# Fibonacci Heaps – Practical Efficiency

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.

Constant factors in $O(\cdot)$ too high?

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

Standard (binary) heap: simple and memory-efficient (implicit representation)

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

Standard (binary) heap: simple and memory-efficient (implicit representation)

$k$-ary heap: generalizes $2$-ary heap (implicit_k in comparison)

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

Standard (binary) heap: simple and memory-efficient (implicit representation)

$k$-ary heap: generalizes $2$-ary heap (implicit_k in comparison)

$k$-ary heap, explicit representation: tree data struture instead of implicit in array

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

Standard (binary) heap: simple and memory-efficient (implicit representation)

$k$-ary heap: generalizes $2$-ary heap (implicit_k in comparison)

$k$-ary heap, explicit representation: tree data struture instead of implicit in array

strict Fibonacci heaps: a heap with same running times as Fibonacci heaps
but worst-case instead of amortized

# Fast Priority Queues - The Competitors

Fibonacci heaps: excellent amortized running times.
Constant factors in $O(\cdot)$ too high?

Binomial heaps: decreaseKey less efficient, but simpler

Pairing heaps: type of self-adjusting binomial heap, relatively simple

Standard (binary) heap: simple and memory-efficient (implicit representation)

$k$-ary heap: generalizes $2$-ary heap (implicit_k in comparison)

$k$-ary heap, explicit representation: tree data struture instead of implicit in array

strict Fibonacci heaps: a heap with same running times as Fibonacci heaps but worst-case instead of amortized

