Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

Idee InsertionSort

- *Die ersten j-1 Elemente sind sortiert (zu Beginn j=2)*
- *Innerhalb eines Schleifendurchlaufs wird das j-te Element in die sortierte Folge eingefügt*
- *Am Ende ist die gesamte Folge sortiert*

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

Beispiel

8	15	3	14	7	6	18	19

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

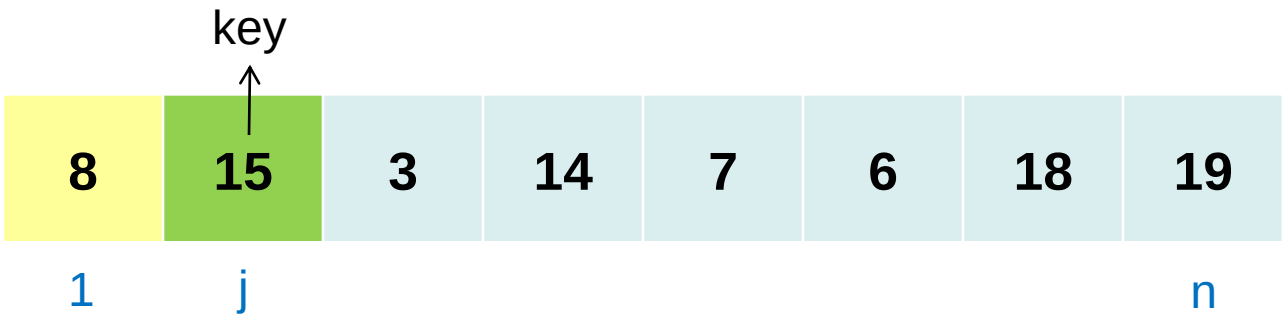
- Eingabegröße n
- length[A] = n

8	15	3	14	7	6	18	19
1	j						n

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

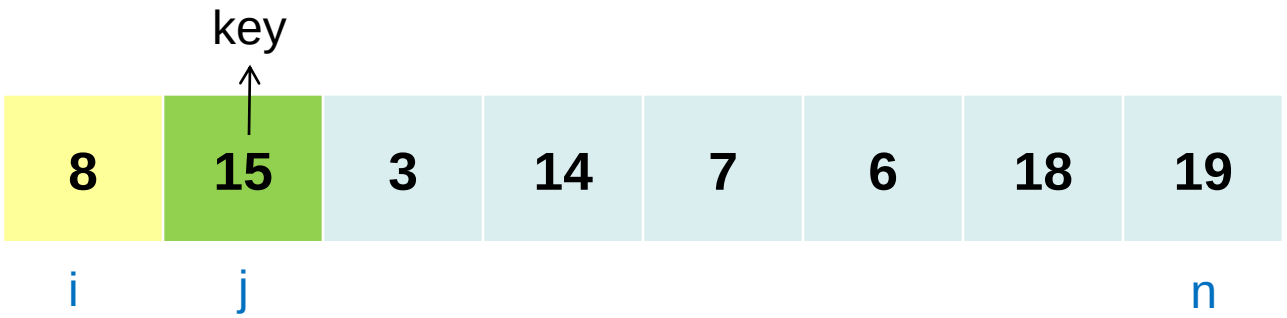
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

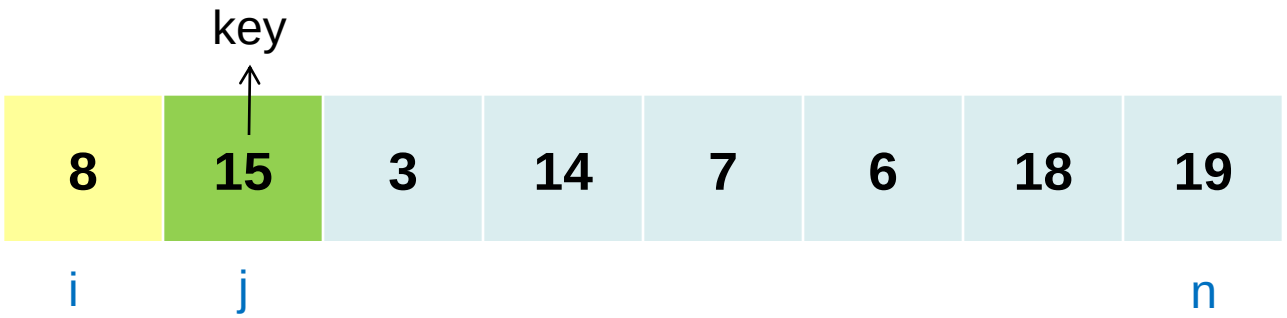
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

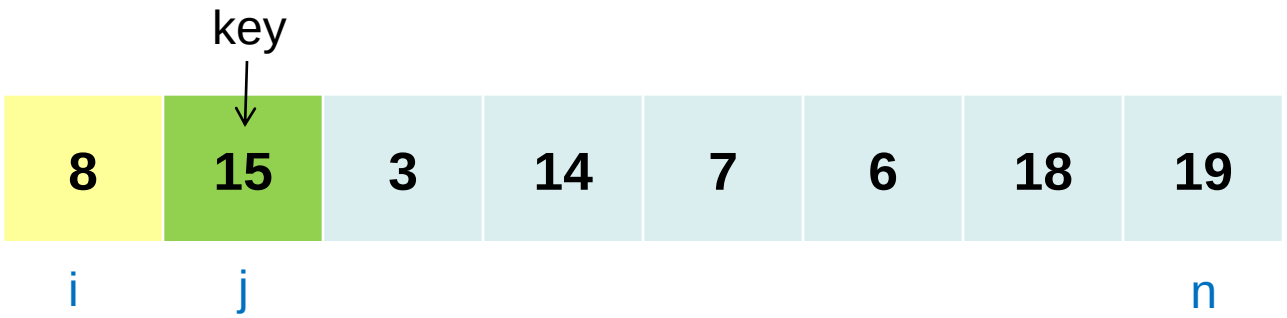
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

