A Simple Algorithm to Calculate S(n)

by John C. McCarthy

Introduction

This short paper first outlines an "obvious" algorithm for calculating S(n) (the smallest integer m such that m! is divisible by n). Doubtless, there exist more subtle and efficient algorithms. I hope some readers will devise these and enlighten me concerning them through this journal.

This is followed by a small scale investigation of the efficiency of the algorithm.

Then there is a short discussion of a simple way of reducing the space required for storage of all S(n) for ranges of n. The storage space required for S(n) for all n which my routines can handle is considered.

Heavily commented listings of an implementation of the algorithm in "C", sample output and timing data are included to help illustrate the algorithm.

The Algorithm

The algorithm is described in detail at the start of the header file "S(n).H". Together with "S(n).C", this forms all the code necessary to implement the algorithm. Note that, for the S(n) function to work correctly, the function make_primes() must first be called from the main program.

The code for printing S(n) and timing the routines has been omitted. These activities are both implementation specific and easily done. They are therefore left as an exercise for the interested reader.

The algorithm hinges on finding the prime factors of n. Improvements on how this is done will most benefit its efficiency.

To be practical, the given implementation of the algorithm only works for $0<n<2^{32}$. However, the algorithm is generally applicable to any non-null integer.

Tables of S(n), constructed using the routines of "S(n).C", for the largest 2000 permitted n are included.[1] My paging routines are rather elaborate. Using them (without printing!), it took 2.4 hours to discover that 3,745,708 pages, as tightly packed as those shown, would be required to print S(n) for all $0<n<2^{32}$.

Efficiency of the Algorithm

In a letter to R. Muller (about computing the Smarandache Function, July 19, 1993), Ian Parberry (editor of <SIGACT News>,

[1] *For the smallest 4800 numbers, see Ibstedt's table (pp. 43-50) of this current journal.*

Denton, Texas) expressed that one can immediately find an algorithm that computes S(n) in O(nlogn/loglogn) time ('A Brief History of the "Smarandache Function"' by Dr. Constantin Dumitrescu, Department of Mathematics, University of Craiova, Romania). Disappointingly, a little analysis of the accompanying timing data on my TI85 advanced scientific calculator reveals that my algorithm is somewhat worse than this.

Trying to fit the version 2 timing data to various O(f(n)), I obtained the following results (x=3355443200 and 10(O(x+99)-O(x-100)) is calculated for comparison with the last entry of the version 1 timing data):

O(f(n))	Correlation Coefficient	$O(2^{32}-1)$ (years)	10(O(x+99)-O(x-100) (milliseconds)
O(n)	0.9928879	0.6092	8909
O(nlogn/loglogn)	0.9944006	0.7906	11827
$O(n\sqrt{n})$	0.9997756	24.2	469178

$O(n\sqrt{n})$ fits the version 2 timing data best, although the time it predicts for the last entry of the version 1 timing data is almost 3 times too large. Hence, I assume the time complexity of my algorithm is a little better than $O(n\sqrt{n})$.

As a rough upper limit on the time my program (on my 20MHz 368DX PC) would take to calculate S(n) for all $0<n<2^{32}$, let us assume that every value of n requires as much time as each n in the range of the last entry of the version 1 timing data (= 159111/199/10 = 79.9553 ms). In this "worst case", it would take 10.882 years. $O(n\sqrt{n})$ time complexity predicts more than twice this value, which is a measure of how pessimistic it is.

I would welcome a more rigorous analysis of the time complexity of my algorithm as I presently lack the necessary expertise.

Simple Compression of Stored S(n)

Without compression, each S(n) would be stored as a 32-bit (= 4 bytes) value. Hence 2^{34} bytes (= 16 Gigabytes) would be required to store S(n) for all $0<n<2^{32}$.

This requirement can be reduced considerably if we use the high bit of each each byte of each value to indicate if it is the last byte of the value. If the bit is set it means that further byte(s) are required and if it is reset it means that the byte is the last byte of the current value. This means that only 7 bits of each byte are used to form the numerical part of the value. Assuming that, as with Intel format, the values are stored low-'byte' (actually 7 bits) first, here are some examples:

i) 127 requires seven bits and so just one byte (with high bit reset to indicate no further bytes).
ii) 16,000 requires 14 bits. So it is stored as two bytes. The

first is 0 (16,000 mod 128) + 128 (to set the high bit indicating there is more to come). The second is 125 (16,000 div 128) (with high bit reset to indicate no further bytes). This reads simply as 0 (with more to follow) + 128*125 (no more to follow).

iii) A number stored as the three bytes 57+128, 93+128 and 125+0 would similarly represent:
57 + 93*128 + 125*128*128 = 2,059,961.

The largest numbers that can be represented by a given number of bytes is thus as follows:

1 byte can code up to 2^7-1 = 127.
2 bytes can code up to $2^{14}-1$ = 16,383.
3 bytes can code up to $2^{21}-1$ = 2,097,151.
4 bytes can code up to $2^{28}-1$ = 268,435,455.
5 bytes can code up to $2^{35}-1$ = 34,359,738,355 (or 8 times the largest unsigned long).

