On the complexity of addition

Emil Jeřábek

Institute of Mathematics
Czech Academy of Sciences
jerabek@math.cas.cz
https://math.cas.cz/~jerabek/

Czech Gathering of Logicians
University of Ostrava, 2 June 2023

Addition algorithms

1 Addition algorithms

2 Amortized analysis

Computational complexity of arithmetic

Time complexity of integer arithmetic operations:

- Standard computational complexity model:
  - multitape Turing machines
    (RAM model has $+$ built in $\implies$ trivial cheat)
  - integers $X, Y, \dots$ written in binary (or decimal)
  - how many steps does it take, measured in terms of the size of input: $n = \|X\| + \|Y\| + \cdots$, $\|X\| \approx \log X$
- $X + Y$, $X - Y$, $X < Y$
  - linear time $O(n)$
  - optimal: need to read the input
- $X \cdot Y$, $\lfloor X/Y \rfloor$
  - still not quite settled after many decades of research
  - best known upper bound: $O(n \log n)$ [HvdH21]
  - lower bounds?

    (network coding conjecture $\implies$ circuit LB: $\Omega(n \log n)$ wires [ACKL19])

School-book addition algorithm

Input tape 0: $\cdots$ 1 1 0 1

Input tape 1: $\cdots$ 1 1 0 1 0 1

Output tape: $\cdots$

State: carry 0

School-book addition algorithm

Input tape 0:	···							1	1	0	1
Input tape 1:	···					1	1	0	1	0	1
Output tape:	···										0

State: carry 1

School-book addition algorithm

Input tape 0: ··· | | | | | | |1|1|0|1|

Input tape 1: ··· | | | | |1|1|0|1|0|1|

Output tape: ··· | | | | | | | | |1|0|

State: carry 0

School-book addition algorithm

Input tape 0:	···							1	1	0	1
Input tape 1:	···					1	1	0	1	0	1
Output tape:	···								0	1	0

State: carry 1

School-book addition algorithm

Input tape 0:	⋯							1	1	0	1
Input tape 1:	⋯					1	1	0	1	0	1
Output tape:	⋯							0	0	1	0

State: carry 1

School-book addition algorithm

Input tape 0:	$\cdots$							1	1	0	1
Input tape 1:	$\cdots$					1	1	0	1	0	1
Output tape:	$\cdots$						0	0	0	1	0

State: carry 1

School-book addition algorithm

Input tape 0:	$\cdots$							1	1	0	1
Input tape 1:	$\cdots$					1	1	0	1	0	1
Output tape:	$\cdots$					0	0	0	0	1	0

State: carry 1

School-book addition algorithm

Input tape 0:	⋯							1	1	0	1
Input tape 1:	⋯					1	1	0	1	0	1
Output tape:	⋯				1	0	0	0	0	1	0

State: halt

Sequence sum

What if we want to add more than two numbers?

SEQSUM

- input: sequence of integers $\langle X_i : i < k \rangle$ separated with "+"
- output: $\sum_{i<k} X_i$

Size of input: $n = k + \sum_{i<k} n_i$, $n_i = \|X_i\|$

Question:

- What is the time complexity of SEQSUM?
- Can we do it in time $O(n)$?

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
$\quad Y \leftarrow Y + X_i$

							X_3			X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape:	$\cdots$																

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$\begin{array}{l} Y \leftarrow 0 \\ \textbf{for } i < k \textbf{ do:} \\ \qquad Y \leftarrow Y + X_i \end{array}$$

							X_3				X_2		X_1					X_0
Input tape:	⋯				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																		Y
Output tape:	⋯																	0

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
    $Y \leftarrow Y + X_i$

						X_3				X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape:	$\cdots$																0	0

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

						X_3				X_2		X_1					X_0
Input tape: ···				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																	Y
Output tape: ···															1	0	0

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

						X_3			X_2	X_1						X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																Y
Output tape:	$\cdots$														1	1	0	0

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$\begin{array}{l} Y \leftarrow 0 \\ \textbf{for } i < k \textbf{ do:} \\ \qquad Y \leftarrow Y + X_i \end{array}$$

						X_3			X_2		X_1					X_0
Input tape:	⋯				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																	Y
Output tape:	⋯														1	1	0	0

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
    $Y \leftarrow Y + X_i$

						X_3				X_2		X_1					X_0
Input tape: $\cdots$				1	0	1	+	1	1	0	+	1	$\boxed{+}$	1	1	0	0

																	Y
Output tape: $\cdots$														1	$\boxed{1}$	0	0

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

							X_3			X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																		Y
Output tape:	$\cdots$														1	1	0	0

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

						X_3			X_2		X_1						X_0
Input tape:	···				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																		Y
Output tape:	···														1	1	0	0

