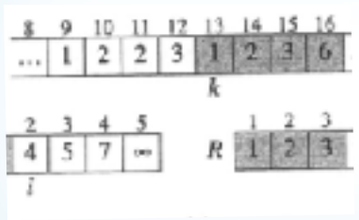


Kapitel 5.6: Quicksort

Algorithmen und Datenstrukturen
WS 2019/20

Prof. Dr. Sándor Fekete

5.6 Quicksort

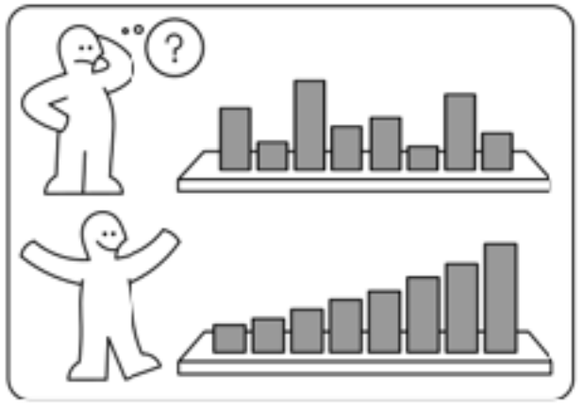
<u>Grundideen:</u>

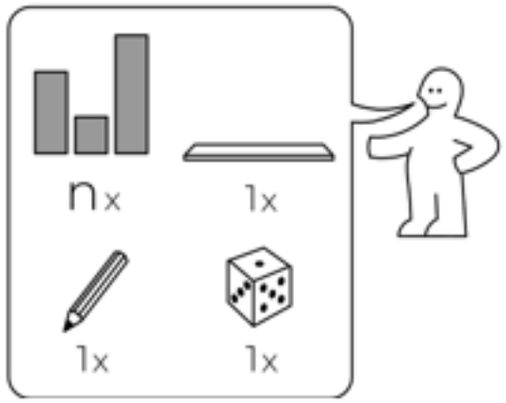
- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**
- **Rekursiv: Sortieren der Teilarrays**
- ***Kein* Merge-Schritt!**
- **Stattdessen Aufteilung der Teilarrays anhand eines “Pivot”-Elements, das die Menge in kleinere und größere Elemente teilt.**
- **Balance der Aufteilung vorher nicht absehbar.**

KVICK SÖRT

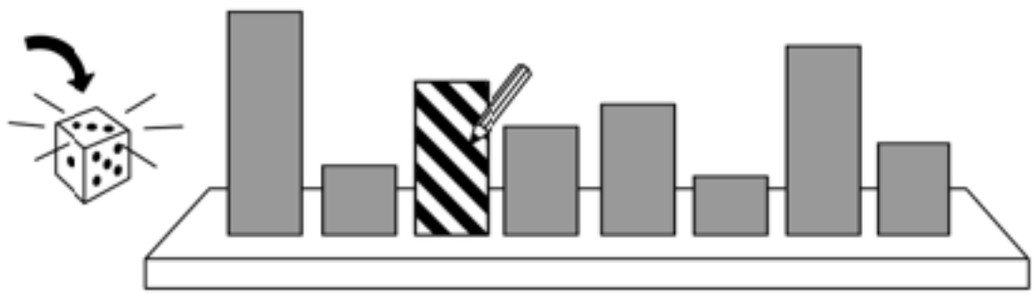
idea-instructions.com/quick-sort/
v1.1, CC by-nc-sa 4.0