sequence heaps: a heap designed for efficient memory access

$\ldots$

# Empirical Study of Priority Queues [Larkin et al., 2014]

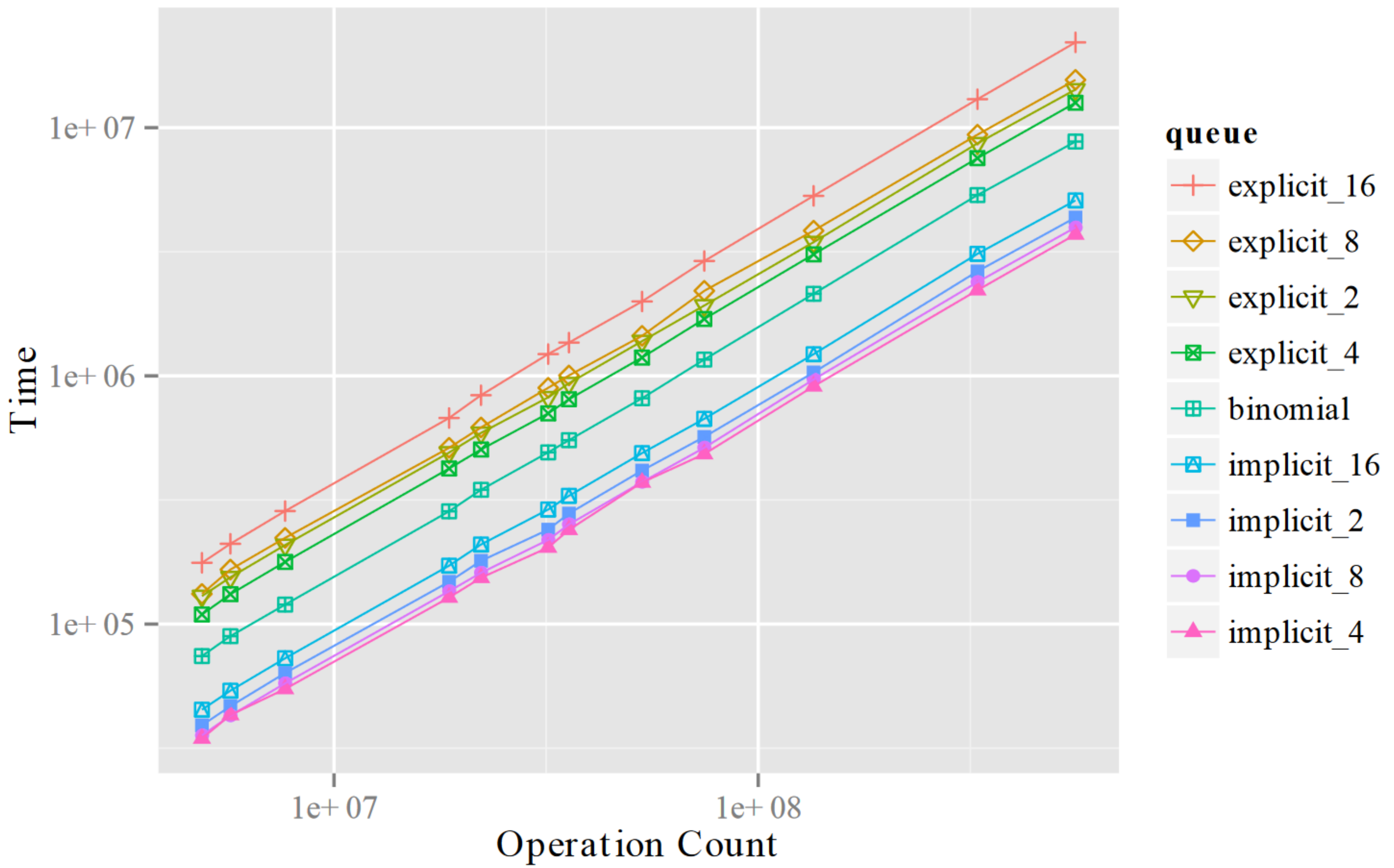
lines of code

| Heap variant | Logical lines of code (lloc) |
|---|---|
| implicit simple | 184 |
| pairing | 186 |
| implicit | 194 |
| Fibonacci | 282 |
| binomial | 317 |
