Showing that Android’s, Java’s and Python’s sorting algorithm is broken and fixing it formally

Stijn de Gouw
Jurriaan Rot, Frank de Boer, Richard Bubel, Reiner Hähnle

CWI Amsterdam / SDL Fredhopper

Formal Methods 2015
Oslo, June 22, 2015

http://www.envisage-project.eu
Library

Collection of commonly used algorithms that are invoked through a well-defined interface
Library

Collection of commonly used algorithms that are invoked through a well-defined interface

Example: Java standard library functions

Programming to interfaces:

- Sorting a given array `a`
  ```java
  static void sort(Object[] a)
  ```

- Searching a value `key` in the array `a`
  ```java
  static int binarySearch(Object[] a, Object key)
  ```

Usability of programming language partially depends on good libraries
Programming languages: Libraries

Library
Collection of commonly used algorithms that are invoked through a well-defined interface

Example: Java standard library functions

Programming to interfaces:

▶ Sorting a given array `a`

```java
static void sort(Object[] a)
```

▶ Searching a value `key` in the array `a`

```java
static int binarySearch(Object[] a, Object key)
```

Usability of programming language partially depends on good libraries

Correctness of library functions is crucial:
used as building blocks in millions of programs
Description

Timsort: a hybrid sorting algorithm (insertion sort + merge sort) optimized for partially sorted arrays (often encountered in real-world data).
Description

Timsort: a hybrid sorting algorithm (insertion sort + merge sort) optimized for partially sorted arrays (often encountered in real-world data).

Timsort is used in

- Java (standard library), used by Oracle
- Python (standard library), used by Google
- Android (standard library), used by Google
- Hadoop (Big data), used by Apache, Facebook and Yahoo
- ... and many more languages / frameworks!

TimSort.rangeCheck appeared in court case between Oracle and Google
Timsort (I)

Description

Timsort: a hybrid sorting algorithm (insertion sort + merge sort) optimized for partially sorted arrays (often encountered in real-world data).

Timsort is used in

- Java (standard library), used by Oracle
- Python (standard library), used by Google
- Android (standard library), used by Google
- Hadoop (Big data), used by Apache, Facebook and Yahoo
- ... and many more languages / frameworks!

TimSort.rangeCheck appeared in court case between Oracle and Google

Why analyze Timsort?

- Complex algorithm, widely used
- Extensively tested + manual code reviews: bugs unlikely!? 
Timsort (II)

The algorithm

- Find next already sorted segment ("runs") extending to length $\geq 16$ with insertion sort.
- Add length of new run to `runLen` array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]$
  2. $\text{runLen}[n-1] > \text{runLen}[n]$

  Merging: if (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$, merge runs at $n-2$ and $n-1$, otherwise at $n-1$ and $n$
- At the end: merge all runs, resulting in a sorted array

Example, ignoring length $\geq 16$ requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
</table>

`runLen`
The algorithm

- Find next already sorted segment ("runs") extending to length $\geq 16$ with insertion sort.
- Add length of new run to runLen array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]$
  2. $\text{runLen}[n-1] > \text{runLen}[n]$

  Merging: if (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$, merge runs at $n-2$ and $n-1$, otherwise at $n-1$ and $n$

- At the end: merge all runs, resulting in a sorted array

Example, ignoring length $\geq 16$ requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1 2 3 4 5 0 1 1 0 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length ≥ 16 with insertion sort.
- Add length of new run to runLen array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  2. runLen[n-1] > runLen[n]
- Merging: if (1) is false and runLen[n-2] < runLen[n], merge runs at n-2 and n-1, otherwise at n-1 and n
- At the end: merge all runs, resulting in a sorted array

Example, ignoring length ≥ 16 requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length $\geq 16$ with insertion sort.
- Add length of new run to \texttt{runLen} array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]
  2. $\text{runLen}[n-1] > \text{runLen}[n]

Merging: if (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$, merge runs at $n-2$ and $n-1$, otherwise at $n-1$ and $n$

