4 Efficient Sorting

3 November 2025

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2025–26)
Philipps-Universität Marburg

version `2025-11-03 15:55`

Learning Outcomes

Unit 4: ***Efficient Sorting***

1. Know principles and implementation of *mergesort* and *quicksort*.
2. Know properties and *performance characteristics* of mergesort and quicksort.
3. Know the comparison model and understand the corresponding *lower bound*.
4. Understand *counting sort* and how it circumvents the comparison lower bound.
5. Know ways how to exploit *presorted* inputs.

Outline

4 Efficient Sorting

Why study sorting?

- fundamental problem of computer science that is still not solved
  Algorithm with optimal #comparisons in worst case?
- building brick of many more advanced algorithms
  - for preprocessing
  - as subroutine
- playground of manageable complexity
  to practice algorithmic techniques

Here:

- "classic" fast sorting method
- exploit **partially sorted** inputs
- **parallel** sorting   later

Part I

The Basics

Rules of the game

- **Given:**
  - array $A[0..n) = A[0..n-1]$ of n objects
  - a total order relation $\leq$ among $A[0], \ldots, A[n-1]$
    (a comparison function)
    Python: elements support <= operator (`__le__()`)
    Java: `Comparable` class (`x.compareTo(y) <= 0`)
- **Goal:** rearrange (i. e., permute) elements within A,
  so that A is *sorted*, i. e., $A[0] \leq A[1] \leq \cdots \leq A[n-1]$

- for now: A stored in main memory (*internal sorting*)
  single processor (*sequential sorting*)

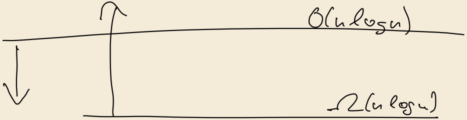
Clicker Question

$\Theta(n \log n)$

What is the complexity of sorting? Type you answer, e. g., as "`Theta(sqrt(n))`"

