807 03611 96229 81358 41598
319	11 86184 98065 18829 52817 06712 46719
320	1 40079 84256 27063 06819 06499 16746
321	2435 79072 21965 54229 62339 49121
322	15 25966 57699 53539 87155 01511 11623
323	1 69829 77104 46041 21145 63251 22499
324	2 63741 60892 64064 50498 97681 38599
325	4 61167 55159 08879 62071 94084 17237
326	3 98508 97267 68151 31902 68129 69326
327	80125 12464 39084 44313 42285 17847
328	25358 25141 13174 63981 15617 02348
329	27539 98951 79861 74202 62006 60349
330	736 38290 36775 34959 12602 87866

k	$g(k)$
331	20 99993 85982 46331 44915 31985 15583
332	1 56353 98925 25226 59787 39317 38108
333	8456 75301 70500 03521 07482 84239
334	53204 32892 68314 42061 49241 71599
335	22796 62357 31196 28995 02247 14111
336	349 35354 92097 35618 92451 13714
337	8 11762 65164 78657 57300 21300 74967
338	9 25106 21191 02282 31521 77652 84338
339	2 13133 77351 76125 57319 62221 93619
340	94762 09789 37827 81019 73160 52471
341	14699 48703 85053 43492 43211 94997
342	13442 11412 62865 55639 32278 13718
343	29494 38065 34718 67793 30339 89119
344	1438 56418 63105 76769 23108 10974
345	815 08296 83685 96664 34364 17497
346	193 67357 40155 38474 19752 26746
347	12044 06321 00589 22956 10198 04123
348	4245 11428 30357 75316 13573 01599
349	1 81690 93222 39869 99065 49003 41599
350	1 09702 38255 35035 97742 40622 77982
351	1 09702 38255 35035 97742 40622 77983
352	4660 85432 99042 93285 18996 37232
353	148 50654 32318 56407 35281 78906 02479
354	24 43080 13874 67081 98567 94505 54234
355	50 76963 11372 13168 17954 82711 30491
356	9 29253 34900 21677 69458 86574 61236
357	2 80803 34667 27432 75770 65998 07359
358	13596 95762 45853 24815 61318 93743
359	23 26745 54326 45579 54488 31750 77239
360	4 73246 49994 16835 77114 34474 66861
361	29 15996 91471 08287 80831 78257 81743

