Java

```
public D find(K k) {
    Listnode⟨K, D⟩ n = findnode(k);
    if(n == null) return null;
    return n.data;
}
```

Java

```
protected Listnode⟨K, D⟩ findnode(K k) {
    Listnode⟨K, D⟩ n;
    head.key = k;
    for(n = head.succ; !n.key.equals(k); n = n.succ) { }
    head.key = null;
    if(n == head) return null;
    return n;
}
```

Einfach verkettete Listen

- Kein Zeiger auf Vorgänger
- Einfachere Datenstruktur
- Operationen können komplizierter sein

Frage: Wie kann ein Element *vor* einen gegebenen Knoten eingefügt werden?

Ersetzen und alten Knoten anfügen!

Vorsicht: Eine gefährliche Technik.

Einfach verkettete Listen

- Kein Zeiger auf Vorgänger
- Einfachere Datenstruktur
- Operationen können komplizierter sein

Frage: Wie kann ein Element *vor* einen gegebenen Knoten eingefügt werden?

Ersetzen und alten Knoten anfügen!

Vorsicht: Eine gefährliche Technik.

Listen – Laufzeit

Manche Operation sind schnell, manche langsam...

Operation	Laufzeit
append()	$\Theta(1)$
delete()*	$\Theta(1)$
delete()	$\Theta(n)$
find()	$\Theta(n)$
insert()	$\Theta(n)$

* direktes Löschen eines Knotens, nicht über Schlüssel

Java

```
public class Queue⟨D⟩ {
  private ADList⟨Object, D⟩ queue;
  private int size;
  public Queue() { queue = new ADList⟨Object, D⟩(); size = 0; }
  public D dequeue() {
    D x = queue.lastnode().getData();
    queue.lastnode().delete();
    size−−;
    return x;
  }
  public void enqueue(D x) { queue.prepend(null, x); size++; }
  public int size() { return size; }
}
```

Übersicht

Übersicht

Lineare Suche

Wir suchen x im Array $a[0, \ldots, n-1]$

Algorithmus

```
function find1(int x) boolean :
for i = 0 to n − 1 do
  if x = a[i] then return true fi
od;
return false
```

Die innere Schleife ist langsam.

Lineare Suche – verbessert

Wir verwenden ein *Sentinel*-Element am Ende.

Das Element $a[n]$ muß existieren und unbenutzt sein.

Algorithmus

function find2(**int** x) **boolean** :
i := 0;
a[n] := x;
while a[i] $\neq$ x **do** i := i + 1 **od**;
return i < n

Die innere Schleife ist sehr schnell.

Lineare Suche – ohne zusätzlichen Platz

Algorithmus

function find3(**int** x) **boolean** :
if $x = a[n-1]$ **then return** true **fi**;
$temp := a[n-1]$;
$a[n-1] := x$;
$i := 0$;
while $a[i] \neq x$ **do** $i := i + 1$ **od**;
$a[n-1] := temp$;
return $i < n-1$

Erheblicher Zusatzaufwand.

Innere Schleife immer noch sehr schnell.

Lineare Suche – Analyse

Wieviele Vergleiche benötigt eine erfolgreiche Suche nach x im Mittel?

Wir nehmen an, daß jedes der n Elemente mit gleicher Wahrscheinlichkeit gesucht wird.

Es gilt $\Pr[x = a[i]] = 1/n$ für $i \in \{0, \ldots, n-1\}$.

Sei C die Anzahl der Vergleiche:
$C = i + 1$ gdw. $x = a[i]$

Lineare Suche – Analyse

Wieviele Vergleiche benötigt eine erfolgreiche Suche nach x im Mittel?

Wir nehmen an, daß jedes der n Elemente mit gleicher Wahrscheinlichkeit gesucht wird.

Es gilt $\Pr[x = a[i]] = 1/n$ für $i \in \{0, \ldots, n-1\}$.

Sei C die Anzahl der Vergleiche:
$C = i + 1$ gdw. $x = a[i]$

Lineare Suche – Analyse

Wir suchen den Erwartungswert von C.

$$\Pr[C = i] = 1/n \text{ für } i = 1, \ldots, n$$

$$E(C) = \sum_{k=1}^{n} k \Pr[C = k] = \sum_{k=1}^{n} \frac{k}{n} = \frac{n+1}{2}$$

Es sind im Mittel $(n+1)/2$ Vergleiche.

Lineare Suche – Analyse

Wir suchen den Erwartungswert von C.

$$\Pr[C = i] = 1/n \text{ für } i = 1, \ldots, n$$

$$E(C) = \sum_{k=1}^{n} k \Pr[C = k] = \sum_{k=1}^{n} \frac{k}{n} = \frac{n+1}{2}$$

Es sind im Mittel $(n+1)/2$ Vergleiche.

Lineare Suche – Analyse

Unser Ergebnis:

Es sind im Mittel $(n+1)/2$ Vergleiche.

Ist das Ergebnis richtig?

Wir überprüfen das Ergebnis für kleine n.

- $n = 1$?
- $n = 2$?

Lineare Suche – Analyse

Unser Ergebnis:

Es sind im Mittel $(n+1)/2$ Vergleiche.

Ist das Ergebnis richtig?

Wir überprüfen das Ergebnis für kleine n.

- $n = 1$?
- $n = 2$?

Binäre Suche

Wir suchen wieder x in $a[0, \ldots, n-1]$.

Algorithmus

```
function binsearch(int x) boolean :
l := 0;  r := n − 1;
while l ≤ r do
  m := ⌊(l + r)/2⌋;
  if a[m] < x then l := m + 1 fi;
  if a[m] > x then r := m − 1 fi;
  if a[m] = x then return true fi
od;
return false
```

Wir halbieren den Suchraum in jedem Durchlauf.

Binäre Suche

Java

```
static public boolean binsearch(int x, int[] a) {
  int l = 0, r = a.length − 1, m, c;
  while(l ≤ r) {
    m = (l + r)/2;
    if(x == a[m]) return true;
    if(x < a[m]) r = m − 1;
    else l = m + 1;
  }
  return false;
}
```