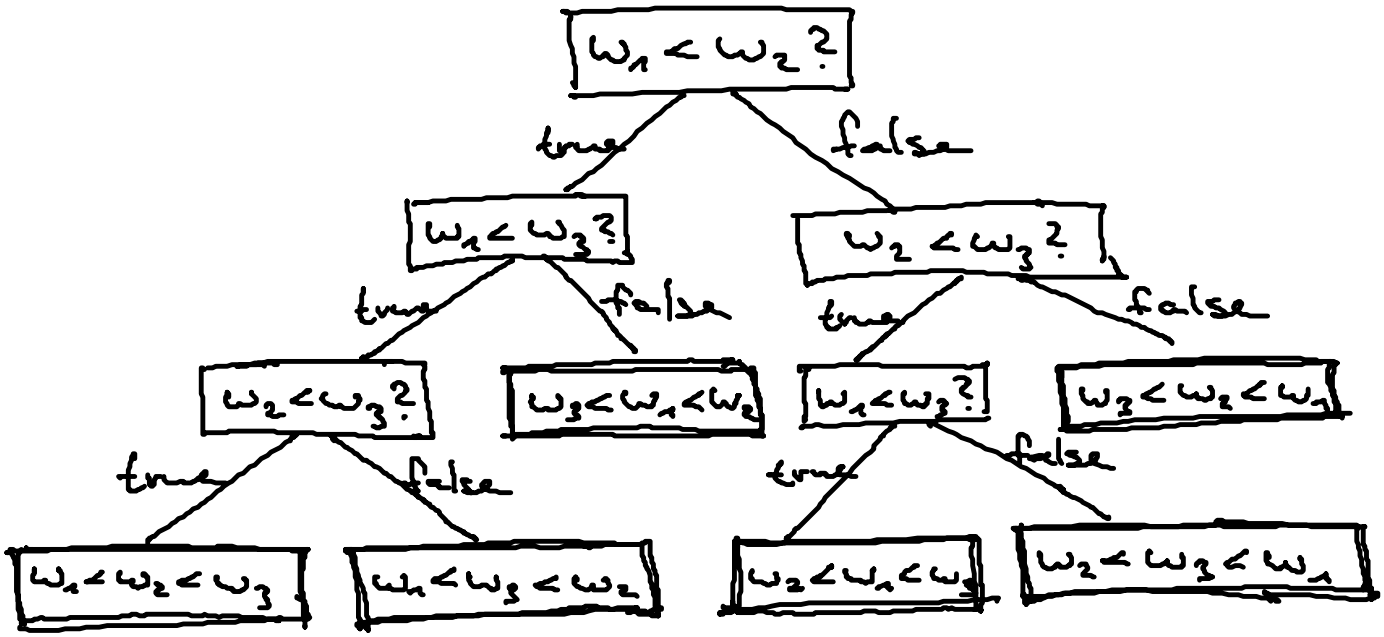
Untere Komplexitätsschranke für Sortieren

Theorem: Jedes Sortierverfahren, das auf paarweisen Vergleichen beruht, benötigt im Worst Case und auch im Average Case mindestens $\Omega(n \cdot \log n)$ Vergleiche.

Beweis: Entscheidungsbaum.

