RUB

Vorkurs Informatik

Algorithmen zum Suchen

Maike Buchin, Michael Walter

20.9.2024

kurze Vorstellung

RUB

Prof. Dr. Maike Buchin
Lehrstuhl Algorithmik

Vorlesung Informatik 2 und
weitere Vorlesungen zu Algorithmen

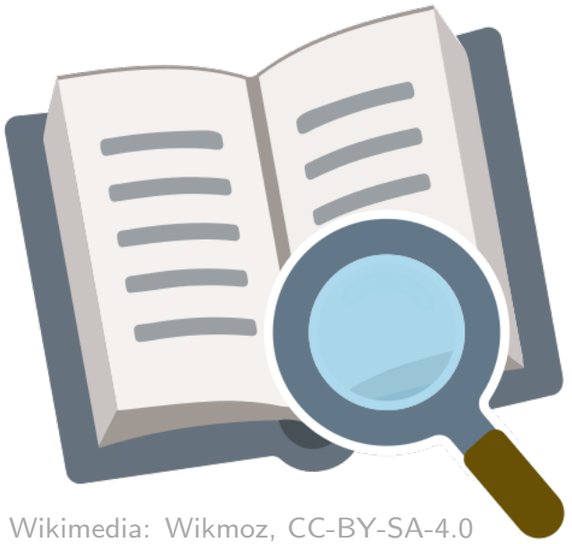
Prof. Dr. Michael Walter
Lehrstuhl Quanteninformationen

Vorlesungen zu Quantencomputing
und Theoretischer Informatik

Motivation

häufig wird in großen Datenmengen nach Einträgen gesucht, z.B.

- wir suchen nach jemandem in der Liste der Teilnehmer:innen des Vorkurs
- wir suchen nach jemandem in der Liste aller Studierenden der Fakultät
- in einer Mitarbeiterliste, Inventarliste, ...



Motivation

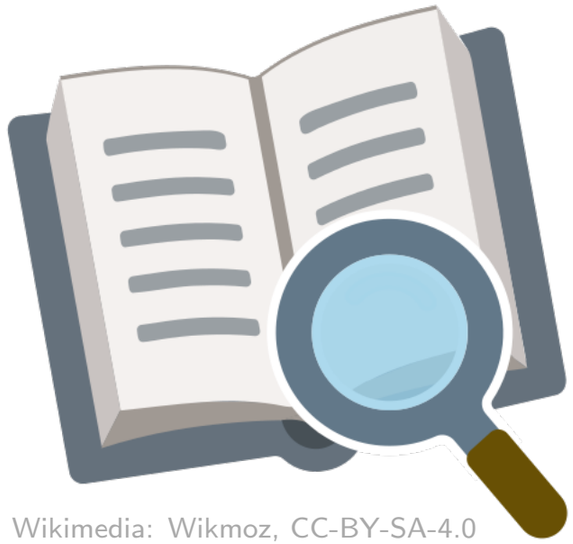
RUB

häufig wird in großen Datenmengen nach Einträgen gesucht, z.B.

- wir suchen nach jemandem in der Liste der Teilnehmer:innen des Vorkurs
- wir suchen nach jemandem in der Liste aller Studierenden der Fakultät
- in einer Mitarbeiterliste, Inventarliste, ...

Fragen:

- wie lösen wir das Problem möglichst effizient?
- wie zeigen wir, das unsere Lösung immer funktioniert?
- wie sind die Daten gegeben?
- welche Datenstrukturen nutzen wir?
- gibt es noch schnellere Algorithmen?



Motivation

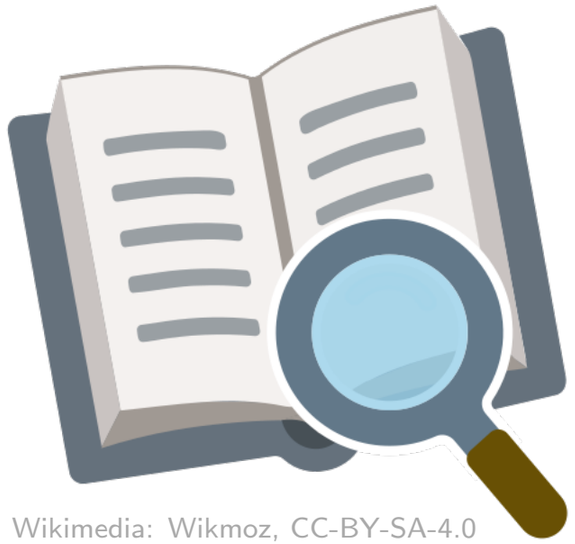
RUB

häufig wird in großen Datenmengen nach Einträgen gesucht, z.B.

