Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Thomas Seidl
Janina Sontheim, Florian Richter

München, 28.05.2019

Algorithmen und Datenstrukturen

SS 2019

Übungsblatt 5: Sortieren

Aufgabe 5-1 *Quicksort*

Gegeben sei folgendes Array:

37	16	9	42	51	17	71	56	89	39	6	3

(a) Wenn sie ein Element des unsortierten Arrays finden wollen, wie lange müssen Sie maximal suchen? Begründen Sie ihre Lösung.

(b) Sortieren Sie das Array mittels Quicksort, nach dem Algorithmus der Vorlesung. Verwenden Sie dabei das letzte Element als Pivot-Element.

Machen Sie die einzelnen Schritte bestmöglich nachvollziehbar. Jeweils eine Rekursionsschritt für alle Teilfolgen des Arrays dürfen in einer Zeile gemacht werden. Kennzeichnen Sie das jeweils aktuelle Pivot-Element und die bereits an die richtige Position getauschten Pivot-Elemente.

(c) Nun da das Array sortiert ist, wie lange brauchen sie im schlechtesten Fall um ein Element zu finden? Bilden Sie aus dem sortierten Array einen binären Baum mit 39 als Wurzel.

(d) Zahlt es sich laufzeittechnisch aus zu sortieren, welche Voraussetzungen braucht es, damit sich dies auszahlt?

(e) Ist dieser Sortieralgorithmus stabil? Begründen Sie.

(f) Der Algorithmus sei nun an seine obere Laufzeit gestoßen ($O(n^2)$). Unter welchen Umständen kommt es zu diesem Worst Case?

(g) Was ist die optimale Laufzeit des Sortieralogrithmus und wie sieht das Array aus, wenn diese auftritt?

(h) Was ist im Allgemeinen eine gute Wahl für das Pivot-Elements?

Lösungsvorschlag:

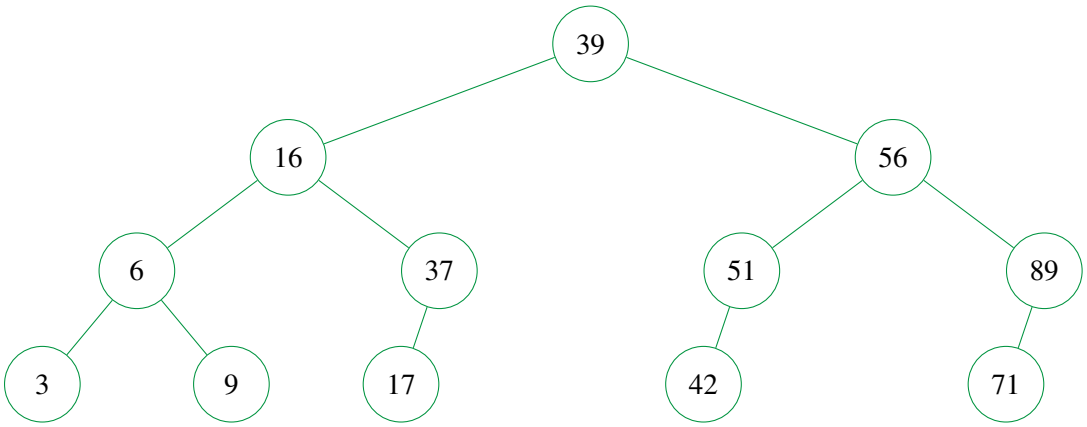
a) Suchzeiten: Bei einem unsortierten Array muss im Worst Case jedes Element betrachtet werden daher suchzeit der Größe n.

b)

37	16	9	42	51	17	71	56	89	39	6	(3)
[3]	37	16	9	42	51	17	71	56	89	39	(6)
[3]	[6]	37	16	9	42	51	17	71	56	89	(39)
[3]	[6]	37	16	9	(17)	[39]	42	51	71	56	(89)
[3]	[6]	16	(9)	[17]	(37)	[39]	42	51	71	(56)	[89]
[3]	[6]	[9]	(16)	[17]	[37]	[39]	42	(51)	[56]	(71)	[89]
[3]	[6]	[9]	[16]	[17]	[37]	[39]	(42)	[51]	[56]	[71]	[89]
[3]	[6]	[9]	[16]	[17]	[37]	[39]	[42]	[51]	[56]	[71]	[89]

