Riv. Mat. Univ. Parma (7) **3*** (2004), 289-299

Ottavio G. Rizzo [*]

On the complexity of the 2^k-ary and of the sliding window algorithms for fast exponentiation [**]

Let E be an (additive) monoid and suppose we want to compute nP where $n \in \boldsymbol{N}$ and $P \in E$. The obvious way, summing n copies of P, is not very efficient: it requires $n - 1$ additions. For example, $12P = 2(2(2P + P))$ uses only one addition and three doublings, rather than eleven additions.

The problem of computing a scalar multiple (or the *power*, in a multiplicative setting) using few additions/doublings is the same as finding an *addition chain* of small length. We recall that an addition chain for $n \in \boldsymbol{N}$ is a sequence $a_1 = 1 < a_2 < \ldots < a_\ell = n$ such that $a_i = a_j + a_k$ with $j, k < i$ for every $i = 2, \ldots, \ell$ (see [**9**] for a very thorough introduction). Finding the shortest addition chain for a given integer n is commonly considered to be a very hard problem (although it has not been proved it is NP-hard, as often – but erroneously – reported in the literature referring to [**5**]: see the remarks in [**1**] about this misconception); so in many applications it is important to find in a reasonable time a chain which is reasonably short.

A few words on what «reasonably short» means: let $\lambda(n) = \lfloor \log_2(n) \rfloor$; using Algorithm 4, we can find an addition chain for n of length $\lambda(n) + (1 + o(1))\, \lambda(n)/\lambda(\lambda(n))$. On the other hand, Erdős proved in [**6**] that, asymptotically, for almost every n, the shortest addition chain for n has length $\lambda(n) + \lambda(n)/\lambda(\lambda(n))$.

(*) Dipartimento di Matematica, Università di Milano, Via Saldini 50, 20133 Milano, Italy; e-mail: Ottavio.Rizzo@mat.unimi.it

(**) Received March 1st 2004 and in revised form January 21st 2005. AMS classification 11 Y 16.

In practical applications, there is one further constraint: one cannot store too many intermediate steps! In the simplest case, we allow only the two following operations: $a_i = 2a_{i-1}$ or $a_i = a_{i-1} + 1$. Such an algorithm, as the one we are presenting in the first section, needs to store just one value. We can improve the algorithm (i.e., shorten the length of the chain) by precalculating some values: if $\mathcal{P} \subset \boldsymbol{N}$ is a finite subset, we say that a given addition chain is based on $\mathcal{P}$ if it involves only operations of the form $a_i = 2a_{i-1}$ or $a_i = a_{i-1} + a_j$ with $a_j \in \mathcal{P}$. Clearly, one has to store $1 + \#\mathcal{P}$ values in order to execute the algorithm.

As a final remark, we will not try to give attributions for the algorithms (a variant of Algorithm 1 was already known to Egyptian mathematicians in 1800 b.C.!): the interested reader may consult [**1**], which provides an excellent bibliography. Furthermore, the main terms of our theorems are of course known (see, for example [4] for Theorem 7; the main term of Theorem 15 appears in [**10**], although the error there is $O(1)$): our goal for this paper is to give a description as good as possible of the small terms. Finally, we have been informed after our presentation that H. Cohen gives similar results in [3] about a right-to-left sliding window algorithm.

1 - The left binary algorithm

Given $n \in \boldsymbol{N}$, let $n = \sum_{i=0}^{\lambda} n_i 2^i$ be its binary expansion, where $n_i \in \{0, 1\}$, $n_\lambda = 1$.

Algorithm 1. The left binary algorithm.

```
1) let $n = (n_\lambda n_{\lambda-1} \dots n_0)_2$, $Q = P$
2) for $i = (\lambda - 1) \dots 0$:
3)     let $Q = 2Q$
4)   if $n_i = 1$: let $Q = Q + P$
5)   return $Q$
```

We want to analyse the complexity of the algorithm, under the assumption that every operation different from the doubling and the addition of points in E is negligible in terms of time and memory. In particular, we are interested in the average number of doublings and additions when n runs over all integers of exactly e bits, that is $\lambda(n) = e - 1$. Let $c_D(n)$, resp. $c_A(n)$, be the number of doublings, resp. additions, that the algorithm requires for a given $n \in \boldsymbol{N}$, not including precomputations. If $\mathcal{P}$ is the set of values that need to be precomputed, let $C_D(\mathcal{P})$

$= \#\{$doublings required by $\mathcal{P}\}$ and define

$$C_D^0(e) = \frac{1}{2^{e-1}} \sum_{\lambda(n)=e-1} c_D(n), \qquad C_D(e) = C_D^0(e) + C_D(\mathcal{P}),$$

as the expected number of doublings (not including, resp. including precomputations) required for a random integer of e bit.