For small values of n, the savings are considerable (400%). However, even large n often have small S(n).

Using this technique to compress all S(n) calculated for some ranges of n (each range was also stored), I obtained the following results:

range of n	compression	time taken (seconds)	size	size after pkzip
1	without	4.5	40,008	19,836
-10,000	with	4.7	15,749	15,267
2,147,478,648	without	827.3	40,008	33,729
-2,147,488,647	with	842.4	33,541	30,836
4,294,957,296	without	1,066.2	40,008	34,320
-4,294,967,295	with	1,085.1	34,330	31,634

The results indicate that this compression is a little better than pkzip's (a commercial file compression utility). Application of pkzip to a pre-compressed file also gives a slight improvement.

Assuming that the savings shown for the middle range of 10,000 n are the average of all ranges of 10,000 n, using my compression together with that of pkzip would permit storage of S(n) for all $0<n<2^{32}$ in about $3.0836*2^{32}$ = 12.3344 Gigabytes. So look out for sets of 19 CD-ROMs with all your favourite numbers on them!

21st November 1993

S(n).H

```
/* (c).1993.11.13.John.C.McCarthy
   "S(n).h"

   Example Implementation of A Simple Algorithm to Calculate S(n),
   The Smarandache Function:

   Because there are more people familiar with C than with C++, this module
   has been written entirely in C (apart from "//" style comments).  The
   module was compiled using Borland C++ version 3.1.

   For efficiency, n is constrained to the limits of an unsigned long.  Hence,
   0 <= n <= 2^32 - 1 (= 4,294,967,295).  ("^" represents exponentiation).
   Although catering for n of vast magnitude is possible, it imposes heavy
   storage and processing overheads.  The range of an unsigned long therefore
   seems a reasonable compromise.

   The algorithm depends on the most elementary properties of S(n):
   1) Calculate the STANDARD FORM (SF) of n:
      In SF: n = +/-(p1^a1)*(p2^a2)*...*(pr^ar) where p1, p2,...pr denote the
      distinct prime factors of n and a1, a2,...ar are their respective
      multiplicities.
   2) S(n) = max[ S(p1^a1),...,S(pr^ar)].
   3) S(p^a), where p is prime, is given by:
      3.1) a<=p  ==>  S(p^a) = p*a.
      3.2) a>p   ==>  S(p^a) = x < p*a.  In this case, fortunately rare, x
           is the smallest integer such that p appears as a factor in the
           list of all integers > 1 and <= x at least a times.  Let the no.
           of times p appears as a factor in the list of all integers > 1
           and <= y be f(y, p).  Then:
           f(y, p) = Σ[int(y/(p^i))] for i>0 while y>=(p^i).
           Hence, x is the smallest integer such that f(x, p)>=a.
           Note that between succesive integer multiples of p there are no
           integers which have p as a factor.  The trick here is to look for
           the largest multiple of p (call it c), such that f(p*c, p)<=a
           (so that x = p*c, if f(p*c, p) = a, else x = p*(c+1)):
           3.2.1) c = a-2  (largest possibilty for c since f(p*(a-1), p)>=a
                  when a>p (Note: f(p*(a-1), p)=a is not sought for slight
                  performance gain)).
           3.2.2) z = f(p*c, p).
           3.2.3) While(z>a):
                  3.2.3.1) d = no. of times p appears as a factor of p*c
                             = (no. of times p appears as a factor of c) + 1.
                  3.2.3.2) c = c-1  (next largest possibility for c).
                  3.2.3.3) z = z-d  (= f(p*c, p)).
           3.2.4) If(z<a), x = p*(c+1).
           3.2.5) Else     x = p*c.

  To calculate the prime factors of all 32-bit n requires use only of primes
  < (2^16) (i.e. all primes expressible as an unsigned short integer).  This
  is because any factor of n remaining after division of n by all its prime
  factors < (2^16) is simply a prime.  Since there are only 6542 16-bit
  primes, the program first creates a list of these (which only takes about
  4 seconds on my 20 MHz 386DX PC) so that they never have to be
  recalculated, thus saving much time.
*/
```

```
#define PRIMES16 6542    // The number of 16-bit primes
#define MAX_SFK 9  /* max. distinct primes in the SF of n.  The smallest
    number with more than 9 distinct primes is the product of the 10 smallest
    primes (= 6,469,693,230), which is substantially more than the largest
    integer expressible as an unsigned long.  Hence, 9 distinct primes are
    more than ample.
*/

typedef unsigned long u_long;
typedef unsigned int  u_int;
typedef enum {false, true} boolean;

struct SF_struct {
  int    sfk;              // no. of distinct primes
  u_long sfp[MAX_SFK];     // the distinct primes
  int    sfa[MAX_SFK];     // respective multiplicities
};

extern u_int prime[PRIMES16+1];  // list of all 16-bit primes
                                 // plus terminating zero.

void make_primes(void);  // construct list of all 16-bit primes (prime[]).
                         // Must be called before calls to getSF() or S().
void getSF(u_long n, struct SF_struct *SF); // calc. SF of n and store in SF
u_long S(u_long n);      // calc. S(n)
u_long Spa(u_long p, int a);  // calc. S(p^a) where p is prime
int f(int x, int p);  /* the number of times the prime p appears as a factor
    in the integers from 1 to x inclusive.  This function is only called from
    Spa(p, a) when a>p with x=p*(a-2) (refer to item (3) of algorithm outline
    above).  Max value of (a) occurs when p is a minimum, n is a maximum and
    (p^a)=n.  So, (2^max(a))=max(n)=(2^32)-1.  Hence max(a)<32.  So, x<60
    when (a) is at its max.  Max value of p (and x) occurs when a=p+1 and
    (p^a)=max(n).  So, max(p)^(max(p)+1)=(2^32)-1.  The upshot is that
    max(p)=9 when a=10.  Hence, max(x)=72.  This explains why it is safe for
    x, p and the return value of f(x,p) to be passed as ints.
*/
```

```
/* (c).1993.11.13.John.C.McCarthy
   "S(n).c"

   Example Implementation of A Simple Algorithm to Calculate S(n),
   The Smarandache Function:

   This is the code for the module.  Refer to "S(n).h" for details.
*/
#include "S(n).h"

u_int prime[PRIMES16+1]; // allocate storage for list of all 16-bit primes
                         // plus terminating zero.

void make_primes(void)
{
  u_int *pp;  // ptr to last prime so far of prime list
  u_int *tp;  // ptr to current test prime
  u_int p;    // number being tested for primality

  pp=prime;  // point to start of prime list
  *pp=2;     // set first prime to 2
  *++pp=3;   // set second prime to 3
  p=5;       // next possible prime.  N.B. p is kept odd so that trial
             // division by 2 is unnecessary.
  while(true) { // infinite loop!:
    tp=prime+1;    // point to first odd test prime
    // whilst test prime <= √p:
    while(((long) *tp)*(*tp)<=p) {
      if(!(p%*tp)) {  // If current test prime divides (is factor of) p:
        p+=2;         // try next odd number
        if(p<*pp) {   // done when p overflows:
          *++pp=0;       // terminate list
          return;
        }
        tp=prime+1;   // point to first odd test prime
      }
      else ++tp;      // Else point to next test prime
    }
    // no prime <= √p divides p so p must be prime:
    *++pp=p;     // so store it next in the list
    p+=2;        // try next odd number
    if(p<*pp) {  // done when p overflows:
      *++pp=0;      // terminate list
      return;
    }
  }
}
```

```
void getSF(u_long n, struct SF_struct *SF)
{
  u_int *pp;  // ptr to current prime
  u_long r;   // 'residue' of n remaining for factoring

  SF->sfk=0;  // no. of distinct prime factors discovered
  r=n;
  pp=prime;   // point to start of prime list

  // whilst current prime <= √r and prime list not exhausted:
  while(((long) *pp)*(*pp)<=r && *pp) {
    if(!(r%*pp)) {              // if current prime is a factor of r:
      SF->sfp[SF->sfk]=*pp;       // store current prime as next prime of SF
      SF->sfa[SF->sfk]=1;         // set its multiplicity to 1
      r/=*pp;                     // 'divide out' current prime
      while(!(r%*pp)) {           // while current prime factors r:
        SF->sfa[SF->sfk]++;         // increment multiplicity
        r/=*pp;                     // 'divide out' current prime
      }
      SF->sfk++;                  // increment count of distinct prime factors
    }
    ++pp;                       // next prime
  }

  if(r>1) {                   // If n contains prime > 2^16:
    SF->sfp[SF->sfk]=r;         // store it as last prime of SF
    SF->sfa[SF->sfk]=1;         // set its multiplicity to 1
    SF->sfk++;                  // increment count of distinct prime factors
  }
}
```