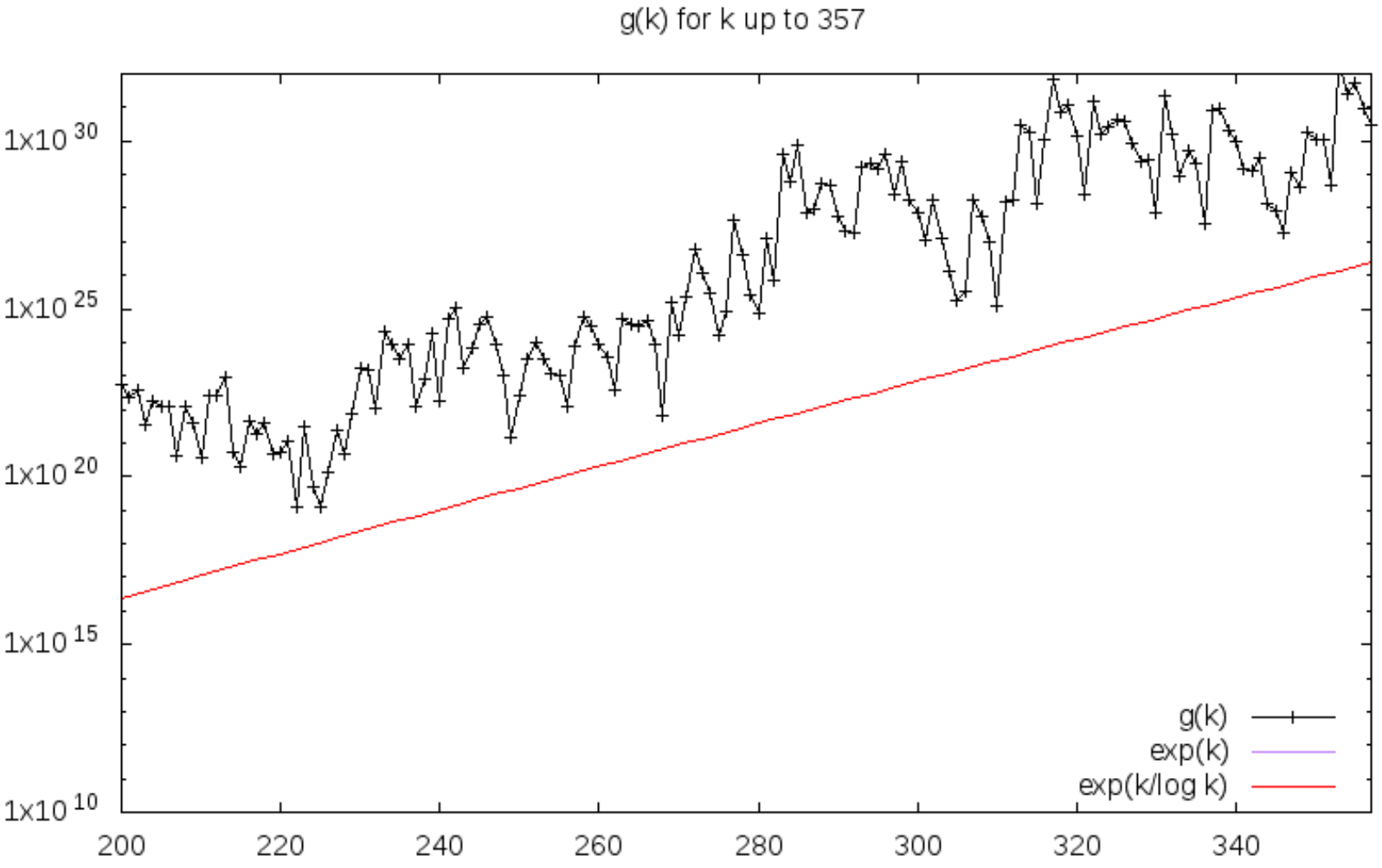
Webster's Conjecture

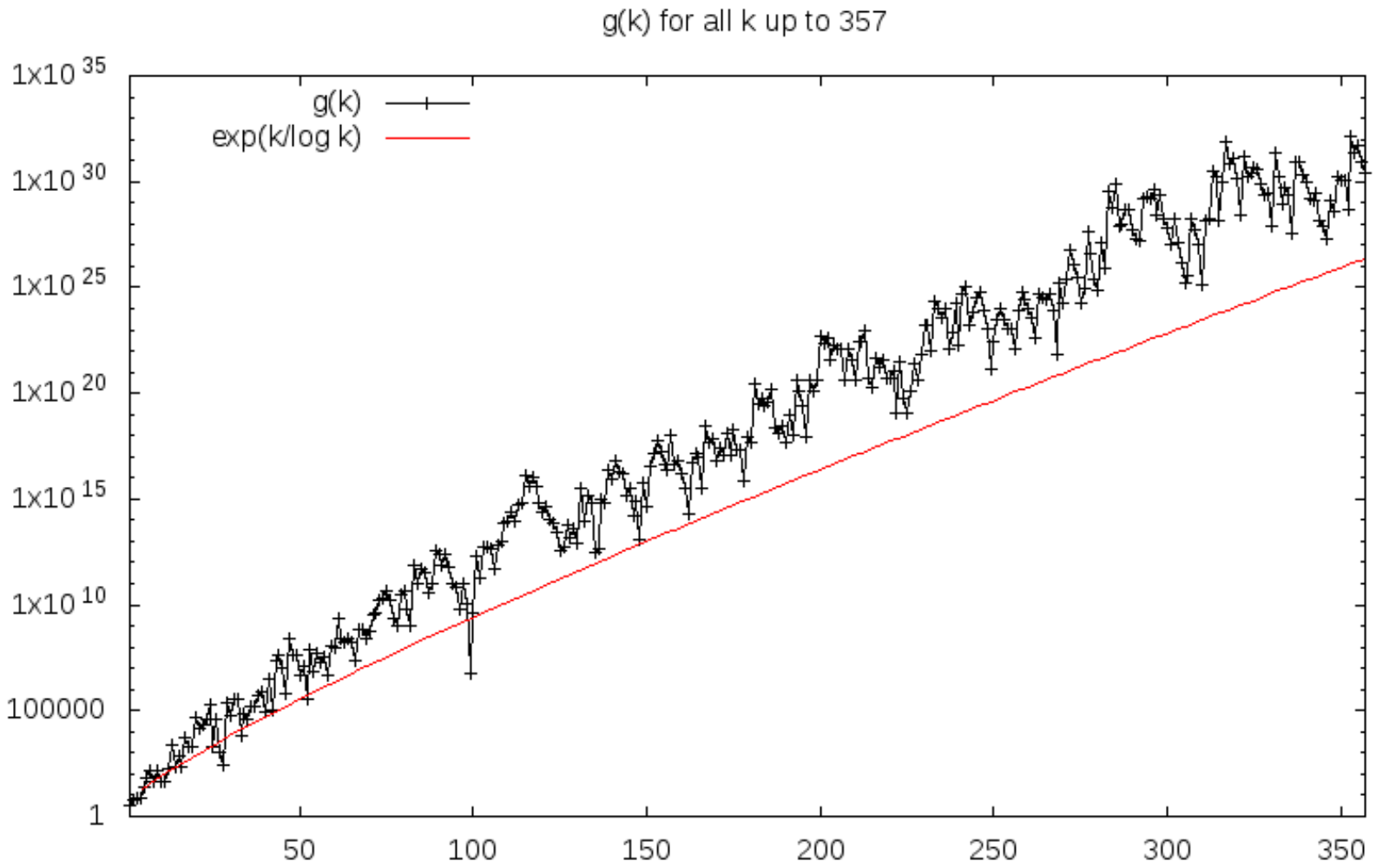
Webster's Conjecture (2019)