- At the end: merge all runs, resulting in a sorted array

Example, ignoring length $\geq 16$ requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length $\geq 16$ with insertion sort.
- Add length of new run to runLen array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]$
  2. $\text{runLen}[n-1] > \text{runLen}[n]$

Merging: if (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$, merge runs at $n-2$ and $n-1$, otherwise at $n-1$ and $n$

- At the end: merge all runs, resulting in a sorted array

Example, ignoring length $\geq 16$ requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment (“runs”) extending to length ≥ 16 with insertion sort.
- Add length of new run to runLen array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  2. runLen[n-1] > runLen[n]
  Merging: if (1) is false and runLen[n-2] < runLen[n], merge runs at n-2 and n-1, otherwise at n-1 and n
- At the end: merge all runs, resulting in a sorted array

Example, ignoring length ≥ 16 requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length \( \geq 16 \) with insertion sort.
- Add length of new run to \( \text{runLen} \) array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. \( \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \)
  2. \( \text{runLen}[n-1] > \text{runLen}[n] \)

Merging: if (1) is false and \( \text{runLen}[n-2] < \text{runLen}[n] \), merge runs at \( n-2 \) and \( n-1 \), otherwise at \( n-1 \) and \( n \)

- At the end: merge all runs, resulting in a sorted array

Example, ignoring length \( \geq 16 \) requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{runLen} )</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length $\geq 16$ with insertion sort.
- Add length of new run to $\text{runLen}$ array
- Merge until last 3 runs satisfy two conditions ("the invariant")
  1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]$
  2. $\text{runLen}[n-1] > \text{runLen}[n]$
  Merging: if (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$, merge runs at $n-2$ and $n-1$, otherwise at $n-1$ and $n$
- At the end: merge all runs, resulting in a sorted array

Example, ignoring length $\geq 16$ requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The algorithm

- Find next already sorted segment ("runs") extending to length \( \geq 16 \) with insertion sort.
- Add length of new run to \( \text{runLen} \) array
- Merge until last 3 runs satisfy two conditions ("invariant")
  1. \( \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \)
  2. \( \text{runLen}[n-1] > \text{runLen}[n] \)

Merging: if (1) is false and \( \text{runLen}[n-2] < \text{runLen}[n] \), merge runs at \( n-2 \) and \( n-1 \), otherwise at \( n-1 \) and \( n \)

- At the end: merge all runs, resulting in a sorted array

Example, ignoring length \( \geq 16 \) requirement

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{runLen}</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Breaking the invariant

Size of \( \text{runLen} \)

1. \( \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \)
2. \( \text{runLen}[n-1] > \text{runLen}[n] \)

If the above invariant is true for all \( n \) and \( \text{runLen}[n] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array
Breaking the invariant

Size of \texttt{runLen}

1. \texttt{runLen[n-2] > runLen[n-1] + runLen[n]}
2. \texttt{runLen[n-1] > runLen[n]}

If the above invariant is true for \textit{all} \texttt{n} and \texttt{runLen[n] >= 16}, then

- (reversed) runlengths grow exponentially fast (... 87 $52+34+17+16$)
- Runs do not overlap: few runs required to cover input array

```java
int stackLen = (len < 120 ? 4 : len < 1542 ? 9 : len < 119151 ? 18 : 39);
runBase = new int[stackLen];
runLen = new int[stackLen];
```
Breaking the invariant

Size of \( \text{runLen} \)

1. \( \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \)
2. \( \text{runLen}[n-1] > \text{runLen}[n] \)

If the above invariant is true for all \( n \) and \( \text{runLen}[n] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array
Breaking the invariant

Size of \textit{runLen}

1. \texttt{runLen[n-2] > runLen[n-1] + runLen[n]}
2. \texttt{runLen[n-1] > runLen[n]}

If the above invariant is true for all \( n \) and \( \texttt{runLen[n]} >= 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