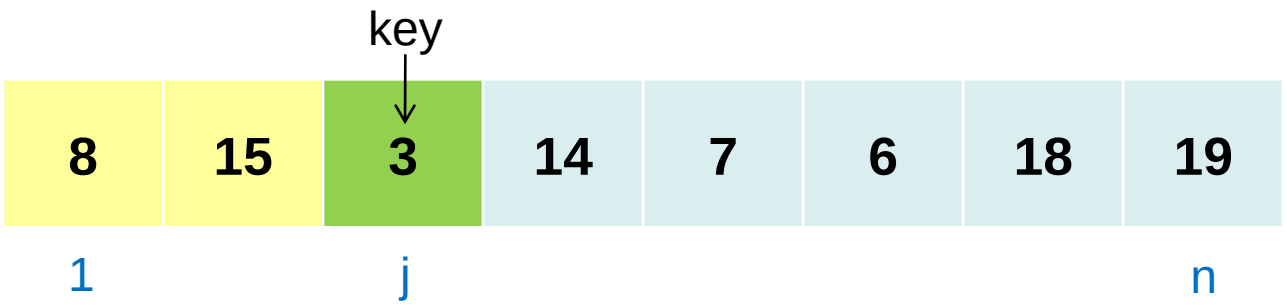
- Eingabegröße n
- length[A] = n

8	15	3	14	7	6	18	19
1		j					n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

key=3

8	15	3	14	7	6	18	19
1	i	j					n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

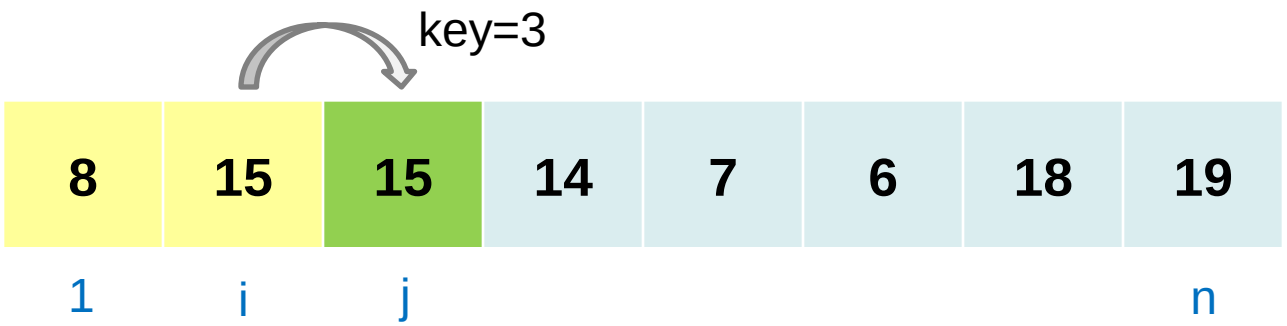
key=3

8	15	3	14	7	6	18	19
1	i	j					n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

key=3

8	15	15	14	7	6	18	19
i		j					n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

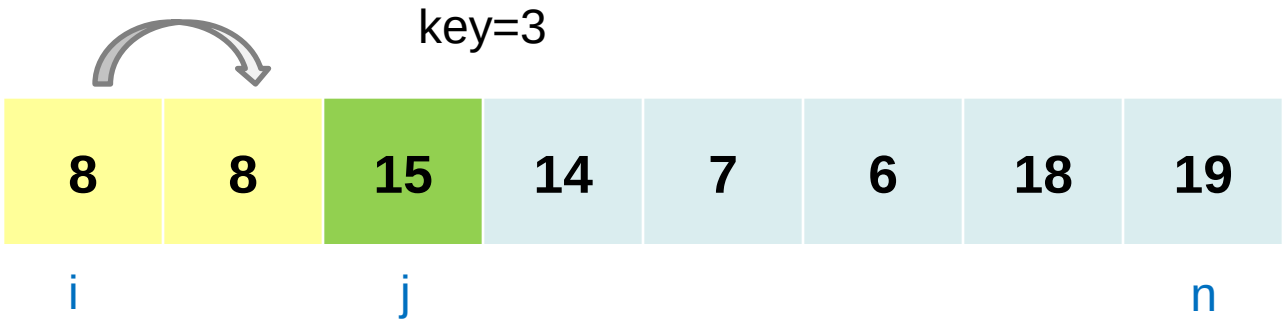
key=3

8	15	15	14	7	6	18	19
i		j					n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

key=3

8	8	15	14	7	6	18	19
i	1		j				n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

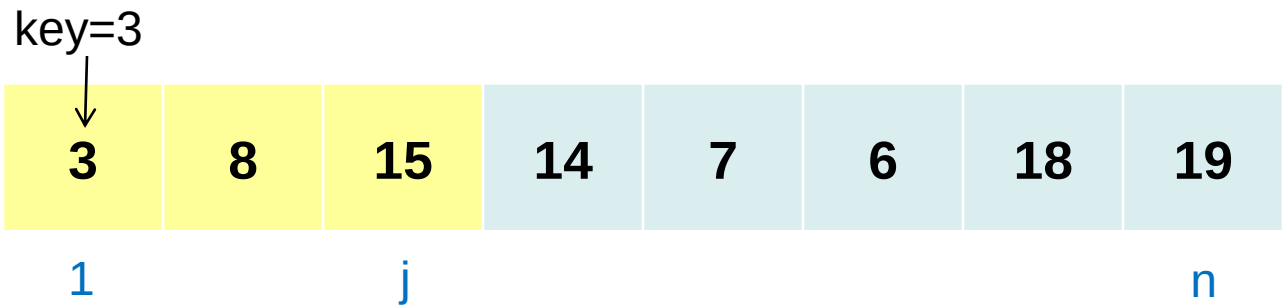
key=3

8	8	15	14	7	6	18	19
i	1		j				n

Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

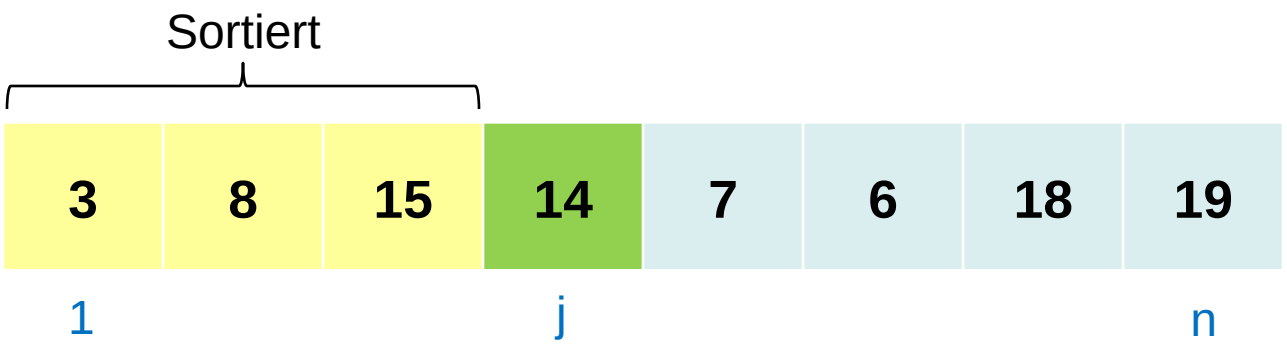
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

