Sortieralgorithmen

Theorie (L)

Inhaltsverzeichnis

1 Selection Sort **3**

2 Insertion Sort **4**

3 Mergesort **5**

Version vom 11. April 2025

1 Selection Sort

Beschreibung

Zuerst wird das kleinste Element im gesamten Array gesucht und mit dem Element an der ersten Stelle vertauscht, selbst wenn das kleinste Element schon an der ersten Stelle steht.

Danach sucht man in der Teilliste ab dem zweiten Element das kleinste Element und tauscht es mit dem Element an der zweiten Stelle.

Auf die gleiche Weise fahren wir fort, bis an der zweiletzten Position das richtige Element steht. Das letzte Element ist dann automatisch am richtigen Platz. Warum?

Beispiel

13	5	2	21	8	Vergleiche	Vertauschungen
2	5	13	21	8	4	1
2	5	13	21	8	3	1
2	5	8	21	13	2	1
2	5	8	(13	21	1	1
					10	4

Python-Implementierung

```python
def selectionsort(A):
    '''Sortiert das Array A aufsteigend/inplace'''
    n = len(A)
    # nach dem i-ten Durchlauf sind die Elemente an
    # den Positionen 0, 1, ..., i sortiert.
    for i in range(0, n-1):
        minpos = i # Positionskandidat
        # Bestimme den Index des kleinsten Elements
        # aus A[i], A[i+1], ..., A[n-1]:
        for j in range(i+1, n):
            if A[j] < A[minpos]:
                minpos = j
        # Vertausche A[i] mit dem kleinsten Element:
        A[minpos], A[i] = A[i], A[minpos]
```

Laufzeitanalyse

Vergleiche: $(n-1)+(n-2)+\cdots+1 = \dfrac{n(n-1)}{2}$

Vertauschungen: $n-1$

Mit gleichen Kosten $T(1) = c$ für Vergleiche und Vertauschungen erhalten wir:

$T(n) = c\left(\frac{1}{2}n^2 - \frac{1}{2}\right) + c(n-1) = c\frac{1}{2}n^2 + c\frac{1}{2}n - c \in O(n^2)$

2 Insertion Sort

Beschreibung

Die erste Zahl links ist ein sortiertes Array der Länge 1.

Dieses Array wird um das nächste rechts stehende Element x erweitert. Ist es grösser als sein linker Nachbar, ist dase erweiterte Array sortiert, Andernfalls wird das links stehende Element um eine Position nach rechts verschoben und das Element x wird an die Anfangsposition gesetzt.

Allgemein wird ein bereits sortiertes Array mit k Elementen um das $(k+1)$-te Element x erweitert. Dieses Element wird der Reihe nach mit den davor liegenden Elementen verglichen. Ist eines davon grösser als x, wird es um eine Position nach rechts verschoben. Andernfalls wird x an die freigewordene Stelle gesetzt und ist an seiner richtigen Position. So fahren wir fort, bis das letzte Elemente wie oben beschrieben einsortiert ist.

Beispiel

13	5	2	21	8	Vergleiche	Verschiebungen
$\underline{5}$	$\underline{13}$	2	21	8	1	1
$\underline{2}$	$\underline{5}$	$\underline{13}$	21	8	2	2
$\underline{2}$	$\underline{5}$	$\underline{13}$	$\underline{21}$	8	1	0
$\underline{2}$	$\underline{5}$	$\underline{8}$	$\underline{13}$	$\underline{21}$	3	2
					7	5

Python-Implementierung

```python
def insertionsort(A):
    '''Sortiert das Array A aufsteigend/inplace '''
    # Beginne mit A[1]
    for i in range(1, len(A)):
        x = A[i] # das einzusortierende Element
        j = i-1  # aktuelle Position von 'cand'
        # So lange die Elemente an den Positionen
        # j, j-1, ..., 0 grösser als x sind, schiebe
        # sie nach rechts:
        while(j >= 0 and A[j] > x):
            A[j+1] = A[j]
            j = j-1
        A[j+1] = x # Fülle die Lücke
```

Laufzeitanalyse (Worst Case)

Gegeben: Array mit n Elementen, das umgekehrt sortiert ist.

Vergleiche: $(n-1)+(n-2)+\cdots+1=\dfrac{n(n-1)}{2}$

Verschiebungen: $1+2+\cdots+(n-1)=\dfrac{n(n-1)}{2}$

Mit gleichen Kosten $T(1)=c$ für Vergleiche und Verschiebungen erhalten wir:

$T(n)=c\frac{1}{2}n^2-c\frac{1}{2}n+c\frac{1}{2}n^2-c\frac{1}{2}n=cn^2-cn\in O(n^2)$

Laufzeitanalyse (Best Case)

