ETH zürich

Übungslektion 7 – Suchen und Sortieren

Informatik II

31. März 2026

Willkommen!

Polybox

Passwort: jschul

Personal Website

`https://jschultev.github.io/personal_website/`

Heutiges Programm

Recap

Verbesserter Insertion Sort

Asymptotische Laufzeit Rekursiver Funktionen

Gruppenübung

Divide & Conquer Sortieralgorithmen

Prüfungsaufgaben

Hausaufgaben

1. Recap

Asymptotische Laufzeit

A: $\Theta(1)$	**B:** $\Theta(\log n)$	**C:** $\Theta(\log^2 n)$	**D:** $\Theta(\log \log n)$
E: $\Theta(n)$	**F:** $\Theta(n^2)$	**G:** $\Theta(n^3)$	**H:** $\Theta(n^4)$
I: $\Theta(n \log n)$	**K:** $\Theta(n^2 \log n)$	**L:** $\Theta(n \log^2 n)$	**M:** $\Theta(n^2 \log^2 n)$
N: $\Theta(n^{\log_2 3})$	**O:** $\Theta(\sqrt{n})$	**P:** $\Theta(\sqrt{n^3})$	**Q:** $\Theta(\sqrt{\log n})$
R: $\Theta(\sqrt{n \log n})$	**S:** $\Theta(\sqrt[3]{2^n})$	**T:** $\Theta(2^n)$	**U:** $\Theta(3^n)$
V: $\Theta(n!)$	**W:** $\Theta(n^n)$	**X:** N/A	

	$f(n)$	$f \in \ldots$		$f(n)$	$f \in \ldots$
0.	$2n$	E			
1.	20^{23}	[redacted]	2.	$\log(5^n)$	[redacted]
3.	$n^2 + n \cdot n^{1/2}$	[redacted]	4.	$\log(n^{2023})$	[redacted]
5.	$\sum_{i=1}^{n} n \cdot i$	[redacted]	6.	$4n^2 + 2^n$	[redacted]

Asymptotische Laufzeit

(1.c.1)

Wenn Algorithmus A asymptotische Laufzeit $O(n)$ hat und Algorithmus B asymptotische Laufzeit $O(n^3)$, dann muss A asymptotisch schneller sein als B. / *If algorithm A has an asymptotic running time of $O(n)$ and algorithm B has asymptotic running time of $O(n^3)$ then A must be asymptotically faster than B.*

True

False

(1.c.2)

Wenn ein Problem Komplexität $\Omega(n^2)$ hat, so kann es keinen Algorithmus mit Laufzeit $O(n \log n)$ für dieses Problem geben. / *If a problem has complexity $\Omega(n^2)$, there cannot be an algorithm with running time $O(n \log n)$ for this problem.*

True

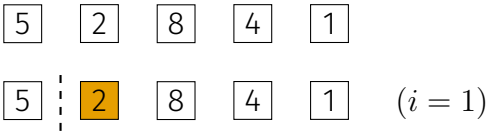
False

2. Verbesserter Insertion Sort

Letztes Mal: Insertion Sort

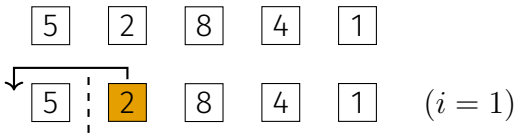
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



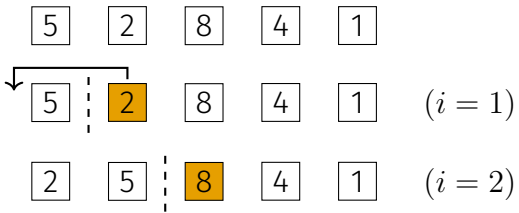
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



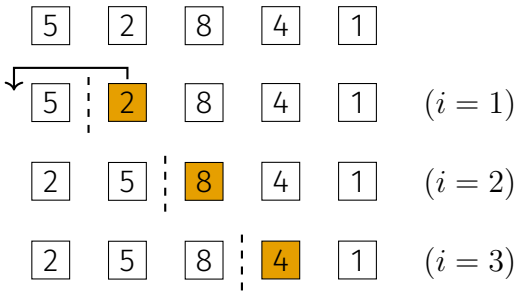
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



