Algorithmen & Datenstrukturen

Quicksort

Florian Heuer - 2243615

Inhaltverzeichnis

1. Pivotsuchverfahren (Quicksort)

Die Grundidee von Quicksort ist das Prinzip „Teile und herrsche“, welches bedeutet das der Algorithmus die zu sortierende Datenstruktur in 2 Hälften teilt, welche kleiner und größer als das [1]Pivotelement sind. Durch Rekursion wird dies solange wiederholt bis die Hälften <=1 sind.

Im folgenden werden 3 Varianten des Quicksortalgorithmus vorgestellt, welche sich in der Bestimmung ihres Pivotelements unterscheiden. Dadurch können die Laufzeiten der unterschiedlichen Varianten beeinflusst werden.

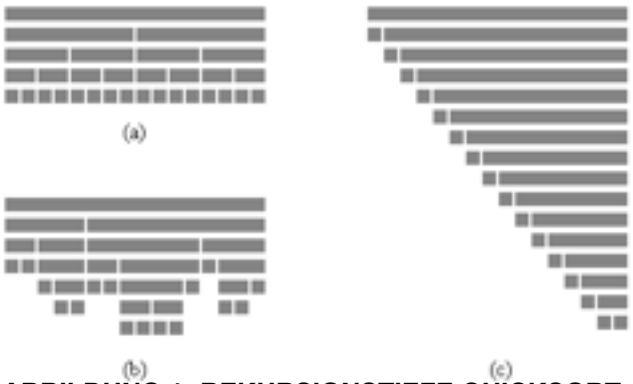

ABBILDUNG 1: REKURSIONSTIEFE QUICKSORT

1.1 Letztes oder Erstes Element des Arrays

Bei dieser Variante wird das erste oder letzte Element im Array als Vergleichselement bzw. Pivotelement genutzt. Dieses Vorgehen ist simpel, kann aber sehr ineffizient sein (siehe Worstcase).

1.1.1 Best- & Worstcase-Betrachtung

Im **Best Case** wird diese Variante eine Komplexität von $O(n \cdot \log n)$ aufweisen. D.h. der Algorithmus arbeitet am effizientesten, um so gleichmäßiger die Verteilung auf die stattfindet (siehe Abb. 1 (a)). Rekursiontiefe ist log(n). Für den Best Case sollte das Pivotelement nahe des Medians der zu sortierenden Menge liegen.

Im **Worst Case** wird eine Hälfte nur um ein Element (n-1) reduziert, die andere Hälfte enthält immer nur ein Element. Das führt dazu, dass die Komplexität auf $O(n^2)$ steigt (siehe Abb. 1 (c)). Die Rekursionstiefe erreicht hier sein Maximum von n-1. Ein möglicher Worst Case wäre eine bereits sortierte Menge.

1.1.2 Testreihen

Listenlänge	Best Case		Avergage Case		Worst Case	
	Vergleiche	Tausch	Vergleiche	Tausch	Vergleiche	Tausch
10	35	7	40	12	63	9
100	718	194	1014	182	5148	99
1000	10332	2766	15319	2609	501498	999

[1] Element das pro Aufruf bestimmt & zum Vergleich genutzt wird, Pivot = frz. für Drehpunkt