S(n).C

```
u_long S(u_long n)
{
  struct SF_struct SF;  // to store SF of n
  int sfi;              // index of current term of SF of n
  u_long Sn;            // current guess at S(n)
  u_long x;             // S(current term of SF of n) where it might exceed
                        // current value of Sn.

  if(n==1) return 0;    // special case

  getSF(n, &SF);        // calc. and store SF of n

  // First guess at S(n) is S(p^a), where p is the largest prime in the SF
  // of n and a is its multiplicity.  This pre-empts the calculation of S(p^a)
  // for the remaining terms where, as is likely, p*a for these terms is <=
  // this initial guess (since S(p^a) <= p*a always):
  sfi=SF.sfk-1;
  Sn=Spa(SF.sfp[sfi],SF.sfa[sfi]);

  while(sfi>0) {  // while more term(s):
    sfi--;              // next term
    if(SF.sfp[sfi]*SF.sfa[sfi]>Sn) {  // if this term may have larger S(p^a):
      x=Spa(SF.sfp[sfi],SF.sfa[sfi]);    // calc. it
      if(x>Sn) Sn=x;                      // if new max., update Sn with it
    }
  }
  return Sn;  // That's all folks!
}

u_long Spa(u_long p, int a)
{
  // Refer to item 3) of the algorithm description in S(n).h.
  int c;  // largest multiple of p such that f(p*c, p) <= a (eventually!)
  int z;  // f(p*c, p)
  int m;  // used to calc. no. of times p appears as factor of c

  if(a<=p) return p*a;

  c=a-2;
  z=f(p*c, p);
  while(z>a) {
    // d in items 3.2.3.1) and 3.2.3.3) of algorithm description is implicit
    // here:
    z--;
    m=c--;
    while(!(m%p)) {  // while p divides m:
      z--;
      m/=p;              // 'divide out' factor of p from m
    }
  }
  if(z<a) return p*(c+1);
  else return p*c;
}
```

S(n).C

```
int f(int x, int p)
{
  int k=0;  // count of appearance of prime p as a factor in the integers
            // from 1 to x.
  int xdp;  // successive divisions of x by p

  xdp=x/p;
  while(xdp>0) {
    k+=xdp;
    xdp/=p;
  }
  return k;
}
```

SMARANDACHE FUNCTION, S(n), for n=4294965296 to n=4294967295

\S of n plus:

