LABORATOIRE D'INFORMATIQUE
GASPARD-MONGE

Sous la co-tutelle de :
CNRS
ÉCOLE DES PONTS PARISTECH
ESIEE PARIS
UPEM • UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

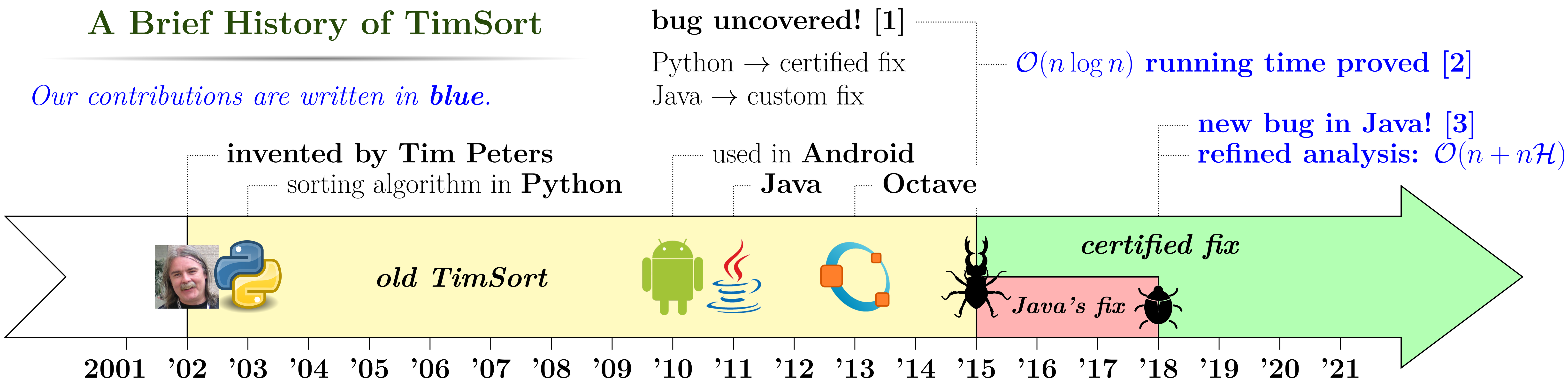
Analysis of TimSort Algorithm

Nicolas Auger, Vincent Jugé, Cyril Nicaud, Carine Pivoteau

MOA – LIGM (UMR 8049)

A Brief History of TimSort

Our contributions are written in ***blue***.



TimSort Principle

Natural decomposition of a sequence into maximal monotonic runs:

← 3 →			← 7 →							← 4 →				← 2 →	
a	b	x	h	f	e	d	c	b	a	b	b	c	d	a	b

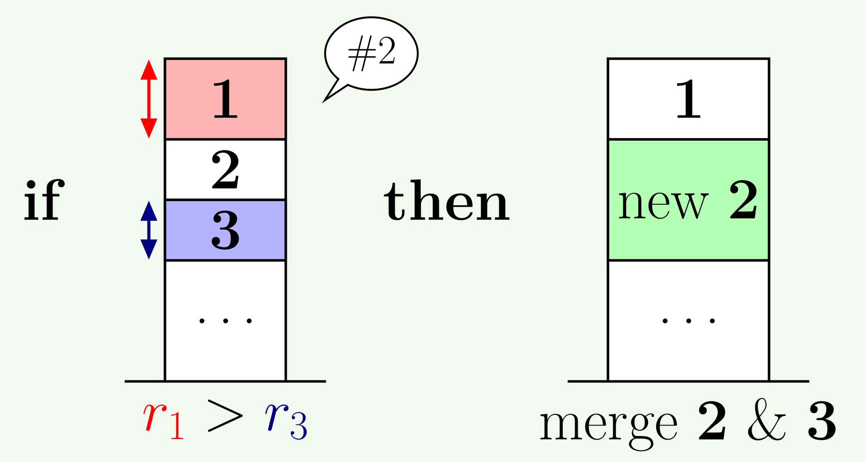
Algorithm

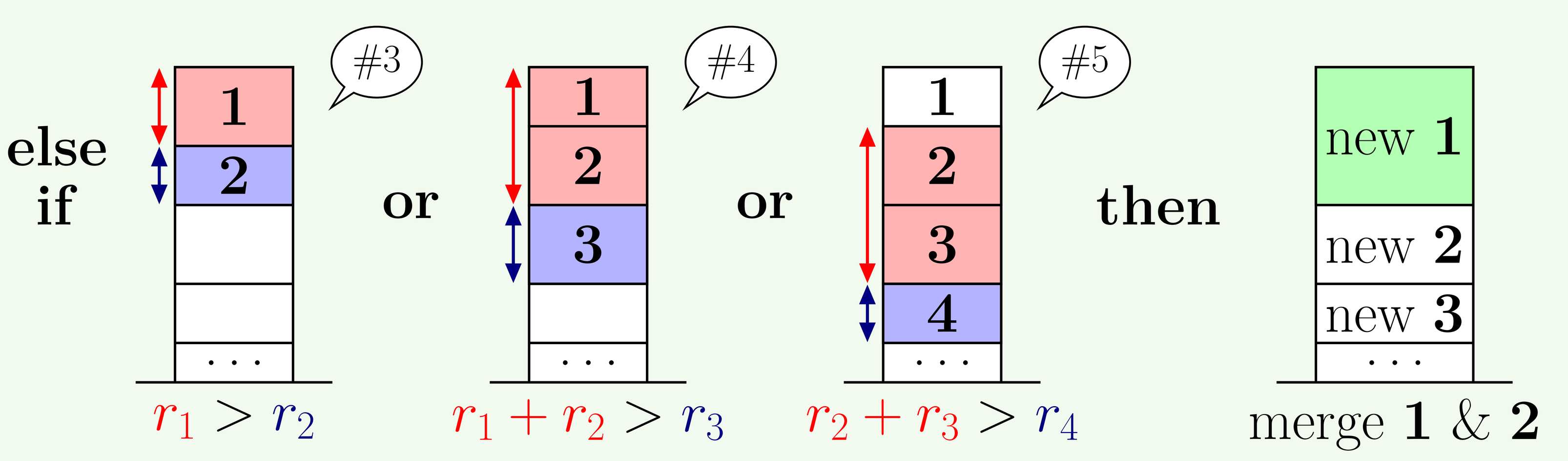
get a run decomposition of the initial sequence
start with an empty stack