- wir suchen nach jemandem in der Liste der Teilnehmer:innen des Vorkurs
- wir suchen nach jemandem in der Liste aller Studierenden der Fakultät
- in einer Mitarbeiterliste, Inventarliste, ...

Fragen:

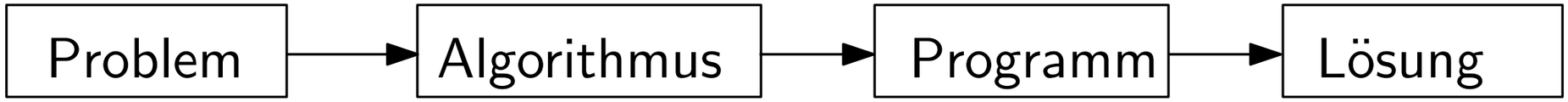
- wie lösen wir das Problem möglichst effizient?
- wie zeigen wir, das unsere Lösung immer funktioniert?
- wie sind die Daten gegeben?
- welche Datenstrukturen nutzen wir?
- gibt es noch schnellere Algorithmen?



mit solchen Fragen beschäftigt sich die **Theoretische Informatik**

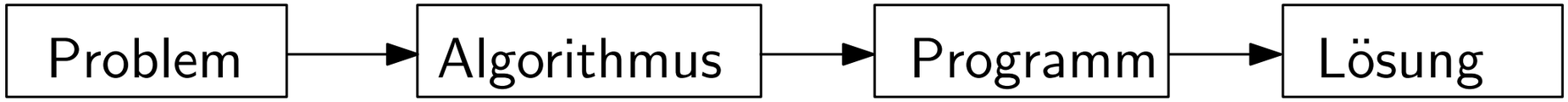
Algorithmen

RUB



Algorithmen

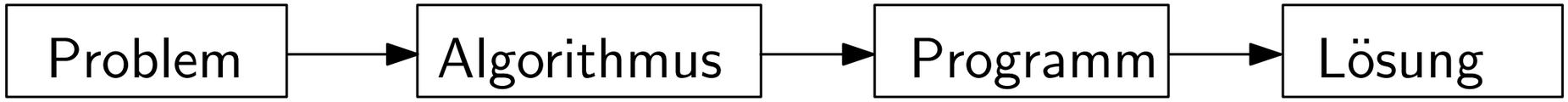
RUB



- Welche Eigenschaften sollten Algorithmen haben?

Algorithmen

RUB



- Welche Eigenschaften sollten Algorithmen haben?

  → Korrektheit
  → Laufzeit
  → und viele mehr

Algorithmen zum Suchen

RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Algorithmen zum Suchen

RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

A

17	5	25	20	14	2	19	12	22	9
1									n

Enthält das Array A die Zahl 23?

Algorithmen zum Suchen

RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

A

17	5	25	20	14	2	19	12	22	9
1									n

Enthält das Array A die Zahl 23?

Algorithmus?

Algorithmen zum Suchen

RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

A

17	5	25	20	14	2	19	12	22	9
1									n

Enthält das Array A die Zahl 23?

Algorithmus: Lineare Suche

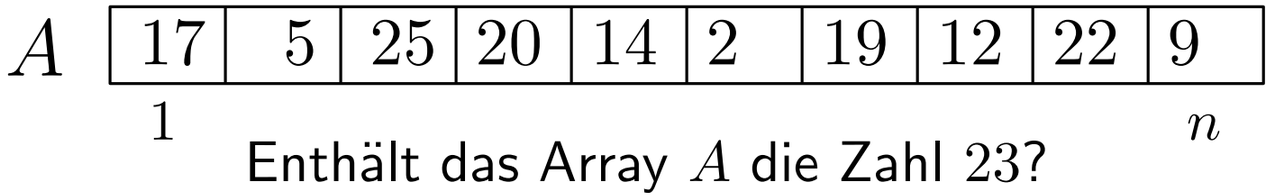
Lineare Suche

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



Enthält das Array A die Zahl 23?

linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Lineare Suche

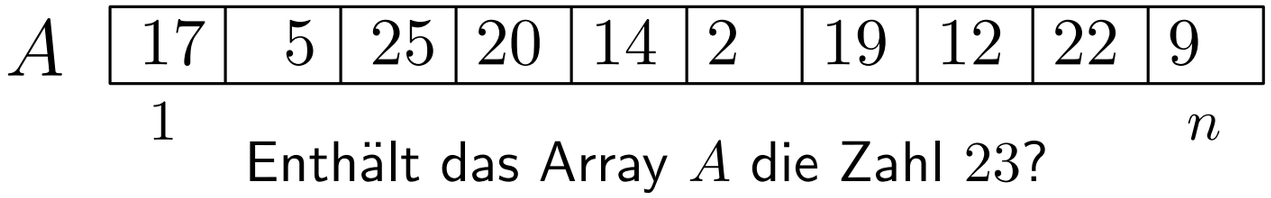
RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



Enthält das Array A die Zahl 23?

linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: ?

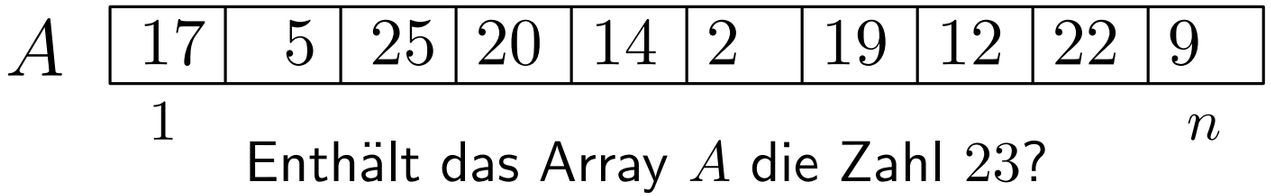
Lineare Suche

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: klar (?), denn x wird mit allen Einträgen in A verglichen

Lineare Suche

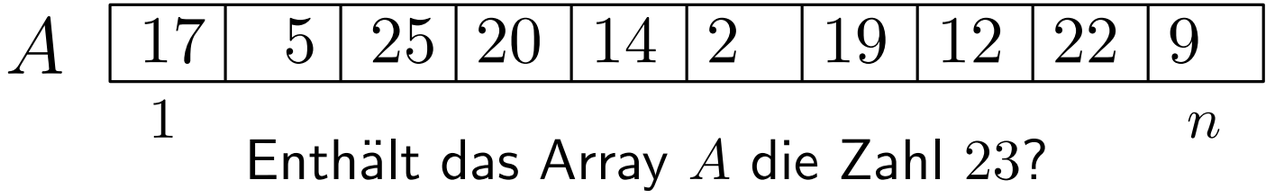
RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



Enthält das Array A die Zahl 23?

linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: klar (?), denn x wird mit allen Einträgen in A verglichen

Laufzeit: ?

Lineare Suche

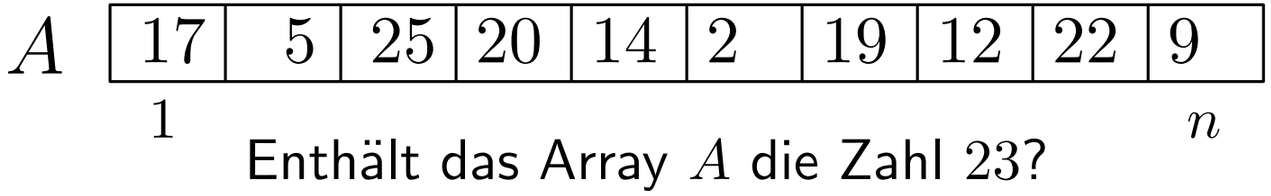
RUB

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



Enthält das Array A die Zahl 23?

linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: klar (?), denn x wird mit allen Einträgen in A verglichen

