Worst-Case Efficient Sorting with QuickMergesort

Stefan Edelkamp[1] and Armin Weiß[2]

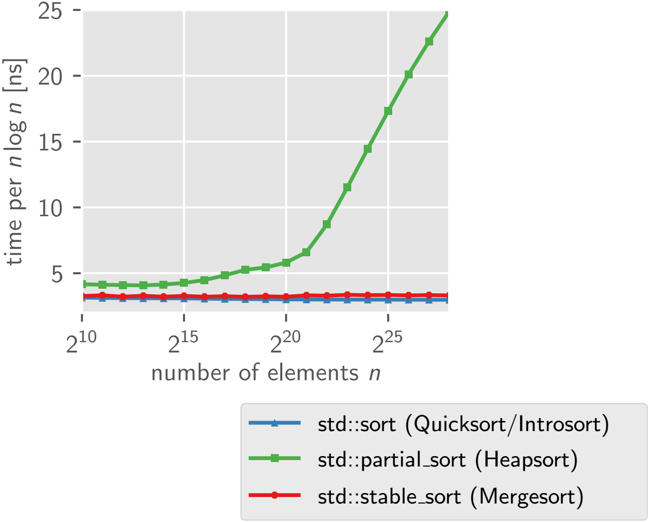
[1]King's College London, UK

[2]FMI, Universität Stuttgart, Germany

San Diego, January 7, 2019

Comparison-based sorting: Quicksort, Heapsort, Mergesort

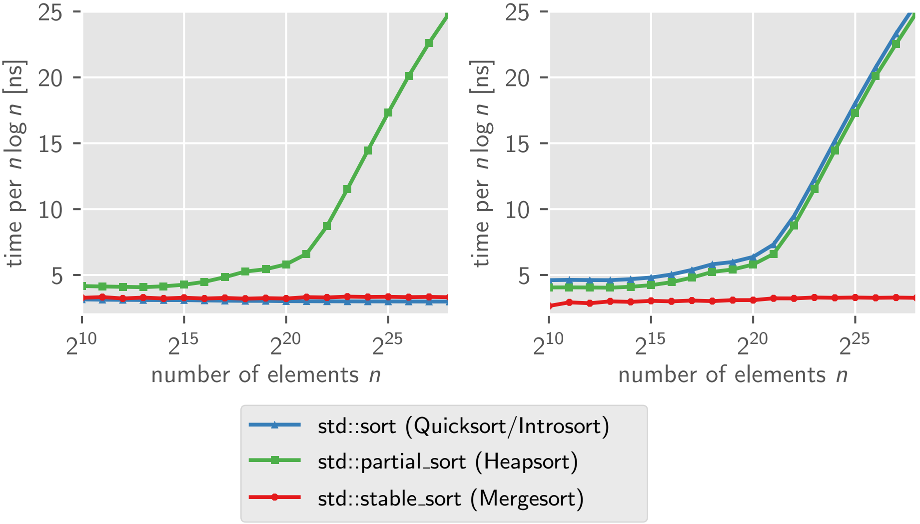
Comparison-based sorting: Quicksort, Heapsort, Mergesort



Running times (divided by $n \log n$) for sorting integers.

Left: random inputs.

Comparison-based sorting: Quicksort, Heapsort, Mergesort



Running times (divided by $n \log n$) for sorting integers.

Left: random inputs.

Right: random with large elements in the middle and end.

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

- Make Quicksort worst-case efficient: Introsort, median-of-medians pivot selection

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	“in place”	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

- Make Quicksort worst-case efficient: Introsort, median-of-medians pivot selection
- Make Heapsort fast: Bottom-up Heapsort (not very fast)

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

- Make Quicksort worst-case efficient: Introsort, median-of-medians pivot selection
- Make Heapsort fast: Bottom-up Heapsort (not very fast)
- Make Mergesort in-place:
  - block based merging (stable implementations: Grailsort, Wikisort)

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

- Make Quicksort worst-case efficient: Introsort, median-of-medians pivot selection
- Make Heapsort fast: Bottom-up Heapsort (not very fast)
- Make Mergesort in-place:
  - block based merging (stable implementations: Grailsort, Wikisort)
  - rotation based merging (stable, but $\mathcal{O}(n \log^2 n)$)

Quicksort, Heapsort, Mergesort

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓

Wish to have three times ✓:

- Make Quicksort worst-case efficient: Introsort, median-of-medians pivot selection
- Make Heapsort fast: Bottom-up Heapsort (not very fast)
- Make Mergesort in-place:
  - block based merging (stable implementations: Grailsort, Wikisort)
  - rotation based merging (stable, but $\mathcal{O}(n \log^2 n)$)
  - use one half as buffer to sort the other half
    (In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012], unstable)

Outline

Outline:

- QuickMergesort
- Our improvements and theoretical bounds
- Experiments

Quicksort

procedure $\textsc{Quicksort}(A[\ell, \ldots, r])$
  if $r > \ell$ **then**
    pivot $\leftarrow$ choosePivot($A[\ell, \ldots, r]$)
    cut $\leftarrow$ partition($A[\ell, \ldots, r]$, pivot)
    Quicksort($A[\ell, \ldots, \mathrm{cut} - 1]$)
    Quicksort($A[\mathrm{cut}, \ldots, r]$)
  end if
end procedure

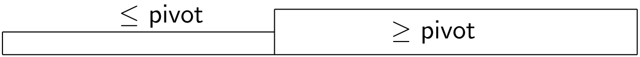
Quicksort

procedure $\textsc{Quicksort}(A[\ell, \ldots, r])$
    if $r > \ell$ **then**
        pivot $\leftarrow$ choosePivot($A[\ell, \ldots, r]$)
        cut $\leftarrow$ partition($A[\ell, \ldots, r]$, pivot)
        Quicksort($A[\ell, \ldots, \mathrm{cut} - 1]$)
        Quicksort($A[\mathrm{cut}, \ldots, r]$)
    end if
end procedure

