XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

Un Algoritmo de Evaluación de Potencias utilizando Cadenas de Suma y Resta

Francois Morain *
Jorge Olivos †

RESUMEN

Se propone un algoritmo simple (autómata finito) para calcular x^n que utiliza multiplicaciones y una sóla división. Este algoritmo mejora, en términos relativos, en un 33% el clásico método binario [11], y en términos absolutos la reducción en el número de operaciones es de un 11.11%.

1. Introducción

Los métodos más recientes empleados en los test de primalidad y algoritmos de factorización de enteros [3, 8, 9, 11, 12, 13] requieren efectuar productos de números de gran tamaño. Esta clase de cálculos corresponde al problema de evaluación de potencias (x^m) y ha sido formalizado en la literatura introduciendo el concepto de *cadena de suma* [11]. Una cadena de suma para el entero positivo m es una secuencia de $(r+1)$ términos $(a_0, a_1, \ldots, a_r)$ tal que:

$$1 = a_0, a_1, \ldots, a_r = m, \tag{1}$$

y con la propiedad,

$$a_i = a_j + a_k, \quad \text{para} \quad k \leq j < i, \tag{2}$$

* Institut National de Recherche en Informatique et en Automatique (INRIA), Domaine de Voluceau, B. P. 105 78153 Le Chesnay CEDEX (France).

† Departamento de Ciencias de la Computación, Universidad de Chile, Casilla 2777. Santiago. Chile. Proyecto Fondecyt 348/87.

para todo $i = 1, 2, \ldots, r$. El largo más corto, r, para el cual existe una cadena de suma para m se denota por $l(m)$. El estudio de esta función, así como de sus generalizaciones, ha sido abordada por numerosos autores [2, 7, 10, 19, 21]. Todos los algoritmos conocidos que evalúan la función l, para un m cualquiera, son de tipo exponencial y más aún se conjetura que el cálculo de $l(m)$ es un problema NP–completo. El método binario [9, 11], ver sección **2**, es un algoritmo, no optimal, que se utiliza frecuentemente en la práctica.

Una *cadena de suma y resta* tiene la regla:

$$a_i = a_j \pm a_k \tag{3}$$

en lugar de (2). En términos prácticos, estas cadenas surgen cuando en la evaluación de x^m se puede utilizar multiplicaciones y divisiones. 31 es el número más pequeño para el cual existe una cadena de suma y resta de largo menor que $l(31)$ y esta es: 1, 2, 4, 8, 16, 32, 31. En el caso de los test de primalidad que motivan este trabajo (F. Morain, J. Olivos, Speeding up the Computations on an Elliptic Curve using Addition-Substractions Chains, Preliminary Report. INRIA), el costo de una resta equivale al de una suma justificando así el interés en estas últimas cadenas. El objetivo de este artículo es proponer un algoritmo muy simple para el cálculo de una cadena de suma y resta cuya complejidad (número de operaciones de suma y resta) esperada supera al método binario.

La sección **2** recuerda el método binario. La sección **3** presenta el nuevo algoritmo en sus dos variantes y en la sección **4** se estudia la complejidad esperada de ambas alternativas.

2. El método binario

Una manera razonable de calcular x^n se basa en la representación binaria de $n = b_r \ldots b_0$. Se asocia a la secuencia de bits una palabra del lenguaje $\{S, M\}^*$, reemplazando cada "1" por SM (salvo b_r) y cada "0" por S, donde S se interpreta como un doble (*squaring*) y M como una suma (*multiply*). Por ejemplo, si $n = 23$ su representación es 10111; y la palabra asociada es $SSMSMSM$. Los cálculos correspondientes son: x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}; para más detalles ver [11]. Una implementación en Pascal del algoritmo es la siguiente (ver [9]):

function *power*(x: *number*; n: *integer*): number;

```
begin
  if n = 0 then power := 1
  else if n = 1 then power := x
  else if (n mod 2) = 0
    then
      power := sqr(power(x, m div 2))
    else
      power := sqr(power(x, m div 2)) * b
end;
```

Si denotamos por $Q_{bp}(n)$ el número de multiplicaciones requeridos por este algoritmo en el cálculo de la enésima potencia de un número, entonces:

$$Q_{bp}(n) = \lambda(n) + \nu(n) - 1 \tag{1}$$

donde,

$$\lambda(n) = \lfloor lg_2 n \rfloor,$$
$$\nu(n) = Card\{i \mid 0 \leq i \leq \lambda(n),\ b_i = 1\}.$$

