DUA: Praesenzaufgaben Blatt 4

Konstantin Krasser

October 24, 2024

Aufgabe 1 (9 points)

In the lecture, you learned about the partition function. This function can not only be used to sort an array A (as in quicksort), but also to find the k-th smallest value in A, i.e., the k-th element in the ascending sorted order of the elements in A.

Example: In the array $A = [6, 7, 2, 9, 3, 1, 0]$, the 4th smallest element is the number 3 (only 0, 1, and 2 are smaller). We assume that every element in A is unique.

1. Answer to question 1:

   What would a naive approach, using comparison-based search algorithms, look like to find the k-th smallest element in A? What is the lower asymptotic bound on the runtime of this approach?

   The naive approach would be to sort the array and then return the k-th element. The time complexity for comparison-based sorting algorithms is:

   (a) Sort A

   (b) Return $A[-1]$

$$O(n \log n) \tag{1}$$

2. Answer to question 2:

   Provide a modified partition function in pseudocode that rearranges the array A such that the pivot element is at position i_p, and all elements to the left of i_p are smaller than the pivot element, and all elements to the right of i_p are greater than the pivot element. The manipulation of A is done in-place, so all changes are made directly in A, and partition does not need to explicitly return the array A, only i_p.

   [1] $p \leftarrow \mathcal{A}[n-1]$
   [2] $i \leftarrow -1$
   [3] **for** $j \leftarrow 0$ **to** $n-2$ **do**
      [1] **if** $\mathcal{A}[j] \leq p$ **then**
         [1] swap$(\mathcal{A}[i+1], \mathcal{A}[j])$
         [2] $i \leftarrow i+1$
   [4] swap$(\mathcal{A}[i+1], \mathcal{A}[n-1])$
   [5] $i \leftarrow i+1$
   [6] **return** $(\mathcal{A}, i)$

3. Answer to question 3:

   Describe in words and in pseudocode how the modified partition function can be used to efficiently find the k-th smallest value in A.

   Was passiert hier? Das Pivotelement p ist das letzte Element von $\mathcal{A}$, siehe Zeile [1]. Wie oben erwähnt, partionieren wir das Array in Elemente, welche kleiner oder gleich p sind, sowie die Elemente, welche größer als p sind. Das Pivotelement p dient erstmal nur zum Vergleichen ([4]), bleibt aber ansonsten außen vor, bis es in Zeile [7] an seine endgültige Position getauscht wird.

   Die endgültige Position von p wird von dem Index i bestimmt. Dabei erfüllt i zu jedem Zeitpunkt die Bedingung, dass alles, was sich links von i befindet (i eingeschlossen), stets kleiner oder gleich p ist (Zeile [4] und [5])! Unter den Elementen, die bereits mit p verglichen wurden (siehe Laufindex j) nimmt i dabei den maximalen Wert ein (Nach Ausführung von Zeile [6], bzw Zeile [8]). Zu beachten ist außerdem, dass durch das rechtzeitige Addieren von $i + 1$ nie ein Arrayzugriff an undefinierter Stelle geschieht([5]), selbst wenn i mit -1 initialisiert wurde([2]).

   Als letztes muss nur noch die Rolle des Laufindex j geklärt werden. Definiert in Zeile [3] startet j beim ersten Element und geht bis zum vorletzten (also exklusiv p). Da pro Schleifendurchgang i um maximal eins inkrementiert werden kann, ist j also stets größergleich i. Dabei zeigt j an, welche Elemente des Arrays bereits mit p verglichen wurden. Findet j mit Zeile [4] ein Element, welches kleiner ist als p, wird dieses in den von i markierten Bereich vertauscht([5]).

4. Answer to question 4:

   What is the runtime of your algorithm in the best case and in the worst case? Provide an example call for both cases. The best/worst case should apply to general k, not a specific k.

   The **best-case** runtime occurs when the pivot always splits the array evenly, giving a time complexity of $O(n)$, since each partition step reduces the problem size by half.

   The **worst-case** runtime occurs when the pivot is always the smallest or largest element, leading to a time complexity of $O(n^2)$ because each partition only reduces the problem size by one element.

   Example best-case: `quickselect([1,2,3,4,5], 0, 4, 2)`.

   Example worst-case: `quickselect([5,4,3,2,1], 0, 4, 2)`.