Laufzeit: benötigt n Vergleiche

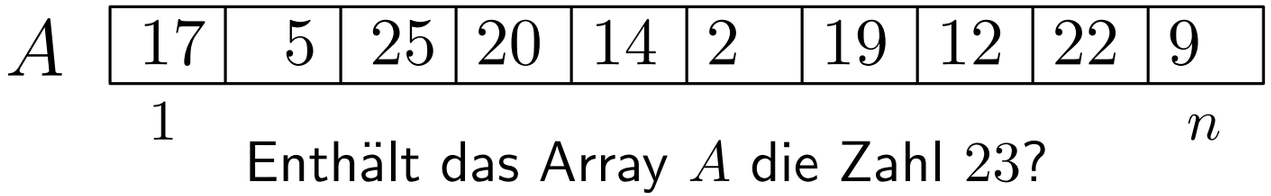
Lineare Suche

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



Enthält das Array A die Zahl 23?

linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: klar (?), denn x wird mit allen Einträgen in A verglichen

Laufzeit: benötigt n Vergleiche

Frage: Geht es nicht auch schneller?

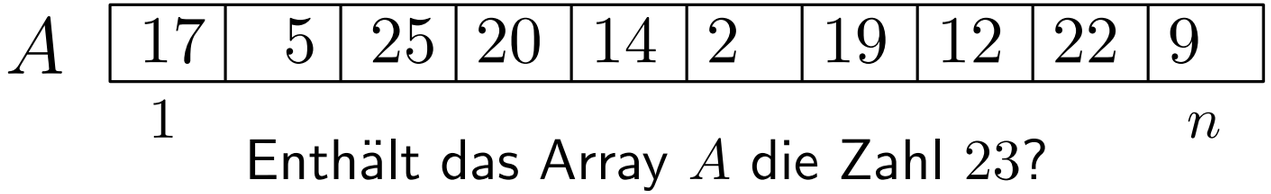
Lineare Suche

Problem:

Gegeben: ein Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:



linSearch$(A[1 \ldots n], x)$

```
for i ← 1 to n do
    if A[i] = x then return "A[i] = x"
return "x ∉ A[1...n]"
```

Korrektheit: klar (?), denn x wird mit allen Einträgen in A verglichen

Laufzeit: benötigt n Vergleiche

Frage: Geht es nicht auch schneller? Ja, wenn die Eingabe sortiert ist!

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

A

2	5	9	12	14	17	19	20	22	25
1									n

Enthält das Array A die Zahl 23?

Frage: Wie lässt sich in einem sortierten Array schneller suchen?

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

A

2	5	9	12	14	17	19	20	22	25
1									n

Enthält das Array A die Zahl 23?

Idee: Vergleiche x mit mittlerem Element in A,
falls x kleiner als dieses, suche weiter links davon, andernfalls rechts

Binäre Suche

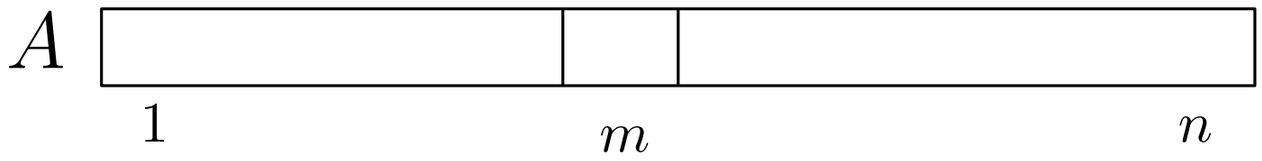
RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Algorithmus:



Vergleiche x und $A[m]$ mit $\lfloor m = (n+1)/2 \rfloor$

- falls $x = A[m] \rightarrow$ gib m zurück
- falls $x < A[m] \rightarrow$ suche weiter in $A[1 \dots m-1]$
- falls $x > A[m] \rightarrow$ suche weiter in $A[m+1 \dots n]$

Binäre Suche

RUB

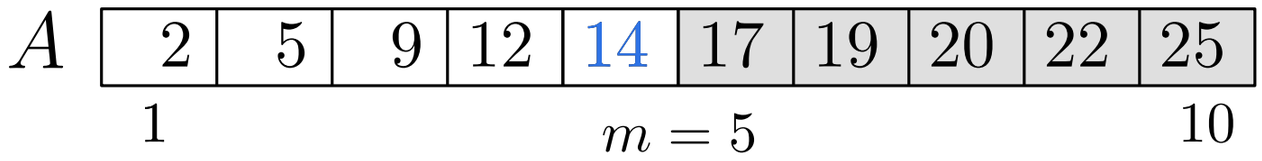
Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

Enthält a die 23?



