Abteilung für
Angewandte Mathematik

SS 2024
Ludwig Striet

# Einführung in die Programmierung für Studierende der Naturwissenschaften

Blatt 8 – 17.06.2024

Abgabe: *Bis 23.06.2024, 23:59 Uhr per E-Mail an Ihre/n Tutor/in*

---

**Aufgabe 1** (5+5 Punkte)**.** Gegeben sei eine sortierte Liste von $N$ Zahlen, d.h. eine Liste von Zahlen

$$e_0, \dots, e_{N-1}$$

s.d. $e_n < e_{n+1}$. Ein Verfahren, um mit logarithmischem Aufwand zu testen, ob ein Element $e$ in der Liste enthalten ist, besteht darin, die Liste wiederholt zu halbieren und nur in der entsprechenden Teilliste zu suchen.

(a) Schreiben Sie das Verfahren als pseudo-code auf und zeigen Sie, dass der Aufwand im worst-case logarithmisch ist, also $\mathcal{A}(N) = C \cdot \log_2(N)$. Dabei dürfen Sie annehmen, dass $N$ eine Zweierpotenz ist, d.h. $N = 2^n$ für ein $n \in \mathbb{N}$. Konstanten, die nicht von $N$ abhängen, können Sie dabei ignorieren.

(b) Implementieren Sie das Verfahren für sortierte Listen beliebiger Länge $N$, also so, dass $N$ nicht notwendigerweise eine Zweierpotenz sein muss.

**Aufgabe 2** ($3 + 2 + 5$ Punkte)**.** Auf der Vorlesungswebseite finden Sie die Datei `merge_sort_templ.cc`, in der der mergesort Algorithmus aus der Vorlesung so implementiert ist, dass damit ein `std::vector<T>` von Objekten beliebigen Typs `T` sortiert werden kann, für die ein Vergleichsoperator `<` implementiert ist. Im angegebenen Beispiel wird damit ein `std::vector<std::string>` lexikographisch sortiert, da der `<`-Operator für `std::string` standardmäßig so implementiert ist.

i) Legen Sie eine Kopie der Datei `merge_sort_templ.cc` an und ändern Sie die Implementierung so, dass die `std::string` nach der *Länge* der Wörter sortiert werden können. Die Länge eines `std::string s` können Sie mit der Methode `length()` abrufen.

In den nächsten Aufgaben ändern wir die Sortier-Funktion so ab, dass damit Objekte beliebigen Typs nach einem beliebigen Vergleichsoperator sortiert werden können. Legen Sie dafür eine neue Kopie der Datei `merge_sort_templ.cc` an.

ii) Implementieren Sie eine Funktion

```
bool lexik_less(std::string s1, std::string s2) {
  //
}
```

die zurückgibt, ob `s1` lexikographisch kleiner ist als `s2` und eine Funktion

```
bool len_less(std::string s1, std::string s2) {
  //
}
```

die zurückgibt ob die Länge von `s1` kürzer ist als die Länge von `s2`. In der `lexik_less` Funktion können Sie den Vergleichsoperator `<` von `std::string` verwenden.

iii) Ändern Sie die `merge_sort(...)` und die `merge(...)` Funktionen so ab, dass diese ein weiteres Argument `bool (*less)(T, T)` haben. Im sortierten `std::vector<T>` soll dann `t1` vor `t2` auftauchen, wenn `less(t1, t2)` gilt.
Testen Sie Ihre Implementierung in einer `main()`-Funktion, in der Sie die `std::string` aus dem Beisiel lexikographisch und nach Länge sortieren und die nicht-negativen Zahlen im Bereich `1..20` nach der Anzahl ihrer von `1` verschiedenen Teiler. Eine Funktion um die Anzahl der Teiler einer nicht negativen Zahl zu berechnen finden Sie auf der Vorlesungswebseite.