n\	0	1	2	3	4	5	6	7	8	9
4294965296	268435331	168887	2799847	60492469	14316551	3226871	49941457	231547	337867	2798023
4294965306	5531	4294965307	1194373	101977	5174657	66569	967	4294965313	113025403	286331021
4294965316	1073741329	4583741	102261079	27356467	1602599	4919777	2147482661	22967729	2117833	24542659
4294965326	2147482663	648101	7255009	2671	57427	4294965331	3741259	75577	85889	858993067
4294965336	21517	11392481	5520521	2199163	12632251	40139863	2286989	595613	979691	4621
4294965346	3919	4294965347	6911	90407	83641	1431655117	1228537	2545919	715827559	1493
4294965356	14969	158597	3109	108907	2339	4294965361	2147482681	23719	118423	38629
4294965366	419	30971	4925419	1431655123	195137	11897411	357913781	104755253	641	1307
4294965376	5527	830587	154573	279857	1199711	204522161	39107	4294965383	6628033	858993077
4294965386	2147482693	21367987	3567247	30034723	7535027	35449	268435337	5749619	2147482697	86357
4294965396	357913783	4040419	35461	27012361	1952257	377513	70867	26349481	511549	977239
4294965406	235289	2347	154807	529393	5167	14461163	672349	4294965413	34301	24683
4294965416	5179	625997	22138997	4013	23860919	15107	17747791	204522163	39727	10105801
4294965426	816223	9965117	12109	6040739	321239	28793	5772803	746561	5003	1453457
4294965436	65983	1530091	79536397	2473	13421767	1551089	308857	2617	28871	858993089
4294965446	178793	65579	36097	564607	28633103	11731	121313	13997	2465537	2207
4294965456	8329	4294965457	567067	423191	70019	4294965461	415937	138547273	536870683	168331
4294965466	10275037	14364433	7615187	1822217	48699	88873	169681	4001	5549051	6079
4294965476	157	1194041	2147482739	1935541	16233	22031	718943	84869	1073741371	8863
4294965486	102261083	4294965487	14639	973253	18673763	2754949	7253	30509	28661	137593
4294965496	50951	14387	715827583	390451409	1227133	835759	16909313	637519	294337	72679
4294965506	17903	72091	29483	3804221	255197	2017	103583	7039	306783251	188417
4294965516	357913793	99882919	4699087	12236369	53687069	832519	715827587	142799	1073741381	1847297
4294965526	2147482763	252645031	57193	4294965529	384509	103391	32789	6381821	12132671	122713301
4294965536	5835551	26881	88919	3296213	2468371	6529	2306641	11831861	64319	106747
4294965546	19979	107077	1073741387	29339	2321683	622549	89478449	179623	17599	891997
4294965556	4948117	20355287	1038937	12569	28057	2293	7561	77933	119304599	78090283
4294965566	13174741	1431655189	67108837	397057	20452217	19709	51829	20507	298801	1031
4294965576	16268809	1618909	3081037	1431655193	11302541	4294965581	10374613	12781	2949839	337
4294965586	19701677	139	2843	2549	429496559	9739151	8801159	72796027	2970239	50529007
4294965596	1073741399	218941	961703	226050821	198841	4294965601	2147482801	33294307	40459	5335361
4294965606	154373	118361	536870701	923053	9283	35527	307	4294965613	78487	21859
4294965616	515231	4294965617	2957	89611	5147	7495577	2147482811	330381971	4409	3061
4294965626	692513	1828423	46684409	2459	251609	9137	170327	1302689	2147482817	3193283
4294965636	9177277	682933	2147482819	1431655213	2551	4294965641	784039	7027	4549	40153
4294965646	2937733	209623	89478451	119747	85899313	62245879	100547	6400843	160033	40961
4294965656	20731	26371	147827	4294965659	1745921	16582879	118757	28071671	20929	858993133
4294965666	9061109	17111417	119557	4603393	429496567	4294965671	59652301	4294965673	93368819	3011
4294965676	1073741419	28825273	715827613	4294965679	101873	18013	505409	4294965683	1360889	12347
4294965686	195225713	1431655229	400949	2038427	47721841	4294965691	130261	45613	113025413	7880671
4294965696•	2729	58741	1753	477218411	42949657	3373	94399	13054607	201907	286331047
4294965706	1059439	594953	1151	75431	61356653	13267	72727	2238127	1831	858993143
4294965716	220889	1286303	2147482859	889411	1556147	4294965721	118273	249287	153391633	29153
4294965726	1716613	849311	601873	8783161	70099	494413	23057	4294965733	2147482867	859853
4294965736	536870717	4294965737	179	7517	16519099	130150477	1689601	7309	6701	1553333
4294965746	2147482873	887573	63161261	650851	121843	226050829	29173	142837	2147482877	6270023
4294965756	2818219	4294965757	7877	204522179	1677721	18755309	89669	82183	1073741441	4457
4294965766	1026031	4294965767	2311	24793	7229	477218419	1000691	19792469	65075239	65497
4294965776	268435361	1404961	1162687	603989	4933	12377423	836573	42223	536870723	16943
4294965786	4286393	244157	84407	1811	3539	41698697	7433	4294965793	306783271	286331053
4294965796	4337	55823	57649	919	21474829	7052489	102559	556559	1451	8753
4294965806	23831	41357	24317	116080157	143165527	38008547	10627	1431655271	17471	122713309
4294965816	6761	677119	349241	2165893	57527	4294965821	20789	81037091	20719	19088737
4294965826	40829	2129	357913819	4799	38669	651641	536870729	1395829	238609213	4999
4294965836	359231	7883	5355319	4294965839	1626881	4294965841	2147482921	48661	1757351	99409
4294965846	1018247	4294965847	36599	1431655283	721843	954649	32749	348117	4303573	733