Si se considera la clase de los enteros n cuyo $\lambda(n)$ es constante se verifica que el número máximo y esperado de $\nu(n)$ es $\lambda(n)$ y $\frac{1}{2}\lambda(n)$, respectivamente. El Teorema siguiente resume el comportamiento del método binario,

Teorema 1. *Para los enteros n que pertenecen al intervalo $[2^s, 2^{(s+1)} - 1]$, se tiene que la complejidad en el caso peor es igual a $2\lambda(n)$ y en el esperado a $\frac{3}{2}\lambda(n)$.*

Observemos que el caso peor se realiza cuando $n = 2^{(s+1)} - 1$, esto es, cuando todos sus dígitos binarios son iguales a uno. Por otra parte, es evidente que todo algoritmo de evaluación de potencias tendrá una complejidad de al menos $\lambda(n)$.

3. Nuevo Algoritmo

El algoritmo siguiente reemplaza el cálculo de n por otros números auxiliares (b y c) verificando: $n + b = c$; de aquí se deduce que $n = c - b$. Esta última operación es la única resta (o división) presente en el algoritmo. Apoyándonos en la representación binaria de los tres números, la propiedad

que tienen los bits de b , iguales a uno, es de "voltear" los bloques de bits correspondientes de n. Estos últimos bloques deben ser de largo mayor o igual a tres. Los dos ejemplos siguientes ilustran el algoritmo.

Ejemplo 4.1 Se utiliza la notación ya mencionada. En primer lugar ilustramos lo que ocurre con el caso peor del método binario.

```
n:   1 1 1 1
b:       1
c:  10 0 0 0
```

a continuación, un caso más general:

```
n:  10011110111000111001100 1
b:        100010000010000000
c:  10100001000001000001100 1
```

en el primer caso el número de operaciones se reduce de 6 a 5 y en el segundo caso de 36 a 31.

Sea $C(n)$ el número de operaciones del nuevo algoritmo, se deduce de inmediato que:

$$C(n) = T(c) + \nu(b) \tag{1}$$

y por construcción:

$$C(b) = Q_{bp}(b). \tag{2}$$

donde surgen dos alternativas. En primer lugar, si c se calcula con el método binario (i.e. $T(c) = Q_{bp}(c)$), entonces el número esperado de operaciones en este caso es de $\frac{11}{8}\lambda(n)$ que significa una reducción relativa de un 25%. Si c se calcula siempre con el nuevo algoritmo, entonces la complejidad esperada se reduce a $\frac{4}{3}\lambda(n)$ y el correspondiente algoritmo representa una mejora relativa, en relación al binario, de un 33%. En términos absolutos la reducciones son de 8.5% y 11.11%, respectivamente.

Ambos algoritmos se visualizan fácilmente en término de autómatas finitos. A continuación se entrega una descripción gráfica de ambos algoritmos. La primera figura corresponde a la primera versión del algoritmo, esto es, cuando se ingresa al estado T11 (bloque de al menos dos unos) se comienzan a voltear los unos correspondientes y se sale de él cuando se lee

