

Gegeben: Array A ganzer Zahlen; Element x

Gesucht: Wo kommt x in A vor?

Naives Vorgehen:

- ▶ Vergleiche x der Reihe nach mit $A[0]$, $A[1]$, usw.
- ▶ Finden wir i mit $A[i] == x$, geben wir i aus.
- ▶ Andernfalls geben wir -1 aus: „Element nicht gefunden“!

Naives Suchen

```
1 Input: Array A mit Laenge n; Element x
2 Output: i mit A[i] == x falls existent
3         sonst -1
4
5 find(A,x)
6     i = 0;
7     while (i < n && A[i] != x)
8         i++;
9     if (i == n)
10        return -1;
11    else
12        return i;
```

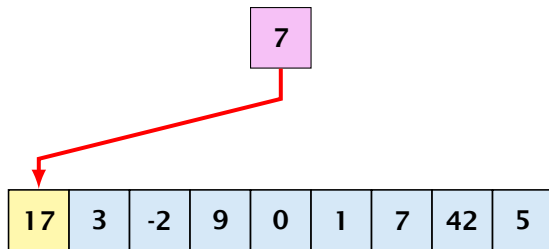
Naives Suchen

Beispiel

7

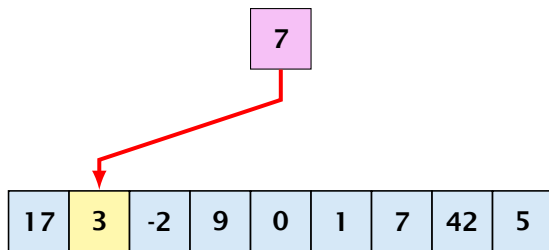
17	3	-2	9	0	1	7	42	5
----	---	----	---	---	---	---	----	---

Beispiel



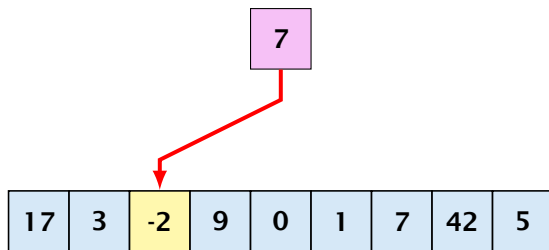
no

Beispiel



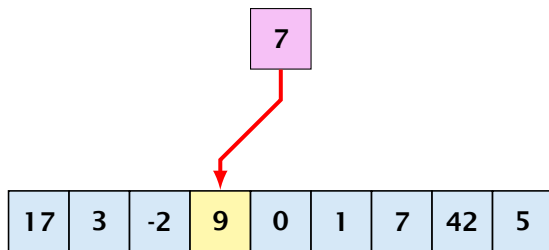
no

Beispiel



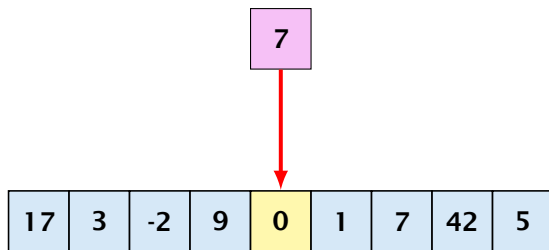
no

Beispiel



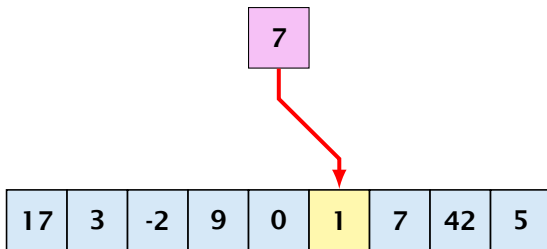
no

Beispiel



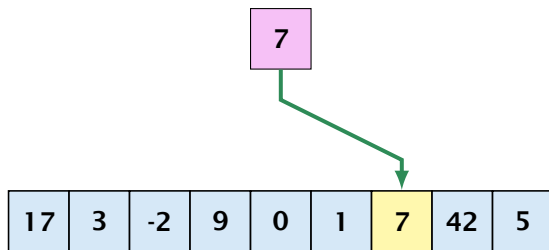
no

Beispiel



no

Beispiel



yes

Laufzeit Naives Suchen

Best-case:

Wenn x an Position 0.

