For a Few Dollars More Verified Fine-Grained Algorithm Analysis Down to LLVM

Maximilian P. L. Haslbeck ¹, Peter Lammich ²

¹Technical University of Munich

haslbema@in.tum.de

²University of Twente

p.lammich@utwente.nl

April, 2021

C++ Standard

```
25.4.1
         Sorting
                                                                                            [alg.sort]
25.4.1.1 sort
                                                                                                 [sort]
template<class RandomAccessIterator>
  void sort(RandomAccessIterator first. RandomAccessIterator last):
template<class RandomAccessIterator, class Compare>
  void sort(RandomAccessIterator first, RandomAccessIterator last,
            Compare comp);
     Effects: Sorts the elements in the range [first.last).
     Requires: RandomAccessIterator shall satisfy the requirements of ValueSwappable (17.6.3.2). The
     type of *first shall satisfy the requirements of MoveConstructible (Table 20) and of MoveAssignable
     (Table 22).
     Complexity: \mathcal{O}(N \log(N)) (where N == last - first) comparisons.
```

Competitive uint64 babelle-LLVVM (notioned) blocke-1-LLVVM (notioned) blocke-1-LLVVM (notioned) blocke-1-LLVVM (notioned)

C++ Standard

25.4.1 Sorting

[alg.sort]

template<class RandomAccessIterator>

25.4.1.1 sort

void sort(Random&ccessIterator first, Random&ccessIterator last);

template<class RandomAccessIterator, class Compare>

void sort(RandomAccessIterator first, RandomAccessIterator last, Compare comp):

Effects: Sorts the elements in the range [first,last).

Requires: RandonAccessIterator shall satisfy the requirements of ValueSwappable (17.6.3.2). The type of *first shall satisfy the requirements of MoveConstructible (Table 20) and of MoveAssignable (Table 22).

Complexity: $\mathcal{O}(N \log(N))$ (where N == last - first) comparisons.

Competitive uint64 u

C++ Standard

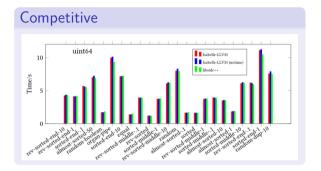
25.4.1 Sorting [alg.sort]

25.4.1.1 sort [sort]

templatectlass NandonAccessiterator)
void sort(DanienAccessiterator)
void sort(DanienAccessiterator).
Anadonaccessiterator last);
templatectlass RandonAccessiterator, class Compare)
void sort(DanienAccessiterator, class Compare)
void sort(DanienAccessiterator, class Compare)
void sort(DanienAccessiterator, class Compare)
void sort(DanienAccessiterator, class Compare)
void sort(DanienAccessiterator)
Reporters. Randonaccessiterator almi suido; the requirements of ValueDanienAccessiterator, class class class compared to the Compared Com

Verified

```
 \begin{aligned} & \{ \mathsf{array}_{\mathcal{A}} \ \mathsf{p} \ \mathsf{xs}_0 \ \star \ \mathsf{snat}_{\mathcal{A}} \ \mathsf{l}_\dagger \ \mathsf{l} \ \star \ \mathsf{snat}_{\mathcal{A}} \ \mathsf{h}_\dagger \ \mathsf{h} \ \star \ \uparrow (\mathsf{l} \le \mathsf{h} \ \land \ \mathsf{h} < |\mathsf{xs}_0|) \ \star \ \$ (\mathsf{introsort\_impl}_{\mathit{cost}} \ (\mathsf{h-l})) \} \\ & \mathsf{introsort\_impl} \ \mathsf{p} \ \mathsf{l}_\dagger \ \mathsf{h}_\dagger \\ & \{ \lambda \mathsf{r}. \ \exists_{\mathcal{A}} \mathsf{xs}. \ \mathsf{array}_{\mathcal{A}} \ \mathsf{r} \ \mathsf{xs} \ \star \ \uparrow (\mathsf{slice\_sort\_aux} \ \mathsf{xs}_0 \ \mathsf{l} \ \mathsf{h} \ \mathsf{xs}) \ \star \ \mathsf{snat}_{\mathcal{A}} \ \mathsf{l}_\dagger \ \mathsf{l} \ \star \ \mathsf{snat}_{\mathcal{A}} \ \mathsf{h}_\dagger \ \mathsf{h} \} \end{aligned}
```



```
C++ Standard

25.4.1 Sorting [alg.cort]

25.4.1.1 sort

tempitatecclass RandonAccessiterator)

void orriChandraccessiterator first, RandonAccessiterator last);

tempitatecclass RandonAccessiterator; class Compared

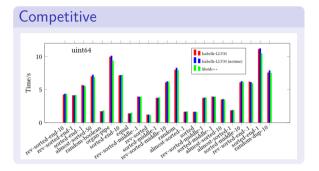
void sort(RandonAccessiterator; class Compared)

Reflects Sortis the channels in the range (first, last)

Reputers RandonAccessiterator; shall satisfy the requirements of ValueDavappahle (17.63.2). The type of first shall satisfy the requirements of NoveConstructivities (Table 20) and of NoveAssignable (Table 20) a
```