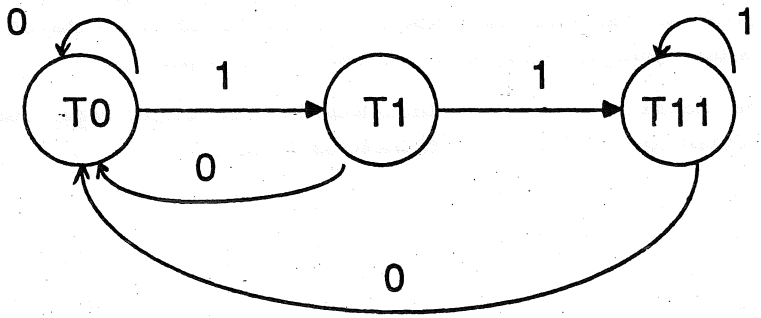

Fig. 1

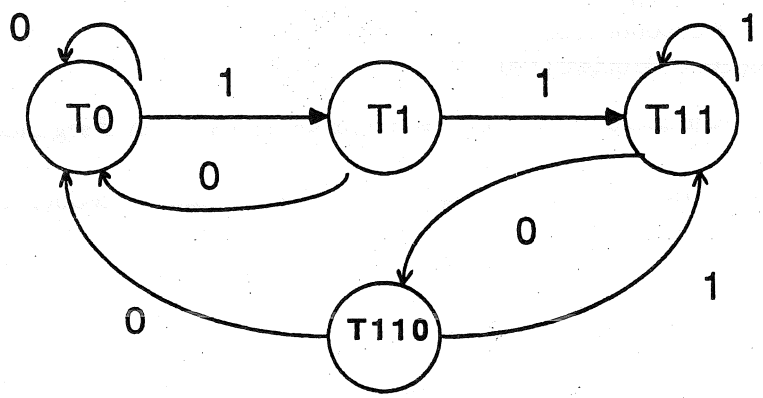

Fig. 2

un cero. En el segundo algoritmo, cuando se está en el estado T11 y se lee un cero se pasa al estado T110 que en caso de recibir un uno regresa al estado anterior (efectuando un puente, como se ilustró en el ejemplo anterior).

4. Análisis del Algoritmo.

En el anexo se entrega el análisis del algoritmo en su segunda versión utilizando el sistema $\Lambda\Upsilon\Omega$ que se desarrolla en el INRIA (P. Flajolet, B. Salvy, P. Zimmermann, Lambda-Upsilon-Omega: An Assistant Algorithms Analyser, AECC-6, Rome, 1988). Este sistema permite efectuar el análisis automático de clases de algoritmos bien definidas operando sobre estructuras de datos que admitan descomposiciones en término de estructuras más simples. Consiste en un Analizador Algebraico que compila especificaciones de algoritmos (en un lenguaje de alto nivel, pág. 1 del anexo) en funciones generatrices de costos promedios (págs. 2–4) y de un Analizador Analítico que extrae información asintótica sobre los coeficientes de las funciones generatrices (págs. 4–7) obtenidas en la fase previa. La parte analítica se basa en el sistema de cálculo simbólico MAPLE, desarrollado en Waterloo.

Se observa que el algoritmo es una traducción inmediata del gráfico de la sección previa y en la página 4 del anexo se encuentra el resultado asintótico (función av_tau_treat_n) correspondiente.

5. Resultados experimentales.

El primer autor ha utilizado la primera versión del algoritmo en la implementación del test de Atkins (F. Morain, Implementation of the Goldwaser-Killian-Atkin test. Preliminary Report. INRIA). Trabajando con números de 100 dígitos decimales se ha obtenido una ganancia en tiempo en el cálculo de la ley sobre una curva elíptica que concuerda con lo anunciado por la teoría.

6. Agradecimientos.

El segundo autor quisiera expresar su reconocimiento al INRIA por una invitación como profesor visitante que le permitió abordar este trabajo. Este trabajo contó tambien con el apoyo del programa de cooperación Franco–Chileno.

Ambos autores agradecen a P. Zimmermann por su excelente disposición a explicar el uso del sistema $\Lambda\gamma^{\Omega}$ que se encuentra actualmente en su fase final de desarrollo.

7. Bibliografía

1. W. Bosma. *Primality testing using elliptic curves.* Report 85–12, Math. Institut, Universeit van Amsterd.

2. A. Brauer. *On addition chains.* Bull. Amer. Math. Soc., 45, p. 736–739.

3. R.P. Brent. *Some integer factorization algorithms using elliptic curves.* Research Report CMA-R32-85, The Australian national University, Canberra, 1985.

4. J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, S.S. Wagstaff Jr. *Factorizations of bn 1. Contemporary Mathematics*, 22, AMS, 1983.

5. J.W.O.S. Cassels. *Diophantine equations with special references to elliptic curves.* J. London Math. Soc., 41, 1966, p. 193–291.

6. D.V. Chudnovsky, G. V. Chudnovsky. *Sequences of numbers generated by addition in formal groups and new primality and factorization tests.* Research report RC 11262, IBM, Yorktown Heights, 1985.

7. P.Erdős. *Remarks on number theory III : on addition chains.* Acta Arithmetica, 6, 1960, p. 77–81.

8. S. Goldwasser, J. Killian. *Almost all primes can be quickly certified.* Proc. 18th STOC, Berkely, 1986, p. 316–329

9. G.H.Gonnet. *Handbook of Algorithms and Data Structures.* Addison-Wesley. 1984.

10. B.S. Kaliski, Jr. *A pseudo-random bit generator based on elliptic logarithms.* Proc. Crypto 86, p. 13–1, 13–21.

11. D.E. Knuth. *Seminumerical algorithms.* The art of computer programming, Vol. 2, Addison-Wesley. 2o. Ed. 1980.

12. N. Koblitz. *Elliptic curve cruptosystems.* Math. of Comp., v 48, 177, jan 1987, p. 203–209.

13. H.W. Lenstra, Jr. *Factoring with elliptic curves.* Report 86–18, Math. Inst., Univ. Amsterdam, 1986.

14. H.W. Lenstra, Jr. *Elliptic curves and number theoretic algorithms.* Report 86–19, Math. Inst., Univ. Amsterdam, 1986.

