

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Was heißt ein Algorithmus ist effizient?

Was messen wir?

- ▶ Speicherverbrauch
- ▶ Laufzeit
- ▶ Anzahl Vergleiche (z.B. Sortieralgorithmen)
- ▶ Anzahl an Multiplikationen (wissenschaftliches Rechnen)
- ▶ Festplattenzugriffe
- ▶ Programmgröße
- ▶ Energieverbrauch
- ▶ ...

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.

- ▶ Theoretische Analyse in einem **Rechenmodell**.

Die Komplexität eines Algorithmus wird immer in Zeit gemessen. Üblicherweise wird das O -Notation verwendet, was kann auch andere Schranken erlauben, jedoch vergleichsweise weniger verbreitet. Insofern wie auch das Ω -Notation für untere Schranken.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
 - ▶ Theoretische Analyse in einem **Rechenmodell**.

Die Komplexität eines Algorithmus ist immer in Θ zu beschreiben. In dieser Hinsicht wird das Θ Rechenmodell verwendet, weil hier auch untere Schranken ermittelbar sind. In der Komplexitätstheorie werden Verfahren basierend auf dem Θ Rechenmodell verglichen.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem **Rechenmodell**.

Wie kann man die Laufzeit eines Algorithmus theoretisch berechnen? Wie kann man sich die Laufzeit eines Algorithmus theoretisch vorstellen? Wie kann man sich die Laufzeit eines Algorithmus theoretisch vorstellen? Wie kann man sich die Laufzeit eines Algorithmus theoretisch vorstellen?

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem Rechenmodell.

Wie kann man die Laufzeit eines Algorithmus berechnen?
→ Komplexitätsanalyse
→ Asymptotische Laufzeit
→ Laufzeitkomplexität
→ Zeitkomplexität
→ Raumkomplexität
→ Zeit-Raum-Komplexität
→ Zeit-Raum-Komplexität

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem Rechenmodell.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem **Rechenmodell**.
 - ▶ Gibt **asymptotische Garantien** wie z.B. „dieser Algorithmus läuft immer in Zeit $\mathcal{O}(n^2)$ “.
 - ▶ Üblicherweise wird der **worst case** betrachtet.
 - ▶ Man kann auch untere Schranken erhalten: „jedes vergleichsbasierte Sortierverfahren benötigt im worst case mindestens $\Omega(n \log n)$ Vergleiche“.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem **Rechenmodell**.
 - ▶ Gibt **asymptotische Garantien** wie z.B. „dieser Algorithmus läuft immer in Zeit $\mathcal{O}(n^2)$ “.
 - ▶ Üblicherweise wird der **worst case** betrachtet.
 - ▶ Man kann auch untere Schranken erhalten: „jedes vergleichsbasierte Sortierverfahren benötigt im worst case mindestens $\Omega(n \log n)$ Vergleiche“.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem **Rechenmodell**.
 - ▶ Gibt **asymptotische Garantien** wie z.B. „dieser Algorithmus läuft immer in Zeit $\mathcal{O}(n^2)$ “.
 - ▶ Üblicherweise wird der **worst case** betrachtet.
 - ▶ Man kann auch untere Schranken erhalten: „jedes vergleichsbasierte Sortierverfahren benötigt im worst case mindestens $\Omega(n \log n)$ Vergleiche“.

Wie messen wir?

- ▶ Implementieren und Testen auf repräsentativen Eingaben.
 - ▶ Welche Eingaben?
 - ▶ Kann sehr aufwendig sein.
 - ▶ Präzise Resultate wenn sorgfältig durchgeführt.
 - ▶ Resultate gelten aber nur für spezifische Hardware, und spezifische Eingaben.
- ▶ Theoretische Analyse in einem **Rechenmodell**.
 - ▶ Gibt **asymptotische Garantien** wie z.B. „dieser Algorithmus läuft immer in Zeit $\mathcal{O}(n^2)$ “.
 - ▶ Üblicherweise wird der **worst case** betrachtet.
 - ▶ Man kann auch untere Schranken erhalten: „jedes vergleichsbasierte Sortierverfahren benötigt im worst case mindestens $\Omega(n \log n)$ Vergleiche“.

Eingabelänge

Die theoretischen Schranken werden als Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ von der **Eingabelänge** auf die Laufzeit (oder Speicherverbrauch, Energieverbrauch etc.) angegeben.

