Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2026
Musterlösung Übungsblatt 1

Abgabe: Dienstag, 28. April 2026, 10:00 Uhr

Aufgabe 1: Anmeldung *(5 Punkte)*

Melden Sie sich beim Kurssystem Daphne an. Den Link dazu finden Sie auch auf der Kurs-Website. Achten Sie darauf, dass Ihre Daten korrekt sind, insbesondere, dass Sie unter der angegebenen E-Mail-Adresse auch erreichbar sind. Führen Sie ein `checkout` auf Ihr SVN-Repository durch.[1]
Bitte registrieren Sie sich auch bei Zulip, indem Sie den Link auf der Webseite benutzen. Zulip wird als Forum für Fragen zur Vorlesung, zu den Übungsblättern und zu organisatorischen Dingen verwendet. Bei Fragen am besten immer hier melden, damit auch andere von der Antwort profitieren können.

Aufgabe 2: Sortieren und Zeitmessungen *(5 Punkte)*

In der Vorlesung haben Sie bereits mehrere Sortieralgorithmen kennengelernt. In dieser Übung geht es darum solche zu Implementieren, Testfälle dafür zu schreiben und die Laufzeit für verschiedene Algorithmen zu messen und zu vergleichen. Konkret sollen Sie sowohl den *Selectionsort*- als auch den *Mergesort*-Algorithmus implementieren. Verwenden Sie dazu die auf der Webseite verlinkte Design-Vorlage `Sort.py` sowie die `Makefile`[2].

- Schreiben Sie Unit-Tests[3] für beide Algorithmen und korrigieren Sie Ihren Code, sollten Sie Fehler finden. Die Unit-Tests sollten grundsätzlich mindestens ein nicht triviales Beispiel überprüfen. Wenn es kritische Grenzfälle gibt, die sich leicht nachprüfen lassen (z. B. Verhalten einer Methode bei leerem Eingabefeld), sollen Sie dies tun. Es macht hier Sinn, auch einige Testfälle zu haben, die speziell für einen der beiden Algorithmen relevant sind, z. B. um zu prüfen, ob Mergesort auch korrekt sortiert, wenn die Größe des Arrays ungerade bzw. gerade ist.

- Messen Sie die Laufzeit Ihrer beiden Implementierungen. Nutzen Sie dafür den Code aus der Vorlage, um zufällige Arrays der Größe n zu generieren. Stellen Sie die Laufzeit über n grafisch dar, indem Sie Arrays der Länge 100 bis 10000 (zum Beispiel inkrementell in 100er-Schritten) zufällig generieren und die Zeit messen.[4] Diskutieren Sie Ihre Ergebnisse kurz in Ihrer `erfahrungen.md` (bzw. `erfahrungen.txt`) (siehe Aufgabe 4).

[1] Ihr SVN-Repository wird bei der ersten Anmeldung bei Daphne automatisch angelegt. Die URL ist: `https://daphne.tf.uni-freiburg.de/ss2026/AlgoDat/svn/ihr-rz-account-name`

[2] Mit dem Befehl `make` können Unit-Tests sowie Checkstyle nach der Flake8 Konvention geprüft werden. Bitte nutzen Sie dies um den Tutoren die Korrektur zu erleichtern.

[3] Nutzen Sie dafür `doctests`, wie sie auch schon in der Vorlage zu finden sind. Mit `python -m doctest Sort.py` oder alternativ mit `make` können Sie Ihren Code testen.

[4] Die Unterschiede der Laufzeiten der Algorithmen werden am deutlichsten, wenn diese gemeinsam in einem einzigen Schaubild aufgetragen werden, mit n auf der x-Achse und der Laufzeit der entsprechenden Variante auf der y-Achse.