Verified

```
 \begin{aligned} & \{ \mathsf{array}_A \ \mathsf{p} \ \mathsf{xs_0} \ \star \ \mathsf{snat}_A \ \mathsf{l}_\dagger \ \mathsf{l} \ \star \ \mathsf{snat}_A \ \mathsf{h}_\dagger \ \mathsf{h} \ \star \ \uparrow (\mathsf{I} \le \mathsf{h} \ \land \ \mathsf{h} < |\mathsf{xs_0}|) \ \star \ \$ (\mathsf{introsort\_impl}_{cost} \ (\mathsf{h}-\mathsf{I})) \} \\ & \mathsf{introsort\_impl} \ \mathsf{p} \ \mathsf{l}_\dagger \ \mathsf{h}_\dagger \\ & \{ \lambda \mathsf{r}. \ \exists_A \mathsf{xs}. \ \mathsf{array}_A \ \mathsf{r} \ \mathsf{xs} \ \star \ \uparrow (\mathsf{slice\_sort\_aux} \ \mathsf{xs_0} \ \mathsf{I} \ \mathsf{h} \ \mathsf{xs}) \ \star \ \mathsf{snat}_A \ \mathsf{l}_\dagger \ \mathsf{I} \ \star \ \mathsf{snat}_A \ \mathsf{h}_\dagger \ \mathsf{h} \} \\ & \mathsf{introsort\_impl}_{cost} \quad \in O(n \log n) \end{aligned}
```



```
C++ Standard

25.4.1 Sorting [alg.sort]
25.4.1.1 sort [sort]
templatecclass Bandondcomalterators
void swrt(Bandondcomalterator) first, Bandondcomesiterator last);
templatecclass Bandondcomalterator, class Compared
void swrt(Bandondcomalterator, class Compared
void swrt(Bandondcomalterator, class Compared)
void swrt(Bandondcomalterator, Class Compared)
void swrt(Bandondcomalterator, Class Compared)
void swrt(Bandondcomalterator, Class Compared)
void swrt(Bandondcomalterator, Sandondcomalterator, Sandondcomalt
```

Verified

```
 \begin{aligned} & \{\mathsf{array}_A \ \mathsf{p} \ \mathsf{xs_0} \ \star \ \mathsf{snat}_A \ \mathsf{l}_\dagger \ \mathsf{l} \ \star \ \mathsf{snat}_A \ \mathsf{h}_\dagger \ \mathsf{h} \ \star \ \uparrow (\mathsf{l} \le \mathsf{h} \ \land \ \mathsf{h} < |\mathsf{xs_0}|) \ \star \ \$ (\mathsf{introsort\_impl}_{cost} \ (\mathsf{h}-\mathsf{l})) \} \\ & \mathsf{introsort\_impl} \ \mathsf{p} \ \mathsf{l}_\dagger \ \mathsf{h}_\dagger \\ & \{\lambda \mathsf{r}. \ \exists_A \mathsf{xs}. \ \mathsf{array}_A \ \mathsf{r} \ \mathsf{xs} \ \star \ \uparrow (\mathsf{slice\_sort\_aux} \ \mathsf{xs_0} \ \mathsf{l} \ \mathsf{h} \ \mathsf{xs}) \ \star \ \mathsf{snat}_A \ \mathsf{l}_\dagger \ \mathsf{l} \ \star \ \mathsf{snat}_A \ \mathsf{h}_\dagger \ \mathsf{h} \} \end{aligned}   (\lambda \mathsf{n}. \ \Sigma \mathsf{c}. \ \mathsf{introsort\_impl}_{cost} \ \mathsf{n} \ \mathsf{c}) \in \mathsf{O}(n \log n)
```

