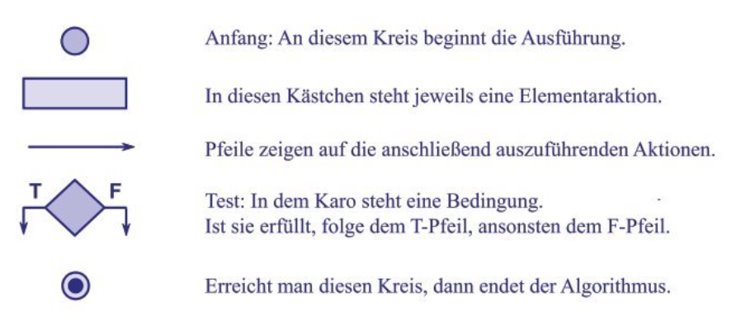
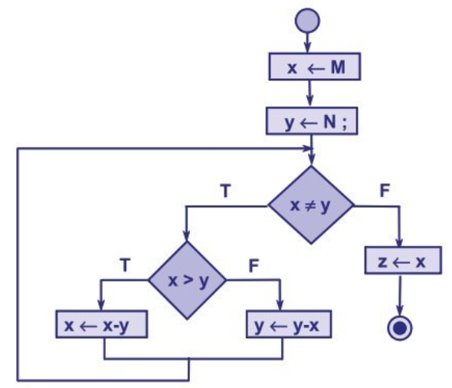
Vorlesung 20.11.2023  
Einführung in die Informatik

### Algorithmen Allgemein

* Algorithmen nehmen in der Informatik einen großen Stellenwert ein.
  + Innerhalb dieser Disziplin geht es darum, Algorithmen zu entwerfen, zu implementieren und anzupassen. ​
* Außerhalb der Informatik werden Algorithmen mitunter als unverständliche Prozesse gesehen.. ​
* eine logische und systematische Abfolge von Rechen- oder Bearbeitungsschritten.​
* Definition:​
  + Ein Algorithmus ist eine detaillierte und explizite Vorschrift zur schrittweisen Lösung eines Problems.​
  + Im Einzelnen beinhaltet diese Definition:​
    - Die Ausführung des Algorithmus erfolgt in einzelnen Schritten. ​
    - Jeder Schritt besteht aus einer einfachen und offensichtlichen Grundaktion. ​
    - Zu jedem Zeitpunkt muss klar sein, welcher Schritt als nächstes kommt.

### Algorithmen Darstellen Flowchart



### 

### Datenstrukturen

* Zusammenfassung gleicher oder verschiedener Datentypen.
* Konstruktionsprinzipien müssen eingehalten werden.
* einfache Datenstruktur ist in vielen Programmiersprachen implementiert. ​
  + Bekannte Beispiele sind Felder oder Arrays
  + Bestehen aus identischen Datentypen
* abstrakten Datenstrukturen zählen komplexere Konzepte wie zum Beispiel Stack (Stapel), Heap (Halde), Trees (Bäume), Graphen und so weiter.
  + Standardmäßig nicht Implementiert
* Array
  + es können mehrere Variablen desselben Grundtyps gespeichert werden können.
  + Benötigt dimension bei Initialisierung
* Verkettete Listen
  + Einträge werden nicht in aufeinanderfolgende Speicheradressen gespeichert
  + Jeder Eintrag in der Liste besteht aus:
    - einen Knoten wo drinne der Datentyp gespeichert wird
    - einen Zeiger welchen auf den nächsten Knoten hinweist
    - Wenn die Liste in beide richtung zeigt also auf das vordere und folgende Glied ist sie doppelt Verkettet
  + Beim Suchen gehen wir jedes Element durch bis wir bei dem Knoten ankommen
  + Allgemein beim Einfügen wird
    - Speicherplatz reserviert
    - dynamisch zugeordnet
  + Beim Einfügen an das ende wird
    - Der Nullzeiger neu ausgerichtet
    - Eintrag an das Ende gesetzt
  + Beim Einfügen zwischen einträgen werden
    - Die Richter von vor dem neuen Objekt auf das neue Element abgeändert werden
    - Richter von den neuen Elementen so ausgerichtet das sie auf das folgende Objekt zeigen
* Stack
  + Last in First Out
  + Speicher wird wie auf einen Stapel gelegt
  + Elemente werden durch Pop ausgelesen
* Zuordnungstabelle
  + Liste von Schlüssel-Wert-Paaren
  + ist eine assoziative Datenstruktur.
  + Zu einem lesbaren Schlüssel, bsp in Form einer Zeichenkette,gibt es einen Eintrag (den Wert) innerhalb der Datenstruktur.
  + Beispiel: Hashtabelle
* Bäume
  + Jeder Knoten kann Kinder haben anzahl kommt dabei auf die Baumart an
  + Elemente werden über Kinder verkettet.