Musterlösung

Auf der Webseite finden Sie die Musterlösungen der Programmieraufgaben in der Sort.py Datei.
Abbildung 1 stellt die Laufzeit der beiden Sortieralgorithmen auf zufällig generierten Arrays der Größe 100 bis 10000 dar. Im oberen Diagramm ist die Laufzeit auf normaler Skala dargestellt, im unteren Diagramm ist die Laufzeit auf logarithmischer Skala dargestellt. Es ist klar ersichtlich, das Selection Sort viel ineffizienter ist, bei größer werdenden Inputs wird die Laufzeit unverhältnismäßig viel größer. Wir werden in der nächsten Vorlesung sehen, dass Selection Sort eine Laufzeit von $O(n^2)$ hat. Im Vergleich dazu ist Mergesort sehr effizient, wird werden sehen, dass Mergesort eine Laufzeit von $O(n \log n)$ hat.
Im Vergleich auf der Log-Skala sehen wir, dass die Linie von Selection Sort ungefähr doppelt so hoch ist wie die von MergeSort. Das deckt sich mit der Theorie, den der Exponent von SelectionSort ist mit $O(n^2)$ eine 2, während der Exponent bei MergeSort in $O(n \log n)$ eine 1 ist. Also ist $\log(n^2) = 2 \log n$ ungefähr doppelt so groß wie $\log(n \log n) = \log n + \log \log n$, wobei $\log \log n$ so klein ist, dass es kaum ins Gewicht fällt.

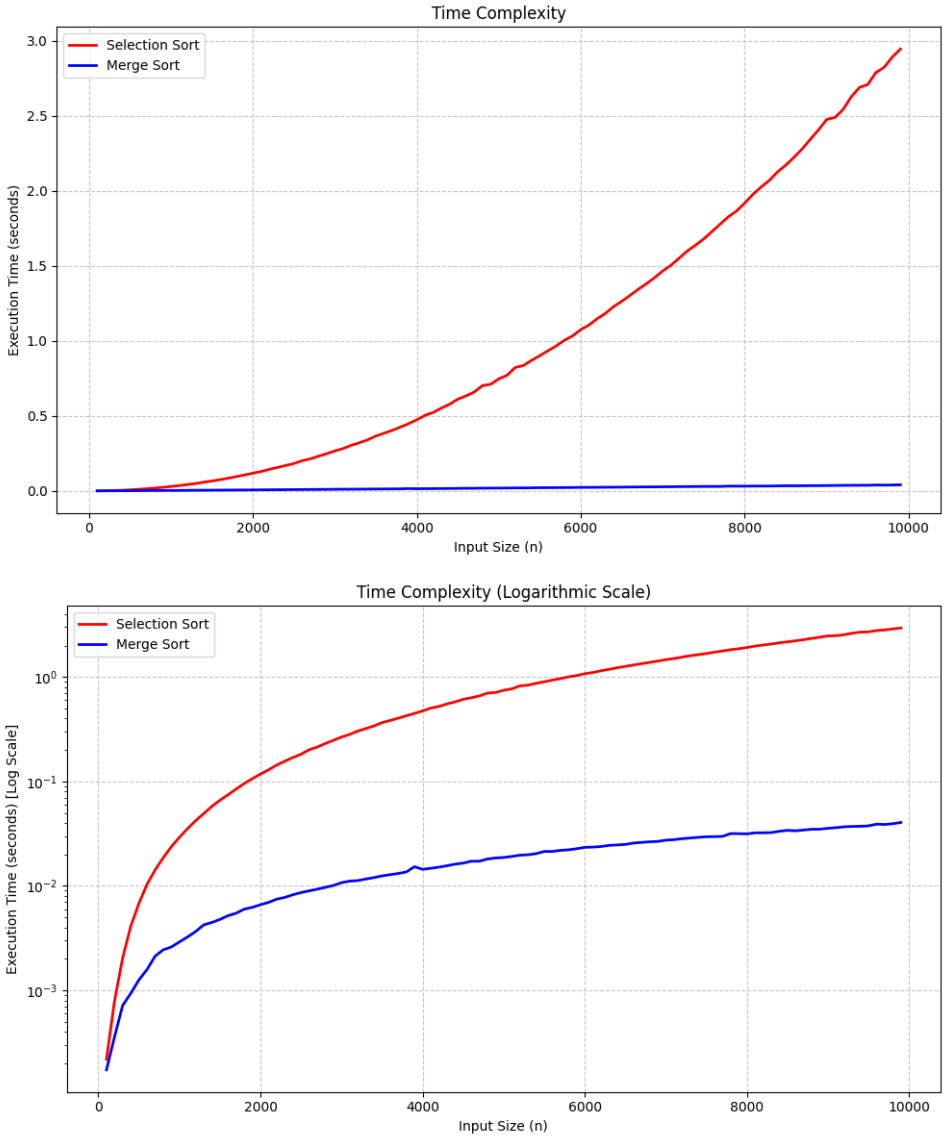

Abbildung 1: Laufzeiten unserer Musterlösung.