Quicksort

1: **procedure** $\text{QUICKSORT}(A[\ell, \ldots, r])$
2: **if** $r > \ell$ **then**
3: pivot $\leftarrow$ choosePivot$(A[\ell, \ldots, r])$
4: cut $\leftarrow$ partition$(A[\ell, \ldots, r]$, pivot$)$
5: Quicksort$(A[\ell, \ldots, \mathrm{cut} - 1])$
6: Quicksort$(A[\mathrm{cut}, \ldots, r])$
7: **end if**
8: **end procedure**

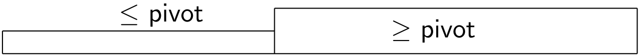
- After line 4:



Quicksort

1: **procedure** $\textsc{Quicksort}(A[\ell, \ldots, r])$
2:     **if** $r > \ell$ **then**
3:         pivot $\leftarrow$ choosePivot($A[\ell, \ldots, r]$)
4:         cut $\leftarrow$ partition($A[\ell, \ldots, r]$, pivot)
5:         Quicksort($A[\ell, \ldots, \mathrm{cut} - 1]$)
6:         Quicksort($A[\mathrm{cut}, \ldots, r]$)
7:     **end if**
8: **end procedure**

- After line 4:

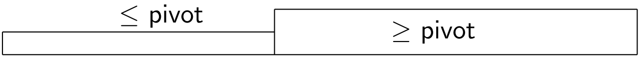


- After line 5:

Quicksort

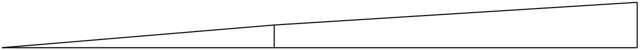
1: **procedure** $\textsc{Quicksort}(A[\ell, \ldots, r])$
2:     **if** $r > \ell$ **then**
3:         pivot $\leftarrow$ choosePivot($A[\ell, \ldots, r]$)
4:         cut $\leftarrow$ partition($A[\ell, \ldots, r]$, pivot)
5:         Quicksort($A[\ell, \ldots, \mathrm{cut} - 1]$)
6:         Quicksort($A[\mathrm{cut}, \ldots, r]$)
7:     **end if**
8: **end procedure**

- After line 4:



- After line 5:

- After line 6: both parts sorted recursively with Quicksort



QuickMergesort

```
procedure QUICKSORT(A[ℓ, ..., r])
    if r > ℓ then
        pivot ← choosePivot(A[ℓ, ..., r])
        cut ← partition(A[ℓ, ..., r], pivot)
        Quicksort(A[ℓ, ..., cut − 1])
        Quicksort(A[cut, ..., r])
    end if
end procedure
```

QuickMergesort

procedure QUICKSORT($A[\ell, \dots, r]$)

    if $r > \ell$ **then**

        pivot $\leftarrow$ choosePivot($A[\ell, \dots, r]$)

        cut $\leftarrow$ partition($A[\ell, \dots, r]$, pivot)

        Mergesort ~~Quicksort~~($A[\ell, \dots, \mathrm{cut} - 1]$)

        Quicksort($A[\mathrm{cut}, \dots, r]$)

    end if

end procedure

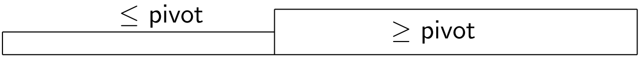
QuickMergesort

procedure QuickMergesort($A[\ell, \ldots, r]$)
    if $r > \ell$ **then**
        pivot $\leftarrow$ choosePivot($A[\ell, \ldots, r]$)
        cut $\leftarrow$ partition($A[\ell, \ldots, r]$, pivot)
        Mergesort($A[\ell, \ldots, \mathrm{cut} - 1]$)
        QuickMergesort($A[\mathrm{cut}, \ldots, r]$)
    end if
end procedure

QuickMergesort

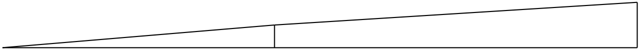
1: **procedure** QuickMergesort($A[\ell, \dots, r]$)
2:     **if** $r > \ell$ **then**
3:         pivot $\leftarrow$ choosePivot($A[\ell, \dots, r]$)
4:         cut $\leftarrow$ partition($A[\ell, \dots, r]$, pivot)
5:         Mergesort($A[\ell, \dots, \mathrm{cut} - 1]$)
6:         QuickMergesort($A[\mathrm{cut}, \dots, r]$)
7:     **end if**
8: **end procedure**

- After line 4:



- After line 5:

- After line 6: both parts sorted recursively with QuickMergesort



QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.

7	11	4	5	6	10	9	2	3	1	0	8

QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.

7	11	4	5	6	10	9	2	3	1	0	8

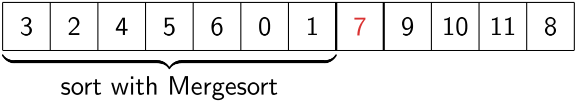
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.

3	2	4	5	6	0	1	7	9	10	11	8

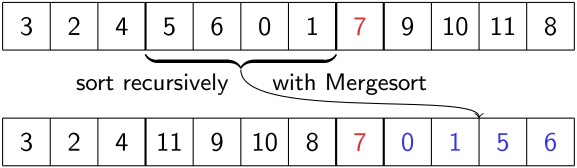
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



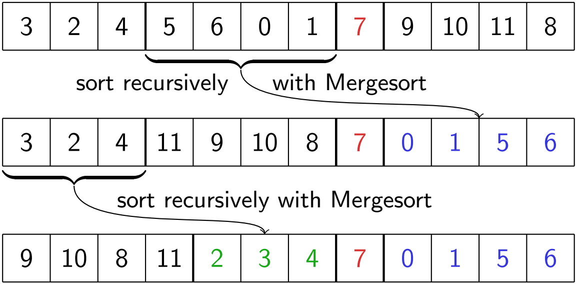
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