Vergleiche x und $A[m]$ mit $\lfloor m = (n+1)/2 \rfloor$

- falls $x = A[m] \rightarrow$ gib m zurück
- falls $x < A[m] \rightarrow$ suche weiter in $A[1 \dots m-1]$
- falls $x > A[m] \rightarrow$ suche weiter in $A[m+1 \dots n]$

Binäre Suche

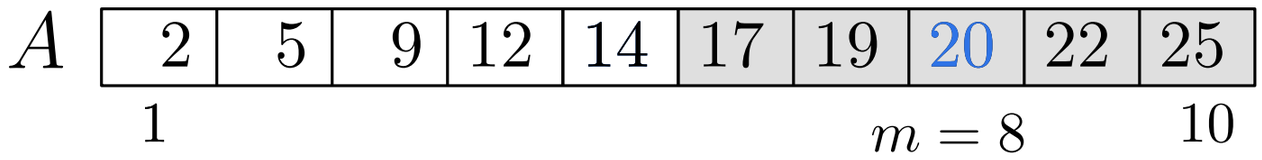
Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

Enthält a die 23?



Vergleiche x und $A[m]$ mit $\lfloor m = (n+1)/2 \rfloor$

- falls $x = A[m] \rightarrow$ gib m zurück
- falls $x < A[m] \rightarrow$ suche weiter in $A[1 \dots m-1]$
- falls $x > A[m] \rightarrow$ suche weiter in $A[m+1 \dots n]$

Binäre Suche

RUB

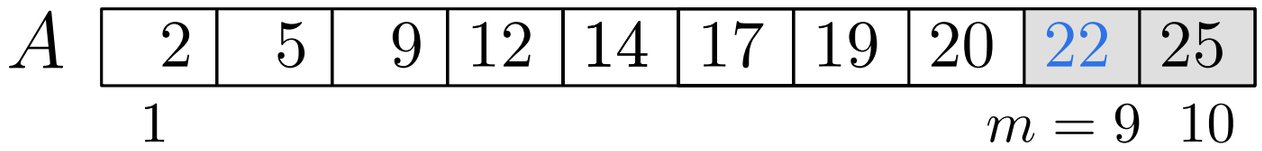
Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

Enthält a die 23?



Vergleiche x und $A[m]$ mit $\lfloor m = (n+1)/2 \rfloor$

- falls $x = A[m] \rightarrow$ gib m zurück
- falls $x < A[m] \rightarrow$ suche weiter in $A[1 \dots m-1]$
- falls $x > A[m] \rightarrow$ suche weiter in $A[m+1 \dots n]$

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Beispiel:

Enthält a die 23?

A	2	5	9	12	14	17	19	20	22	25

Vergleiche x und $A[m]$ mit $\lfloor m = (n+1)/2 \rfloor$

- falls $x = A[m] \rightarrow$ gib m zurück
- falls $x < A[m] \rightarrow$ suche weiter in $A[1 \dots m-1]$
- falls $x > A[m] \rightarrow$ suche weiter in $A[m+1 \dots n]$

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

binSearch$(A[1 \dots n], x)$

$\ell, r \leftarrow 0, n+1$

`while true do`

  `if` $\ell + 1 = r$ `then return` "$A[\ell] < x < A[\ell+1]$"

  $m \leftarrow \lfloor (r+\ell)/2 \rfloor$

  `if` $A[m] = x$ `then return` "$A[m] = x$"

  `if` $A[m] > x$ `then` $r \leftarrow m$ `else` $\ell \leftarrow m$

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

binSearch$(A[1 \dots n], x)$

$\ell, r \leftarrow 0, n+1$

`while true do`

  `if` $\ell + 1 = r$ `then return` "$A[\ell] < x < A[\ell+1]$"

  $m \leftarrow \lfloor (r+\ell)/2 \rfloor$

  `if` $A[m] = x$ `then return` "$A[m] = x$"

  `if` $A[m] > x$ `then` $r \leftarrow m$ `else` $\ell \leftarrow m$

Korrektheit: ?