### Suchen und sortieren

**Sequenzielle Suche:**

* In der Praxis kommen bei der Programmierung of Sortiervorgänge vor.
* Sortieren von Kontoauszügen nach der Kontonummer
* Sortieren von Personen nach Alter
* Sortieren nach Preis
* Einfachster Fall: Array mit Zahlen wird sortiert.
* Muss man eine bestimmte Zahl finden kann man ein Algorithmus entwerfen, welcher das gesamte Array von Anfang bis Ende durchsucht
* Optimalfall: Gesuchtes Element ist erstes Element
* worst-case: Element nicht vorhanden.
* folgende Vorgehensweise ist als lineare oder sequenzielle Suche bekannt
* lineares Suchen ist in der Praxis sehr ineffizient bei großen Datenmengen, weshalb effizientere Suchalgorithmen erfunden wurden

**Binäre Suche:**

* ist eine Suchstrategie
* braucht eine sortierte Liste (nach einer Suchbedingung z.B. Alphabet)
* Der Suchalgorithmus teilt den Datensatz in zwei Teile und vergleicht den mittleren Eintrag des Datenbestands mit dem Suchwert.
* Da der Datensatz geordnet ist, kann eindeutig festgestellt werden, ob der Suchwert größer oder kleiner als der mittlere Eintrag ist.
* Dadurch kann man einen Teil der Liste ausschließen wo das Element sich nicht befindet und in dem Bereich weitersuchen wo das Element sich befindet.

Anwendung:



Damit die Binäre Suche funktioniert, muss folgendes eingehalten werden:

* In der Liste die Elemente an Positionen gespeichert sind
* man über die Position auf das Element zugreifen kann
* die Liste geordnet ist (sortiert ist)
* die Elemente der Ordnung entsprechen

**Mergesort:**

* Zu sortierende Menge für in Teilbereiche aufgeteilt die getrennt voneinander sortiert werden ([divide and conquer](https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm))
* Anschließend fügt der Algorithmus die sortierten Teillisten in der korrekten Reihenfolge zusammen, um so eine sortierte Gesamtliste zu bekommen (wie der Name schon sagt “merge”)
* Beim Zusammenfügen dieser Teillisten wird dann der Vergleich durchgeführt, um die Sortierung zu erreichen.
* Es finden Merge-Schritte statt um die Liste vollständig zu sortieren
* Laufzeit beträgt n \* log n
* Mergesort wird bevorzugterweise für größere Listen eingesetzt

**Quicksort:**

* Auch hier Divide and Conquer
* es findet eine Partitionierung (aufteilung) statt, wobei ein Pivotelement gewählt wird, anhand derer man die Elemente sortiert. Dabei gehen Elemente < Pivot nach links und Elemente > Pivot nach rechts
* Es gibt zwei Zeiger (Zwei Indexwerte) einer am Ende einer am Anfang des Teilbereiches den der Algorithmus aktuell sortiert. Die Sortierung findet solange statt bis der Indexwert am Anfang kleiner oder gleich ist als der Indexwert am Ende.
* Quicksort arbeitet im Gegensatz zu Mergesort „in-place“, das heißt, der Algorithmus benötigt für die Daten keinen zusätzlichen externen Speicher, sondern überschreibt die Eingabe- mit den Ausgabedaten.
* Laufzeit n \* log n
* worst-case n²

### Qualität der Algorithmen

* Komplexität
  + Ressourcen aufwand
  + ein Algorithmus soll möglichst effizient bei der Nutzung von Ressourcen sein und wenig Speicher (Speichereffizienz) oder Rechenzeit (Laufzeiteffizienz) benötigen.
  + Funktion für Zeitkomplexität = T(n)
  + Funktion f[r Speicherkomplexität = S(n)
  + n = Eingabedaten
  + Folgen der O-Notation