Vorkursvortrag: Algorithmen und Datenstrukturen

Vorkurs WiSe 2023/24

Raeesa Yousaf, raeeesa@mathphys.info

Ruprecht-Karls-Universität Heidelberg

October 8, 2023

Inhalt

Inhalt

Was ist ein Algorithmus?

"Algorithmen" sind überall!

Was ist ein Algorithmus?

"Algorithmen" sind überall!

- Firmen nutzen Algorithmen für "streamlining of workflow"
- Soziale Netzwerke nutzen Algorithmen für content recommendations
- Etc...

Was ist ein Algorithmus?

"Algorithmen" sind überall!

- Firmen nutzen Algorithmen für "streamlining of workflow"
- Soziale Netzwerke nutzen Algorithmen für content recommendations
- Etc...

Was heißt dieses Wort denn überhaupt?

Definition Algorithmus

Definition 1.1: Algorithmus

Ein Algorithmus ist eine Liste von endlich vielen, elementaren Schritten, um ein Problem zu lösen.

Beispiele: Kuchen backen, IKEA-Möbel zusammenbauen, Quests in Videospielen...

Definition Algorithmus

Definition 1.1: Algorithmus

Ein Algorithmus ist eine Liste von endlich vielen, elementaren Schritten, um ein Problem zu lösen.

Beispiele: Kuchen backen, IKEA-Möbel zusammenbauen, Quests in Videospielen...
Für Unsere Zwecke: Suchen, Sortieren

Suchen

Definition 1.2: Suchen

Einen Suchbereich durchlaufen und Elemente mit gewissen Charakteristiken finden.

Charakteristiken: ist gleich, größer als, Hamming-Distanz kleiner als 2, ...

Suchen

Definition 1.2: Suchen

Einen Suchbereich durchlaufen und Elemente mit gewissen Charakteristiken finden.

Charakteristiken: ist gleich, größer als, Hamming-Distanz kleiner als 2, ...
Suchverfahren:

- Linear
- Binär

(für unsere Zwecke: Liste mit Zufallszahlen; suche ein bestimmtes Element)

Lineare Suche

Laufe die Liste von Anfang an durch und vergleiche alle Elemente mit dem, welches wir suchen

Lineare Suche

Laufe die Liste von Anfang an durch und vergleiche alle Elemente mit dem, welches wir suchen

```
for i in range(10):
        if list[i]==gesucht:
                return list[i]
return NULL
```

Ist die einzige Option, wenn die Liste nicht sortiert ist.

Binäre Suche

Funktioniert nur, wenn die Liste sortiert ist!

1. Finde die Mitte der Liste
2. Vergleiche den Wert mit dem, der gesucht wird
3. Wenn der gesuchte Wert größer ist als die Mitte, wiederhole den Prozess mit der oberen Hälfte; analog mit der unteren Hälfte

Binäre Suche

Funktioniert nur, wenn die Liste sortiert ist!

1. Finde die Mitte der Liste
2. Vergleiche den Wert mit dem, der gesucht wird
3. Wenn der gesuchte Wert größer ist als die Mitte, wiederhole den Prozess mit der oberen Hälfte; analog mit der unteren Hälfte

```
min = 0
max = x-1
while min <= max:
        i=math.ceil((max+min)/2.0)
        if gesucht > list[i]:
                min = i+1
        elif gesucht < list[i]:
                max = i-1
        else:
                return list[i]
return NULL
```

Frage:

Intuitiv sieht man ja, dass die zweite Art zu suchen schneller ist...

Frage:

Intuitiv sieht man ja, dass die zweite Art zu suchen schneller ist...
... aber wie kann man dies wissenschaftlich ausdrücken?