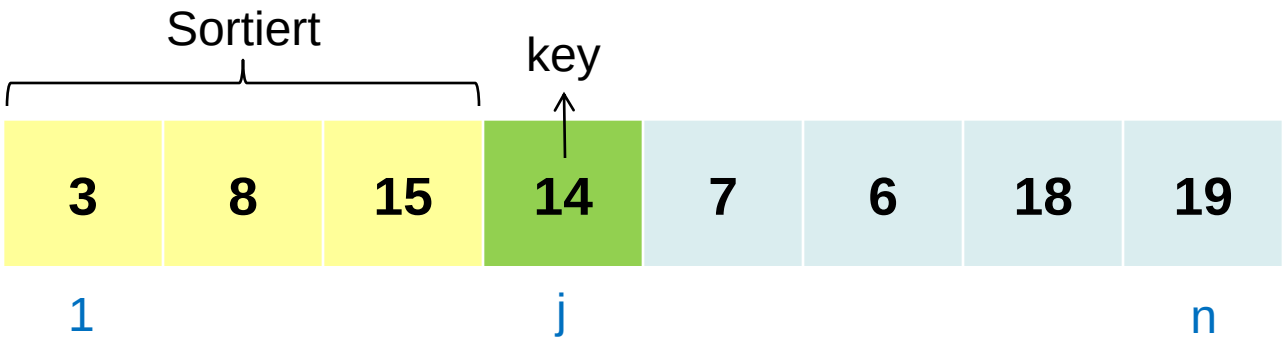
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

