



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Organisatorisches

### *Heimübungsblatt 4*

- Aufgabe 1 wurde ausgetauscht
- Falls Sie die alte 1 bereits gemacht haben, geben Sie sie mit ab

### *Praktikum*

- Ab dem nächsten Blatt (Blatt 9) fließt nur noch eine Aufgabe der Präsenzübung in die Wertung ein
- Weitere Aufgaben sind optional

## Stand der Dinge

### *Gierige Algorithmen*

- Konstruiere Lösung Schritt für Schritt
- In jedem Schritt: Optimierte ein einfaches, lokales Kriterium

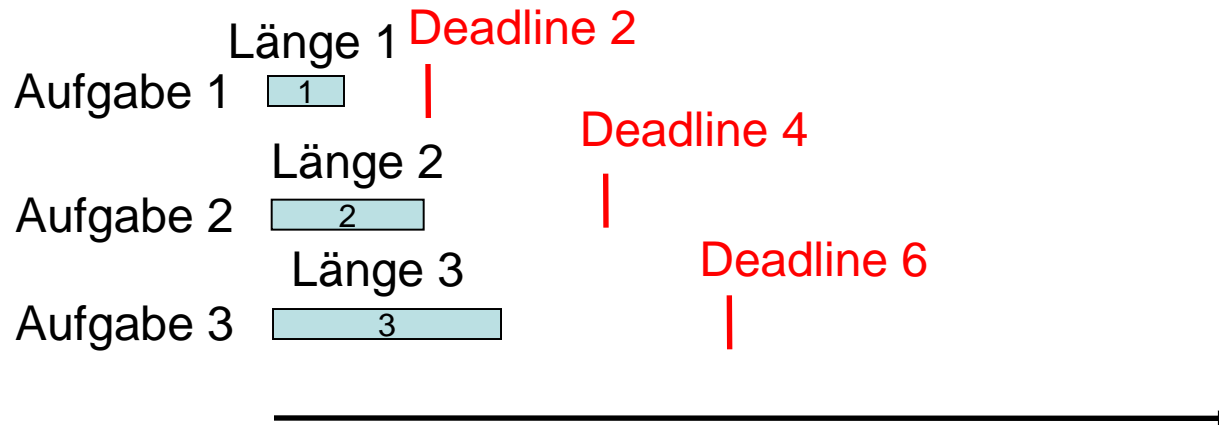
### *Beobachtung*

- Man kann viele unterschiedliche gierige Algorithmen für ein Problem entwickeln
- Nicht jeder dieser Algorithmen löst das Problem korrekt

## Gierige Algorithmen

### Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll

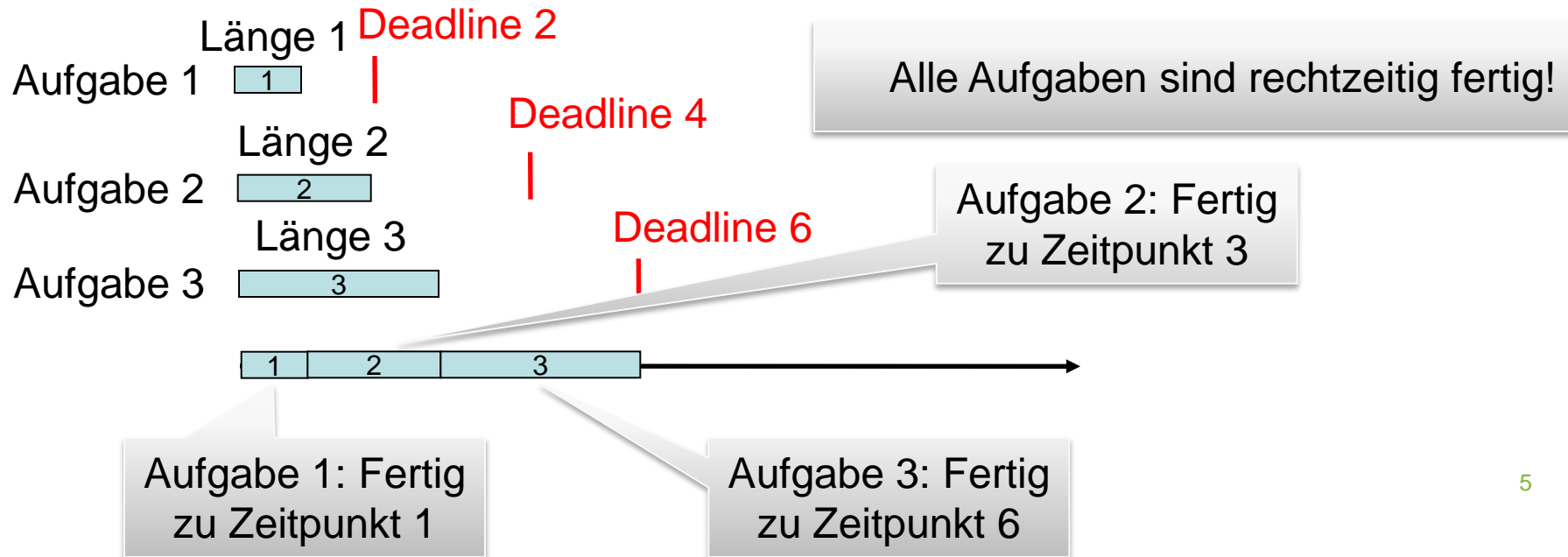


O.b.d.A. Resource steht ab Zeitpunkt 0 zur Verfügung

## Gierige Algorithmen

### Scheduling mit Deadlines

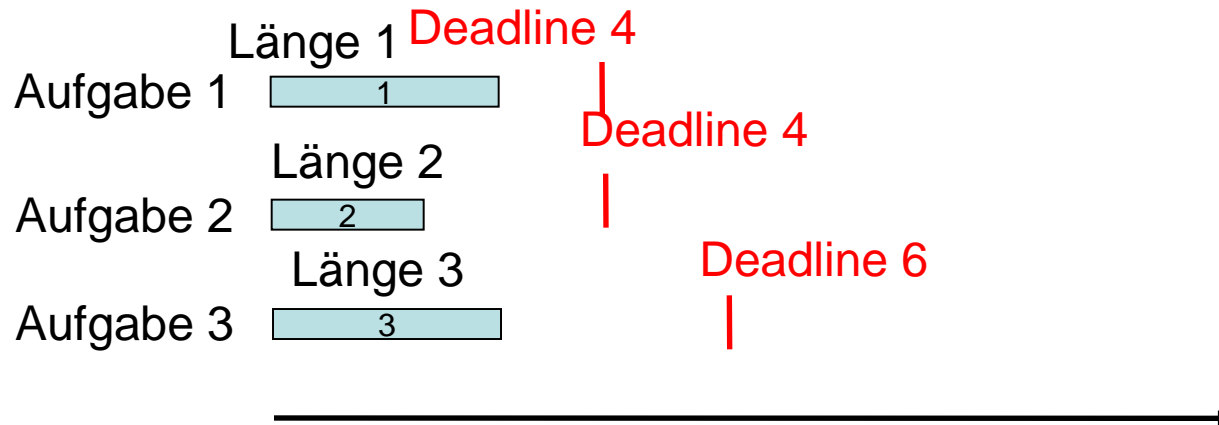
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll



## Gierige Algorithmen

### *Scheduling mit Deadlines*

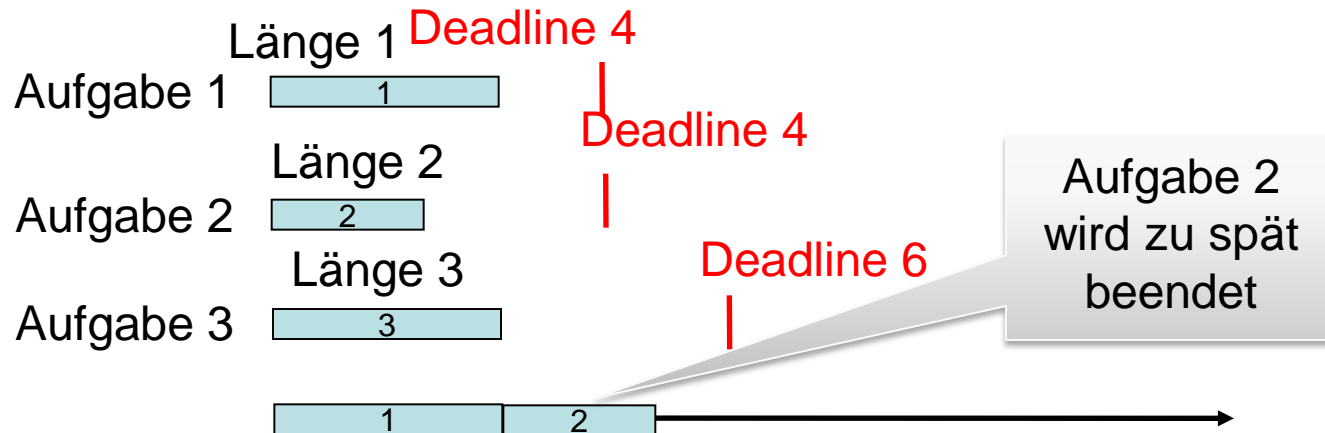
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll



## Stand der Dinge

### Scheduling mit Deadlines

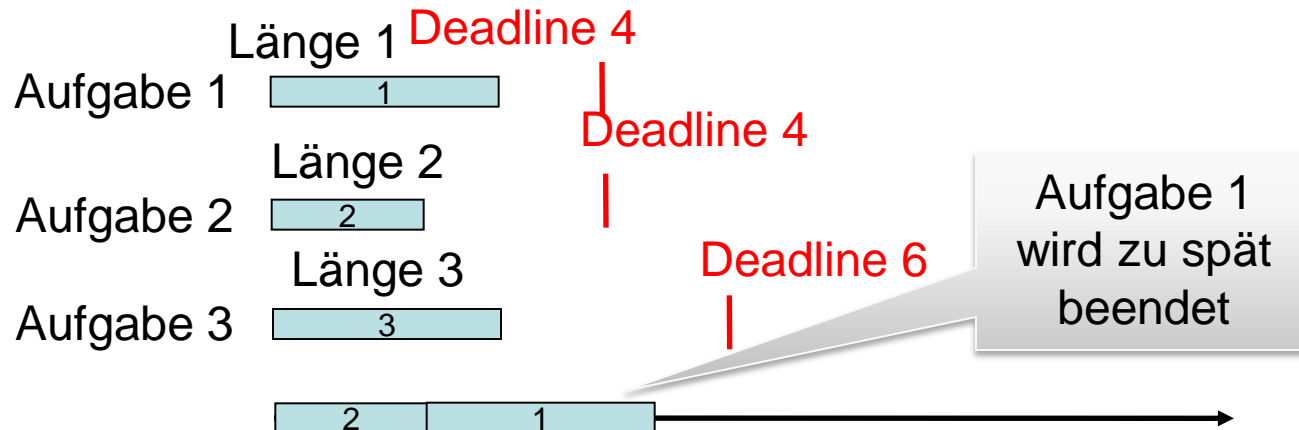
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll



## Gierige Algorithmen

### Scheduling mit Deadlines

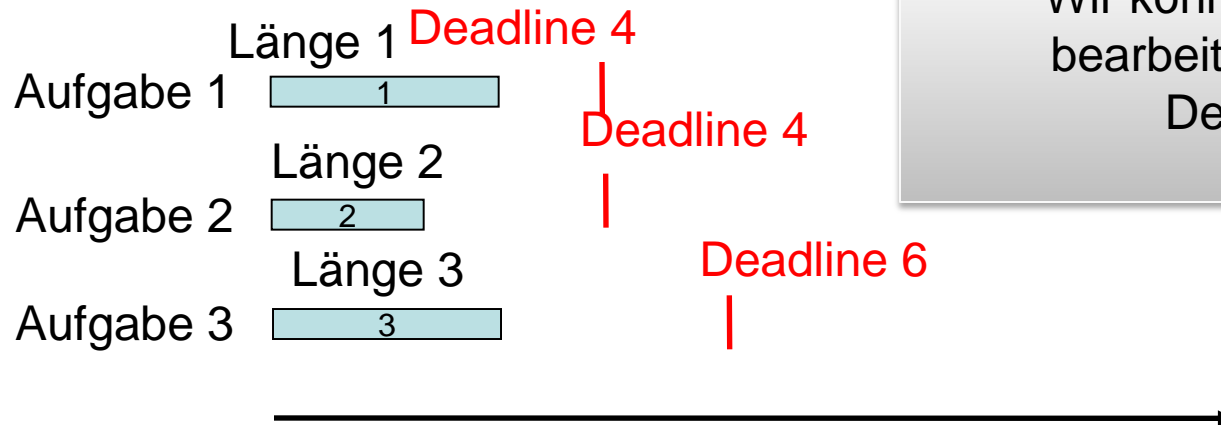
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



## Gierige Algorithmen

### *Scheduling mit Deadlines*

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll

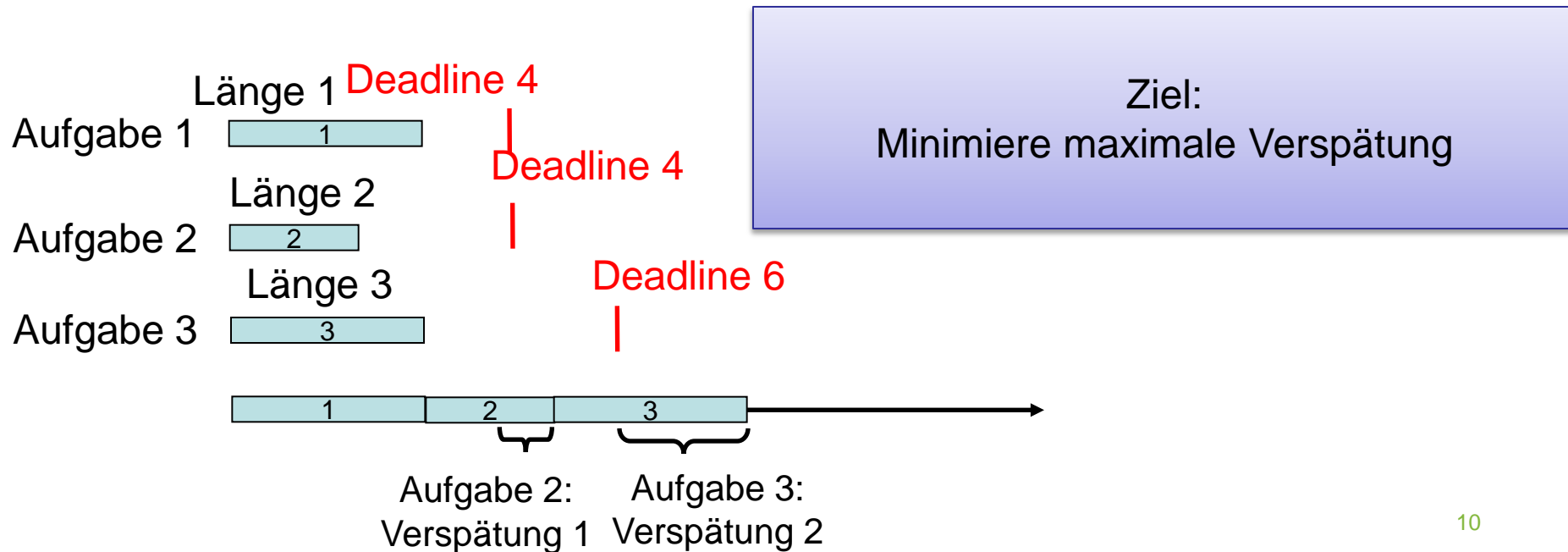


Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

## Gierige Algorithmen

### Scheduling mit Deadlines

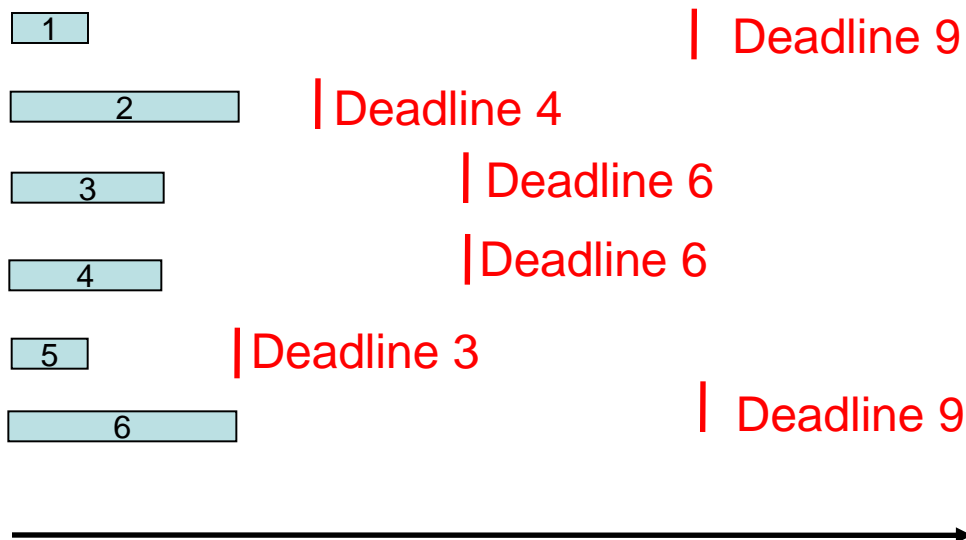
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit  $t$  benötigt und bis Zeitpunkt  $d$  bearbeitet sein soll



## Gierige Algorithmen

### Strategie 1

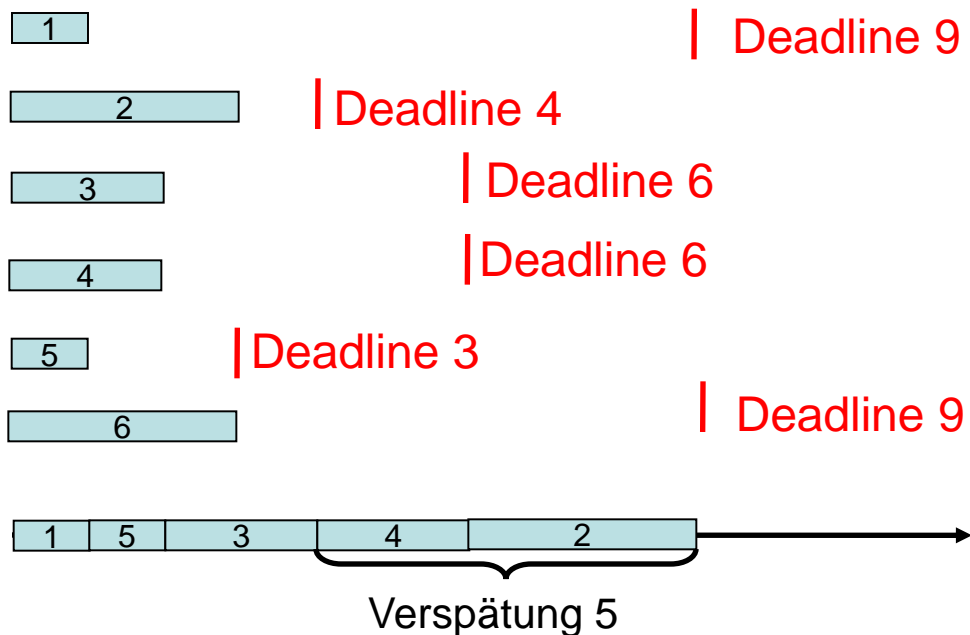
- Bearbeite die Jobs nach ansteigender Länge