Gegeben: Array mit n Elementen, das bereits sortiert ist

Vergleiche: $1+1+\cdots+1=n-1$

Verschiebungen: $0+0+\cdots+0=0$

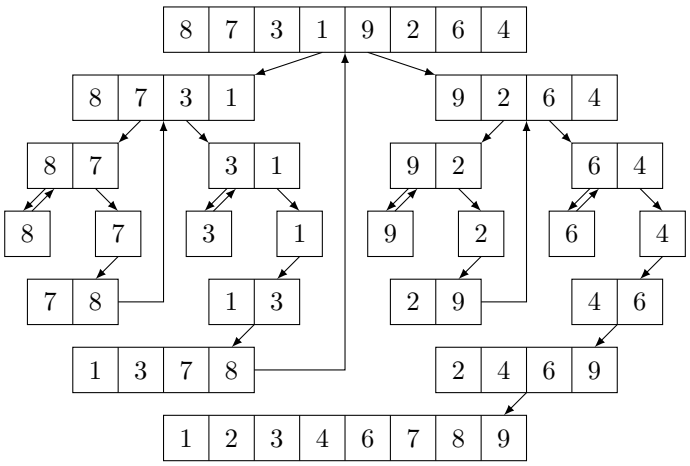
Insgesamt: $T(n)=c\cdot(n-1)\in O(n)$

3 Mergesort

Mergesort ist ein Sortieralgorithmus, der nach dem Prinzip *divide and conquer* arbeitet.

1. *Divide:* Zerlege ein Array mit n Elementen in zwei Arrays mit jeweils $n/2$ Elementen
2. *Conquer:* Wenn das resultierende Array aus weniger als zwei Elementen besteht, dann ist es sortiert. Ansonsten wende Mergesort rekursiv an.
3. *Combine:* Mische die zwei sortierten Teilarrays zu einem sortierten Array zusammen.

Beispiel



Python-Code für die Rekursion

```python
def mergesort_recursive(A):
    mergesort_helper(A, 0, len(A))

def mergesort_helper(A, left, right):
    if left < right-1:
        mid = (left + right) // 2
        mergesort_helper(A, left, mid)
        mergesort_helper(A, mid, right)
        merge(A, left, mid, right)
```

Python-Code für die Merge-Operation

```python
def merge(A, left, mid, right):
    L = A[left:mid] + [float('inf')]
    R = A[mid:right] + [float('inf')]
    i = 0
    j = 0
    for k in range(left, right):
        if L[i] <= R[j]:
            A[k] = L[i]
            i = i+1
        else:
            A[k] = R[j]
            j = j+1
```

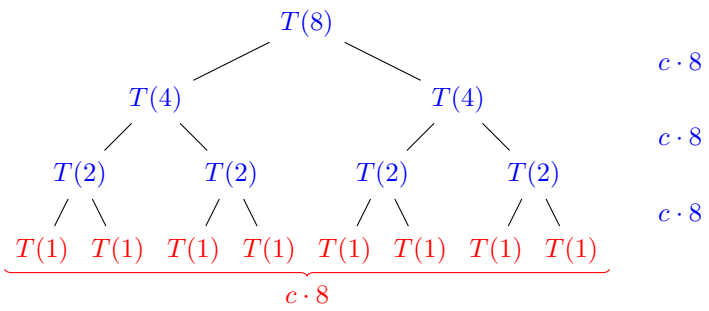
Die Analyse von Mergesort

Geht man bei der Berechnung davon aus, dass die Anzahl der zu sortierenden Elemente eine Zweierpotenz ist, So lässt sich der Berechnungsaufwand durch die *Rekursionsgleichung*

$$T(n) = T(n/2) + T(n/2) + c_1 \cdot 1 + c_2 \cdot n$$

ausdrücken. Die ersten zwei Summanden rechts besagen, dass die Aufgabe in zwei halb so grosse Teile zerlegt wird. Der dritte Summand stellt den Aufwand dar, die Mitte des ursprünglichen Arrays zu bestimmen und der letzte, die sortierten Teillisten wieder zusammenzufügen. Für ein Problem der Grösse 1 berechnen wir am Ende den konstanten Aufwand $T(1) = c_3$.

Zur Vereinfachung ersetzen wir nun alle Konstanten durch $c = \max\{c_1, c_2, c_3\}$ und ignorieren $c \cdot 1$, da $c \cdot 1 + c \cdot n \in O(n)$.



$T(8) = c \cdot 8 \cdot 3 + c \cdot 8$

oder allgemein:

$T(n) = c \cdot n \cdot \log_2 n + c \cdot n \in O(n \log n)$

Im Vergleich zu Quicksort hat Mergesort eine garantierte Laufzeit von $O(n \log_2 n)$.

Jedoch müssen beim Merge-Schritt, die Arrays vorgängig kopiert werden, was bedeutet, dass das Verfahren zusätzlichen Speicher in der Grösse des zu sortierenden Arrays benötigt.

Eine iterative Variante von Mergesort

```
def mergesort_iterative(A):
    n = len(A)
    i = 1
    while i < n:
        j = 0
        while j < n:
            left = j
            mid = j+i
            right = min(j+2*i, n) # len(A) ist nicht immer Zweierpotenz
            merge(A, left, mid, right)
            j = j + 2*i
        i = 2*i
```

Die Merge-Funktion ist dieselbe wie bei der rekursiven Version.

Beispiel

7	3	4	1	8	2	5
3	7	1	4	2	8	5
1	3	4	7	2	5	8
1	2	3	4	5	7	8