Expected linear time sorting for large integers

Jesper Sindahl Nielsen

Joint work with:
Djamal Belazzougui
Gerth Stølting Brodal
Problem definition
Problem definition

Given $n$ integers $x_1, x_2, \ldots, x_n$ produce the list $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$ such that $x_{\pi(i)} \leq x_{\pi(i+1)}$ for $1 \leq i \leq n - 1$ and $\pi$ is a permutation.
Problem definition

Given \( n \) integers \( x_1, x_2, \ldots, x_n \) produce the list \( x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)} \) such that \( x_{\pi(i)} \leq x_{\pi(i+1)} \) for \( 1 \leq i \leq n - 1 \) and \( \pi \) is a permutation.

We work in the word-RAM model with word size \( w \).

i.e. each \( x_i \in [2^w] = \{0, 1, \ldots, 2^w - 1\} \)
Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
</table>

Jesper Sindahl Nielsen
### Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
</tbody>
</table>
# Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
</tbody>
</table>
## Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
<tr>
<td>Kirkpatrick &amp; Reisch (1984)</td>
<td>$O(n \log \frac{w}{\log n})$</td>
<td>$O(2^w)$</td>
<td>Only asymptotically better than vEB when $w = O(\log n)$</td>
</tr>
</tbody>
</table>
## Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
<tr>
<td>Kirkpatrick &amp; Reisch (1984)</td>
<td>$O(n \log \frac{w}{\log n})$</td>
<td></td>
<td>Only asymptotically better than vEB when $w = O(\log n)$</td>
</tr>
<tr>
<td>Signature sort (1995)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>Only for $w = \Omega(\log^{2+\varepsilon} n)$</td>
</tr>
</tbody>
</table>
## Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
<tr>
<td>Kirkpatrick &amp; Reisch (1984)</td>
<td>$O(n \log \frac{w}{\log n})$</td>
<td></td>
<td>Only asymptotically better than vEB when $w = O(\log n)$</td>
</tr>
<tr>
<td>Signature sort (1995)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>Only for $w = \Omega(\log^{2+\varepsilon} n)$</td>
</tr>
<tr>
<td>Han &amp; Thorup (2002)</td>
<td>$O(n\sqrt{\log(w/\log n)})$</td>
<td>$O(n)$</td>
<td>Randomized with multiplication</td>
</tr>
</tbody>
</table>
## Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
<tr>
<td>Kirkpatrick &amp; Reisch (1984)</td>
<td>$O(n \log \frac{w}{\log n})$</td>
<td></td>
<td>Only asymptotically better than vEB when $w = O(\log n)$</td>
</tr>
<tr>
<td>Signature sort (1995)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>Only for $w = \Omega(\log^{2+\varepsilon} n)$</td>
</tr>
<tr>
<td>Han &amp; Thorup (2002)</td>
<td>$O(n \sqrt{\log(w/\log n)})$</td>
<td>$O(n)$</td>
<td>Randomized with multiplication</td>
</tr>
<tr>
<td>Thorup (2002)</td>
<td>$O(n \log \log n)$</td>
<td>$O(n)$</td>
<td>Randomized, $AC^0$</td>
</tr>
</tbody>
</table>
## Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix sort (1887?)</td>
<td>$O(n \cdot \frac{w}{\log n})$</td>
<td>$O(n)$</td>
<td>optimal for $w = O(\log n)$</td>
</tr>
<tr>
<td>vEB sort (1975)</td>
<td>$O(n \log w)$</td>
<td>$O(2^w)$</td>
<td>Only $O(n)$ space with Y-fast trie</td>
</tr>
<tr>
<td>Kirkpatrick &amp; Reisch (1984)</td>
<td>$O(n \log \frac{w}{\log n})$</td>
<td></td>
<td>Only asymptotically better than vEB when $w = O(\log n)$</td>
</tr>
<tr>
<td>Signature sort (1995)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>Only for $w = \Omega(\log^{2+\varepsilon} n)$</td>
</tr>
<tr>
<td>Han &amp; Thorup (2002)</td>
<td>$O(n\sqrt{\log(w/\log n)})$</td>
<td>$O(n)$</td>
<td>Randomized with multiplication</td>
</tr>
<tr>
<td>Thorup (2002)</td>
<td>$O(n \log \log n)$</td>
<td>$O(n)$</td>
<td>Randomized, $AC^0$</td>
</tr>
<tr>
<td>Han (2004)</td>
<td>$O(n \log \log n)$</td>
<td>$O(n)$</td>
<td>Deterministic, with multiplication</td>
</tr>
</tbody>
</table>
Previous work