## Gierige Algorithmen

### Strategie 1

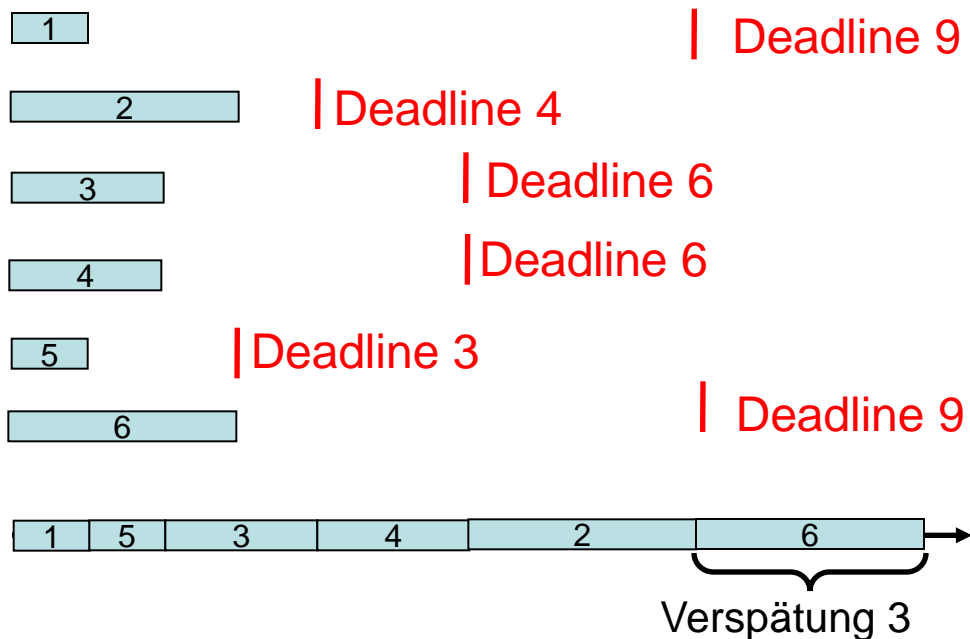
- Bearbeite die Jobs nach ansteigender Länge



## Gierige Algorithmen

### Strategie 1

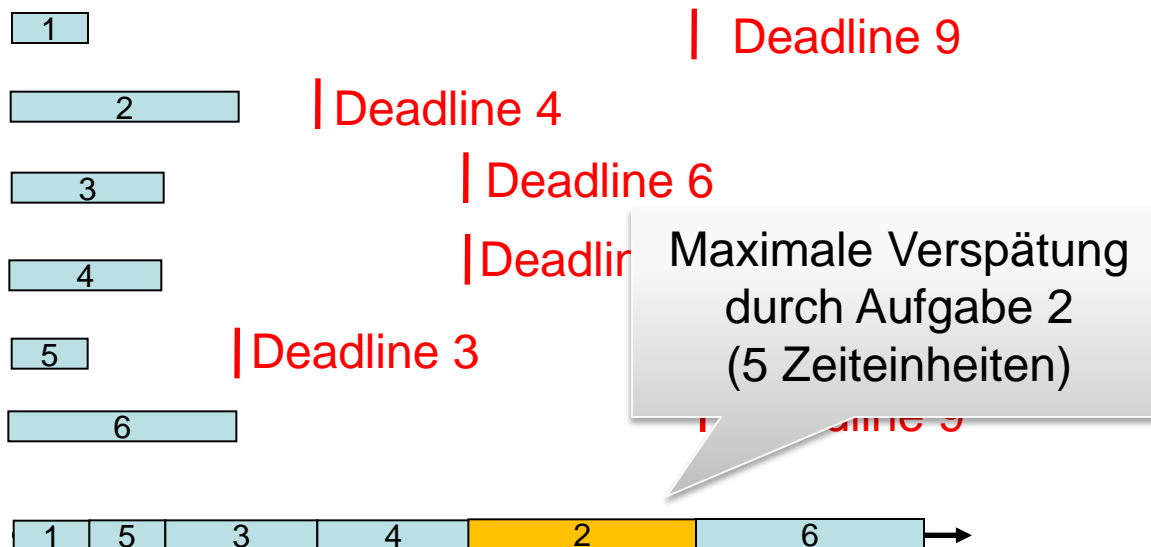
- Bearbeite die Jobs nach ansteigender Länge



## Gierige Algorithmen

### Strategie 1

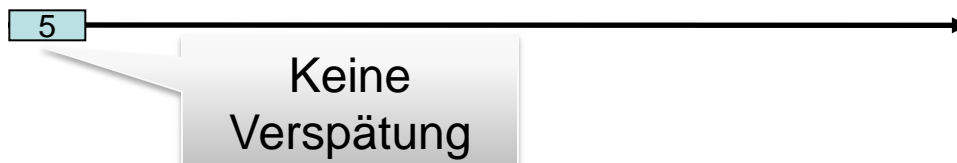
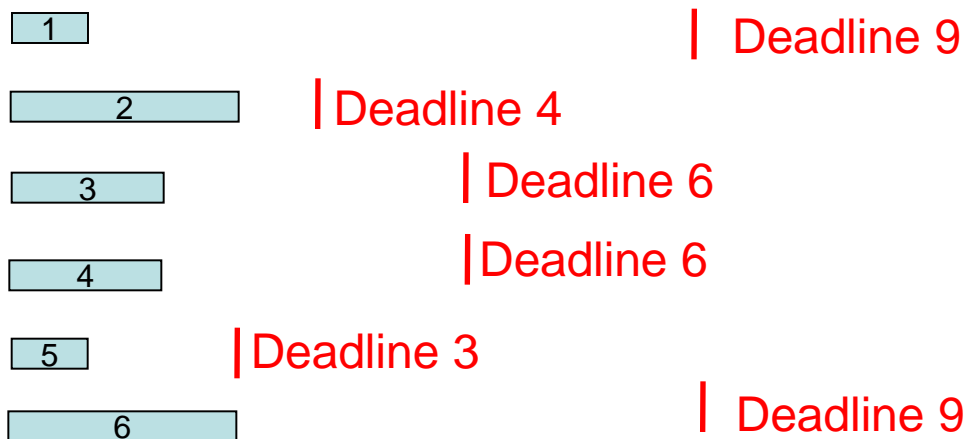
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



## Gierige Algorithmen

### Strategie 1

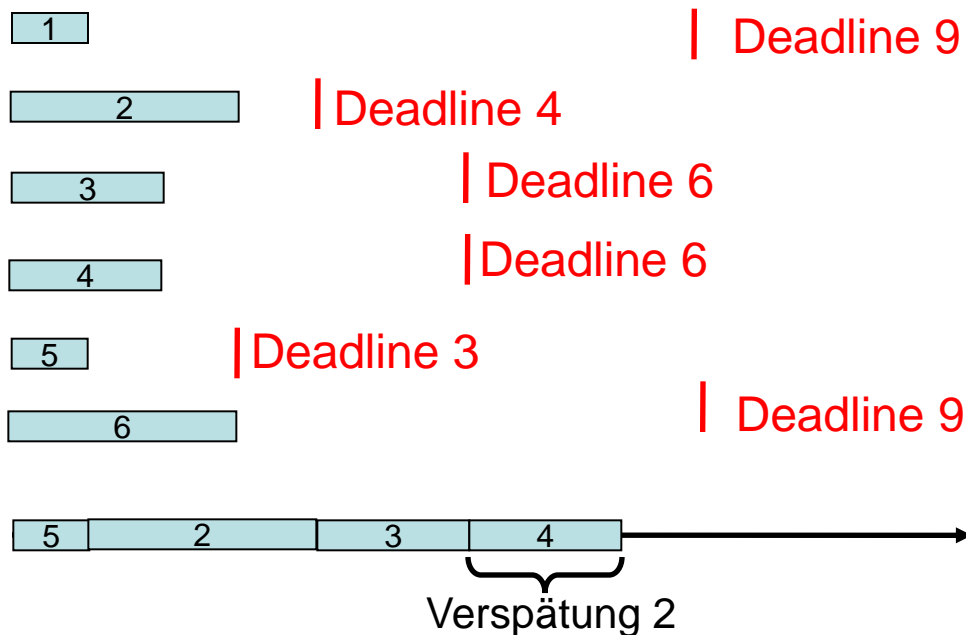
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



## Gierige Algorithmen

### Strategie 1

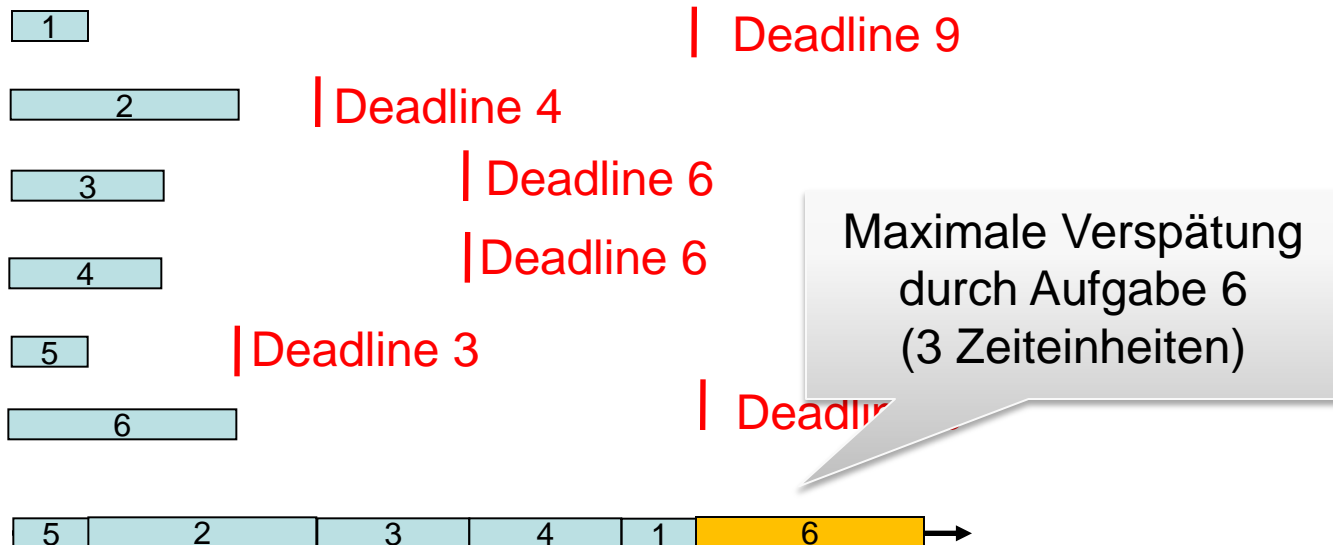
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