4294965856	7129	237173	72211	148102271	214748293	11098103	309391	4864061	25565273	7933
4294965866	107069	1186127	1073741467	33353	407879	613566553	268435367	65173	2147482937	15959
4294965876	63113	36749	306783277	477218431	107374147	186737647	28031	70309	1091	5843491
4294965886	2147482943	4294965887	2179	2081903	2161	366997	6869	620929	6949783	78090289
4294965896	19289	84121	2147482949	613566557	14316553	252645053	2147482951	9633	1009	858993181
4294965906	6871	364691	1073741477	110127331	429496591	4294965911	87767	7829	7588279	944987
4294965916	12064511	75217	167759	18481	103643	485683	25799	125941	351931	171798637
4294965926	256049	3119	1574401	52181	181913	97553	831713	2243	43826183	3886847
4294965936	89478457	4294965937	1621	130150483	214748297	49803	639703	1290167	536870743	19927
4294965946	74051137	263	3933119	4294965949	4651	79259	3947579	25718359	1494421	122713313
4294965956	5417	38693387	6221	138547289	627919	30034727	306783283	1431655321	241453	858993193
4294965966	1191061	4294965967	2591	8699	991909	4294965971	13597	311749	1990253	457
4294965976	76695821	4294965977	68389	1321	690509	1431655327	911	74471	91867	80513
4294965986	7430737	36709111	26188817	15259	229	61333	117709	5282861	5276371	858993199
4294965996	6907	5956957	33331	4289	131	4294966001	715827667	1322749	38053	32497
4294966006	36398017	4294966007	178956917	60492479	13159	243769	1073741503	471301	1318283	1340083
4294966016	15101	14737	415133	2579	5939	40031	19701679	6537239	529981	19219
4294966026	279511	35495587	258173	254879	2225371	116080163	1439	3637	358691	11003
4294966036	56512711	2235797	65075243	47197429	110581	320927	93368827	4294966043	357913837	1128769
4294966046	238001	163003	134217689	273617	9544369	3256229	3343	238649	126322531	277363
4294966056	4519	2213	2147483029	59069	2788939	54366659	24683713	351269	28627	16231
4294966066	2147483033	613566581	3224449	9695183	10475527	3853	940229	4294966073	11987	171798643
4294966076	103613	477218453	182129	91382257	1033	12521767	303617	46182431	7110871	68791
4294966086	79536409	4294966087	3701	1431655363	4013987	11392483	357913841	1081583	8803	13634813
4294966096	20543	44278001	45137	4294966099	829	85733	306783293	340573	2657	1433
4294966106	2147483053	669937	1073741527	35083	233549	4554577	7064089	167269	69273647	78090293
4294966116	131441	3271109	2147483059	33294311	22171	4294966121	14035837	2633333	1823	11453243
4294966126	2749	73583	257863	4294966129	674249	1759	242873	665783	31122943	2340581
4294966136	536870767	521	2147483069	252645067	1213	1328477	58040083	792283	4793489	27709459
4294966146	1993949	340519	8873897	9697	40009	8641783	4153	4294966153	2147483077	286331077
4294966156	4858559	186737659	77647	390451469	53687077	5168431	288679	4294966163	578213	122713319
4294966166	15675059	155801	905347	251	13015049	82139	16087	6661	9629969	171798647
4294966176	481067	4294966177	74051141	2237	9336883	390451471	55063669	817933	97987	86531
4294966186	128613	4294966187	18837571	80737	119737	21367991	24403217	14965039	191	365063
4294966196	1012009	47279	2008871	7548271	1022611	352133	6317	18313	2562629	19976587
4294966206	1063637	2113	5162219	57847	1633067	2245147	525571	6719	21503	286331081
4294966216	14510021	4294966217	43481	237409	10853	1748053	30246241	1361	5263439	15618059
4294966226	16007	890887	9787	446323	199673	4294966231	536870779	1739557	4029049	812671
4294966236	1121987	4294966237	95287	1190071	246271	38239	23743	4294966243	1073741561	15070057
4294966246	2147483123	566693	462421	26676809	3307	1431655417	1109	116080169	763957	9323
4294966256	38347913	477218473	171401	12744707	13463	5501	16393001	29671	821	29620457
4294966266	163543	355573	1073741567	381673	61356661	96263	683	2217329	2147483137	179
4294966276	859681	87652373	715827713	2963	9761287	4073	6691	35537	58169	858993257
4294966286	165191011	15391	143779	16417	2663	311	63161269	6721387	74051143	37347533
4294966296	304867	4294966297	3883333	332711	26141	72796039	380557	35747	1254371	40904441
4294966306	251197	2458481	357913859	127663	919693	477218479	2753	390451483	15230377	31547
4294966316	26188819	5940479	108191	6074917	5051	226050859	139801	49367429	1600211	197243
4294966326	1571	116080171	536870791	159072827	430789	19489	174337	613566619	49941469	397
4294966336	17159	4294966337	1411889	325697	1614649	2706343	2147483171	252645079	178956931	858993269
4294966346	195225743	68174069	77813	39403361	28633109	135347	5209	54133	200381	858993271
4294966356	56893	8761	2147483179	75350287	107053	146681	23091217	1762399	1367	25733
4294966366	283571	4294966367	581029	401887	429496637	1431655457	117877	4294966373	170557	701219
4294966376	26249	1907	113025431	390451489	14251	908603	108827	159072829	268435399	73727
4294966386	3851	186737669	151637	2464123	273843	33818633	890333	1057613	126322541	19991
4294966396	32783	1013681	17459213	399941	1342177	5557	48761	109001	357913867	858993281
4294966406	116099	1431655469	6047	41698703	3407	22067	97612873	149333	1679033	858993283