| explicit | 319 |
| rank-pairing | 376 |
| quake | 383 |
| violation | 481 |
| rank-relaxed weak | 638 |
| strict Fibonacci | 1009 |

# Empirical Study of Priority Queues [Larkin et al., 2014]

## Dijkstra on US street network, plot 1 of 2

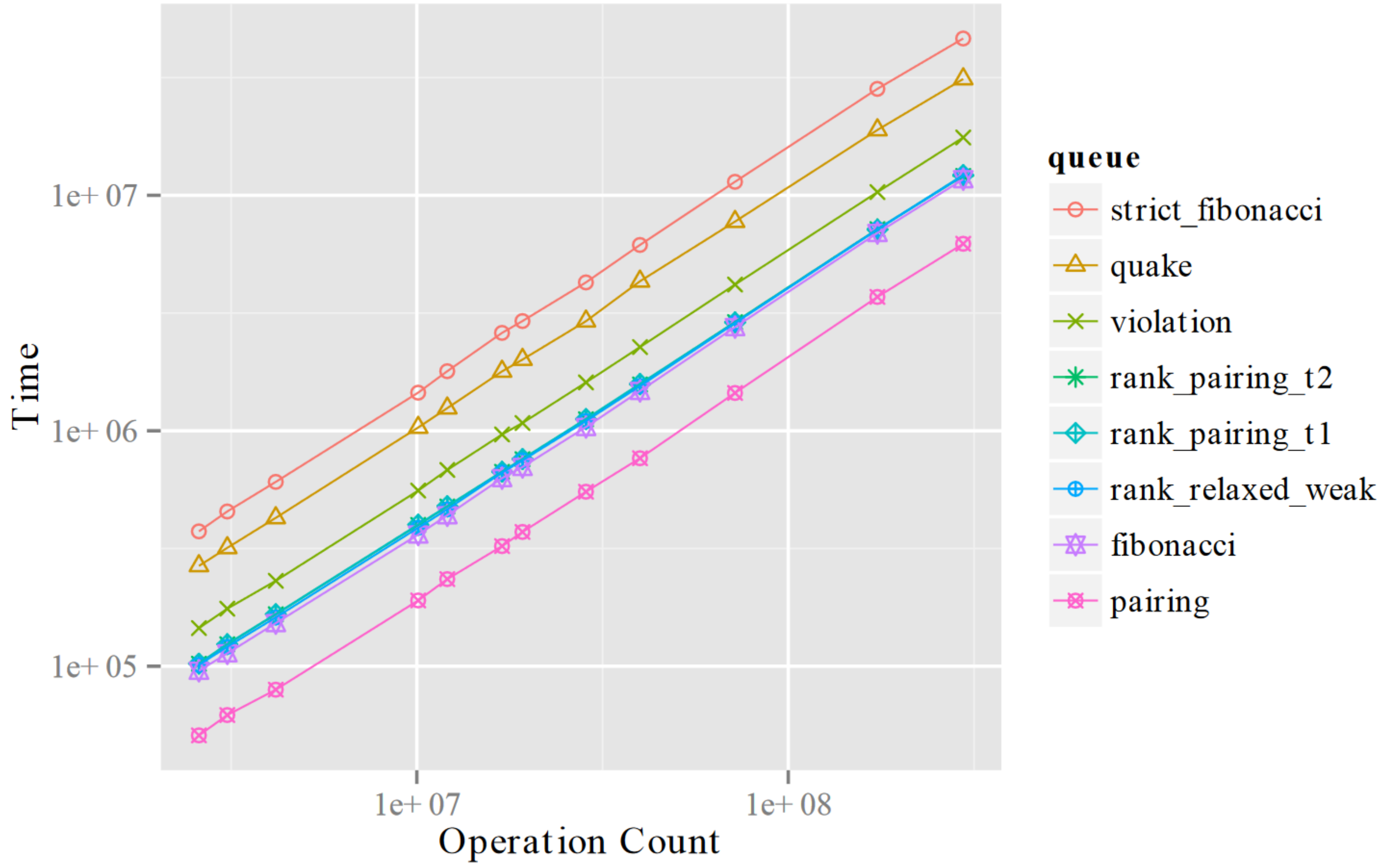
Operation Count = number of Insert + DeleteMin + DecreaseKey operations