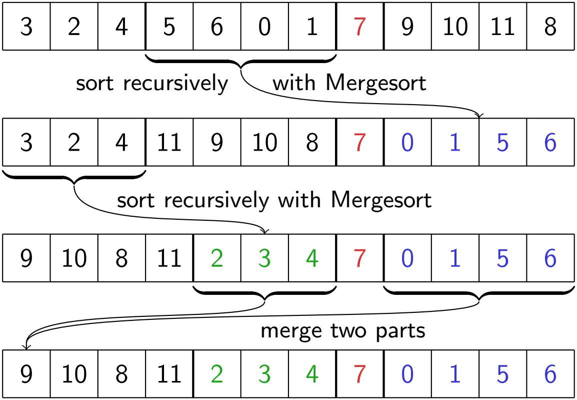
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



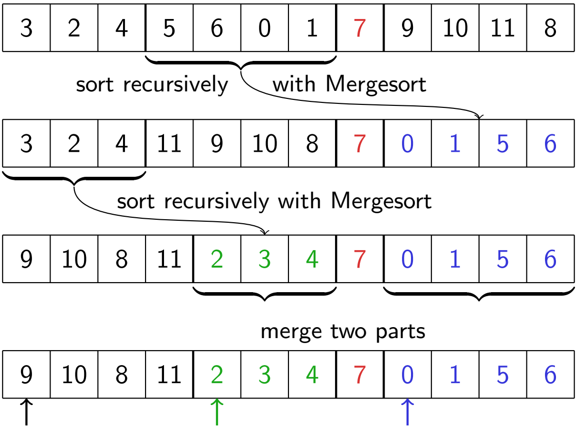
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



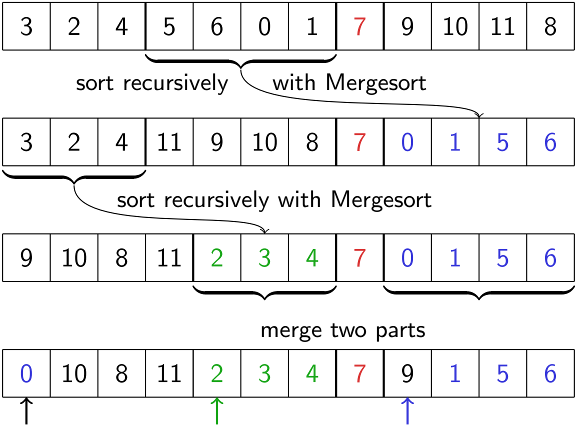
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



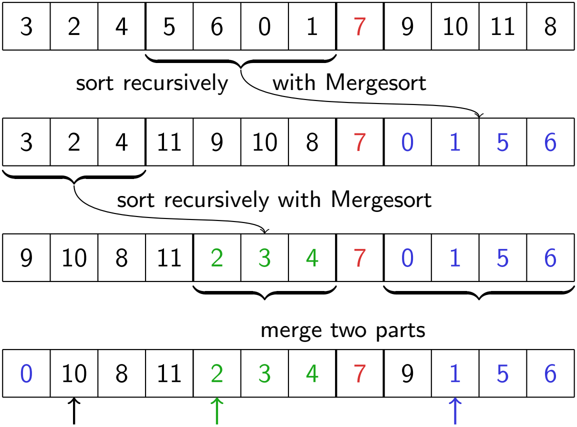
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



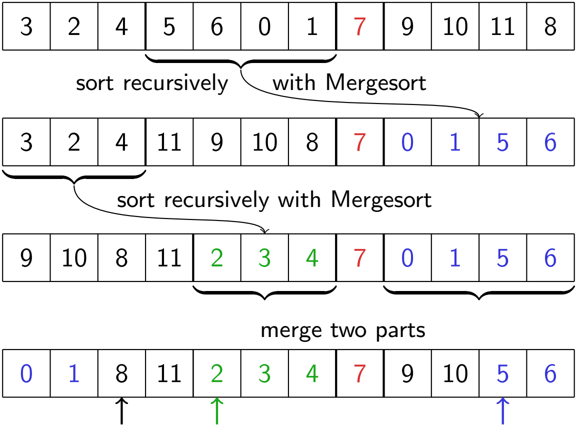
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



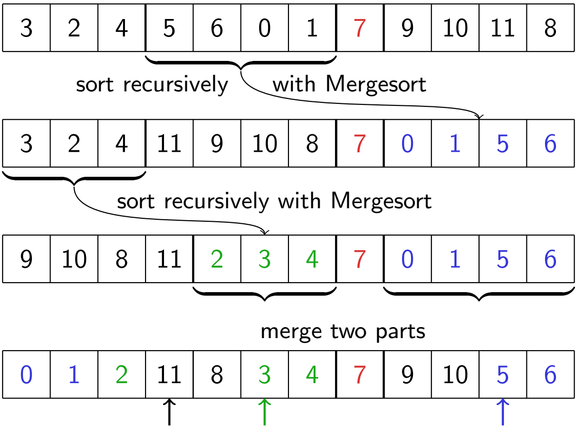
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



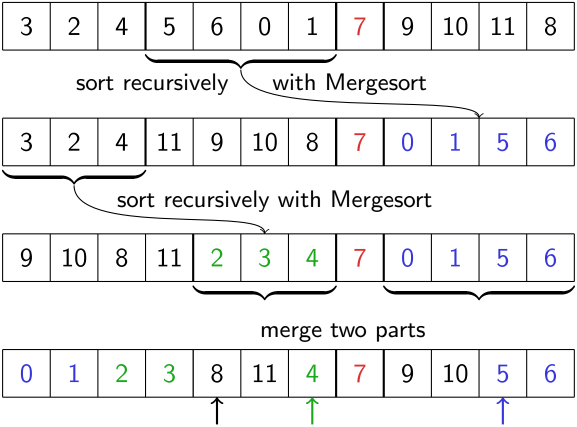
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



