Suchen auf Arrays

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de, Raum 149B
Version: 1.0

Quelle: Die Folien sind teilweise inspiriert von oder basierend auf Lehrmaterial von Prof. Dr. Ritterbusch

Folien: https://delors.github.io/theo-algo-suchen_auf_arrays/folien.de.rst.html

https://delors.github.io/theo-algo-suchen_auf_arrays/folien.de.rst.html.pdf

Fehler melden:

https://github.com/Delors/delors.github.io/issues

1. Einführung

Skalierung von Daten

Welche Skalierung haben gesuchte Daten sind im Array?

nominal: Nur Vergleich auf Gleichheit, keine natürliche Ordnung oder Zahlbegriff.
ordinal: Es gibt Größenvergleiche und damit eine Sortierung, aber kein Zahlbegriff.
kardinal: Es gibt Größenvergleiche und Zahlbegriff.

- *Unsortiert oder nominal* führt (zunächst) zur linearen Suche.
- *Ordinale und kardinale Werte* können sortiert werden für binäre Suche.
- *Kardinale Größen* können modelliert werden für interpolierende Suche.

Hinweis

Für unsere Betrachtung gehen wir im Folgenden davon aus, dass die Daten sortiert sind. Beim Vergleich der Algorithmen beschränken wir uns auf eine Betrachtung der Anzahl der Elementzugriffe.

- Ein Beispiel für eine *nominal* skalierte Datenmenge wäre die Menge der Farben. Es gibt keine natürliche Ordnung der Farben, und es gibt auch keinen natürlichen Zahlenbegriff, der die Farben beschreibt. Ein weiteres Beispiel ist eine Liste von Wohnorten.4
- Ein Beispiel für eine *ordinale* skalierte Datenmenge wäre die Menge der Kleidergrößen (S,M,L,XL,...). Es gibt eine natürliche Ordnung der Kleidergrößen, aber es gibt keinen natürlichen Zahlenbegriff, der die Kleidergrößen beschreibt. Ein weiteres Beispiel ist die Bewertung von Filmen auf einer Skala von 1 bis 5 Sternen.

Lineare Suche

```
1 Algorithm linearSearch(A,n,needle)
2     for i= 1,...,n do
3         if A[i] == needle then
4             return i
5     return nil
```

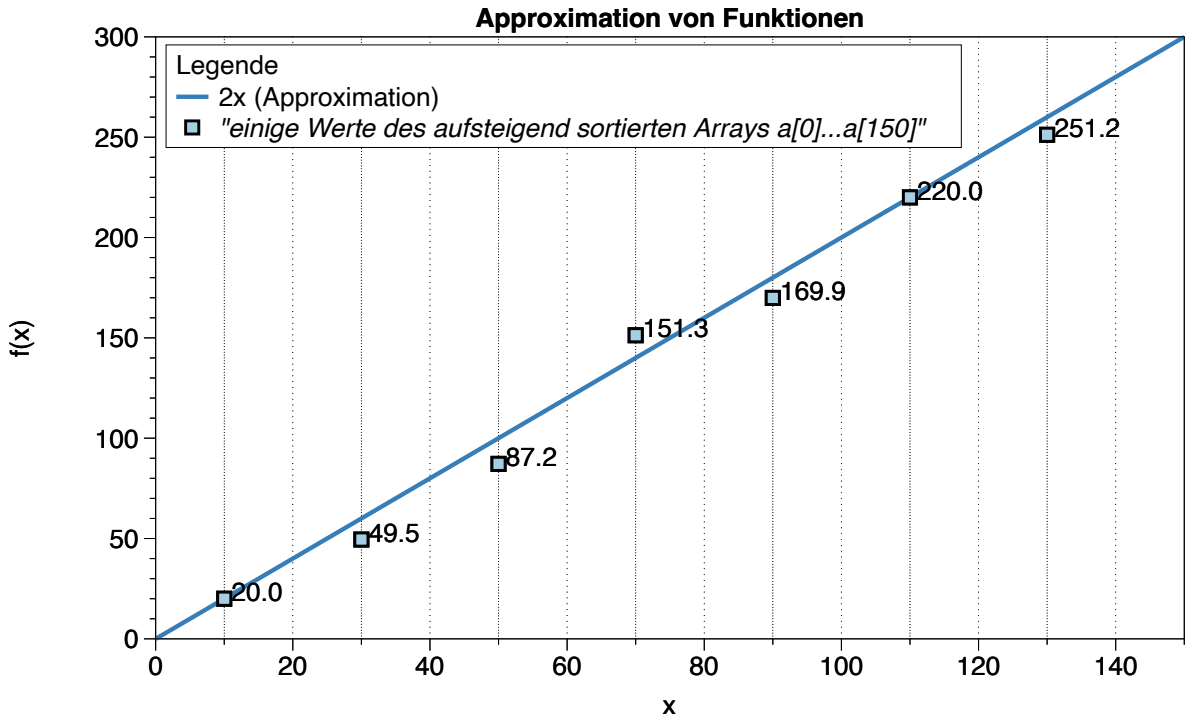
Laufzeit und Elementzugriffe kann asymptotisch durch $O(n)$ abgeschätzt werden.