1.1.3 Darstellung

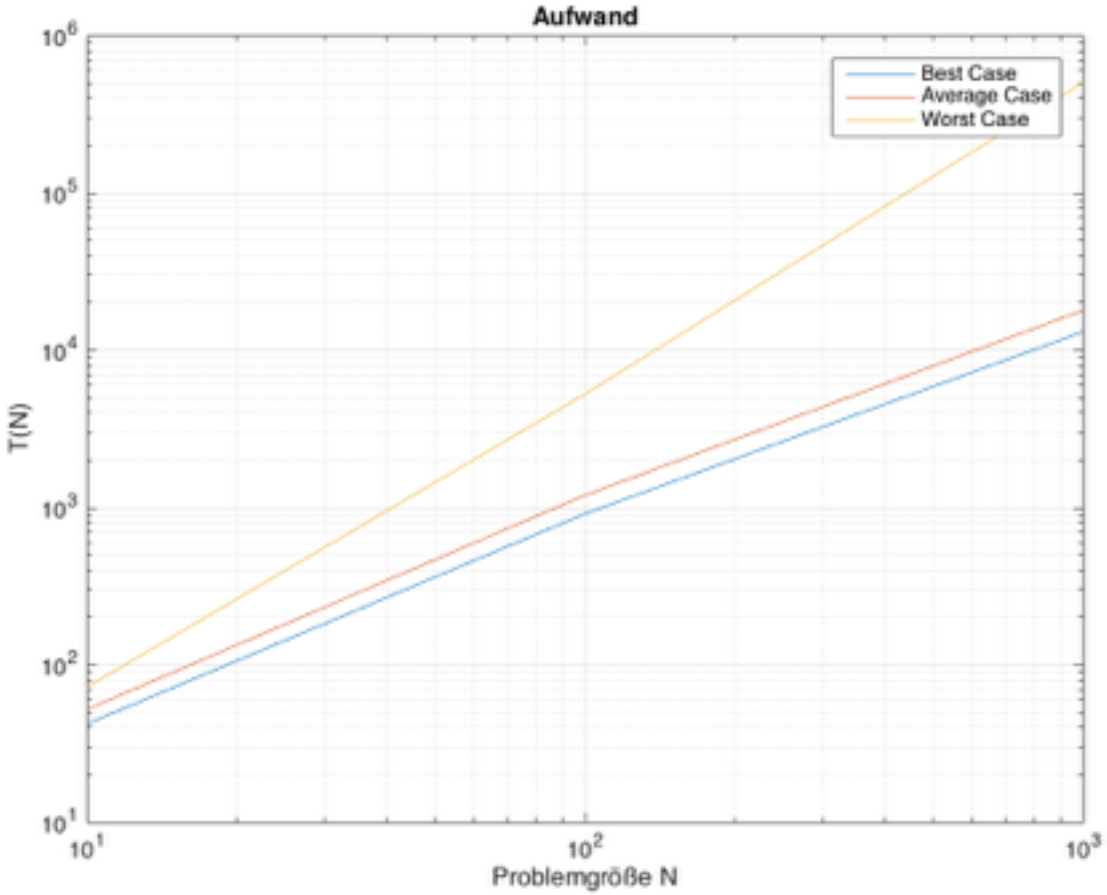

ABBILDUNG 2: DARSTELLUNG DER TESTREIHEN

1.2 Mittleres oder randomisiertes Element des Arrays

Das Pivotelement wird bei dieser Variante zufällig bestimmt. Geht man davon aus, dass die Daten der zu sortierenden Menge gleichmäßig verteilt sind, kann man auch den [2]approximativen Median wählen. Das führt zu gleicher Effizienz.

1.2.1 Best- & Worstcase-Betrachtung

Der **Best Case** wird etwas schlechter als O(n · log n) sein, da die Wahrscheinlichkeit gering ist genau das Pivotelement zu treffen, was die zu sortierende Menge in der Mitte teilt und so für einen niedrigen Rekursionsbaum sorgt.

Der **Worst Case** wird mit geringer Wahrscheinlichlkeit eintreten. D.h. die Komplexität dieser Variante wird unterhalb von O(n^2) liegen.

1.2.2 Testreihen

Geht man davon aus in dieser Variante Best Case und Worst Case eintreten, werden sich die Testreihen wie 1.2 ähneln. Nur tritt der Worst Case mit geringerer Warschlichkeit auf. Der Average Case ist etwas besser.

---

[2] = a[(L + R) / 2]

Listenlänge	Best Case		Avergage Case		Worst Case	
	Vergleiche	Tausch	Vergleiche	Tausch	Vergleiche	Tausch
10	35	7	39	9	63	9
100	718	194	885	183	5148	99
1000	10332	2766	14123	2532	501498	999

1.2.3 Darstellung

Davon ausgehend, dass Best- & Worst Case Fälle aus 1.1 auftreten können, ist die Folgerung das diese Werte fast gleich ausfallen. Diese Werte wurden übernommen. Der Average Case schneidet in dieser Variante aber besser ab . Um ein Vergleich zu 1.1 zu haben wurde auch der Average Case übernommen. Der neue Average Case ist in der Abb. 3 als AC Random bezeichnet. Man sieht eine minimale, aber doch erkennbare Verbesserung gegenüber der Variante aus 1.1.