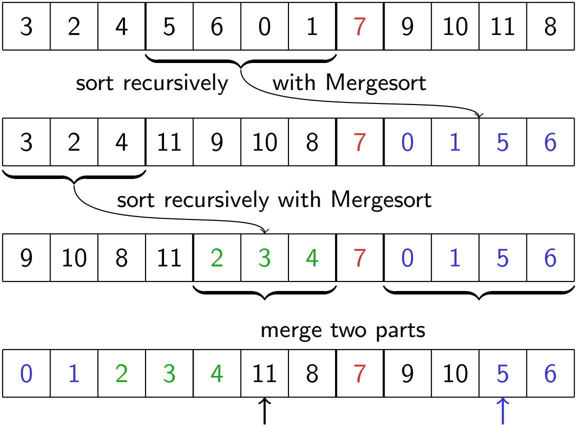
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



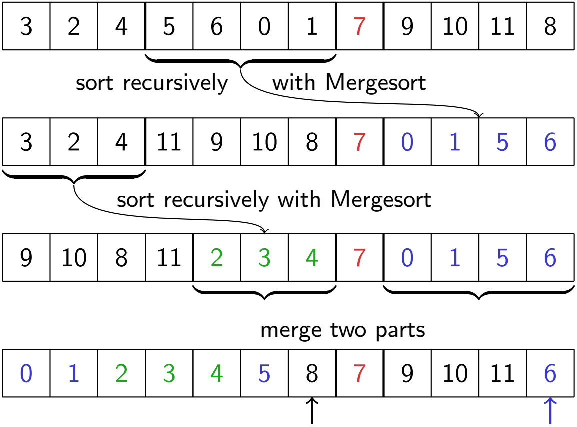
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



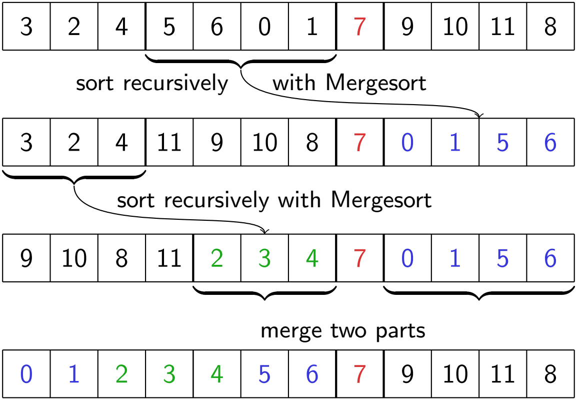
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



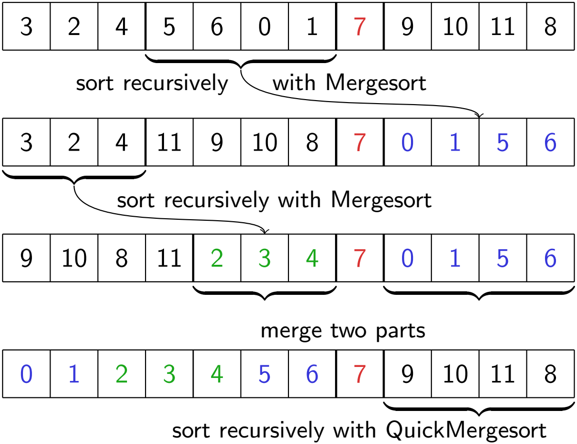
QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



QuickMergesort

1. Partition according to some pivot element.
2. Sort one part with Mergesort.
3. Sort the the other part recursively with QuickMergesort.



Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)!

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n\log n - 0.83n$ *(resp.* $n\log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)!

Solution: choose the median as pivot:

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)! ☹

Solution: choose the median as pivot:

- Quickselect resp. Introselect.

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)!

Solution: choose the median as pivot:

- Quickselect resp. Introselect.
  - $\rightsquigarrow$ In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012]

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)! ☹

Solution: choose the median as pivot:

- Quickselect resp. Introselect. Problem: worst case
  $\rightsquigarrow$ In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012]

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)! ☹

Solution: choose the median as pivot:

- Quickselect resp. Introselect. Problem: worst case
  $\rightsquigarrow$ In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012]
- Median-of-medians algorithm [Blum, Floyd, Pratt, Rivest, Tarjan 1973]

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)!

Solution: choose the median as pivot:

- Quickselect resp. Introselect. Problem: worst case
  $\rightsquigarrow$ In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012]
- Median-of-medians algorithm [Blum, Floyd, Pratt, Rivest, Tarjan 1973]

Theorem (Folklore)

The median-of-medians algorithms needs at most $20n + o(n)$ *comparisons to find the median of n elements.*

Worst case of QuickMergesort

Theorem (Edelkamp, W. 2014, Wild 2018)

QuickMergesort needs $\leq n \log n - 0.83n$ *(resp.* $n \log n - 1.24n$*)* $+o(n)$ *comparisons on average with median-of-three (resp. median of* $\sqrt{n}$*).*

Still quadratic worst-case (with median-of-three)! ☹

Solution: choose the median as pivot:

- Quickselect resp. Introselect. Problem: worst case
  $\rightsquigarrow$ In-situ Mergesort [Elmasry, Katajainen, Stenmark 2012]
- Median-of-medians algorithm [Blum, Floyd, Pratt, Rivest, Tarjan 1973]

Theorem (Folklore)

The median-of-medians algorithms needs at most $20n + o(n)$ *comparisons to find the median of n elements.*

Need to find the median of n, $\frac{n}{2}$, $\frac{n}{4}$, $\ldots$ elements $\rightsquigarrow$ $40n$ comparisons

Median-of-medians QuickMergesort

Key observation: we do not need the exact median.

Median-of-medians QuickMergesort

Key observation: we do not need the exact median.

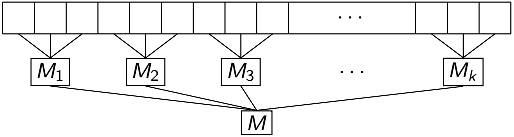
Sufficient if one third is smaller/greater than the pivot:

Median-of-medians QuickMergesort

Key observation: we do not need the exact median.