15. D.P. McCarthy. *The optimal algorithm to evaluate x^n using elementary multiplication methods.* Math. of Comp., v 31, 137, jan 1977, p. 251–256.

16. D.P. McCarthy. *Effect of improved multiplication efficiency on exponentiation algorithms derived from addition chains.* Math. of Comp., v 46, 174, apr 1986, p. 603–608.
17. P.L. Montgomery. *Modular multiplication without trial division.* Math. of Comp., v 44, 170, apr 1985, p. 519–521.
18. J. Olivos. *On Vectorial Addition Chains.* Journal of Algorithms, 2, 1981, p. 13–21.
19. J.M. Pollard. *Theorems on factorization and primality testing.* Proc. Cambridge Phil. Soc., 76, 1974, p. 521–528.
20. R.L. Rivest, A. Shamir, L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* Comm. of the ACM, v 21, 2, feb 1978, p. 120–126.
21. A. Schönhage. *A lower bound for the length of addition chains.* Theor. Comput. Science, 1, 1, 1975, p. 1–12.
22. J.T.Tate. *The arithmetic of elliptic curves.* Inventiones Math., 23, 1974, p. 179–206.

```
*************************************************************************
type chain = sequence(bit);     % A + B = C then A = C - B %
     valid_chain = product(chain,one);
     bit = zero | one;
     zero,one = atom(1);

procedure treat(c:valid_chain);
case c of
  (c1,one) : begin treatStart(c1); divide end
end;

procedure treatStart(c:chain);
case c of
  () : nil;
  (zero,c1) : begin treat0(c1) end;
  (one,c1)  : begin treat1(c1) end
end;

procedure treat0(c:chain);
case c of
  () : begin squaring; multiplyC end;
  (zero,c1) : begin squaring; treat0(c1) end;
  (one,c1)  : begin squaring; treat1(c1) end
end;

procedure treat1(c:chain);      % one 1 has been recognized %
case c of
  () : begin multiplyC; multiplyC end;
  (zero,c1) : begin multiplyC; squaring; treat0(c1) end;
  (one,c1) : begin squaring; multiplyB; treat11(c1) end
end;

procedure treat11(c:chain);     % at least two 1 have been recognized %
case c of
  () : begin squaring; squaring; multiplyC end;
  (zero,c1) : begin squaring; treat110(c1) end;
  (one,c1)  : begin squaring; treat11(c1) end
end;

procedure treat110(c:chain); % at least two 1 and one 0 have been recognized %
case c of
  () : begin multiplyB; squaring; multiplyC end;
  (zero,c1) : begin multiplyC; squaring; treat0(c1) end;
  (one,c1)  : begin multiplyB; squaring; treat11(c1) end
end;

procedure naive(c:chain);
case c of
  () : nil;
  (zero,c1) : begin squaring; naive(c1) end;
  (one,c1)  : begin squaring; multiply; naive(c1) end
end;

measure multiplyB : 1;
        multiplyC : 1;
```

```
        multiply  : 1;
        divide    : 1;
        squaring  : 1;
*************************************************************************

Script started on Wed May 25 18:31:42 1988
yquem% maplecaml
   CAML (sun) (V 2-5) by INRIA Fri Jan 22

#load"luo";;

#order:=2;; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
2 : num

#analyze"chains2";;
Generating functions are ordinary.

Counting generating functions:
  chain=Q(bit)
  valid_chain=chain*one
  bit=zero+one
  zero=z
  one=z

Complexity descriptors:
  tau_treat=tau_treatStart*one+1*chain*one
  tau_treatStart=zero*tau_treat0+one*tau_treat1
  tau_treat0=1*1+1*1+1*zero*Q(bit)+zero*tau_treat0+1*one*Q(bit)+one*tau_treat1
  tau_treat1=1*1+1*1+1*zero*Q(bit)+1*zero*Q(bit)+zero*tau_treat0+1*one*Q(bit)+1*
one*Q(bit)+one*tau_treat11
  tau_treat11=1*1+1*1+1*1+1*zero*Q(bit)+zero*tau_treat110+1*one*Q(bit)+one*tau_t
reat11
  tau_treat110=1*1+1*1+1*1+1*zero*Q(bit)+1*zero*Q(bit)+zero*tau_treat0+1*one*Q(b
it)+1*one*Q(bit)+one*tau_treat11
  tau_naive=1*zero*Q(bit)+zero*tau_naive+1*one*Q(bit)+1*one*Q(bit)+one*tau_naive

Solving generating functions ...
                                  one = z

                                 zero = z

                           valid_chain = Q(2 z) z

                                  bit = 2 z

                                chain = Q(2 z)

Solving complexity descriptors ...
tau_treat0

                                2
    = - (- one Q(bit) + 2 one  zero Q(bit) - 2 - one zero Q(bit) - zero Q(bit)

          2                              .2          3                 . 2
      - one  Q(bit) + 2 one zero - one  zero + one  Q(bit) + one zero  Q(bit)
```