Bsp: SelectionSort für



Bsp: MergeSort

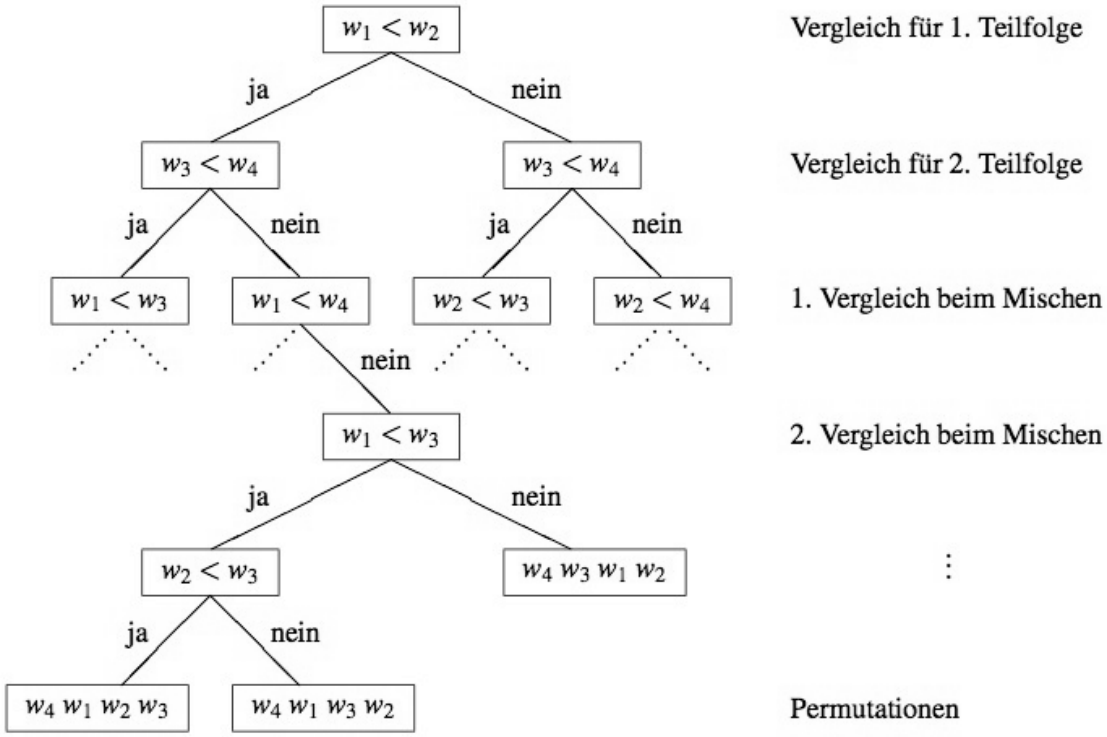


Abbildung 11.1: Der Entscheidungsbaum für Mergesort.

Beobachtung: Der Entscheidungsbaum hat genau $n!$ Blätter, die den möglichen Permutationen der Eingabe entsprechen.

Betrachte beliebigen, vergleichsbasierten Sortieralgorithmus A und seinen Entscheidungsbaum T_A.

Worst Case # Vergleiche von A:

$$C_A(n) = \max_{\substack{v \text{ Blatt} \\ \text{von } T_A}} h(v) = h(T_A)$$

↑ Schicht, in dem Blatt v liegt.

Average Case # Vergleiche von A:

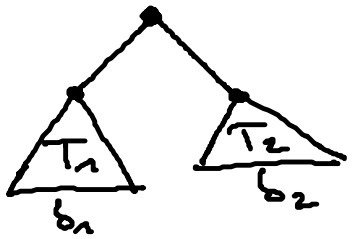
$$\overline{C}_A(n) = \frac{1}{n!} \cdot \underbrace{\sum_{v \text{ Blatt}} h(v)}_{=: H(T_A) \text{ „Blätterhöhensumme“}} = \frac{H(T_A)}{n!}$$

Lemma 1: Sei T binärer Baum mit b Blättern. Dann gilt $h(T) \geq \log b$.

Beweis: Induktion über $h(T)$:

$h(T) = 0 \Rightarrow$ Baum besteht nur aus Wurzel
$\rightarrow b = 1$, $\log b = 0$

Induktionsschritt:



$h(T) = 1 + \max\{h(T_1), h(T_2)\}$

$\delta = \delta_1 + \delta_2$

Annahme (oBdA): $\delta_1 \geq \delta_2$

$$h(T) = 1 + \max\{h(T_1), h(T_2)\} \geq 1 + h(T_1)$$

$$\overset{\text{Ind.}}{\geq} 1 + \log \delta_1 = \log 2 + \log \delta_1 = \log(2 \cdot \delta_1)$$

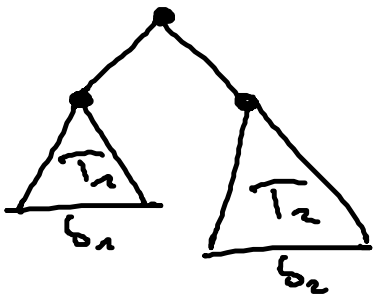
$$\geq \log(\delta_1 + \delta_2) = \log(\delta). \qquad \square$$

<u>Lemma 2:</u> Sei T binärer Baum mit δ Blättern. Dann gilt $H(T) \geq \delta \cdot \log \delta$

<u>Beweis:</u> Induktion nach $h(T)$.