If (1) is false and \( \texttt{runLen[n-2]} < \texttt{runLen[n]} \): merge at idx \( n-2 \) and \( n-1 \), otherwise merge runs at indices \( n-1 \) and \( n \)

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
runLen & 120 & 80 & 25 & 20 \\
\hline
\end{tabular}
\end{center}
Breaking the invariant

**Size of** $\text{runLen}$

1. $\text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]$
2. $\text{runLen}[n-1] > \text{runLen}[n]$

If the above invariant is true for all $n$ and $\text{runLen}[n] \geq 16$, then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

---

Breaking the invariant - checking last 3 runs is insufficient

If (1) is false and $\text{runLen}[n-2] < \text{runLen}[n]$: merge at idx $n-2$ and $n-1$, otherwise merge runs at indices $n-1$ and $n$

| runLen | 120 | 80 | 25 | 20 | 30 |
Breaking the invariant

Size of \textit{runLen}

1. \texttt{runLen[n-2] > runLen[n-1] + runLen[n]}
2. \texttt{runLen[n-1] > runLen[n]}

If the above invariant is true for all \textit{n} and \texttt{runLen[n] \geq 16}, then

\begin{itemize}
  \item (reversed) runlengths grow exponentially fast (\ldots 87 52 34 17 16)
  \item Runs do not overlap: few runs required to cover input array
\end{itemize}

Breaking the invariant - checking last 3 runs is insufficient

\textbf{If (1) is false and runLen [n-2] < runLen [n]: merge at idx n-2 and n-1, otherwise merge runs at indices n-1 and n}

\begin{tabular}{|c|c|c|c|c|}
  \hline
  runLen & 120 & 80 & 25 & 20 & 30 \\
  \hline
\end{tabular}
Breaking the invariant

Size of \textit{runLen}

1. \[ \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \]
2. \[ \text{runLen}[n-1] > \text{runLen}[n] \]

If the above invariant is true for all \( n \) and \( \text{runLen}[n] \geq 16 \), then

\begin{itemize}
  \item (reversed) runlengths grow exponentially fast (\ldots 87 52 34 17 16)
  \item Runs do not overlap: few runs required to cover input array
\end{itemize}

Breaking the invariant - checking last 3 runs is insufficient

If (1) is false and \[ \text{runLen}[n-2] < \text{runLen}[n] \]: merge at idx \( n-2 \) and \( n-1 \), otherwise merge runs at indices \( n-1 \) and \( n \)

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
runLen & 120 & 80 & 45 & 30 \\
\hline
\end{tabular}
\end{center}
Breaking the invariant

Size of `runLen`

1.  
   \( \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n] \)

2.  
   \( \text{runLen}[n-1] > \text{runLen}[n] \)

If the above invariant is true for all \( n \) and \( \text{runLen}[n] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

If (1) is false and \( \text{runLen}[n-2] < \text{runLen}[n] \): merge at idx \( n-2 \) and \( n-1 \), otherwise merge runs at indices \( n-1 \) and \( n \)

\[
\begin{array}{c|c|c|c}
\text{runLen} & 120 & 80 & 45 \end{array}
\begin{array}{c}
\text{30}
\end{array}
\]
Our work (I)

Wrote program that generates testcase

- that exploits breaking the invariant, by generating too many “short” runs
- Triggers exception: insufficient size for runLen to store run lengths

<table>
<thead>
<tr>
<th>Language</th>
<th>Smallest array that triggers error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>65.536 ($2^{16}$)</td>
</tr>
<tr>
<td>Java</td>
<td>67.108.864 ($2^{26}$)</td>
</tr>
<tr>
<td>Python</td>
<td>562.949.953.421.312 ($2^{49}$)</td>
</tr>
</tbody>
</table>

Most powerful supercomputer (Tianhe-2) has $\approx 2^{50}$ bytes of mem.
Wrote program that generates testcase

- that exploits breaking the invariant, by generating too many “short” runs

- **Triggers exception: insufficient size for `runLen` to store run lengths**

<table>
<thead>
<tr>
<th>Language</th>
<th>Smallest array that triggers error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>65.536 ( (2^{16}) )</td>
</tr>
<tr>
<td>Java</td>
<td>67.108.864 ( (2^{26}) )</td>
</tr>
<tr>
<td>Python</td>
<td>562.949.953.421.312 ( (2^{49}) )</td>
</tr>
</tbody>
</table>

Most powerful supercomputer (Tianhe-2) has \( \approx 2^{50} \) bytes of mem.

