technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Weiterführende Fragestellungen

Grundlegende Fragestellungen

- Kann man zeigen, dass ein Problem nicht effizient(er) gelöst werden kann?
- Wie geht man mit Problemen um, die man nicht effizient lösen kann?

Untere Schranken

Vergleichsbasiertes Sortieren

- Ein Algorithmus ist ein vergleichsbasierter Sortierer, wenn er
- (1) für eine Eingabe von n unterschiedlichen Zahlen $a_1, \dots, a_n$ eine Reihenfolge π berechnet, so dass $a_{\pi(1)} < \cdots < a_{\pi(n)}$ gilt und
- (2) wenn sich die Reihenfolge bereits zwingend aus den vom Algorithmus durchgeführten Vergleichen (<,>) zwischen Eingabeelementen ergibt

Untere Schranken

Vergleichsbasiertes Sortieren

- Ein Algorithmus ist ein vergleichsbasierter Sortierer, wenn er
- (1) für eine Eingabe von n unterschiedlichen Zahlen $a_1, \dots, a_n$ eine Reihenfolge π berechnet, so dass $a_{\pi(1)} < \cdots < a_{\pi(n)}$ gilt und
- (2) wenn sich die Reihenfolge bereits zwingend aus den vom Algorithmus durchgeführten Vergleichen (<,>) zwischen Eingabeelementen ergibt

Erste Beobachtung

- InsertionSort und MergeSort sind vergleichsbasiert
  (die Algorithmen führen zwar ≤ Operationen durch, wenn die Eingabe jedoch aus unterschiedlichen Zahlen besteht, kann man diese durch < bzw. > ersetzen)

Untere Schranken

Vergleichsbasiertes Sortieren

- Ein Algorithmus ist ein vergleichsbasierter Sortierer, wenn er
- (1) für eine Eingabe von n unterschiedlichen Zahlen $a_1, \dots, a_n$ eine Reihenfolge π berechnet, so dass $a_{\pi(1)} < \cdots < a_{\pi(n)}$ gilt und
- (2) wenn sich die Reihenfolge bereits zwingend aus den vom Algorithmus durchgeführten Vergleichen (<,>) zwischen Eingabeelementen ergibt

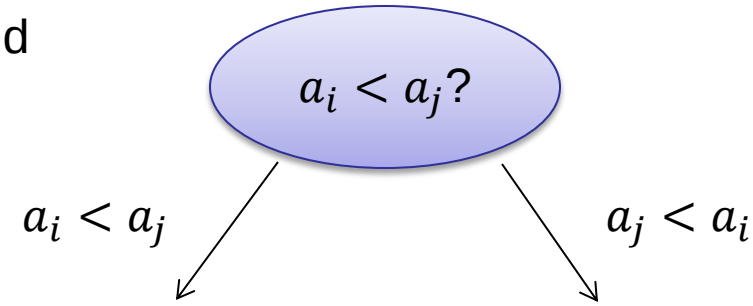
Zweite Beobachtung

- Jeder Vergleich benötigt $\boldsymbol{\Omega}(1)$ Zeit.
- Benötigt ein Algorithmus $f(n)$ Vergleiche, so ist seine Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$
- Ziel: Zeige, dass jeder vergleichsbasierte Sortierer $\boldsymbol{\Omega}(n \log n)$ Vergleiche benötigt

Untere Schranken

Baumdarstellung eines vergleichsbasierten Sortierers

- Wir geben Ablauf der Vergleiche an, die der Algorithmus bei Eingabe der Länge n ausführt
- Der Algorithmus führt einen eindeutigen ersten Vergleich aus, dieser wird Wurzel des Baum
- Je nach Ausgang des Vergleichs wird der Algorithmus auf unterschiedliche Weise fortgesetzt
- Das linke Kind eines Knotens entspricht dem Vergleichsausgang $a_i < a_j$, das rechte Kind dem Ausgang $a_i > a_j$
- Jedes Blatt wird mit einer Reihenfolge π bezeichnet



Untere Schranken

Baumdarstellung eines vergleichsbasierten Sortierers (Beispiel InsertionSort, $n = 3$*)*

InsertionSort(Array A)

