A note on the complexity of addition

Emil Jeřábek*

Institute of Mathematics, Czech Academy of Sciences

Žitná 25, 115 67 Praha 1, Czech Republic, email: jerabek@math.cas.cz

June 14, 2023

Abstract

We show that the sum of a sequence of integers can be computed in linear time on a Turing machine. In particular, the most obvious algorithm for this problem, which appears to require quadratic time due to carry propagation, actually runs in linear time by amortized analysis.

Keywords: computational complexity, integer addition, amortized analysis

1 Introduction

Elementary arithmetic operations on integers are some of the most basic algorithmic tasks, and their computational complexity is of fundamental importance. Even for such simple problems, accurate determination of their asymptotic time complexity (always measured on *multi-tape Turing machines* (*TM*) in this note) may be surprisingly difficult: while the complexity of $+$, $-$, and $<$ is clearly $\Theta(n)$ (the school-book algorithms run in time $O(n)$, which is optimal as any algorithm must at least read the whole input), the complexity of $\times$ and $/$ is still not settled after decades of research—the recent $O(n \log n)$ upper bound due to Harvey and van der Hoeven [3] is regarded as optimal by many, but no nontrivial lower bounds are known[1].

The problem we are interested in in this note is nowhere near as difficult to analyze, however it is just as basic, and yet it appears to have been missed by standard literature (cf. [4]). We consider the generalization of $+$ to summation of *many* integers: that is, the input is a sequence $\langle X_i : i < k\rangle$ of nonnegative integers written in binary and separated by $+$ signs (say), and the output is $\sum_{i<k} X_i$. (All our indices start from 0.) We denote this problem $\textsc{SeqSum}$. The size of the input is $\max\{k-1, 0\} + \sum_i n_i$, where $n_i = |X_i|$ denotes the length of X_i in binary; in order to simplify bounds, we will use the slightly different parameter $n = k + \sum_i n_i$. Can we still compute the result in time $O(n)$?

*Supported by the Czech Academy of Sciences (RVO 67985840) and GA ČR project 23-04825S.

[1]Assuming the network coding conjecture, Afshani et al. [1] proved an $\Omega(n \log n)$ bound on the number of wires in multiplication circuits, but besides being conditional, this does not imply any super-linear lower bounds on TM time complexity.

Algorithm 1 The accumulator algorithm

1: **procedure** SeqSum(X,Y)
2:     **while** X.head $\neq$ ␣ **do**
3:         $c \leftarrow 0$ ▷ carry bit
4:         **while** X.head $\in \{\mathtt{0}, \mathtt{1}\}$ **do**   ▷ addition loop
5:             $\langle c, Y.\text{head}\rangle \leftarrow \text{divmod}(X.\text{head} + Y.\text{head} + c, 2)$
6:             $X.\text{left}, Y.\text{left}$
7:         **while** $c \neq 0$ **do**   ▷ carry propagation
8:             $\langle c, Y.\text{head}\rangle \leftarrow \text{divmod}(Y.\text{head} + c, 2)$
9:             Y.left
10:         X.left   ▷ skip separator
11:         rewind Y back to the right end

2 The accumulator algorithm

Alg. 1 describes a baseline TM algorithm for SeqSum, which uses one tape as an "accumulator" to which we add the input numbers X_i one by one. In the pseudocode, X denotes the input tape, and Y the accumulator tape (which may be the output tape if we do not insist on its being write-only, or it may be a work tape that is duplicated to the output). We assume the tapes are infinite to the left, and the machine starts with heads at the right-most positions; the input sequence is written from right to left, with each integer starting with the least-significant bit on the right (as humans are used to). The symbol under the head of a tape T is accessed with T.head, and the T.left instruction moves the head one position to the left. The carry bit $c \in \{0, 1\}$ is included in the TM state. The divmod(a, b) operator computes the pair $\langle \lfloor a/b \rfloor, a \bmod b\rangle$; note that lines 5 and 6 together really denote a single step of the machine that updates Y.head and c (i.e., the state) based on the current contents of X.head, Y.head, and c, and moves both tape heads. (See [5] for a working implementation of the Turing machine, in an environment with somewhat different tape conventions.)