Algorithm | Time | Notes
---|---|---
1. Radix sort (1887?) | $O(n \cdot \frac{w}{\log n})$ | Optimal for $w = O(\log n)$
2. Signature sort (1995) | $O(n)$ | Only for $w = \Omega(\log^{2+\varepsilon} n)$
3. Han & Thorup (2002) | $O\left(n\sqrt{\log\left(\frac{w}{\log n}\right)}\right)$ | Randomized with multiplication
Our result
Our result

Question: if $w = \omega(\log n)$ and $w = o(\log^{2+\varepsilon} n)$
Our result

Question: if $w = \omega(\log n)$ and $w = o(\log^{2+\varepsilon} n)$

Can we still sort in $O(n)$ time?
Our result

Question: if $w = \omega(\log n)$ and $w = o(\log^{2+\varepsilon} n)$

Can we still sort in $O(n)$ time?

Our result: Yes, if $w = \Omega(\log^2 n \log \log n)$
Our result

Question: if \( w = \omega(\log n) \) and \( w = o(\log^{2+\varepsilon} n) \)

Can we still sort in \( O(n) \) time?

Our result: Yes, if \( w = \Omega(\log^2 n \log \log n) \)

Another problem (packed sorting): Given \( n \) integers using \( \frac{w}{b} \) bits each packed in \( \frac{n}{b} \) words, how fast can we sort them?
Our result

Question: if $w = \omega(\log n)$ and $w = o(\log^{2+\varepsilon} n)$

Can we still sort in $O(n)$ time?

Our result: Yes, if $w = \Omega(\log^2 n \log \log n)$

Another problem (packed sorting): Given $n$ integers using $\frac{w}{b}$ bits each packed in $\frac{n}{b}$ words, how fast can we sort them?

Our result: $O\left(\frac{n}{b} (\log n + \log^2 b)\right)$
Our result

Question: if $w = \omega(\log n)$ and $w = o(\log^{2+\varepsilon} n)$

Can we still sort in $O(n)$ time?

Our result: Yes, if $w = \Omega(\log^2 n \log \log n)$

Another problem (packed sorting): Given $n$ integers using $\frac{w}{b}$ bits each packed in $\frac{n}{b}$ words, how fast can we sort them?

Our result: $O(\frac{n}{b}(\log n + \log^2 b))$

Note when $b \geq \log n \log \log n$ we get $O(n/ \log \log n)$
Structure of presentation

- Packed sorting
- Integer sorting
- Implications
- Conclusion and open problems
Packed sorting
Packed sorting

Main idea: implement sorting network in RAM
Packed sorting

Main idea: implement sorting network in RAM

Goodrich (2011): randomized Shell-sort
Packed sorting

Main idea: implement sorting network in RAM

Goodrich (2011): randomized Shell-sort

Oblivious: the next comparison is independent of outcome of previous comparison
Packed sorting

Main idea: implement sorting network in RAM

Goodrich (2011): randomized Shell-sort

Oblivious: the next comparison is independent of outcome of previous comparison

High probability:
∀c : ∃ an implementation with error ≤ 1/n^c
Packed sorting

Main idea: implement sorting network in RAM

Goodrich (2011): randomized Shell-sort

Oblivious: the next comparison is independent of outcome of previous comparison

High probability:
\[ \forall c : \exists \text{ an implementation with error } \leq \frac{1}{n^c} \]

Generates sequence of \( O(n \log n) \) comparisons
Time: \( O(n \log n) \)
Packed sorting

Input is \(n/b\) words each containing \(b\) integers:

\[
\begin{array}{cccccc}
  x_{1,1} & x_{2,1} & x_{3,1} & \cdots & x_{n/b,1} \\
  x_{1,2} & x_{2,2} & x_{3,2} & \cdots & x_{n/b,2} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  x_{1,b} & x_{2,b} & x_{3,b} & \cdots & x_{n/b,b} \\
  X_1 & X_2 & X_3 & \cdots & X_{n/b} \\
\end{array}
\]
Packed sorting

Desired output:

\[
\begin{align*}
X_1 & \quad x_{1,1} \quad x_{1,2} \quad \ldots \quad x_{1,b} \\
X_2 & \quad x_{2,1} \quad x_{2,2} \quad \ldots \quad x_{2,b} \\
X_3 & \quad x_{3,1} \quad x_{3,2} \quad \ldots \quad x_{3,b} \\
& \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
X_{n/b} & \quad x_{n/b,1} \quad x_{n/b,2} \quad \ldots \quad x_{n/b,b}
\end{align*}
\]
Packed sorting