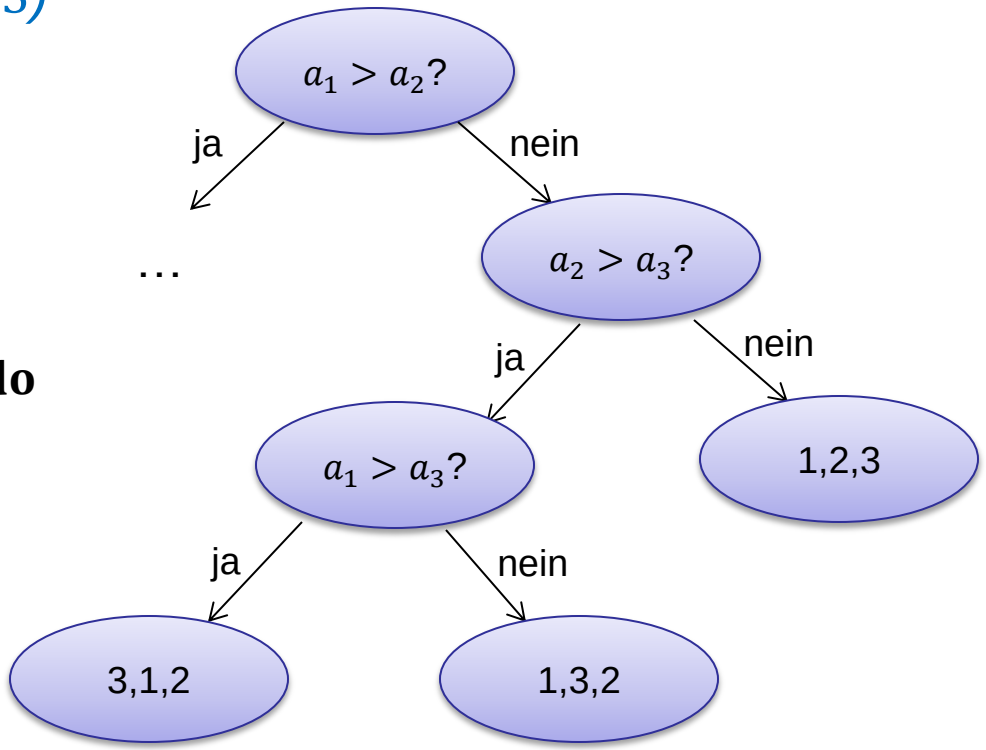
1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
2. key $\leftarrow A[j]$
3. $i \leftarrow j - 1$
4. **while** $i > 0$ and $A[i] >$ key **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i+1] \leftarrow$ key

Untere Schranken

Baumdarstellung eines vergleichsbasierten Sortierers
(Beispiel InsertionSort, $n = 3$*)*

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
2. key $\leftarrow A[j]$
3. $i \leftarrow j - 1$
4. **while** $i > 0$ and $A[i] >$ key **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i+1] \leftarrow$ key



Untere Schranken

Beobachtungen

- Die Tiefe eines Sortierbaums ist eine untere Schranke für die Worst-Case Laufzeit bei Eingabegröße n
- Jeder Sortierbaum ist ein Binärbaum
- Der Sortierbaum hat für jede Ausgabereihenfolge mindestens ein Blatt
- Es gibt $n!$ Ausgabereihenfolgen

Untere Schranken

Beobachtungen

- Die Tiefe eines Sortierbaums ist eine untere Schranke für die Worst-Case Laufzeit bei Eingabegröße n
- Jeder Sortierbaum ist ein Binärbaum
- Der Sortierbaum hat für jede Ausgabereihenfolge mindestens ein Blatt
- Es gibt $n!$ Ausgabereihenfolgen

Überlegung

- Wie tief ist ein Binärbaum mit $n!$ Blättern mindestens?
- Ein Binärbaum der Tiefe k hat höchstens 2^k Blätter (vollständiger Binärbaum)
- Umgekehrt: Ein Binärbaum mit $n!$ Blättern hat mindestens Tiefe $\log(n!)$

Untere Schranken

Korollar 74

- Jeder vergleichsbasierte Sortieralgorithmus hat eine Laufzeit von $\mathbf{\Omega}(n \log n)$.

Beweis

- Die Baumdarstellung eines vergleichsbasierten Sortieralgorithmus bei Eingabegröße n hat $n!$ Blätter und somit Tiefe $\mathbf{\Omega}(n \log n)$.

Untere Schranken

Korollar 74

- Jeder vergleichsbasierte Sortieralgorithmus hat eine Laufzeit von $\mathbf{\Omega}(n \log n)$.

Beweis

- Die Baumdarstellung eines vergleichsbasierten Sortieralgorithmus bei Eingabegröße n hat $n!$ Blätter und somit Tiefe $\mathbf{\Omega}(n \log n)$.
- Die Tiefe der Baumdarstellung gibt eine untere Schranke für die Worst-Case Laufzeit des Algorithmus, da der Algorithmus bei entsprechender Eingabe alle Vergleiche des längsten Astes durchführt und für jeden Vergleich $\mathbf{\Omega}(1)$ Zeit benötigt.

Untere Schranken

Korollar 74

- Jeder vergleichsbasierte Sortieralgorithmus hat eine Laufzeit von $\mathbf{\Omega}(n \log n)$.

Beweis

- Die Baumdarstellung eines vergleichsbasierten Sortieralgorithmus bei Eingabegröße n hat $n!$ Blätter und somit Tiefe $\mathbf{\Omega}(n \log n)$.
- Die Tiefe der Baumdarstellung gibt eine untere Schranke für die Worst-Case Laufzeit des Algorithmus, da der Algorithmus bei entsprechender Eingabe alle Vergleiche des längsten Astes durchführt und für jeden Vergleich $\mathbf{\Omega}(1)$ Zeit benötigt.
- Somit folgt das Korollar.