In order to analyze the time complexity of Alg. 1, consider an input $\langle X_i : i < k\rangle$, write $n_i = |X_i|$ and $n = k + \sum_{i<k} n_i$ as in Sec. 1, and put $Y_i = \sum_{j<i} X_j$ and $m_i = |Y_i|$. The ith (counting from 0) iteration of the outer loop performs the addition $X_i + Y_i \to Y_{i+1}$, and takes time $O(\max\{n_i, m_i\}) \subseteq O(n)$ as $m_i \leq n$; the overall running time is thus $O(nk) \subseteq O(n^2)$. While these estimates are somewhat wasteful, there is no obvious way how to improve them. In particular, the addition of X_i to Y_i may cost time $\approx m_i$ even if n_i is much smaller than m_i due to carry propagation, and there are instances where $m_i = \Omega(n)$ for all $i \neq 0$, and $k = \Omega(n)$, whence $\sum_{i<k} m_i = \Omega(n^2)$: e.g., take $|X_0| = n/2$ and $X_i = 1$ for $1 \leq i < k = n/4$. Thus, Alg. 1 appears to require quadratic time at first sight.

Nonetheless, we will show that Alg. 1 only needs time $O(n)$ by a more refined argument.

3 Amortized analysis

We will estimate the running time of Alg. 1 more accurately using basic methods of amortized complexity [6]. The point is that even though a single addition $X_i + Y_i \to Y_{i+1}$ may cost time up to m_i due to long carry propagation, the latter cannot happen very often; but we need to quantify this idea properly to see that it indeed works out to an $O(n)$ overall cost.

A well-known simple special case is the *binary counter* (see e.g. [2, pp. 451 ff.]): the counter holds an integer in binary, starting with 0, and we perform n increment operations; this is virtually identical to Alg. 1 applied to an input of the form $1 + 1 + \cdots + 1$. Each increment may access up to $\log n$ bits, hence a naïve bound on the running time is $O(n \log n)$. But while all n increments access the 0th bit of the counter, only $n/2$ access the 1st bit, $n/4$ the 2nd bit, etc., thus the overall running time is only $O(n)$.

For a general instance of Alg. 1, it seems difficult to directly count the number of carries in a similar way. However, we can prove a linear upper bound using the "accounting method".

Theorem 1 *Algorithm 1 solves* SeqSum *in time* $4n + 1$.

Proof: For each $i < k$, let t_i denote the furthest bit position (starting from 0) of Y accessed by Alg. 1 during the addition of X_i to the accumulator. This addition takes $2(t_i+1)$ steps (including the rewind on line 11), hence the total running time[2] of the algorithm is $t = 2\sum_i (t_i + 1) + 1$.

If there is no carry propagation, then $t_i = n_i$, and the addition loop modifies up to n_i bits of Y in positions $0, \ldots, n_i - 1$.

If there *is* carry propagation, then $t_i \geq n_i$, the main addition loop may modify some bits in positions $0, \ldots, n_i - 1$, and the carry propagation loop modifies positions $n_i, \ldots, t_i$: the bit in position t_i changes from 0 to 1, and the $t_i - n_i$ bits in positions $n_i, \ldots, t_i - 1$ change from 1 to 0. Since the accumulator starts with 0, each of the latter $t_i - n_i$ changes can be uniquely associated with the latest change of 0 to 1 in the same position during the addition of X_j to the accumulator for some $j < i$. We have already seen that the number of such changes for a given j is at most $n_j + 1$, thus

$$\sum_{i<k} (t_i - n_i) \leq \sum_{j<k} (n_j + 1) = n.$$

It follows that

$$t = 2\Big(\sum_{i<k} (n_i + 1) + \sum_{i<k} (t_i - n_i)\Big) + 1 \leq 4n + 1.$$

(In other words, we charge 2 coins for each change $0 \to 1$, one of which is spent right away, and the other is saved on that position as credit; changes $1 \to 0$ are then paid from credit.)