$$- \text{one}^2)$$

$$/ (\text{one zero}^2 + \text{one}^2 \text{ zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

tau_treatStart

$$= - (- 3 \text{ one zero } Q(\text{bit}) + 2 \text{ one}^2 \text{ zero } Q(\text{bit}) - 2 \text{ zero} - \text{zero}^2 Q(\text{bit})$$

$$+ \text{one}^2 \text{ zero } Q(\text{bit}) + 2 \text{ one zero}^2 + \text{one}^3 \text{ zero } Q(\text{bit}) + \text{one zero}^3 Q(\text{bit})$$

$$+ \text{one}^2 \text{ zero} - 2 \text{ one} - 2 \text{ one}^2 Q(\text{bit}) + \text{one}^3 Q(\text{bit}) - \text{one}^2)$$

$$/ (\text{one zero}^2 + \text{one}^2 \text{ zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

tau_treat11

$$= - (- \text{one zero } Q(\text{bit}) + \text{one}^2 \text{ zero } Q(\text{bit}) + \text{zero}^3 Q(\text{bit})$$

$$+ 2 \text{ one zero}^2 Q(\text{bit}) - \text{zero } Q(\text{bit}) - \text{zero}^2 Q(\text{bit}) - \text{one } Q(\text{bit}) + \text{zero}^2$$

$$+ \text{one zero}^2 + 3 \text{ one zero} - 3)$$

$$/ (\text{one zero}^2 + \text{one}^2 \text{ zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

tau_treat110

$$= - (- 3 - 2 \text{ one } Q(\text{bit}) - 2 \text{ zero } Q(\text{bit}) + 2 \text{ one zero } Q(\text{bit}) + \text{zero}^2 Q(\text{bit})$$

$$+ \text{zero} + \text{one}^2 Q(\text{bit}) + 3 \text{ one zero} - \text{one}^2 \text{ zero})$$

$$/ (\text{one zero}^2 + \text{one}^2 \text{ zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

tau_treat1

$$= - (\text{zero}^2 Q(\text{bit}) + 2 \text{ one zero } Q(\text{bit}) - 2 \text{ zero } Q(\text{bit}) - 2 - 2 \text{ one } Q(\text{bit})$$

$$+ 2 \text{ one zero} + \text{one}^2 Q(\text{bit}) + \text{one zero}^2 - \text{one})$$

$$/ (\text{one zero}^2 + \text{one}^2 \text{ zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

$$/\ (\text{one zero} + \text{one zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

$$\text{tau\_naive} = -\ \frac{Q(\text{bit})\ (\text{zero} + 2\ \text{one})}{-\ 1 + \text{zero} + \text{one}}$$

tau_treat

$$= \text{one}$$

$$(3\ \text{one zero}^2\ Q(\text{bit}) - 2\ \text{one}^2\ \text{zero}^2\ Q(\text{bit}) + 2\ \text{zero} + \text{zero}^2\ Q(\text{bit})$$

$$- \text{one}^2\ \text{zero}\ Q(\text{bit}) - 2\ \text{one}^2\ \text{zero}^2 - \text{one}^3\ \text{zero}\ Q(\text{bit}) - \text{one zero}^3\ Q(\text{bit})$$

$$- \text{one}^2\ \text{zero} + 2\ \text{one} + 2\ \text{one}^2\ Q(\text{bit}) - \text{one}^3\ Q(\text{bit}) + \text{one}^2$$

$$+ \text{chain one zero}^2 + \text{chain one}^2\ \text{zero} + \text{chain} - \text{chain one zero}$$

$$- \text{chain zero} - \text{chain one})$$

$$/\ (\text{one zero}^2 + \text{one}^2\ \text{zero} + 1 - \text{one zero} - \text{zero} - \text{one})$$