IDEA

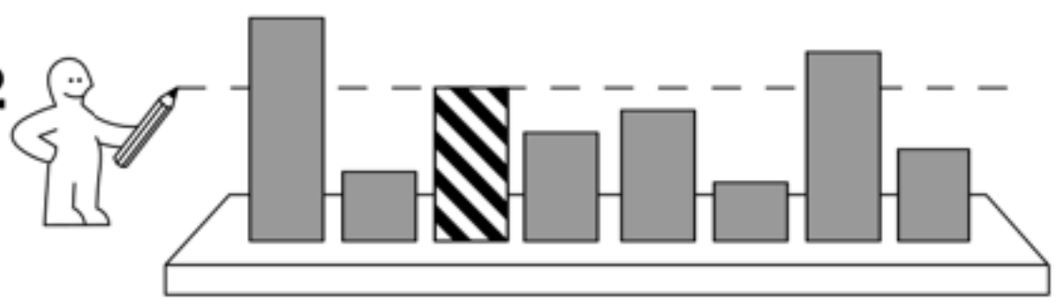




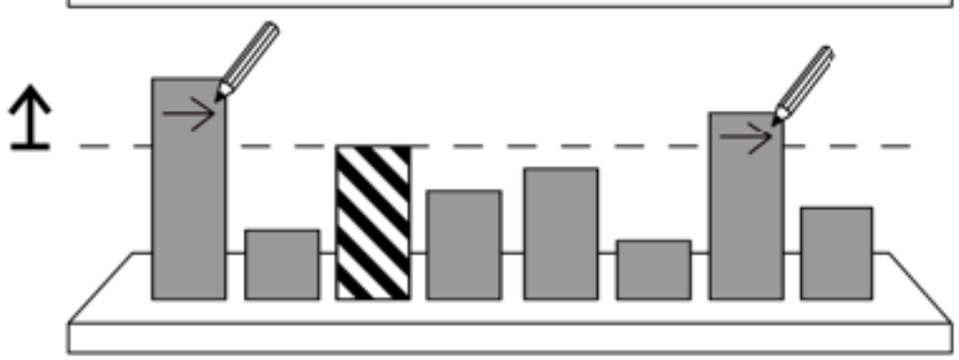
1



2

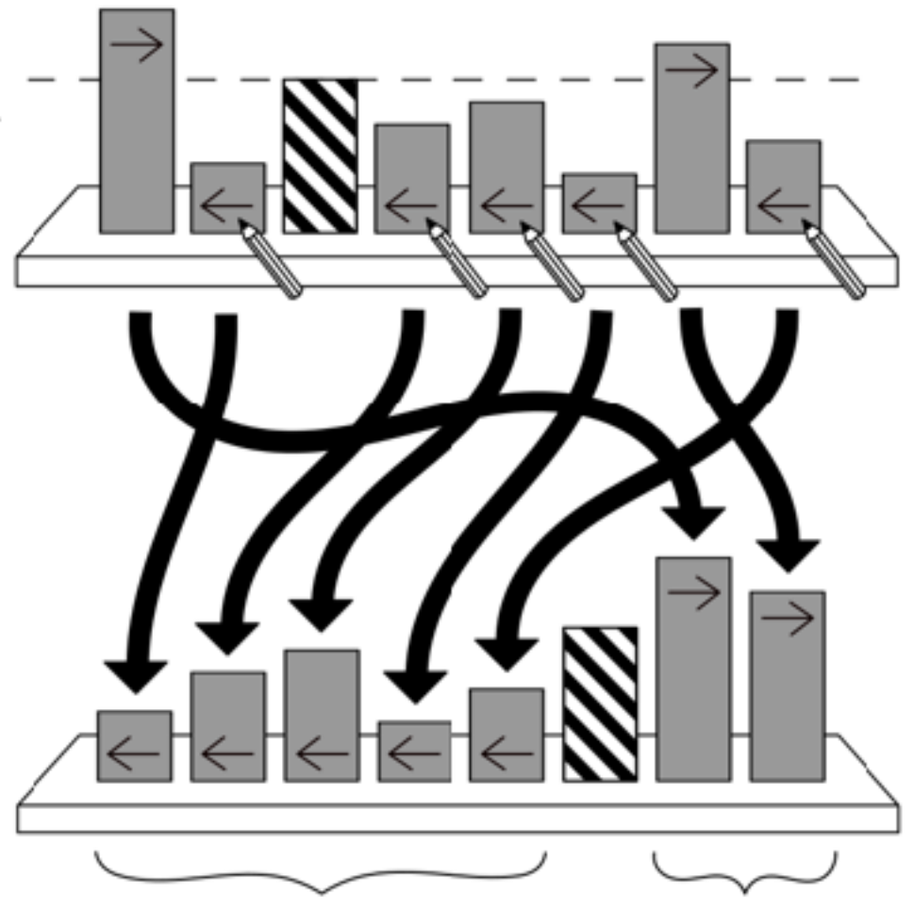


3

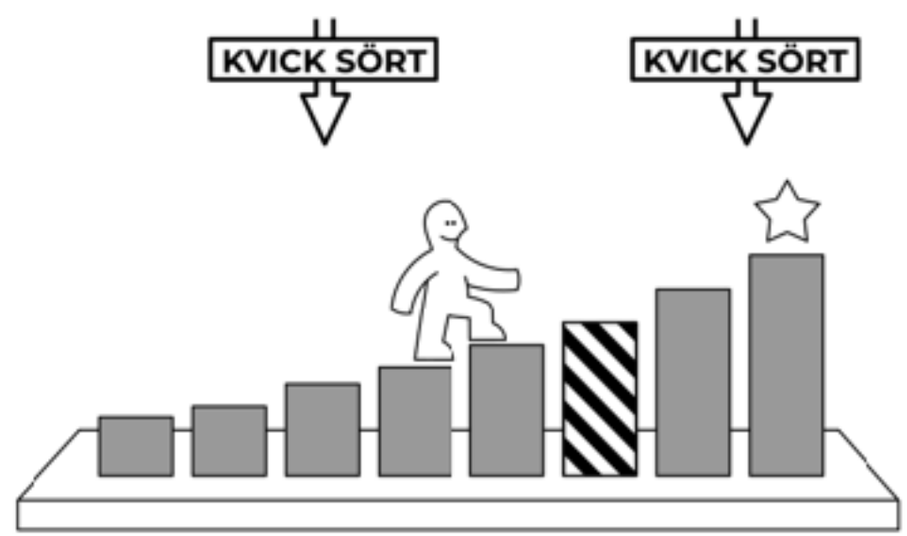


4

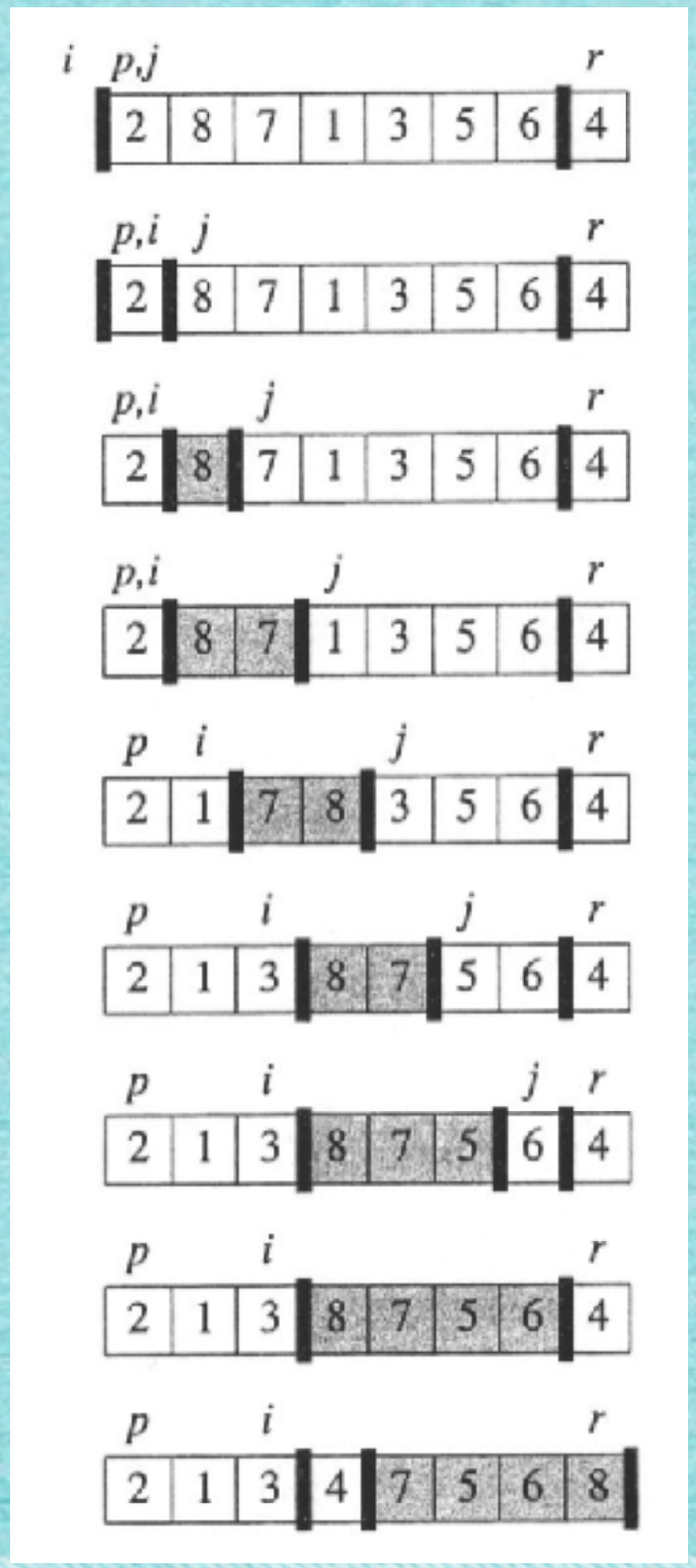