Sufficient if one third is smaller/greater than the pivot:

- form groups of 3 elements
- compute the median of each group
- compute the median of the $n/3$ medians

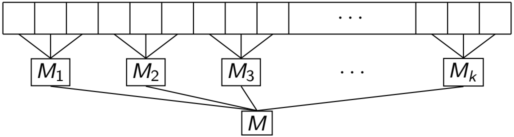


Median-of-medians QuickMergesort

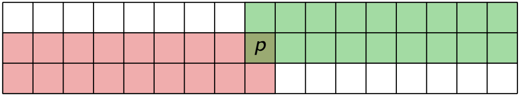
Key observation: we do not need the exact median.

Sufficient if one third is smaller/greater than the pivot:

- form groups of 3 elements
- compute the median of each group
- compute the median of the $n/3$ medians



One third are $\leq p$:

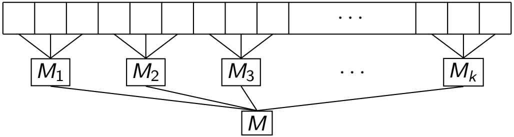


Median-of-medians QuickMergesort

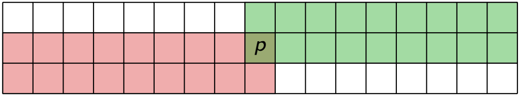
Key observation: we do not need the exact median.

Sufficient if one third is smaller/greater than the pivot:

- form groups of 3 elements
- compute the median of each group
- compute the median of the $n/3$ medians



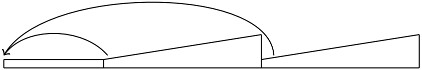
One third are $\leq p$:



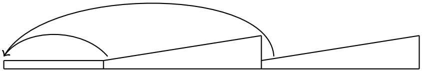
Theorem

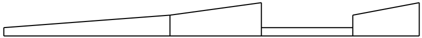
Basic MoMQuickMergesort needs at most $n \log n + 13.8n + o(n)$ *comparisons.*

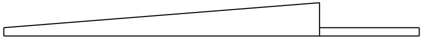
Merging with less buffer space (Reinhardt 1992)

- Step 1 (merge from the left): 

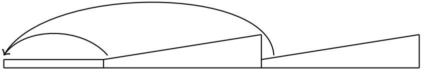
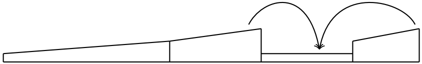
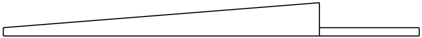
Merging with less buffer space (Reinhardt 1992)

- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.

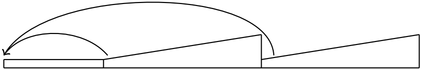
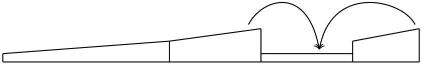
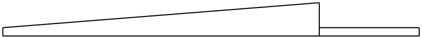


- Expected result: 

Merging with less buffer space (Reinhardt 1992)

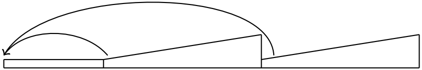
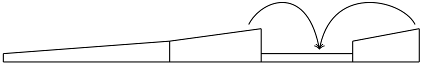
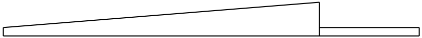
- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right): 
- Result: 

Merging with less buffer space (Reinhardt 1992)

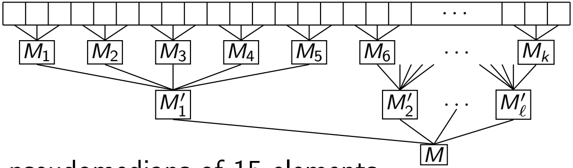
- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right): 
- Result: 

Need a guarantee that one fifth are smaller/greater than the pivot:

Merging with less buffer space (Reinhardt 1992)

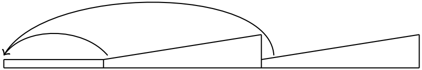
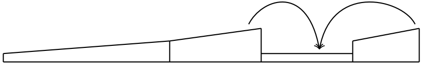
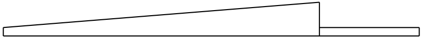
- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right): 
- Result: 

Need a guarantee that one fifth are smaller/greater than the pivot:

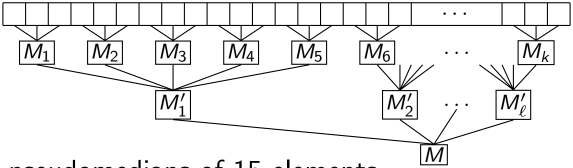


$\rightsquigarrow$ median of pseudomedians of 15 elements

Merging with less buffer space (Reinhardt 1992)

- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right): 
- Result: 

Need a guarantee that one fifth are smaller/greater than the pivot:



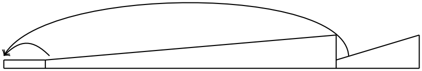
$\rightsquigarrow$ median of pseudomedians of 15 elements

Theorem

MoMQuickMergesort needs at most $n \log n + 4.57n + o(n)$ *comparisons.*

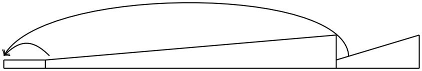
Unbalanced merging and undersampling

For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left): 

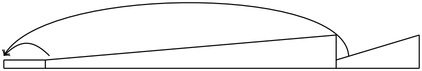
Unbalanced merging and undersampling

For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.