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$\begin{array}{l} Y \leftarrow 0 \\ \textbf{for } i < k \textbf{ do:} \\ \qquad Y \leftarrow Y + X_i \end{array}$$

						X_3			X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																	Y
Output tape:	$\cdots$														1	1	0	0

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

						X_3				X_2	X_1						X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	$\boxed{+}$	1	+	1	1	0	0
																	Y
Output tape:	$\cdots$														1	1	$\boxed{0}$	1

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{aligned}
&Y \leftarrow 0\\
&\textbf{for } i < k \textbf{ do:}\\
&\qquad Y \leftarrow Y + X_i
\end{aligned}
$$

Input tape:

						X_3				X_2		X_1					X_0
$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

Output tape:

																	Y
$\cdots$														1	1	0	1

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

Input tape:

						X_3				X_2		X_1					X_0
$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

Output tape:

																	Y
$\cdots$														1	1	0	1

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

Input tape:

						X_3				X_2		X_1					X_0
$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

Output tape:

																	Y
$\cdots$														1	1	0	1

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

							X_3				X_2		X_1				X_0
Input tape: $\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																	Y
Output tape: $\cdots$														1	1	1	1

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

							X_3				X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																		Y
Output tape:	$\cdots$														1	0	1	1

State: carry 1

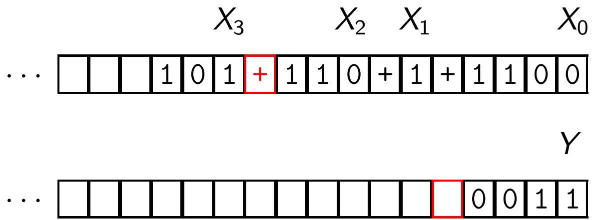
Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

Input tape:

Output tape:



State: carry 1

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

						X_3				X_2		X_1					X_0
Input tape: $\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape: $\cdots$													1	0	0	1	1

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

Input tape: ···

			X_3					X_2		X_1					X_0
			1	0	1	+	1	1	0	+	1	+	1	1	0	0

Output tape: ···

															Y
												1	0	0	1	1

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

						X_3				X_2		X_1					X_0
Input tape: $\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0
																	Y
Output tape: $\cdots$													1	0	0	1	1

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
$\quad Y \leftarrow Y + X_i$

$X_3 \quad X_2 \quad X_1 \quad X_0$

Input tape: $\cdots$ | | | |1|0|1|+|1|1|0|+|1|+|1|1|0|0|

Y

Output tape: $\cdots$ | | | | | | | | | | | | |1|0|0|1|1|

State: rewind

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

						X_3			X_2		X_1					X_0
Input tape:	$\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape:	$\cdots$													1	0	0	1	1

State: carry 0

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

							X_3				X_2		X_1				X_0
Input tape:	⋯				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape:	⋯													1	0	0	1	0

State: carry 1

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$$
\begin{array}{l}
Y \leftarrow 0 \\
\textbf{for } i < k \textbf{ do:} \\
\qquad Y \leftarrow Y + X_i
\end{array}
$$

X_3 X_2 X_1 X_0

Input tape: ⋯ | | | |1|0|1|+|1|1|0|+|1|+|1|1|0|0|

Y

Output tape: ⋯ | | | | | | | | | | | | |1|0|0|0|0|

State: carry 1

Simple SeqSum algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
    $Y \leftarrow Y + X_i$

						X_3			X_2		X_1					X_0
Input tape:	…				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape:	…												1	0	0	0	0

State: carry 1

Simple SEQSUM algorithm

Use one tape as an accumulator Y:

$Y \leftarrow 0$
for $i < k$ **do:**
  $Y \leftarrow Y + X_i$

					X_3			X_2		X_1						X_0
Input tape: $\cdots$				1	0	1	+	1	1	0	+	1	+	1	1	0	0

																	Y
Output tape: $\cdots$													1	1	0	0	0

State: halt

Time complexity analysis

The content of Y before adding X_i: $Y_i = \sum_{j<i} X_j$, $m_i = \|Y_i\|$

$Y \leftarrow Y + X_i$ takes time $O(n_i + m_i) \subseteq O(n)$ as $m_i \leq n$

$\implies$ total time: $O(nk) \subseteq O(n^2)$

- even if $n_i < m_i$, $Y \leftarrow Y + X_i$ may take time up to $\approx m_i$ due to carry propagation
- we may have $m_i = \Omega(n)$ for all $i > 0$, and $k = \Omega(n)$: take $\text{huge } X_0$ and constant-size $X_1, \dots, X_{k-1}$, $k \approx \|X_0\|$

The complexity of $\textsc{SeqSum}$

$\textsc{SeqSum}$ is computable in time $O(n^2)$

Question: Can we do better?