Binäre Suche

```
1  Algorithm binarySearch(A,l,u,needle)
2      upper = u
3      lower = l
4      repeat
5          pos = round((upper+lower)/2)
6          value = A[pos]
7          if value == needle then
8              return pos
9          else if value > needle then
10             upper = pos-1
11         else
12             lower = pos + 1
13     until upper < lower
14     return nil
```

Laufzeit ist $O(\log(n))$, genauer im Schnitt $\log_2(n) - 1$ Zugriffe.

Effizientere Suche bei bekannter Verteilung (hier linear)



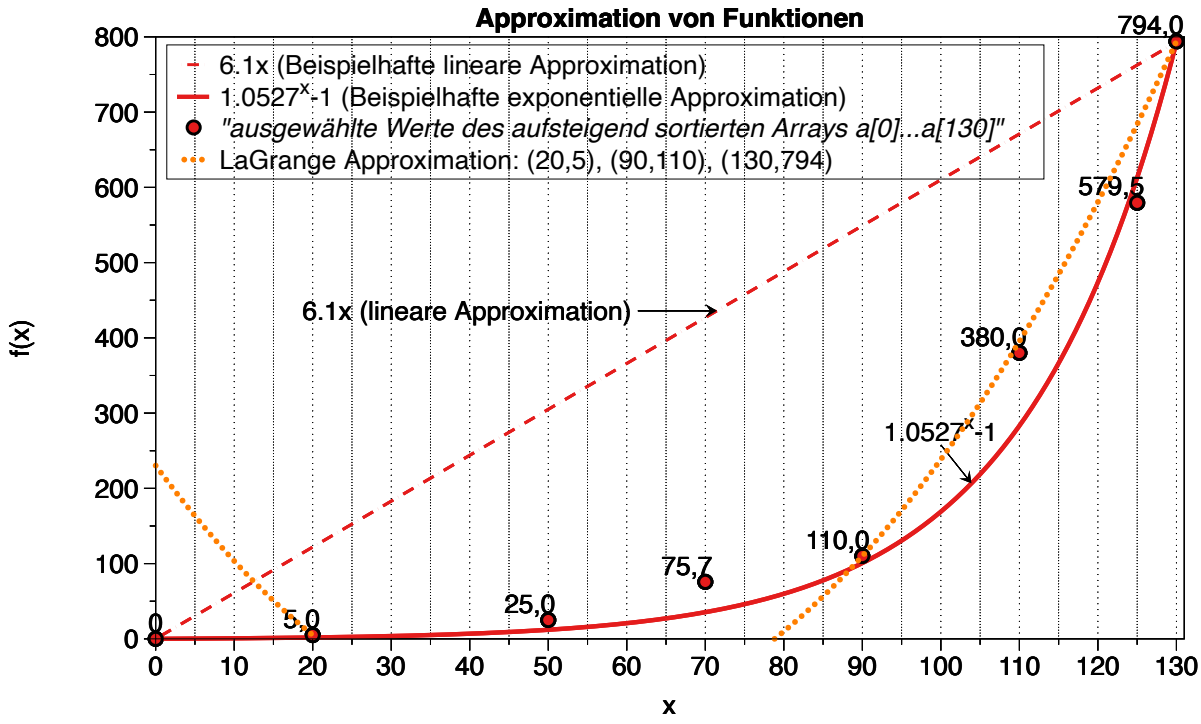
In diesem Beispiel gehen wir davon aus, dass die Werte *im Wesentlichen* linear verteilt sind. Das bedeutet, dass die Differenz zwischen zwei aufeinanderfolgenden Werten immer gleich ist.