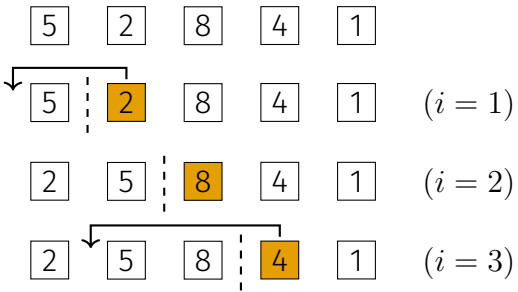
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



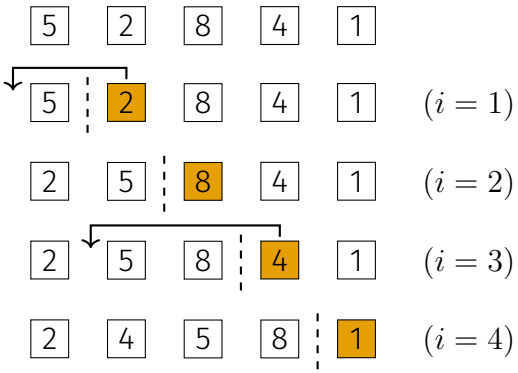
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



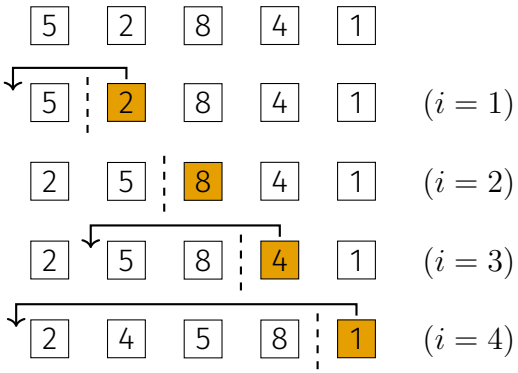
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



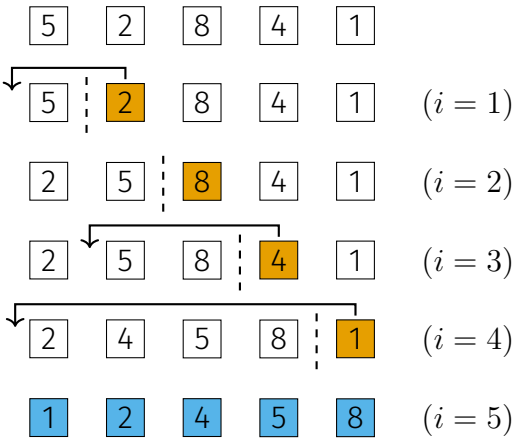
- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort



- **Schleifeninvariante**: Vor Iteration i sind Elemente in `li[:i]` sortiert.
- Bei Iteration i, das i-te Element an der korrekten Position in die sortierte Teilliste `li[:i]` einfügen.
- Wiederholen bis das gesamte Array sortiert ist ($i = n$).

Letztes Mal: Insertion Sort

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

```
1 for i in range(1,len(A)):
2     j = i
3     while j>0 and A[j-1] > A[j]:
4         A[j], A[j-1] = A[j-1], A[j]
5         j = j-1
```

- Anzahl Vergleiche im schlechtesten Fall? [redacted]
- Anzahl Vertauschungen im schlechtesten Fall? [redacted]
- **Frage**: Wie können wir die Anzahl Vergleiche für den schlechtesten Fall verbessern?

[redacted]

Recap - Binäre Suche

In einer **sortierten Liste** können Elemente binär gesucht werden:

```
def bin_search(A, left, right, val):
    if right < left:
        return None
    else:
        mid = (left + right) // 2
        if A[mid] == val:
            return mid
        elif val < A[mid]:
            return bin_search(A, left, mid - 1, val)
        else:
            return bin_search(A, mid + 1, right, val)
```

Binärer Insertion Sort

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

```
1         for i in range(1,len(A)):
2           x = A[i]
3           p = BinarySearchIndex(A,0,i-1,x)
4           for j in range(i-1,p-1,-1):
5               A[j+1] = A[j]
6           A[p] = x
```

- Anzahl Vergleiche im schlechtesten Fall? [redacted]
- Anzahl Kopiervorgänge im schlechtesten Fall? [redacted]

3. Asymptotische Laufzeit Rekursiver Funktionen

Übung 1

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ-Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 #pre: n is a positive integer
2 def g(n):
3     count = 0
4     while n // (2 ** count) >= 1:
5         f()
6         count += 1
```

Übung 2

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ-Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2   def g(n):
3     if n >= 1:
4         f()
5         g(n // 2)
```

Übung 3

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ-Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2   def g(n):
3     if n >= 1:
4         for i in range(n):
5             f()
6         g(n // 2)
```

Übung 4

Geben Sie die Anzahl Aufrufe der Funktion f abhängig von n in der Θ-Notation für den untenstehenden Beispiel an. Kürzen Sie ihr Ergebnis soweit wie möglich. Geben Sie eine kurze Begründung.