We can alternatively frame the argument using the "potential method": let Φ_i be the number of 1s in Y_i. The discussion above shows that $\Phi_{i+1} - \Phi_i \leq n_i - (t_i - n_i) + 1$, i.e.,

$$t_i + 1 \leq 2(n_i + 1) + \Phi_i - \Phi_{i+1},$$

[2] The final $+1$ is for a transition to a halting state; depending on the exact details of the formalization of TM, it may be unnecessary. The reader may check that it can be shaved off anyway as long as $n \geq 1$.

therefore

$$t \leq 4\sum_{i<k}(n_i+1) + 2\sum_{i<k}(\Phi_i - \Phi_{i+1}) + 1 = 4n - 2\Phi_k + 1 \leq 4n+1$$

as $\Phi_0 = 0$. □

We mention that even though the problem is stated for binary integers for convenience, the result actually holds for any base $b \geq 2$. (There is also a trivial linear-time algorithm for unary integers.) Moreover, it generalizes to sums of possibly negative integers, represented by a sign bit and absolute value: it suffices to sum the positive and negative entries separately, and subtract the results.

Let us briefly discuss the role of the model of computation. The main alternative to TM in algorithmic complexity are *random-access machines* (*RAM*). There is a lot of variation in RAM models both in terms of available features and in terms of the definition of time complexity. Typically, simple arithmetic operations on registers such as addition and subtraction are included as primitive operations. It follows that in unit-cost RAM models with unlimited register size, the SeqSum problem is trivially solvable in linear time. If we limit register size to $O(\log n)$ bits (word RAM) or in logarithmic-cost models, the problem becomes nontrivial again as the obvious algorithm needs quadratic time (or at least appears so, as in the case of TM). We can still solve SeqSum in time $O(n)$ by Theorem 1 using an array of constant-size registers to simulate TM tapes, as long as we can access the individual entries of the array with unit cost. But if indexing register number r costs $\log r$, this algorithm requires time $\Theta(n \log n)$; in such a model, $\Theta(n \log n)$ time is necessary even just to read the input if it is also represented as an array of bits.

4 Conclusion

We analyzed the performance of a straightforward algorithm for summing a sequence of integers on a TM, and found that it takes time $O(n)$, despite first appearances suggesting it is only $O(n^2)$. The argument is quite simple, and it is an extension of a standard example in amortized complexity (the binary counter). Nevertheless, to the best of our knowledge it has not been observed before in commonly available literature; given the fundamental nature of the problem, we believe it is worthwhile to point out the result explicitly.

References

[1] Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen, *Lower bounds for multiplication via network coding*, in: 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, eds.), Leibniz International Proceedings in Informatics (LIPIcs) vol. 132, 2019, pp. 10:1–12.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, fourth ed., MIT Press, Cambridge, Massachusetts, 2022, 1312 pp.

[3] David Harvey and Joris van der Hoeven, *Integer multiplication in time $O(n \log n)$*, Annals of Mathematics 193 (2021), no. 2, pp. 563–617.

[4] Emil Jeřábek, *Can we do integer addition in linear time?*, Theoretical Computer Science Stack Exchange, 2023, `https://cstheory.stackexchange.com/q/52391`.

[5] ———, *Sequence sum*, Martin Ugarte's Turing machine simulator, 2023, `https://turingmachinesimulator.com/shared/thnutsdfym`.

[6] Robert E. Tarjan, *Amortized computational complexity*, SIAM Journal on Algebraic and Discrete Methods 6 (1985), no. 2, pp. 306–318.