Analogously for $C_A^0(e)$, $C_A(\mathcal{P})$ and $C_A(e)$. It is clear that, in the case of Algorithm 1, $c_D(n) = \lambda(n)$ and $c_A(n) = w(n) - 1$, where $w(n) = \#\{n_i \neq 0\}$ is the *Hamming weight* of n. Since $\mathcal{P} = \{1\}$, i.e. there are no precomputations, $C_D(\mathcal{P}) = C_A(\mathcal{P}) = 0$. It is then a simple exercise on induction (but cf. the proof of Theorem 7) to prove that:

Theorem 2. *For Algorithm* 1: $C_D(e) = e - 1$ *and* $C_A = (e-1)/2$ *for any* $e \in \boldsymbol{N}$.

The binary algorithm can been improved in many ways, but the number of doublings is (essentially) unavoidable. The variants we will present will reduce the number of additions, at the cost of precomputing (and storing!) some more values: in other words, given Erdős's estimate, the goal is to have $C(\mathcal{P}) + C_A(e) \sim e/\lambda(e)$ for $e \gg 1$.

2 - The 2^k-ary algorithm

Let k be a positive integer, which we will usually suppose > 1. The idea is to write n in the base 2^k: $n = \sum_{i=0}^{\ell} n_i 2^{ki}$ where $n_i \in \{0, \ldots, 2^k - 1\}$, $n_\ell \neq 0$ and $\ell = \lfloor \lambda(n)/k \rfloor$.

Algorithm 3. The 2^k-ary algorithm, simple form.

```
1) for $i = 2, \ldots, 2^k - 1$: store $iP$
2) let $n = \sum_{i=0}^{\ell} n_i 2^{ki}$, $Q = n_\ell P$
3) for $i = (\ell - 1) \ldots 0$:
4)   let $Q = 2^k Q$
5)   if $n_i \neq 0$: let $Q = Q + n_i P$
6) return $Q$
```

Notice that $C_D(\mathcal{P}) = 1$, $C_A(\mathcal{P}) = 2^k - 3$ and $2^k - 2$ multiples of P need to be stored—besides P, of course. We can cut these requirements in half if we note that we do not really need to store the even multiples:

Algorithm 4. The 2^k-ary algorithm, better form.

```
1) for $i = 3, 5, \dots, 2^k - 1$: store $iP$
2) let $n = \sum_{i=0}^{\ell} n_i 2^{ki}$, let $n_i = \nu_i 2^{\varepsilon_i}$ // where $\nu_i$ is odd and $\varepsilon_i = 0$ if $n_i = 0$
3) let $Q = \nu_\ell P$; let $Q = 2^{\varepsilon_\ell} Q$
4) for $i = (\ell - 1) \dots 0$:
5)   let $Q = 2^{k - \varepsilon_i} Q$
6)   if $n_i \neq 0$:
7)     let $Q = Q + \nu_i P$
8)     let $Q = 2^{\varepsilon_i} Q$
9) return $Q$
```

Let $\varepsilon(n) = \varepsilon_\ell(n)$; i.e., $\varepsilon(n) = v_2(\lfloor n/2^{k\ell} \rfloor)$, where v_2 is the 2-adic valuation. It is clear that $c_A(n)$ is the same for both algorithms. On the other hand, Algorithm 4 will require exactly $\varepsilon(n)$ more doublings than Algorithm 3. Although in practice only Algorithm 4 would be used, we will study the complexity of its simpler version and use the following proposition to deduce its complexity.