## Gierige Algorithmen

### Strategie 1

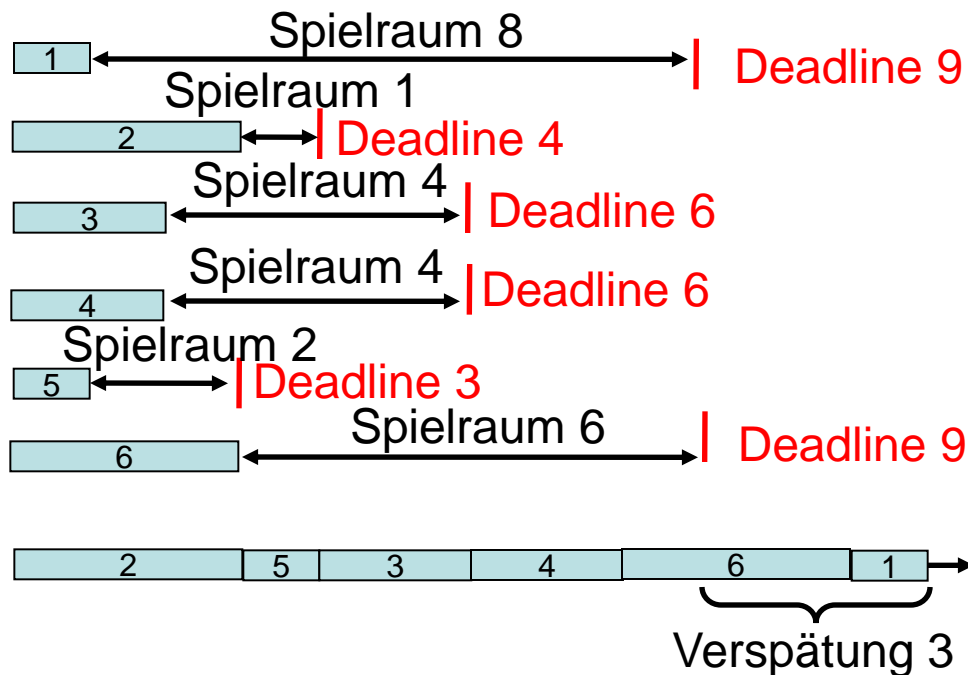
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?
- Problem: Ignoriert Deadlines völlig



## Gierige Algorithmen

### Strategie 2

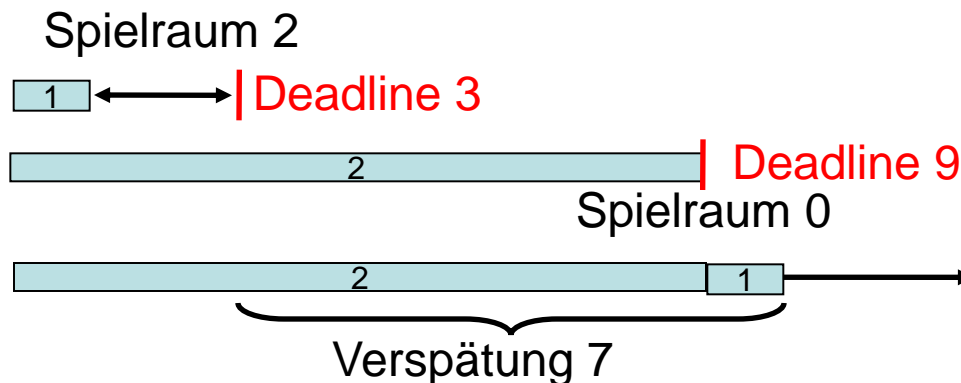
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



## Gierige Algorithmen

### Strategie 2

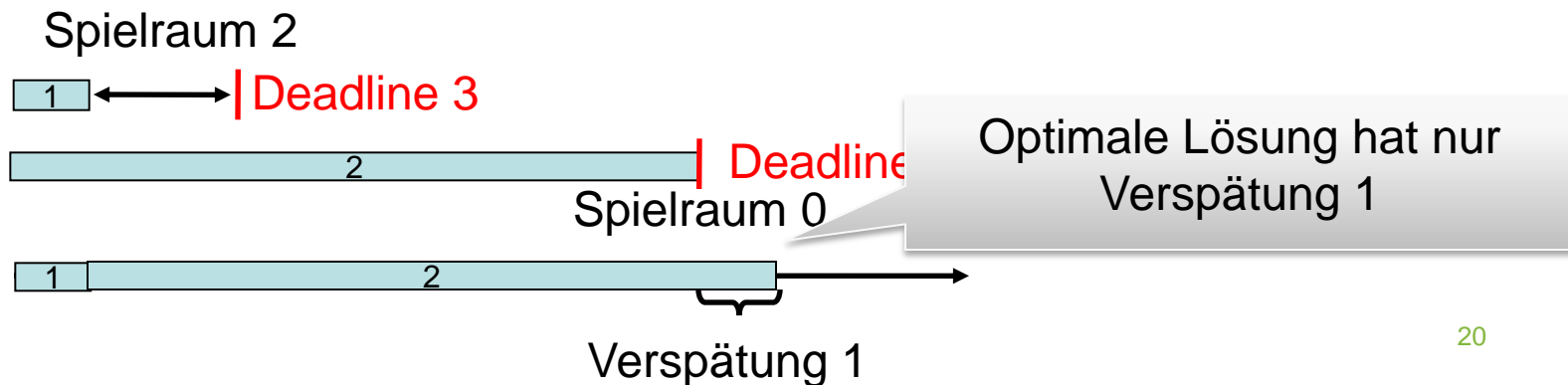
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum  $d-t$
- Optimalität?



## Gierige Algorithmen

### Strategie 2

- Bearbeite zunächst die Aufgaben mit geringstem Spielraum  $d-t$
- Optimalität?

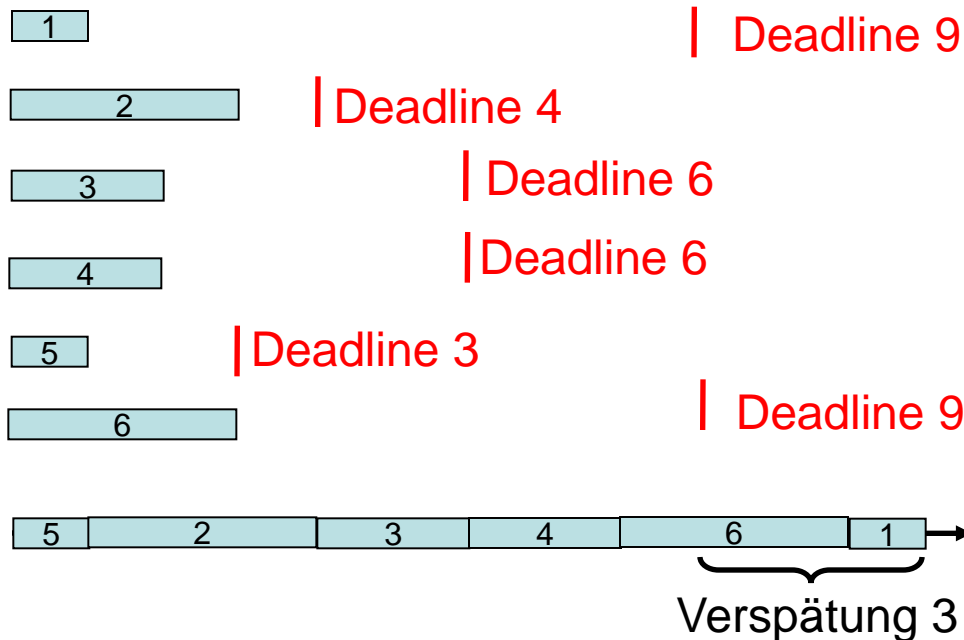


## Gierige Algorithmen

### Strategie 3

- Bearbeite zunächst die Aufgabe mit der frühesten Deadline
- Algorithmus ist optimal!

Komisch, da Strategie unabhängig von der Länge der Aufträge



Lösung optimal!

## Gierige Algorithmen

### *Formale Problemformulierung*

- Problem: Scheduling mit Deadline
- Eingabe: Felder t und d
  - t[i] enthält Länge des i-ten Intervals
  - d[i] enthält Deadline
- Ausgabe: Startzeitpunkte der Intervalle

### *Wichtige Annahme*

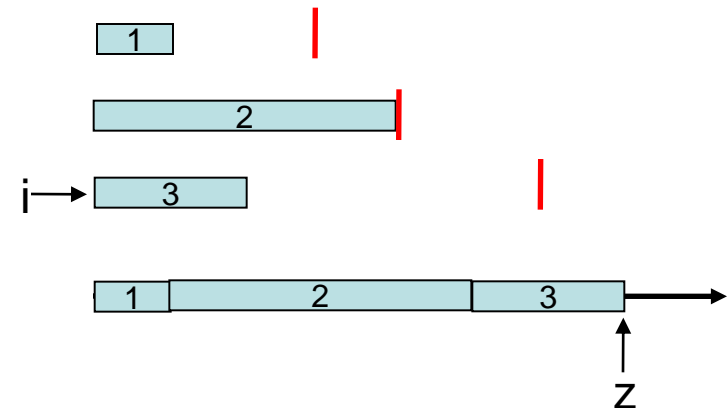
- Eingabe sortiert nach Deadlines
- $d[1] \leq d[2] \leq \dots \leq d[n]$

## Gierige Algorithmen (siehe Vollversion)

LatenessScheduling(t,d)

1.  $n \leftarrow \text{length}[t]$
2. new array  $A[1..n]$
3.  $z \leftarrow 0$
4. **for**  $i \leftarrow 1$  **to**  $n$  **do**
5.      $A[i] \leftarrow z$
6.      $z \leftarrow z + t[i]$
7. **return**  $A$

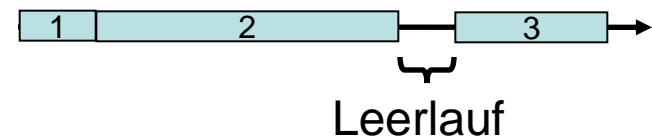
t	1	4	2
d	3	4	6



## Gierige Algorithmen

### *Beobachtung*

Es gibt eine optimale Lösung ohne Leerlaufzeit.



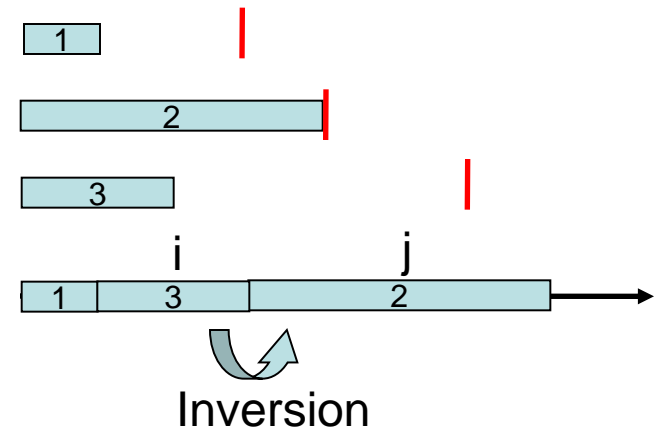
## Gierige Algorithmen

### Lemma 34

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

### Definition

Lösung hat Inversion, wenn Aufgabe  $i$  mit Deadline  $d_i$  vor Aufgabe  $j$  mit Deadline  $d_j < d_i$  bearbeitet wird.