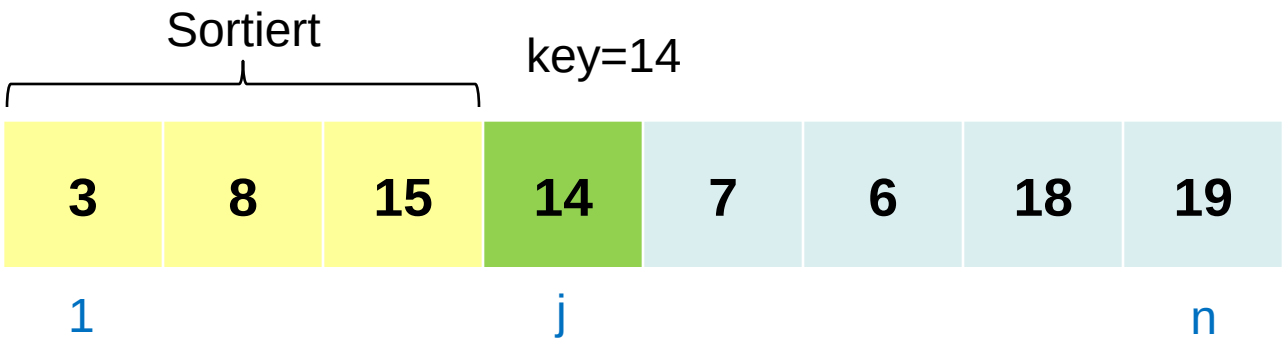
- Eingabegröße n
- length[A] = n



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

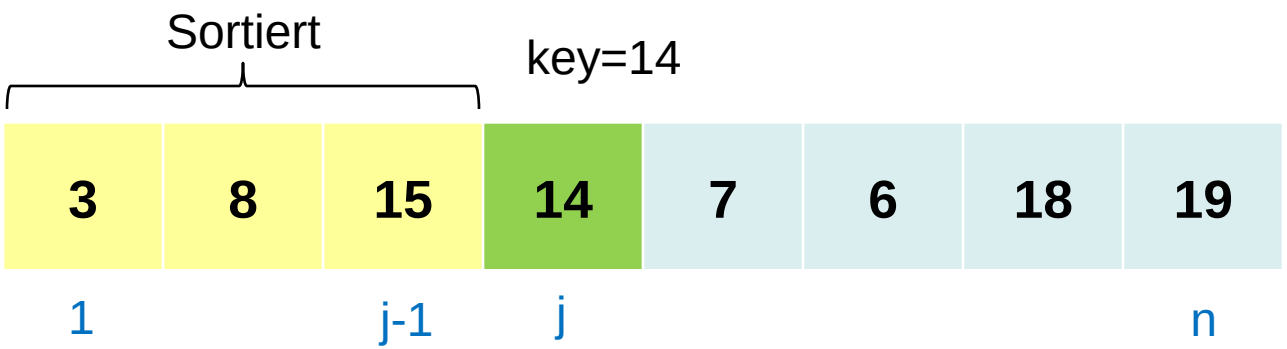


Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts

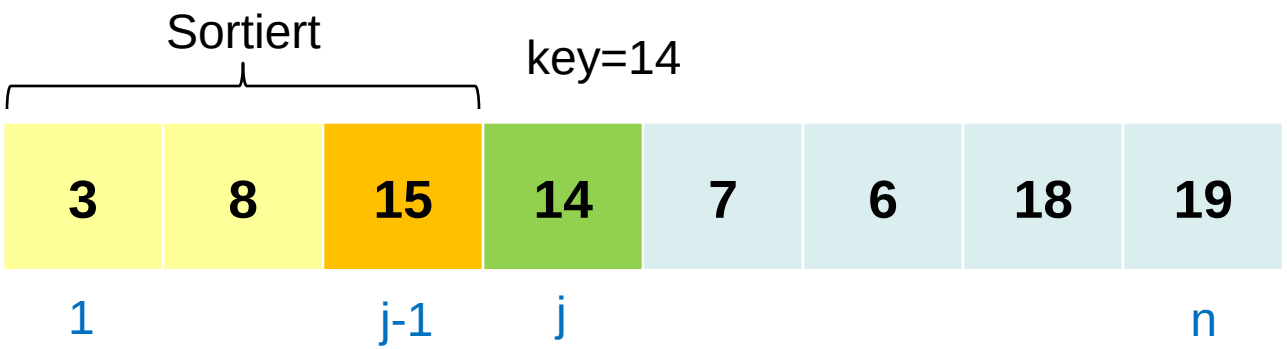


Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts

key=14

3	8	15	15	7	6	18	19
1	i	j-1	j				n

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n

- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts

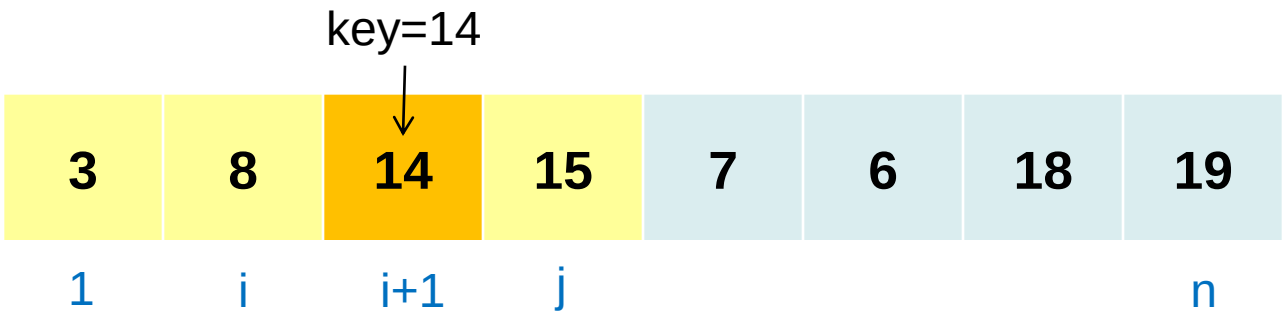
key=14

3	8	15	15	7	6	18	19
1	i	i+1	j				n

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

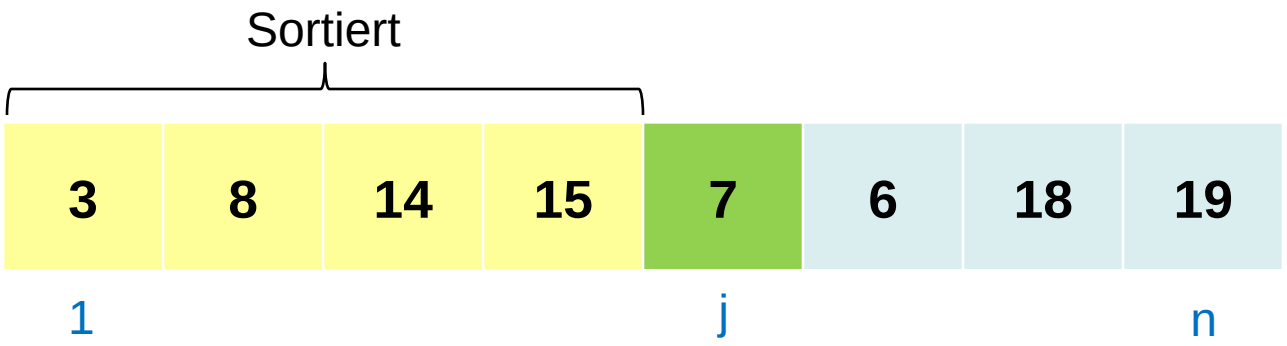
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