Aufgabe 3: Rätsel: Die korrupten Polizisten *(5 Punkte)*

Lösen Sie folgendes Rätsel und schreiben Sie die Lösung (wie in Zukunft alle Theorieaufgaben) in einer PDF-Datei auf. Wir empfehlen LaTeX, Sie können aber auch auf Alternativen ausweichen solange die Lösung klar und lesbar ist.

Stellen Sie sich vor, Sie ermitteln in einem Revier mit 64 Polizisten. Einige von ihnen sind korrupt, aber glücklicherweise wissen wir, dass die absolute Mehrheit (mindestens 33) ehrlich ist. Innerhalb des Reviers weiß jeder genau, wer korrupt ist und wer nicht.

Ihre Aufgabe: Finden Sie heraus, wer auf welcher Seite steht, sprich, welche der 64 Polizisten korrupt und welche ehrlich sind. Sie dürfen den Polizisten dafür nur eine einzige *Art* von Frage stellen: *Polizist A, ist Ihr Kollege B korrupt?* Sie dürfen jeden Fragen und Sie dürfen so viele Fragen stellen, wie Sie möchten, aber jede Frage muss sich auf genau zwei Polizisten (wie in der Fragestellung) beziehen.

Dabei müssen Sie Folgendes beachten: Ehrliche Polizisten antworten immer mit der Wahrheit. Korrupte Polizisten hingegen antworten völlig willkürlich – sie können Sie gezielt anlügen, aber auch die Wahrheit sagen um sich zu schützen.

Beschreiben Sie eine Strategie, um mit Sicherheit alle korrupten Polizisten zu identifizieren. Wie viele Fragen müssen Sie mindestens stellen, um mit Sicherheit alle korrupten Polizisten zu identifizieren?

Musterlösung

Es gibt verschiedene Strategien, aber wahrscheinlich alle effizienten starten damit einen einzigen ehrlichen Polizisten zu identifizieren. Dieser kann dann quasi als Orakel genutzt werden um die restlichen Polizisten zu bestimmen.

Bestimmung eines ehrlichen Polizisten Unsere Strategie ist eine Kombination aus zwei interessanten Beobachtungen:

Beobachtung 1: Wenn Polizist A behauptet, dass Polizist B korrupt ist, dann ist entweder A korrupt oder B korrupt. Jetzt können wir einfach so tun als ob es Polizisten A und B niemals gegeben hat, den bei den übrigbleibenden Polizisten ist ja immer noch eine Mehrheit ehrlich.

Aus Beobachtung 1 wissen wir, dass jedes Mal wenn wir als Antwort 'korrupt' bekommen wir Fortschritt machen (Wir tun einfach so als hätten wir mit 2 Polizisten weniger gestartet). Unklar ist was passiert, wenn alle Antworten 'ehrlich' sind. Die zweite Idee zeigt wie wir hier Fortschritt machen.

Beobachtung 2: Angenommen wir haben eine geordnete Liste an Polizisten $p_1, p_2, p_3, \ldots, p_L$, so jeder Polizist von seinem Nachfolger behauptet dieser wäre ehrlich. Also p_1 behauptet p_2 wäre ehrlich, p_2 wiederum behauptet p_3 wäre ehrlich und so weiter. Dann sind die korrupten Polizisten ein Präfix der Liste, also alle korrupten Polizisten sind am Anfang der Liste.

Der Grund dafür ist, dass wenn Polizist Nummer i ehrlich ist, dann ist auch p_{i+1} ehrlich und so weiter. Angenommen wir haben noch n Polizisten übrig, dann gilt wegen Beobachtung 2, dass sobald unsere Liste länger ist als $\frac{n}{2}$, dann muss die letzte Person der Liste ein ehrlicher Polizist sein.

Das einzige was jetzt noch zu tun ist, ist die obigen Erkentnisse in eine funktionierende Strategie zusammen zu bauen:

- Falls unsere Liste leer ist, fügen wir einen der übrigen Polizisten als erste Person in die Liste.
- Wir befragen den letzten Polizisten p_i der Liste $p_1, p_2, \ldots, p_i$ nach einem der übrigen Polizisten p':
  - Wenn p_i mit 'ehrlich' antwortet, füge p' als p_{i+1} der Liste hinzu.