$h(T) = 0 \Rightarrow H(T) = 0 = 1 \cdot \log 1.$
$\delta = 1$

Induktionsschritt:



$$H(T) = H(T_1) + H(T_2) + \delta$$

$$\overset{\text{Ind.}}{\geq} \delta_1 \cdot \log \delta_1 + \delta_2 \cdot \log \delta_2 + \delta_1 + \delta_2$$

$$= \delta_1 + \delta_2 + \delta_1 \cdot \log \delta_1 + (\delta - \delta_1) \cdot \log(\delta - \delta_1)$$

$$\geq \min_{\delta_1 \in \{0, \ldots, \delta\}} \underbrace{\delta_1 + (\delta - \delta_1)}_{\text{konstant}} + \delta_1 \cdot \log \delta_1 + (\delta - \delta_1) \cdot \log(\delta - \delta_1)$$

Kurvendiskussion:

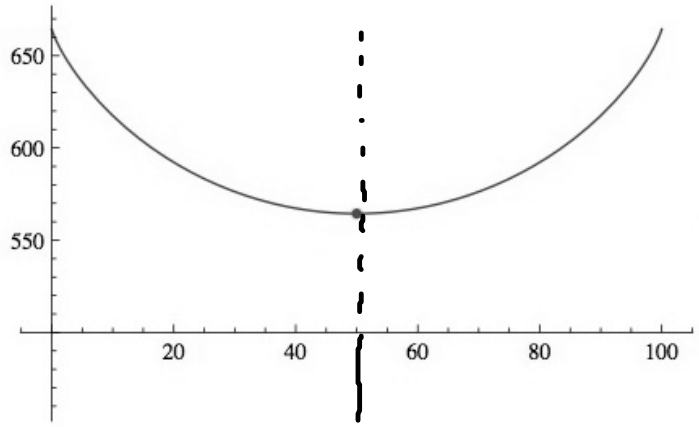
Input interpretation:

minimize	$x \log_2(x) + (100 - x) \log_2(100 - x)$

Global minimum:

$$\min\{x \log_2(x) + (100 - x) \log_2(100 - x)\} = \frac{100 \log(50)}{\log(2)} \text{ at } x = 50$$

Plot:



(x from −3 to 103)

$$= 6 + 2 \cdot \frac{6}{2} \cdot \log \frac{6}{2}$$

$$= 6 + 6 \cdot (\log 6 - \underbrace{\log 2}_{=1}) = 6 \cdot \log 6. \quad \square$$

Frage: Wie groß ist $\log 6$ für $6 = n!$

$$\log n! = \log(\underbrace{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 5 \cdot 4 \cdot 3 \cdot 2}_{\geq \left(\frac{n}{2}\right)^{n/2}})$$

$$\geq \frac{n}{2} \cdot \log \frac{n}{2} = \frac{n}{2} \cdot (\log n - 1)$$

$$\in \Omega(n \log n)$$

$$\overset{\text{Lem 1}}{\Rightarrow} C_A(n) \in \Omega(n \log n)$$

(alternative Abschätzung von $\log n!$ mit Hilfe der Stirling'schen Formel:

$$n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$)

$$\overline{C}_A(n) = \frac{1}{n!} \cdot H(T_A) \overset{\text{La2}}{\geq} \frac{1}{n!} \cdot n! \cdot \log n!$$