## Gierige Algorithmen

### *Lemma 34*

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

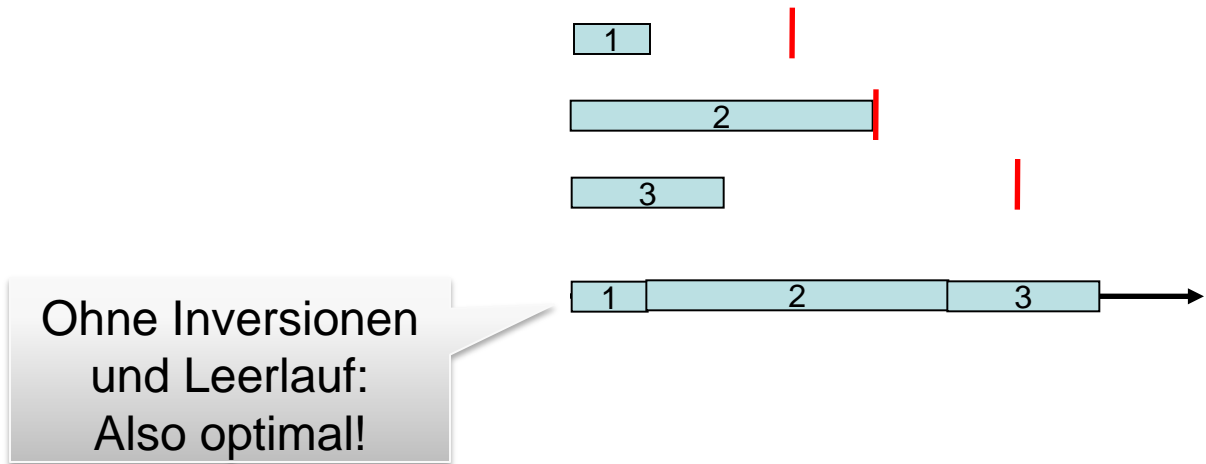
### *Beweis*

- Haben zwei Schedules weder Inversionen noch Leerlaufzeiten, so haben sie zwar nicht notwendigerweise dieselbe Ordnung, aber sie können sich nur in der Ordnung der Aufgaben mit identischer Deadline unterscheiden. Betrachten wir eine solche Deadline  $d$ . In beiden Schedules werden alle Aufgaben mit Deadline  $d$  nacheinander ausgeführt. Unter den Aufgaben mit Deadline  $d$  hat die letzte die größte Verzögerung und diese hängt nicht von der Reihenfolge der Aufgaben ab.

## Gierige Algorithmen

### Lemma 35

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.



## Gierige Algorithmen

### *Lemma 35*

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### *Beweis*

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (a) Wenn  $O$  eine Inversion hat, dann gibt es ein Paar Aufgaben  $i$  und  $j$ , so dass  $j$  direkt nach  $i$  auftritt und  $d_j < d_i$  ist.  
(D.h. eine Inversion von aufeinanderfolgenden Aufgaben)

Deadline 9

Deadline 5



Deadline  $\geq 9$

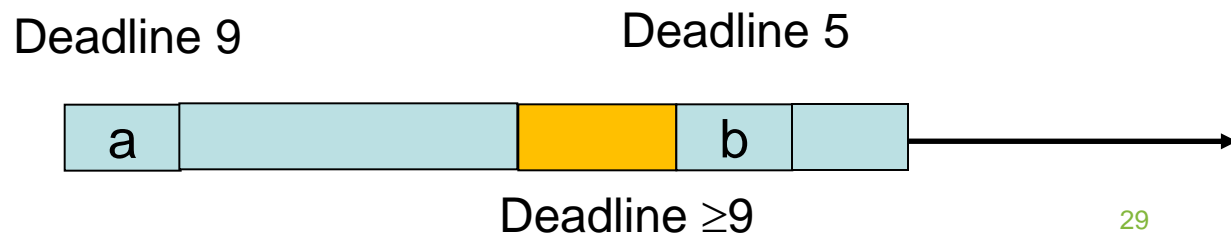
## Gierige Algorithmen

### *Lemma 35*

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### *Beweis*

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (a) Wenn  $O$  eine Inversion hat, dann gibt es ein Paar Aufgaben  $i$  und  $j$ , so dass  $j$  direkt nach  $i$  auftritt und  $d_j < d_i$  ist.  
(D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



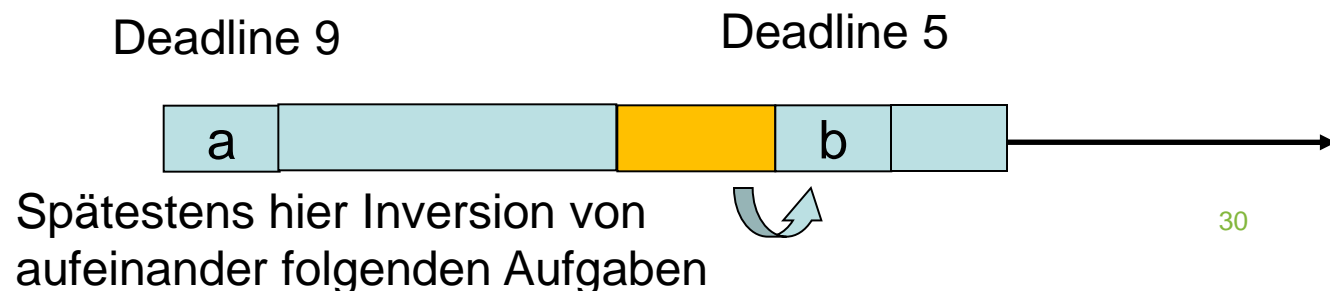
## Gierige Algorithmen

### Lemma 35

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### Beweis

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (a) Wenn  $O$  eine Inversion hat, dann gibt es ein Paar Aufgaben  $i$  und  $j$ , so dass  $j$  direkt nach  $i$  auftritt und  $d_j < d_i$  ist.  
(D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



## Gierige Algorithmen

### *Lemma 35*

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### *Beweis*

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (b) Nach dem Austauschen von einer benachbarten Inversion  $i$  und  $j$  erhalten wir ein Schedule mit einer Inversion weniger.
- Es wird die Inversion von  $i$  und  $j$  durch das Vertauschen aufgehoben und es wird keine neue Inversion erzeugt.

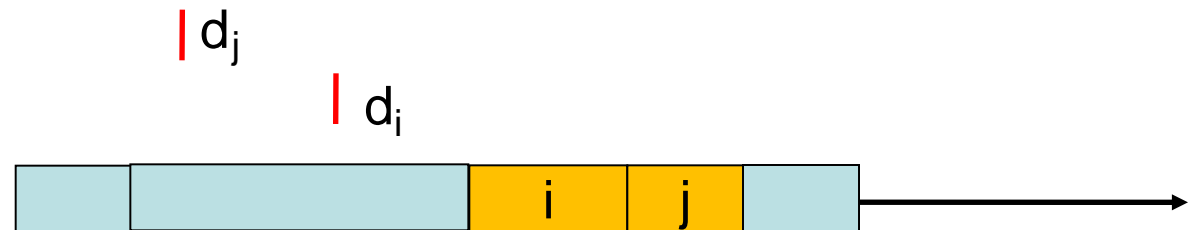
## Gierige Algorithmen

### Lemma 35

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### Beweis

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (c) Das Tauschen von  $i$  und  $j$  erhöht nicht die maximale Verzögerung.



Aufeinander folgende Inversion (i,j)

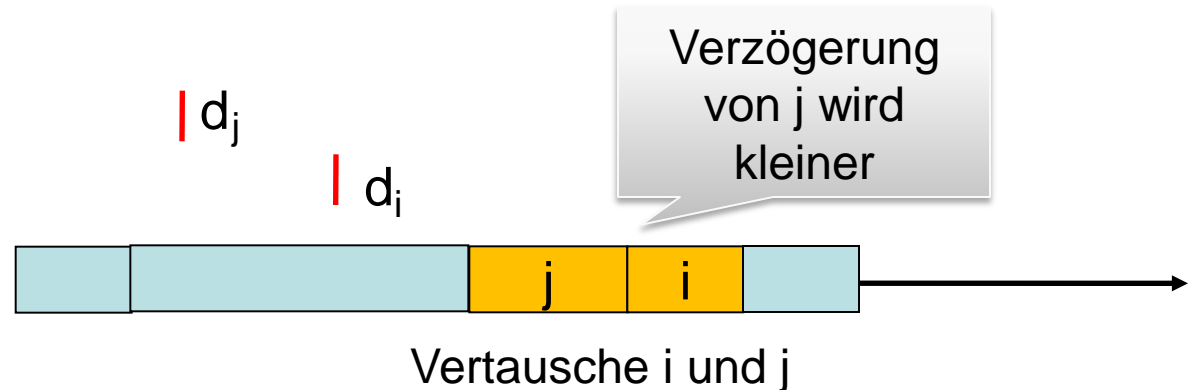
## Gierige Algorithmen

### Lemma 35

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### Beweis

- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (c) Das Tauschen von  $i$  und  $j$  erhöht nicht die maximale Verzögerung.