(a) $O(n \log n)$ algorithm solving the problem

(b) lower bound for problem $\Omega(n \log n)$

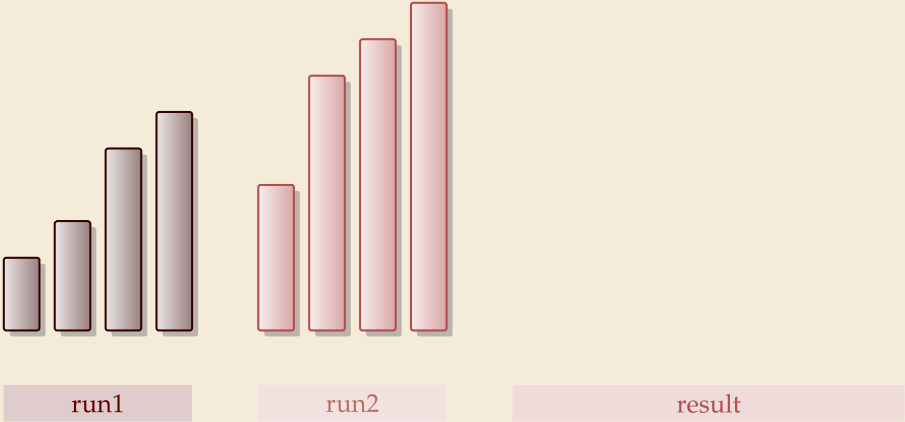


→ *sli.do/cs566*

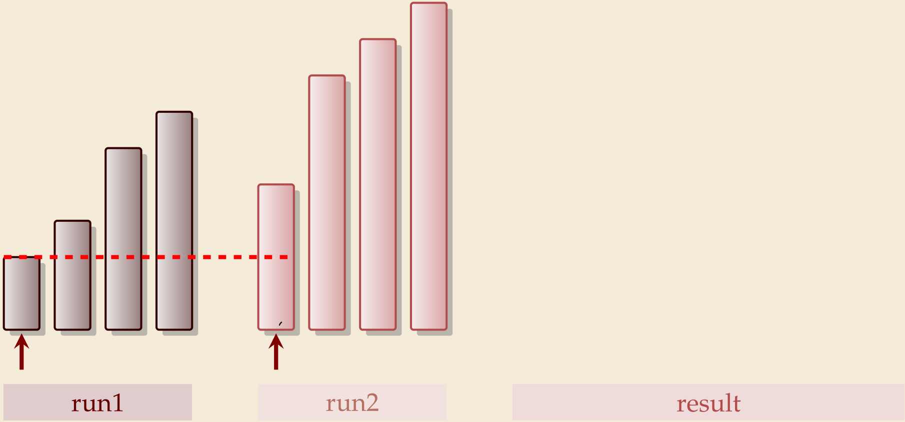
4.1 Mergesort

Merging sorted lists

Merging sorted lists

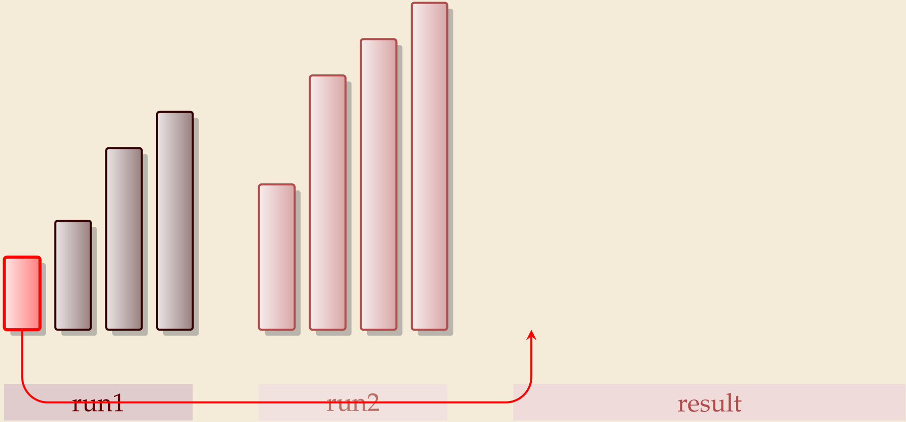


Merging sorted lists

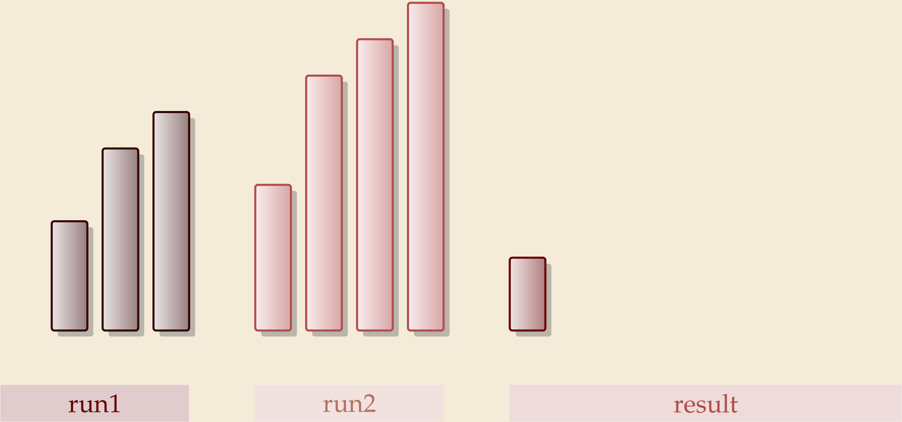


Merging sorted lists

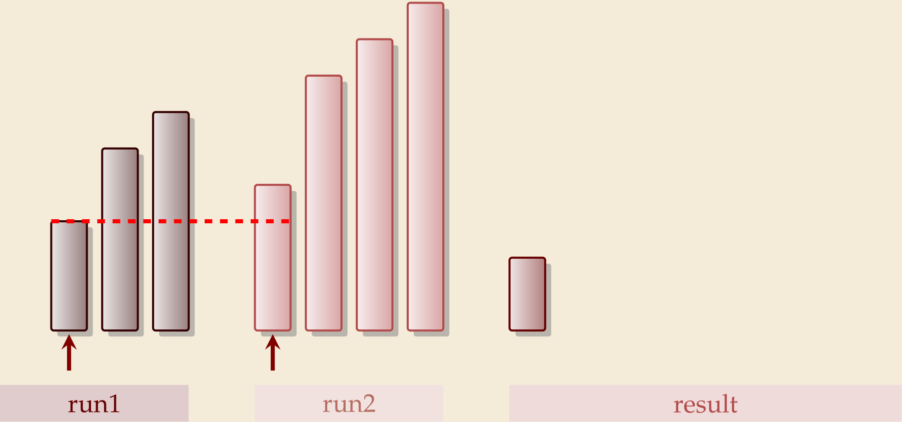
Merging sorted lists



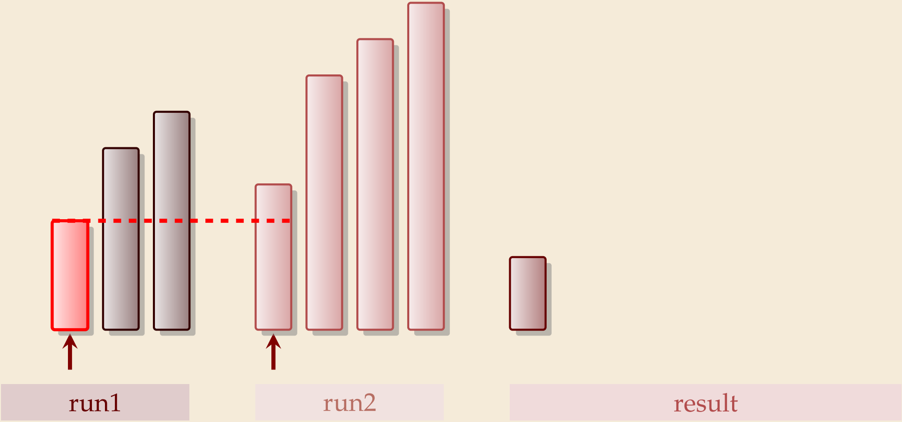
Merging sorted lists



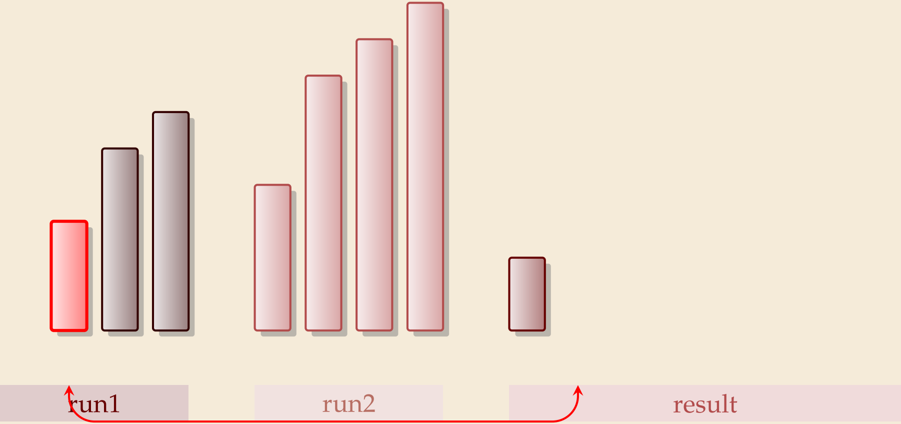
Merging sorted lists



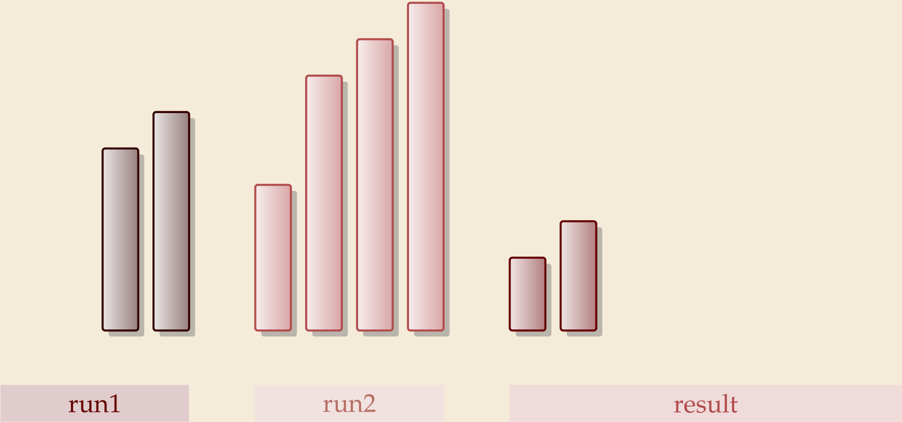