```
1 # pre: n is a positive integer
2   def g(n):
3     if n >= 1:
4         for i in range(n):
5             f()
6         g(n // 2)
7         g(n // 2)
```

4. Gruppenübung

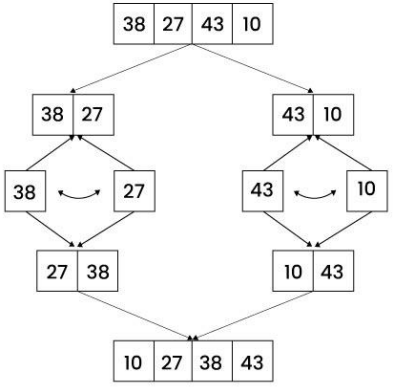
Gruppenübung: Timsort

https://expert.ethz.ch/enrolled/SS26/mavt2/codeExamples

Timsort, eine Kombination aus Insertion Sort und Merge Sort, ist der Standard-Sortieralgorithmus in Python. Mehr dazu in der Aufgabenbeschreibung auf Code Expert.

5. Divide & Conquer Sortieralgorithmen

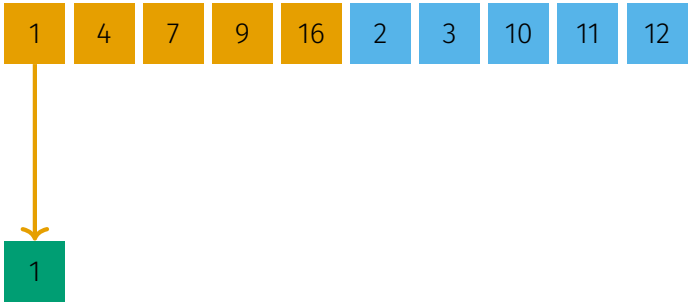