Average case analysis of function treat:
==========

1) Number of arguments of treat of size n is:

$$1/2\ 2^n + O((1/n)^{\text{infinity}}\ 2^n)$$

2) Total cost of treat on all arguments of size n is:

$$2/3\ 2^n\ n + \frac{17}{36}\ 2^n + O(\frac{2^n}{n})$$

3) Average cost of treat on arguments of size n is:

$$\text{av\_tau\_treat\_n} := 4/3\ n + \frac{17}{18} + O(1/n)$$

4) Floating point evaluation :

$$1.33333\ n + .944444 + O(1/n)$$

Average case analysis of function treatStart:
==========

1) Number of arguments of treatStart of size n is:

$$2^n + O((1/n)^{\infty})\,2^n$$

2) Total cost of treatStart on all arguments of size n is:

$$4/3\ 2^n\ n + \frac{23}{18}\ 2^n + O(\frac{2^n}{n}).$$

3) Average cost of treatStart on arguments of size n is:

$$av\_tau\_treatStart\_n := 4/3\ n + \frac{23}{18} + O(1/n)$$

4) Floating point evaluation :

$$1.33333\ n + 1.27778 + O(1/n)$$

Average case analysis of function treat0:
==========

1) Number of arguments of treat0 of size n is:

$$2^n + O((1/n)^{\infty})\,2^n$$

2) Total cost of treat0 on all arguments of size n is:

$$4/3\ 2^n\ n + \frac{41}{18}\ 2^n + O(\frac{2^n}{n})$$

3) Average cost of treat0 on arguments of size n is:

$$av\_tau\_treat0\_n := 4/3\ n + \frac{41}{18} + O(\frac{(1/2)^{(-n)}}{(1/2)^n\ n})$$

4) Floating point evaluation :

$$1.33333\ n + 2.27778 + O(\frac{(1/2)^{(-n)}}{(1/2)^n\ n})$$

Average case analysis of function treat1:
==========

1) Number of arguments of treat1 of size n is:

$$2^n + O((1/n)^{\infty})\,2^n$$

2) Total cost of treatl on all arguments of size n is:

$$4/3\ 2^n + \frac{53}{18}\,2^n + O\left(\frac{2^n}{n^2}\right)$$

3) Average cost of treatl on arguments of size n is:

$$av\_tau\_treatl\_n := 4/3\ n + \frac{53}{18} + O\left(\frac{1}{n^2}\right)$$

4) Floating point evaluation :

$$1.33333\ n + 2.94444 + O\left(\frac{1}{n^2}\right)$$

Average case analysis of function treatll:

1) Number of arguments of treatll of size n is:

$$2^n + O((1/n)\ 2^{infinity})^n$$

2) Total cost of treatll on all arguments of size n is:

$$4/3\ 2^n + \frac{41}{18}\,2^n + O\left(\frac{2^n}{n}\right)$$

3) Average cost of treatll on arguments of size n is:

$$av\_tau\_treatll\_n := 4/3\ n + \frac{41}{18} + O(1/n)$$

4) Floating point evaluation :

$$1.33333\ n + 2.27778 + O(1/n)$$

Average case analysis of function treatll0:

1) Number of arguments of treatll0 of size n is:

$$2^n + O((1/n)\ 2^{infinity})^n$$

2) Total cost of treatll0 on all arguments of size n is:

$$4/3\ 2^n\ n + \frac{53}{18}\ 2^n + O\left(\frac{2^n}{n}\right)$$

3) Average cost of treat110 on arguments of size n is:

$$av\_tau\_treat110\_n := 4/3\ n + \frac{53}{18} + O\left(\frac{1}{2^n}\right)$$

4) Floating point evaluation :

$$1.33333\ n + 2.94444 + O\left(\frac{1}{2^n}\right)$$

Average case analysis of function naive:
==========

1) Number of arguments of naive of size n is:

$$2^n + O((1/n)^{infinity}\ 2^n)$$

2) Total cost of naive on all arguments of size n is:

$$3/2\ 2^n\ n + O((1/n)^{infinity}\ 2^n)$$

3) Average cost of naive on arguments of size n is:

$$av\_tau\_naive\_n := 3/2\ n$$

4) Floating point evaluation :

$$1.50000\ n$$

() : void

#quit();;
A bientot ...
yquem%
script done on Wed May 25 18:36:49 1988