Provided worst-case analysis of broken version

- Shows the actual minimally required `runLen.length`
Our work (II)

Fixed the algorithm

- Check that last 4 runs satisfy invariant
- Executed existing benchmarks (result: same performance) and unit tests (all passed)

```java
/** ...
   * merges adjacent runs until the stack invariants are reestablished:
   * 1. runLen[i - 3] > runLen[i - 2] + runLen[i - 1]
   * 2. runLen[i - 2] > runLen[i - 1]
   */

private void mergeCollapse() {
    while (stackSize > 1) {
        int n = stackSize - 2;
        if ( (n >= 1 && runLen[n-1] <= runLen[n] + runLen[n+1])
            || (n >= 2 && runLen[n-2] <= runLen[n-1] + runLen[n]) ) {
            if (runLen[n - 1] < runLen[n + 1])
                n--;
            else if (runLen[n] > runLen[n + 1]) {
                break; // Invariant is established
            }
        }
        mergeAt(n);
    }
}
```
Analyzing "Real" Software

“because truly understanding it essentially requires doing a formal correctness proof, it’s difficult to maintain”

“Yet another large mass of difficult code can make for a real maintenance burden after I’m dead”
- Tim Peters on Timsort, python-dev mailing list, 2002

Implementation uses features for performance that complicate analysis: break statements, low-level bitwise ops., arithmetic overflows
Analyzing “Real” Software

“because truly understanding it essentially requires doing a formal correctness proof, it’s difficult to maintain”

“Yet another large mass of difficult code can make for a real maintenance burden after I’m dead”

- Tim Peters on Timsort, python-dev mailing list, 2002

Implementation uses features for performance that complicate analysis: break statements, low-level bitwise ops., arithmetic overflows

Mechanically proved fixed version with KeY (Java theorem prover)

- absence of the bug, and all other run-time exceptions
- termination
- this requires: formal specifications for all functions
Method contracts

- precondition (\texttt{requires}): condition on the input
- postcondition (\texttt{ensures}): condition on the output / result

```java
/*@ requires
  @ stackSize > 0;
  @ ensures
  @ (\forall int i; 0<=i && i<stackSize-2;
  @     elemInv(runLen, i, 16))
  @ && elemBiggerThanNext(runLen, stackSize-2)
  @*/
private void mergeCollapse()
```
Specifying Java Code with JML

Method contracts

- **precondition** (*requires*): condition on the input
- **postcondition** (*ensures*): condition on the output / result

```java
/*@ requires
@   stackSize > 0;
@ ensures
@   (\forall int i; 0<=i && i<stackSize-2;
@       elemInv(runLen, i, 16))
@   && elemBiggerThanNext(runLen, stackSize-2)
@*/
private void mergeCollapse()
```

Class Invariant

Property that all instances of a class must satisfy before and after every method (call)

- Can be **assumed** in method precondition
- Must be **established** at all call sites and method postcondition
/*@
   @ invariant
   @
   @  runBase.length == runLen.length
   @  && (a.length < 120  ==> runLen.length==4)
   @  && (a.length >= 120 && a.length < 1542 ==> runLen.length==9)
   @  && (a.length >= 1542 && a.length<119151 ==> runLen.length==18)
   @  && (a.length >= 119151 ==> runLen.length==39)
   @  && (0 <= stackSize && stackSize <= runLen.length)
   @  && (\forall int i; 0<=i && i<stackSize-4;
         @
         @    elemInv(runLen, i, 16))
   @  && (elemLargerThanBound(runBase, 0, 0))
   @  && (\forall int i; 0<=i && i<stackSize-1;
         @
         @    runBase[i] + runLen[i] == runBase[i+1]);
   @*/