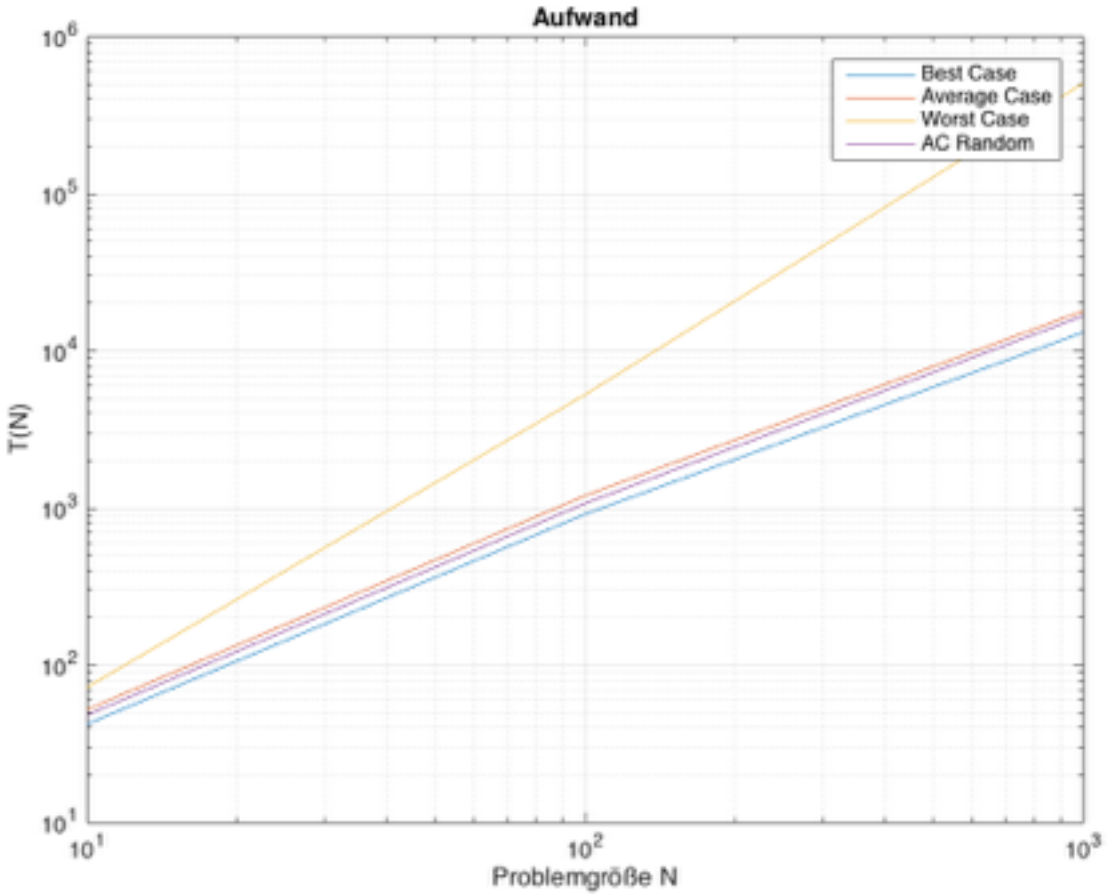

ABBILDUNG 3: DARSTELLUNG DER TESTREIHEN

1.3 Median als Pivotelement

Um den Worst Case auszuschliessen, wird bei dieser Variante der Median der Menge gewählt und für das Pivotelement verwendet. Das bedeutet, dass der Algorithmus am effizientesten funktioniert, da hier sichergestellt wird, dass die Menge immer an der richtigen Stelle geteilt wird und so der Rekursionsbaum bei log(n) liegt. Der Nachtteil ist, das ein hoher Aufwand betrieben werden muss um den Median zu bestimmen. Um den genauen Median zu bestimmen muss die Menge sortiert sein. Ein zusätzlicher Aufwand von O(n · log n) kommt im Mittel pro Rekursion hinzu.

1.3.1 Best- & Worstcase-Betrachtung

Vorausgesetzt die Kosten zum finden des Medians würden ausser Acht gelassen, wählt der Algorithmus in dieser Variante immer den idealen Median und arbeitet immer sehr effizient. D.h. eine Unterscheidung in **Worst**-, **Average** und **Best Case** gibt es nicht, da hier immer der Best Case gewählt wird. Da dies natürlich nicht die Realität abbildet. Wird im folgenden Punkt 1.3.2 bestimmt, wie oft der Median für die Problemgrößen N=10, N=100 & N=1000 in diesem Verfahren wird.