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

binSearch$(A[1 \dots n], x)$

$\ell, r \leftarrow 0, n+1$

`while true do`

  `if` $\ell + 1 = r$ `then return` "$A[\ell] < x < A[\ell+1]$"

  $m \leftarrow \lfloor (r+\ell)/2 \rfloor$

  `if` $A[m] = x$ `then return` "$A[m] = x$"

  `if` $A[m] > x$ `then` $r \leftarrow m$ `else` $\ell \leftarrow m$

Korrektheit: folgt aus der Invariante: gesuchter Wert liegt zwischen $A[\ell]$ und $A[r]$, also $A[\ell] < x < A[r]$ mit $A[0] = -\infty$ und $A[n+1] = \infty$

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

binSearch$(A[1 \dots n], x)$

$\ell, r \leftarrow 0, n+1$
`while true do`
  `if` $\ell + 1 = r$ `then return` "$A[\ell] < x < A[\ell+1]$"
  $m \leftarrow \lfloor (r+\ell)/2 \rfloor$
  `if` $A[m] = x$ `then return` "$A[m] = x$"
  `if` $A[m] > x$ `then` $r \leftarrow m$ `else` $\ell \leftarrow m$

Laufzeit: ?

Binäre Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

binSearch$(A[1 \ldots n], x)$

$\ell, r \leftarrow 0, n+1$
`while true do`
  `if` $\ell + 1 = r$ `then return` "$A[\ell] < x < A[\ell+1]$"
  $m \leftarrow \lfloor (r+\ell)/2 \rfloor$
  `if` $A[m] = x$ `then return` "$A[m] = x$"
  `if` $A[m] > x$ `then` $r \leftarrow m$ `else` $\ell \leftarrow m$

Laufzeit: höchstens $\log_2 n$ Vergleiche, denn

- in jedem Durchlauf (außer dem letzten) halbiert sich die Größe des zu durchsuchenden Arrays
- ein Durchlauf benötigt konstante Zeit

Laufzeitvergleich

RUB

Wieviel schneller ist die binäre Suche als die lineare Suche?

n	$\log_2 n$
8	
1024	?
$1.073.741.824$	

Laufzeitvergleich

RUB

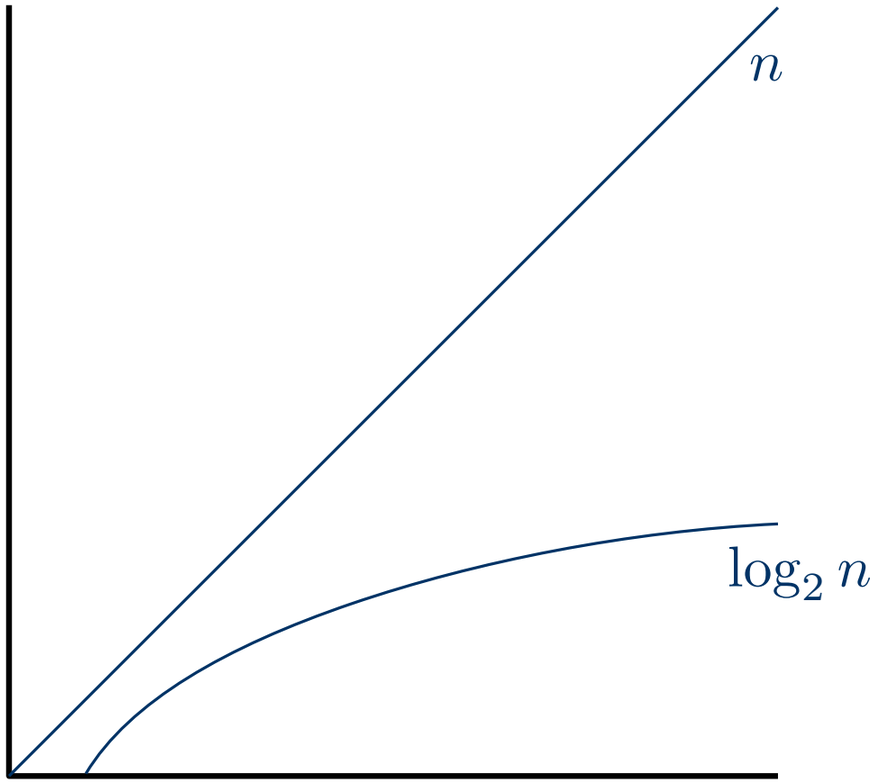
Wieviel schneller ist die binäre Suche als die lineare Suche?