5



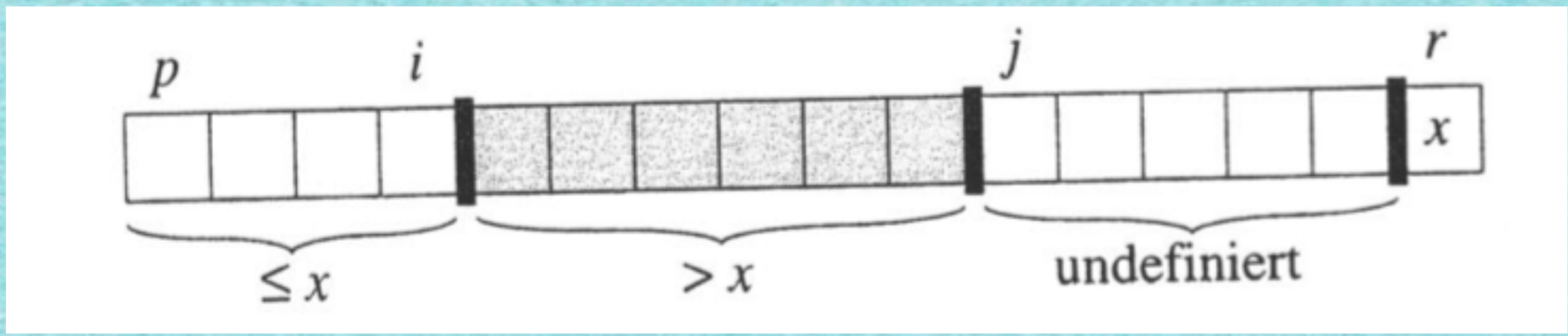
6



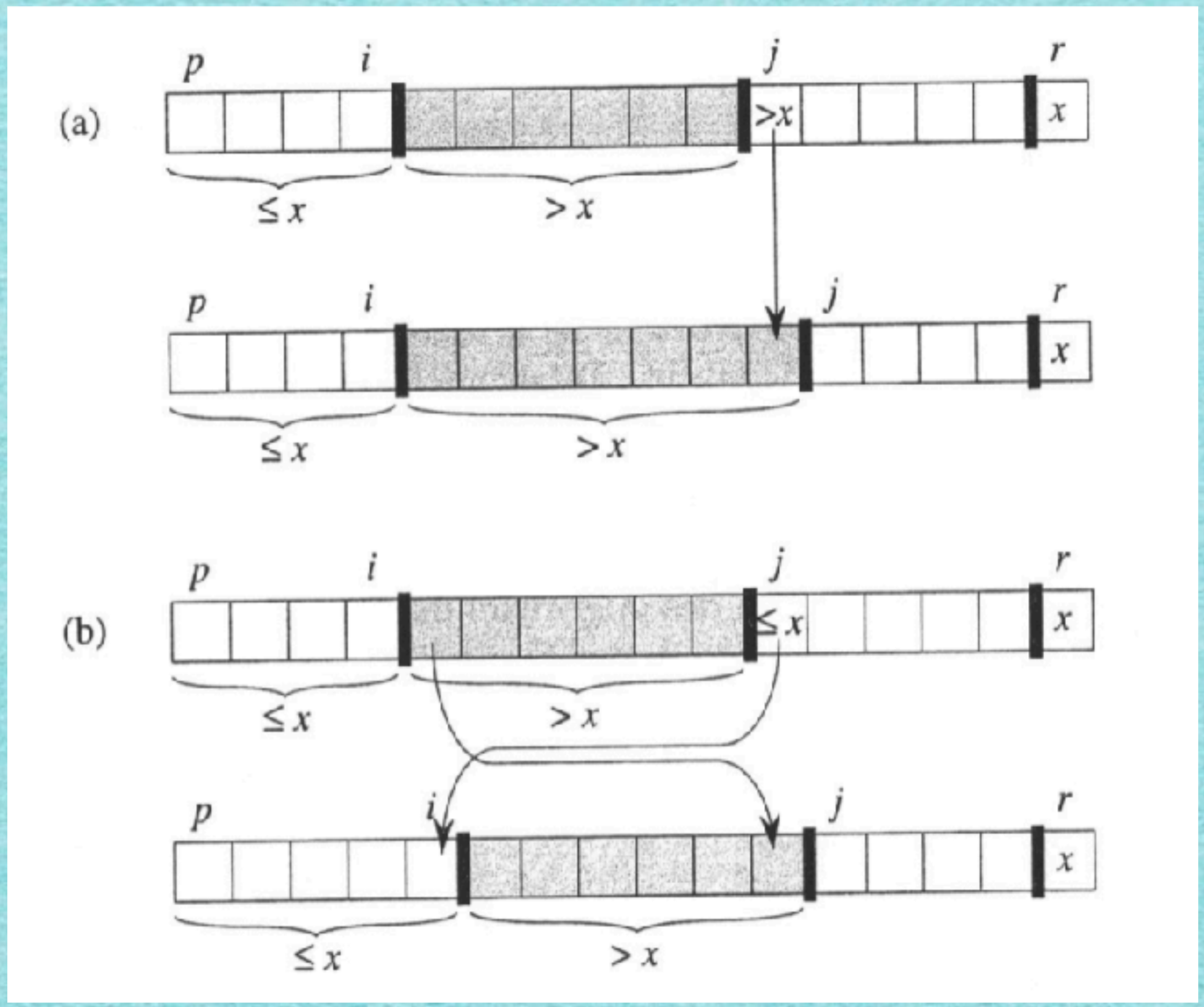
5.61 Ablauf Quicksort



5.61 Ablauf Quicksort



5.61 Ablauf Quicksort



5.6.2 Algorithmische Beschreibung