Untere Schranken

Korollar 74

- Jeder vergleichsbasierte Sortieralgorithmus hat eine Laufzeit von $\mathbf{\Omega}(n \log n)$.

Beweis

- Die Baumdarstellung eines vergleichsbasierten Sortieralgorithmus bei Eingabegröße n hat $n!$ Blätter und somit Tiefe $\mathbf{\Omega}(n \log n)$.
- Die Tiefe der Baumdarstellung gibt eine untere Schranke für die Worst-Case Laufzeit des Algorithmus, da der Algorithmus bei entsprechender Eingabe alle Vergleiche des längsten Astes durchführt und für jeden Vergleich $\mathbf{\Omega}(1)$ Zeit benötigt.
- Somit folgt das Korollar.

Aufgabe

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat

Untere Schranken

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat

Generelle Beweisidee

- Baue Algorithmus C der Problem B löst und dabei einen optimalen Algorithmus für Problem A als Unterprogramm benutzt

Untere Schranken

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$ hat

Generelle Beweisidee

- Baue Algorithmus C der Problem B löst und dabei einen optimalen Algorithmus für Problem A als Unterprogramm benutzt
- Zeige: Die Laufzeit des Algorithmus ist $\mathbf{o}\big(f(n)\big)$+Laufzeit für Problem A

Untere Schranken

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}(f(n))$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}(f(n))$ hat

Generelle Beweisidee

- Baue Algorithmus C der Problem B löst und dabei einen optimalen Algorithmus für Problem A als Unterprogramm benutzt
- Zeige: Die Laufzeit des Algorithmus ist $\mathbf{o}(f(n))$+Laufzeit für Problem A
- Dann hat jeder Algorithmus für Problem A eine Laufzeit von $\boldsymbol{\Omega}(f(n))$

Untere Schranken

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}(f(n))$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}(f(n))$ hat

Generelle Beweisidee

- Baue Algorithmus C der Problem B löst und dabei einen optimalen Algorithmus für Problem A als Unterprogramm benutzt
- Zeige: Die Laufzeit des Algorithmus ist $\mathbf{o}(f(n))$+Laufzeit für Problem A
- Dann hat jeder Algorithmus für Problem A eine Laufzeit von $\boldsymbol{\Omega}(f(n))$
- (Wäre dies nicht so, dann gäbe es einen Algorithmus mit Laufzeit $\mathbf{o}(f(n))$ für Problem A. Somit kann ich Problem B mit Algorithmus C in $\mathbf{o}(f(n))$ Zeit lösen. Widerspruch, da die Laufzeit für Problem B $\boldsymbol{\Omega}(f(n))$ ist)

Untere Schranken

Wie kann man untere Schranken für andere Probleme zeigen?

- Will zeigen, dass jeder Algorithmus für Problem A Laufzeit $\boldsymbol{\Omega}(f(n))$ hat
- Ich weiß, dass jeder Algorithmus für Problem B Laufzeit $\boldsymbol{\Omega}(f(n))$ hat

Generelle Beweisidee

- Baue Algorithmus C der Problem B löst und dabei einen optimalen Algorithmus für Problem A als Unterprogramm benutzt
- Zeige: Die Laufzeit des Algorithmus ist $\mathbf{o}(f(n))$+Laufzeit für Problem A
- Dann hat jeder Algorithmus für Problem A eine Laufzeit von $\boldsymbol{\Omega}(f(n))$
- (Wäre dies nicht so, dann gäbe es einen Algorithmus mit Laufzeit $\mathbf{o}(f(n))$ für Problem A. Somit kann ich Problem B mit Algorithmus C in $\mathbf{o}(f(n))$ Zeit lösen. Widerspruch, da die Laufzeit für Problem B $\boldsymbol{\Omega}(f(n))$ ist)

Untere Schranken

Satz 75

Die Berechnung der konvexen Hülle einer Punktmenge von n Punkten in der Ebene benötigt $\boldsymbol{\Omega}(n \log n)$ Zeit.

Beweis

- Annahme: Ich kann die konvexe Hülle von n Punkten in der Ebene in $f(n) = \mathbf{o}(n \log n)$ Zeit berechnen

Untere Schranken

Satz 75

Die Berechnung der konvexen Hülle einer Punktmenge von n Punkten in der Ebene benötigt $\boldsymbol{\Omega}(n \log n)$ Zeit.

Beweis

- Annahme: Ich kann die konvexe Hülle von n Punkten in der Ebene in $f(n) = \mathbf{o}(n \log n)$ Zeit berechnen
- Dann sei Algorithmus FastHull ein Algorithmus der dies tut

Untere Schranken

Satz 75

Die Berechnung der konvexen Hülle einer Punktmenge von n Punkten in der Ebene benötigt $\boldsymbol{\Omega}(n \log n)$ Zeit.