The complexity of SeqSum

SeqSum is computable in time $O(n^2)$

Question: Can we do better?

Answer:

- Yes, we can! SeqSum is computable in time $O(n)$
- We don't even need a better algorithm:
  we just need a better analysis!

Amortized analysis

1 Addition algorithms

2 Amortized analysis

Amortized complexity

Identified as a concept and named by [Tar85]

- If an operation is used many times in an algorithm, it may happen that its average (amortized) cost is smaller than its maximal cost
- NOT average-case analysis: still worst-case wrt the input!
- Typical use case: data structures
- Example: stack implemented by an array
  - when capacity exhausted, reallocate double size and copy
  - algorithm performs n stack operations (`push`, `pop`)
    $\Longrightarrow$ each operation may cost up to $O(n)$ steps
  - but: the average cost is only $O(1)$!
    total cost of reallocations is $O(n + \frac{n}{2} + \frac{n}{4} + \cdots) = O(n)$
- Basic strategies: aggregate analysis, accounting method, potential method

Binary counter

Basic example (see e.g. [CLRS22]):

Counter

- holds an integer in binary
- starts with 0, performs n increments
  $0 \to 1 \to \cdots \to n$

Cost of an increment:

- maximal $O(\log n)$: carry propagation
- amortized $O(1)$

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100

Binary counter

Basic example (see e.g. [CLRS22]):

Counter

- holds an integer in binary
- starts with 0, performs n increments
  $0 \to 1 \to \cdots \to n$

Cost of an increment:

- maximal $O(\log n)$: carry propagation
- amortized $O(1)$: aggregate analysis

updates: $n \times$ position 0, $\frac{n}{2} \times$ pos. 1, $\frac{n}{4} \times$ pos. 2, ...

$\Longrightarrow$ total cost $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots < 2n$

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100

Increments $\to$ sums?

Counter $\approx$ accumulator $\textsc{SeqSum}$ algorithm for $1 + 1 + \cdots + 1$

Can we generalize the amortized analysis to the full algorithm?

- direct aggregate analysis not easy
- accounting method:
  - pay the cost of excess carries from "credits" saved earlier
- potential method:
  - define "potential energy" of TM configurations
  - changes of the potential account for work on carries

Improved analysis of SeqSum

Recall: input $\langle X_i : i < k \rangle$, $n_i = \|X_i\|$, $n = k + \sum_{i<k} n_i$

The cost of one addition $Y \leftarrow Y + X_i$:

X_i:									1	0	1	1	0	0	1
old Y:			1	0	0	1	1	1	0	1	1	0	1	0	1
new Y:			1	0	1	0	0	0	0	0	0	1	1	1	0

n_i

- regular costs: $n_i + 1 \implies$ total: $k + \sum_{i<k} n_i = n$
- paid from credit: carries $1 \to 0$
  - the "1" got there by a regular change $0 \to 1$ earlier!
    $\implies$ cover all credits by charging regular costs twice
- grand total: $2n$ (actually $4n$ due to rewinds)

Potential method

Potential function Φ = the number of 1s in Y

Φ_i = the value of Φ before the addition $Y \leftarrow Y + X_i$

By the same argument: the cost of $Y \leftarrow Y + X_i$ is at most

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

Since $\Phi_0 = 0$ and $\Phi_k \geq 0$, the total cost is at most

$$\sum_{i<k} \bigl(2(n_i + 1) + \Phi_i - \Phi_{i+1}\bigr) = 2\Bigl(k + \sum_{i<k} n_i\Bigr) + \Phi_0 - \Phi_k \leq 2n$$

Summary

Computational complexity of $\sum_{i<k} X_i$:

- the obvious algorithm appears to require time $O(n^2)$ on the first sight
- it actually runs in time $4n$
- extension of a common example in amortized complexity
- seems to be missing in standard literature, even though it is a fundamental algorithmic problem

References

- P. Afshani, C. B. Freksen, L. Kamma, K. G. Larsen: Lower bounds for multiplication via network coding, ICALP 2019, LIPIcs 132 (2019), 10:1–12
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to algorithms, MIT Press, 2022 (4th ed.)
- D. Harvey, J. van der Hoeven: Integer multiplication in time $O(n \log n)$, Ann. of Math. 193 (2021), 563–617
- E. Jeřábek: Can we do integer addition in linear time?, Theoretical Computer Science Stack Exchange, `https://cstheory.stackexchange.com/q/52391`
- R. E. Tarjan: Amortized computational complexity, SIAM J. Algebraic Discrete Methods 6 (1985), 306–318