Sei beispielsweise ein Array `a` mit folgenden Werten geben (Auszug):

Index i	**Wert**
i = 10	a[i = 10] = 20.0
...	...
i = 30	49.5
...	...
i = 50	87.2
...	...
i = 70	151.3
...	...
i = 90	169.9
...	...
i = 110	220.0
...	...
i = 130	251.2

Wenn man jetzt exemplarisch die Paare: $(x = 10, y = 20.0)$, und $(x = 110, y = 220)$ betrachtet, dann kann man zu dem Schluss kommen, dass die Funktion $f(x) = 2.0 \cdot x$ eine Approximation der Verteilung der Werte ist. Würde man also nach dem Wert (y) 170 suchen wollen, dann wäre es gut als erstes den Wert von `a[85]` zu überprüfen, $170 = 2x \rightarrow \frac{170}{2} = 85 = x$.

Effizientere Suche bei bekannter Verteilung (hier expo.)



In diesem Beispiel gehen wir davon aus, dass die Werte *im Wesentlichen* exponentiell verteilt sind. Das bedeutet, dass die Differenz zwischen zwei aufeinanderfolgenden Werten immer größer wird.

Sei beispielsweise ein Array **a** mit folgenden Werten geben (Auszug):

x	y
0	0
...	...
20	5
...	...
50	25
...	...
70	75.7
...	...
90	110
...	...
110	380
...	...
125	579.5
...	...
130	794

Wenn man jetzt exemplarisch die Paare: $(x = 20, y = 5.0)$, und $(x = 130, y = 794)$ betrachtet, und eine lineare Approximation durchführt, dann könnte man zu dem Schluss kommen, dass die Funktion $f(x) = 6.1 \cdot x$ eine gute Approximation ist.

Würde man eine quadratische Approximation mit Hilfe von Lagrange durchführen, zum Beispiel mit den Werten

$(x = 20, y = 5.0)$, $(x = 90, y = 110)$, und $(x = 130, y = 794)$. Dann wäre der Fehler zwischen der realen Verteilung und der angenommen deutlich geringer, da die quadratische Funktion die Werte besser approximiert.

In diesem Fall Fall wäre die Funktion: $p(x) = \frac{39}{275}x^2 - \frac{141}{10}x + \frac{2533}{11}$ In diesem Fall können wir die Position des Wertes 650 im Array besser abschätzen (durch die Aufstellung der Umkehrfunktion und dann einsetzen von 650): ≈ 123.

Warnung

Eine vernünftige Interpolation ist nur dann möglich, wenn die Verteilung der Werte im Wesentlichen bekannt ist.

Approximation der Verteilung

Wichtig

Wenn wir die Verteilung der Werte kennen, können wir effizientere Algorithmen entwickeln.

Beispiel:

Wenn wir wissen, dass die Werte quadratisch verteilt sind (`Array[10] a = { 1, 4, 9,16, ..., 100 }`), und wir zum Beispiel wissen, dass der kleinste Wert im Array 1 und der größte Wert 100 (an Stelle/mit Index 10) ist, den wir im Array gespeichert haben, dann macht es „keinen“ Sinn den Wert 85 oder 5 in der Mitte zu suchen! (85 findet sich vermutlich an Stelle $9 = \lfloor\sqrt{85}\rfloor$).

Modellierung durch Interpolation: hier Lagrange-Polynome

Speichert unser Array kardinal skalierte Daten, so können diese modelliert werden. Das einfachste Prinzip ist die Polynominterpolation mittels Lagrange-Polynomen.

Das Ziel ist es, ein Polynom $p(x)$ zu finden, das eine Funktion $f(x)$ an einer gegebenen Menge von Punkten $(x_1, y_1), \ldots, (x_n, y_n)$ exakt interpoliert. Das heißt:

$$p(x_i) = y_i \quad \text{für alle } i = 1, \ldots, n$$

1

Satz

Das Lagrange-Interpolationspolynom $p(x)$ wird als Summe von Lagrange-Basispolynomen $l_i(x)$ aufgebaut:

$$p(x) = \sum_{i=1}^{n} (y_i \cdot l_i(x))$$

wobei $l_i(x)$ das i-te Lagrange-Basispolynom gegeben ist durch:

$$l_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

2

Sind n Tupel $(x_n, y_n) \in \mathbb{R}_2$ reeller Zahlen gegeben mit $x_l \neq x_m$ für $l \neq m$.