Algorithmus 5.14

<u>INPUT:</u> Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

<u>OUTPUT:</u> Sortierter Subarray

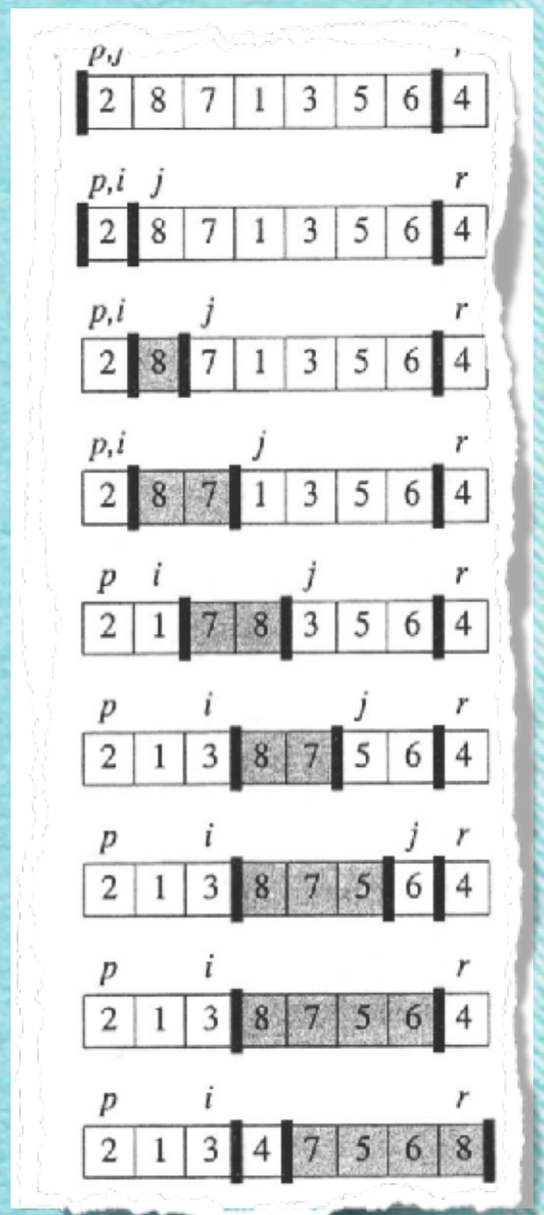
```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

Subroutine 5.1.5

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
        do if A[j] ≤ x
                then i ← i + 1
                        vertausche A[i] ↔ A[j]
vertausche A[i + 1] ↔ A[r]
return i + 1
```