Beweis

- Annahme: Ich kann die konvexe Hülle von n Punkten in der Ebene in $f(n) = \mathbf{o}(n \log n)$ Zeit berechnen
- Dann sei Algorithmus $\mathrm{FastHull}$ ein Algorithmus der dies tut
- Zeige: Man kann dann Algorithmus $\mathrm{FastSort}$ konstruieren, der in $\mathbf{o}(n \log n)$ sortiert

Untere Schranken

Satz 75

Die Berechnung der konvexen Hülle einer Punktmenge von n Punkten in der Ebene benötigt $\boldsymbol{\Omega}(n \log n)$ Zeit.

Beweis

- Annahme: Ich kann die konvexe Hülle von n Punkten in der Ebene in $f(n) = \mathbf{o}(n \log n)$ Zeit berechnen
- Dann sei Algorithmus $\mathrm{FastHull}$ ein Algorithmus der dies tut
- Zeige: Man kann dann Algorithmus $\mathrm{FastSort}$ konstruieren, der in $\mathbf{o}(n \log n)$ sortiert
- Dies ist aufgrund unserer unteren Schranke nicht möglich
  (streng genommen gilt dies natürlich nur für vergleichsbasierte Algorithmen; wir schummeln also hier ein wenig)

Untere Schranken

Satz 75

Die Berechnung der konvexen Hülle einer Punktmenge von n Punkten in der Ebene benötigt $\boldsymbol{\Omega}(n \log n)$ Zeit.

Beweis

- Annahme: Ich kann die konvexe Hülle von n Punkten in der Ebene in $f(n) = \mathbf{o}(n \log n)$ Zeit berechnen
- Dann sei Algorithmus $\mathrm{FastHull}$ ein Algorithmus der dies tut
- Zeige: Man kann dann Algorithmus $\mathrm{FastSort}$ konstruieren, der in $\mathbf{o}(n \log n)$ sortiert
- Dies ist aufgrund unserer unteren Schranke nicht möglich
  (streng genommen gilt dies natürlich nur für vergleichsbasierte Algorithmen; wir schummeln also hier ein wenig)

Untere Schranken

Beweis

FastSort(A)

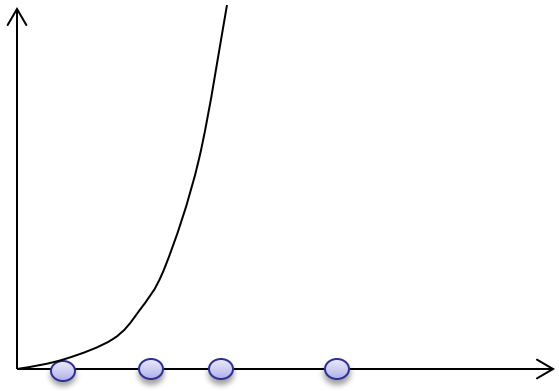
1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

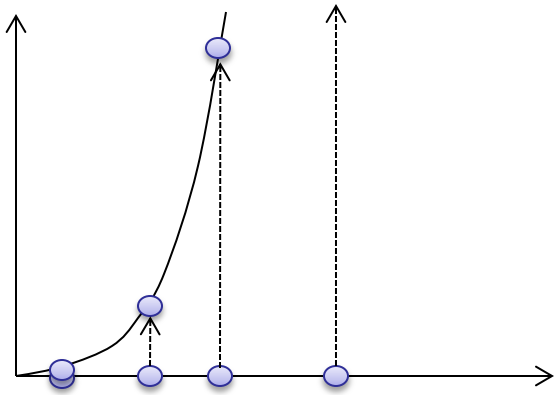


Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

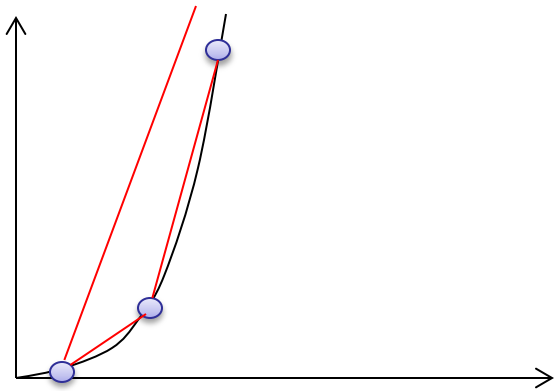


Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

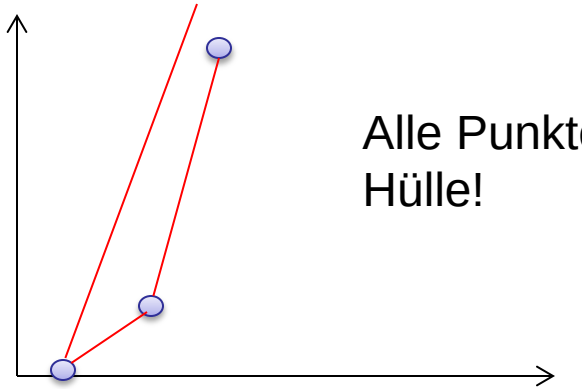


Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten



Alle Punkte liegen auf der konvexen Hülle!

Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

- Die Laufzeit von FastSort ist $\mathbf{O}(n) + f(n)$

Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

- Die Laufzeit von FastSort ist $\mathbf{O}(n) + f(n)$
- Hat also FastHull eine Laufzeit von $\mathbf{o}(n \log n)$, so hat auch FastSort eine solche Laufzeit

Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

- Die Laufzeit von FastSort ist $\mathbf{O}(n) + f(n)$
- Hat also FastHull eine Laufzeit von $\mathbf{o}(n \log n)$, so hat auch FastSort eine solche Laufzeit
- Da jeder (vergleichsbasierte) Sortieralgorithmus Laufzeit $\mathbf{\Omega}(n \log n)$ hat, kann dies nicht sein und FastHull (und jeder andere vergleichsbasierte Algorithmus zur Berechnung der konvexen Hülle) hat Laufzeit $\mathbf{\Omega}(n \log n)$.

Untere Schranken

Beweis

FastSort(A)

1. Initialisiere Feld B für n Punkte
2. **for** $i \leftarrow 1$ **to** n **do**
3. Für Zahl $x = A[i]$ schreibe Punkt (x, x^2) in Feld B
4. Berechne konvexe Hülle mit FastHull(B)
5. Gib Punkte in der Reihenfolge aus, in der sie auf der Hülle auftreten

- Die Laufzeit von FastSort ist $\mathbf{O}(n) + f(n)$
- Hat also FastHull eine Laufzeit von $\mathbf{o}(n \log n)$, so hat auch FastSort eine solche Laufzeit
- Da jeder (vergleichsbasierte) Sortieralgorithmus Laufzeit $\mathbf{\Omega}(n \log n)$ hat, kann dies nicht sein und FastHull (und jeder andere vergleichsbasierte Algorithmus zur Berechnung der konvexen Hülle) hat Laufzeit $\mathbf{\Omega}(n \log n)$.

Untere Schranken

3SUM

- Sei S eine Menge von n Integers. Gibt es 3 unterschiedliche Zahlen in S, die sich zu 0 aufsummieren?

Vermutung

- 3SUM kann nicht in $\mathbf{o}(n^2)$ Laufzeit gelöst werden

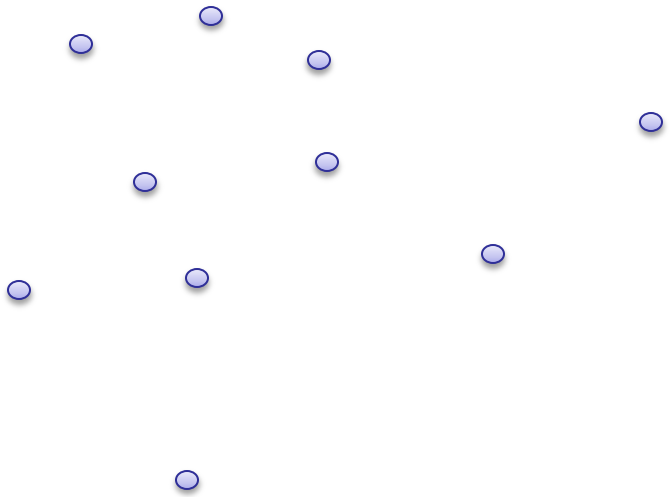
Kommentar

- Die Laufzeit hängt immer auch vom genauen Rechenmodell ab
- In bestimmten Rechenmodellen gibt es einen Algorithmus, dessen Laufzeit etwas besser ist als $\mathbf{O}(n^2)$

Untere Schranken

Kolinearitätsproblem

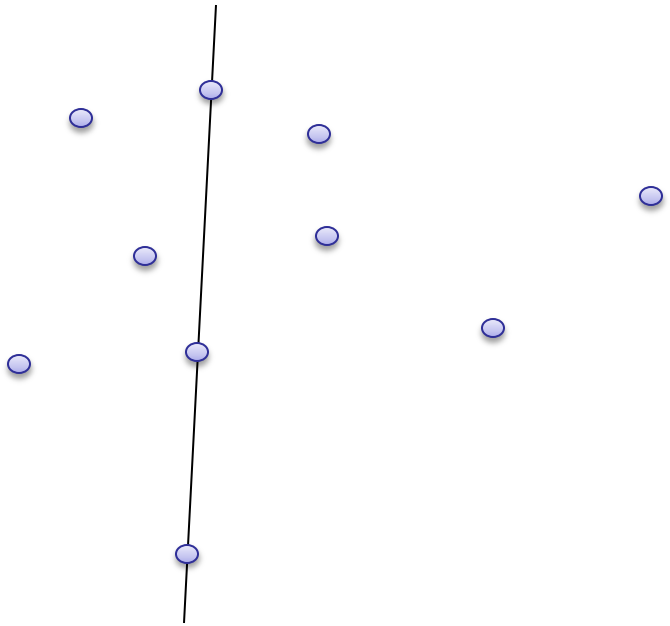
- Seien n Punkte in der Ebene gegeben. Gibt es 3 Punkte, die auf einer Linie liegen?