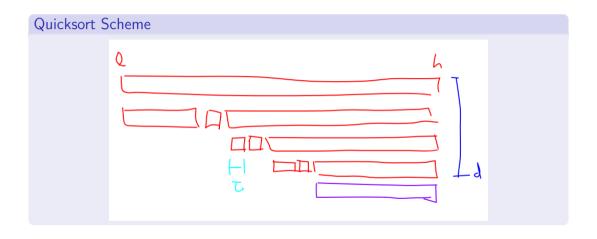
Top-Down Approach

- First verification of a competitive implementation of Introsort with Time Bound
- Stepwise Refinement Calculus with Resource Currencies
- Correctness-and-Time-Bound-Preserving Synthesis Mechanism
- LLVM Semantics with Cost Model
- ullet Basic Reasoning Infrastructure (SL + TC)

Top-Down Approach

- \bullet First verification of a competitive implementation of $\operatorname{INTROSORT}$ with Time Bound
- Stepwise Refinement Calculus with Resource Currencies
- Correctness-and-Time-Bound-Preserving Synthesis Mechanism
- LLVM Semantics with Cost Model
- ullet Basic Reasoning Infrastructure (SL + TC)

Introsort



Introsort

Musser's Pseudocode

```
Algorithm Introsort(A, f, b)
                                   Inputs: A. a random access data structure containing the sequence
                                      of data to be sorted, in positions A[f], ..., A[b-1];
                                     f, the first position of the sequence
                                     b, the first position beyond the end of the sequence
                                   Output: A is permuted so that A[f] \le A[f+1] \le ... \le A[b-1]
      Introsort_Loop(A, f, b, 2 * Floor_Lg(b - f))
      INSERTION_SORT(A, f, b)
Algorithm INTROSORT_LOOP(A, f, b, depth_limit)
                                   Inputs: A. f. b as in Introsort:
                                     depth_limit, a nonnegative integer
                                  Output: A is permuted so that A[i] < A[i]
                                     for all i, i: f \le i \le i \le b and size_threshold \le i - i
      while b - f > size threshold
           do if depth_limit = 0
                       then HEAPSORT(A, f, b)
                             return
                 depth\_limit := depth\_limit - 1
                 \label{eq:definition} D := \operatorname{PARTITION}(A, \, f, \, b, \, \operatorname{MEDIAN\_OF\_3}(A[f], \, A[f+(b-f)/2], \, A[b-1]))
                 INTROSORT_LOOP(A, p. b, depth_limit)
                 b := p
```

Quicksort Scheme

```
1 introsort xs I h =
2 n ← return h-I;
```

 $\mathtt{if}_c \; \mathtt{n} > 1 \; \mathtt{then}$

return xs else return xs

 $xs \leftarrow almost_sort_{spec} xs l h; ($_{almost_sort})$ $xs \leftarrow final_sort_{spec} xs l h; ($_{final_sort})$

 $(\$_{sub})$ $(\$_{lt})$

```
introsort xs | h = slice\_sort_{spec} $\$ slice_sort \left\{ sub \} \\ if_c \ n > 1 \text{ then } \quad (\$_{lt}) \\ xs \left \text{ almost_sort_spec} \text{ xs | h; } (\$_{almost_sort})
```

 $xs \leftarrow final_sort_{spec} xs l h;$ (\$final_sort)

return xs else return xs

```
1 introsort xs | h = 
2    n \leftarrow return h-|;    ($sub)
3    if c n > 1 then    ($lt)
4    xs \leftarrow almost_sortspec xs | h; ($slinest sort)

• introsort \leq \psi_C E_1 (slice_sortspec $slice_sort)

• E_1 :: currency \rightarrow currency \rightarrow \mathbb{N}
```

 $(\$_{final_sort})$

 $xs \leftarrow final_sort_{spec} xs l h$

return xs else return xs

```
1 introsort xs | h = 

2    n \leftarrow return h-l; ($sub)

3    if c n > 1 then ($slice_sort_spec_start) | E1 :: currency \rightarrow currency \rightarrow N

4    xs \leftarrow almost_sort_spec_xs | h; ($slice_sort) | E1 slice_sort = $sub_start_spec_start | Since_sort | Since_so
```

return xs else return xs + \$almost sort + \$final sort

```
1. introsort_rec xs | h d = 

2. assert (| \leq h);

3. n \leftarrow h - |; ($sub)

4. if<sub>c</sub> n > \tau then ($t<sub>t</sub>)

5. slice_sort<sub>spec</sub> xs | h ($sort<sub>c</sub> (\mu (h-|)))
```

 $(\$_{sub})$

 $(xs, m) \leftarrow partition_{spec} xs l h; ($partition_c (h-l))$

 $d' \leftarrow d - 1$;

return xs

else return xs

xs ← introsort_rec xs I m d';

 $xs \leftarrow introsort_rec xs m h d';$

10

11

12

13

 $(\$_{eq})$ $(\$_{sort_c} (\mu (h-l)))$

(\$_{sub})