n\ \S of n plus:	0	1	2	3	4	5	6	7	8	9
4294966416	1877	813566631	48677	8088449	251	30899039	4804213	390451493	7213	57266219
4294966426	1564081	4294966427	7017919	95569	2939	204522211	577	389849	12541	636763
4294966436	37571	17674759	809	15913	261251	4294966441	614093	1431655481	41539	656221
4294966446	3911627	4294966447	1172207	1137137	965161	2669339	78301	3929521	199	5039
4294966456	26861	288427	6567227	172787	202021	19611719	126322543	99882941	2485513	858993293
4294966466	306783319	130150499	2848121	17970571	143165549	12763	6468323	1949	2147483237	13381
4294966476	38891	4294966477	3308911	7001	661	330382037	321143	2176871	1073741621	2951867
4294966486	89273653	613566641	178956937	3455323	429496649	159072833	56512717	6806603	7866239	858993299
4294966496	7895159	49367431	2147483249	4999961	86767	813566643	5413	1431655501	536870813	6151
4294966506	234467	416623	1153	477218501	81083	1868189	443	390487	1256573	26407
4294966516	306871	49891	1951	124193	8259551	345727	3984199	9565627	10631	7469507
4294966526	2251	477218503	16777213	613566647	443237	1210873	4493	32611	29417579	40123
4294966536	8521759	482093	2147483269	16086017	396947	2307881	715827757	613566649	149713	56509
4294966546	165191021	252645091	16193	3023	12271333	1431655517	536870819	4294966553	238609253	32443
4294966556	73883	1129957	8910719	330382043	64373	119083	52377641	617359	530767	16207421
4294966566	34781	65563	463219	511123	201547	25577	5107	186163	648199	364753
4294966576	1405421	293	37693	138547309	214748329	21577	22139003	4294966583	6170929	9439487
4294966586	58040089	75350291	133169	12739	47721851	4294966591	222953	21367993	298303	179593
4294966596	383	4441	9717119	688627	32587	465781	20369	208261	1073741651	286331107
4294966606	15661	58835159	29826157	22253713	1523	110127349	5088823	7993	9061111	490573
4294966616	870131	105323	764501	4294966619	6551	1693	21262211	92347	11719	16901
4294966626	79536419	45751	12064513	23469763	1297573	312839	719	8597	306783331	2219621
4294966636	873871	330382049	7639	4294966639	5821	15671	74051149	14549	153151	4567
4294966646	2147483323	1281697	1080223	284831	2202547	4294966651	1073741663	799361	195225757	15877
4294966656	17449	4294966657	4121849	139361	86627	4294966661	20063	756023	63997	26030101
4294966666	556487	4294966667	152239	214309	429496667	1237	8659207	1033189	715827779	120223
4294966676	757	8251771	49941473	18979	6053	24826397	86477	12030719	1073741671	5867
4294966686	17959	390451517	2383	36709117	21401	109451	357913891	54366667	1028981	286331113
4294966696	536870837	16582883	21691751	109961	87119	75350293	38569	138547313	12782639	218629
4294966706	2147483353	975907	4492643	16976153	143165557	2280917	536870839	1431655571	701	2111
4294966716	119304631	3607	919	20164163	431	2194	482689	69371	20259277	4241
4294966726	52377643	64103981	13765919	208889	144563	84349	397	889043	4079	924643
4294966736	1020667	30460757	11864549	32292983	71582779	40129	195225761	559459	36209	858993349
4294966746	115811	18041	38653	1431655583	17179867	4141723	14913079	69463	79841	35671
4294966756	1073741689	5811863	1299143	198391	4789	89417	46411	15176561	756689	27709463
4294966766	321047	191	27751	4294966769	47721853	116080183	107021	24265349	306783341	15618061
4294966776	1844917	9133	99223	477218531	16519103	2387419	715827797	6938557	178481	154523
4294966786	1490273	148102303	34019	85199	7459	562979	536870849	330382061	10683997	78311
4294966796	1789	43383503	2147483399	17821439	3579139	93229	93503	1431655601	4481	317323
4294966806	761	99882949	32173	204522229	429496681	1072937	32779	4294966813	4021	88129
4294966816	3257	1516049	359171	1766749	214748341	13899569	26561	2907899	25679	61291
4294966826	29417581	13229	9502139	4294966829	35081	91382273	349981	477218537	27059	858993367
4294966836	48623	613566691	15449521	88609	349753	38351	20347	19259941	275389	1579
4294966846	2147483423	186737689	14983	131437	2381	6299	983	782183	4137733	65557