Untere Schranken

Kolinearitätsproblem

- Seien n Punkte in der Ebene gegeben. Gibt es 3 Punkte, die auf einer Linie liegen?



Untere Schranken

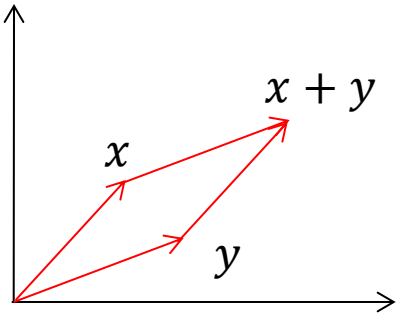
Wie testet man Kolinearität von 3 Punkten?

Untere Schranken

Wie testet man Kolinearität von 3 Punkten?

- Seien $x = (x_1, x_2)$ und $y = (y_1, y_2)$ zwei Punkte/Vektoren.
- Dann gibt

$$\det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$



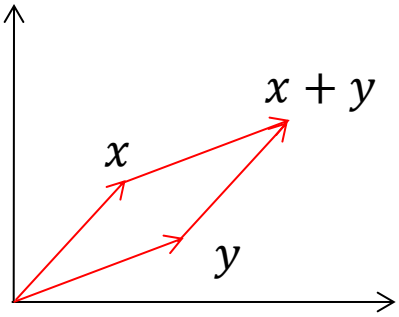
die Fläche des von den beiden Vektoren aufgespannten Parallelograms an

Untere Schranken

Wie testet man Kolinearität von 3 Punkten?

- Seien $x = (x_1, x_2)$ und $y = (y_1, y_2)$ zwei Punkte/Vektoren.
- Dann gibt

$$\det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$



die Fläche des von den beiden Vektoren aufgespannten Parallelograms an

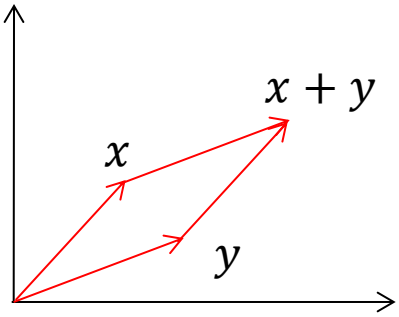
- Sind die Vektoren x und y linear abhängig, so hat das Parallelogram Fläche 0

Untere Schranken

Wie testet man Kolinearität von 3 Punkten?

- Seien $x = (x_1, x_2)$ und $y = (y_1, y_2)$ zwei Punkte/Vektoren.
- Dann gibt

$$\det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$



die Fläche des von den beiden Vektoren aufgespannten Parallelograms an

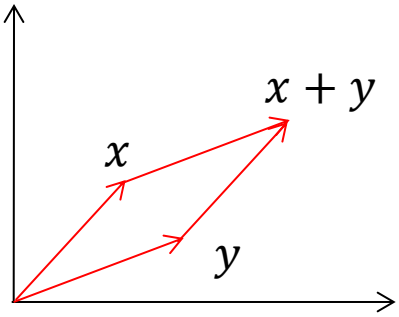
- Sind die Vektoren x und y linear abhängig, so hat das Parallelogram Fläche 0
- Hat man nun drei Punkte a, b, c, so kann man Kolinerität testen, indem man testet, ob $b - a$ und $c - a$ linear abhängig sind

Untere Schranken

Wie testet man Kolinearität von 3 Punkten?

- Seien $x = (x_1, x_2)$ und $y = (y_1, y_2)$ zwei Punkte/Vektoren.
- Dann gibt

$$\det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$



die Fläche des von den beiden Vektoren aufgespannten Parallelograms an

- Sind die Vektoren x und y linear abhängig, so hat das Parallelogram Fläche 0
- Hat man nun drei Punkte a, b, c, so kann man Kolinerität testen, indem man testet, ob $b - a$ und $c - a$ linear abhängig sind

Untere Schranken

Satz 76

Sei $f(n)$ eine untere Schranke für die Worst-Case Laufzeit des besten Algorithmus für 3SUM. Dann hat auch das Kolinearitätsproblem eine Laufzeit von $\boldsymbol{\Omega}\big(f(n)\big)$.

Untere Schranken

Satz 76

Sei $f(n)$ eine untere Schranke für die Worst-Case Laufzeit des besten Algorithmus für 3SUM. Dann hat auch das Kolinearitätsproblem eine Laufzeit von $\boldsymbol{\Omega}(f(n))$.

Beweis

- Sei $\mathrm{Kolinear}$ ein optimaler Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n)$.

Untere Schranken

Satz 76

Sei $f(n)$ eine untere Schranke für die Worst-Case Laufzeit des besten Algorithmus für 3SUM. Dann hat auch das Kolinearitätsproblem eine Laufzeit von $\boldsymbol{\Omega}\big(f(n)\big)$.