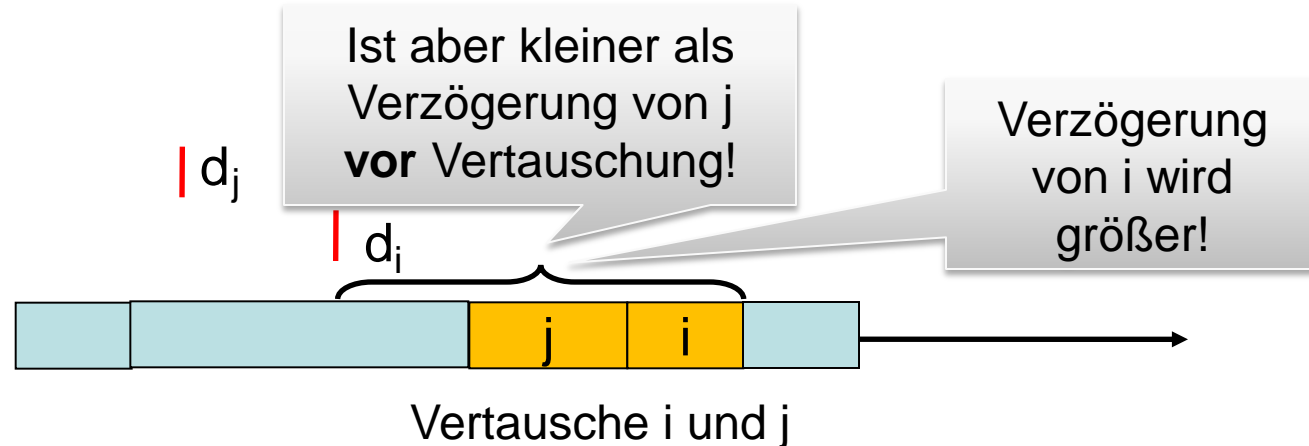
## Gierige Algorithmen

### Lemma 35

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### Beweis

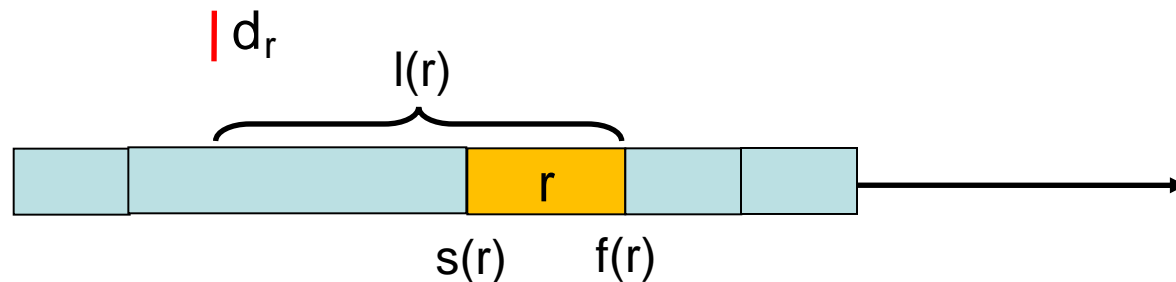
- Sei  $O$  ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (c) Das Tauschen von  $i$  und  $j$  erhöht nicht die maximale Verzögerung.



## Gierige Algorithmen

### Formaler Beweis von (c)

- Notation für  $O$ : Aufgabe  $r$  wird im Intervall  $[s(r), f(r)]$  ausgeführt und hat Verzögerung  $l(r)$ . Sei  $L = \max l(r)$  die maximale Verzögerung dieses Schedules.

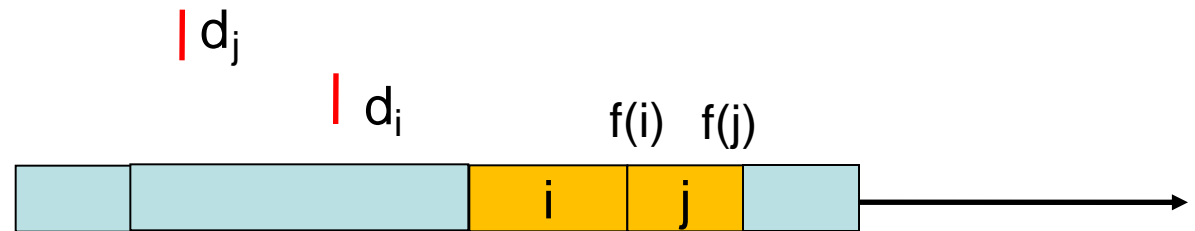


- Notation für das Schedule  $O^*$  nach Austauschen:  $s^*(r)$ ,  $f^*(r)$ ,  $l^*(r)$  und  $L^*$  mit der entsprechenden Bedeutung wie oben.
- $s(r)$ ,  $s^*(r)$  heißt Startzeit
- $f(r)$ ,  $f^*(r)$  heißt Abarbeitungszeit

## Gierige Algorithmen

### *Formaler Beweis von (c)*

- Betrachten wir nun die benachbarte Inversion von  $i$  und  $j$ . Die Abarbeitungszeit  $f(j)$  von  $j$  vor dem Austauschen ist gleich der Abarbeitungszeit  $f^*(i)$  von  $i$  nach dem Austauschen. Daher haben alle anderen Aufgaben vor und nach dem Tauschen dieselbe Abarbeitungszeit.

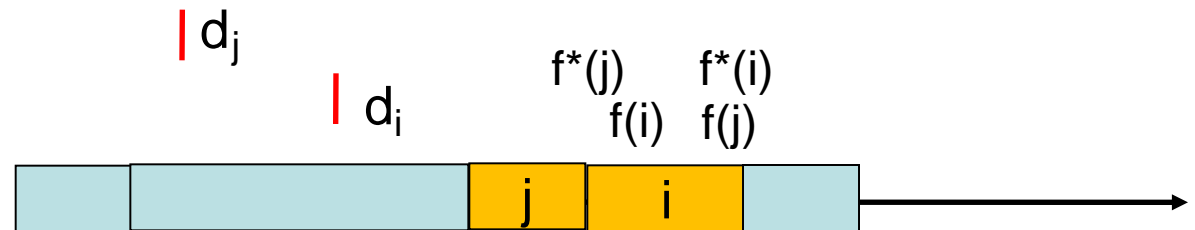


Aufeinander folgende Inversion (i,j)

## Gierige Algorithmen

### Formaler Beweis von (c)

- Betrachten wir nun die benachbarte Inversion von  $i$  und  $j$ . Die Abarbeitungszeit  $f(j)$  von  $j$  vor dem Austauschen ist gleich der Abarbeitungszeit  $f^*(i)$  von  $i$  nach dem Austauschen. Daher haben alle anderen Aufgaben vor und nach dem Tauschen dieselbe Abarbeitungszeit.
- Für Aufgabe  $j$  ist das neue Schedule besser, d.h.  $f^*(j) < f(j)$ .

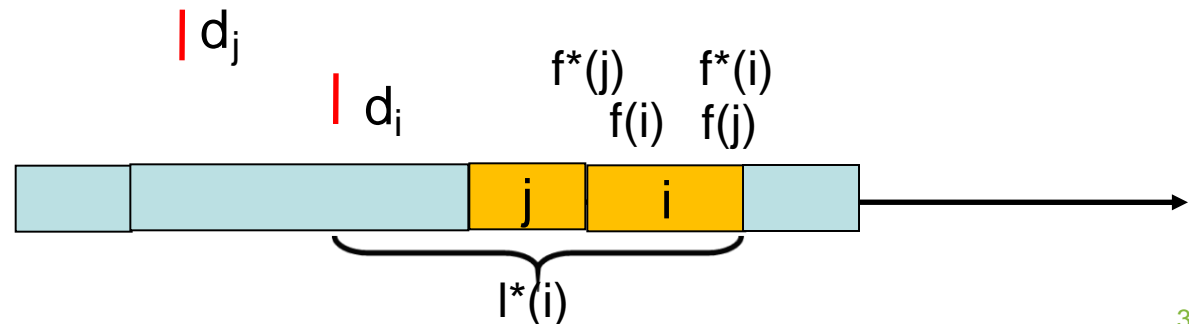


Aufeinander folgende Inversion (i,j)

## Gierige Algorithmen

### Formaler Beweis von (c)

- Betrachte nur Aufgabe i: Nach dem Tauschen ist die Verzögerung  $l^*(i) = f^*(i) - d_i$ .
- Wegen  $d_i > d_j$  folgt  $l^*(i) = f(j) - d_i < f(j) - d_j = l(j)$ .
- Damit wird die maximale Verzögerung nicht erhöht.



## Gierige Algorithmen

### *Lemma 35*

Es gibt eine **optimale Lösung** ohne Inversionen und Leerlaufzeit.

### *Beweis*

- (a) Wenn  $O$  eine Inversion hat, dann gibt es ein Paar Aufgaben  $i$  und  $j$ , so dass  $j$  direkt nach  $i$  auftritt und  $d_j < d_i$  ist.
- (b) Nach dem Austauschen von einer benachbarten Inversion  $i$  und  $j$  erhalten wir ein Schedule mit einer Inversion weniger.
- (c) Das Tauschen von  $i$  und  $j$  erhöht nicht die maximale Verzögerung.
- Die Anzahl Inversionen ist zu Beginn höchstens  $\binom{n}{2}$ . Wir können (a)-(c) solange anwenden, bis keine Inversionen mehr vorhanden sind.

## Gierige Algorithmen

### *Satz 36*

Die Lösung A, die von Algorithmus LatenessScheduling berechnet wird, hat optimale (d.h. minimale) maximale Verzögerung.

### *Beweis*

Aus dem ersten Lemma folgt, dass es ein optimales Schedule ohne Inversionen gibt. Aus dem zweiten Lemma folgt, dass alle Schedules ohne Inversionen dieselbe maximale Verzögerung haben. Damit ist die Lösung des gierigen Algorithmus optimal.

## Gierige Algorithmen

### *Zusammenfassung*

- Löse globales Optimierungsproblem durch lokale Optimierungsstrategie
- Liefert häufig recht einfache Algorithmen
- Funktioniert leider nicht immer und es ist manchmal nicht ganz einfach, die ‚richtige‘ Strategie zu finden

### *Algorithmische Entwurfsmethoden*

- Teile & Herrsche
- Dynamische Programmierung
- Gierige Algorithmen

## Datenstrukturen

### *Was ist eine Datenstruktur?*

- Eine Datenstruktur ist eine Anordnung von Daten, die effizienten Zugriff auf die Daten ermöglicht
- Datenstrukturen für viele unterschiedliche Anfragen vorstellbar

### *Ein grundlegendes Datenbank-Problem*

- Speicherung von Datensätzen

### *Beispiel*

- Kundendaten (Name, Adresse, Wohnort, Kundennummer, offene Rechnungen, offene Bestellungen,...)