# Empirical Study of Priority Queues [Larkin et al., 2014]

## Dijkstra on US street network, plot 2 of 2

Operation Count = number of Insert + DeleteMin + DecreaseKey operations



# Empirical Study of Priority Queues [Larkin et al., 2014]

## Dijkstra on US street network, table

Heap Size – max = 4200, average = 2489
Ratio of Operations – INSERT : DELETEMIN : DECREASEKEY = 13.98 : 13.98 : 1.00

| queue | time | inst | l1_rd | l1_wr | l2_rd | l2_wr | br | l1_m | l2_m | br_m |
|---|---|---|---|---|---|---|---|---|---|---|
| implicit_4 | **1.00** | **1.00** | **1.00** | 1.10 | 1.35 | 1.06 | **1.00** | **1.00** | 1.00 | **1.00** |
| implicit_8 | 1.07 | 1.12 | 1.12 | 1.03 | 1.61 | 1.18 | 1.01 | 1.07 | 1.20 | 1.01 |
| implicit_2 | 1.17 | 1.10 | 1.01 | 1.27 | 1.35 | **1.00** | 1.33 | 1.05 | **1.00** | 1.33 |
| implicit_16 | 1.37 | 1.42 | 1.38 | **1.00** | 2.20 | 1.35 | 1.21 | 1.24 | 1.63 | 1.21 |
| pairing | 1.68 | 1.09 | 1.12 | 2.95 | 1.71 | 28.57 | 1.39 | 1.60 | 1.75 | 1.39 |
| binomial | 2.37 | 1.49 | 1.83 | 3.49 | 1.30 | 34.57 | 1.49 | 2.24 | 1.56 | 1.49 |
| fibonacci | 3.15 | 2.00 | 2.09 | 5.03 | 1.73 | 79.53 | 2.91 | 2.85 | 2.67 | 2.91 |
| violation | 4.74 | 2.85 | 2.67 | 3.92 | 2.60 | 4.24 | 4.38 | 2.95 | 1.97 | 4.38 |
| explicit_16 | 5.94 | 3.94 | 4.56 | 6.81 | 8.02 | 276.59 | 7.13 | 5.06 | 10.76 | 7.13 |
| quake | 8.40 | 5.84 | 6.82 | 10.69 | 3.45 | 137.91 | 6.90 | 7.72 | 4.97 | 6.90 |
| strict_fibonacci | 12.49 | 9.47 | 12.50 | 22.07 | 6.96 | 84.51 | 11.47 | 14.83 | 6.58 | 11.47 |

column titles have been abbreviated: `time` is wallclock time, `inst` is the dynamic instruction count, `l1_rd` and `l1_wr` are the number of L1 reads and writes respectively, `l2_rd` and `l2_wr` are the L2 reads and writes respectively, `br` is the number of dynamic branches, and `l1_m`, `l2_m` and `br_m` are the number of L1 misses, L2 misses, and branch mispredictions.