Start by computing (phase a):

\[
\begin{array}{cccccc}
x_{1,1} & \leq & x_{2,1} & \leq & x_{3,1} & \leq \cdots \leq & x_{\frac{n}{b},1} \\
x_{1,2} & \leq & x_{2,2} & \leq & x_{3,2} & \leq \cdots \leq & x_{\frac{n}{b},2} \\
\vdots & & \vdots & & \vdots & & \vdots \\
x_{1,b} & \leq & x_{2,b} & \leq & x_{3,b} & \leq \cdots \leq & x_{\frac{n}{b},b} \\
X_1 & \leq & X_2 & \leq & X_3 & \leq \cdots \leq & X_{\frac{n}{b}} \end{array}
\]
Packed sorting

Start by computing (phase a):

Followed by transposition and bitonic merging

\[
\begin{align*}
    x_{1,1} & \leq x_{2,1} & \leq x_{3,1} & \leq \cdots & \leq x_{\frac{n}{b},1} \\
    x_{1,2} & \leq x_{2,2} & \leq x_{3,2} & \leq \cdots & \leq x_{\frac{n}{b},2} \\
    \vdots & & & & \vdots \\
    x_{1,b} & \leq x_{2,b} & \leq x_{3,b} & \leq \cdots & \leq x_{\frac{n}{b},b} \\
    X_1 & & X_2 & & X_3 & & \cdots & & X_{\frac{n}{b}}
\end{align*}
\]
Packed sorting (phase a)
Packed sorting (phase a)

Run Shell-sort with $N = n/b$
Packed sorting (phase a)

Run Shell-sort with $N = n/b$

When comparing elements $\ell$ and $k$ we compare $x_{\ell,i}$ with $x_{k,i}$ for all $1 \leq i \leq b$
Packed sorting (phase a)

Run Shell-sort with $N = n/b$

When comparing elements $\ell$ and $k$ we compare $x_{\ell,i}$ with $x_{k,i}$ for all $1 \leq i \leq b$

\[
\begin{array}{ccc}
  1 x_{\ell,1} & 1 x_{\ell,2} & \cdots & 1 x_{\ell,b} \\
  0 x_{k,1} & 0 x_{k,2} & \cdots & 0 x_{k,b} \\
\end{array}
\]

\[
\begin{array}{c}
  r_1 \cdots \\
  r_2 \cdots \\
  r_b \cdots \\
\end{array}
\]
Packed sorting (phase a)

Run Shell-sort with \( N = n/b \)

When comparing elements \( \ell \) and \( k \) we compare \( x_{\ell,i} \) with \( x_{k,i} \) for all \( 1 \leq i \leq b \)

\[
\begin{array}{cccc}
1 & x_{\ell,1} & 1 & x_{\ell,2} & \cdots & 1 & x_{\ell,b} \\
0 & x_{k,1} & 0 & x_{k,2} & \cdots & 0 & x_{k,b} \\
\hline
r_1 & \cdots & r_2 & \cdots & \cdots & r_b & \cdots
\end{array}
\]

Lemma: \( r_i = 1 \) if and only if \( x_{\ell,i} \geq x_{k,i} \)
Packed sorting (phase a)

Run Shell-sort with $N = n/b$

When comparing elements $\ell$ and $k$ we compare $x_{\ell,i}$ with $x_{k,i}$ for all $1 \leq i \leq b$

\[
\begin{bmatrix}
1 x_{\ell,1} & 1 x_{\ell,2} & \cdots & 1 x_{\ell,b} \\
0 x_{k,1} & 0 x_{k,2} & \cdots & 0 x_{k,b}
\end{bmatrix}
\]

Lemma: $r_i = 1$ if and only if $x_{\ell,i} \geq x_{k,i}$

Based on $r_1, \ldots, r_b$ we can also swap in $O(1)$ time

Jesper Sindahl Nielsen
Packed sorting (phase b)

Lemma: (Thorup)
Transpose $b$ words with
$b$ elements in $O(b \log b)$
time
Packed sorting (phase b)

Lemma: (Thorup)

Transpose \( b \) words with \( b \) elements in \( O(b \log b) \) time

We currently have:

\[
\begin{align*}
&x_{1,1} \leq x_{2,1} \leq x_{3,1} \leq \ldots \leq x_{\frac{n}{b},1} \\
&x_{1,2} \leq x_{2,2} \leq x_{3,2} \leq \ldots \leq x_{\frac{n}{b},2} \\
&
\vdots & \vdots & \vdots & \vdots \\
&x_{1,b} \leq x_{2,b} \leq x_{3,b} \leq \ldots \leq x_{\frac{n}{b},b} \\
&X_1 & X_2 & X_3 & \ldots & X_{\frac{n}{b}}
\end{align*}
\]
### Packed sorting (phase b)