### *Anforderungen*

- Schneller Zugriff
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

## Datenstrukturen

### *Zugriff auf Daten*

- Jedes Datum (Objekt) hat einen Schlüssel
- Eingabe des Schlüssels liefert Datensatz
- Schlüssel sind vergleichbar (es gibt totale Ordnung der Schlüssel)

### *Beispiel*

- Kundendaten (Name, Adresse, Kundennummer)
- Schlüssel: Name
- Totale Ordnung: Lexikographische Ordnung

### *Beispiel:*

- Kundendaten (Name, Adresse, Kundennummer)
- Schlüssel: Kundennummer
- Totale Ordnung:  $\leq$

## Datenstrukturen

### *Problem:*

- Gegeben sind  $n$  Objekte  $O_1, \dots, O_n$  mit zugehörigen Schlüsseln  $s(O_i)$

### *Operationen:*

- **Suche(x)**; Ausgabe  $O$  mit Schlüssel  $s(O) = x$ ;  
**nil**, falls kein Objekt mit Schlüssel  $x$  in Datenbank
- **Einfügen(O)**; Einfügen von Objekt  $O$  in Datenbank
- **Löschen(O)**; Löschen von Objekt  $O$  mit aus der Datenbank

## Datenstrukturen

### *Vereinfachung:*

- Schlüssel sind natürliche Zahlen
- Eingabe nur aus Schlüsseln

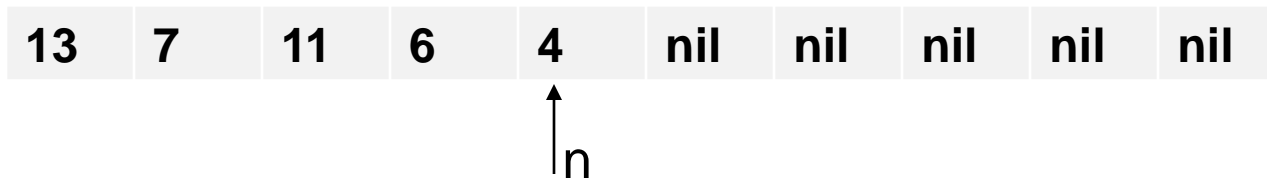
### *Analyse von Datenstrukturen*

- Platzbedarf in  $\Theta$ - bzw.  $O$ -Notation
- Laufzeit der Operationen in  $\Theta$ - bzw.  $O$ -Notation

## Datenstrukturen

### *Einfaches Feld*

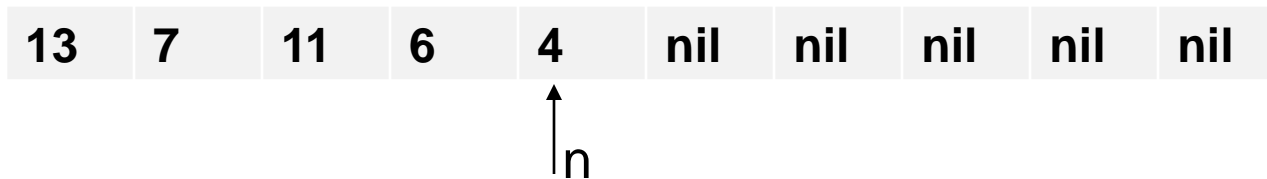
- Feld  $A[1, \dots, \max]$
- Integer  $n$ ,  $1 \leq n \leq \max$
- $n$  bezeichnet Anzahl Elemente in Datenstruktur



## Datenstrukturen

### *Einfügen(s)*

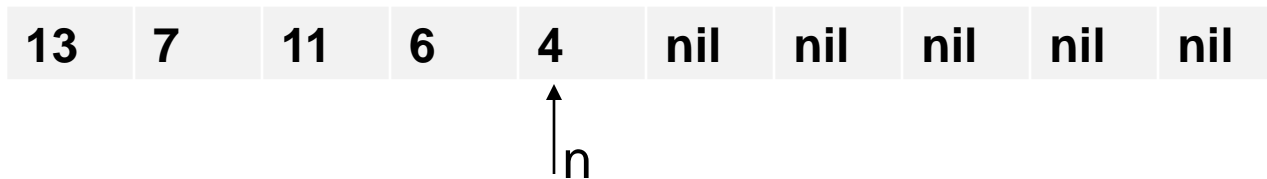
1. **if**  $n = \text{max}$  **then** Ausgabe „Fehler: Kein Platz in Datenstruktur“
2. **else**
3.  $n \leftarrow n + 1$
4.  $A[n] \leftarrow s$



## Datenstrukturen

### *Einfügen(s)*

1. **if**  $n = \text{max}$  **then** Ausgabe „Fehler: Kein Platz in Datenstruktur“
2. **else**
3.  $n \leftarrow n + 1$
4.  $A[n] \leftarrow s$

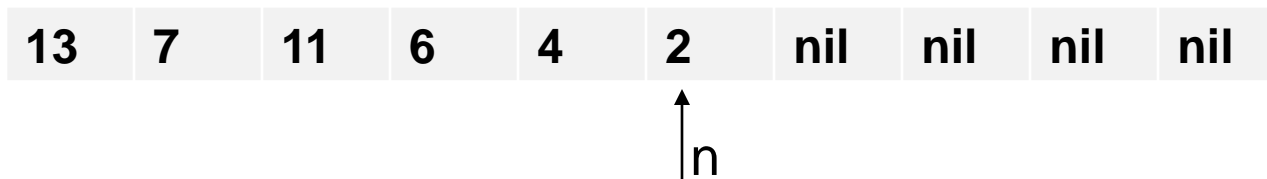


Einfügen(2)

## Datenstrukturen

### *Einfügen(s)*

1. **if**  $n = \text{max}$  **then** Ausgabe „Fehler: Kein Platz in Datenstruktur“
2. **else**
3.  $n \leftarrow n + 1$
4.  $A[n] \leftarrow s$

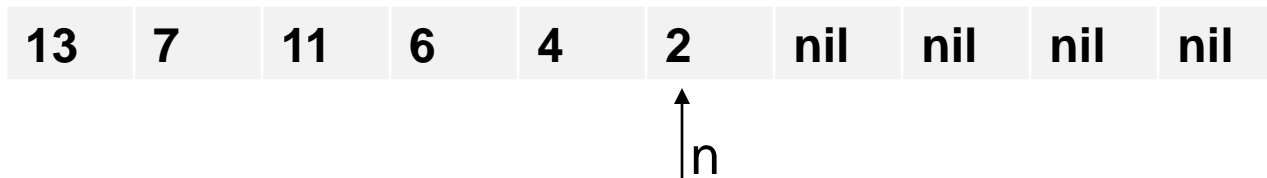


Einfügen(2)

## Datenstrukturen

### *Suche(x)*

1. **for**  $i \leftarrow 1$  **to**  $n$  **do**
2.     **if**  $A[i] = x$  **then return**  $i$
3. **return nil**

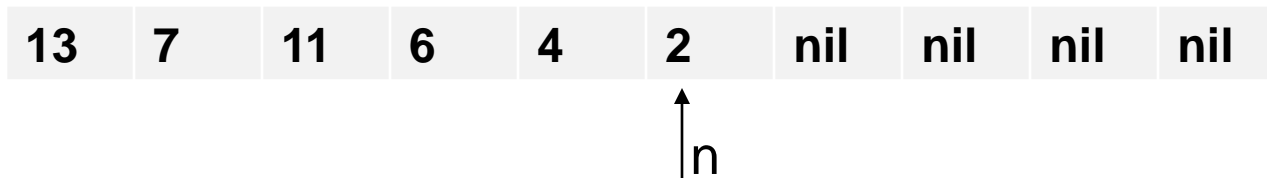


## Datenstrukturen

### Löschen(*i*)

1.  $A[i] \leftarrow A[n]$
2.  $A[n] \leftarrow \mathbf{nil}$
3.  $n \leftarrow n-1$

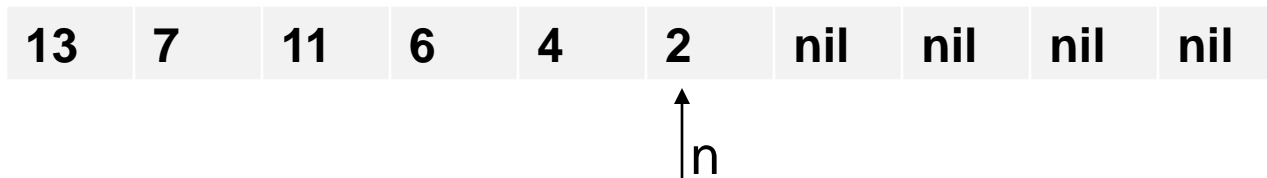
Annahme:  
Wir bekommen  
Index *i* des zu  
löschenden Objekts



## Datenstrukturen

### Löschen(*i*)

1.  $A[i] \leftarrow A[n]$
2.  $A[n] \leftarrow \mathbf{nil}$
3.  $n \leftarrow n-1$

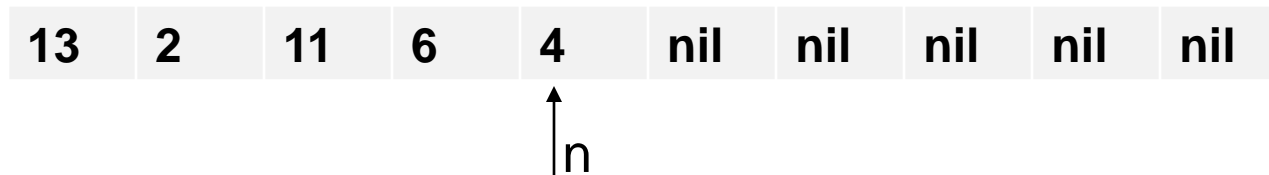


### Löschen(2)

## Datenstrukturen

### *Löschen(i)*

1.  $A[i] \leftarrow A[n]$
2.  $A[n] \leftarrow \mathbf{nil}$
3.  $n \leftarrow n-1$



### Löschen(2)

## Datenstrukturen

### *Datenstruktur Feld*

- Platzbedarf  $\Theta(\max)$
- Laufzeit Suche:  $\Theta(n)$
- Laufzeit Einfügen/Löschen:  $\Theta(1)$