Proposition 5. *Write* $i \bmod k$ *for* $i - \lfloor i/k \rfloor k$. *Then, for every integer* $e \geq 1$:

$$\frac{1}{2^{e-1}} \sum_{\lambda(n) = e-1} \varepsilon(n) = 1 - 2^{-((e-1) \bmod k)}.$$

Proof. If $n_\ell = \lfloor n/2^{\ell k} \rfloor$ and $n' = n - n_\ell 2^{\ell k}$ then

$$n' = n - \left\lfloor \frac{n}{2^{\ell k}} \right\rfloor 2^{\ell k} = \left(\frac{n}{2^{\ell k}} - \left\lfloor \frac{n}{2^{\ell k}} \right\rfloor \right) 2^{\ell k} \in [0, 2^{\ell k}),$$

and

$$\lambda(n_\ell) = \lfloor \log_2 \left((n - n') 2^{-\ell k} \right) \rfloor = \lfloor \log_2 (n - n') \rfloor - \ell k = e - 1 - \ell k .$$

Write $\tilde{e}$ for $e - 1 - \ell k$. Then, thanks to Lemma 6,

$$\sum_{\lambda(n) = e-1} \varepsilon(n) = \sum_{\lambda(n_\ell) = \tilde{e}} \sum_{n'=0}^{2^{\ell k} - 1} \varepsilon(n_\ell 2^{\ell k} + n') = \sum_{\lambda(n_\ell) = \tilde{e}} \sum_{n'=0}^{2^{\ell k} - 1} \varepsilon(n_\ell 2^{\ell k})$$

$$= 2^{\ell k} \sum_{\lambda(n_\ell) = \tilde{e}} v_2(n_\ell) = 2^{\ell k}(2^{\tilde{e}} - 1) = 2^{e-1} - 2^{\ell k}.$$

The proposition follows immediately, once we notice that $\tilde{e} = (e-1 \bmod k)$. ■

Lemma 6. *For every integer $e \geqslant 0$ we have* $\sum_{\lambda(n)=e} v_2(n) = 2^e - 1$.

Proof. Denote $\theta(e)$ such a sum. Clearly, $\theta(0) = v_2(1) = 0$. Suppose now that $e > 0$; if n is odd, then $v_2(n) = 0$, thus

$$\theta(e) = \sum_{\lambda(n)=e} v_2(n) = \sum_{\lambda(n')=e-1} v_2(2n') = \sum_{\lambda(n')=e-1} 1 + v_2(n') = 2^{e-1} + \theta(e-1).$$

It is straightforward to verify that only $\theta(e) = 2^e - 1$ satisfies these inductive properties. ■

Notice that in Algorithm 3 additions occur only in line 5, and only when $n_i \neq 0$ with $0 \leqslant i < \ell$. Thus, if $w_k(n) = \#\{n_i \neq 0$, where $n = \sum n_i 2^{ki}\}$, the number of additions is $c_A(n) = w_k(n) - 1$, for every $n > 0$. In the same algorithm, k doublings are executed each time line 4 is run, which happens ℓ times; thus $c_D(n) = \ell k$. Therefore, considering Proposition 5, averaging, and applying Lemma 8, we get:

Theorem 7. *If $e > k$, we have for Algorithm* 4:

$$C_A^0(e) = \left\lfloor \frac{e-1}{k} \right\rfloor (1 - 2^{-k}), \qquad C_D^0(e) = \left\lfloor \frac{e-1}{k} \right\rfloor k + 1 - 2^{-((e-1) \bmod k)};$$

$$C_A(\mathcal{P}) = 2^{k-1} - 1 \qquad\qquad C_D(\mathcal{P}) = 1.$$

Lemma 8. *If $1 \leqslant e \leqslant k$, then* $2^{1-e} \sum_{\lambda(n)=e-1} w_k(n) = 1$; *while if $e > k$,*

$$\frac{1}{2^{e-1}} \sum_{\lambda(n)=e-1} w_k(n) = 1 + \left\lfloor \frac{e-1}{k} \right\rfloor (1 - 2^{-k}).$$

Proof. Write $\omega(e)$ for such a sum. Since $w_k(n) = 1$ for every $n \in \boldsymbol{N}$ with $\lambda(n) < k$, we get $\omega(e) = 1$ for every e such that $1 \leqslant e \leqslant k$.