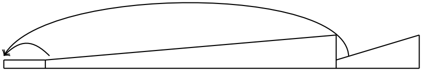
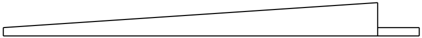
Unbalanced merging and undersampling

For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right): 

Unbalanced merging and undersampling

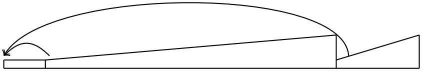
For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left): 
- Once the buffer is full, the final position for the largest element is "free".
- Step 2 (merge from the right): 
- Result: 

Unbalanced merging and undersampling

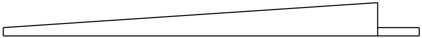
For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left):



- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right):

- Result:

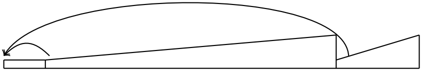


⇝ trade-off: effort to find good pivots vs. increased merging costs

Unbalanced merging and undersampling

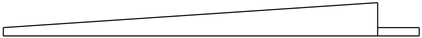
For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left):



- Once the buffer is full, the final position for the largest element is “free”.
- Step 2 (merge from the right):

- Result:



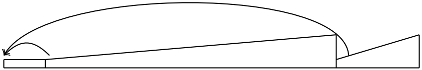
$\rightsquigarrow$ trade-off: effort to find good pivots vs. increased merging costs

Undersampling: for $\theta \geq 1$ apply the median-of-pseudomedians-of-15 strategy to n/θ elements.

Unbalanced merging and undersampling

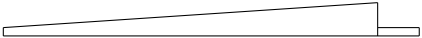
For merging sequences of different size a smaller buffer suffices:

- Step 1 (merge from the left):



- Once the buffer is full, the final position for the largest element is "free".
- Step 2 (merge from the right):

- Result:



$\rightsquigarrow$ trade-off: effort to find good pivots vs. increased merging costs

Undersampling: for $\theta \geq 1$ apply the median-of-pseudomedians-of-15 strategy to n/θ elements.

Theorem

MoMQuickMergesort with undersampling factor $\theta = 2.2$ needs at most $n \log n + 1.59n + o(n)$ comparisons.

Experiments

- Experiments with random permutations of 32-bit integers (other data types in proceedings) in C++.

Experiments

- Experiments with random permutations of 32-bit integers (other data types in proceedings) in C++.
- Not clear how to find worst-case instances.

Experiments

- Experiments with random permutations of 32-bit integers (other data types in proceedings) in C++.
- Not clear how to find worst-case instances.
- Simulation of worst cases:

Experiments

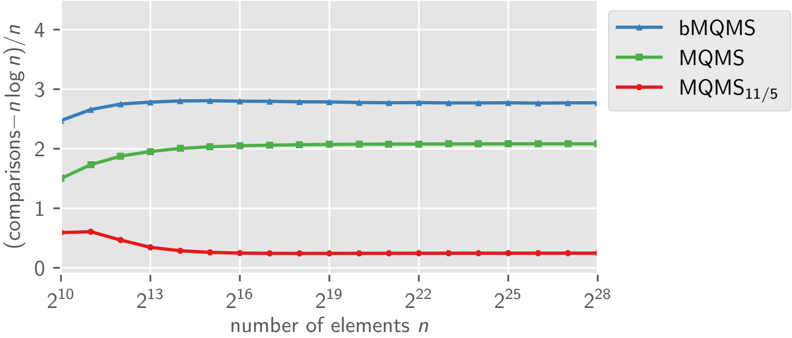
- Experiments with random permutations of 32-bit integers (other data types in proceedings) in C++.
- Not clear how to find worst-case instances.
- Simulation of worst cases:
    - Discard the computed pivots and compute worst-case pivots using Quickselect.

Experiments

- Experiments with random permutations of 32-bit integers (other data types in proceedings) in C++.
- Not clear how to find worst-case instances.
- Simulation of worst cases:
    - Discard the computed pivots and compute worst-case pivots using Quickselect.
    - Minor details (e. g. random shuffle before Mergesort).

Counting comparisons

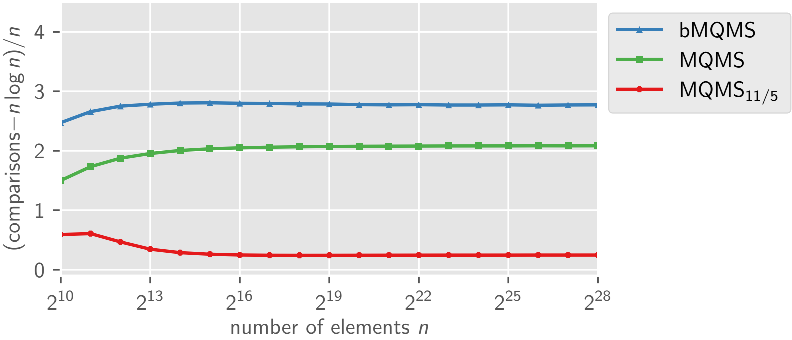
Algorithm	average case		worst case	
	exp.	theo.	exp.	theo.
bMQMS	2.772 ± 0.02			
MQMS	2.084 ± 0.001			
$\mathrm{MQMS}_{11/5}$	0.246 ± 0.01			



Number of comparisons (linear term) of MoMQuickMergesort variants

Counting comparisons

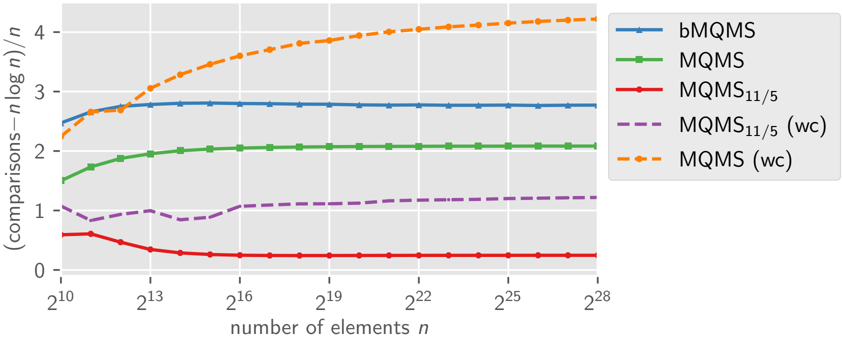
Algorithm	average case		worst case	
	exp.	theo.	exp.	theo.
bMQMS	2.772 ± 0.02	–		
MQMS	2.084 ± 0.001	2.094		
$\text{MQMS}_{11/5}$	0.246 ± 0.01	0.275		



