

Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2023 · tcs.uni-frankfurt.de/algo1/ · 2023-07-06 · ed6968a








Suchen und Sortieren (Woche 2)

Eigenständige Vorbereitung:

Lies  CLRS Kapitel 2 und schau dir die  Videos der Woche an. Beantworte dabei die folgenden Leitfragen:

- Wieso können wir sicher sein, dass binäre Suche immer das gesuchte Element findet, obwohl er sich nicht alle Elemente des Arrays anschaut? Wie viele Elemente schaut er sich an?
- An welchen Stellen in der Analyse von binärer Suche, insertion sort oder merge sort wird der kleine Satz von Gauß bzw. die geometrische Reihe benutzt?

Zeichenlegende:

-  Schriftliche Aufgabe, die du fristgerecht in Moodle abgibst. In der Klausur wirst du alle Aufgaben schriftlich bearbeiten, daher ist das Feedback der Tutoren wichtig, damit du deine Schreibfähigkeiten verbessern kannst.
-  Diese Art von Aufgabe musst du sicher können, um die Klausur zu bestehen.
-  Diese Art von Aufgabe musst du weitgehend können, um die Klausur zu bestehen.
-  Diese Art von Aufgabe musst du können, um eine gute Note zu erhalten.
-  Diese Aufgabe ist als Knobelspaß gedacht, der das algorithmische Verständnis vertieft.

Aufgabe 2.1 (Von Hand laufen lassen und Eigenschaften).

- (einfach) Beschreibe den Ablauf von insertion sort, wenn die Eingabe ein Feld $A = [31, 41, 59, 26, 41, 58]$ ist.
- (einfach) Verändere den Pseudocode für insertion sort, um das Eingabefeld monoton fallend anstatt monoton wachsend zu ordnen.
- Analysiere die Laufzeit von linearer Suche *im mittleren Fall*: Wenn das gesuchte Element x zufällig und gleichwahrscheinlich irgendeines der Elemente von A ist, wie viele Einträge von A prüft lineare Suche dann, um x zu finden?
- (einfach) Beschreibe den Ablauf von merge sort auf dem Feld $A = [3, 41, 52, 26, 38, 57, 9, 49]$.
- Überzeuge dich, dass insertion sort auch rekursiv wie folgt formuliert werden kann: um $A[0..n-1]$ zu sortieren, sortieren wir $A[0..n-2]$ rekursiv und fügen dann $A[n-1]$ in das sortierte Feld $A[0..n-2]$ ein. Stelle die Rekursionsgleichung für die Laufzeit dieses Algorithmus auf und löse sie.
- Ein Freund schlägt vor, dass du binäre Suche nutzen solltest, um den Einfügeschritt von insertion sort schneller zu machen. Funktioniert das, und welchen Effekt hat es auf die Laufzeit des Algorithmus?

Aufgabe 2.2 (Duplikate und nahe Nachbarn). Sei $A[0..n-1]$ ein Feld von ganzen Zahlen.

- (einfach) Ein *Duplikat* in A ist ein Paar (i, j) von unterschiedlichen Indizes mit $A[i] = A[j]$. Entwirf einen Algorithmus an, der in Zeit $O(n^2)$ findet, ob A ein Duplikat hat.
- Entwirf einen Algorithmus an, der in Zeit $O(n \log n)$ findet, ob A ein Duplikat hat. *Hinweis*: Nutze merge sort.
- Ein *nächstes Paar* in A ist ein Paar (i, j) mit $i \neq j$, sodass der Abstand $|A[i] - A[j]|$ minimal ist unter allen Paaren von Einträgen. Entwerfe einen Algorithmus, der den Abstand eines nächsten Pairs in Zeit $O(n \log n)$ findet.