Das Lagrange-Interpolationspolynom hat dann höchstens den Grad $n - 1$ und es gilt $p(x_i) = y_i$ für alle $i = 1, \ldots, n$.

3

Beispiel

Gegeben sein die zwei Punkte: $(x_1, y_1) = (1, 2)$ und $(x_2, y_2) = (3, 4)$.

Das Lagrange-Polynom $p(x)$ wäre dann:

1. $$l_1(x) = \frac{x - x_2}{x_1 - x_2} = \frac{x - 3}{1 - 3} = \frac{3 - x}{2}$$

2. $$l_2(x) = \frac{x - x_1}{x_2 - x_1} = \frac{x - 1}{3 - 1} = \frac{x - 1}{2}$$

3. $$p(x) = y_1 \cdot l_1(x) + y_2 \cdot l_2(x) = 2 \cdot \frac{3 - x}{2} + 4 \cdot \frac{x - 1}{2} = x + 1$$

Nach Ausmultiplizieren und Zusammenfassen ergibt das ein Polynom, das durch beide Punkte verläuft.

4

Wenn zwei Punkte gegeben sind, ist das Lagrange Polynom somit:

$$p(x) = y_1 \cdot \frac{x - x_2}{x_1 - x_2} + y_2 \cdot \frac{x - x_1}{x_2 - x_1}$$

Wenn drei Punkte gegeben sind ist das Lagrange Polynom somit:

$$\begin{aligned} p(x) = \quad & y_1 \cdot \tfrac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + \\ & y_2 \cdot \tfrac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + \\ & y_3 \cdot \tfrac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \end{aligned}$$

5

9

Der Grad unseres Lagrange-Polynoms ist immer um 1 kleiner als die Anzahl der gegebenen Punkte (die Terme des Basispolynom sind nur für $j \neq i$ definiert). Das bedeutet, dass wir für zwei Punkte ein lineares Polynom erhalten, für drei Punkte ein quadratisches Polynom, für vier Punkte ein kubisches Polynom, und so weiter.

Übung

Bestimme p(2)

Bestimmen Sie direkt $p(2)$ für das quadratische Polynom mit den Eigenschaften:

$$p(-10) = 3, p(-8) = 1, p(-4) = -1$$

Bestimme p(-1)

Für die gegebenen Punkte, bestimmen Sie erst das Lagrange Polynom $p(x)$ im Allgemeinen und rechnen Sie dann den Wert für $p(-1)$ aus.

$$p(2) = 4, p(4) = 6, p(7) = 3$$

Bestimme p(2)

Bestimmen Sie direkt $p(2)$ für das quadratische Polynom mit den Eigenschaften:

$$p(-10) = 3, p(-8) = 1, p(-4) = -1$$

Bestimme p(-1)

Für die gegebenen Punkte, bestimmen Sie erst das Lagrange Polynom $p(x)$ im Allgemeinen und rechnen Sie dann den Wert für $p(-1)$ aus.

$$p(2) = 4, p(4) = 6, p(7) = 3$$

Interpolierende Suche - lineare Approximation

Beispiel

Gegeben: Vom Array `a` sei bekannt: `a[1] = 0`, `a[20] = 30` und `a[40] = 120`.

Frage: Ist der Wert `50` im Array enthalten?

Lösung: Das Lagrangepolynom $p(x)$ mit $p(30) = 20$ und $p(120) = 40$ lautet:

$$p(x) = 20 \cdot \frac{x - 120}{30 - 120} + 40 \cdot \frac{x - 30}{120 - 30}$$

Für den gesuchten Wert `50` ergibt sich als zu untersuchende Position:

$$p(50) = 20 \cdot \frac{50 - 120}{30 - 120} + 40 \cdot \frac{50 - 30}{120 - 30} = \frac{220}{9} \approx 24$$

Bemerkung

Wir möchten die Position des Wertes 50 im Array abschätzen! Deswegen sind im linearen Modell die Paare $(x_1, y1) = (30, 20)$ und $(x_2, y_2) = (120, 40)$ zu wählen. D. h. die Indizes sind unsere y-Werte.