Merge Sort



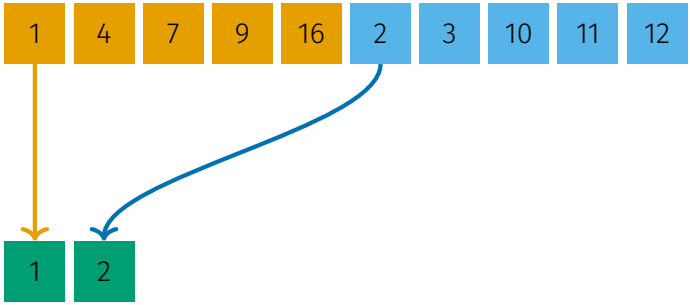
Merge Sort

Merge

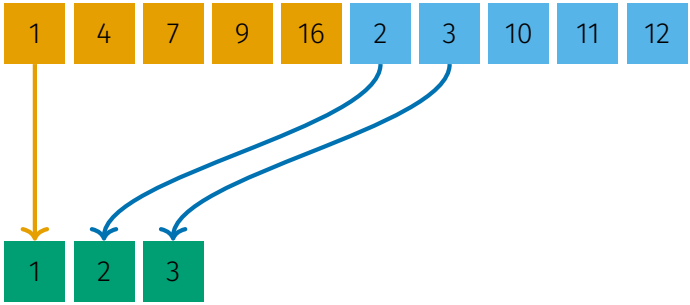
Merge



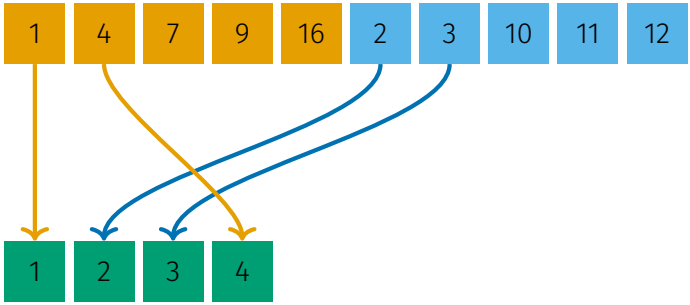
Merge



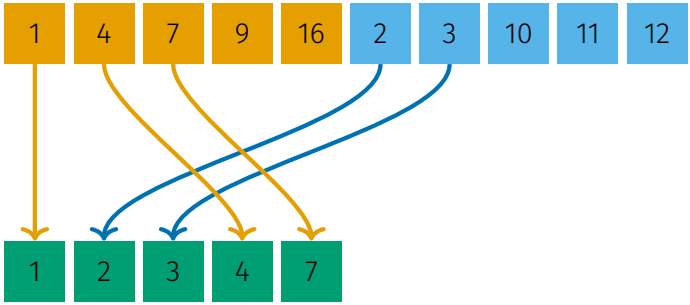
Merge



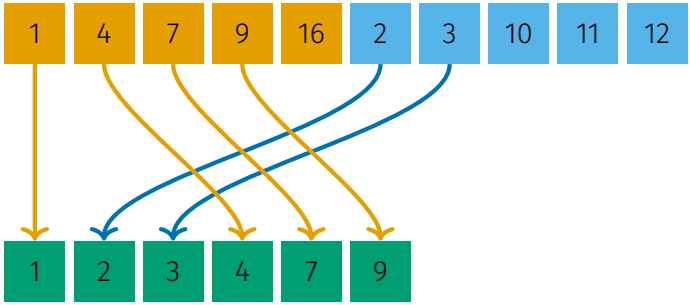
Merge



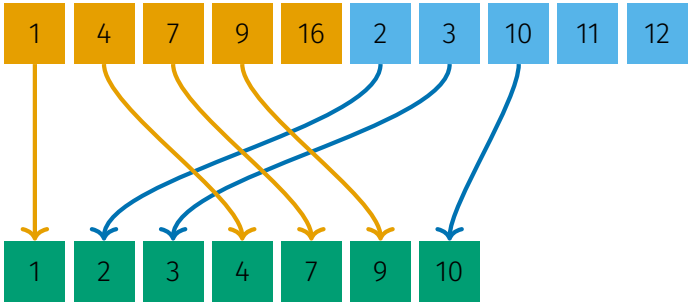
Merge



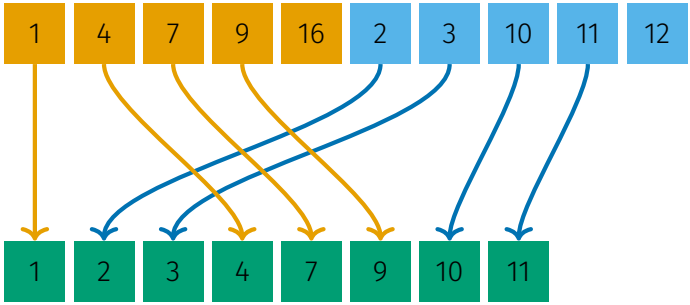
Merge



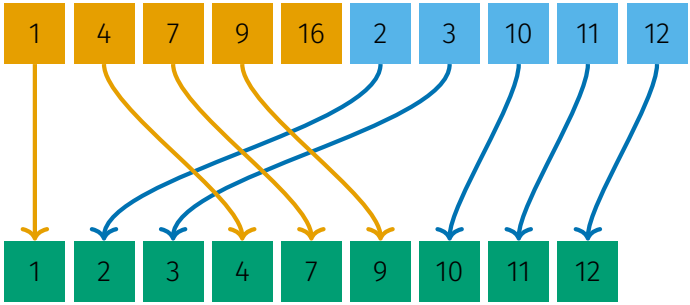
Merge



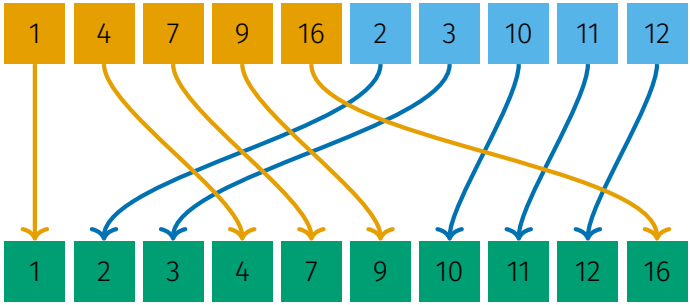
Merge



Merge



Merge



Algorithmus `merge`