If $n > 2^k$, write $n = n_\ell 2^{\ell k} + n'$ as in the proof of Proposition 5; then $w_k(n) = 1 + w_k(n')$. Thus, if $e > k$ and $\tilde{e} = e - 1 - \ell k$,

$$2^{e-1}\omega(e) = \sum_{\lambda(n_\ell)=\tilde{e}} \sum_{n'=0}^{2^{\ell k}-1} 1 + w_k(n') = 2^{e-1} + 2^{\tilde{e}} \sum_{n'=0}^{2^{kl}-1} w_k(n').$$

It is an easy exercise to check that $\sum_{n=0}^{2^{\ell k}-1} w_k(n) = \ell(2^k-1)2^{k(\ell-1)}$. Hence, $\omega(e) = 1 + 2^{1-e}2^{\tilde{e}} \cdot 2^{\ell k} \cdot \ell(2^k-1) = 1 + \lfloor (e-1)/k \rfloor (1-2^{-k})$. ■

Remark 9. For highest efficiency, the parameter k should be chosen (see page 11 of [4]) to be the smallest integer such that

$$e \leqslant \frac{k(k+1)2^{2k}}{2^{k+1}-k-2} + 1\,.$$

Remark 10. Moreover, if a larger e is desirable, e should be chosen divisible by k. Indeed, if $e = \ell k + 1$, $e' = k(\ell+1)$, $e'' = e' + 1$, then Theorem 7 yields:

$$C_A(e') = C_A(e), \qquad C_D(e') = C_D(e) + 1 - 2^{1-k};$$
$$C_A(e'') = C_A(e) + 1 - 2^{-k}, \quad C_D(e'') = C_D(e) + k.$$

3 - Sliding windows (left-to-right)

We can further improve the 2^k-ary algorithm if we notice that an addition is performed each time a window of length k contains a non-zero bit; thus, if we allow ourselves more freedom in positioning these windows (we let them «slide»), we might end up with fewer additions, especially if k is small. For example, if $k = 3$ and $n = 791 = 1.100.010.111_2$, Algorithm 4 requires three additions, but if we subdivide $n = 11.000.101.11_2$, just two additions are required.

Before we introduce the algorithm, we explicitly state how we want to subdivide the binary digits of n (and we suppose that the time to do this is negligible):

Lemma 11. *Fix k. Every integer $n \in \boldsymbol{N}$ can be written in a unique way as $\sum_{i=0}^{d} \nu_i 2^{\varepsilon_i}$, where $\nu_i \in \mathcal{P} = \{1, 3, 5, \ldots, 2^k-1\}$, $e-1 \geqslant \varepsilon_0 > \varepsilon_1 > \ldots > \varepsilon_d \geqslant 0$ and, for $0 \leqslant i < d$, $\varepsilon_i + \lambda(\nu_i) - \varepsilon_{i+1} \geqslant k$.*

Algorithm 12.

```
1) for $i = 3, 5, \ldots, 2^k-1$: store $iP$
2) let $n = \sum_{i=0}^{d} \nu_i 2^{\varepsilon_i}$, $\varepsilon_{d+1} = 0$ // where the $\nu_i$ and $\varepsilon_i$ are as in Lemma 11
3) let $Q = \nu_0 P$; let $Q = 2^{\varepsilon_0 - \varepsilon_1} Q$
4) for $i = 1 \ldots d$:
5)   let $Q = Q + \nu_i P$
6)   let $Q = 2^{\varepsilon_i - \varepsilon_{i+1}} Q$
7) return $Q$
```

The algorithm is valid since it is clear that its output is $\sum_{i=0}^{d} \nu_i 2^{\varepsilon_i} P$. We have $c_A(n) = d$, while $c_D(n) = (\varepsilon_0 - \varepsilon_1) + (\varepsilon_1 - \varepsilon_2) + \ldots + (\varepsilon_d - 0) = \varepsilon_0$.