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>elemBiggerThanNext2(arr, idx)</td>
<td>(0 ≤ idx ∧ idx + 2 &lt; arr.length) → arr[idx] &gt; arr[idx + 1] + arr[idx + 2]</td>
</tr>
<tr>
<td>elemBiggerThanNext(arr, idx)</td>
<td>0 ≤ idx ∧ idx + 1 &lt; arr.length → arr[idx] &gt; arr[idx + 1]</td>
</tr>
<tr>
<td>elemLargerThanBound(arr, idx, v)</td>
<td>0 ≤ idx &lt; arr.length → arr[idx] ≥ v</td>
</tr>
<tr>
<td>elemInv(arr, idx, v)</td>
<td>elemBiggerThanNext2(arr, idx) ∧ elemBiggerThanNext(arr, idx) ∧ elemLargerThanBound(arr, idx, v)</td>
</tr>
</tbody>
</table>
Class Invariant (simplified)

```java
/*@ invariant
runBase.length == runLen.length
@
&& (a.length < 120 ==> runLen.length==4)
&& (a.length >= 120 && a.length < 1542 ==> runLen.length==9)
&& (a.length >= 1542 && a.length<119151 ==> runLen.length==18)
&& (a.length >= 119151 ==> runLen.length==39)
@ && (0 <= stackSize && stackSize <= runLen.length)
@ && (∀ int i; 0<=i && i<stackSize-4;
    @
    elemInv(runLen, i, 16))
@ && (elemLargerThanBound(runBase, 0, 0))
@ && (∀ int i; 0<=i && i<stackSize-1;
    @
    runBase[i] + runLen[i] == runBase[i+1]);
@*/
```

Length of runlen in terms of input length
Class Invariant (simplified)

```java
/*@ invariant 
  @ runBase.length == runLen.length 
  @ && (a.length < 120  
        ===> runLen.length==4) 
  @ && (a.length >= 120 && a.length < 1542  
        ===> runLen.length==9) 
  @ && (a.length >= 1542 && a.length<119151  
        ===> runLen.length==18) 
  @ && (a.length >= 119151  
        ===> runLen.length==39) 
  @ && (0 <= stackSize && stackSize <= runLen.length) 
  @ && (\forall int i; 0<=i && i<stackSize-4; 
        elemInv(runLen, i, 16)) 
  @ && (elemLargerThanBound(runBase, 0, 0)) 
  @ && (\forall int i; 0<=i && i<stackSize-1; 
        runBase[i] + runLen[i] == runBase[i+1]); 
/*@ */
```

Bounds on stackSize (in-use part of \texttt{runLen})
Class Invariant (simplified)

```java
/*@ invariant
  runBase.length == runLen.length
  @& (a.length < 120 ==> runLen.length==4)
  @& (a.length >= 120 && a.length < 1542 ==> runLen.length==9)
  @& (a.length >= 1542 && a.length<119151 ==> runLen.length==18)
  @& (a.length >= 119151 ==> runLen.length==39)
  @& (0 <= stackSize && stackSize <= runLen.length)
  @& (forall int i; 0<i && i<stackSize-4; elemInv(runLen, i, 16))
  @& (elemLargerThanBound(runBase, 0, 0))
  @& (forall int i; 0<i && i<stackSize-1; runBase[i] + runLen[i] == runBase[i+1]);
*/

All but the last 4 runs satisfy the invariant while merging
/*@ invariant */
@ runBase.length == runLen.length
@ && (a.length < 120       ==> runLen.length==4)
@ && (a.length >= 120 && a.length < 1542 ==> runLen.length==9)
@ && (a.length >= 1542 && a.length<119151 ==> runLen.length==18)
@ && (a.length >= 119151  ==> runLen.length==39)
@ && (0 <= stackSize && stackSize <= runLen.length)
@ && (\forall int i; 0<=i && i<stackSize-4;
@     elemInv(runLen, i, 16))
@ && (elemLargerThanBound(runBase, 0, 0))
@ && (\forall int i; 0<=i && i<stackSize-1;
@     runBase[i] + runLen[i] == runBase[i+1]);
@*/
Class Invariant (simplified)

1 /*@ invariant 
2 @ runBase.length == runLen.length 
3 @ && (a.length < 120 ==> runLen.length==4) 
4 @ && (a.length >= 120 && a.length < 1542 ==> runLen.length==9) 
5 @ && (a.length >= 1542 && a.length<119151 ==> runLen.length==18) 
6 @ && (a.length >= 119151 ==> runLen.length==39) 
7 @ && (0 <= stackSize && stackSize <= runLen.length) 
8 @ && (@forall int i; 0<i && i<stackSize-4; 
9 @ elemInv(runLen, i, 16)) 
10 @ && (elemLargerThanBound(runBase, 0, 0)) 
11 @ && (@forall int i; 0<=i && i<stackSize-1; 
12 @ runBase[i] + runLen[i] == runBase[i+1]); 
13 */

There are no gaps between consecutive runs
Loop Invariant (simplified)