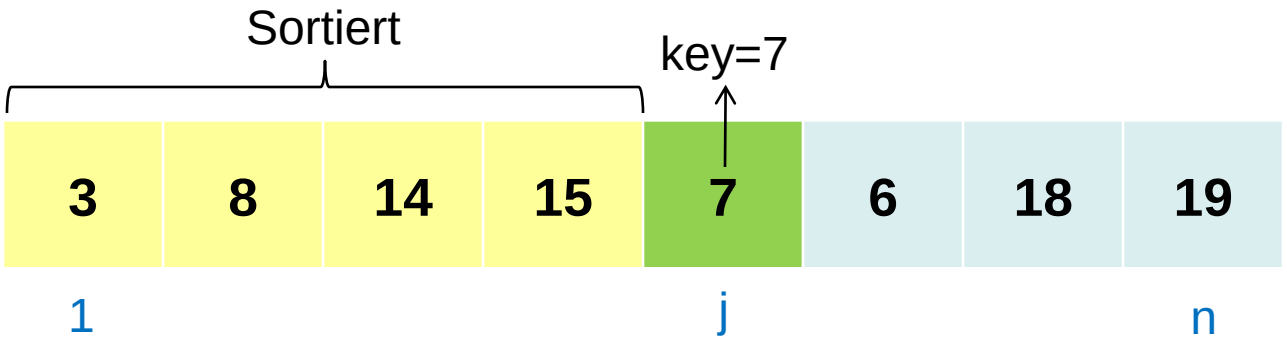
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.      key ← A[j]
3.      i ← j-1
4.      while i>0 and A[i]>key do
5.        A[i+1] ← A[i]
6.        i ← i-1
7.      A[i+1] ← key
```

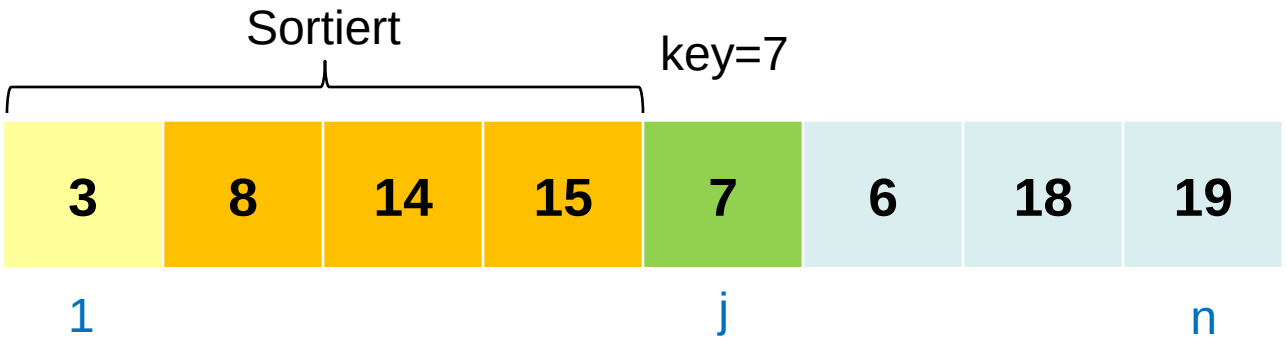
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus A[1…j-1], die größer als key sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

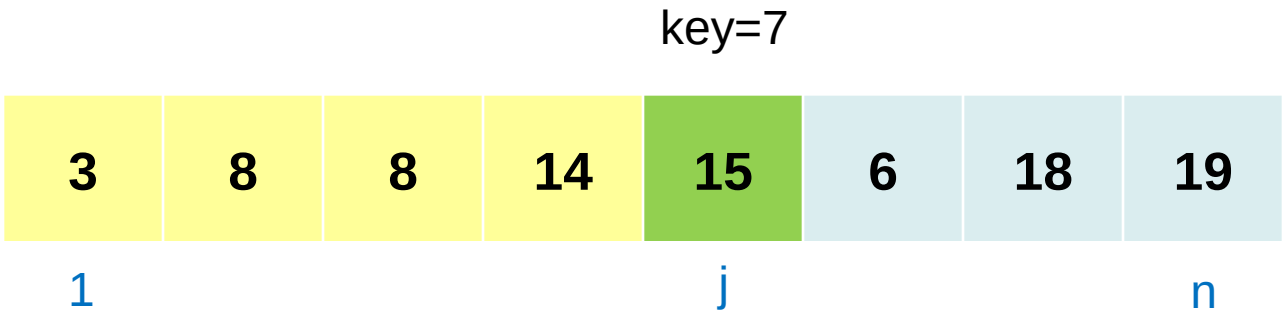
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

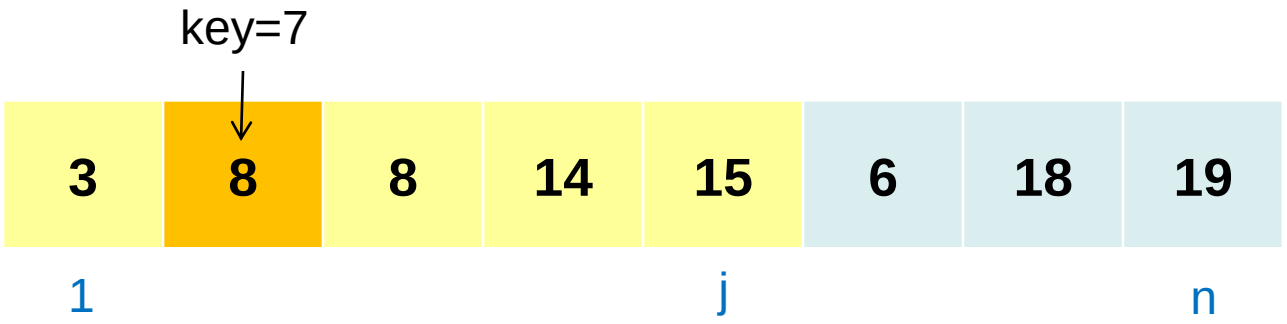
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

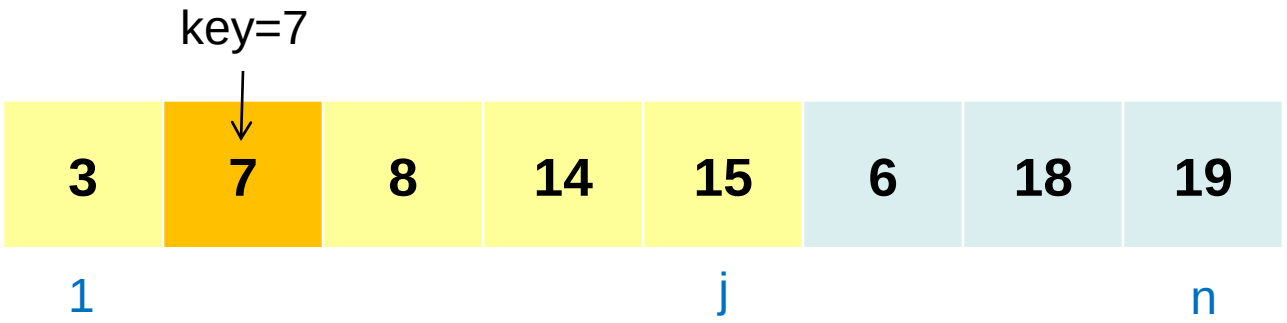
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

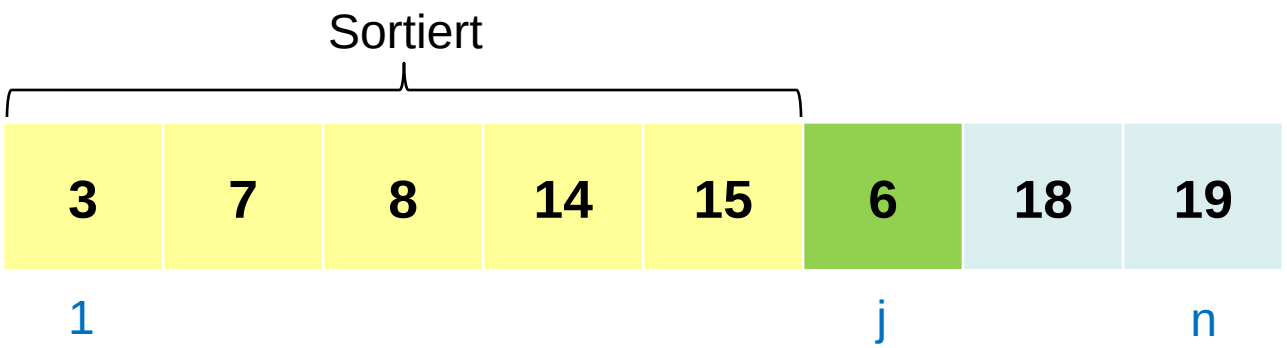
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus A[1…j-1], die größer als key sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

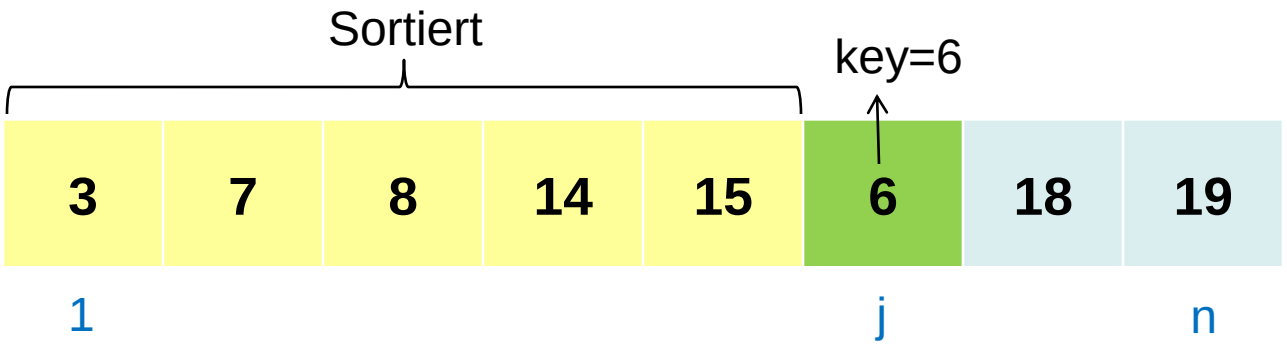
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.      key ← A[j]
3.      i ← j-1
4.      while i>0 and A[i]>key do
5.        A[i+1] ← A[i]
6.        i ← i-1
7.      A[i+1] ← key
```