Beweis

- Sei $\mathrm{Kolinear}$ ein optimaler Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n)$.
- Wir entwerfen zunächst einen Algorithmus $3\mathrm{SUM\text{-}Fast}$ für 3SUM mit Laufzeit $g(n) + \mathbf{O}(n)$, der Algorithmus $\mathrm{Kolinear}$ benutzt.

Untere Schranken

Satz 76

Sei $f(n)$ eine untere Schranke für die Worst-Case Laufzeit des besten Algorithmus für 3SUM. Dann hat auch das Kolinearitätsproblem eine Laufzeit von $\boldsymbol{\Omega}\big(f(n)\big)$.

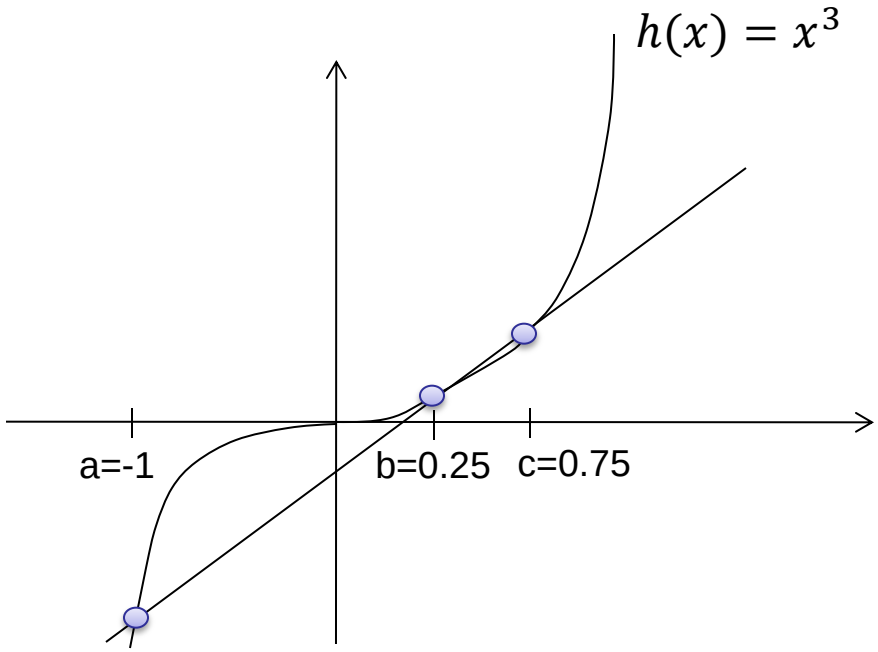
Beweis

- Sei $\mathrm{Kolinear}$ ein optimaler Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n)$.
- Wir entwerfen zunächst einen Algorithmus $3\mathrm{SUM\text{-}Fast}$ für 3SUM mit Laufzeit $g(n) + \mathbf{O}(n)$, der Algorithmus $\mathrm{Kolinear}$ benutzt.

Untere Schranken

3SUM-Fast(S)

1. $P \leftarrow \emptyset$
2. Für jede Zahl $x \in S$ füge Punkt (x, x^3) zu Punktmenge P hinzu
3. **return** Kolinear(P)



Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Die Laufzeit folgt sofort, da nur die n Zahlen aus S in n Punkte umgeformt werden müssen und dann Kolinear aufgerufen wird.

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Die Laufzeit folgt sofort, da nur die n Zahlen aus S in n Punkte umgeformt werden müssen und dann Kolinear aufgerufen wird.
- Wir müssen zeigen, dass die Punkte (a, a^3), (b, b^3) und (c, c^3) genau dann kollinear sind, wenn $a + b + c = 0$ ist (d.h. 3SUM erfüllt ist). Dabei sind a, b, c unterschiedliche Zahlen aus S

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Die Laufzeit folgt sofort, da nur die n Zahlen aus S in n Punkte umgeformt werden müssen und dann Kolinear aufgerufen wird.
- Wir müssen zeigen, dass die Punkte (a, a^3), (b, b^3) und (c, c^3) genau dann kollinear sind, wenn $a + b + c = 0$ ist (d.h. 3SUM erfüllt ist). Dabei sind a, b, c unterschiedliche Zahlen aus S

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Um zu überprüfen, ob die 3 Punkte kollinear sind, rechnen wir die Determinante von $(b - a, b^3 - a^3), (c - a, c^3 - a^3)$ aus

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Um zu überprüfen, ob die 3 Punkte kollinear sind, rechnen wir die Determinante von $(b-a, b^3-a^3), (c-a, c^3-a^3)$ aus

$$\det\begin{pmatrix} b-a & b^3-a^3 \\ c-a & c^3-a^3 \end{pmatrix} = (b-a)(c^3-a^3) - (c-a)(b^3-a^3)$$

$$= -(\mathrm{a}-\mathrm{b})(\mathrm{a}-\mathrm{c})(\mathrm{b}-\mathrm{c})(\mathrm{a}+\mathrm{b}+\mathrm{c})$$