Aufgabe 2.3 (Korrektheit von Merge Sort 🚩). Beweise, dass Merge Sort alle gegebenen Felder korrekt sortiert. Hierbei darfst du annehmen, dass die Verflechtungsoperation MERGE auf sortierten Feldern korrekt arbeitet. *Hinweis: benutze vollständige Induktion.*

Aufgabe 2.4 (2-Summe und 3-Summe). Sei $A[0..n-1]$ ein Feld von ganzen Zahlen (positive und negative Zahlen sind erlaubt). Das Feld A hat eine *2-Summe*, wenn es zwei Einträge i und j gibt mit $A[i] + A[j] = 0$. Analog hat A eine *3-Summe*, wenn es drei Einträge i , j , und k gibt, sodass $A[i] + A[j] + A[k] = 0$.

- 👍 Gib einen einfachen Algorithmus an, der in Zeit $O(n^2)$ feststellt, ob A eine 2-Summe hat.
- 🚩 Gib einen Algorithmus an, der in Zeit $O(n \log n)$ feststellt, ob A eine 2-Summe hat. *Hinweis: binäre Suche.*
- 👍 Gib einen Algorithmus an, der in Zeit $O(n^3)$ feststellt, ob A eine 3-Summe hat.
- 🚩 Gib einen Algorithmus an, der in Zeit $O(n^2 \log n)$ feststellt, ob A eine 3-Summe hat. *Hinweis: binäre Suche.*
- 🚩 (sehr schwer) Gib einen Algorithmus an, der in Zeit $O(n^2)$ feststellt, ob A eine 3-Summe hat.

Aufgabe 2.5 (Auswahl, Partition, und Quick Sort). Sei $A[0..n-1]$ ein Feld von unterschiedlichen ganzen Zahlen. Der Eintrag von *Rang* k in A ist die k t kleinste Zahl unter den Zahlen in A . Der *Median* von A ist die Zahl von Rang $\lceil (n/2) \rceil$.


- 🚩 Gib einen Algorithmus mit Laufzeit $O(n \log n)$ an, der A und k als Eingabe nimmt und die Zahl in A ausgibt, die Rang k hat.

Eine *Partition* von A ist eine Aufteilung in Felder A_{low} und A_{high} , sodass A_{low} alle Zahlen von A enthält, die kleiner oder gleich dem Median von A sind, und A_{high} alle Zahlen von A enthält, die größer sind als der Median von A . Es gibt einen Linearzeitalgorithmus, der den Median eines Feldes findet; diesen Algorithmus darfst du im Folgenden einfach annehmen.

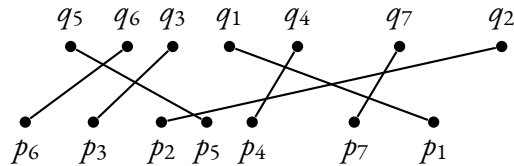
- 👍 Gib einen Algorithmus an, der eine Partition von A in Zeit $O(n)$ ausrechnet.
- 🚩 Gib einen Algorithmus an, der A in Zeit $O(n \log n)$ sortiert, indem er rekursiv partitioniert.
- 🚩 (sehr schwer) Gib einen Algorithmus mit Laufzeit $O(n)$ an, der A und k als Eingabe nimmt und die Zahl in A ausgibt, die Rang k hat.

Aufgabe 2.6 (k -Merge Sort). Professorin M. Erge stellt ihren neuen Algorithmus, 3-Merge Sort, vor. 3-Merge Sort funktioniert genau wie das uns bekannte Merge Sort, nur wird rekursiv in drei Teile anstatt zwei aufgeteilt, die daraufhin sortiert und wieder verflochten werden. Löse die folgenden Teilaufgaben.


- 🚩 Zeige, dass sich drei sortierte Felder in asymptotischer Linearzeit verflechten lassen.
- 👍 Führe eine Laufzeitanalyse für 3-Merge Sort durch. Stelle hierzu die Rekursionsgleichung für die Laufzeit $T(n)$ auf und löse diese.
- 🚩 Verallgemeinere den Algorithmus und die Analyse von 3-Merge Sort zu k -Merge Sort für $k > 3$. Hat Prof. M. Erge den Durchbruch geschafft? Stellt k -Merge Sort eine Verbesserung gegenüber dem klassischen 2-Merge Sort dar?

