Dustin Scharf

Sorting Algorithms *28.07.2021*

Best/average/worst cases einiger Sortierverfahren mit kurzer Erklärung.

Insertion Sort

Best case O(n)
Das Array ist bereits sortiert und beim ersten und einzigen Durchlaufen muss nichts unternommen werden, weil jedes Element schon an der korrekten Position steht

Worst case O(n^2)
Das Array ist umgekehrt sortiert und jeder Wert muss mit allen anderen Werten getauscht werden

Selection Sort

Best case O(n^2)
Worst case O(n^2)
Das Array wird für jede Stelle des final sortieren Arrays stets vollständig durchlaufen um den kleinsten Wert für die nächste Stelle im final sortierten Array zu finden, das ist immer gleich und unabhängig vom Input-Array

Bubble Sort

Best case O(n)
Das Array ist bereits sortiert und keine Elemente müssen beim ersten und einzigen Durchlaufen getauscht werden, weil alle Elemente schon in der richtigen Position sind

Worst case O(n^2)
Das Array ist umgekehrt sortiert und jeder Wert muss mit allen anderen Werten getauscht werden

Bucket Sort

Best case O(n)
In alle Buckets kommt nur ein Element

Worst case O(*Algorithmus für die Sortierung innerhalb der Buckets*)
Alle Elemente kommen in einen einzigen Bucket und der Sortieralgorithmus für den Bucket muss eigentlich das gesamte Array sortieren

Counting Sort

Best case O(n + K) **K beschreibt die Anzahl der möglichen Keys im Input-Array*
Worst case O(n + K)
Erst müssen alle Elemente im Array gezählt werden und dann muss noch das gesamte Counter Array mit den gezählten Werten durchgegangen werden um die Werte wieder richtig sortiert in das finale Array einzufügen, das ist immer gleich und unabhängig vom Input-Array

Radix Sort

Best case O(n)

Alle Zahlen sind einstellig und müssen somit nur einmal in der Buckets für die Ziffern eingefügt werden

**L beschreibt die Anzahl der Ziffern der Nummer mit den meisten Ziffern im Input-Array*

Worst case O(nL)

So oft wie die längste Zahl Ziffern hat muss das Array durchlaufen werden, also wäre es sinnlos, den Algorithmus zu verwenden, wenn es z. B. nur eine Zahl mit sehr vielen Ziffern, aber sonst nur einstellige Zahlen gibt

(Ein absoluter worst case wäre, wenn das Input-Array nicht aus Zahlen besteht, denn dann funktioniert Radix Sort überhaupt nicht)

Shell Sort

Best case O(n log(n))

Das Array ist bereits sortiert, und muss nur für jeden Split einmal durchlaufen werden, ohne dass dabei etwas passiert

Worst case O(n log²(n))

Das Array ist umgekehrt sortiert und jeder Wert muss mit allen anderen Werten mit den Abständen der Folge 1, 2, 3, 4, 6, …, 2^p3^q *(Pratt)* getauscht werden

[Mit einer Folge wie 1, 2, 4, 8, 16, …, 2^k *(Donald Shell)* wäre der worst case $O(n^2)$, diese Folge wird heutzutage aber nicht mehr genutzt, die Folge 1, 3, 7, 15, 31, …, $2^k - 1$ *(Hibbard)* bringt $O(n^{1.5})$ und ist somit auch gut, wenn nicht sogar besser, weil sie sich leichter berechnen lässt]

Merge Sort

Best case O(n log(n))

Worst case O(n log(n))

Das Array wird immer gesplittet / **halbiert bis es nur noch einzelne Werte gibt**, die dann wieder zusammengefügt werden, das passiert immer und unabhängig vom Input-Arrays

Quick Sort

Best case O(n log(n))

Es müssen alle Partitionen nur einmal durchlaufen werden, weil das Array bereits sortiert ist und keine Vertauschungen vorgenommen werden müssen

Worst case $O(n^2)$

Das Array ist z. B. bereits aufsteigend sortiert und man wählt immer das letzte / größte Element als Pivot Element

Somit muss nämlich immer das gesamte Array für alle Pivot Elemente durchlaufen werden

Heap Sort

Best case O(n log(n))

Worst case O(n log(n))

Weil beim Löschen aus einem Heap immer eins der kleinsten Elemente in die Wurzel gebracht wird, was dann wieder den Heap abwärts laufen muss, ist die Zeitkomplexität unabhängig vom Input-Array

Zusammenfassung der Laufzeiten der Sortieralgorithmen

*K beschreibt die Anzahl der möglichen Keys im Input-Array
*L beschreibt die Anzahl der Ziffern der Nummer mit den meisten Ziffern im Input-Array

Algorithmus	**Laufzeitkomplexität**		
	best case	**average case**	**worst case**
Insertion Sort	O(n)	O(n²)	O(n²)
Selection Sort	O(n²)	O(n²)	O(n²)
Bubble Sort	O(n)	O(n²)	O(n²)
Bucket Sort	O(n)	O(n)	O(*Algorithmus für die Sortierung innerhalb der Buckets*)
Counting Sort	O(n + K)	O(n + K)	O(n + K)
Radix Sort	O(n)	O(nL)	O(nL)
Shell Sort	O(n log(n))	O(n log²(n))	O(n log²(n))
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))
Quick Sort	O(n log(n))	O(n log(n))	O(n²)
Heap Sort	O(n log(n))	O(n log(n))	O(n log(n))