```
 1  def merge(a1, a2):
 2    b, i, j = [], 0, 0
 3    while i < len(a1) and j < len(a2):
 4      if a1[i] < a2[j]:
 5        b.append(a1[i])
 6        i += 1
 7      else:
 8        b.append(a2[j])
 9        j += 1
10    b += a1[i:]
11    b += a2[j:]
12    return b
```

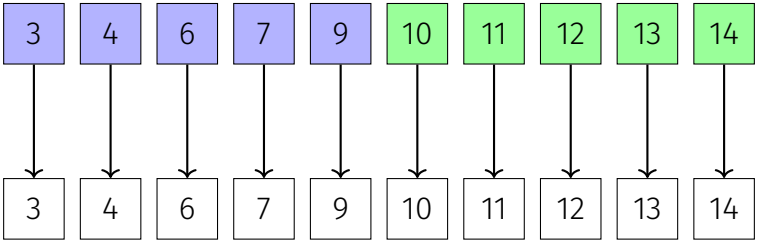
Algorithmus `merge_sort`

```
1 def merge_sort(a):
2   if len(a) <= 1:
3     return a
4   else:
5     sorted_a1 = merge_sort(a[:len(a) // 2])
6     sorted_a2 = merge_sort(a[len(a) // 2:])
7     return merge(sorted_a1, sorted_a2)
```

Mergesort hat immer $\mathcal{O}(n \log(n))$ vergleiche. **IMMER!**

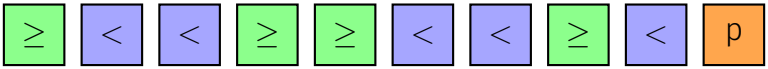
Recap - Quick Sort

- **Was ist der Nachteil von Merge Sort?**
  Benötigt zusätzlich $\Theta(n)$ Speicherplatz für das Verschmelzen.
- **Wie könnte man das Verschmelzen einsparen?**
  Jedes Element im linken Teil kleiner ist als alle Elemente im rechten Teil.



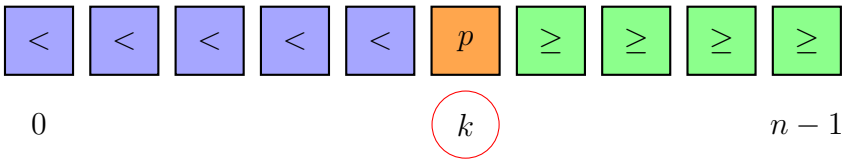
- **Wie erreicht man das?**
  Pivotieren und Aufteilen!

Recap - Quick Sort: Pivot



1. Wähle ein (beliebiges) Element **p** als Pivotelement
2. Teile a in zwei Teile auf:
   - einen Teil **L** der Elemente mit $a[i] < p$
   - und einen Teil **R** der Elemente mit $a[i] \geq p$.

Recap - Quick Sort: Pivot



1. Wähle ein (beliebiges) Element **p** als Pivotelement
2. Teile a in zwei Teile auf:
   - einen Teil **L** der Elemente mit $a[i] < p$
   - und einen Teil **R** der Elemente mit $a[i] \geq p$.
3. Quick Sort: Rekursion auf Teilen **L** und **R**

6. Prüfungsaufgaben

Sorting Algorithm

Geben Sie für die folgenden Algorithmen jeweils an, welche der folgenden Strategien dem Algorithmus zugrunde liegt.

For the following algorithms, indicate which of the following strategies underlies the algorithm.

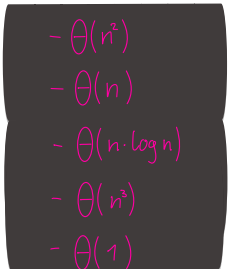
- Divide and Conquer (DC)
- Greedy (G)
- Dynamische Programmierung / Dynamic Programming (DP)
- Brute Force (BF)

Merge Sort / *Merge Sort*

Lineare Suche in einem Array / *Linear search in an array*

Sorting Algorithm

Nehmen Sie an, dass Sie in jedem Schritt von Quick Sort ein Pivot-Element finden, das Ihre Daten in zwei Teile aufteilt mit Grössenverhältnis 1:2 (der eine Teil ist doppelt so gross wie der andere). Was ist dann die Laufzeit von Quick Sort?



Sorting Algorithm

Sie lassen Insertion Sort auf einem Array für mindestens 3 Iterationen laufen. Wie könnte das Array aussehen?

You let insertion sort run on an array for at least 3 iterations. How could the array look like?

1,2,3,4,5,6 4,5,2,6,3,1 4,5,6,3,2,1 1,2,3,6,5,4
2,4,6,1,3,5 1,3,5,2,4,6 6,5,4,3,2,1

Sorting Algorithm

Ordnen Sie den folgenden Aussagen die Sortieralgorithmen Merge Sort (M), Selection Sort (S) und Insertion Sort (I) zu. Nach 7 Schritten (d.h., Iterationen oder Merge-Schritten) ...

Assign the sorting algorithms merge sort (M), selection sort (S), and insertion sort (I) to the following invariants. After 4 steps (i.e., iterations or merge steps), ...

1. ... sind die ersten 7 Elemente sortiert. / *the first 7 elements are in sorted order.*