Number of comparisons (linear term) of MoMQuickMergesort variants

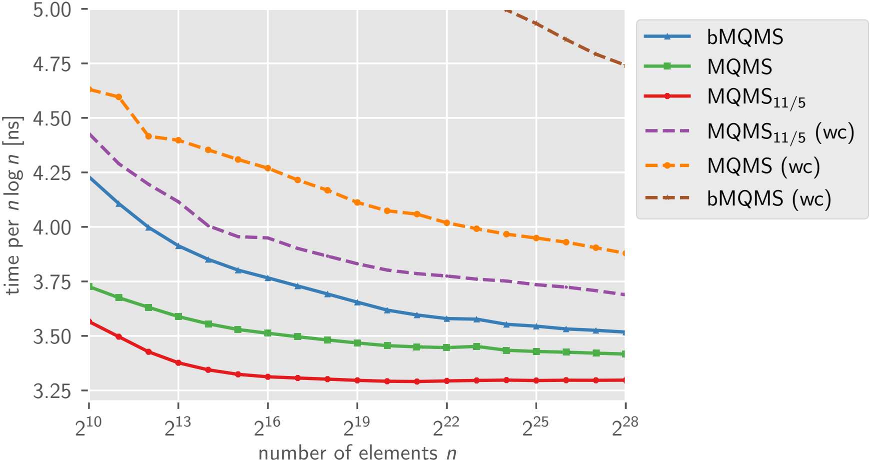
Counting comparisons

Algorithm	average case		worst case	
	exp.	theo.	exp.	theo.
bMQMS	2.772 $\pm$ 0.02	–	13.05 $\pm$ 0.17	13.8
MQMS	2.084 $\pm$ 0.001	2.094	4.220 $\pm$ 0.007	4.57
$\mathrm{MQMS}_{11/5}$	0.246 $\pm$ 0.01	0.275	1.218 $\pm$ 0.011	1.59



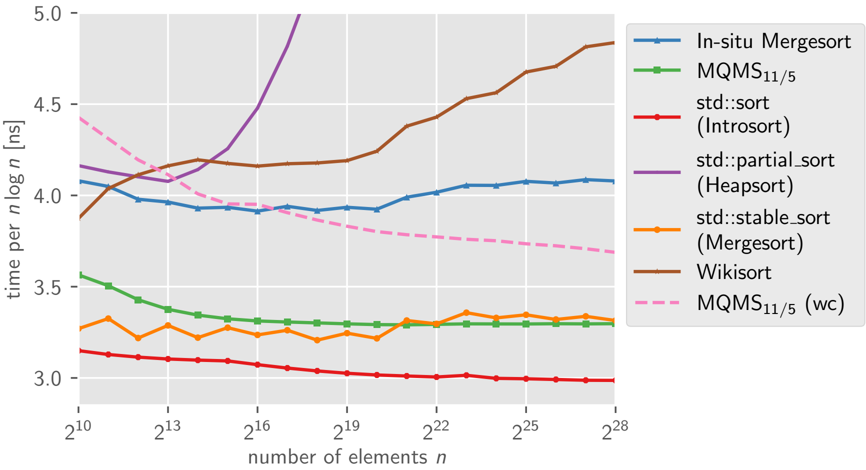
Number of comparisons (linear term) of MoMQuickMergesort variants and simulated worst cases.

Running times



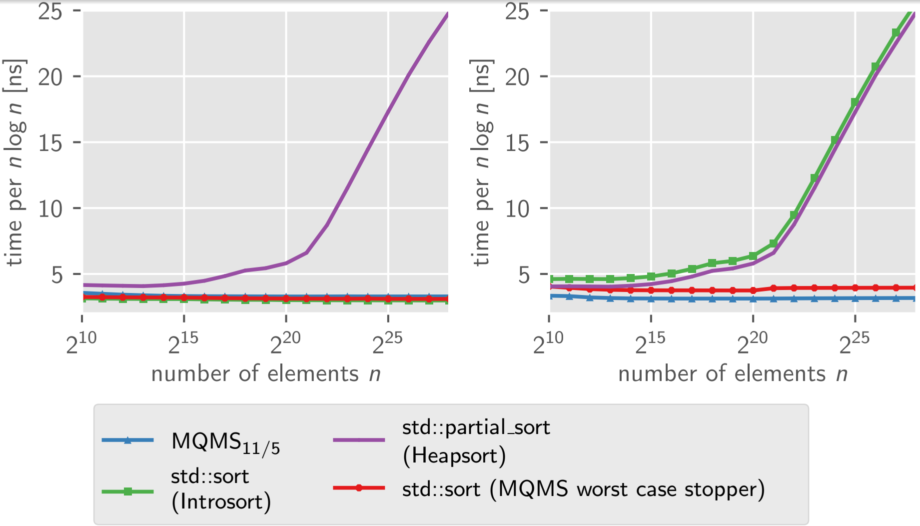
Running times of different MoMQuickMergesort variants and their simulated worst cases for random permutations of 32-bit integers.

Running times



Running times for random permutations of 32-bit integers.