Die **Eingabelänge** ist z.B.

die Größe der Eingabe (Anzahl der Bits)

die Anzahl der Register

die Anzahl der Speicherzellen
die Anzahl der Speicherzellen, die für die Eingabe benötigt werden

Eingabelänge

Die theoretischen Schranken werden als Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ von der **Eingabelänge** auf die Laufzeit (oder Speicherverbrauch, Energieverbrauch etc.) angegeben.

Die **Eingabelänge** ist z.B.

Eingabelänge

Die theoretischen Schranken werden als Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ von der **Eingabelänge** auf die Laufzeit (oder Speicherverbrauch, Energieverbrauch etc.) angegeben.

Die **Eingabelänge** ist z.B.

- ▶ die Größe der Eingabe (Anzahl an bits)
- ▶ die Anzahl der Argumente

Example 1

Angenommen n Zahlen aus dem Bereich $\{1, \dots, N\}$ sollen sortiert werden. Wir sagen üblicherweise, dass die Eingabelänge n ist, anstatt z.B. $n \log N$, was der Anzahl an Bits entsprechen würde.

Eingabelänge

Die theoretischen Schranken werden als Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ von der **Eingabelänge** auf die Laufzeit (oder Speicherverbrauch, Energieverbrauch etc.) angegeben.

Die **Eingabelänge** ist z.B.

- ▶ die Größe der Eingabe (Anzahl an bits)
- ▶ die Anzahl der Argumente

Example 1

Angenommen n Zahlen aus dem Bereich $\{1, \dots, N\}$ sollen sortiert werden. Wir sagen üblicherweise, dass die Eingabelänge n ist, anstatt z.B. $n \log N$, was der Anzahl an Bits entsprechen würde.

Eingabelänge

Die theoretischen Schranken werden als Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ von der **Eingabelänge** auf die Laufzeit (oder Speicherverbrauch, Energieverbrauch etc.) angegeben.

Die **Eingabelänge** ist z.B.

- ▶ die Größe der Eingabe (Anzahl an bits)
- ▶ die Anzahl der Argumente

Example 1

Angenommen n Zahlen aus dem Bereich $\{1, \dots, N\}$ sollen sortiert werden. Wir sagen üblicherweise, dass die Eingabelänge n ist, anstatt z.B. $n \log N$, was der Anzahl an Bits entsprechen würde.

Wie messen wir

Wie lange die Laufzeit in einem bestimmten Rechenmodell
auf einem Prozess (Machine) M benötigt
• Abhängigkeit von Rechengrößen wie z.B. Anzahl an
Werten, Multiplikationen, Additionen etc.

Wie messen wir

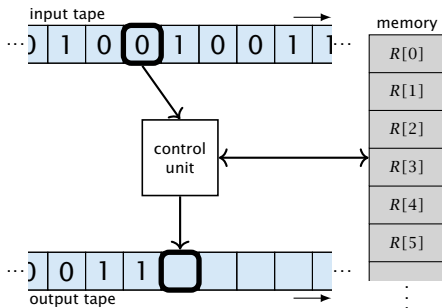
1. Berechne die Laufzeit in einem idealisierten Rechenmodell (z.B. Random Access Machine (RAM))
2. Berechne Anzahl von Basisoperationen wie z.B. Anzahl an Vergleichen, Multiplikationen, Festplattenzugriffen etc.

Wie messen wir

1. Berechne die Laufzeit in einem idealisierten Rechenmodell (z.B. Random Access Machine (RAM))
2. Berechne Anzahl von Basisoperationen wie z.B. Anzahl an Vergleichen, Multiplikationen, Festplattenzugriffen etc.

Random Access Machine (RAM)

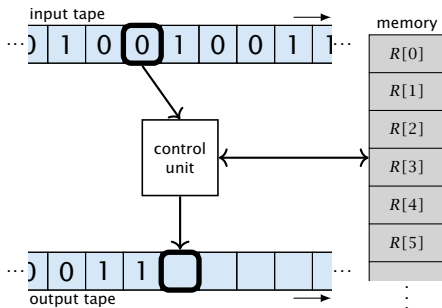
- ▶ Ein- und Ausgabeband (Folge von Einsen und Nullen; unbeschränkte Länge).
- ▶ Speicher: unendlich viele Register $R[0], R[1], R[2], \dots$
- ▶ Register können beliebige Integer speichern.
- ▶ Indirekte Adressierung.