# Empirical Study of Priority Queues [Larkin et al., 2014]

## Dijkstra on US street network, table

Heap Size – max = 4200, average = 2489
Ratio of Operations – INSERT : DELETEMIN : DECREASEKEY = 13.98 : 13.98 : 1.00

| queue | time | inst | l1_rd | l1_wr | l2_rd | l2_wr | br | l1_m | l2_m | br_m |
|---|---|---|---|---|---|---|---|---|---|---|
| implicit_4 | **1.00** | **1.00** | **1.00** | 1.10 | 1.35 | 1.06 | **1.00** | **1.00** | 1.00 | **1.00** |
| implicit_8 | 1.07 | 1.12 | 1.12 | 1.03 | 1.61 | 1.18 | 1.01 | 1.07 | 1.20 | 1.01 |
| implicit_2 | 1.17 | 1.10 | 1.01 | 1.27 | 1.35 | **1.00** | 1.33 | 1.05 | **1.00** | 1.33 |
| implicit_16 | 1.37 | 1.42 | 1.38 | **1.00** | 2.20 | 1.35 | 1.21 | 1.24 | 1.63 | 1.21 |
| pairing | 1.68 | 1.09 | 1.12 | 2.95 | 1.71 | 28.57 | 1.39 | 1.60 | 1.75 | 1.39 |
| binomial | 2.37 | 1.49 | 1.83 | 3.49 | 1.30 | 34.57 | 1.49 | 2.24 | 1.56 | 1.49 |
| fibonacci | 3.15 | 2.00 | 2.09 | 5.03 | 1.73 | 79.53 | 2.91 | 2.85 | 2.67 | 2.91 |
| rank_pairing_t2 | 3.26 | 1.98 | 2.16 | 2.85 | 1.34 | 35.46 | 3.19 | 2.29 | 1.61 | 3.19 |
| rank_relaxed_weak | 3.27 | 2.21 | 2.72 | 3.62 | 2.34 | 10.01 | 3.08 | 2.90 | 1.89 | 3.08 |
| rank_pairing_t1 | 3.29 | 1.98 | 2.16 | 2.85 | 1.33 | 35.35 | 3.19 | 2.29 | 1.60 | 3.19 |
| explicit_4 | 3.39 | 2.69 | 2.83 | 4.11 | 1.97 | 104.57 | 4.22 | 3.11 | 3.29 | 4.22 |
| explicit_2 | 3.84 | 3.35 | 3.39 | 4.84 | **1.00** | 74.61 | 5.01 | 3.71 | 2.05 | 5.01 |
| explicit_8 | 4.20 | 3.01 | 3.32 | 5.00 | 4.50 | 168.91 | 5.04 | 3.70 | 6.28 | 5.04 |
| violation | 4.74 | 2.85 | 2.67 | 3.92 | 2.60 | 4.24 | 4.38 | 2.95 | 1.97 | 4.38 |
| explicit_16 | 5.94 | 3.94 | 4.56 | 6.81 | 8.02 | 276.59 | 7.13 | 5.06 | 10.76 | 7.13 |
| quake | 8.40 | 5.84 | 6.82 | 10.69 | 3.45 | 137.91 | 6.90 | 7.72 | 4.97 | 6.90 |
| strict_fibonacci | 12.49 | 9.47 | 12.50 | 22.07 | 6.96 | 84.51 | 11.47 | 14.83 | 6.58 | 11.47 |

# Empirical Study of Priority Queues [Larkin et al., 2014]

$n$ insertions, then $n$ times: insert + decreaseKey with new minimum + deleteMin

Table 6: Randomized DecreaseKey – Min, $c = 1$, $k = 1$
Heap Size – max = 8388609, average = 7340032
Ratio of Operations – Insert : DeleteMin : DecreaseKey = 2.00 : 1.00 : 1.00

