On Consistent Inconsistencies

Amir Sapir
amirsa@cs.bgu.ac.il
Department of Computer Science, Sapir Academic College, Sha’ar haNegev, Israel

1 abstract

The most common measure for an algorithm’s efficiency is its running time, described as a function of the input length. Traditionally, however, several inconsistencies have found their way even to the analysis of the most classical algorithms. In some cases these are ‘slight’ inconsistencies, in the sense that their effect may change the result by a factor of \( \lg n \). However, some others have a polynomial effect, and on the extreme – an exponential one.

In [2] we discuss this matter, bringing pairs of examples for each of the categories of effect: in the first member of the pair the algorithm’s running time is calculated accurately, whereas in the other it is being traditionally analyzed incorrectly, demonstrating that the computing community is being inconsistent when it comes to algorithm analysis. Since this seems to be shared by all accepted textbooks (with the exception of the textbook of Brassard and Bratley [1, ch. 7.7], as far as the author knows), it cannot be attributed to errors in analysis; rather, it seems to be a matter of conventions that won their place among the people in the area.

The author’s work, discussing the three inconsistency types, has been published in ACM INROADS, March 2013, as [2].

In this work we describe the inconsistency which results in exponential effect (type II in the paper). We do it by presenting the well-known algorithm for calculating the \( n \)'th Fibonacci number, \( \text{IterFib}(N) \), denoted by \( A_{\text{ITER}} \) (Algorithm 1). Let \( t(A_{\text{ITER}}, n) \) be its running time on input of \( n \) bits.

In the classroom we describe \( A_{\text{ITER}} \) as an efficient, linear algorithm. Clearly \( A_{\text{ITER}} \) is linear in \( N \); however, the data size is \( n = \lceil \lg N \rceil \), so \( t(A_{\text{ITER}}, n) \in \Theta(2^{n}) \). Why do we consider it as an inconsistency? Let us compare the analysis to that of \( \text{UltraNaivePrime}(N) \) (Algorithm 2), an algorithm for checking whether an integer is prime or composite. Denote the algorithm by \( A_{\text{NVE}} \). Where does \( t(A_{\text{NVE}}, n) \) belong? As in the case of calculating Fibonacci numbers, the size of the data is \( n = \lceil \lg N \rceil \). The loop tests for division up until \( \sqrt{N} \), which requires \( \frac{1}{2}n \) bits, leading to \( O(\sqrt{2}^{n}) \) iterations.

In both problems, that of calculating the \( n \)'th Fibonacci number and that of primality testing, the input comprises \( \lg N \) bits. However, for the former it is common in CS education
Algorithm 1 IterFib(N)

/* The iterative algorithm. */
if N = 0 or N = 1 then
  return N
else
  a = 0; b = 1
  for i ← 2 to N do
    res ← a + b; a ← b; b ← res
  end for
end if
return res

Algorithm 2 UltraNaivePrime(N)

/* The algorithm decides whether N is prime by testing whether there is a number, up */
/* until $\sqrt{N}$, dividing N. For simplicity, we avoid more efficient algorithms. */
r ← ⌊$\sqrt{N}$⌋; i ← 2
  while (i ≤ r) and (N mod i ≠ 0) do
    i ← i + 1
  end while
  if N mod i = 0 then
    return true
  else
    return false
  end if

to refer erroneously to the size of the number as if it was the size of the input, leading to a
wrong conclusion as to the running time, whereas in the latter the input length is correctly
considered as $\lg N$ bits, and thus the conclusion regarding the running time is correct.

When the input is a list of numbers rather than a single one, counting the bits required to
represent a number is usually insignificant, and better be avoided in favor of clarity of anal-
ysis. However, in case the numbers are large (in the range of $O(2^n)$, say), this may become
(depending on the algorithm) important. We will elaborate on this issue, as well as on the
inconsistency which results in polynomial effect (type 1 in the paper), in the presentation.

Acknowledgement. I would like to thank Daniel Berend from BGU for the thorough
discussions and improvements while working on the subject and to Ariel Sapir from BGU for
the careful typing of the text and formulae of the paper as well as the presentation.

References