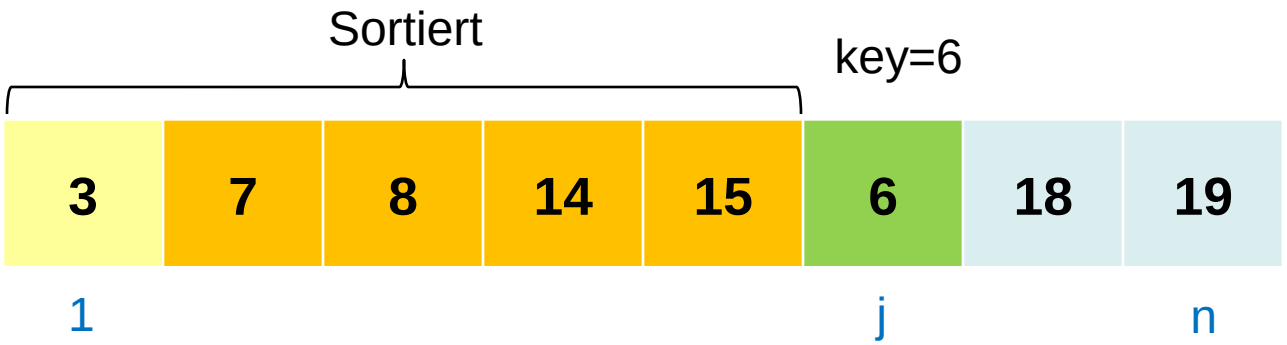
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus A[1…j-1], die größer als key sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

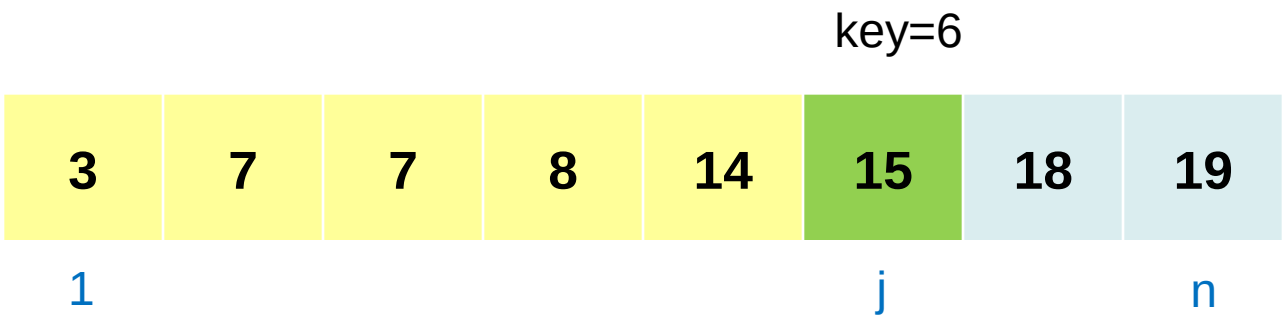
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

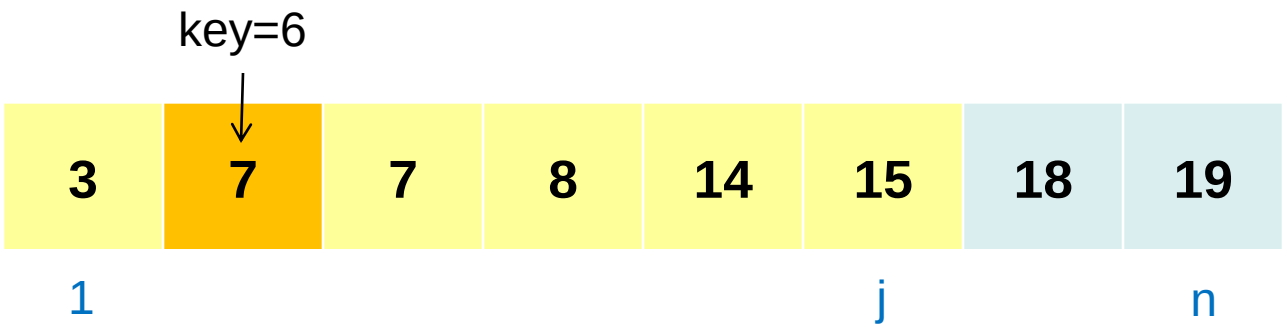
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