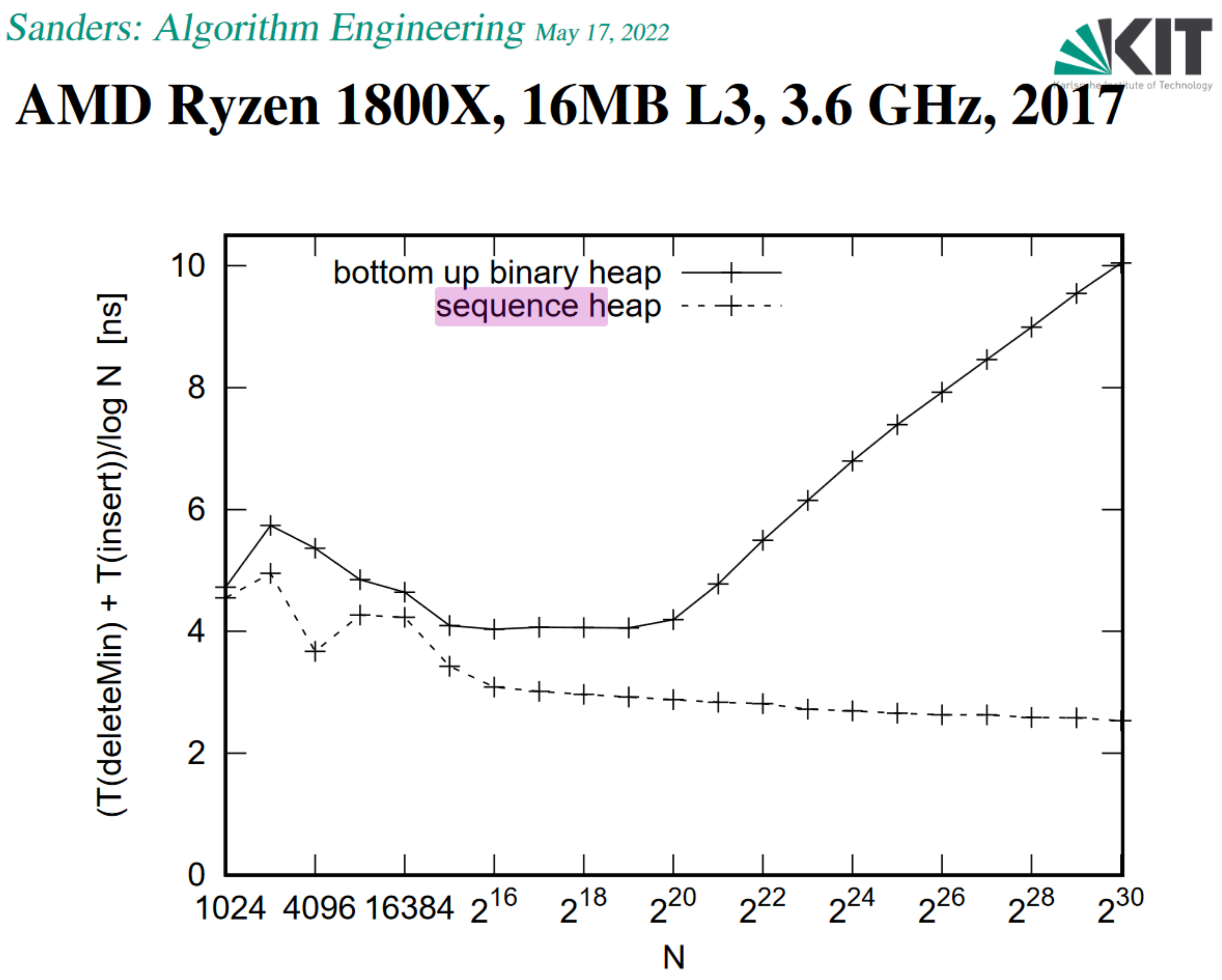
| queue | time | inst | l1_rd | l1_wr | l2_rd | l2_wr | br | l1_m | l2_m | br_m |
|---|---|---|---|---|---|---|---|---|---|---|
| pairing | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 1.65 | **1.00** | **1.00** | **1.00** | **1.00** |
| fibonacci | 3.04 | 2.62 | 2.21 | 2.58 | 3.00 | 3.52 | 4.91 | 2.37 | 2.43 | 4.91 |
| rank_relaxed_weak | 3.65 | 3.16 | 3.05 | 2.18 | 4.66 | 2.01 | 5.89 | 2.68 | 2.40 | 5.89 |
| rank_pairing_t2 | 5.43 | 3.42 | 2.57 | 1.96 | 5.94 | 1.58 | 7.54 | 2.32 | 2.67 | 7.54 |
| rank_pairing_t1 | 5.95 | 3.39 | 2.56 | 1.95 | 5.94 | 1.57 | 7.46 | 2.30 | 2.67 | 7.46 |
| violation | 6.32 | 4.54 | 3.32 | 2.40 | 5.93 | 1.04 | 10.47 | 2.93 | 2.46 | 10.47 |
| quake | 6.55 | 6.59 | 4.37 | 3.53 | 5.28 | 2.61 | 13.31 | 4.02 | 2.86 | 13.31 |
| implicit_8 | 6.98 | 3.58 | 2.42 | 1.24 | 40.06 | 1.02 | 5.55 | 1.92 | 14.19 | 5.55 |
| implicit_4 | 7.25 | 3.47 | 2.25 | 1.41 | 33.17 | **1.00** | 6.20 | 1.90 | 11.81 | 6.20 |
| strict_fibonacci | 7.39 | 8.44 | 8.38 | 8.31 | 9.76 | 4.81 | 13.05 | 8.35 | 5.27 | 13.05 |
| implicit_16 | 9.40 | 4.35 | 2.94 | 1.14 | 56.29 | 1.05 | 6.02 | 2.18 | 19.79 | 6.02 |
| implicit_2 | 10.17 | 4.48 | 2.59 | 1.93 | 38.66 | 1.01 | 9.67 | 2.31 | 13.70 | 9.67 |
| binomial | 12.14 | 5.90 | 5.55 | 6.54 | 30.10 | 14.21 | 10.45 | 5.97 | 16.01 | 10.45 |
| explicit_4 | 14.95 | 12.84 | 9.64 | 8.33 | 24.42 | 18.16 | 30.31 | 9.09 | 15.63 | 30.31 |
| explicit_2 | 16.24 | 17.12 | 12.53 | 10.60 | 24.32 | 12.53 | 38.42 | 11.71 | 13.36 | 38.42 |
| explicit_8 | 21.07 | 13.80 | 10.88 | 9.74 | 39.17 | 28.94 | 34.46 | 10.40 | 25.00 | 34.46 |
| explicit_16 | 31.01 | 17.68 | 14.40 | 12.85 | 60.51 | 48.34 | 47.08 | 13.74 | 40.06 | 47.08 |