Ein- und Ausgabeband sind gerichtet und ein Lese- oder Schreibzugriff bewegt das entsprechend Band zur nächsten Position.

Random Access Machine (RAM)

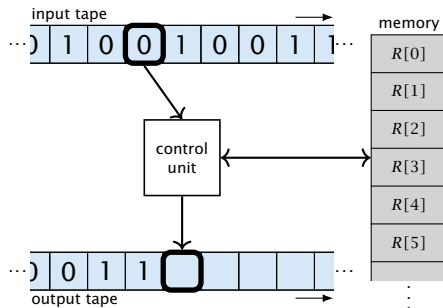
- ▶ Ein- und Ausgabeband (Folge von Einsen und Nullen; unbeschränkte Länge).
- ▶ Speicher: unendlich viele Register $R[0], R[1], R[2], \dots$
 - ▶ Register können beliebige Integer speichern.
 - ▶ Indirekte Adressierung.



Ein- und Ausgabeband sind gerichtet und ein Lese- oder Schreibzugriff bewegt das entsprechend Band zur nächsten Position.

Random Access Machine (RAM)

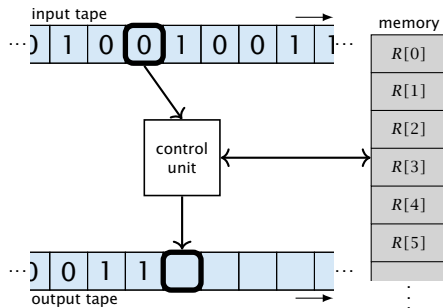
- ▶ Ein- und Ausgabeband (Folge von Einsen und Nullen; unbeschränkte Länge).
- ▶ Speicher: unendlich viele Register $R[0], R[1], R[2], \dots$
- ▶ Register können beliebige Integer speichern.
- ▶ Indirekte Adressierung.



Ein- und Ausgabeband sind gerichtet und ein Lese- oder Schreibzugriff bewegt das entsprechend Band zur nächsten Position.

Random Access Machine (RAM)

- ▶ Ein- und Ausgabeband (Folge von Einsen und Nullen; unbeschränkte Länge).
- ▶ Speicher: unendlich viele Register $R[0], R[1], R[2], \dots$
- ▶ Register können beliebige Integer speichern.
- ▶ Indirekte Adressierung.



Ein- und Ausgabeband sind gerichtet und ein Lese- oder Schreibzugriff bewegt das entsprechend Band zur nächsten Position.

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
- ▶ Registertransfers
- ▶ indirekte Adressierung

▶ $R[i]$ enthält den Inhalt des i -ten Registers in das j -te Register

▶ $R[j]$ enthält den Inhalt des i -ten Registers in das j -te Register

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
- ▶ Registertransfers
- ▶ indirekte Adressierung

▶ $R[i]$ enthält den Inhalt des i -ten Registers in das i -te Register

▶ $R[i]$ enthält den Inhalt des i -ten Registers in das j -te Register

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
- ▶ indirekte Adressierung

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
- ▶ indirekte Adressierung

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ indirekte Adressierung

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ indirekte Adressierung

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ indirekte Adressierung

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ **indirekte** Adressierung
 - ▶ $R[j] := R[R[i]]$
lädt den Inhalt des $R[i]$ -ten Registres in das j -te Register.
 - ▶ $R[R[i]] := R[j]$
lädt den Inhalt des j -ten Registers in das $R[i]$ -te Register

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ **indirekte** Adressierung
 - ▶ $R[j] := R[R[i]]$
lädt den Inhalt des $R[i]$ -ten Registres in das j -te Register.
 - ▶ $R[R[i]] := R[j]$
lädt den Inhalt des j -ten Registers in das $R[i]$ -te Register

Random Access Machine (RAM)

Operationen

- ▶ Eingabeoperationen (input tape $\rightarrow R[i]$)
 - ▶ READ i
- ▶ Ausgabeoperationen ($R[i] \rightarrow$ output tape)
 - ▶ WRITE i
- ▶ Registertransfers
 - ▶ $R[j] := R[i]$
 - ▶ $R[j] := 4$
- ▶ **indirekte** Adressierung
 - ▶ $R[j] := R[R[i]]$
lädt den Inhalt des $R[i]$ -ten Registres in das j -te Register.
 - ▶ $R[R[i]] := R[j]$
lädt den Inhalt des j -ten Registers in das $R[i]$ -te Register

