Algorithmen und Datenstrukturen
Hausaufgabenserie

Ole Bergens, 221200097 Antonin Gräser, 221201792
Blazej Schott, 221200610 Nils Martin, 221202136

June 10, 2022

Aufgabe 1)
Man könnte meinen, dass wenn der gleiche Schlüssel mehrmals auftritt so würde *qsort* einen Vorteil daraus erhalten, dies ist aber nicht der Fall. Der Sortieralgorithmus macht zwar weiterhin seine Arbeit und sortiert eine Liste mit dem vorkommen von gleichen Schlüsseln, zieht aber durch das vorkommen gleicher Schlüssel keinen Vorteil daraus.
Wenn ein Sortieralgorithmus einen Vorteil daraus ziehen kann, so nennt man es *smooth* bzw. dt. *glatt*. So würde dieses Sortierverfahren bei N verschiedenen Schlüssel im Mittel $O(NlogN)$ haben und bei N gleichen Schlüsseln $O(N)$ Schritte zum sortieren brauchen. Somit würde die Komplexität von *qsort* weiterhin bei einer Sortierung von Folgen, die nur aus Kopien des gleichen Schlüssels bestehen, bei $O(n^2)$ liegen. Das Problem lässt sich aber lösen, wenn man bei der Aufteilung eines Bereiches $arr[links]...arr[rechts]$ nach dem Schlüssel des rechtesten Elementes $arr[rechts]$ alle Elemente im aufzuteilenden Bereich, deren $schluessel = pivot.schluessel$ sind in der Mitte zu sammeln. Somit wird statt einer Zerlegung in zwei Folgen $Folge_1$ und $Folge_2$ mit dem Pivotelement mittendrinnen eine Zerlegung in drei Folgen angestrebt $Folge_{links}$, $Folge_{rechts}$ und $Folge_{mitte}$ für die dann gilt (folgend ist $v = arr[rechts].schluessel$ das Pivotelement), dass in der Folge F_{links} alle Elemente mit dem Schlüssel $< v$, in F_{rechts} sind dann alle Elemente $> v$ und in F_{mitte} sind dann alle Elemente für deren Schlüssel gilt, dass $= v$. Da man ja aber nicht weiß, wo die endgültige Position des Pivotelements ist, ist dabei noch ein Problem entstanden: wo speichert man nun die Elemente zwischen, deren Schlüssel mit dem des Pivotelements übereinstimmen. Man könnte die Elemente am Anfang und Ende des aufzuteilenden Bereiches sammeln und diese dann später in die Mitte befördern, oder man kann sie auch nur am Anfang oder am Ende sammeln. Man könnte natürlich auch diese Elemente als einen starren Block behandeln, welcher durch das Array wandern würde bis zur richtigen Position.

Aufgabe 2)
Wie gehen Sie vor, um den jetzt unsortierten Stapel von Lochkarten in linearer Zeit zu sortieren? Es gibt mindestens zwei Optionen. Erläutern Sie jeweils für beide Optionen, wie diese anzuwenden sind und erklären Sie, welche Annahmen Sie dabei machen müssen.
Es gibt die Möglichkeit des Radixsort bzw. genauer gesagt des Distributionsort. Dort machen wir für die Lochkarten die Annahme, dass jede Lochkarte einen Schlüssel von fester Länge zugeordnet bekommen hat. Bei Distributionsort ist der Aufwand dann bei einer festen Schlüssellänge N gleich $T(n) = N \cdot n$ und somit $O(n)$. Damals wurde beim IBM-Kartensortierer auch der Radixsort angewendet um sehr große Mengen an Lochkarten zu sortieren. Dort wurden Die Karten in eine Art Trichter gegeben und eingezogen und dann dort anhand der in einer Spalte auf den Karten gestanzten Daten in einen von 13 Ausgabekörben sortiert.
Dann gibt es noch die Möglichkeit des Countingsort. Dafür ist die Annahme, dass die Schlüssel aus natürlichen Zahlen aus einem beschränkten Intervall bestehen. Das ist hier ja der Fall, da wir das Intervall in welchem die Schlüssel vorkommen durch die Anzahl der Lochkarten beschränken können. Nun zählt der Algorithmus dann wie oft jeder dieser Werte in der Eingabe vorkommt. Diese Anzahl an Vorkommen wird dann in einen zusätzlichen Array gespeichert und durch dessen Hilfe anschließend für jeden Schlüsselwert die Zielposition in der Ausgabe brechechnet wird. Somit lässt sich diese Menge an Lochkarten dann auch mit Countingsort in einer linearen Zeit sortieren.