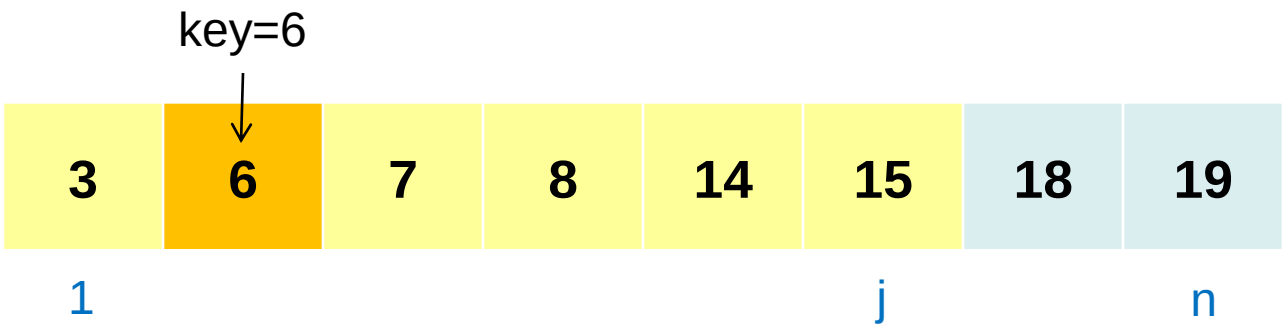
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.      for j ← 2 to length[A] do
2.          key ← A[j]
3.          i ← j-1
4.          while i>0 and A[i]>key do
5.              A[i+1] ← A[i]
6.              i ← i-1
7.          A[i+1] ← key
```

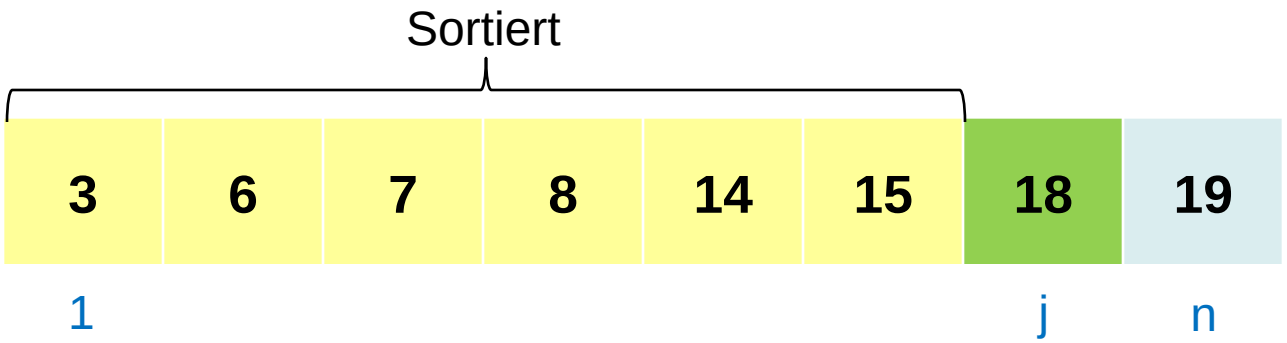
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.        key ← A[j]
3.        i ← j-1
4.        while i>0 and A[i]>key do
5.            A[i+1] ← A[i]
6.            i ← i-1
7.        A[i+1] ← key
```

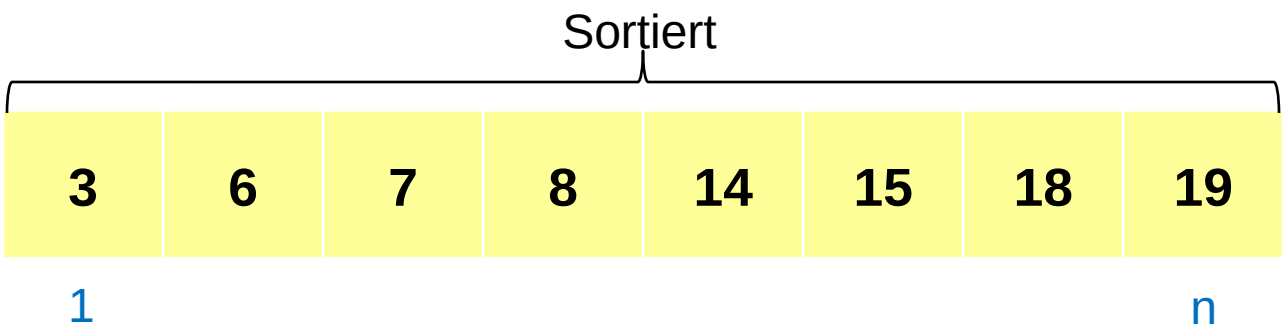
- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke

3	6	7	8	14	15	18	19
1						j	n

Insertion Sort

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke



Laufzeitanalyse

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

- Eingabegröße n
- length[A] = n
- verschiebe alle Elemente aus
- A[1…j-1], die größer als key
- sind eine Stelle nach rechts
- Speichere key in Lücke

Fragestellung

Wie kann man die Laufzeit eines Algorithmus vorhersagen?

Laufzeitanalyse

Laufzeit hängt ab von

- Größe der Eingabe (Parameter n)
- Art der Eingabe
  (Insertionsort ist schneller auf sortierten Eingaben)

Analyse

- Parametrisiere Laufzeit als Funktion der Eingabegröße
- Finde obere Schranken (Garantien) an die Laufzeit

Laufzeitanalyse

Worst-Case Analyse

- Für jedes n definiere Laufzeit
  T(n) = Maximum über alle Eingaben der Größe n
- Garantie für jede Eingabe
- Standard

Average-Case Analyse

- Für jedes n definiere Laufzeit
  T(n) = Durchschnitt über alle Eingaben der Größe n
- Hängt von Definition des Durchschnitts ab (wie sind die Eingaben verteilt)

Laufzeitanalyse

Laufzeit hängt auch ab von

- Hardware
  (Prozessor, Cache, Pipelining)
- Software
  (Betriebssystem, Programmiersprache, Compiler)

Aber

- Analyse soll unabhängig von Hard- und Software gelten

Laufzeitanalyse

Maschinenmodell

- Eine Pseudocode-Instruktion braucht einen Zeitschritt
- Wird eine Instruktion r-mal aufgerufen, werden r Zeitschritte benötigt
- Formales Modell: Random Access Machines (RAM Modell)

Idee

- Ignoriere rechnerabhängige Konstanten
- Betrachte Wachstum von T(n) für $n \to \infty$

„Asymptotische Analyse"

Laufzeitanalyse

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

Was ist die Eingabegröße?

Laufzeitanalyse

```
InsertionSort(Array A)
1.    for j ← 2 to length[A] do
2.       key ← A[j]
3.       i ← j-1
4.       while i>0 and A[i]>key do
5.          A[i+1] ← A[i]
6.          i ← i-1
7.       A[i+1] ← key
```

Was ist die Eingabegröße?

Die Länge des Feldes A

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	
3. i ← j-1	
4. **while** i>0 and A[i]>key **do**	
5. A[i+1] ← A[i]	
6. i ← i-1	
7. A[i+1] ← key	

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	
4. **while** i>0 and A[i]>key **do**	
5. A[i+1] ← A[i]	
6. i ← i-1	
7. A[i+1] ← key	

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	
5. A[i+1] ← A[i]	
6. i ← i-1	
7. A[i+1] ← key	