⇒ Laufzeit: $\mathcal{O}(1)$.

Worst-case:

Wenn x nicht vorkommt.

⇒ Laufzeit: $\mathcal{O}(n)$.

...geht das besser?

Binäre Suche

Annahme: Input ist sortiert.

Idee:

- ▶ Vergleiche x mit dem Wert, der in der Mitte steht.
- ▶ Liegt Gleichheit vor, sind wir fertig.
- ▶ Ist x kleiner, brauchen wir nur noch links weitersuchen.
- ▶ Ist x größer, brauchen wir nur noch rechts weiter suchen.

⇒ binäre Suche

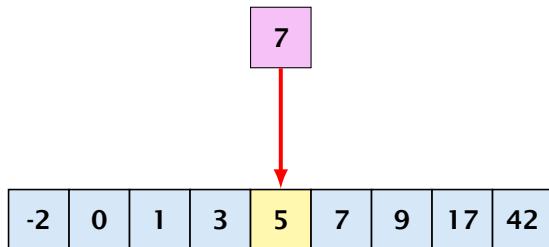
Beispiel

7

-2	0	1	3	5	7	9	17	42
----	---	---	---	---	---	---	----	----

► wir benötigen nur **drei** Vergleiche

Beispiel



no

► wir benötigen nur drei Vergleiche

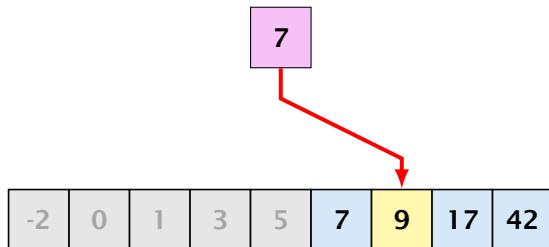
Beispiel

7

-2	0	1	3	5	7	9	17	42
----	---	---	---	---	---	---	----	----

► wir benötigen nur **drei** Vergleiche

Beispiel



no

► wir benötigen nur drei Vergleiche

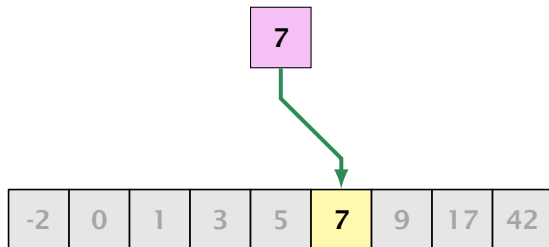
Beispiel

7

-2	0	1	3	5	7	9	17	42
----	---	---	---	---	---	---	----	----

► wir benötigen nur **drei** Vergleiche

Beispiel



yes

- ▶ wir benötigen nur **drei** Vergleiche

Implementierung

```
1 Input: sortiertes Array A; Element x;
2         linker Index n1; rechter Index nr; n1<=nr
3 Output: Index i; n1 ≤ i ≤ nr mit A[i]==x falls existent
4         sonst -1
5 find(A, x, n1, nr) // Inputlänge ist n=nr-n1+1
6     t = (n1 + nr) / 2;
7     if (A[t] == x)
8         return t;
9     if (n1 == nr)
10        return -1;
11    if (x > A[t])
12        return find(A, x, t+1, nr);
13    if (n1 < t)
14        return find(A, x, n1, t-1);
15    return -1;
```

Laufzeit Binäre Suche

Laufzeit:

$$T(n) = \begin{cases} \mathcal{O}(1) & A[t] == x \\ \mathcal{O}(1) & nl == nr \\ \mathcal{O}(1) + T(nr - t) & x > A[t] \\ \mathcal{O}(1) + T(t - nl) & nl < t \end{cases}$$

oder

$$T(n) \leq \begin{cases} \mathcal{O}(1) & n = 1 \\ \mathcal{O}(1) + T(\lfloor n/2 \rfloor) & \text{sonst} \end{cases}$$

Laufzeit Binäre Suche

Lösen der Rekursionsgleichung:

$$T(n) \leq \begin{cases} \mathcal{O}(1) & n = 1 \\ \mathcal{O}(1) + T(\lfloor n/2 \rfloor) & \text{sonst} \end{cases}$$

Üblicherweise nur für $n = 2^k$; z.B. durch vollständige Induktion.