Aufgabe 2.7 (Kreuzende Linien ). Gegeben seien zwei Sequenzen von Punkten in der $x y$ -Ebene, nämlich p_1, \dots, p_n auf der Linie $y = 0$ und q_1, \dots, q_n auf der Linie $y = 1$. Erzeuge nun n Strecken, indem jeder Punkt p_i mit q_i verbunden wird. Beschreibe und analysiere einen *divide-and-conquer* Algorithmus, der in Zeit $O(n \log n)$ ausrechnet, wie viele Paare von Strecken sich schneiden.

Zum Beispiel gibt es unter den folgenden 7 Strecken genau 8 Paare von Strecken, die sich schneiden:



Die Eingabe soll gegeben sein als zwei Felder $p[1..n]$ und $q[1..n]$ von ganzen Zahlen, sodass $p[i]$ die x -Koordinate von p_i beschreibt und $q[j]$ die x -Koordinate von q_j . Du darfst annehmen, dass alle x -Koordinaten verschieden sind.

Anforderungen an schriftliche Abgaben. In der Klausur gibt es zum Bestehen notwendige -Aufgaben, in denen du die Korrektheit und die Laufzeit eines Algorithmus beweisen musst. In den schriftlichen Abgaben übst du, wie man das richtig macht. Es gibt bewusst nur eine schriftliche Aufgabe pro Woche, damit du dich intensiv mit dem Schreiben beschäftigen kannst. Achte hierbei auf die folgenden Aspekte:

Autor:innen. Alle Autor:innen des Dokuments sind namentlich genannt.

Schreibstil. Dein Text sollte durchgehend in grammatikalisch korrekten und ganzen Sätzen geschrieben sein. Achte darauf, dass alle Sätze möglichst kurz sind und keine zu verschachtelten Nebensätze enthalten. Wissenschaftssprache ist dann am besten, wenn sie stilistisch gesehen *einfach* ist. *If you prefer to submit your solutions in English, please do so.*

Struktur. Dein Text muss nachvollziehbar und zielgerichtet strukturiert sein. Die Lösung zu einer Algorithmenaufgabe ist typischerweise wie folgt strukturiert:

- Grobe Idee: Zunächst beschreibst du in einer **kurzen** Einleitung die Idee des Algorithmus in natürlicher Sprache auf. Ein kurzer Absatz genügt in den meisten Fällen.
- Formale Beschreibung: Du beschreibst den Algorithmus in Pseudocode oder Code.
- Korrektheitsbeweis: Du beweisst die Korrektheit des Algorithmus.
- Laufzeitanalyse: Du analysierst die Laufzeit des Algorithmus.

Insbesondere sollten diese vier Aspekte klar getrennt sein und nicht ineinander fließen.

Korrektheit und Präzision. Dein Text darf keine Widersprüche oder unzulässigen Folgerungen beinhalten, er muss stets präzise und eindeutig sein. **Stell dir bei jedem Satz die Frage, wie ein Leser ihn missverstehen könnte, und formuliere den Satz so lange um, bis keine Missverständnisse mehr möglich sind.**

Zielorientiert. Dein Text muss durchgehend zielgerichtet und kompakt sein. Stell dir bei jedem Satz die Frage, ob er nicht doch weg gelassen oder gekürzt werden kann. Irrelevante Ausführungen führen in der Klausur selbst dann zu Punktabzug, wenn sie nicht falsch sind.

Leserliche Handschrift. (falls relevant) Die Handschrift ist durchgehend ohne Mühe lesbar.

Mathematik. Mathematische Ausdrücke sind korrekt gesetzt (z.B. n^2 anstatt $n^{\wedge}2$) und die mathematischen Ausdrücke sind grammatikalisch korrekt in die Satzstruktur eingebettet.

Wissenschaftssprache. Der sprachliche Ausdruck ist sachorientiert mit treffender Wortwahl und präzisen Formulierungen. Fachbegriffe werden einheitlich, präzise und auf den Inhalt der Kurse bezogen verwendet.