4294966856	511793	82559	761249	3731509	23860927	902873	1030957	35983	747731	4517
4294966866	191449	2999	1073741717	2909	293	330382067	1345541	1327	611993	2290649
4294966876	1073741719	4294966877	5076793	911689	26843543	1431655627	58040093	138053	27531839	221219
4294966886	306783349	338213	23633	1931	3491843	8311	1073741723	911	50647	9651611
4294966896	1489	330382069	1246363	154423	6135667	113783	715827817	59791	22193	95443709
4294966906	5639	178207	357913909	4294966909	2029	1431655637	70051	13687	2797	858993383
4294966916	7307	10928669	82339	641327	1154561	36092159	8555711	453199	3271	36191
4294966926	715827821	4294966927	572357	8263	5436667	104755291	57943	12744709	35204647	21839
4294966936	18121	2403451	42107519	186737693	19522577	8297	306783353	4294966943	9151	858993389
4294966946	20069939	5990191	56512723	47197439	53923	9817	536870869	1431655651	2147483477	683
4294966956	1381907	353699	23087	397351	36497	447439	217643	87652387	1073741741	393853
4294966966	70067	226050893	59652319	97169	7621	1431655657	449	390451543	715827829	12113
4294966976	67108859	163487	2147483489	260791	947	4294966981	20297	1431655661	78341	41177
4294966986	12558383	569851	602887	1588963	98081	3427747	323027	18341	2147483497	193
4294966996	119159	4294966997	26107	16330673	4294967	110127359	4106087	554977	13256071	6458597
4294967006	11483869	65519	5835553	1597	143165567	22453	21913097	375467	12707003	858993403
4294967016	38149	795217	14412641	150053	180007	365747	5549053	17401	14128181	10141
4294967026	23917	18541	32537629	4294967029	11608019	5683	536870879	3764213	3392549	858993407
4294967036	2445881	1413283	8355967	20549	241	10710641	96703	9851	34636831	11767033
4294967046	16111	39163	141319	477218561	5009	21582749	357913921	37201	2879	1228889
4294967056	5347	252645121	83459	1755197	214748353	7121	4007	768193	564533	19976591
4294967066	15761	132967	2599859	2408843	143165569	65551	12201611	16086019	1779191	791699
4294967076	119304641	23761	5250571	110127361	107374177	417839	102261121	390451553	342283	12917
4294967086	2147483543	4294967087	1132639	613566727	429496709	21467	9209	11332367	21691753	395303
4294967096	76695841	1431655699	2147483549	96553	362	181967	66977	68174081	5197	9199
4294967106	1583	2135737	2609	1431655703	1252177	4294967111	182423	88463	675097	286331141
4294967116	63841	10058471	754297	15053	236507	102871	12859183	17321	81031	14753
4294967126	2147483563	21841	85639	161471	463319	12899	1073741783	80021	186689	129347
4294967136	13147	138547327	1467863	159072857	214748357	1163	2476913	4294967143	7321	576119
4294967146	93368851	13990121	2251031	2044249	85899343	77249	16381	33818639	70853	515293
4294967156	1073741789	1932059	2147483579	5156023	647	4294967161	34919	4886197	381707	858993433
4294967166	1262483	3229	1801	179383	184571	651839	357913931	9739	2147483587	7283
4294967176	87481	104755297	317159	62929	34981	3583	195225781	14767	6197	44171
4294967186	152293	204522247	5449	4294967189	143165573	82791	23342213	29333	7134497	45210181
4294967196	2521	4294967197	1296007	46182443	31033	11576731	238609289	26029	97612891	286331147
4294967206	1809169	91382281	18539	488009	25264513	477218579	292493	75167	65981	13109
4294967216	3780781	1431655739	52377649	11959	25411	39403369	813749	3089	1726273	231223
4294967226	22037	252645131	10631107	69779	13854733	4294967231	8191	3557	318949	22025473
4294967236	417961	825479	29389	70409299	23327	1431655747	145681	619139	45083	69001
4294967246	28111	65543	1877171	15966421	4111	1664071	1527371	1431655751	862097	4259
4294967256	59652323	82613	2147483629	4817	32771	281	19867	64103989	262657	95443717
4294967266	367531	37549	29587	35809	115861	3449	536870909	395959	6118187	171798691
4294967276	361651	1861711	18046081	4294967279	127	8951	795659	1226783	46684427	122713351
4294967286	715827881	65539	2089	1039	22605091	4294967291	331	464773	2147483647	65537