```latex
/*@ loop_invariant
  @ \forall int i; 0\leq i \&\& i<\text{stackSize}-4;
  @ \quad \text{elemInv}(\text{runLen}, i, 16);
  @ */
```

The main verif. condition (simplified)

\[
(\text{loop-inv} \&\& n=\text{stackSize}-2 \&\& n \geq 0
\&\& n \geq 1 \implies \text{runLen}[n-1] > \text{runLen}[n] + \text{runLen}[n+1]
\&\& n \geq 2 \implies \text{runLen}[n-2] > \text{runLen}[n-1] + \text{runLen}[n]
\&\& \text{runLen}[n] > \text{runLen}[n+1])
\implies \text{ensures(mergeCollapse)}
\]

Recall that \text{ensures(mergeCollapse)} is (substituting stackSize-2==n):

\[
(\forall \text{int } i; 0\leq i \&\& i<n; \text{elemInv}(\text{runLen}, i, 16))
\&\& \text{elemBiggerThanNext}(\text{runLen}, n)
\]
/@ normal_behavior
@ requires
@ (runLen > 0 && runBase >= 0)
@ && (stackSize > 0 ==> runBase ==
@ this.runBase[stackSize-1]+this.runLen[stackSize-1])
@ && (runLen + runBase <= a.length)
@ && (forall int i; 0<=i && i<stackSize-2;
@ elemInv(this.runLen,i,16))
@ && elemBiggerThanNext(this.runLen, stackSize-2)
@ && elemLargerThanBound(this.runLen, stackSize-1, 16)
@ ensures
@ this.runBase[old(stackSize)] == runBase
@ && this.runLen[old(stackSize)] == runLen
@ && stackSize == old(stackSize)+1;
@*/

private void pushRun(int runBase, int runLen) {
    this.runBase[stackSize] = runBase;
    this.runLen[stackSize] = runLen;
    stackSize++;
}
The new run has positive length and starts directly after the last run
pushRun contract (simplified)

```java
/*@ normal_behavior
@ requires
@   (runLen > 0 && runBase >= 0)
@   && (stackSize > 0 ==> runBase ==
@     this.runBase[stackSize-1]+this.runLen[stackSize-1])
@   && (runLen + runBase <= a.length)
@   && (\forall int i; 0<=i && i<stackSize-2;
@     elemInv(this.runLen,i,16))
@   && elemBiggerThanNext(this.runLen, stackSize-2)
@   && elemLargerThanBound(this.runLen, stackSize-1, 16)
@ ensures
@   this.runBase[\old(stackSize)] == runBase
@   && this.runLen[\old(stackSize)] == runLen
@   && stackSize == \old(stackSize)+1;
@*/
private void pushRun(int runBase, int runLen) {
    this.runBase[stackSize] = runBase;
    this.runLen[stackSize] = runLen;
    stackSize++;
}
```

The new run cannot extend beyond length of the input array
pushRun contract (simplified)

