TH Brandenburg
Online Studiengang Medieninformatik
Fachbereich Informatik und Medien
Algorithmen und Datenstrukturen
Prof. Dr. rer. nat. Ulrich Baum

Einsendeaufgabe 2

Sommersemester 2021
Abgabetermin XX.05.2021

Maximilian Schulke
Matrikel-Nr. 20215853

1 Finde die falsche Münze

a) Man kann das Problem lösen, indem man die erste Münze nacheinander mit allen anderen vergleicht. Geben Sie den Worst-Case und Best-Case Aufwand dafür an.

Falls der Best-Case entritt (die falsche Münze liegt an einer der ersten beiden Positionen) haben wir nur einen einzigen Vergleich dementsprechen $O(1)$. Der Worst- Case tritt dementsprechen im umgekehrten Fall ein, also wenn die falsche Münze ganz am Ende der Folge liegt - dann benötigen wir $n-1$ Vergleiche und liegen somit in $O(n)$.

b) Entwerfen und beschreiben Sie umgangssprachlich einen rekursiven Divide-And-Conquer Algorithmus, der das Problem für große n deutlich schneller löst.

Am besten geht man hier mit einer modifizierten Version der binären Suche vor. Da n immer glatt durch 2 teilbar ist, haben wir immer eine identische Menge an Münzen auf der linken und auf der rechten Seite, wenn wir die Ausgangsmenge in der exakten Mitte teilen. Nun kann man diese beiden Mengen miteinander vergleichen. Falls die Anzahl der Münzen in jeder Hälfte größer als 1 ist, muss dieser Vorgang wiederholt werden, bis nurnoch 2 Münzen miteinander verglichen werden, dann ist die leichtere der beiden die falsche Münze.

c) Analysieren Sie den Worst-Case und Best-Case Aufwand dafür. Zählen Sie die Anzahl der Wiegevorgänge dabei genau (nicht asymptotisch)

Nun unterscheidet sich der o.g. Algorithmus von der normalen binären Suche dadurch, dass wir keinen Vergleich mit einem einzelnen Element haben, mit dem wir vorzeitig abbrechen könnten. Wir müssen also jedes mal, selbst wenn die falsche Münze in der Mitte liegt, die Hälften immer ganz abarbeiten was bei 2k Elementen zu genau $log_2 n$ oder anders ausgedrückt k Vergleichen führt.

2 Modifizierter Mergesort

a) Beschreiben Sie die Arbeitsweise von $merge4$ umgangssprachlich. Bestimmen Sie die worst-case-Anzahl von Vergleichen von Vergleichen für $merge4$ auf 4 gleich großen sortierten Folgen von je m Elementen. Zählen Sie die Vergleiche genau (keine O-Notation).

Beim normalen *merge* werden zwei in sich sortierten Listen zu einer großen sortierten Liste zusammen gefügt, in dem immer das kleinere der beiden Elemente der beiden Teillisten zu erst in die neue Liste geschrieben wird, dieser Vorgang kostet einen Vergleich.

Nun müssen bei $merge4$ nicht 2 sondern 4 Listen verglichen werden, es muss also bei jedem Vorgang das minimum von 4 Elementen identifiziert werden, das kostet 3 Vergleiche. Sobald die Anzahl aller Elemente der Teillisten 4 ist brauchen wir nur 3! Vergleiche um die restlichen Elemente einzusortieren.

Also anders ausgedrückt braucht $merge4$ im Worst-Case $((m-1)*4*3)+3!$ Vergleiche. Der Worst-Case tritt ein, wenn die Elemente gleichmäßig auf die Teillisten verteilt ist, also bei $m=2$

die erste Liste $[1, 5]$, die zweite Liste $[2, 6]$, die dritte Liste $[3, 7]$ und die vierte Liste $[4, 8]$ ist. Diese Aufteilung verhindert das vorzeitige Abarbeiten einer einzelenen Liste wodurch Vergleiche eingespart werden könnten.

b) T(n) sei die worst-case Anzahl von Vergleichen des Sortierverfahrens auf einer Inputfolge der Länge n. Nehmen Sie an, dass n ein Vielfaches von 4 ist. Stellen Sie die Rekursionsgleichung für T(n) auf.

$$T(4) = 0$$
$$T(n) = 4T(\frac{n}{4}) + merge4(\frac{n}{4})$$

$$merge4(m) = ((m-1) * 4 * 3) + 3!$$

c) Lösen Sie die Rekursionsgleichung asymptotisch mit dem Master-Theorem

d) Wird diese Variante in der Praxis schneller sein als der Standard-Mergesort?

Nein. Der Aufwand der einzelenen Merge-Vorgänge ist deutlich höher, viel mehr Elemente gleichzeitig betrachtet werden müssen.

3 Umdrehen von Listen

```
def reverse(head):
    first = copy(head)

    while first.next is not Null:
        old_head = copy(head)
        head = first.next
        first.next = head.next
        head.next = old_head

    return head
```