- Eine binäre Suche würde in diesem Fall mit der Position $\frac{40+20}{2} = 30$ beginnen.
- **Hinweis**

  Das Lagrangepolynom kann per Konstruktion die Position der Werte 30 und 120 perfekt bestimmen:

  $$p(30) = 20 \cdot \frac{30 - 120}{30 - 120} + 40 \cdot \frac{30 - 30}{120 - 30} = 20$$

  $$p(120) = 20 \cdot \frac{120 - 120}{30 - 120} + 40 \cdot \frac{120 - 30}{120 - 30} = 40$$

Interpolierende Suche - quadratische Approximation

Beispiel

Gegeben: Vom Array `a` sei bekannt: `a[1] = 0`, `a[20] = 30` und `a[40] = 120`.

Frage: Ist der Wert `50` im Array enthalten?

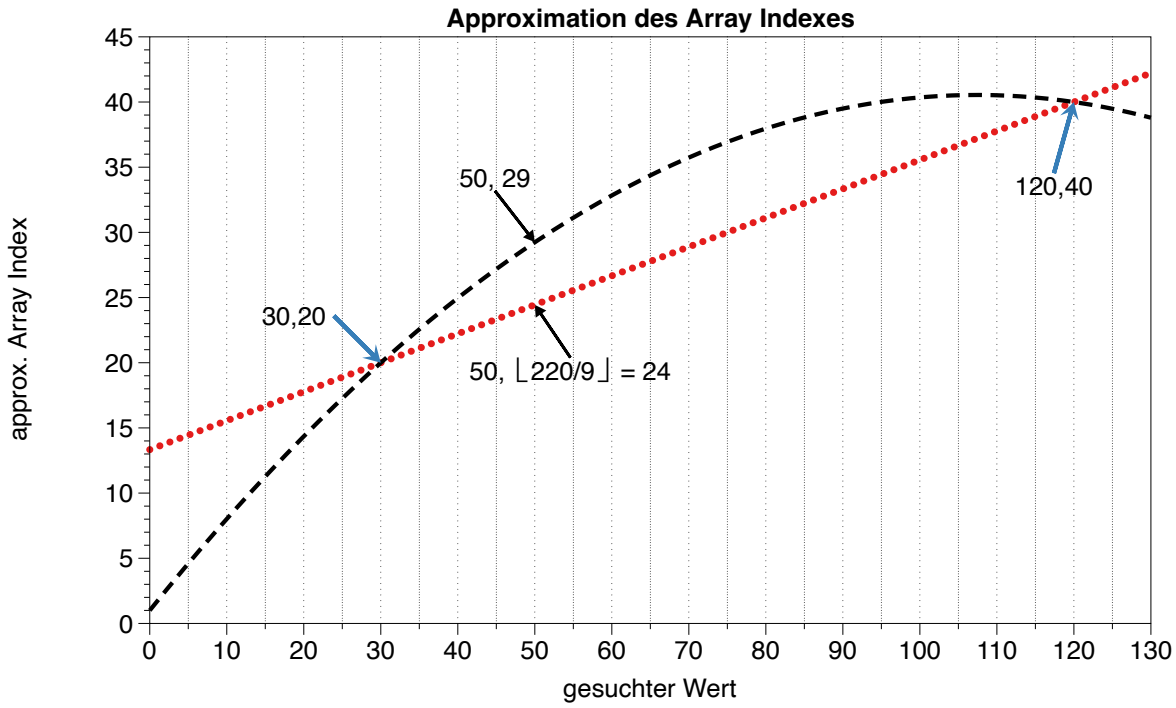
Lösung: $p(x)$ mit $p(0) = 1$, $p(30) = 20$ und $p(120) = 40$ lautet:

$$\begin{aligned} p(x) = \quad & 1 \cdot \tfrac{(x-30)(x-120)}{(0-30)(0-120))} + \\ & 20 \cdot \tfrac{(x-0)(x-120)}{(30-0)(30-120)} + \\ & 40 \cdot \tfrac{(x-0)(x-30)}{(120-0)(120-30)} \end{aligned}$$

Für den gesuchten Wert `50` ergibt sich als zu untersuchende Position:

$$p(50) \approx 29$$

Interpolierende Suche - Vergleich



1

- Auf gleichverteilten Daten hat lineare Interpolationssuche O(log log n).
- Auf anderen Verteilungen ist lineare Interpolation oft schlechter als binäre Suche.
- Quadratische Interpolation hat ein erweitertes Modell und schlägt binäre Suche häufig.