$g(k) + 1 = g(k + 1)$ infinitely often.

$g(2) + 1 = g(3) = 7$
$g(10) + 1 = g(11) = 47$
$g(18) + 1 = g(19) = 2099$
$g(36) + 1 = g(37) = 152189$
$g(350) + 1 = g(351) = 1097023825535035977424062277983$

What we did: $g(k)$

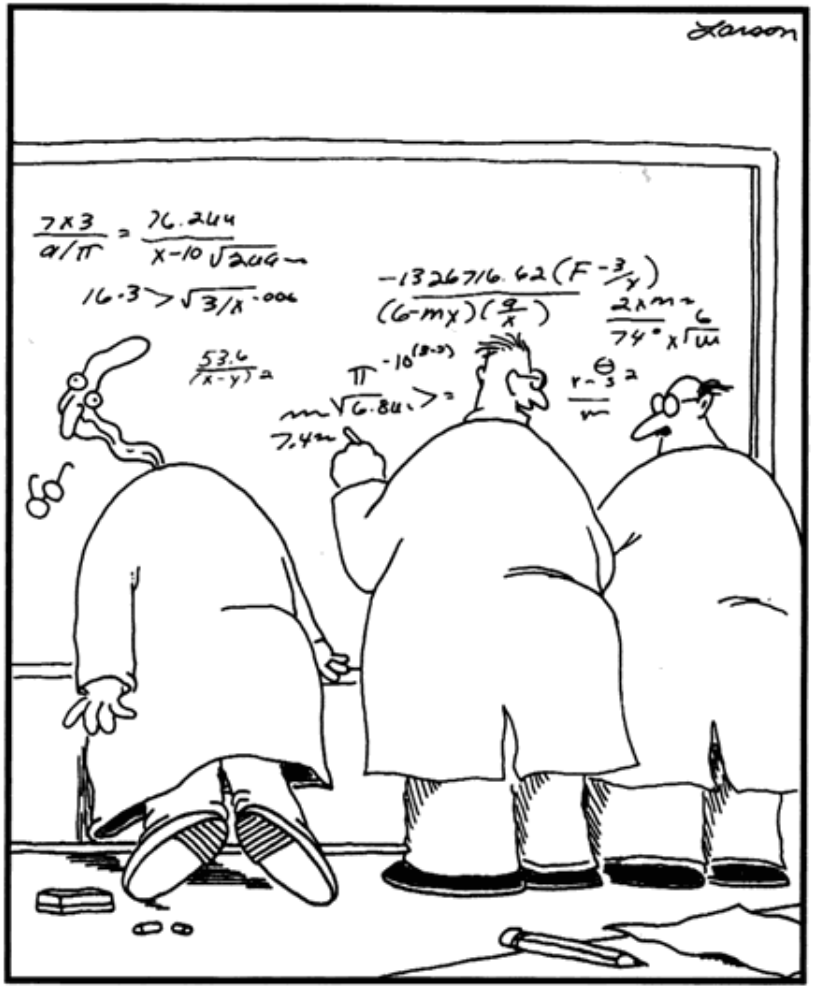



g(k) for all k up to 357
g(k)
exp(k/log k)
1x10 35
1x10 30
1x10 25
1x10 20
1x10 15
1x10 10
100000
1
50
100
150
200
250
300
350

What we did: $g(k)$

- New algorithm with sublinear running time
- Computed $g(k)$ for $200 < k \leq 377$
- Defined $\hat{g}(k)$, UDH implies $g(k) \approx \hat{g}(k)$
- Better Knapsack algorithm (senior thesis for Brianna)

Thank you!



"Ha! Webster's blown his cerebral cortex."