n	$\log_2 n$
$8 = 2^3$	
$1024 = 2^{10}$	?
$1.073.741.824 = 2^{30}$	

Laufzeitvergleich

Wieviel schneller ist die binäre Suche als die lineare Suche?

n	$\log_2 n$
$8 = 2^3$	3
$1024 = 2^{10}$	10
$1.073.741.824 = 2^{30}$	30



Exponentielle Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \ldots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Frage: Was machen wir, wenn das Array unbeschränkt oder seine Größe unbekannt ist?

Beispiel:

A

2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
1																

Enthält das Array A die Zahl 23?

Exponentielle Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

Exponentielle Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

expSearch($A[1 \dots], x$)

$k \leftarrow 1$

`while` $k <$ `size and` $A[k] < x$ `do`

  $k \leftarrow 2k$

`return` binSearch$(A, \frac{k}{2}, \min(k, \text{size}), x)$

Exponentielle Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

expSearch$(A[1 \dots], x)$

$k \leftarrow 1$

`while` $k <$ `size and` $A[k] < x$ `do`

  $k \leftarrow 2k$

`return` binSearch$(A, \frac{k}{2}, \min(k, \text{size}), x)$

Korrektheit: ?

Exponentielle Suche

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

expSearch$(A[1 \dots], x)$

$k \leftarrow 1$

`while` $k <$ `size and` $A[k] < x$ `do`

  $k \leftarrow 2k$

`return` binSearch$(A, \frac{k}{2}, \min(k, \text{size}), x)$

Korrektheit: durch Verdoppeln erreichen wir alle Einträge, dann Korrektheit von binärer Suche auf beschränktem Bereich

Exponentielle Suche

RUB

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

expSearch$(A[1 \dots], x)$

$k \leftarrow 1$

`while` $k <$ `size and` $A[k] < x$ `do`

  $k \leftarrow 2k$

`return` binSearch$(A, \frac{k}{2}, \min(k, \text{size}), x)$

Korrektheit: durch Verdoppeln erreichen wir alle Einträge, dann Korrektheit von binärer Suche auf beschränktem Bereich

Laufzeit: ?

Exponentielle Suche

Problem:

Gegeben: ein geordnetes Array $A[1 \dots n]$ mit paarweise verschiedenen Einträgen, d.h. $A[1] < \cdots < A[n]$, wobei $n = \infty$ oder unbekannt, sowie ein Element x.

Gefragt: Ist x in A? Wenn ja, für welchen Index k gilt $A[k] = x$?

Idee: Wir verdoppeln das Suchintervall bis wir ein beschränktes Intervall haben, dann nutzen wir binäre Suche.

Beispiel:

A	2	5	9	12	14	17	19	20	22	25	32	43	44	51	58	62	…
	1																

Enthält das Array A die Zahl 23?

expSearch$(A[1 \dots], x)$

$k \leftarrow 1$

`while` $k <$ `size and` $A[k] < x$ `do`

  $k \leftarrow 2k$

`return` binSearch$(A, \frac{k}{2}, \min(k, \text{size}), x)$

Korrektheit: durch Verdoppeln erreichen wir alle Einträge, dann Korrektheit von binärer Suche auf beschränktem Bereich

Laufzeit: Sei k der größte Index mit $A[k] \leq x$.
Dann benötigt die Suche höchstens $2 \log_2 k$ Vergleiche

Geht es schneller?

RUB

Ausblick: Suche auf Graphen

RUB

Zusammenfassung

RUB

Algorithmen:

- Problemstellung
- Korrektheit
- Laufzeit

Suchen in sortiertem Array

- schneller mit binärer Suche als mit linearer Suche
- exponentielle Suche bei unbeschränktem Array