Merging sorted lists



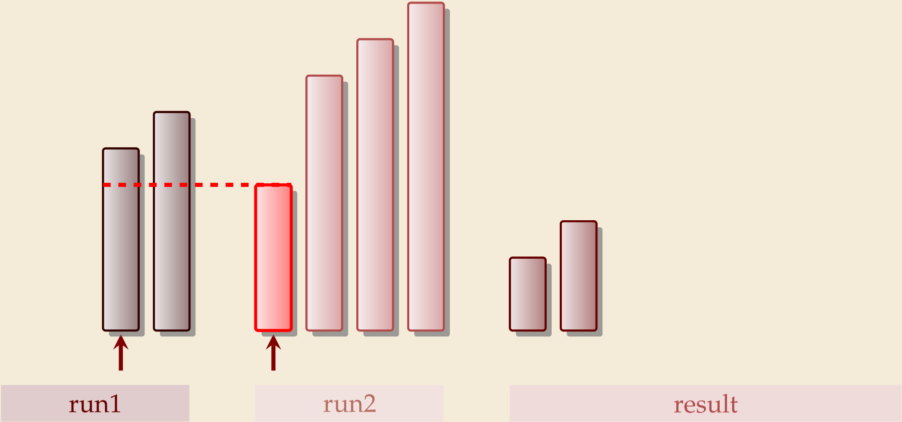
Merging sorted lists



Merging sorted lists

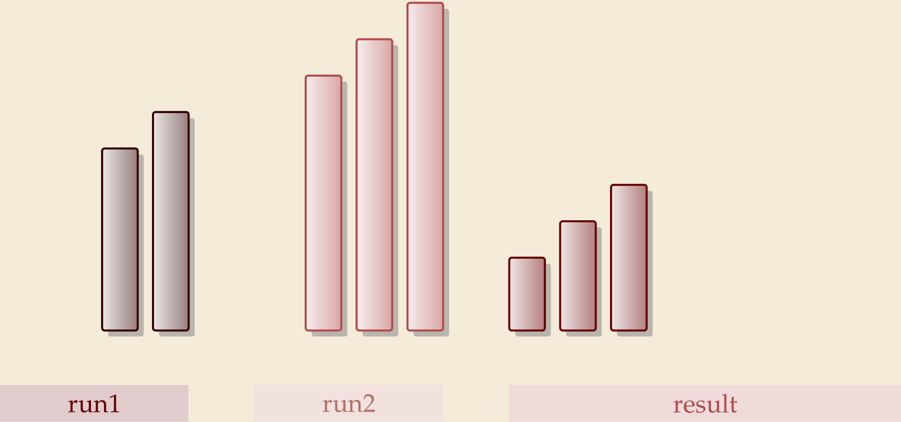


Merging sorted lists



Merging sorted lists

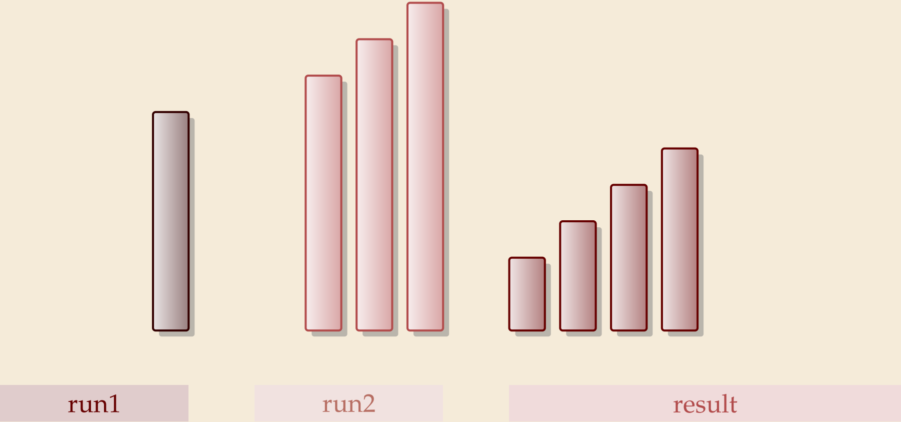
Merging sorted lists



Merging sorted lists

Merging sorted lists

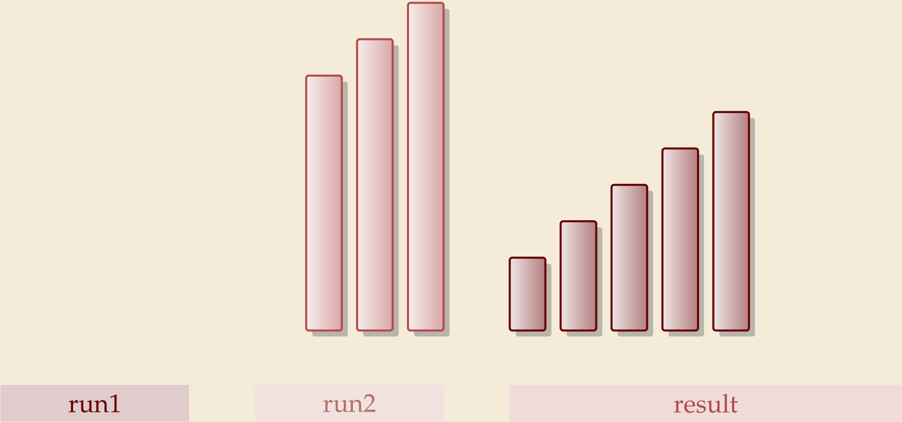
Merging sorted lists



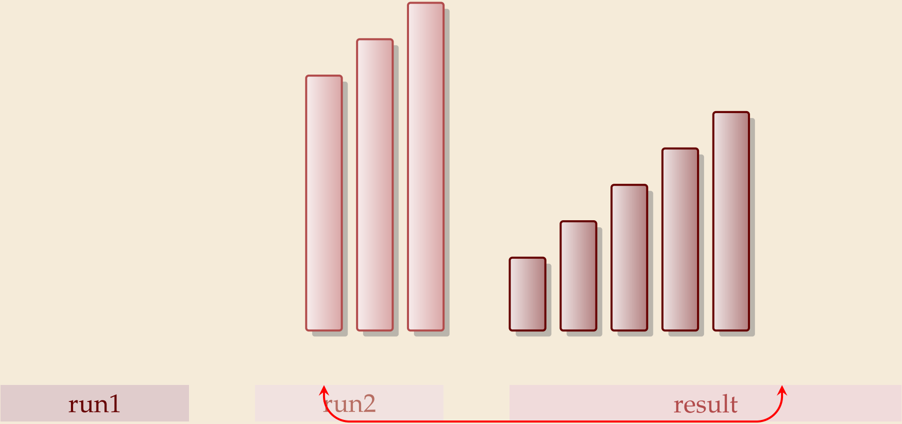
Merging sorted lists

Merging sorted lists

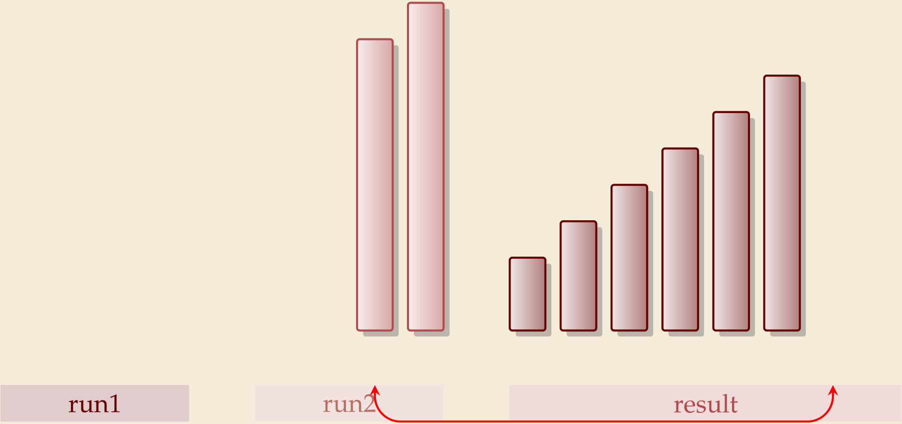
Merging sorted lists



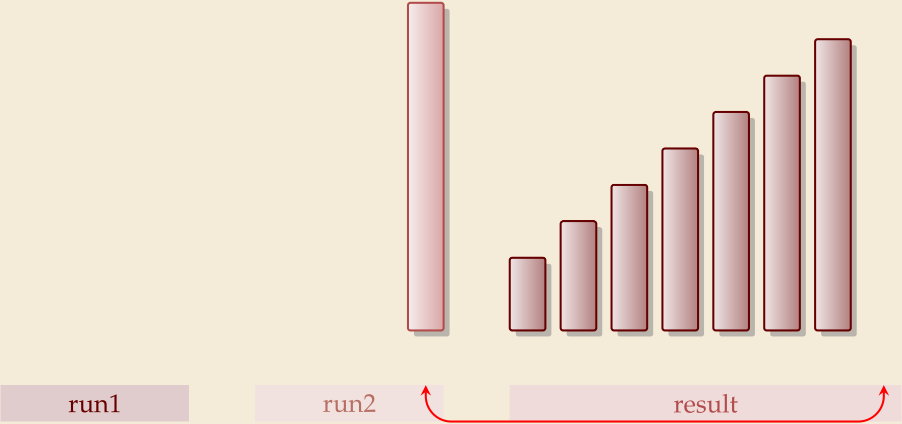
Merging sorted lists



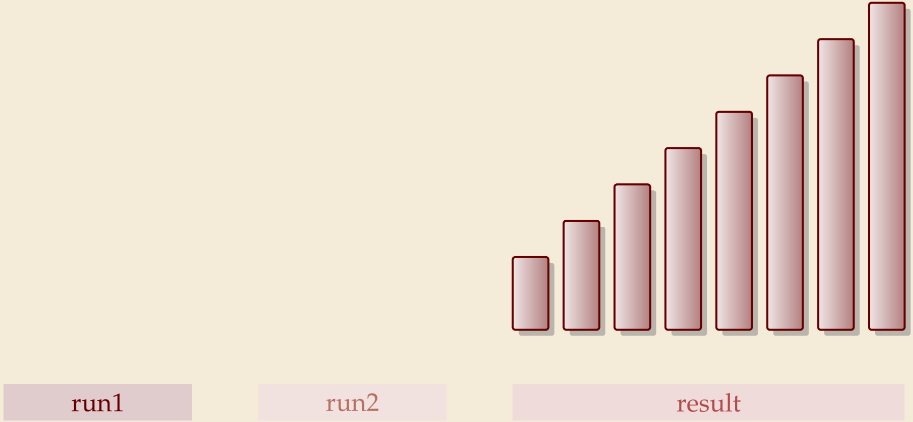