Lösungsvorschlag:

c) Da das Array sortiert ist, kann man zum Beispiel in der Mitte anfangen und abhängig vom Größenverhältnis nach links (kleiner) oder nach rechts (größer) in die jeweilige Mitte gehen. Dadurch ergibt sich eine Laufzeit von log(n).



d) Es zahlt sich aus zu sortieren, sobald ein mehrfacher Zugriff auf die Daten stattfindet. Für einen einmaligen Zugriff zahlt sich das Sortieren meist nicht aus. Das Entscheidungskriterium hierfür ist die Differenz zwischen der Laufzeit des Sortierens und der Ersparnis durch eventuelles Sortieren.

e) Der Sortieralgorithmus ist nicht stabil. Stabilität hängt von der Pivotstrategie ab: Angenommen, das Pivot ist immer mittleres Element. Dann ist [5,5,5] schon ein Gegenbeispiel. Die mittlere 5 ist das Pivot und die anderen beiden 5 werden entweder links oder rechts (abhängig vom Vergleich) einsortiert. Bei Randomized Pivot ist es super instabil, weil irgendein Element in die Mitte gesetzt wird.

f) Der Algorithmus stößt an seine obere Laufzeit, wenn das gewählte Pivot-Element immer dem größten oder kleinsten Schlüssel des Datensegments entspricht.

Zum Beispiel wenn eine bereits sortierte Liste vorliegt und als Pivot-Element immer das erste gewählt wird.

g) Die optimale Laufzeit liegt in O(n log(n)). Dann wird das Datensegment immer in zwei gleich große Teile unterteilt.

$$\mathrm{T}(n) = \begin{cases} 2*\mathrm{T}(\frac{n-1}{2})+\mathrm{n}\text{-}1, & \text{wenn } n > 1, \\ 0, & \text{wenn } n = 1. \end{cases}$$

Lösung mit dem Mastertheorem ergibt dann O(n log(n)).

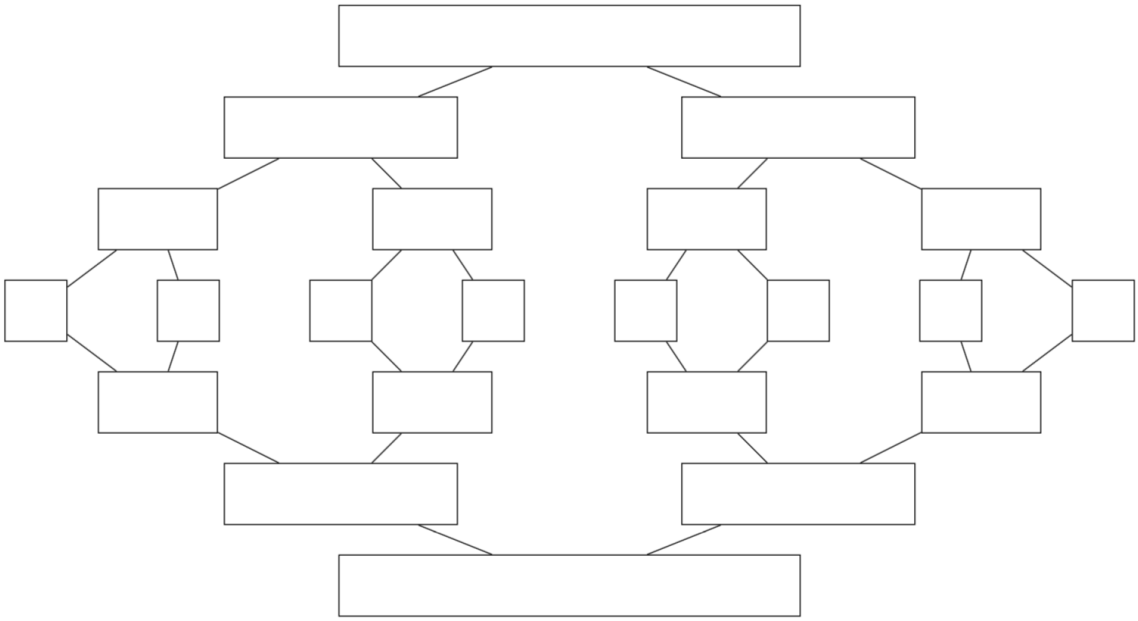
h) Die naiven Ansätze wie das Wählen des ersten, mittleren oder letzten Elements als Pivot-Element sind relativ ineffizient. Bei (teilweise) vorsortierten Listen kann es leicht zum Worst-Case kommen. Eine gute Wahl wäre ein zufälliges Element (sehr geringe Chance, dass der Worst-Case eintritt) oder gar der Median (kein Worst-Case). Es ist relativ aufwendig, den Median eines Datensatztes zu berechnen (dafür muss der Datensatz sortiert sein), deswegen werden häufig drei (am besten zufällige) Elemente zur Bildung dessen verwendet.