  - Behauptet p_i hingegen, p' sei korrupt, so entfernen wir beide Polizisten (da mindestens einer korrupt ist). Wir fahren mit der Liste $p_1, p_2, \ldots, p_{i-1}$ fort.

- Auf diese Weise konstruieren wir eine Kette von Polizisten $p_1, p_2, \ldots, p_L$, sodass jeder Polizist seinen Nachfolger als ehrlich bezeichnet.
- Sei k die Anzahl der entfernten Polizisten. Sobald die Länge der Liste $> (64 - k)/2$ ist, wissen wir, dass der letzte Polizist p_L der Kette ehrlich ist.

Nun können wir diesen Polizisten verwenden, um zu erfahren, welche der anderen Polizisten korrupt sind.

Anzahl der Fragen Wir analysieren die Anzahl der Fragen zur Bestimmung eines ehrlichen Polizisten.
Jede Frage hat genau einen der folgenden Effekte:

- Antwort "ehrlich": Die Liste wächst um 1.
- Antwort "korrupt": Zwei Polizisten werden entfernt und die Liste schrumpft um 1 (da das letzte Element entfernt wird).

Seien:

- t die Anzahl der "ehrlich"-Antworten
- c die Anzahl der "korrupt"-Antworten

Dann gilt:

$$L = 1 + t - c$$

(Die Liste startet mit einem Element.)
Außerdem werden insgesamt $2c$ Polizisten entfernt, d.h. es bleiben noch:

$$64 - 2c$$

Polizisten übrig.
Wir stoppen, sobald:

$$L > \frac{64 - 2c}{2}.$$

Einsetzen von L ergibt:

$$1 + t - c > 32 - c.$$

Kürzen von $-c$ auf beiden Seiten:

$$1 + t > 32 \quad \Rightarrow \quad t \geq 32.$$

Das ist der entscheidende Punkt: Die Anzahl der benötigten "ehrlich"-Antworten ist unabhängig von c.

Die Gesamtzahl der Fragen ist:

$$t + c.$$

Im Worst Case maximieren wir diese Zahl unter der Nebenbedingung $t \geq 32$.
Da jede "korrupt"-Antwort zwei Polizisten entfernt, gilt:

$$2c \leq 64 \quad \Rightarrow \quad c \leq 32.$$

Also:

$$t + c \leq 32 + 32 = 64.$$

Damit benötigt man im Worst Case höchstens 64 **Fragen, um einen ehrlichen Polizisten zu finden.**

Anschließend befragt man den ehrlichen Polizisten über alle übrigen Polizisten. Dies sind höchstens 63 weitere Fragen.

Insgesamt ergibt sich als maximale Anzahl an Fragen

$$64 + 63 = 127$$

Hinweis Die Analyse ist nicht optimal. Für die Interessierten, eine optimale Strategie bracht nur $n + c$ fragen, wobei n die Anzahl an Polizisten insgesamt ist und c die Anzahl korrupter Polizisten (wir benötigen immer $c < n/2$, sonst ist das Problem nicht lösbar). Es steht Ihnen frei die Diskussion auf Zulip weiterzuführen.

Aufgabe 4: Abgabe *(5 Punkte)*

Committen Sie Ihren Code (inkl. Tests, `Makefile` und Schaubilder) von Aufgabe 2 sowie die PDF mit der Lösung zu Aufgabe 3 in das SVN (Daphne), in einen eigenen Unterordner `uebungsblatt-01`. Gehen Sie dabei so vor, wie in der Vorlesung vorgeführt. Stellen Sie sicher, dass alles in Daphne (inkl. Style Check und Unit Tests) fehlerfrei durchläuft. Um zu sehen dass alles korrekt ist, sollte ein grünes Häkchen sichtbar sein.
Committen Sie in diesem Unterordner außerdem eine Markdown/Textdatei `erfahrungen.md` oder `erfahrungen.txt`. Beschreiben Sie dort in ein paar Sätzen Ihre Erfahrungen mit diesem Übungsblatt und den Vorlesungen dazu. Insbesondere: Wie lange haben Sie ungefähr gebraucht? An welchen Stellen gab es Probleme und wie viel Zeit hat Sie das gekostet?