```java
/*@ normal_behavior
@ requires
@   (runLen > 0 && runBase >= 0)
@   && (stackSize > 0 ==> runBase ==
@   this.runBase[stackSize-1]+this.runLen[stackSize-1])
@   && (runLen + runBase <= a.length)
@   && (forall int i; 0 <= i && i < stackSize-2;
@     elemInv(this.runLen,i,16))
@   && elemBiggerThanNext(this.runLen, stackSize-2)
@   && elemLargerThanBound(this.runLen, stackSize-1, 16)
@ ensures
@   this.runBase[old(stackSize)] == runBase
@   && this.runLen[old(stackSize)] == runLen
@   && stackSize == old(stackSize)+1;
@*/
private void pushRun(int runBase, int runLen) {
    this.runBase[stackSize] = runBase;
    this.runLen[stackSize] = runLen;
    stackSize++;
}
```

The invariant is satisfied by all runs
pushRun contract (simplified)

```java
/*@ normal_behavior
@ requires
@   (runLen > 0 && runBase >= 0)
@   && (stackSize > 0 ==> runBase == 
@   this.runBase[stackSize-1]+this.runLen[stackSize-1])
@   && (runLen + runBase <= a.length)
@   && (\forall int i; 0<=i && i<stackSize-2;
@   elemInv(this.runLen,i,16))
@   && elemBiggerThanNext(this.runLen, stackSize-2)
@   && elemLargerThanBound(this.runLen, stackSize-1, 16)
@ ensures
@   this.runBase[\old(stackSize)] == runBase
@   && this.runLen[\old(stackSize)] == runLen
@   && stackSize == \old(stackSize)+1;
@*/
private void pushRun(int runBase, int runLen) {
    this.runBase[stackSize] = runBase;
    this.runLen[stackSize] = runLen;
    stackSize++;}
```

The new run is stored at index stackSize-1
No ArrayIndexOutOfBoundsException if

requires(pushRun) && cl. invariant \implies stackSize < len.length
No ArrayIndexOutOfBoundsException if

\[ \text{requires}(\text{pushRun}) \land \text{cl. invariant} \implies \text{stackSize} < \text{len.length} \]

**Proof.**

Note first: \( \text{cl. invariant} \implies \text{stackSize} \leq \text{len.length} \).

Assume by contradiction that \( \text{stackSize} = \text{len.length} \) and do a case distinction on \( \text{a.length} \). We treat \( \text{a.length} \leq 119 \):

1. \( \text{len.length} = 4 \) (from \( \text{cl. invariant} \), line 3).
2. **Abbreviate** \( \text{len}[0]+...+\text{len}[3] = \text{SUM} \), then (\( \text{pushRun} \) line 7–10)
   
   \[
   \text{len}[3] \geq 16, \text{len}[2] \geq 17, \text{len}[1] \geq 34 \text{ and } \text{len}[0] \geq 52.
   
   Therefore: \( \text{SUM} \geq 16+17+34+52 = 119 \)
3. \( \text{base}[3] + \text{len}[3] = \text{base}[0] + \text{SUM} \) (from \( \text{cl. invariant} \), line 11–12)
4. **Previous line, with** \( \text{pushRun} \) line 4–5 implies:
   
   \[
   \text{runBase} + \text{runLen} = \text{base}[0] + \text{SUM} + \text{runLen}
   
   \]
5. **But** \( \text{base}[0] \geq 0 \) (\( \text{cl invariant} \) line 10) and \( \text{runLen} > 0 \) (\( \text{pushrun} \) line 3), contradicting \( \text{runBase} + \text{runLen} \leq 119 \) (\( \text{pushRun} \) line 6)
One proof step in KeY
### Proof Stats - summary

<table>
<thead>
<tr>
<th></th>
<th># Rule Apps</th>
<th># Interactive</th>
<th>LoSpec</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>total</strong></td>
<td>2.211.263</td>
<td>5.029</td>
<td>334</td>
<td>333</td>
</tr>
<tr>
<td><strong>pushRun</strong></td>
<td>26.248</td>
<td>94</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td><strong>mergeCollapse</strong></td>
<td>415.133</td>
<td>1.529</td>
<td>47</td>
<td>13</td>
</tr>
</tbody>
</table>
Evaluation

Proof Stats - summary

<table>
<thead>
<tr>
<th></th>
<th># Rule Apps</th>
<th># Interactive</th>
<th>LoSpec</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>2.211.263</td>
<td>5.029</td>
<td>334</td>
<td>333</td>
</tr>
<tr>
<td>pushRun</td>
<td>26.248</td>
<td>94</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>mergeCollapse</td>
<td>415.133</td>
<td>1.529</td>
<td>47</td>
<td>13</td>
</tr>
</tbody>
</table>

Evaluation of the problem

- Bug unlikely to be encountered by accident
- Possible security hazard: bug may be exploitable in DoS attack
- Extensive testing unable to expose bug:
  input size too large, structure too complex
- Manual code reviews (Google) unable to expose bug
- Core libraries in widely used languages can contain subtle bugs undetected for years
- Scientific paper (CAV 2015), articles (ERCIM, Bits & Chips)
- Published blog post viewed 361274 times
Responses: general public

- Scientific paper (CAV 2015), articles (ERCIM, Bits & Chips)
- Published blog post viewed 361274 times

Joshua Bloch
@pabloch

Congratulations to Stijn de Gouw et al. for finding and fixing a bug in TimSort using formal methods!

1233 AM - 25 Feb 2015
128 retweets 65 favorites

Damian Grzski
@dgryski

JetBrains IntelliJ IDEA Blog (March 10, 2015):
"The first of its kind, this result is both an important development for the Java community and a proof of concept for the feasibility of formal verification and automated theorem proving. Perhaps more importantly, the tool was used to detect and identify this bug is completely open source and available to try yourself!"

Stijn de Gouw
Timsort
Oslo, June 22, 2015 26 / 1
Java

- Submitted bug report to Java issue tracker
Responses: developer communities

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`