2. ... sind die ersten 8 Elemente sortiert. / *the first 8 elements are in sorted order.*
3. ... sind die 7 kleinsten Elemente sortiert an den ersten 7 Stellen. / *the 7 smallest elements are sorted at the first 7 positions.*

M,I,S M,S,I I,M,S I,S,M S,M,I S,I,M

Sorting Algorithm

(3.a) Was ist die asymptotische Komplexität von binärer Suche in einem sortierten Array im schlechtesten Fall? / *What is the worst-case complexity of binary search on a sorted array?*

O(logn) O(n) O(nlogn) O(n^2)

(3.b) Welche der folgende ist eine Vorbedingung für binäre Suche? / *Which of the following is a precondition for binary search?*

Input list is sorted in increasing order

Input list is not sorted

First half of the input list is sorted

Second half of the input list is sorted

Sorting Algorithm

Unter welcher Bedingung hätte Quicksort eine Laufzeit von $O(n^2)$? / *Under which condition would quicksort have a runtime of $O(n^2)$?*

[redacted] *Input list is sorted in decreasing order*

[redacted] *Input list is not sorted*

[redacted] *First half of the input list is sorted*

[redacted] *Second half of the input list is sorted*

Nehmen wir an, wir sortieren eine Liste mit Quicksort. Nach der ersten Partitionierung sieht die Liste folgendermassen aus: / *Assume that we are sorting a list using quicksort. After the first partitioning, the list looks like this:* 3, 6, 2, 8, 10, 13, 12, 11. Welche der folgenden Aussagen ist wahr? / *Which of the following is true?*

[redacted] *The pivot could be 8, but not 10*

[redacted] *The pivot could be 10, but not 8*

[redacted] *The pivot could be 8 or 10*

[redacted] *The pivot cannot be 8 nor 10*

Sorting Algorithm

Welche ist die asymptotische Komplexität von Quicksort im schlechtesten Fall? / *What is quicksort's worst-case asymptotic complexity?*

O(1) O(logn) O(n) O(nlogn) O(n^2)

(2.a.2)

Welche ist die asymptotische Komplexität des Sortierens eines Arrays mit Heapsort im schlechtesten Fall? / *What is the worst-case asymptotic complexity of sorting an array with heap sort?*

O(1) O(logn) O(n) O(nlogn) O(n^2)

Sorting Algorithm

Die Komplexität von Mergesort hängt nicht von der anfänglichen Reihenfolge des Arrays ab. / *Mergesort's complexity does not depend on the initial order of the array.*

(2.b.2)

Quicksort benötigt zusätzlichen Speicherplatz in der Grösse von $O(n)$. / *Quicksort needs $O(n)$ additional space.*

Rekursive Runtime Analysis

```
def g(n):
    f()
    f()
    if n >= 1:
        g(n - 2)
```

Asymptotische Anzahl Aufrufe von `f` / *Asymptotic number of calls to* `f`:

1 n $\log n$ $\sqrt{n}$ n^2 2^n $n \log n$ n^3

7. Hausaufgaben

Übung 6: Search and Sort

Auf `https://expert.ethz.ch/enrolled/SS26/mavt2/exercises`

Übung 6: Search and Sort codebo Einfach, Mittel, Schwer

- Eierwerfen (Probiert es, sonst Youtube)
- Vergleich von Sortieralgorithmen (ZF und Vorlesung helfen!)
- Verbesserter Insertion Sort (Vorlesung, Binäre Suche verstehen!)
- Invarianten der Suchalgorithmen (Versucht euch zu erinnern! Sonst slides)

KEINE HARDCODIERUNG

Fragen oder Anregungen?

Feedback

https://jschultev.github.io/personal_website/Feedback