**Lemma: (Thorup)** Transpose $b$ words with $b$ elements in $O(b \log b)$ time

<table>
<thead>
<tr>
<th>$x_{1,1}$</th>
<th>$x_{2,1}$</th>
<th>$x_{3,1}$</th>
<th>...</th>
<th>$x_{n/b,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{1,2}$</td>
<td>$x_{2,2}$</td>
<td>$x_{3,2}$</td>
<td>...</td>
<td>$x_{n/b,2}$</td>
</tr>
<tr>
<td>$x_{1,b}$</td>
<td>$x_{2,b}$</td>
<td>$x_{3,b}$</td>
<td>...</td>
<td>$x_{n/b,b}$</td>
</tr>
</tbody>
</table>

**Transposition:**

- $x_{1,1}$, $x_{2,1}$, $x_{3,1}$, ... $x_{n/b,1}$
- $x_{1,2}$, $x_{2,2}$, $x_{3,2}$, ... $x_{n/b,2}$
- $x_{1,b}$, $x_{2,b}$, $x_{3,b}$, ... $x_{n/b,b}$

<table>
<thead>
<tr>
<th>$x_{1,1}$</th>
<th>$x_{2,1}$</th>
<th>$x_{3,1}$</th>
<th>...</th>
<th>$x_{n/b,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{1,2}$</td>
<td>$x_{2,2}$</td>
<td>$x_{3,2}$</td>
<td>...</td>
<td>$x_{n/b,2}$</td>
</tr>
<tr>
<td>$x_{1,b}$</td>
<td>$x_{2,b}$</td>
<td>$x_{3,b}$</td>
<td>...</td>
<td>$x_{n/b,b}$</td>
</tr>
</tbody>
</table>

$X_1$ $X_2$ $X_3$ $X_{n/b}$
Lemma: (Thorup)

Transpose \( b \) words with \( b \) elements in \( O(b \log b) \) time

Transposition:

We have \( n/b \) internally sorted words.

Note \( x_{1,b} \leq x_{b+1,1} \leq x_{b+1,b} \leq x_{2b+1,1} \cdots \)

We actually have \( b \) sorted lists! (of \( n/b^2 \) words each)
Packed sorting (phase b)

Lemma: (Thorup)

Transpose $b$ words with $b$ elements in $O(b \log b)$ time

Transposition:

We have $n/b$ internally sorted words.

Use bitonic merging on $b$ lists to get desired output

Check output is sorted, otherwise redo packed sorting
Packed sorting (analysis)
Packed sorting (analysis)

Running Shell-sort: $O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right)$
Packed sorting (analysis)

Running Shell-sort: \( O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right) \)

We have to bound the error:
Error: \( \frac{1}{N^c} \), union bound: \( \frac{b}{N^c} = \frac{b}{(n/b)^c} = \frac{b^{c+1}}{n^c} < O\left(\frac{1}{n^{c-1}}\right) \)
Packed sorting (analysis)

Running Shell-sort: \( O(N \log N) = O(\frac{n}{b} \log \frac{n}{b}) \)

We have to bound the error:

Error: \( \frac{1}{N^c} \), union bound: \( \frac{b}{N^c} = \frac{b}{(n/b)^c} = \frac{b^{c+1}}{n^c} < O(\frac{1}{n^{c-1}}) \)

Thorup’s transposition: \( O(b \log b) \cdot \frac{n}{b^2} = O(\frac{n}{b} \log b) \)
Packed sorting (analysis)

Running Shell-sort: \( O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right) \)

We have to bound the error:
Error: \( \frac{1}{N^c} \), union bound: \( \frac{b}{N^c} = \frac{b}{(n/b)^c} = \frac{b^{c+1}}{n^c} < O\left(\frac{1}{n^{c-1}}\right) \)

Thorup’s transposition: \( O(b \log b) \cdot \frac{n}{b^2} = O\left(\frac{n}{b} \log b\right) \)

Collecting lists in sorted order: \( O\left(\frac{n}{b}\right) \)
Packed sorting (analysis)

Running Shell-sort: $O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right)$

We have to bound the error:

Error: $\frac{1}{N^c}$, union bound: $\frac{b}{N^c} = \frac{b}{(n/b)^c} = \frac{b^{c+1}}{n^c} < O\left(\frac{1}{n^{c-1}}\right)$

Thorup’s transposition: $O(b \log b) \cdot \frac{n}{b^2} = O\left(\frac{n}{b} \log b\right)$

Collecting lists in sorted order: $O\left(\frac{n}{b}\right)$

Bitonic merging: $b$ lists gives $O(\log b)$ rounds
Packed sorting (analysis)

Running Shell-sort: \( O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right) \)

We have to bound the error:

Error: \( \frac{1}{N^c} \), union bound: \( \frac{b}{N^c} = \frac{b}{(n/b)^c} = \frac{b^{c+1}}{n^c} < O\left(\frac{1}{n^{c-1}}\right) \)

Thorup’s transposition: \( O(b \log b) \cdot \frac{n}{b^2} = O\left(\frac{n}{b} \log b\right) \)

Collecting lists in sorted order: \( O\left(\frac{n}{b}\right) \)

Bitonic merging: \( b \) lists gives \( O(\log b) \) rounds

Each round: \( O\left(\frac{n}{b} \log b\right) \)
Packed sorting (analysis)

Running Shell-sort: \( O(N \log N) = O\left(\frac{n}{b} \log \frac{n}{b}\right) \)

We have to bound the error:

Error: \( \frac{1}{N^{c}} \), union bound: \( \frac{b}{N^{c}} = \frac{b}{(n/b)^{c}} = \frac{b^{c+1}}{n^{c}} < O\left(\frac{1}{n^{c-1}}\right) \)

Thorup’s transposition: \( O(b \log b) \cdot \frac{n}{b^2} = O\left(\frac{n}{b} \log b\right) \)

Collecting lists in sorted order: \( O\left(\frac{n}{b}\right) \)

Bitonic merging: \( b \) lists gives \( O(\log b) \) rounds

Each round: \( O\left(\frac{n}{b} \log b\right) \)

Total running time \( O\left(\frac{n}{b} (\log n + \log^2 b)\right) \)
Structure of presentation

- Packed sorting
- Integer sorting
- Implications
- Conclusion and open problems
Integer sorting
Integer sorting

Given $n$ integers $x_1, x_2, \ldots, x_n$ produce the list $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$ such that $x_{\pi(1)} \leq x_{\pi(i+1)}$ for $1 \leq i \leq n - 1$ and $\pi$ is a permutation
Integer sorting

Given \( n \) integers \( x_1, x_2, \ldots, x_n \) produce the list \( x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)} \) such that \( x_{\pi(i)} \leq x_{\pi(i+1)} \) for \( 1 \leq i \leq n - 1 \) and \( \pi \) is a permutation.

We work in the word-RAM model with word size \( w \).

I.e. each \( x_i \in [2^w] = \{0, 1, \ldots, 2^w - 1\} \)
Integer sorting

Given $n$ integers $x_1, x_2, \ldots, x_n$ produce the list $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$ such that $x_{\pi(1)} \leq x_{\pi(i+1)}$ for $1 \leq i \leq n - 1$ and $\pi$ is a permutation.

We work in the word-RAM model with word size $w$

I.e. each $x_i \in [2^w] = \{0, 1, \ldots, 2^w - 1\}$

We consider the case $w = \Omega(\log^2 n \log \log n)$
Integer sorting - observation
Integer sorting - observation

We have $n$ integers using $r$ bits each
We have $n$ integers using $r$ bits each
Consider two integers $x = x_1 x_2 \cdots x_r$ and $y = y_1 y_2 \cdots y_r$
Integer sorting - observation

We have \( n \) integers using \( r \) bits each.

Consider two integers \( x = x_1 x_2 \cdots x_r \) and \( y = y_1 y_2 \cdots y_r \).

If \( x_1 x_2 \cdots x_{r/2} = y_1 y_2 \cdots y_{r/2} \) then rank of \( x \) and \( y \) among the other integers are given by the rank of most significant half (MSH), and their individual rank is given by least significant half (LSH).
We have $n$ integers using $r$ bits each.

Consider two integers $x = x_1x_2\cdots x_r$ and $y = y_1y_2\cdots y_r$

if $x_1x_2\cdots x_{r/2} = y_1y_2\cdots y_{r/2}$ then rank of $x$ and $y$ among the other integers are given by the rank of most significant half (MSH), and their individual rank is given by least significant half (LSH)

If no integer shares MSH with $x$, then LSH of $x$ is irrelevant wrt the rank of $x$
Integer sorting - observation

We have $n$ integers using $r$ bits each

Consider two integers $x = x_1x_2\cdots x_r$ and $y = y_1y_2\cdots y_r$

if $x_1x_2\cdots x_{r/2} = y_1y_2\cdots y_{r/2}$ then rank of $x$ and $y$ among the other integers are given by the rank of most significant half (MSH), and their individual rank is given by least significant half (LSH)

If no integer shares MSH with $x$, then LSH of $x$ is irrelevant wrt the rank of $x$

Idea: throw away constant fraction of the $nr$ bits and recursively sort $r/2$ bit integers
Integer sorting - observation
Integer sorting - observation

Suppose we have rank of MSH and LSH for all elements.
Integer sorting - observation

Suppose we have rank of MSH and LSH for all elements.