Running times

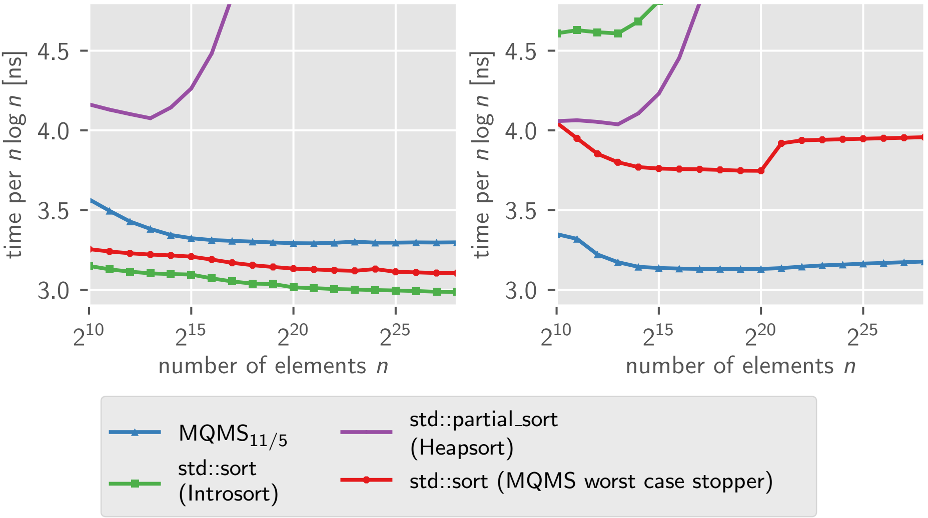


Running times for sorting integers.

Left: random inputs.

Right: Random with large elements in the middle and end.

Running times



Running times for sorting integers.

Left: random inputs.

Right: Random with large elements in the middle and end.

Conclusion

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓
MoMQuickMergesort	✓	✓	✓

[1] Code available at `https://github.com/weissan/QuickXsort`

Conclusion

Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓
MoMQuickMergesort	✓	✓	✓

- $\mathrm{MQMS}_{11/5}$ is an unstable sorting algorithm with
  - $n \log n + 1.59n + o(n)$ comparisons in the worst case
  - $n \log n + 0.275n + o(n)$ comparisons in the average case
- Implementation with `stl`-style interface[1].

[1] Code available at `https://github.com/weissan/QuickXsort`

Conclusion

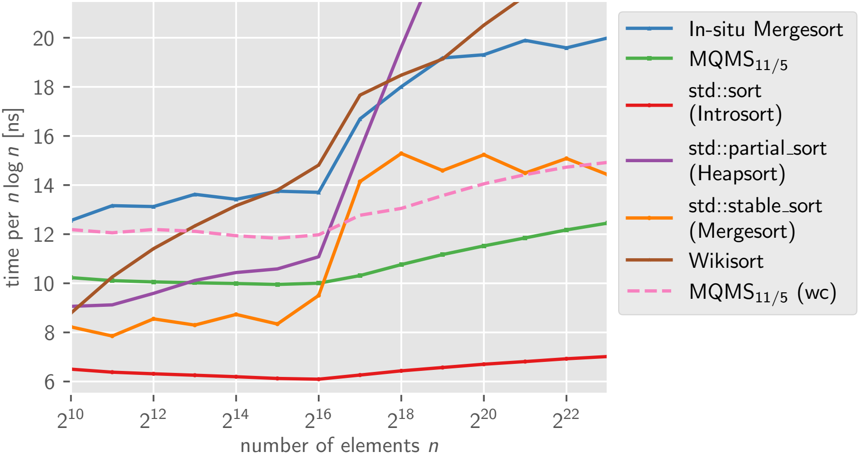
Algorithm	Fast on average	"in place"	$\mathcal{O}(n \log n)$ worst case
Quicksort	✓	✓	✗
Heapsort	✗	✓	✓
Mergesort	✓	✗	✓
MoMQuickMergesort	✓	✓	✓

- $\mathrm{MQMS}_{11/5}$ is an unstable sorting algorithm with
  - $n \log n + 1.59n + o(n)$ comparisons in the worst case
  - $n \log n + 0.275n + o(n)$ comparisons in the average case
- Implementation with `stl`-style interface[1].

Thank you!

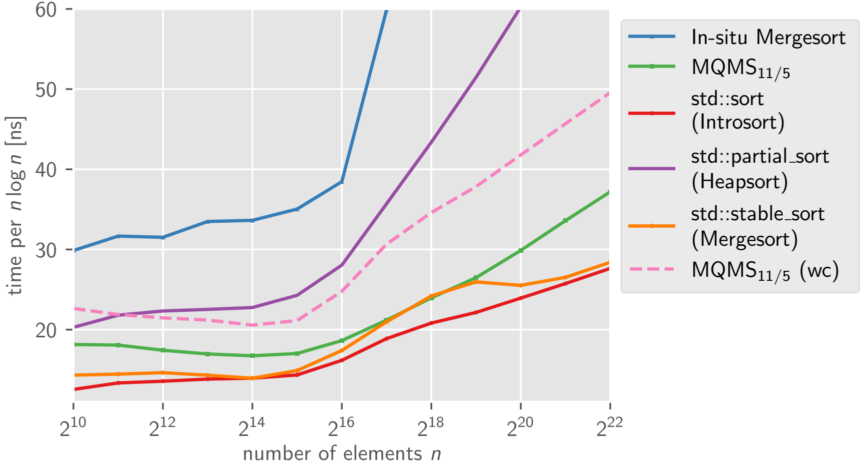
[1]Code available at `https://github.com/weissan/QuickXsort`

Experiments with larger records



Running times of MoMQuickMergesort (average and simulated worst case), hybrid QMS and other algorithms for random permutations 44-byte records with 4-byte keys.

Experiments sorting pointers



Running times of MoMQuickMergesort (average and simulated worst case), hybrid QMS and other algorithms for random permutations of pointers to records.

Experimental setup

- Experiments with random permutations of 32bit integers (other data types in proceedings).
- running time and comparison count
- ≥ 100 measurements for each data point
- Test environment:
  - Intel Core i5-2500K CPU (3.30GHz) with 16GB RAM
  - Ubuntu Linux 64bit version 14.04.4
  - g++ (4.8.4) compiler with flags `-O3 -march=native`