Inhalt

Landau-Notation

- Auch bekannt als: Landau-Symbole, big O notation
- Erfunden von Paul Bachmann und Edmund Landau
- Beschreiben das asymptotische Verhalten von Funktionen und Folgen
- Beantworten die Frage: "Wenn meine Liste x-mal so groß wäre, wie viel mehr Aufwand/Speicher/Zeit brauch ich, um das Problem zu lösen?"
- Fünf Symbole: $o, \mathcal{O}, \Theta, \Omega, \omega$ (aber fürs Erste ist nur $\mathcal{O}$ wichtig)

Das Landau-O

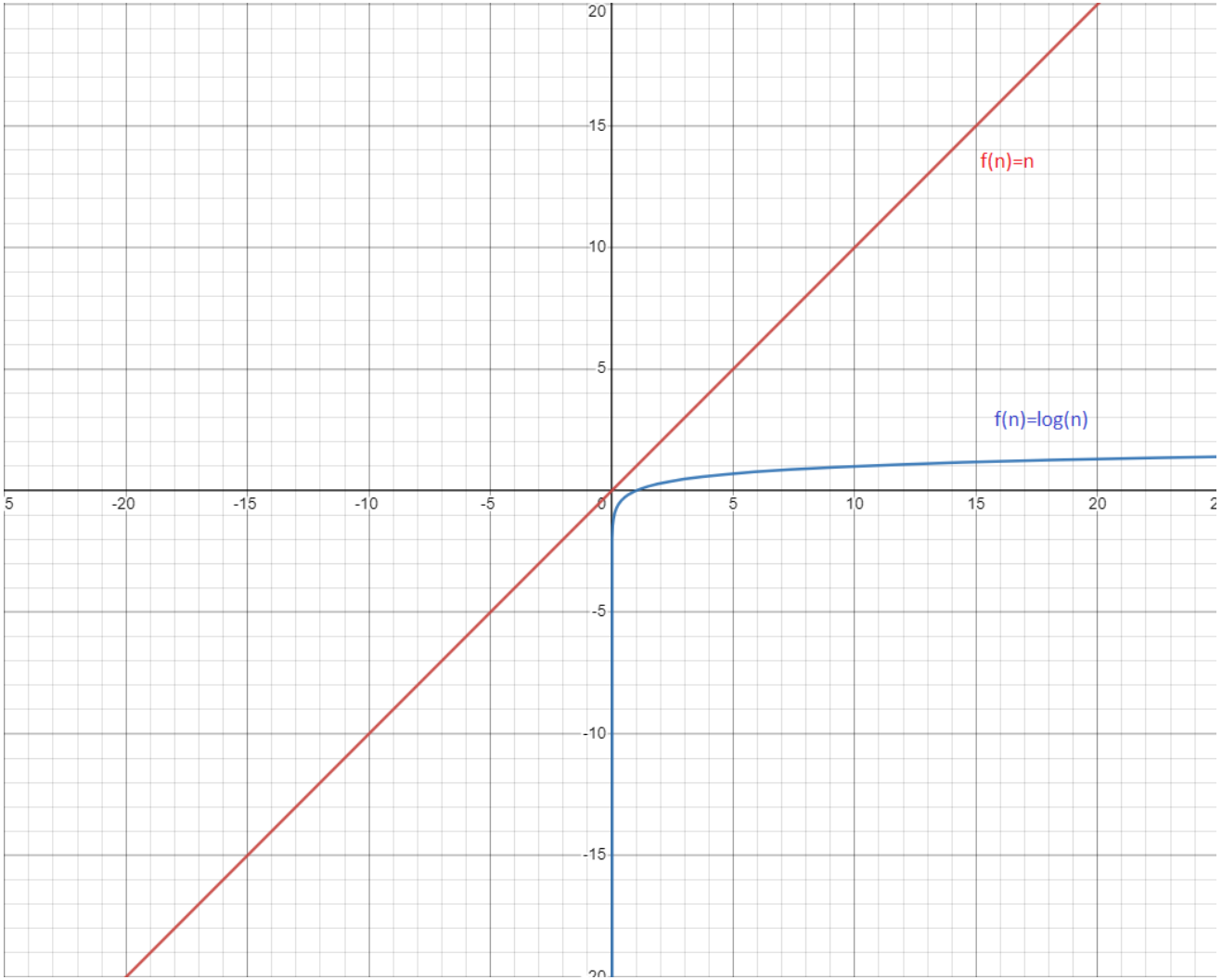
- Notation $g(x) \in \mathcal{O}(f(x))$
- Bedeutet: g(x) ist ungefähr gleich komplex wie f(x) (oder weniger)
- Beispiel: "Suche mir im Telefonbuch den allerersten Namen raus" $= \mathcal{O}(1)$
- Beispiel: "Drucke bitte n Seiten aus" $= \mathcal{O}(n)$
- Wichtig: Doppelseitig oder nicht ist irrelevant!