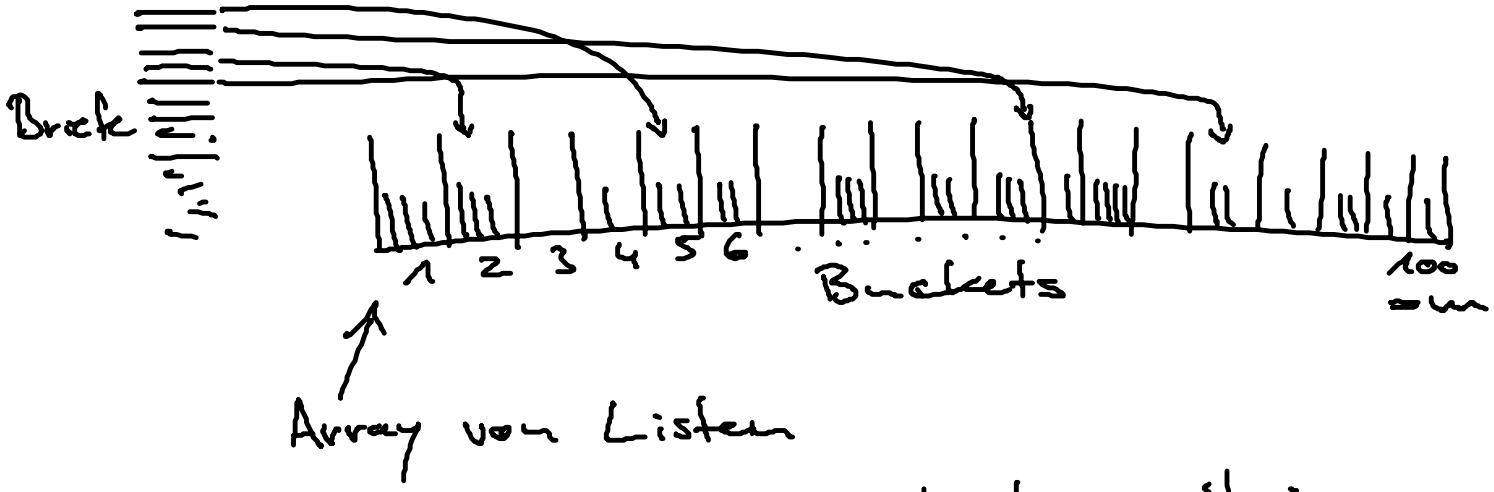
$$\in \Omega(n \log n)$$

Th. 1

$\Rightarrow$ MergeSort und HeapSort sind unter allen vergleichsbasierten Sortierverfahren asymptotisch optimal. QuickSort ist im Average Case asympt. optimal.

BucketSort:

Bsp: Briefträger sortiert Briefe einer Straße nach Hausnummern



Array von Listen

Am Ende werden diese Listen miteinander verkettet und ergeben so die sortierte Liste.

Aufwand: $O(n+m)$

Insbesondere: Falls $m \leq n$, $m \in O(n)$

$\Rightarrow$ Aufwand $O(n)$

Etwas formaler:

Fächer / Buckets $\leftrightarrow$ Array aus Listen / Queues

Hausnummern $\leftrightarrow$ Array-Indices

Briefe (vor Sortieren) $\leftrightarrow$ Liste (vor Sortieren)

Briefe (nach u) $\leftrightarrow$ Liste (nach u)

gegeben: n Objekte $a_1, \ldots, a_n$ (Liste mit Schlüsseln)

Schlüssel $s(a_1), \ldots, s(a_n) \in \{0, \ldots, m-1\}$

Algorithmus 1.1 (Einfaches Bucketsort)

1. Initialisiere ein Array mit m leeren Queues Q_i (Buckets), je eine für jeden Wert $i = 0, 1, \ldots, m-1$ und je einer Referenz (head bzw. tail) auf den Anfang und das Ende der Queue Q_i.

2. Durchlaufe L und füge das Objekt a_j entsprechend seines Schlüsselwertes in die Queue $Q_{s(a_j)}$ ein.

3. Konkateniere die Queues $Q_0, Q_1, \ldots, Q_{m-1}$ über die head- und tail-Referenzen zu einer Liste L und gebe L zurück.

Beispiel:

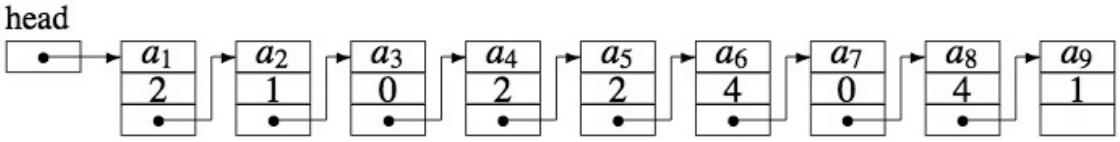


Abbildung 1.1: Liste L der zu sortierenden Elemente

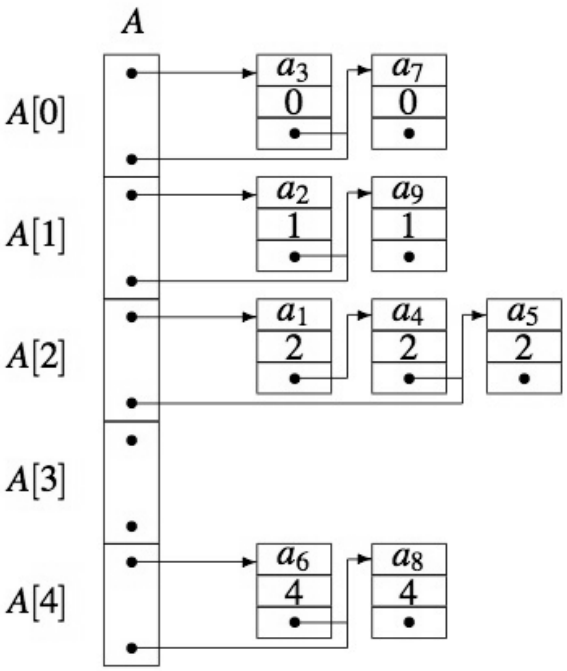


Abbildung 1.2: Queues nach Abarbeitung der Liste

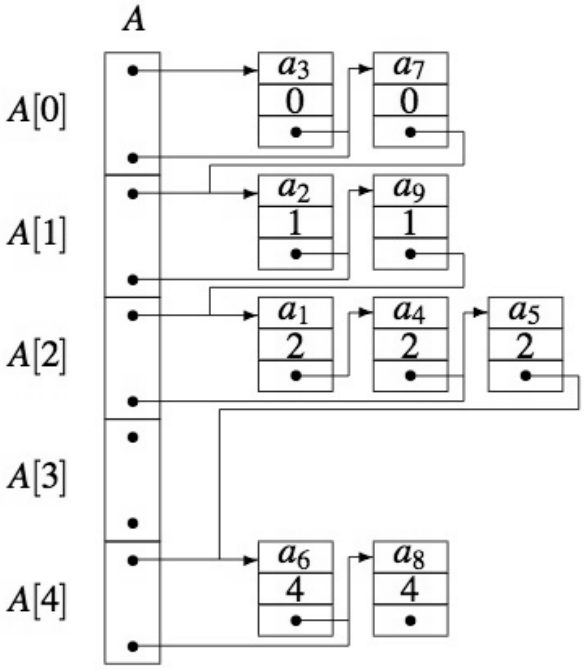


Abbildung 1.3: Queues nach Konkatenation der Einzel

Implementierung der Konkatenierung:

```
k = 0;
while ((A[k].head == NULL) && (k < m))
        k++;
// k is now the first nonempty list in A, if there is one
i = k + 1;
while (i < m) { // while-loop: O(m)
        // at this point k is the last list we have already concatenated
        // we will now look for the next nonempty list after k

        while ((A[i].head == NULL) && (i < m))
                i++;
        if (i==m)
                break;
        // if (i==m), we have iterated through all nonempty lists
        // in A and are finished

        // we have found a new nonempty list, concatenate it to k
        A[k].tail.setNext(A[i].head);
        // and prepare k for the next iteration
        k = i;
        i++;
} // endwhile
```

→ Aufwand für Konkatenieren ist $O(m)$.

Erweiterung von Bucket Sort zum Sortieren von Strings

Lexikographische Ordnung:

Hall $<_{lex}$ Hallo

Arbeit $>_{lex}$ Album

Def: A, B seien Strings über einem Alphabet (Zeichenmenge) S.
S sei linear geordnet, d.h. $s,t \in S$,
$s \neq t \Rightarrow s < t$ oder $t < s$

Dann ist $A <_{lex} B$ wenn einer der folgenden Fälle eintritt:

1) A ist Präfix von B $\quad A = a_1 a_2 \dots a_p$
$B = b_1 b_2 \dots b_q$
$p < q \wedge a_i = b_i \; \forall i = 1, \dots, p$

2) A und B unterscheiden sich an mind. einer Stelle und an der ersten solchen Stelle ℓ gilt $a_\ell < b_\ell$.

Zunächst Spezialfall:
- Strings gleicher Länge k
- $S = \{0, 1\}$

Bsp: $A_1 = 010$, $A_2 = 011$, $A_3 = 101$, $A_4 = 100$