### *Vorteile*

- Schnelles Einfügen und Löschen

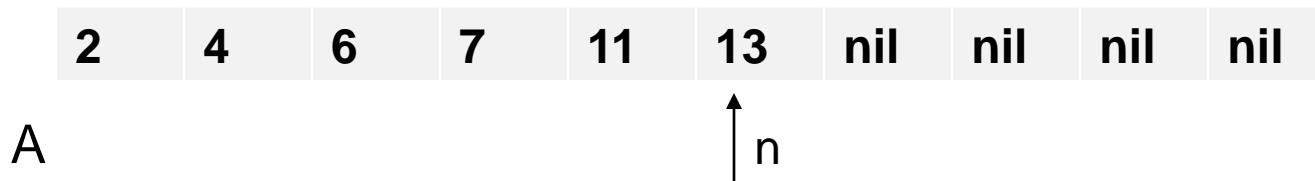
### *Nachteile*

- Speicherbedarf abhängig von max (nicht vorhersagbar)
- Hohe Laufzeit für Suche

## Datenstrukturen

### *Datenstruktur „sortiertes Feld“*

- **Sortiertes** Feld  $A[1, \dots, \max]$
- Integer  $n$ ,  $1 \leq n \leq \max$
- $n$  bezeichnet Anzahl Elemente in Datenstruktur



## Datenstrukturen (siehe Vollversion)

### *Einfügen(s)*

1.  $n \leftarrow n+1$
2.  $i \leftarrow n$
3. **while**  $s < A[i-1]$  **do**
4.      $A[i] \leftarrow A[i-1]$
5.      $i \leftarrow i-1$
6.  $A[i] \leftarrow s$

Laufzeit  $O(n)$



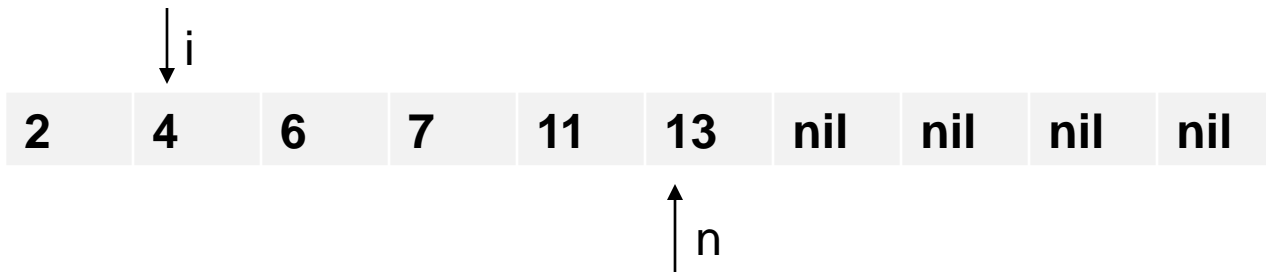
Einfügen(10)

## Datenstrukturen

### Löschen(*i*)

1. **for**  $j \leftarrow i$  **to**  $n-1$  **do**
2.      $A[j] \leftarrow A[j+1]$
3.  $A[n] \leftarrow \text{nil}$
4.  $n \leftarrow n-1$

Parameter ist der Index des zu löschenden Objekts

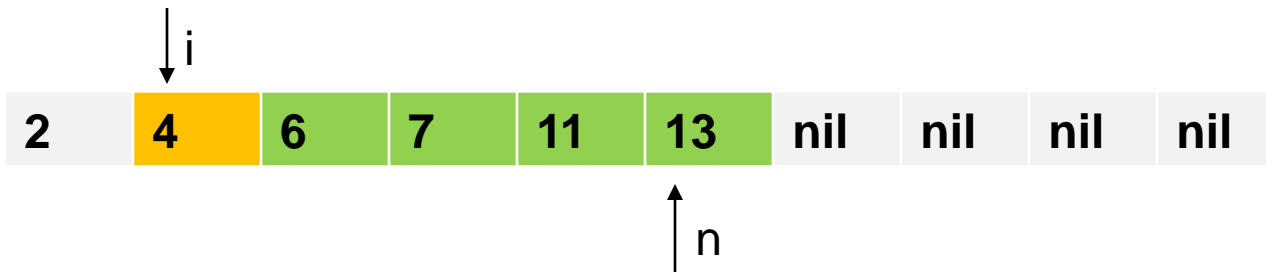


Löschen(2)

## Datenstrukturen

### Löschen(*i*)

1. **for**  $j \leftarrow i$  **to**  $n-1$  **do**
2.      $A[j] \leftarrow A[j+1]$
3.  $A[n] \leftarrow \text{nil}$
4.  $n \leftarrow n-1$

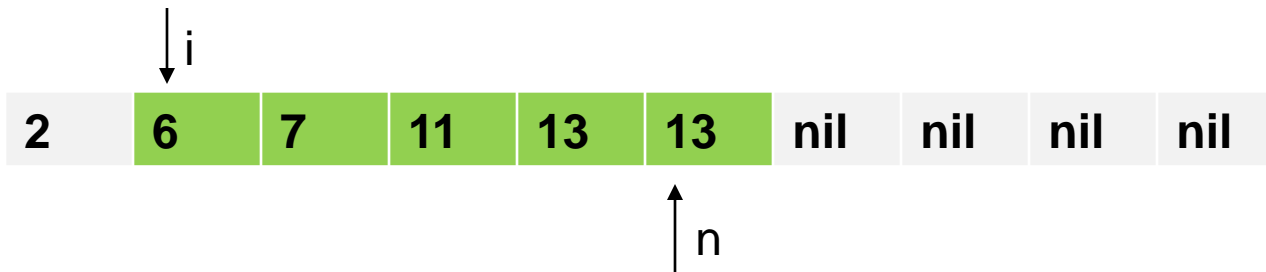


Löschen(2)

## Datenstrukturen

### Löschen(*i*)

1. **for**  $j \leftarrow i$  **to**  $n-1$  **do**
2.      $A[j] \leftarrow A[j+1]$
3.  $A[n] \leftarrow \text{nil}$
4.  $n \leftarrow n-1$

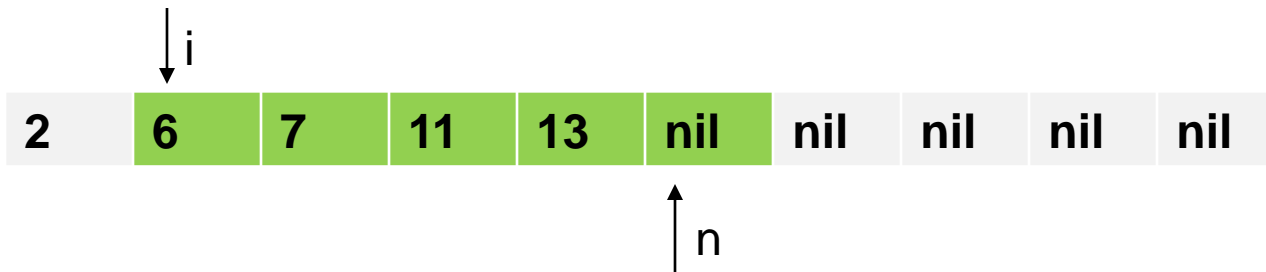


Löschen(2)

## Datenstrukturen

### Löschen(*i*)

1. **for**  $j \leftarrow i$  **to**  $n-1$  **do**
2.      $A[j] \leftarrow A[j+1]$
3.      $A[n] \leftarrow \text{nil}$
4.      $n \leftarrow n-1$

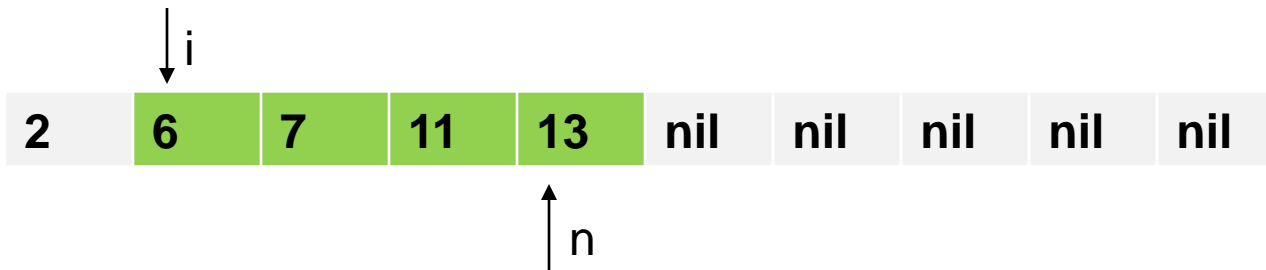


Löschen(2)

## Datenstrukturen

### Löschen(*i*)

1. **for**  $j \leftarrow i$  **to**  $n-1$  **do**
2.      $A[j] \leftarrow A[j+1]$
3.  $A[n] \leftarrow \text{nil}$
4.  $n \leftarrow n-1$

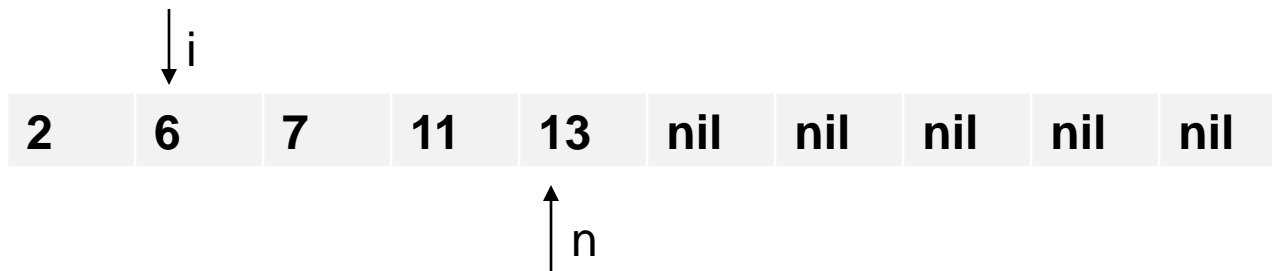


Löschen(2)

## Datenstrukturen

### Suchen(x)

- Binäre Suche
- Laufzeit  $O(\log n)$



Löschen(2)

## Datenstrukturen

### *Datenstruktur sortiertes Feld*

- Platzbedarf  $\Theta(\max)$
- Laufzeit Suche:  $\Theta(\log n)$
- Laufzeit Einfügen/Löschen:  $\Theta(n)$

### *Vorteile*

- Schnelles Suchen

### *Nachteile*

- Speicherbedarf abhängig von  $\max$  (nicht vorhersagbar)
- Hohe Laufzeit für Einfügen/Löschen