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	n-1 + $\sum t_j$
5. A[i+1] ← A[i]	
6. i ← i-1	
7. A[i+1] ← key	

t_j : Anzahl Wiederholungen der while-Schleife bei Laufindex j

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	n-1 + $\sum t_j$
5. A[i+1] ← A[i]	$\sum t_j$
6. i ← i-1	
7. A[i+1] ← key	

t_j : Anzahl Wiederholungen der while-Schleife bei Laufindex j

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	n-1 + $\sum t_j$
5. A[i+1] ← A[i]	$\sum t_j$
6. i ← i-1	$\sum t_j$
7. A[i+1] ← key	

t_j : Anzahl Wiederholungen der while-Schleife bei Laufindex j

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	n-1 + $\sum t_j$
5. A[i+1] ← A[i]	$\sum t_j$
6. i ← i-1	$\sum t_j$
7. A[i+1] ← key	n-1

t_j : Anzahl Wiederholungen der while-Schleife bei Laufindex j

Laufzeitanalyse

InsertionSort(Array A)	Zeit:
1. **for** j ← 2 **to** length[A] **do**	n
2. key ← A[j]	n-1
3. i ← j-1	n-1
4. **while** i>0 and A[i]>key **do**	n-1 + $\sum t_j$
5. A[i+1] ← A[i]	$\sum t_j$
6. i ← i-1	$\sum t_j$
7. A[i+1] ← key	n-1
	$5n-4+3\sum t_j$

t_j : Anzahl Wiederholungen der while-Schleife bei Laufindex j

Laufzeitanalyse

Worst-Case Analyse

- t_j =j-1 für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{j=2}^{n} (j-1) = 2n - 4 + 3 \cdot \sum_{j=1}^{n} j$$

Laufzeitanalyse

Worst-Case Analyse

- t_j =j-1 für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{j=2}^{n} (j-1) = 2n - 4 + 3 \cdot \sum_{j=1}^{n} j$$

$$= 2n - 4 + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2 + 7n - 8}{2}$$

Laufzeitanalyse

Worst-Case Analyse

- t_j =j-1 für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{j=2}^{n} (j-1) = 2n - 4 + 3 \cdot \sum_{j=1}^{n} j$$

$$= 2n - 4 + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2 + 7n - 8}{2}$$

- Abstraktion von multiplikativen Konstanten $\rightarrow$ O-Notation

Laufzeitanalyse

Diskussion

- Die konstanten Faktoren sind wenig aussagekräftig, da wir bereits bei den einzelnen Befehlen konstante Faktoren ignorieren
- Je nach Rechnerarchitektur und genutzten Befehlen könnte also z.B. 3n+4 langsamer sein als 5n+7
- Betrachte nun Algorithmus A mit Laufzeit 100n und Algorithmus B mit Laufzeit $5n^2$
- Ist n klein, so ist Algorithmus B schneller
- Ist n groß, so wird das Verhältnis Laufzeit B / Laufzeit A beliebig groß
- Algorithmus B braucht also einen beliebigen Faktor mehr Laufzeit als A (wenn die Eingabe lang genug ist)

Laufzeitanalyse

Idee (asymptotische Laufzeitanalyse)

- Ignoriere konstante Faktoren
- Betrachte das Verhältnis von Laufzeiten für $n \to \infty$
- Klassifiziere Laufzeiten durch Angabe von „einfachen Vergleichsfunktionen“

Laufzeitanalyse

O-Notation

- $O(f(n)) = \{g(n) : \exists\ c>0, n_0>0$, so dass für alle $n \geq n_0$ gilt $g(n) \leq c \cdot f(n)\}$
- (wobei $f(n), g(n)>0$)

Interpretation

- $g(n) \in O(f(n))$ bedeutet, dass $g(n)$ für $n \to \infty$ höchstens genauso stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Laufzeitanalyse

Beispiele

- $10n \in O(n)$
- $10n \in O(n^2)$
- $n^2 \notin O(1000n)$
- $O(1000n) = O(n)$

Hierarchie

- $O(\log n) \subseteq O(\log^2 n) \subseteq O(\log^c n) \subseteq O(n^\varepsilon) \subseteq O(\sqrt{n}) \subseteq O(n)$
- $O(n) \subseteq O(n^2) \subseteq O(n^c) \subseteq O(2^n)$
- (für $c >= 2$ und $0 < \varepsilon \leq 1/2$)

Laufzeitanalyse

Ω-Notation

- $\Omega(f(n)) = \{g(n) : \exists\ c>0, n_0>0$, so dass für alle $n \geq n_0$ gilt $g(n) \geq c \cdot f(n)\}$
- (wobei f(n), g(n)>0)

Interpretation

- $g(n) \in \Omega(f(n))$ bedeutet, dass g(n) für $n \to \infty$ mindestens so stark wächst wie f(n)
- Beim Wachstum ignorieren wir Konstanten

Laufzeitanalyse

Beispiele

- $10n \in \Omega(n)$
- $1000n \notin \Omega(n^2)$
- $n^2 \in \Omega(n)$
- $\Omega(1000n) = \Omega(n)$
- $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

Laufzeitanalyse

Θ-Notation

- $g(n) \in \Theta(f(n)) \Leftrightarrow g(n)=O(f(n))$ und $g(n)=\Omega(f(n))$

Beispiele

- $1000n \in \Theta(n)$
- $10n^2+1000n \in \Theta(n^2)$
- $n^{1-\sin n} \notin \Theta(n)$

Laufzeitanalyse

o-Notation

- $o(f(n)) \in \{g(n): \forall c>0\ \exists n_0 >0$, so dass für alle $n \geq n_0$ gilt $c \cdot g(n) < f(n)\}$
- $(f(n), g(n) >0)$

ω-Notation

- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

Laufzeitanalyse

Beispiele

- $n \in o(n^2)$
- $n \notin o(n)$

Eine weitere Interpretation

- Grob gesprochen sind O,Ω,Θ,o,ω die „asymptotischen Versionen“ von $\leq,\geq,=,<,>$ (in dieser Reihenfolge)

Schreibweise

- Wir schreiben häufig $f(n)=O(g(n))$ anstelle von $f(n) \in O(g(n))$

Laufzeitanalyse

Worst-Case Analyse (Insertion Sort)

- t_j =j-1 für absteigend sortierte Eingabe (schlechtester Fall)

$$T(n) = 5n - 4 + 3 \cdot \sum_{j=2}^{n} (j-1) = 2n - 4 + 3 \cdot \sum_{j=1}^{n} j$$

$$= 2n - 4 + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2 + 7n - 8}{2} = \Theta(n^2)$$

Zusammenfassung

Rechenmodell

- Abstrahiert von maschinennahen Einflüssen wie Cache, Pipelining, Prozessor, etc.
- Jede Pseudocodeoperation braucht einen Zeitschritt

Laufzeitanalyse

- Normalerweise Worst-Case, manchmal Average-Case (sehr selten auch Best-Case)
- Asymptotische Analyse für $n \to \infty$
- Ignorieren von Konstanten $\rightarrow$ O-Notation