```java
int stackLen = (len < 120 ? 5 :
    len < 1542 ? 10 :
    len < 119151 ? 19 : 40);

runBase = new int[stackLen];
runLen = new int[stackLen];
```
Responses: developer communities

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length`
  based on worst-case analysis

Discussion on OpenJDK mailing list

*Stack length increased previously by JDK-8011944 was insufficient for some cases. Please review and push - Lev Priima, 11 Feb 2015

```java
int stackLen = (len < 120 ? 5 :
  len < 1542 ? 10 :
  len < 119151 ? 24 :
  40 49 );
runBase = new int[stackLen];
runLen = new int[stackLen];
```
Responses: developer communities

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis

Discussion on OpenJDK mailing list

Stack length increased previously by JDK-8011944 was insufficient for some cases. Please review and push
- Lev Priima, 11 Feb 2015

Hi Lev, The fix looks fine. Did you consider the improvements suggested in the paper to reestablish the invariant?
- Roger Riggs, Feb 11, 2015

Just briefly looked at it, w/o evaluating formal proof ... 
- Lev Priima, Feb 11, 2015
Responses: developer communities

Java
- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis
- Purported class invariant still broken
- Not amenable to mechanic verification

Python
- Bug report filed by Tim Peters
- Bug fixed by checking last 4 runs (verified version)

Android
- No bug report or fix so far
Formal methods work!
## Useful links

**Blog post**

http://tinyurl.com/timsort-bug

**Website with full paper, test programs and proofs**

http://www.envisage-project.eu/timsort-specification-and-verification

**KeY (Java theorem prover)**

http://www.key-project.org

**Timsort description**

http://bugs.python.org/file4451/timsort.txt

**OpenJDK dev discussion**

http://mail.openjdk.java.net/pipermail/core-libs-dev/2015-February/thread.html#31405