Komplexität: Suchen

Lineares Suchen:
Man muss jedes Element überprüfen, bis man das Richtige findet;
Bei doppelt so langer Liste muss man doppelt so lange suchen
$\Rightarrow \mathcal{O}(n)$

Binäres Suchen:
Nach jedem Schritt wird der Suchbereich halbiert; Bei doppelt so langer Liste nur einen Schritt mehr $\Rightarrow \mathcal{O}(\log n)$

Komplexität: Suchen



Komplexität

Wichtig: Nur weil ein Algorithmus ein "besseres $\mathcal{O}$" hat, heißt es (im echten Leben) nicht immer, dass der Algorithmus besser/schneller/etc... ist!

Inhalt

Definition 1.3: Sortieren

Elemente im Sortierbereich in eine von einer Charakteristik abhängigen Reihenfolge bringen.

Definition 1.3: Sortieren

Elemente im Sortierbereich in eine von einer Charakteristik abhängigen Reihenfolge bringen.

- Insertion Sort
- Selection Sort
- Bubblesort
- Mergesort
- Quicksort
- Miscsort

Insertion Sort

- Fange am Anfang an
- Sortiere bei jedem Schritt das “linkeste” Element des noch unsortierten Teils der Liste in den sortierten Teil der Liste ein

Insertion Sort

- Fange am Anfang an
- Sortiere bei jedem Schritt das "linkeste" Element des noch unsortierten Teils der Liste in den sortierten Teil der Liste ein

```
for i in range(1, len(arr)):
            key = arr[i]
            j = i-1
            while j >=0 and key < arr[j] :
                    arr[j+1] = arr[j]
                    j -= 1
            arr[j+1] = key
```

Insertion Sort

- Fange am Anfang an
- Sortiere bei jedem Schritt das "linkeste" Element des noch unsortierten Teils der Liste in den sortierten Teil der Liste ein

```
for i in range(1, len(arr)):
            key = arr[i]
            j = i-1
            while j >=0 and key < arr[j] :
                    arr[j+1] = arr[j]
                    j -= 1
            arr[j+1] = key
```

Durchschnittliche Komplexität: $\mathcal{O}(n^2)$

Selection Sort

- Fange am Anfang an
- Suche bei Schritt n das n-kleinste Element in der unsortierten Liste und tausche dies mit dem Element auf Platz n

Selection Sort

- Fange am Anfang an
- Suche bei Schritt n das n-kleinste Element in der unsortierten Liste und tausche dies mit dem Element auf Platz n

```
for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
                if arr[min_idx] > arr[j]:
                        min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Selection Sort

- Fange am Anfang an
- Suche bei Schritt n das n-kleinste Element in der unsortierten Liste und tausche dies mit dem Element auf Platz n

```
for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
                if arr[min_idx] > arr[j]:
                        min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Durchschnittliche Komplexität: $\mathcal{O}(n^2)$ (aber idR schlechtere Performance als Insertion Sort!)

