

Gegeben: Ein sortiertes Array **a** mit den **n** Komponenten **a[0], . . . , a[n-1]**

Aufgabe: Entscheide, ob ein für die Variable **value** eingegebener Wert als Wert einer Komponente des Arrays **a** vorkommt.

Beispiel

value = 13

n = len(a) = 10

Wir übergeben **value** und die Liste **a[0], . . . , a[9]** der Booleschen Funktion **binarysearch**, welche **a[0], . . . , a[9]** als lokale Liste **array[0], . . . , array[9]** fortführt.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
3	4	5	5	7	8	11	13	19	21

array[0]	array [1]	array [2]	array [3]	array [4]	array [5]	array [6]	array [7]	array [8]	array [9]
3	4	5	5	7	8	11	13	19	21

1. Schritt:

Wir bestimmen den mittleren Index des Arrays array: $\text{len}(\text{array})//2 = 5$

2. Schritt:

midvalue = array[len(array)//2] = array[10//2] = array[5] = 8

Wir vergleichen value mit midvalue:

Falls $\text{value} == \text{midvalue}$: **binarysearch** gibt den Wert **True** zurück; gefunden!

Falls $\text{value} < \text{midvalue}$: suche in der Liste $a[0], \dots, a[4]$ links von $a[5]$

Falls $\text{value} > \text{midvalue}$: suche in der Liste $a[6], \dots, a[9]$ rechts von $a[5]$

hier: wegen $13 > 8$ suchen wir in der Liste $a[6], \dots, a[9]$

Suche **value** in der Liste **$a[6], \dots, a[9]$**

$a[6]$	$a[7]$	$a[8]$	$a[9]$
11	13	19	21

Diese Liste **$a[6], \dots, a[9]$** und **value** übergeben wir der Booleschen Funktion **binarysearch**, welche **$a[6], \dots, a[9]$** als lokale Liste **$\text{array}[0], \dots, \text{array}[3]$** fortführt.

$\text{array}[0]$	$\text{array}[1]$	$\text{array}[2]$	$\text{array}[3]$
11	13	19	21

1. Schritt:

Wir bestimmen den mittleren Index des Arrays array: $\text{len}(\text{array})//2 = 4//2 = 2$

2. Schritt:

$\text{midvalue} = \text{array}[\text{len}(\text{array})//2] = \text{array}[4//2] = \text{array}[2] = 19$

Wir vergleichen value mit midvalue:

Falls $\text{value} == \text{midvalue}$: **binarysearch** gibt den Wert **True** zurück; gefunden!

Falls $\text{value} < \text{midvalue}$: suche in der Liste $\text{array}[0], \dots, \text{array}[1]$ links von $\text{array}[2]$

Falls $\text{value} > \text{midvalue}$: suche in der Liste $\text{array}[3]$ rechts von $\text{array}[2]$

hier: wegen $13 < 19$ suchen wir in der Liste $\text{array}[0], \dots, \text{array}[1]$

Suche **value** in der Liste **$\text{array}[0], \dots, \text{array}[1]$**

$\text{array}[0]$	$\text{array}[1]$
11	13

Diese Liste **$\text{array}[0], \dots, \text{array}[1]$** und **value** übergeben wir der Booleschen Funktion **binarysearch**, welche **$\text{array}[0], \dots, \text{array}[1]$** als lokale Liste **$\text{array}[0], \dots, \text{array}[1]$** fortführt.

1. Schritt:

Wir bestimmen den mittleren Index des Arrays array : $\text{len}(\text{array})//2 = 2//2 = 1$

2. Schritt:

$\text{midvalue} = \text{array}[\text{len}(\text{array})//2] = \text{array}[2//2] = \text{array}[1] = 13$

Wir vergleichen value mit midvalue :

Falls $\text{value} == \text{midvalue}$: **binarysearch** gibt den Wert **True** zurück; gefunden!

Falls $\text{value} < \text{midvalue}$: suche in der Liste $\text{array}[0]$ links von $\text{array}[1]$

Falls $\text{value} > \text{midvalue}$: suche in der leeren Liste $[]$ rechts von $\text{array}[1]$, dann: **binarysearch** gibt den Wert **False** zurück; nicht gefunden!

hier: wegen $13 = \text{value} = \text{midvalue} = 13$: **binarysearch** gibt den Wert **True** zurück; gefunden!

Der Booleschen Funktion **binarysearch** werden das aus **n** Komponenten bestehende sortierte Feld **a** (in Python: Liste) und der zu suchende Wert **value** übergeben; **binarysearch** liefert den Wert **True**, falls eine Komponente von **a** mit **value** übereinstimmt, andernfalls den Wert **False**.