Merging sorted lists



Merging sorted lists



Merging sorted lists



Clicker Question

What is the worst-case running time of mergesort?

A $\Theta(1)$

B $\Theta(\log n)$

C $\Theta(\log \log n)$

D $\Theta(\sqrt{n})$

E $\Theta(n)$

F $\Theta(n \log \log n)$

G $\Theta(n \log n)$

H $\Theta(n \log^2 n)$

I $\Theta(n^{1+\epsilon})$

J $\Theta(n^2)$

K $\Theta(n^3)$

L $\Theta(2^n)$

→ `sli.do/cs566`

Clicker Question

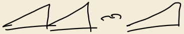
What is the worst-case running time of mergesort?

- **A** $\cancel{\Theta(1)}$
- **B** $\cancel{\Theta(\log n)}$
- **C** $\cancel{\Theta(\log \log n)}$
- **D** $\cancel{\Theta(\sqrt{n})}$
- **E** $\cancel{\Theta(n)}$
- **F** $\cancel{\Theta(n \log \log n)}$
- **G** $\Theta(n \log n)$
- **H** $\cancel{\Theta(n \log^2 n)}$
- **I** $\cancel{\Theta(n^{1+\epsilon})}$
- **J** $\cancel{\Theta(n^2)}$
- **K** $\cancel{\Theta(n^3)}$
- **L** $\cancel{\Theta(2^n)}$

→ sli.do/cs566

Mergesort

```
procedure mergesort(A[l..r]):
    n := r − l
    if n ≤ 1 return
    m := l + ⌊n/2⌋
    mergesort(A[l..m])
    mergesort(A[m..r])
    merge(A[l..m), A[m..r), buf)
    copy buf to A[l..r)
```



- recursive procedure
- merging needs
  - temporary storage *buf* for result (of same size as merged runs)
  - to read and write each element twice (once for merging, once for copying back)

Mergesort

```
procedure mergesort(A[l..r]):
    n := r − l
    if n ≤ 1 return
    m := l + ⌊n/2⌋
    mergesort(A[l..m))
    mergesort(A[m..r))
    merge(A[l..m), A[m..r), buf)
    copy buf to A[l..r)
```

- recursive procedure
- merging needs
  - temporary storage *buf* for result (of same size as merged runs)
  - to read and write each element twice (once for merging, once for copying back)

Analysis: count *"element visits"* (read and/or write)

$$C(n) \;=\; \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

same for best and worst case!