5.6.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

<u>Unterscheidung:</u>

- **Worst Case**
- **Best Case**
- **Average Case**

5.6.3 Quicksort: Laufzeit

Bester Fall: Pivot liegt genau in der Mitte, d.h. nach PARTITION haben beide Teilarrays i.W. die Länge n/2.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

Man sieht, z.B. mit dem Mastertheorem:

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

5.6.3 Quicksort: Laufzeit

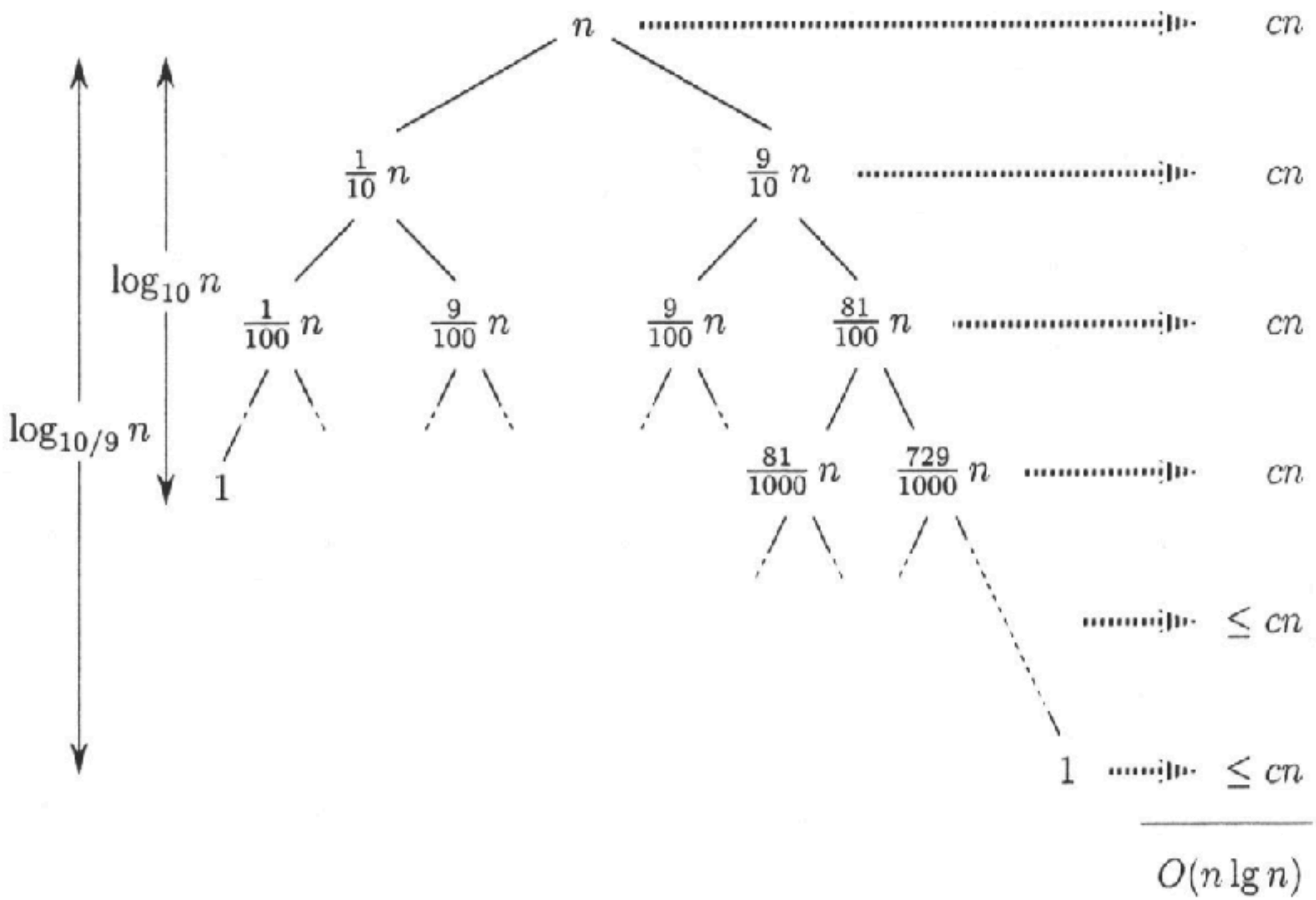
Das funktioniert auch noch, wenn PARTITION ein *annähernd* balanciertes Ergebnis liefert:

$$T(n) \leq T(9n/10) + T(n/10) + cn$$

Man sieht wieder mit dem Mastertheorem:

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

5.6.3 Quicksort: Laufzeit



5.6.3 Quicksort: Laufzeit

<u>Satz 5.13 (Durchschnittliche Laufzeit von Quicksort)</u>
Für einen n-elementigen Array A hat Quicksort eine erwartete Laufzeit von O(n log n).