# Practical Heaps: Conclusion and Libraries

- Fibonacci heaps perform well, but pairing heaps and implicit $k$-ary heaps perform typically better
- choice of heap depends on input

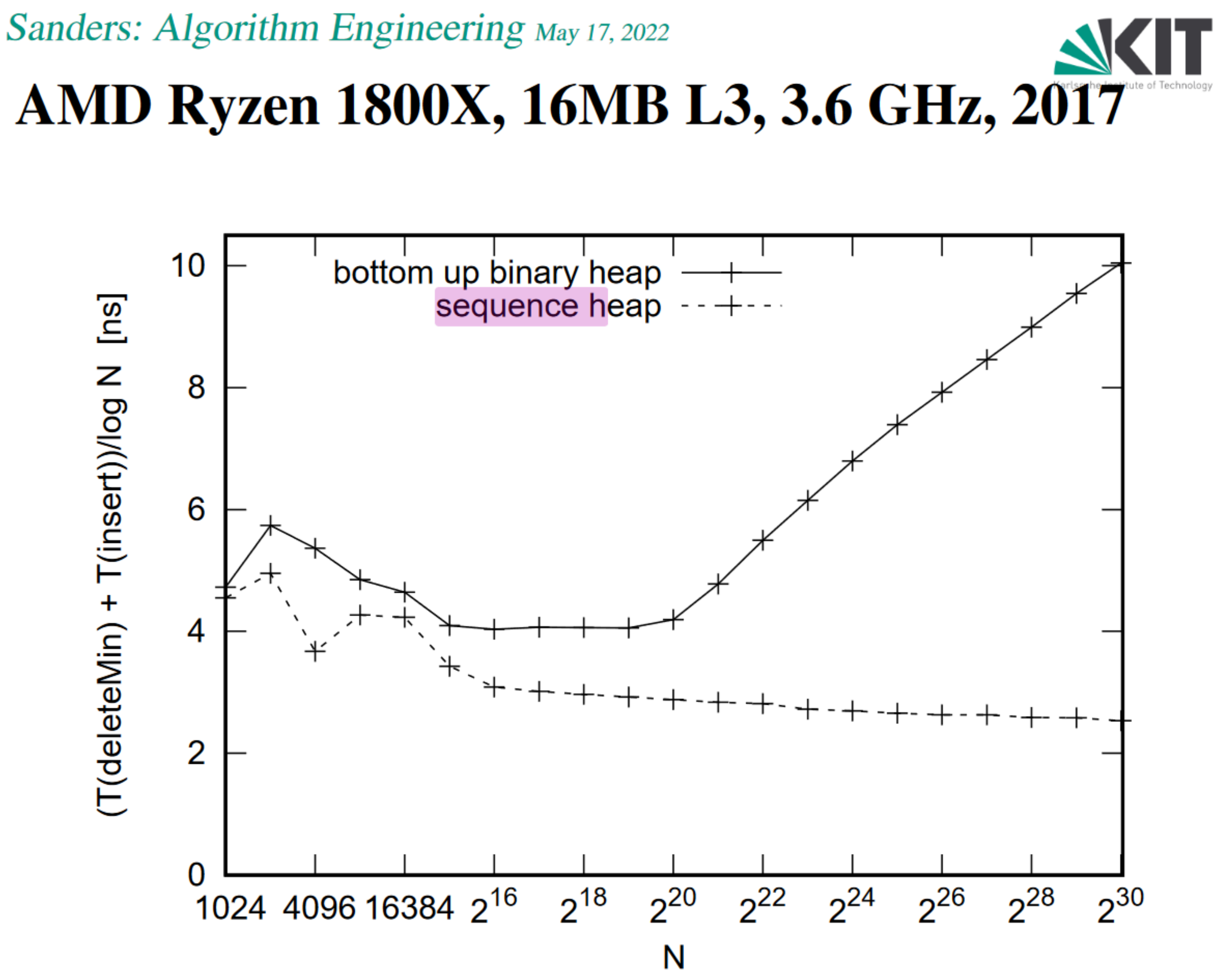
# Practical Heaps: Conclusion and Libraries

- Fibonacci heaps perform well, but pairing heaps and implicit $k$-ary heaps perform typically better
- choice of heap depends on input
- very efficient but not (fully) included in comparison: sequence heaps [P. Sanders, 2000]



# Practical Heaps: Conclusion and Libraries

- Fibonacci heaps perform well, but pairing heaps and implicit $k$-ary heaps perform typically better
- choice of heap depends on input
- very efficient but not (fully) included in comparison: sequence heaps [P. Sanders, 2000]




## Fibonacci heaps in libraries

- Fibonacci, Binomial, Pairing, $k$-ary heaps are included in boost (C++)
- Pairing heaps in GNU C++ Library

# Practical Heaps: References


Larkin, Daniel H., Siddhartha Sen, and Robert E. Tarjan. "A back-to-basics empirical study of priority queues." Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX), 2014.

Sanders, Peter. "Fast priority queues for cached memory." Journal of Experimental Algorithmics (JEA) 5 (2000): 7-es. (plot from Peter Sanders *Algorithm Engineering* lecture)

Fredman, Michael L., et al. "The pairing heap: A new form of self-adjusting heap." Algorithmica 1.1-4 (1986): 111-129.

Mehlhorn, Kurt, Peter Sanders, and Peter Sanders. Algorithms and data structures: The basic toolbox. Vol. 55. Berlin: Springer, 2008. (for reading about pairing heaps)


boost: `https://www.boost.org/`

GNU C++ library: `https://gcc.gnu.org/onlinedocs/libstdc++/`