Bubblesort

- Fange am Anfang an
- Bringe nach jedem Schritt das größte Element so weit nach hinten wie möglich, bis du entweder ein größeres findest oder am Ende angelangst

Bubblesort

- Fange am Anfang an
- Bringe nach jedem Schritt das größte Element so weit nach hinten wie möglich, bis du entweder ein größeres findest oder am Ende angelangst

```
n = len(arr)
for i in range(n-1):
        for j in range(0, n-i-1):
                if arr[j] > arr[j+1] :
                        arr[j], arr[j+1] = arr[j+1], arr[j]
```

Bubblesort

- Fange am Anfang an
- Bringe nach jedem Schritt das größte Element so weit nach hinten wie möglich, bis du entweder ein größeres findest oder am Ende angelangst

```
n = len(arr)
for i in range(n-1):
        for j in range(0, n-i-1):
                if arr[j] > arr[j+1] :
                        arr[j], arr[j+1] = arr[j+1], arr[j]
```

Durchschnittliche Komplexität: $\mathcal{O}(n^2)$

Mergesort

- Teile die Liste in Einzelteile, bis du nicht mehr kannst
- Sortiere die Elemente beim Zusammenfügen wieder

Mergesort

- Teile die Liste in Einzelteile, bis du nicht mehr kannst
- Sortiere die Elemente beim Zusammenfügen wieder

```
ERROR: CODE ZU LANG FÜR PRÄSENTATION
```

Mergesort

- Teile die Liste in Einzelteile, bis du nicht mehr kannst
- Sortiere die Elemente beim Zusammenfügen wieder

`ERROR: CODE ZU LANG FÜR PRÄSENTATION`

Durchschnittliche Komplexität: $\mathcal{O}(n \log n)$

Quicksort

- Suche dir ein Element als Pivot aus
- Füge die restlichen Elemente in eine von zwei Teillisten ein, je nachdem, ob die größer oder kleiner als das Pivot sind
- Sortiere die Teillisten

Quicksort

- Suche dir ein Element als Pivot aus
- Füge die restlichen Elemente in eine von zwei Teillisten ein, je nachdem, ob die größer oder kleiner als das Pivot sind
- Sortiere die Teillisten

```
ERROR: CODE ZU LANG FÜR PRÄSENTATION
```

Quicksort

- Suche dir ein Element als Pivot aus
- Füge die restlichen Elemente in eine von zwei Teillisten ein, je nachdem, ob die größer oder kleiner als das Pivot sind
- Sortiere die Teillisten

`ERROR: CODE ZU LANG FÜR PRÄSENTATION`

Durchschnittliche Komplexität: $\mathcal{O}(n \log n)$

Miscsorts

- Heapsort

Miscsorts

- Heapsort
- Cocktail shaker sort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort
- Bogosort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort
- Bogosort
- QUANTUM Bogosort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort
- Bogosort
- QUANTUM Bogosort
- Pancake sort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort
- Bogosort
- QUANTUM Bogosort
- Pancake sort
- I Can't Believe It Can sort

Miscsorts

- Heapsort
- Cocktail shaker sort
- Tree sort
- Bogosort
- QUANTUM Bogosort
- Pancake sort
- I Can't Believe It Can sort
- ...

Inhalt

Definition Datenstruktur

Definition 1.4: Datenstruktur

Eine Datenstruktur ist ein Objekt zur Speicherung und Organisation von Daten. Dieses Objekt ist durch die enthaltenen Daten, aber vor allem durch die Operationen auf diesen Daten/dem Objekt charakterisiert.

Definition Datenstruktur

Definition 1.4: Datenstruktur

Eine Datenstruktur ist ein Objekt zur Speicherung und Organisation von Daten. Dieses Objekt ist durch die enthaltenen Daten, aber vor allem durch die Operationen auf diesen Daten/dem Objekt charakterisiert.

- Array
- Vector
- (Linked) List
- Container

Definition Datenstruktur

Definition 1.4: Datenstruktur

Eine Datenstruktur ist ein Objekt zur Speicherung und Organisation von Daten. Dieses Objekt ist durch die enthaltenen Daten, aber vor allem durch die Operationen auf diesen Daten/dem Objekt charakterisiert.