Simplification $\boxed{n = 2^k}$

$$C(2^k) \;=\; \begin{cases} 0 & k \le 0 \\ 2 \cdot C(2^{k-1}) + 2 \cdot 2^k & k \ge 1 \end{cases} \;=\; 2 \cdot 2^k + 2^2 \cdot 2^{k-1} + 2^3 \cdot 2^{k-2} + \cdots + 2^k \cdot 2^1 \;=\; 2k \cdot 2^k$$

$$C(n) \;=\; 2n\lg(n) \;=\; \Theta(n \log n)$$

(arbitrary n: $C(n) \;\le\; C(\text{next larger power of 2}) \;\le\; 4n\lg(n) + 2n \;=\; \Theta(n \log n)$)

Mergesort

```
procedure mergesort(A[l..r]):
    n := r − l
    if n ≤ 1 return
    m := l + ⌊n/2⌋
    mergesort(A[l..m))
    mergesort(A[m..r))
    merge(A[l..m), A[m..r), buf)
    copy buf to A[l..r)
```

- recursive procedure
- merging needs
  - temporary storage *buf* for result (of same size as merged runs)
  - to read and write each element twice (once for merging, once for copying back)

Analysis: count *"element visits"* (read and/or write)

$$C(n) \;=\; \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

same for best and worst case!

(precisely(!) solvable *without* assumption $n = 2^k$: $C(n) = 2n\lg(n) + \big(2 - \{\lg(n)\} - 2^{1-\{\lg(n)\}}\big)2n$ with $\{x\} := x - \lfloor x \rfloor$)

Simplification $\boxed{n = 2^k}$

$$C(2^k) \;=\; \begin{cases} 0 & k \le 0 \\ 2 \cdot C(2^{k-1}) + 2 \cdot 2^k & k \ge 1 \end{cases} \;=\; 2 \cdot 2^k + 2^2 \cdot 2^{k-1} + 2^3 \cdot 2^{k-2} + \cdots + 2^k \cdot 2^1 \;=\; 2k \cdot 2^k$$

$$C(n) \;=\; 2n\lg(n) \;=\; \Theta(n \log n)$$

(arbitrary n: $C(n) \;\le\; C(\text{next larger power of 2}) \;\le\; \underline{4n\lg(n) + 2n} \;=\; \Theta(n \log n)$)

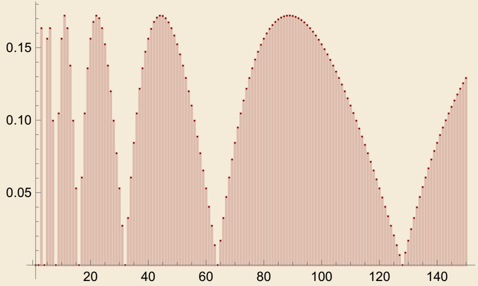
Linear Term of $C(n)$

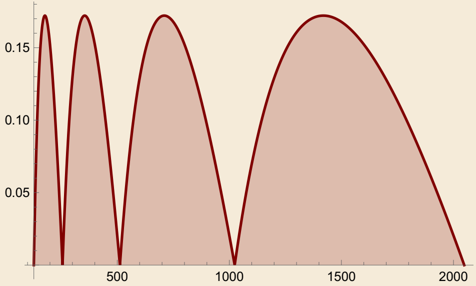
$\notin$ exam

Recall:

$C(n) \;=\; 2n\lg(n) + \left(2 - \{\lg(n)\} - 2^{1-\{\lg(n)\}}\right)2n$ with $\{x\} := x - \lfloor x \rfloor$

Plot of $2\left(2 - \{\lg(n)\} - 2^{1-\{\lg(n)\}}\right)$





$\rightsquigarrow$ Can prove: $\boxed{C(n) \;\le\; 2n\lg n + 0.172n}$

Mergesort – Discussion

- optimal time complexity of $\Theta(n \log n)$ in the worst case
- *stable* sorting method i. e., retains relative order of equal-key items
- memory access is sequential (scans over arrays)
- requires $\Theta(n)$ extra space

there are in-place merging methods,
but they are substantially more complicated
and not (widely) used