Lemma 13. *Let $d(n)$, resp. $\varepsilon_0(n)$, be the value of d, resp. ε_0, as in Lemma 11; let $\delta_e = \sum_{n=1}^{2^e-1} d(n)$ and $\eta_e = \sum_{\lambda(n)=e-1} \varepsilon_0(n)$. Then $\delta_e = 0$ for $e \leqslant k$, while if $e > k$,*

$$\delta_e = \delta_{e-1} + 2^{k-1}\delta_{e-k} + 2^{e-1} - 2^{k-1}, \qquad \eta_e = 2^{e-1}(e-k+1) - 2^{e-k}.$$

Proof. Using the notation of Lemma 11, let $n_0 = \nu_0 2^{\varepsilon_0 + k - e}$. Then $n = n_0 2^{e-k} + n'$ where $2^{k-1} \leqslant n_0 < 2^k$ and $0 \leqslant n' < 2^{e-k}$—this corresponds to taking the first window of length exactly k. If $n' = 0$, then $d(n) = 0$; otherwise, since $n' = \sum_{i=1}^{d} \nu_i 2^{\varepsilon_i} > 0$, it is clear that $d(n) = 1 + d(n')$. Therefore:

$$\delta_e - \delta_{e-1} = \sum_{\lambda(n_0)=k-1} d(n_0 2^{e-k} + n')$$

$$= \sum_{\lambda(n_0)=k-1} \sum_{n'=0}^{2^{e-k}-1} 1 + d(n') = 2^{k-1}\big(2^{e-k} - 1 + \delta_{e-k}\big),$$

which gives the first part. Now, since ν_0 is odd, $v_2(n_0) = \varepsilon_0 + k - e$. Thus, if $e > k$,

$$\eta_e = \sum_{\lambda(n_0)=k-1} \sum_{n'=0}^{2^{e-k}-1} \varepsilon_0(n_0 2^{e-k} + n') = \sum_{\lambda(n_0)=k-1} \sum_{n'=0}^{2^{e-k}-1} v_2(n_0) + e - k$$

$$= 2^{e-k}2^{k-1}(e-k) + 2^{e-k} \sum_{\lambda(n_0)=k-1} v_2(n_0),$$

which, by Lemma 6, gives $\eta_e = 2^{e-1}(e-k+1) - 2^{e-k}$. ■

Proposition 14. *Let $F(z) = \sum_{e=0}^{\infty} \delta_e z^e$ be the generating function of δ_e. Then*

$$F(z) = \frac{2^{k-1} z^{k+1}}{(1-2z)^2(1-z)\,p(z)}$$

where $p(z) \in \mathbf{Z}[z]$ has degree $k-1$, all roots have norm $> 1/2$, there is one real root if k is even (in this case the root is $< -1/2$) and none if k is odd.

Proof. By the previous lemma,

$$F(z) = \sum_{e=0}^{\infty} \delta_{e-1} z^e + 2^{k-1} \sum_{e=0}^{\infty} \delta_{e-k} z^e + \sum_{e=k}^{\infty} 2^{e-1} z^e - 2^{k-1} \sum_{e=k}^{\infty} z^e$$

$$= \left(z + 2^{k-1} z^k\right) F(z) + \frac{2^{k-1} z^{k+1}}{(1-2z)(1-z)} .$$

Thus,

$$F(z) = \frac{2^{k-1} z^{k+1}}{(1-2z)(1-z)(1-z-2^{k-1} z^k)} .$$

We have $1 - z - 2^{k-1} z^k = (1-2z)\left(1 + z + 2z^2 + \ldots + 2^{k-2} z^{k-1}\right)$; let $q(z)$ be the LHS and $p(z) = q(z)/(1-2z)$. We leave it to the reader to verify that 1/2 is the only rational root of $q(z)$. Since $q'(z) = -1 - k(2z)^{k-1}$, the stationary points of $q(z)$ are $\zeta^j \xi$, where $\zeta = \exp(2\pi i/(k-1))$, $\xi = \frac{1}{2} \sqrt[k-1]{-k}$ and $j = 0, \ldots, k-2$. Now, $q(\zeta^j \xi) = 1 + \zeta^j \xi (k^{-1} - 1) \neq 0$, since otherwise $\zeta^j \xi$ would be a rational root. This shows that all roots of $q(z)$ are simple.

Moreover, there is one real stationary point if k is even, none if k is odd; hence, remarking that $q(-1/2) = 1$, we deduce that in the former case $q(z)$ has exactly one other real root (which is then $< -1/2$) and that it has none in the latter case.