- Array
- Vector
- (Linked) List
- Container
  - Stack
  - Queue
  - Dictionary

Array

Ein Array ist eine Sammlung einer bestimmten Anzahl von Elementen, die in aufeinanderfolgenden Speicheradressen gespeichert wurden.
Vorteil: Schnelle Zugriffe ($\mathcal{O}(1)$)
Nachteil: Man kann nicht direkt Elemente hinzufügen

Vector

Auch in aufeinanderfolgenden Adressen gespeichert, aber beliebig erweiterbar

Vorteil: genauso schnell wie Array und beliebig erweiterbar

Nachteil: speicherintensiver als Arrays

Linked List

Liste, wo jedes Element zwei Teile enthält: Einen Wert und einen pointer auf das nächste Element.
Vorteil: Wieder flexible Größe, Änderungen zwischen Elementen simpel
Nachteil: Kein Indexzugriff möglich

Container

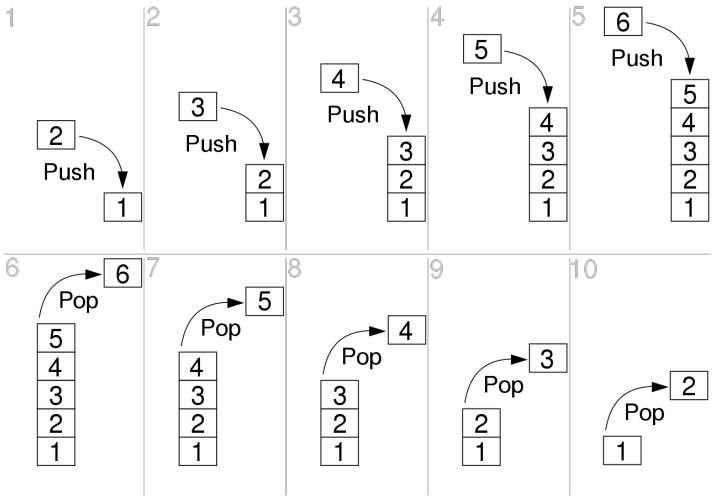
Container sind Abstrakte Datentypen, die andere Daten speichern können; Operationen und Semantik werden dazudefiniert

- Stack
- Queue
- Dictionary

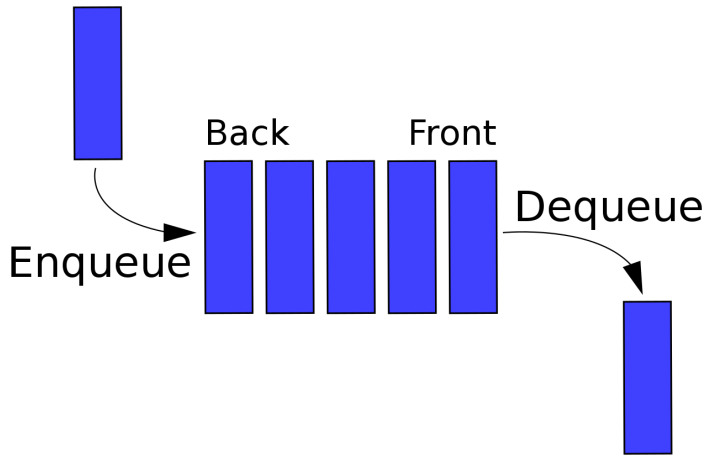
Container: Stack und Queue

Beides sind Container, in denen Elemente “temporär” gespeichert werden, um dann wieder in einer bestimmten Reihenfolge entfernt zu werden;

Stack: LIFO



Queue: FIFO



Container: Dictionary

Ein Dictionary speichert Wertepaare; Ein Wertepaar besteht aus "Key" und "Value"
Wird öfter für Lookups benutzt

Inhalt

- Ihr habt die *Absoluten Basics* der Informatik kennengelernt

- Ihr habt die *Absoluten Basics* der Informatik kennengelernt
- Turingmaschinen, Programmierparadigmen, Klassen/Vererbung, Betriebssysteme, Internet, Datenbanken Und Und Und!!!!!

- Ihr habt die *Absoluten Basics* der Informatik kennengelernt
- Turingmaschinen, Programmierparadigmen, Klassen/Vererbung, Betriebssysteme, Internet, Datenbanken Und Und Und!!!!!
- Informatik ist ein vielseitiges und interessantes Gebiet