Technische Universität Dortmund
Fakultät Statistik
Lehrstuhl Computergestützte Statistik

Dr. Uwe Ligges
M.Sc. Marieke Stolte
Luca Sauer
Rudi Zulauf

Übung zur Vorlesung

Computergestützte Statistik

Wintersemester 2022/2023

Übungsblatt Nr. 2

Abgabe ist Dienstag der 25.10.2022 bis 08:00 Uhr im Moodle

Aufgabe 1 (4 Punkte)

Implementieren Sie Quicksort in 2 Varianten. Versuchen Sie dabei möglichst effizient zu programmieren:

a) (1 Punkt) Eine einfache, rekursiven Variante (d.h. verzichte Sie auf das aufwendige von beiden Seiten aus zu kleine / große Elemente finden und dann tauschen, sondern machen Sie einfach 2 rekursive Aufrufe: einen mit allen Elementen kleiner, einen mit allen Elementen größer als das Pivot)

b) (2 Punkte) Eine iterative Variante in Anlehnung an Algorithm 2 auf Folie 69.

c) (1 Punkt) Vergleichen Sie die Laufzeiten, die die beiden Varianten zum Sortieren gleicher Vektoren benötigen (hier wirklich Zeiten, keine Vergleiche oder Vertauschungen). Wie erklären Sie sich das Ergebnis?

Aufgabe 2 (4 Punkte)

Im letzten Schritt von Insertion Sort muss jeweils die richtige Position von a_{i+1} in a bestimmt werden. Dazu wird der Algorithmus der binären Suche verwendet:

Sei X dazu ein bereits sortierter Vektor, in den ein weiteres Element x eingefügt werden soll. Bestimme zunächst des mittlere Element von X. Vergleiche x mit diesem mittleren Element. Falls es kleiner ist, fahre mit der Suche (rekursiv) in der ersten Hälfte von X fort, ansonsten in der zweiten Hälfte. Sobald die Länge der rekursiv verkürzten Liste 1 beträgt, kann mit einem Vergleich die richtige Position gefunden werden: Vor oder hinter dem verbliebenen Element.

a) (1 Punkt) Implementieren Sie den Algorithmus der binären Suche. Denken Sie wie üblich an Dokumentation und Tests.

b) (1 Punkt) Ergänzen Sie Ihre Funktion so, dass zusätzlich die Anzahl der benötigten Vergleiche Ausgabe der Funktion soll eine benannte Liste sein, die als Elemente *result* (Position, hinter der x in X eingefügt werden soll) und *n.comps* (Anzahl benötigter Vergleiche). Stellen Sie die Anzahl Vergleiche im Verhältnis zur Länge n des Vektors dar.

c) (1 Punkt) Bestimmen Sie formal die Anzahl Vergleiche, die der Algorithmus zum Einfügen eines Elementes benötigt. Geben Sie ein passendes Landau-Symbol an.

Hinweis: Hier wird eine Herleitung benötigt. Die bloße Angabe der Anzahl Vergleiche gibt keine Punkte.

d) (1 Punkt) Zeigen Sie, dass man beim Insertion Sort Algorithmus mit binärer Suche (wie in der Vorlesung ab Folie 60 vorgestellt) im Worst Case nur $O(n \log n)$ Vergleiche braucht. (Allerdings braucht man immer noch quadratisch viele Verschiebungen, dies müssen Sie jedoch nicht beweisen.)

Aufgabe 3 **(4 Punkte)**

Betrachten Sie zuerst eine einfachere Variante des Insertion Sort Algorithmus mit linearer statt binärer Suche:

Algorithm 1 INSERTION SORT Pseudocode (linear search)

Require: $\vec{a} \in \mathbb{R}^n$ (vector of inputs)
Ensure: sorted $\vec{a}$

```
for j = 2 to n do
    tmp ← a_j
    i ← j − 1
    while i > 0 and a_i > tmp do
        a_{i+1} ← a_i
        i ← i − 1
    end while
    a_{i+1} ← tmp
end for
return a
```

a) (1 Punkt) Zeigen Sie: Im Worst Case braucht der Algorithmus $\Theta(n^2)$ Vergleiche.

b) (0.5 Punkte) Implementieren Sie eine Funktion `insertionSort` in `R`, die den Pseudocode in Algorithm 1 umsetzt.

c) (0.5 Punkte) Erweitern Sie Ihre Insertion-Sort-Implementierung so, dass sowohl die Anzahl der Vergleiche als auch die Anzahl der Vertauschungen gezählt werden. Der Rückgabewert soll nun eine benannte Liste mit den 3 Elementen *sorted.vec*, *nof.comps* und *nof.switches* sein. Denken Sie an ordentliche Dokumentation, Tests und Parameter-Checks.

Hinweis: Sie müssen natürlich nur eine (die finale) Implementierung des Algorithmus abgeben – und nicht 2 verschiedenen Implementierungen für die Aufgabenteile b) und c).

d) (1 Punkt) Schreiben Sie eine zweite Funktion `insertionSort2`, die den Insertion Sort Algorithmus mit binärer statt wie in Teil b) und c) mit linearer Suche durchführt. Zählen Sie auch hier die Vergleiche und Vertauschungen und geben diese in einer benannten Liste mit den 3 Elementen *sorted.vec*, *nof.comps* und *nof.switches* zurück. Denken Sie an Dokumentation und Parameter-Checks und testen Sie die neue Funktion mit ihrer Testfunktion aus c).

Hinweis: Für die binäre Suche dürfen Sie gerne Ihre Funktion aus Aufgabe 2 verwenden.

e) (1 Punkt) Überprüfen Sie empirisch die *Laufzeit* Ihrer Funktionen. Zählen Sie dafür die Anzahl der benötigten Vergleiche und Vertauschungen. Erzeugen Sie dazu zufällige Vektoren der Länge n, $n \in \{100, 200, \ldots, 1000\}$ und visualisieren Sie die Anzahl Vergleiche und die Anzahl Vertauschungen im *average case*. Wählen Sie dabei jeweils 10 Wiederholungen für jedes n, um eine stabilere Schätzung der Laufzeit zu erhalten. Decken sich Ihre Ergebnisse mit den theoretischen Laufzeiten?