Each rank uses at most $\log n$ bits (there are $n$ elements..)
Integer sorting - observation

Suppose we have rank of MSH and LSH for all elements.

Each rank uses at most $\log n$ bits (there are $n$ elements..)

Concatenating MSH and LSH gives $2\log n$ bits.
Integer sorting - observation

Suppose we have rank of MSH and LSH for all elements.

Each rank uses at most $\log n$ bits (there are $n$ elements..)

Concatenating MSH and LSH gives $2 \log n$ bits.

Sorting by their concatenation gives the correct order of original elements.
Integer sorting - observation

Suppose we have rank of MSH and LSH for all elements.

Each rank uses at most $\log n$ bits (there are $n$ elements..)

Concatenating MSH and LSH gives $2\log n$ bits.

Sorting by their concatenation gives the correct order of original elements

We can fit at least $\frac{w}{\log n} \geq \log n \log \log n$ ranks pr word, i.e. we can do packed sorting in $O(\frac{n}{\log \log n})$ time
Integer sorting - algorithm
Integer sorting - algorithm

We now develop a recursive Monte-Carlo algorithm:
We now develop a recursive Monte-Carlo algorithm:

Input: \((id_1, x_1), (id_2, x_2), \ldots, (id_m, x_m)\)
Integer sorting - algorithm

We now develop a recursive Monte-Carlo algorithm:

Input: \((id_1, x_1), (id_2, x_2), \ldots, (id_m, x_m)\)

Output: rank\((x_1)\) within \(id_1\), rank\((x_2)\) within \(id_2\), \ldots, rank\((x_m)\) within \(id_m\)
We now develop a recursive Monte-Carlo algorithm:

**Input:** \((id_1, x_1), (id_2, x_2), \ldots, (id_m, x_m)\)

**Output:** \(\text{rank}(x_1)\) within \(id_1\), \(\text{rank}(x_2)\) within \(id_2\), \ldots, \(\text{rank}(x_m)\) within \(id_m\)

We always have \(|id_j| \leq O(\log n)|
Integer sorting - algorithm

We now develop a recursive Monte-Carlo algorithm:

Input: \((id_1, x_1), (id_2, x_2), \ldots, (id_m, x_m)\)

Output: \(\text{rank}(x_1)\) within \(id_1\), \(\text{rank}(x_2)\) within \(id_2\), \ldots, \(\text{rank}(x_m)\) within \(id_m\)

We always have \(|id_j| \leq O(\log n)\)

This solves original problem: plug in 0 as \(id\) for all input elements and permute.
Integer sorting - algorithm
Integer sorting - algorithm

Intuitively we build a compressed trie of $x_1, \ldots, x_n$ by gradually refining the alphabet.
Integer sorting - algorithm

Intuitively we build a compressed trie of \( x_1, \ldots x_n \) by gradually refining the alphabet.

Definition: Compressed trie of \( x_1, \ldots x_n \) of detail \( i \), denoted \( T^i \) is the compressed trie of \( x_1, \ldots, x_n \) with alphabet \( \Sigma^i = \{0, 1\}^{w/2^i} \).
Integer sorting - algorithm

Intuitively we build a compressed trie of $x_1, \ldots x_n$ by gradually refining the alphabet.

Definition: Compressed trie of $x_1, \ldots x_n$ of detail $i$, denoted $T^i$ is the compressed trie of $x_1, \ldots, x_n$ with alphabet $\Sigma^i = \{0, 1\}^{w/2^i}$

Algorithm: build $T^1, T^2, \ldots, T^{\log \log n}$ incrementally
Integer sorting - algorithm

Intuitively we build a compressed trie of $x_1, \ldots x_n$ by gradually refining the alphabet.

Definition: Compressed trie of $x_1, \ldots x_n$ of detail $i$, denoted $T^i$ is the compressed trie of $x_1, \ldots, x_n$ with alphabet $\Sigma^i = \{0, 1\}^{w/2^i}$.

Algorithm: build $T^1, T^2, \ldots, T^{\log \log n}$ incrementally.