1.3.2 Testreihen

Der Median wird im Mittel für N10 = 8 mal, N100 = 85 mal & N100 = 887 mal bestimmt.
Wenn man annimmt ,das mit jeder Bestimmung eine Sortierung einhergeht wie in 1.2 ergeben sich die folgen Testreihen.

Listenlänge	Best Case		Avergage Case		Worst Case	
	Vergleiche	Tausch	Vergleiche	Tausch	Vergleiche	Tausch
10	280	56	312	72	504	72
100	61.030	16490	75225	15555	437580	8415
1000	9.164.484	2.453.442	12.527.101	2.245.884	444.828.726	886.113

1.3.3 Darstellung

Um die neuen Graphen besser einordnen zu können, sind alle bisherigen Darstellungen mit in die Abbildung 4 übernommen worden. Zu sehen ist wie unter der Annahme aus 1.3.2 sich die Aufwände sehr stark erhöhen (obere 3 Graphen).

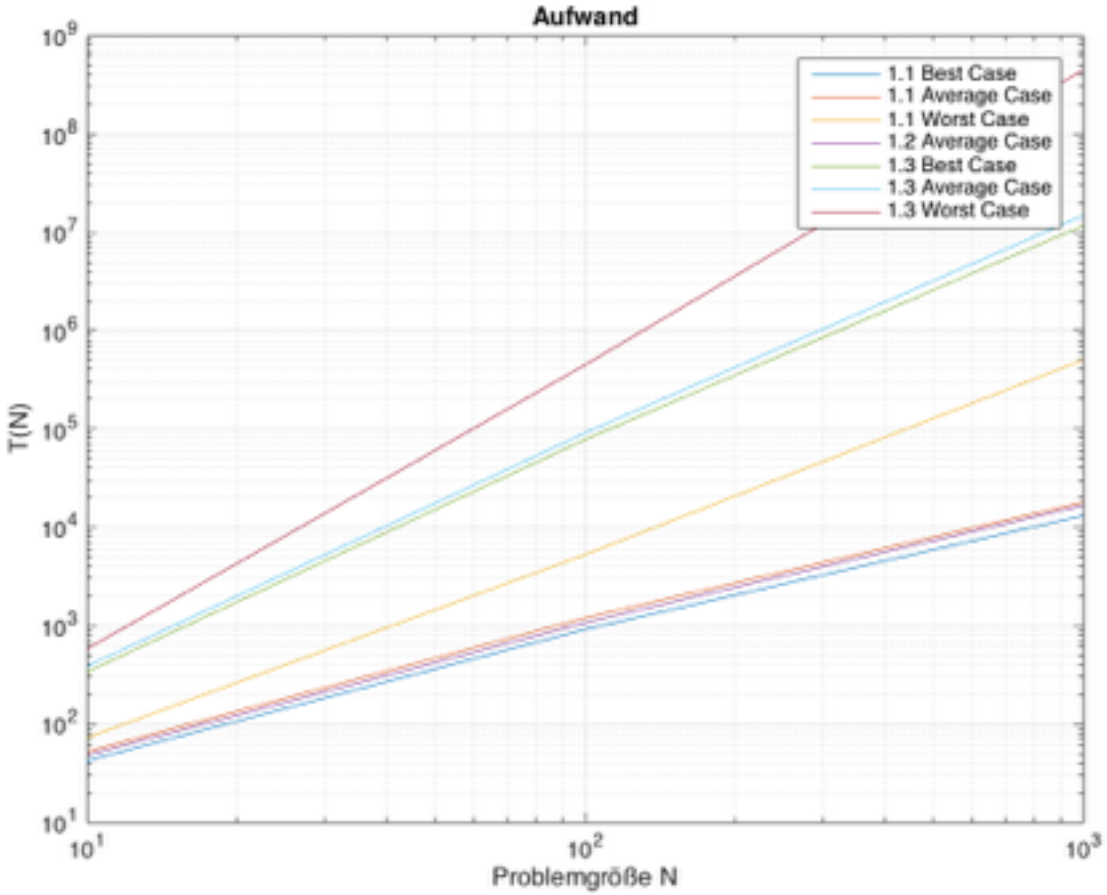

ABBILDUNG 4: DARSTELLUNG DER TESTREIHEN