 $(xs, m) \leftarrow partition_{spec} xs l h; ($partition_c (h-l))$

 $if_c d = 0 then$

 $d' \leftarrow d - 1$;

return xs

else return xs

else

10

11 12

13

slice_sort_{spec} xs I h

xs ← introsort_rec xs I m d'; xs ← introsort_rec xs m h d';

```
assert (I \le h);
n \leftarrow h - I; (\$_{sub}) introsort_rec \le \bigvee_C E_2 (almost_sort_spec \$_{almost\_sort})

if _c n > \tau then (\$_{tb})
if _c d = 0 then (\$_{eq})
slice_sort_spec xs | h (\$_{sort_c} (\mu (h-I)))
else
(xs, m) \leftarrow partition_spec xs | h; (\$_{partition_c} (h-I))
```

(\$_{sub})

• $\mu n = n \log n$

introsort rec xs l h d =

 $d' \leftarrow d - 1$:

return xs

else return xs

13

xs ← introsort_rec xs l m d'; xs ← introsort_rec xs m h d';

```
assert (I < h):
                                                           • introsort\_rec \leq \bigcup_C E_2 (almost\_sort_{spec} \$_{almost\_sort})
n \leftarrow h - 1:
                                      ($<sub>sub</sub>)
if n > \tau then
                                       (\$_{lt})
                                                           • introsort rec_{cost} (n. d) =
 if_c d = 0 then
                                       (\$_{ea})
                                                                 \$_{\text{sort}_c}(\mu n) + \$_{\text{partition}_c} d * n
```

 $(\$_{sort_c} (\mu (h-I)))$ slice_sort_{spec} xs I h else

(\$_{sub})

 $(xs, m) \leftarrow partition_{spec} \times s \mid h; \quad (\$_{partition_c} \quad (h-l))$

 $\bullet \mu n = n \log n$

 $xs \leftarrow introsort_rec xs I m d';$ $xs \leftarrow introsort_rec xs m h d';$

introsort rec xs I h d =

 $d' \leftarrow d - 1$

return vs

else return vs

13

 $+((d+1)*n)*(\$_{if}\ 2+\$_{call}\ 2+\$_{eq}\ +\$_{lt}\ +\$_{sub}\ 2)$

```
assert (I < h):
                                                       • introsort_rec \leq \Downarrow_C E_2 (almost_sort_spec _{almost_sort})
n \leftarrow h - 1:
                                    ($<sub>sub</sub>)
if n > \tau then
                                    (\$_{lt})
                                                       • introsort rec_{cost} (n. d) =
 if_c d = 0 then
                                    ($00)
```

introsort rec xs l h d = $\bullet \mu n = n \log n$

 $xs \leftarrow introsort_rec xs m h d';$

return vs

else return vs

13

 $(\$_{sort_c} (\mu (h-I)))$ $\$_{sort_c} (\mu n) + \$_{partition_c} d * n$

else

 $+((d+1)*n)*(\$_{if} 2 + \$_{call} 2 + \$_{eq} + \$_{lt} + \$_{sub} 2)$

 $(xs, m) \leftarrow partition_{spec} \times s \mid h; \quad (\$_{partition_c} (h-l))$

 $d' \leftarrow d - 1$ (\$_{sub})

• E_2 almost sort = introsort rec_{cost} (h - l, d) $xs \leftarrow introsort_rec xs I m d';$

Stepwise Refinement

- Refine $slice_sort_{spec}$ with HEAPSORT in $O(n \log n)$
- Refine $partition_{spec}$ in O(n)
- Refine $final_sort_{spec}$ with INSERTIONSORT in $O(\tau \ n)$
 - Unguarded Optimization
- Synthesis to LLVM Code

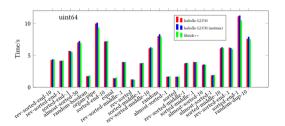
 $(\textit{introsort}_3, \textit{slice_sort}_{\textit{spec}} \; \textit{introsort_impl}_{\textit{cost}}) \in \textit{Id} \rightarrow \textit{Id} \rightarrow \textit{Id} \rightarrow \textit{Id}$