We build $T^{i+1}$ based on $T^i$. 
Integer sorting - algorithm
Integer sorting - algorithm

Maintain the following invariants for the \(i\)th recursion:
Integer sorting - algorithm

Maintain the following invariants for the $i$th recursion:

1) Number of bits in an element $|x_j| = w/2^i$
Integer sorting - algorithm

Maintain the following invariants for the $i$th recursion:

1) Number of bits in an element $|x_j| = w/2^i$

2) Bijection from $i$ds to non-leaf nodes in $T^i$
Integer sorting - algorithm

Maintain the following invariants for the $i$th recursion:

1) Number of bits in an element $|x_j| = w/2^i$

2) Bijection from $ids$ to non-leaf nodes in $T^i$

3) The pair $(id, e)$ is in the input iff $v \in T^i$ corresponding to $id$ has a downgoing edge labeled by a string where $e \in \Sigma^i$ is the first character
Integer sorting - algorithm
Integer sorting - algorithm

Base case of algorithm:
Integer sorting - algorithm

Base case of algorithm:

if $|x_j| = \frac{w}{\log n}$, use packed sorting, compute ranks and return
Integer sorting - algorithm

Base case of algorithm:

if $|x_j| = \frac{w}{\log n}$, use packed sorting, compute ranks and return

Remember input is packed in words: we have to use bit tricks
Integer sorting - algorithm

Base case of algorithm:

if \(|x_j| = \frac{w}{\log n}\), use packed sorting, compute ranks and return

Remember input is packed in words: we have to use bit tricks

If you really want to know how, read the paper
Integer sorting - algorithm
Integer sorting - algorithm

At level $i$ we do 2 things: prepare level $i + 1$ and use result to solve current level
Integer sorting - algorithm

At level $i$ we do 2 things: prepare level $i + 1$ and use result to solve current level

Getting $T^{i+1}$ from $T^i$: 
Integer sorting - algorithm

At level $i$ we do 2 things: prepare level $i + 1$ and use result to solve current level

Getting $T^{i+1}$ from $T^i$:

Detail $i$
Integer sorting - algorithm

At level $i$ we do 2 things: prepare level $i + 1$ and use result to solve current level

Getting $T^{i+1}$ from $T^{i}$:

Detail $i$ 

Detail $i + 1$
Integer sorting - algorithm
Integer sorting - algorithm

Essentially we need to be able to find duplicates of MSH within a node
Integer sorting - algorithm

Essentially we need to be able to find duplicates of MSH within a node

How?
Integer sorting - algorithm

Essentially we need to be able to find duplicates of MSH within a node

How? hash MSH to $O(\log n)$ bits, use packed sorting then scan
Integer sorting - algorithm

Essentially we need to be able to find duplicates of MSH within a node

How? hash MSH to $O(\log n)$ bits, use packed sorting then scan

This requires even more bit tricks.
If you really want to know how, read the paper
Integer sorting - algorithm
Integer sorting - algorithm

Returning from level $i$: 
Integer sorting - algorithm

Returning from level $i$:

We get ranks of (almost) all MSH and LSH from recursion
Integer sorting - algorithm

Returning from level $i$:

- We get ranks of (almost) all MSH and LSH from recursion
- We do not get rank for an LSH if no elements shared MSH
Integer sorting - algorithm

Returning from level $i$:

We get ranks of (almost) all MSH and LSH from recursion

We do not get rank for an LSH if no elements shared MSH
put in 0 as rank in that case
Integer sorting - algorithm

Returning from level $i$:

- We get ranks of (almost) all MSH and LSH from recursion.
- We do not get rank for an LSH if no elements shared MSH.
- Put in 0 as rank in that case.
- If many elements shared MSH, we only get rank once.
Integer sorting - algorithm

Returning from level $i$:

We get ranks of (almost) all MSH and LSH from recursion

We do not get rank for an LSH if no elements shared MSH
put in 0 as rank in that case

If many elements shared MSH, we only get rank once

Propagate that rank to those without
Returning from level $i$:

We get ranks of (almost) all MSH and LSH from recursion.

We do not get rank for an LSH if no elements shared MSH.

Put in 0 as rank in that case.

If many elements shared MSH, we only get rank once.

Propagate that rank to those without.

Now all elements have rank of MSH and LSH.

Use packed sorting to sort by the concatenation.

Extract ranks based on this and return.
Integer sorting - analysis
Integer sorting - analysis

We have $O(\log \log n)$ recursions
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i} / w)$
Integer sorting - analysis

We have $O(\log \log n)$ recursions.

In each recursion we scan input: $O(n \cdot \frac{w}{2^i}/w)$

Gives total $O(n)$ since it is geometrically decreasing.
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i} / w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i}/w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.

Packed sorting: $O(n/\log \log n)$ - In total $O(n)$
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i} / w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.

Packed sorting: $O(n / \log \log n)$ - In total $O(n)$

What is the error probability?
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i}/w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.

Packed sorting: $O(n/\log \log n)$ - In total $O(n)$

What is the error probability?

Pr [ Packed sorting fails at some level ] $\leq O(\log \log n/n^c)$
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i} / w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.

Packed sorting: $O(n / \log \log n)$ - In total $O(n)$

What is the error probability?