Timing of S(n).c Module for Calculation of Smarandache Function, version 1

Time taken to calculate S(n) depends on how easy it is to factor n. Less time is required if n has "small" prime factors. So, in the following table, the values of n shown are the mid-points of ranges (n-99 thru n+99). Times shown are for calculating S(n) for all integers in each range 10 times over:

n	time (ms)
100	268
200	308
400	345
800	387
1600	432
3200	490
6400	571
12800	661
25600	766
51200	919
102400	2450
204800	4036
409600	5670
819200	7977
1638400	10423
3276800	13004
6553600	16302
13107200	23438
26214400	29642
52428800	37011
104857600	50330
209715200	62363
419430400	77888
838860800	108179
1677721600	158480
3355443200	159111

Timing of S(n).c Module for Calculation of Smarandache Function, version 2

"Time to n" is the time taken to calculate S(n) for all n <= that shown. "Time add." is the time taken to calculate S(n) for all n > previous n and <= current n. All times are in milliseconds (as per version 1):

n	Time to n	Time add.
50000	18223	18223
100000	66763	48540
150000	139191	72428
200000	229634	90443
250000	335252	105618
300000	452539	117287
350000	579419	126880
400000	715146	135727
450000	859963	144816
500000	1012335	152372
550000	1171221	158886
600000	1336899	165678
650000	1508825	171927
700000	1686808	177983
750000	1870023	183215
800000	2058983	188961
850000	2252457	193473
900000	2450892	198435
950000	2653620	202728
1000000	2860734	207115
1050000	3072049	211314
1100000	3288502	216454
1150000	3509106	220603
1200000	3733965	224860
1250000	3962171	228206
1300000	4194158	231987
1350000	4429331	235173
1400000	4668560	239229
1450000	4910513	241953
1500000	5155601	245088
1550000	5404652	249051
1600000	5656512	251859
1650000	5911306	254794
1700000	6169686	258380
1750000	6431383	261697
1800000	6696172	264789
1850000	6963206	267034
1900000	7232974	269768
1950000	7505412	272438
2000000	7779763	274351
2050000	8056579	276816
2100000	8336442	279863
2150000	8620053	283611
2200000	8905641	285588
2250000	9194727	289086
2300000	9486449	291722
2350000	9780105	293655
2400000	10076920	296815
2450000	10375202	298282
2500000	10676383	301181

John McCarthy
17 Mount Street
Mansfield
Notts.
NG19 7AT
United Kingdom