 $(introsort_3, slice_sort_{spec} \ introsort_impl_{cost}) \in Id \rightarrow Id \rightarrow Id \rightarrow Id$ $(introsort_{impl}, introsort_3) \in array_A \rightarrow snat_A \rightarrow snat_A \rightarrow array_A$

```
 \begin{array}{l} (\textit{introsort}_3, \, \textit{slice\_sort}_{\textit{spec}} \, \, \textit{introsort\_impl}_{\textit{cost}}) \in \textit{Id} \rightarrow \textit{Id} \rightarrow \textit{Id} \rightarrow \textit{Id} \\ \\ (\textit{introsort}_{\textit{impl}}, \, \textit{introsort}_3) \in \textit{array}_A \rightarrow \textit{snat}_A \rightarrow \textit{snat}_A \rightarrow \textit{array}_A \\ \\ \{\textit{array}_A \, p \, xs_0 \star \textit{snat}_A \, l_\dagger \, l \star \textit{snat}_A \, h_\dagger \, h \star \uparrow (l \leq h \, \land \, h < |xs_0|) \star \$ (\textit{introsort\_impl}_{\textit{cost}} \, (h-l)) \} \\ \quad \text{introsort\_impl} \, p \, l_\dagger \, h_\dagger \\ \{\lambda r. \, \exists_A xs. \, \textit{array}_A \, r \, xs \, \star \uparrow (\textit{slice} \, \textit{sort} \, \textit{aux} \, xs_0 \, l \, h \, xs) \star \textit{snat}_A \, l_\dagger \, l \star \textit{snat}_A \, h_\dagger \, h \} \\ \end{array}
```

```
(introsort<sub>3</sub>, slice_sort<sub>spec</sub> introsort_impl<sub>cost</sub>) \in Id \rightarrow Id \rightarrow Id \rightarrow Id \rightarrow Id (introsort<sub>impl</sub>, introsort<sub>3</sub>) \in array<sub>A</sub> \rightarrow snat<sub>A</sub> \rightarrow snat<sub>A</sub> \rightarrow array<sub>A</sub> (array<sub>A</sub> p xs<sub>0</sub> * snat<sub>A</sub> l<sub>†</sub> l * snat<sub>A</sub> h<sub>†</sub> h * \uparrow(l \leq h \land h < |xs<sub>0</sub>|) * $(introsort_impl<sub>cost</sub> (h-l))} introsort_impl p l<sub>†</sub> h<sub>†</sub> {\lambdar. \exists_Axs. array<sub>A</sub> r xs * \uparrow(slice_sort_aux xs<sub>0</sub> l h xs) * snat<sub>A</sub> l<sub>†</sub> l * snat<sub>A</sub> h<sub>†</sub> h} (\lambdan. \Sigmac. introsort_impl<sub>cost</sub> n c) \in O(n log n) (\lambdan. introsort_impl<sub>cost</sub> n cmp) \in O(n log n)
```

```
(introsort<sub>3</sub>, slice_sort<sub>spec</sub> introsort_impl<sub>cost</sub>) \in Id \rightarrow Id \rightarrow Id \rightarrow Id \rightarrow Id (introsort<sub>impl</sub>, introsort<sub>3</sub>) \in array<sub>A</sub> \rightarrow snat<sub>A</sub> \rightarrow snat<sub>A</sub> \rightarrow array<sub>A</sub> 
{array<sub>A</sub> p xs<sub>0</sub> * snat<sub>A</sub> l<sub>†</sub> l * snat<sub>A</sub> h<sub>†</sub> h * \uparrow(l \leq h \land h < |xs<sub>0</sub>|) * $(introsort_impl<sub>cost</sub> (h-l))} introsort_impl p l<sub>†</sub> h<sub>†</sub> 
{\lambdar. \exists<sub>A</sub>xs. array<sub>A</sub> r xs * \uparrow(slice_sort_aux xs<sub>0</sub> l h xs) * snat<sub>A</sub> l<sub>†</sub> l * snat<sub>A</sub> h<sub>†</sub> h} 
(\lambdan. \Sigmac. introsort_impl<sub>cost</sub> n c) \in O(n log n) (\lambdan. introsort_impl<sub>cost</sub> n cmp) \in O(n log n)
```



In the Paper

- Nondeterministic Result Monad with Time (NREST)
- Refinement Patterns + Automation
- Synthesis Mechanism (Connecting NREST with LLVM Monad)
- LLVM Semantics + Cost Model
 - Basic Reasoning Infrastructure

Conclusion

- Verification of a State-of-the-Art Sorting Algorithm
 - Competitive and Verified
 - Stepwise Refinement with Resource Analysis
- Future Work
 - Improved Tooling
 - Further Case Studies
 - Other Resources: e.g. Stack Size

Conclusion

- Verification of a State-of-the-Art Sorting Algorithm
 - Competitive and Verified
 - Stepwise Refinement with Resource Analysis
- Future Work
 - Improved Tooling
 - Further Case Studies
 - Other Resources: e.g. Stack Size

Formalization & Benchmarks & More:

https://www21.in.tum.de/~haslbema/llvm-time/



Thank you!