Pr [ Packed sorting fails at some level ] $\leq O(\log \log n / n^c)$

Pr [ hashing fails at some level ] $\leq O(\log \log n / n^{c'})$
Integer sorting - analysis

We have $O(\log \log n)$ recursions

In each recursion we scan input: $O(n \cdot \frac{w}{2^i}/w)$

Gives total $O(n)$ since it is geometrically decreasing

At each level we sort $O(n)$ integers with $O(\log n)$ bits.

Packed sorting: $O(n/\log \log n)$ - In total $O(n)$

What is the error probability?

Pr [ Packed sorting fails at some level ] $\leq O(\log \log n/n^c)$

Pr [ hashing fails at some level ] $\leq O(\log \log n/n^{c'})$

I.e. we sort with high probability
Structure of presentation

- Packed sorting
- Integer sorting
- Implications
- Conclusion and open problems
Implications
Implications

Lemma (Thorup): \( n \) integers can be split into \( O(\sqrt{n}) \) sets \( X_1, X_2, \ldots, X_k \) with \( O(\sqrt{n}) \) elements in each, such that all elements in \( X_i \) are less than all elements in \( X_{i+1} \) in \( O(n) \) time.
Implications

Lemma (Thorup): $n$ integers can be split into $O(\sqrt{n})$ sets $X_1, X_2, \ldots, X_k$ with $O(\sqrt{n})$ elements in each, such that all elements in $X_i$ are less than all elements in $X_{i+1}$ in $O(n)$ time.

Recursively apply Thorup’s lemma to get size $n_0, n_i = \sqrt{n_{i-1}}$ until $\log^2 n_j \log \log n_j \leq w \ldots \log n_j \approx \sqrt{w/\log w}$
Implications

Lemma (Thorup): $n$ integers can be split into $O(\sqrt{n})$ sets $X_1, X_2, \ldots, X_k$ with $O(\sqrt{n})$ elements in each, such that all elements in $X_i$ are less than all elements in $X_{i+1}$ in $O(n)$ time.

Recursively apply Thorup’s lemma to get size $n_0$, $n_i = \sqrt{n_{i-1}}$

until $\log^2 n_j \log \log n_j \leq w$ ... $\log n_j \approx \sqrt{w/\log w}$

Happens after $O(\log \frac{\log n}{\sqrt{w/\log w}})$ steps. Sort all sets in $O(n)$ time.
Implications

Lemma (Thorup): $n$ integers can be split into $O(\sqrt{n})$ sets $X_1, X_2, \ldots, X_k$ with $O(\sqrt{n})$ elements in each, such that all elements in $X_i$ are less than all elements in $X_{i+1}$ in $O(n)$ time.

Recursively apply Thorup’s lemma to get size $n_0$, $n_i = \sqrt{n_{i-1}}$ until $\log^2 n_j \log \log n_j \leq w$ ... $\log n_j \approx \sqrt{w/\log w}$

Happens after $O(\log \frac{\log n}{\sqrt{w/\log w}})$ steps. Sort all sets in $O(n)$ time.

Total time: $O(n \log \frac{\log n}{\sqrt{w/\log w}})$ for general $w$
Implications

Lemma (Thorup): $n$ integers can be split into $O(\sqrt{n})$ sets $X_1, X_2, \ldots, X_k$ with $O(\sqrt{n})$ elements in each, such that all elements in $X_i$ are less than all elements in $X_{i+1}$ in $O(n)$ time.

Recursively apply Thorup’s lemma to get size $n_0$, $n_i = \sqrt{n_{i-1}}$ until $\log^2 n_j \log \log n_j \leq w$ ... $\log n_j \approx \sqrt{w/\log w}$

Happens after $O(\log \frac{\log n}{\sqrt{w/\log w}})$ steps. Sort all sets in $O(n)$ time.

Total time: $O(n \log \frac{\log n}{\sqrt{w/\log w}})$ for general $w$

We can actually sort $n$ integers in $o(n\sqrt{\log \log n})$ time for some $w = o(\log^2 n)$, eg $w = O\left(\frac{\log^2 n}{(\log \log n)^c}\right)$ gives $O(n \log \log \log n)$
Open problems
Open problems

Can this be done deterministically?
Open problems

Can this be done deterministically?

Can we do this using only $AC^0$ operations?
Open problems

Can this be done deterministically?

Can we do this using only $AC^0$ operations?

Integer and string sorted are closely related in the RAM model.
Open problems

Can this be done deterministically?
Can we do this using only $AC^0$ operations?
Integer and string sorted are closely related in the RAM model
String sorting in I/O model in $O(\text{scan}(N) + \text{sort}(n))$?
Thank You