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Um zu überprüfen, ob die 3 Punkte kollinear sind, rechnen wir die Determinante von $(b-a, b^3-a^3), (c-a, c^3-a^3)$ aus

$$\det\begin{pmatrix} b-a & b^3-a^3 \\ c-a & c^3-a^3 \end{pmatrix} = (b-a)(c^3-a^3) - (c-a)(b^3-a^3)$$

$$= -(\mathrm{a}-\mathrm{b})(\mathrm{a}-\mathrm{c})(\mathrm{b}-\mathrm{c})(\mathrm{a}+\mathrm{b}+\mathrm{c})$$

- Da a, b und c unterschiedliche Zahlen sind, wird dieses Polynom genau dann 0, wenn $a + b + c = 0$ ist.

Untere Schranken

Behauptung

3SUM-Fast löst das 3SUM Problem in Laufzeit $g(n) + \mathbf{O}(n)$.

Beweis

- Um zu überprüfen, ob die 3 Punkte kollinear sind, rechnen wir die Determinante von $(b-a, b^3-a^3), (c-a, c^3-a^3)$ aus

$$\det\begin{pmatrix} b-a & b^3-a^3 \\ c-a & c^3-a^3 \end{pmatrix} = (b-a)(c^3-a^3) - (c-a)(b^3-a^3)$$

$$= -(\mathrm{a}-\mathrm{b})(\mathrm{a}-\mathrm{c})(\mathrm{b}-\mathrm{c})(\mathrm{a}+\mathrm{b}+\mathrm{c})$$

- Da a, b und c unterschiedliche Zahlen sind, wird dieses Polynom genau dann 0, wenn $a + b + c = 0$ ist.

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$
- Angenommen, es gibt einen Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n) = \mathbf{o}\big(f(n)\big)$

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$
- Angenommen, es gibt einen Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n) = \mathbf{o}\big(f(n)\big)$
- Dann hat Algorithmus 3SUM-Fast eine Laufzeit $g(n) + \mathbf{O}(n) = \mathbf{o}\big(f(n)\big)$.

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$
- Angenommen, es gibt einen Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n) = \mathbf{o}\big(f(n)\big)$
- Dann hat Algorithmus 3SUM-Fast eine Laufzeit $g(n) + \mathbf{O}(n) = \mathbf{o}\big(f(n)\big)$.
- Widerspruch zur Annahme, dass $f(n)$ eine untere Schranke für die Laufzeit des besten 3SUM Algorithmus ist.

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$
- Angenommen, es gibt einen Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n) = \mathbf{o}\big(f(n)\big)$
- Dann hat Algorithmus 3SUM-Fast eine Laufzeit $g(n) + \mathbf{O}(n) = \mathbf{o}\big(f(n)\big)$.
- Widerspruch zur Annahme, dass $f(n)$ eine untere Schranke für die Laufzeit des besten 3SUM Algorithmus ist.
- Also hat jeder Algorithmus für das Kolinearitätsproblem Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$

Untere Schranken

Beweis von Satz 76 (fortgesetzt)

- Wir nehmen nun an, dass $f(n)$ die Laufzeit des besten 3SUM Algorithmus ist. Ist $f(n) = \mathbf{O}(n)$, so müssen wir nichts zeigen, da man sich für das Lösen des Kolinearitätsproblems alle Eingabepunkte angucken muss
- Sei also $f(n) = \boldsymbol{\omega}(n)$
- Angenommen, es gibt einen Algorithmus für das Kolinearitätsproblem mit Laufzeit $g(n) = \mathbf{o}\big(f(n)\big)$
- Dann hat Algorithmus 3SUM-Fast eine Laufzeit $g(n) + \mathbf{O}(n) = \mathbf{o}\big(f(n)\big)$.
- Widerspruch zur Annahme, dass $f(n)$ eine untere Schranke für die Laufzeit des besten 3SUM Algorithmus ist.
- Also hat jeder Algorithmus für das Kolinearitätsproblem Laufzeit $\boldsymbol{\Omega}\big(f(n)\big)$

Untere Schranken

Die 1.000.000$ Frage

- Zeigen Sie, dass es keine Konstante c gibt, so dass das Rucksackproblem in $\mathbf{O}(n^c)$ Zeit gelöst werden kann
- Dabei dürfen die Eingabezahlen exponentiell in n groß sein!

Untere Schranken

Zusammenfassung

- Für einige (sehr wenige) Probleme können wir untere Schranken beweisen
- Die Schranken können vom gewählten Modell abhängen
- Will man zeigen, dass ein Problem A schwer ist und hat man bereits eine untere Schranke für ein anderes Problem B, so kann man einen Algorithmus entwickeln, der Problem B mit Hilfe von Problem A löst und so die Schwierigkeit der Probleme zueinander in Relation setzen
- Es ist i.a. sehr schwer untere Schranken zu zeigen