Finally, suppose $z \in \boldsymbol{C}$, $|z| \leq 1/2$ and $z \neq 1/2$. Then $|z-1| > 1/2$ and $|2^{k-1} z^k| = 2^{k-1} |z|^k \leq 1/2 < |z-1|$; hence $|q(z)| > 0$. ■

Theorem 15. *With respect to Algorithm* 12:

1. $C_A(\mathscr{P}) = 2^{k-1} - 1$, $C_D(\mathscr{P}) = 1$. If $e > k$,

$$C_A^0(e) = \left(\frac{e}{k+1} - k \frac{k+3}{2(k+1)^2} \right) + o(1), \qquad C_D^0(e) = e - k + 1 - 2^{1-k}.$$

2. *Let $p(z)$ be as in Proposition* 14. *If k is even, call $\varrho_0 < 0$ the real root of $p(1/2z)$ (otherwise let $\varrho_0 = 0$). Let $\varrho_i, \bar{\varrho}_i$ be the complex roots of $p(1/2z)$, where $j = 1, \ldots, s = \lfloor (k-1)/2 \rfloor$. Define α_j, ψ_j, θ_j such that*

$$\alpha_j e^{\psi_j i} = \frac{1 - 2\varrho_j}{\varrho_j^{k+1} Q'(1/2\varrho_j)}, \; |\varrho_j| e^{\theta_j i} = \varrho_j, \quad \textit{if } j > 0;$$

$$\alpha_0 = \frac{1 - 2\varrho_0}{2\varrho_0^{k+1} Q'(1/2\varrho_0)} \quad \textit{if } k \textit{ even};$$

where $Q(z) = (1-2z)^2 (1-z) p(z)$. Then the error term for $C_A^0(e)$ is $\alpha_0 \varrho_0^e + \sum_{j=1}^{s} \alpha_i \cos(\psi_j + e\theta_j) |\varrho_j|^e$.

Proof. Thanks to Lemma 13 and Proposition 14, the only thing we need to do is to find the power series development of $F(z)$. We will suppose k even; if not, just forget about the real root ϱ_0. Define $p^R(z) = z^{k-1}p(1/z) = z^{k-1} + z^{k-2} + 2z^{k-3} + \ldots + 2^{k-2}$ and let $\sigma_1, \ldots, \sigma_{k-1}$ be its roots (so $\{\sigma_i/2\}_i = \{\varrho_0, \varrho_j, \overline{\varrho}_j\}_j$). We have, by Theorem 7.30 of [7] and Proposition 14, $\delta_e = (Ae+B)2^e + C + \sum_{i=1}^{k-1} D_i \sigma_i^e$. Thus, after rearranging, we get

$$\delta_e = (Ae+B)2^e + C + D_0\cdot(2\varrho_0)^e + \sum_{j=1}^{s} 2\ \mathrm{Re}\left(D_j\cdot(2\varrho_j)^e\right), \tag{1}$$

$$C_A^0(e) = \frac{\delta_e - \delta_{e-1}}{2^{e-1}} = (Ae+B+A) + \frac{D_0(2\varrho_0-1)}{\varrho_0}\varrho_0^e + \sum_{j=1}^{s}\mathrm{Re}\left(2D_j\frac{2\varrho_j-1}{\varrho_j}\varrho_j^e\right) \tag{2}$$

where $A = 2/Q''(1/2) = 1/(k+1)$, $C = -2^{k-1}/Q'(1) = 1$, for any j $D_j = -1/2\varrho_j^k Q'(1/2\varrho_j)$. In order to compute B, recall that $\delta_e = 0$ for $e = 0, \ldots, k-1$. Thus, Eq. (1) gives:

$$\begin{bmatrix} 2^0 & \sigma_1^0 & \cdots & \sigma_{k-1}^0 \\ 2^1 & \sigma_1^1 & \cdots & \sigma_{k-1}^1 \\ \vdots & \vdots & \ddots & \vdots \\ 2^{k-1} & \sigma_1^{k-1} & \cdots & \sigma_{k-1}^{k-1} \end{bmatrix} \begin{bmatrix} B \\ D_1 \\ \vdots \\ D_{k-1} \end{bmatrix} = -\frac{1}{k+1}\begin{bmatrix} k+1 \\ k+1+1\cdot 2^1 \\ \vdots \\ k+1+(k-1)\cdot 2^{k-1} \end{bmatrix}. \tag{3}$$