Aufgabe 5-2 *MergeSort*

Gegeben sei folgende Liste von Zahlen: 24, 15, 76, 6, 43, 9, 41, 68

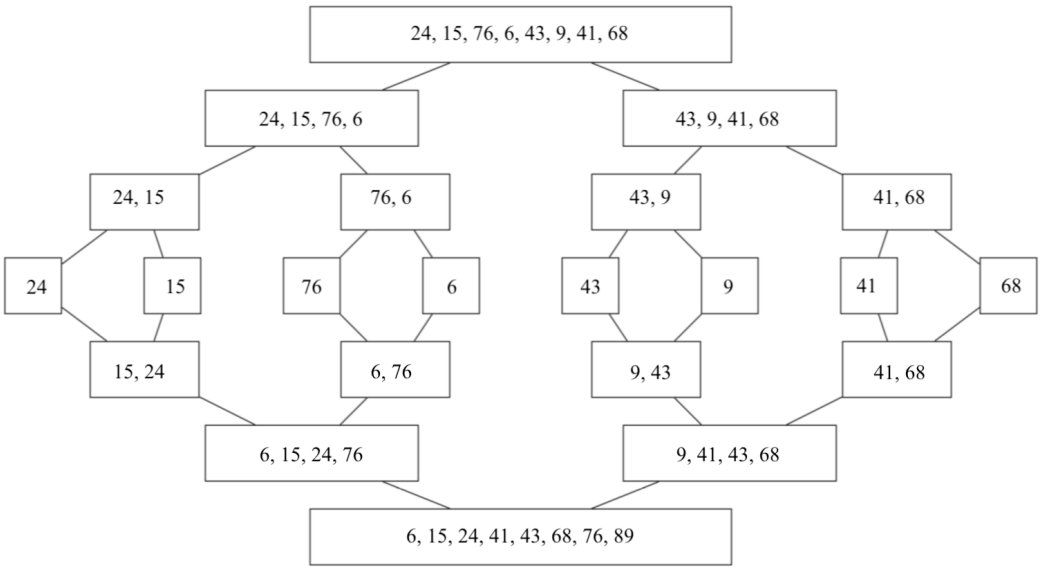
(a) Sortieren Sie die Zahlenfolge mit dem MergeSort-Algorithmus, und geben Sie dabei alle Zwischenschritte an. Verwenden sie dazu die unten gegebene Vorlage.

(b) Wie viele Tauschoperationen wurden beim Sortieren der Liste ausgeführt?



Lösungsvorschlag:

(a) Sortierte Zeichenfolge:



(b) Beim MergeSort werden keine Einträge vertauscht. In jedem ”Zwischenschritt” beim Mergen werden die Einträge in der richtigen Reihenfolge in ein neues Array eingetragen.