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz x R[i]`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz x $R[i]$`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz $x R[i]$`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz $x R[i]$`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz $x R[i]$`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$

▶ $R[i] := R[j] + R[k];$
▶ $R[i] := -R[k];$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Random Access Machine (RAM)

Operationen

- ▶ Verzweigungen (inklusive Schleifen) abhängig von Vergleichen
 - ▶ `jump x`
springe zur Position x im Programm
setze Befehlszähler auf x
der nächste Befehl wird aus Register $R[x]$ gelesen
 - ▶ `jumpz $x R[i]$`
springe zu x falls $R[i] = 0$
falls nicht wird der Befehlszähler um 1 erhöht
 - ▶ `jumpi i`
springe zu $R[i]$ (indirekter Sprung);
- ▶ arithmetische Operationen: $+$, $-$, \times , $/$
 - ▶ $R[i] := R[j] + R[k];$
 $R[i] := -R[k];$

Die Sprungbefehle sind sehr ähnlich zu den Sprungbefehlen in verschiedenen Assemblersprachen.

Rechenmodell

Man nimmt normalerweise an, dass jeder Befehl eine Zeiteinheit kostet.

Komplexitätsschranken

Es gibt **unterschiedliche Komplexitätsschranken**:

- ▶ **best-case** Komplexität:

$$C_{bc}(n) := \min\{C(x) \mid |x| = n\}$$

Normalerweise einfach zu analysieren; nicht sehr hilfreich

- ▶ **worst-case** Komplexität:

$$C_{wc}(n) := \max\{C(x) \mid |x| = n\}$$

Standard. Manchmal zu pessimistisch.

- ▶ **average case** Komplexität:

$$C_{avg}(n) := \frac{1}{|I_n|} \sum_{|x|=n} C(x)$$

Manchmal schwierig zu analysieren.

$C(x)$	Kosten für Eingabe x
$ x $	Eingabelänge von x
I_n	Menge der Eingaben mit Länge n

Komplexitätsschranken

Es gibt **unterschiedliche Komplexitätsschranken**:

- ▶ **best-case** Komplexität:

$$C_{bc}(n) := \min\{C(x) \mid |x| = n\}$$

Normalerweise einfach zu analysieren; nicht sehr hilfreich

- ▶ **worst-case** Komplexität:

$$C_{wc}(n) := \max\{C(x) \mid |x| = n\}$$

Standard. Manchmal zu pessimistisch.

- ▶ **average case** Komplexität:

$$C_{avg}(n) := \frac{1}{|I_n|} \sum_{|x|=n} C(x)$$

Manchmal schwierig zu analysieren.

$C(x)$	Kosten für Eingabe x
$ x $	Eingabelänge von x
I_n	Menge der Eingaben mit Länge n

Komplexitätsschranken

Es gibt **unterschiedliche Komplexitätsschranken**:

- ▶ **best-case** Komplexität:

$$C_{bc}(n) := \min\{C(x) \mid |x| = n\}$$

Normalerweise einfach zu analysieren; nicht sehr hilfreich

- ▶ **worst-case** Komplexität:

$$C_{wc}(n) := \max\{C(x) \mid |x| = n\}$$

Standard. Manchmal zu pessimistisch.

- ▶ **average case** Komplexität:

$$C_{avg}(n) := \frac{1}{|I_n|} \sum_{|x|=n} C(x)$$

Manchmal schwierig zu analysieren.

$C(x)$	Kosten für Eingabe x
$ x $	Eingabelänge von x
I_n	Menge der Eingaben mit Länge n

Komplexitätsschranken

Es gibt **unterschiedliche Komplexitätsschranken**:

- ▶ **best-case** Komplexität:

$$C_{bc}(n) := \min\{C(x) \mid |x| = n\}$$

Normalerweise einfach zu analysieren; nicht sehr hilfreich

- ▶ **worst-case** Komplexität:

$$C_{wc}(n) := \max\{C(x) \mid |x| = n\}$$

Standard. Manchmal zu pessimistisch.

- ▶ **average case** Komplexität:

$$C_{avg}(n) := \frac{1}{|I_n|} \sum_{|x|=n} C(x)$$

Manchmal schwierig zu analysieren.

$C(x)$	Kosten für Eingabe x
$ x $	Eingabelänge von x
I_n	Menge der Eingaben mit Länge n