On the LHS there is a Vandermonde matrix, whose inverse $M = (m_{ij})$ is well known (see Ex. 1.2.3.40 of [8]): if we denote $[z^e]\,f(z)$ the coefficient of z^e in $f(z)$, then

$$m_{1j} = \frac{[z^{j-1}]\prod_{i=1}^{k-1}(z-\sigma_i)}{\prod_{i=1}^{k-1}(2-\sigma_i)} = \frac{[z^{j-1}]\,p^R(z)}{p^R(2)} = \begin{cases} 2^{1-j}/(k+1) & \text{if } j = 1, \ldots, k-1, \\ 2^{2-k}/(k+1) & \text{if } j = k. \end{cases}$$

Eq. (3), hence, gives $B = -\frac{1}{k+1}\sum_{j=1}^{k} m_{1j}(k+1+(j-1)2^{j-1}) = -\frac{k^2+5k+2}{2(k+1)^2}$. Since $|\sigma_i| < 2$ by Proposition 14, i.e. $|\varrho_j| < 1$, the first part of the theorem follows. The second part is just Eq. (2) expressed in polar coordinates. ∎

Remark 16. The theorem implies that a large k gives asymptotically better results; but a larger k would increase the cost of precomputations, so it is advantageous only if e is large enough. Once we know the relative cost of an addition and a multiplications, it is straightforward to derive the optimal value for k. For example, if both operations cost the same, k should be no larger than two if $e \leqslant 14$, three if $e \leqslant 62$, four if $e \leqslant 212$, five if $e \leqslant 631$, six if $e \leqslant 1737$.

Remark 17. For every k, let $\varrho(k)$ be the largest in norm between the ϱ_j. We have

k	2	3	4	5	6
$\lvert\varrho(k)\rvert$	$\frac{1}{2}$	0.707	0.822	0.885	0.921
$\lvert\alpha_{\varrho(k)}\rvert$	0.444	0.534	0.439	0.396	0.373

It follows that it could be slightly more convenient to choose e odd if $k = 2$ (the error term being $\frac{4}{9}(-\frac{1}{2})^e$) and, if $k = 3$, to choose e such that $\cos(\psi_3 + e\theta_3) \sim 1$. For $k \geqslant 4$ the error is completely negligible, e being too large.

Acknowledgements. I wish to thank Roberto Avanzi for the fruitful discussions on addition chains, Wenchang Chu and Alessandro Zaccagnini for telling me about generating functions, Elisa Salvioni for pointing out many typos and some mistake.

References

[1] D. J. BERNSTEIN, *Pippinger's exponentiation algorithm*, Draft; available at http://cr.yp.to/papers/pippenger.pdf, 2002.

[2] W. BOSMA, *Signed bits and fast exponentiation*, J. Théor. Nombres Bordeaux **13** (2001), 27-41. XXI Journées Arithmétiques, Rome 2001.

[3] H. COHEN, *Analysis of the flexible window powering algorithm*, Preprint; available at http://www.math.u-bordeaux.fr/~cohen. It will appear in the J. Cryptology.

[4] H. COHEN, *A course in computational algebraic number theory*, GTM, Springer-Verlag, Berlin 1993.

[5] P. DOWNEY, B. LEONG and R. SETHI, *Computing sequences with addition chains*, SIAM J. Comput. **10** (1981), 638-646.

[6] P. ERDŐS, *Remarks on number theory III. On addition chains*, Acta Arith. **6** (1960), 77-81.

[7] R. L. GRAHAM, D. E. KNUTH and O. PATASHNIK, *Concrete mathematics*, Addison-Wesley, Reading, Mass. 1989.

[8] D. E. KNUTH, *The art of computer programming*. Vol. 1: *Fundamental algorithms*, Addison-Wesley, 3rd edition, Reading, Mass. 1997.

[9] D. E. KNUTH, *The art of computer programming*. Vol. 2: *Seminumerical algorithms*, Addison-Wesley, 3rd edition, Reading, Mass. 1998.

[10] N. KUNIHIRO and H. YAMAMOTO, *Window and extended window methods for addition chain and addition-subtraction chain, IEICE Trans. Fund.* E **81**-A(1) (1998), 72-81.

Abstract

We present the 2^k-ary and the sliding window algorithms for fast exponentiation. We give a precise formula for the error terms of their complexity and we discuss how to choose the parameters or the exponent optimally.

* * *