5.6.3 Quicksort: Laufzeit

Warum interessiert das?

*** 1987**

Sebastian Wild

*** 1981**

Zlatan Ibrahimovic

5.6.3 Quicksort: Laufzeit

Warum interessiert das?

*** 1987**

Sebastian Wild

Average Case Analysis of Java 7's Dual Pivot Quicksort*

Sebastian Wild and Markus E. Nebel

Fachbereich Informatik, Technische Universität Kaiserslautern
{s_wild,nebel}@cs.uni-kl.de

Abstract. Recently, a new Quicksort variant due to Yaroslavskiy was chosen as standard sorting method for Oracle's Java 7 runtime library. The decision for the change was based on empirical studies showing that on average, the new algorithm is faster than the formerly used classic Quicksort. Surprisingly, the improvement was achieved by using a dual pivot approach, an idea that was considered not promising by several theoretical studies in the past. In this paper, we identify the reason for this unexpected success. Moreover, we present the first precise average case analysis of the new algorithm showing e.g. that a random permutation of length n is sorted using $1.9n \ln n - 2.46n + \mathcal{O}(\ln n)$ key comparisons and $0.6n \ln n + 0.08n + \mathcal{O}(\ln n)$ swaps.

1 Introduction

Due to its efficiency in the average, Quicksort has been used for decades as general purpose sorting method in many domains, e.g. in the C and Java standard libraries or as UNIX's system sort. Since its publication in the early 1960s by Hoare [1], classic Quicksort (Algorithm 1) has been intensively studied and many modifications were suggested to improve it even further, one of them being the following: Instead of partitioning the input file into two subfiles separated by a single pivot, we can create s partitions out of $s-1$ pivots.

Sedgewick considered the case $s = 3$ in his PhD thesis [2]. He proposed and analyzed the implementation given in Algorithm 2. However, this dual pivot Quicksort variant turns out to be clearly inferior to the much simpler classic algorithm. Later, Hennequin studied the comparison costs for any constant s in his PhD thesis [3], but even for arbitrary $s \geq 3$, he found no improvements that would compensate for the much more complicated partitioning step.[1] These negative results may have discouraged further research along these lines.

Recently, however, Yaroslavskiy proposed the new dual pivot Quicksort implementation as given in Algorithm 3 at the Java core library mailing list[2]. He

* This research was supported by DFG grant NE 1379/3-1.

[1] When s depends on n, we basically get the Samplesort algorithm from [4]. [5], [6] or [7] show that Samplesort can beat Quicksort if hardware features are exploited. [8] even shows that Samplesort is asymptotically optimal with respect to comparisons. Yet, due to its inherent intricacies, it has not been used much in practice.

[2] The discussion is archived at http://permalink.gmane.org/gmane.comp.java.openjdk.core-libs.devel/2628

L. Epstein and P. Bonagaura (Eds.): ESA 2012, LNCS 7501, pp. 826–838, 2012
© Springer-Verlag Berlin Heidelberg 2012

5.6.3 Quicksort: Laufzeit

Warum interessiert das?

*** 1987**

Sebastian Wild

	Comparisons
Classic Quicksort (Algorithm 1)	$2(n+1)\mathcal{H}_{n+1} - \frac{8}{3}(n+1)$ $\approx 2n\ln n - 1.51n + \mathcal{O}(\ln n)$
Sedgewick (Algorithm 2)	$\frac{32}{15}(n+1)\mathcal{H}_{n+1} - \frac{856}{225}(n+1) + \frac{3}{2}$ $\approx 2.13n\ln n - 2.57n + \mathcal{O}(\ln n)$
Yaroslavskiy (Algorithm 3)	$\frac{19}{10}(n+1)\mathcal{H}_{n+1} - \frac{711}{200}(n+1) + \frac{3}{2}$ $\approx 1.9n\ln n - 2.46n + \mathcal{O}(\ln n)$

“Best Paper Award”

Mehr an der Tafel!

s.fekete@tu-bs.de