2

Lineare interpolierende Suche

```
1  Algorithm linearInterpolatingSearch(A,needle)
2      lower = 1
3      upper = length(A)
4      vL = A[lower]
5      if vL == needle then return lower
6      vU = A[upper]
7      if vU == needle then return upper
8      while upper > lower do
9          pos = round(lower·(needle−vU)/(vL−vU) +
10                     upper·(needle−vL)/(vU−vL))
11         value = A[pos]
12         if value == needle then
13             return pos
14         else if value < needle then
15             lower = max(pos, lower+1), vL = A[lower]
16         else
17             upper = min(pos, upper−1), vU = A[upper]
18     return nil
```

Exponentielle Suche im sortierten (unbeschränkten) *Array*

```
1 Algorithmus ExponentialSearch(A,needle)
2     i = 1
3     while A[i] < needle do
4         i = i * 2
5     return BinarySearch(A,floor(i/2) + 1,i,needle)
```


Die Idee ist erst mit einer exponentiellen Schrittweite zu springen, um dann mit einer binären Suche den Wert zu finden. Die Laufzeit ist $O(\log(i))$ wobei i die Position des gesuchten Wertes ist. Die Laufzeit ist also $O(\log(n))$.

Übung

Wer sucht, der findet 5?

Folgende Werte sind vom Array A bekannt:

$$A[1] = -27,\ A[15] = 13,\ A[29] = 29$$

Gesucht wird der potentielle Index des Wertes `5`. Welcher Index `i` sollte als nächstes untersucht werden bei binärer, linearer oder quadratisch interpolierender Suche?

Wer sucht, der findet -1?

Folgende Werte sind vom Array A bekannt:

$$A[1] = -13,\ A[7] = -4,\ A[13] = 11$$

Gesucht wird der potentielle Index des Wertes `-1`. Welcher Index `i` sollte als nächstes untersucht werden bei binärer, linearer oder quadratisch interpolierender Suche?

Wer sucht, der findet 5?

Folgende Werte sind vom Array A bekannt:

$A[1] = -27,\ A[15] = 13,\ A[29] = 29$

Gesucht wird der potentielle Index des Wertes `5`. Welcher Index `i` sollte als nächstes untersucht werden bei binärer, linearer oder quadratisch interpolierender Suche?

Wer sucht, der findet -1?

Folgende Werte sind vom Array A bekannt:

$$A[1] = -13,\ A[7] = -4,\ A[13] = 11$$

Gesucht wird der potentielle Index des Wertes `-1`. Welcher Index `i` sollte als nächstes untersucht werden bei binärer, linearer oder quadratisch interpolierender Suche?

Übung

Lineare Interpolierende Suche

Setzen Sie den Algorithmus für die lineare interpolierende Suche in einer Programmiersprache Ihrer Wahl um.

Testen Sie den Algorithmus mit folgenden Arrays:

```
A = [1, 3, 5, 7, 9, 11, 13, 15] # linear verteilt (2x-1)

B = [0, 7, 13, 22, 27, 32, 44, 49] # approx. linear verteilt (approx. 7x)

C = [0, 4, 16, 36, 64, 100, 144, 196] # quadratisch verteilt (4x^2)
```

Wie viele Schritte (im Sinne von Schleifendurchläufen) sind maximal notwendig, um festzustellen ob ein Wert im Array enthalten ist oder nicht?

Lineare Interpolierende Suche

Setzen Sie den Algorithmus für die lineare interpolierende Suche in einer Programmiersprache Ihrer Wahl um.

Testen Sie den Algorithmus mit folgenden Arrays:

```
A = [1, 3, 5, 7, 9, 11, 13, 15] # linear verteilt (2x-1)

B = [0, 7, 13, 22, 27, 32, 44, 49] # approx. linear verteilt (approx. 7x)

C = [0, 4, 16, 36, 64, 100, 144, 196] # quadratisch verteilt (4x^2)
```

Wie viele Schritte (im Sinne von Schleifendurchläufen) sind maximal notwendig, um festzustellen ob ein Wert im Array enthalten ist oder nicht?

Übung

Exponentiell Interpolierende Suche

Wann macht es Sinn die exponentiell interpolierende Suche zu verwenden?

Bedenken Sie zum Beispiel ein (virtuelles) Array. D. h. ein Array, dass durch ein Bildungsgesetz definiert ist.

Exponentiell Interpolierende Suche

Wann macht es Sinn die exponentiell interpolierende Suche zu verwenden?

Bedenken Sie zum Beispiel ein (virtuelles) Array. D. h. ein Array, dass durch ein Bildungsgesetz definiert ist.