while the run decomposition is not empty **do**
  take the next run and push it onto the stack (#1)
  while this is possible **do**
    apply one of the merge rules (#2) (#3) (#4) (#5)

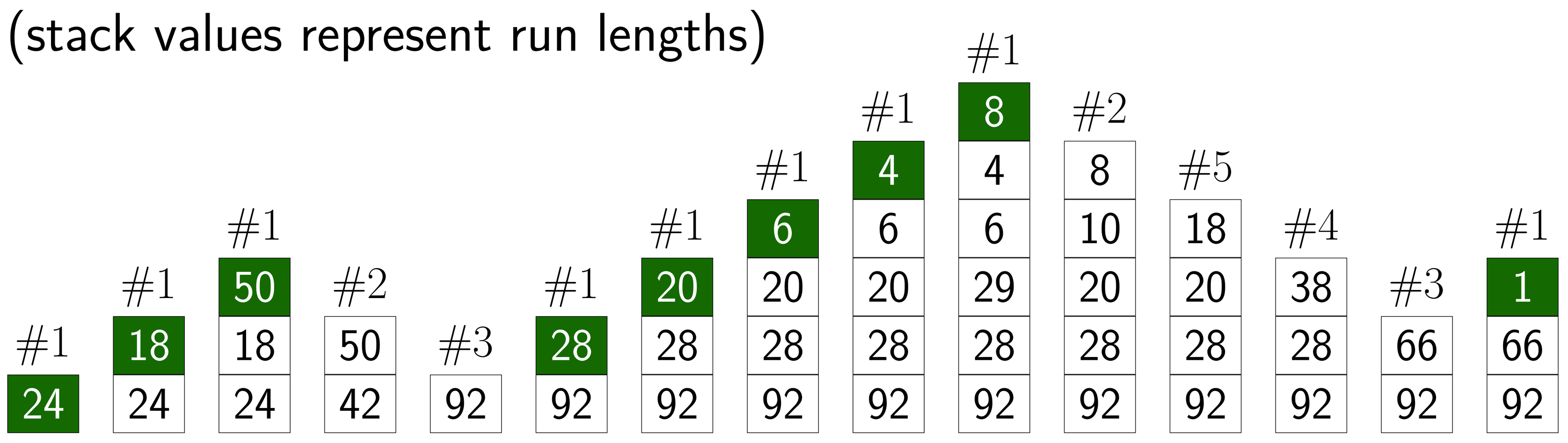
while the stack has more than 1 element **do**
  merge the two topmost runs

$1, 2, 3, 4, \ldots$ are the topmost runs on the stack. The length of the i^{th} run is denoted by r_i.





Example of stack evolution for runs of length 24, 18, 50, 28, 20, 6, 4, 8, 1:
(stack values represent run lengths)



Results

13 years after its announcement by Tim Peters, we obtained the first proof of its worst-case running time:

Theorem [2, 3]: TimSort runs in $\mathcal{O}(n \log n)$ time.

By design TimSort is well suited for **partially sorted** data. It falls into the **adaptive sort** family. Taking the number of runs ρ as a (natural) parameter for a refined analysis we obtained:

Theorem [3]: TimSort runs in $\mathcal{O}(n + n \log \rho)$ time.

Going further, we introduce the notion of **entropy** $\mathcal{H}$ to take the variations in run lengths into account, defined by:

$$\mathcal{H} = -\sum \frac{r_i}{n} \log_2 \left(\frac{r_i}{n}\right)$$

Theorem [3]: TimSort runs in $\mathcal{O}(n + n\mathcal{H})$ time.

Our in-depth analysis of the algorithm enabled us to produce a bug in Java sorting! It was fixed within a few weeks after we reported it.

References

[1] S. de Gouw, J. Rot, F. de Boer, R. Bubel, and R. Hähnle. OpenJDK's Java.utils.Collection.sort() is broken: The good, the bad and the worst case. In *CAV*, 2015.
[2] N. Auger, C. Nicaud, and C. Pivoteau. Merge strategies: From Merge Sort to TimSort. Research Report hal-01212839, hal, 2015.
[3] N. Auger, V. Jugé, C. Nicaud, and C. Pivoteau. On the worst-case complexity of TimSort. In *ESA*, 2018.