Die Variable **z** ermittelt die Anzahl der Aufrufe von **binarysearch**.

Quelltext in Python:

```
.....

z = 0
. . . . .

def binarysearch(array,value):
    global z
    z += 1
    print(array)
    if array == [] or (len(array) == 1 and array[0] != value):
        return False
    else:
        midvalue = array[len(array)//2]
        if midvalue == value:
            return True
        elif value < midvalue:
            return binarysearch(array[:len(array)//2],value)
        else:
            return binarysearch(array[len(array)//2 + 1:],value)
```

Aufruf der Funktion **binarysearch**:

```
binarysearch(a,value)
```

Komplexität des Algorithmus **binarysearch**:

Die Komplexität und damit der Rechenaufwand **A(n)** wird wesentlich bestimmt durch die Anzahl **z** der Aufrufe der rekursiv definierten Funktion **binarysearch**; o. B. d. A. sei **n** eine Potenz von 2, d. h. **n = 2^k** mit **k = 0, 1, 2, 3,**

Beachte: die maximale Anzahl von Aufrufen (worst case) tritt ein, falls die Suche ergebnislos ist.

```
k = 0 ⇔ n = 1
# Aufrufe binarysearch = 1

k = 3 ⇔ n = 8
gesuchte Zahl: 79
[14, 50, 52, 70, 74, 80, 89, 97]
[80, 89, 97]
[80]
79 wurde nicht gefunden
# Aufrufe binarysearch = 3

k = 4 ⇔ n = 16
gesuchte Zahl: 80
[13, 33, 42, 42, 44, 44, 45, 45, 47, 52, 57, 59, 62, 72, 92, 94]
[52, 57, 59, 62, 72, 92, 94]
[72, 92, 94]
[72]
80 wurde nicht gefunden
# Aufrufe binarysearch = 4
```

Eine Verdopplung von n impliziert höchstens einen weiteren Aufruf von **binarysearch**!

Offensichtlich gilt:

$z = k$

Wegen $n = 2^k \Leftrightarrow k = \log_2(n)$ folgt:

$z = \log_2(n)$

Somit hat der Algorithmus **binarysearch** logarithmische Komplexität:

$$A(n) \sim \log_2(n)$$

Bemerkung:

Falls im ungünstigsten Fall binarysearch noch die leere Liste [] übergeben wird, gilt: $z = k + 1$

Modifikation des Algorithmus binarysearch:

Die rekursive Funktion **binarysearch** liefert den booleschen Wert **False**, falls **value** nicht gefunden wird, andernfalls den Index **index** der betreffenden Komponente.

Außer **a** und **value** sind die Indices **begin** und **end** an die Funktion **binarysearch** zu übergeben, so daß **binarysearch** die Teilliste **a[begin] , . . . , a[end]** durchsucht.

Beachte: **index** wird innerhalb des Funktionsrumpfs als globale Variable definiert.

$z = 0$

.

```
def binarysearch(array, value, begin, end):
    global index
    global z
    z += 1
    print(array[begin:end+1])
    if begin > end: return False
    middle = (begin + end) // 2
    print('mittleres Element: a[' , middle , ' ] = ' , array[middle])
    if array[middle] == value:
        index = middle
    elif array[middle] < value:
        return binarysearch(array, value, middle + 1, end)
    else:
        return binarysearch(array, value, begin, middle - 1)
```

Aufruf der Funktion **binarysearch** zur Suche von **value** in der sortierten Liste

a[0] , , a[n-1]:

binarysearch(a, value, 0, len(a)-1)

```
gesuchte Zahl: 521
[120, 162, 163, 181, 205, 392, 444, 521, 528, 557, 643, 663, 689, 810, 847, 899, 913, 992]
mittleres Element: a[ 8 ] = 528
[120, 162, 163, 181, 205, 392, 444, 521]
mittleres Element: a[ 3 ] = 181
[205, 392, 444, 521]
mittleres Element: a[ 5 ] = 392
[444, 521]
mittleres Element: a[ 6 ] = 444
[521]
mittleres Element: a[ 7 ] = 521
521 wurde gefunden an der Stelle 7
a[ 7 ] = 521
# Aufrufe binarysearch = 5

gesuchte Zahl: 241
[173, 183, 187, 243, 265, 307, 345, 376, 589, 622, 868, 976]
mittleres Element: a[ 5 ] = 307
[173, 183, 187, 243, 265]
mittleres Element: a[ 2 ] = 187
[243, 265]
mittleres Element: a[ 3 ] = 243
[]
241 wurde nicht gefunden
# Aufrufe binarysearch = 4
```