Efficiëntie van sorteeralgoritmen op willekeurige inputrijen

Konsta Kuosmanen
R0829061

31 maart 2021

Inhoudsopgave

1 Inleiding

In dit artikel worden drie sorteeralgoritmen besproken: QuickSort, InsertionSort, en SelectionSort. De vraag dat er wordt gestelt is: *Hoe efficiënt zijn selection sort, insertion sort, en quicksort op willekeurig verdeelde inputrijen?*. Om de efficiëntie van de sorteeralgoritmen te analyseren wordt er naar de theorie en praktijk erachter gekeken. De drie sorteeralgoritmen worden getest op wilekeurige inputrijen, slechste geval inputrijen, en beste geval inputrijen. Daarbij is er ook een doubling ratio experiment uitgevoert op InsertionSort en QuickSort.

2 Definities

Problem size de grootte van een array.

Invoerrij een array nummers met de grootte n.

Willekeurige invoerrij een invoerrij met nummers tussen 0 en de grootte van de invoerrij.

Best-case een invoerrij met nummers willekeurige nummers tussen 0 en de grootte van de invoerrij, die van klein tot groot zijn geordend. Het is dus niet eigenlijk een best-case geval voor elk sorteeralgoritme.

Worst-case een invoerrij met nummers willekeurige nummers tussen 0 en de grootte van de invoerrij, die van groot naar klein zijn geordend. Het is dus niet eigenlijk een worst-case geval voor elk sorteeralgoritme.

AS(N) is de grootte van de array met grootte N.

MT(N) is de Measured Time of gemeten tijd van een sorteeralgoritme voor een array van lengte N.

MR(N) is de Measured Ratio of gemeten verhouding van een sorteeralgoritme voor een array van lengte N.

PT(N) is de Predicted Time of voorspelde tijd van een sorteeralgoritme voor een array van lengte N.

PR(N) is de Predicted Ratio of voorspelde verhouding van een sorteeralgoritme voor een array van lengte N.

3 Onderzoeksmethode

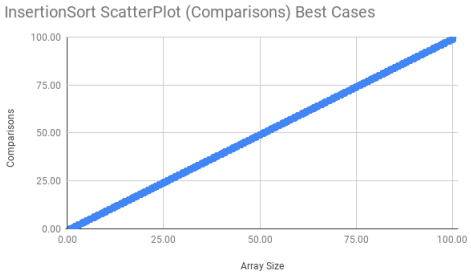
De sorteeralgoritmen zijn in de Java programmeertaal geschreven volgens hun standaardversies.

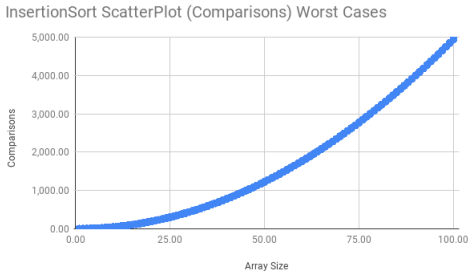
Om de algemene efficiëntie van de sorteeralgoritmen te meten werden er drie soorten invoerrijen in de algoritmen gevoert: willekeurige invoerrijen, beste-geval invoerrijen, en slechst-geval invoerrijen. Elke invoer werd 50 keer getest om kleine een zo groot mogelijke steekproefomvang te hebben. Elke invoer grootte tussen 0 en 101 werden getest.

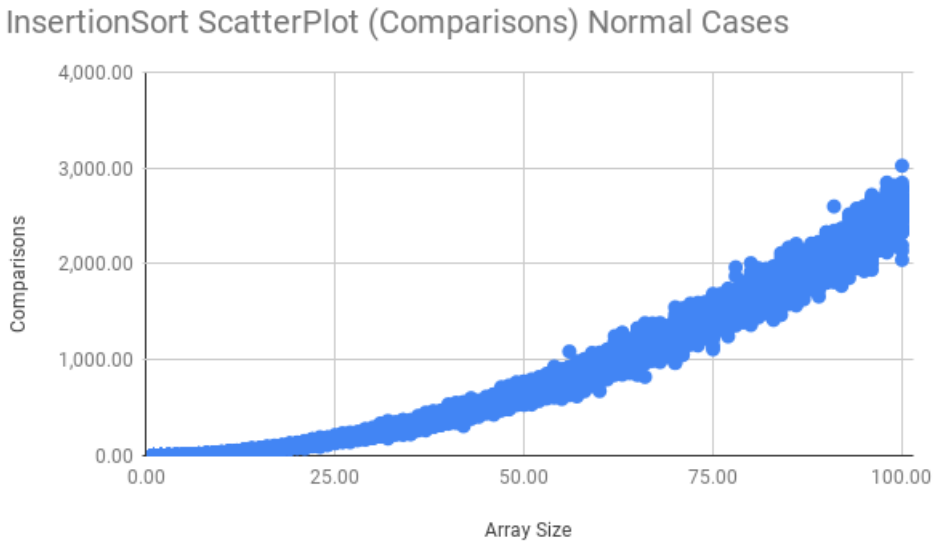
Het doubling-ratio van InsertionSort en QuickSort werd gemeten door willekeurige invoerrijen te genereren voor een bepaalde grootte N. Deze N werd dan gedubbeld en opnieuw gemeten. Dit werd 6 keer in totaal gedaan, waarna er een voorspelling wordt gemaakt voor de volgende 3 grootte N gebaseerd op de data.

4 InsertionSort

4.1 Algemeen







InsertionSort is een sorteeralgoritme dat het definitief gesorteerde array een item tegelijk opbouwd. Theoretisch is InsertionSort een $\sim n^2/4$ algoritme.

We kunnen zien dat in normale gevallen het algoritme afhanklijk is van de data in de array, en dat het sterk op een kwadratische functie lijkt.

Herkenbaar is ook dat het algoritme in het beste geval n vergelijkingen gaat maken, en dat in het slechste geval $(n^2-n)/2$ vergelijkingen gaat maken.

4.2 Doubling Ratio

AS(N)	**MT(N)**	**MR(N)**
640	0.85 ms	1.70
1 280	1.15 ms	1.35
2 560	4.70 ms	4.09
5 120	18.45 ms	3.93
10 240	74.65 ms	4.05
20 480	328.65 ms	4.40

Tabel 1: InsertionSort Doubling Ratio Measurements

Het doubling ratio experiment voor InsertionSort levert de bovenstaande tabel op. We verwachten dat het doubling ratio 4 nadert, want $T(2) = 2^2 = 4$. We zien ook dat de gemeten doubling ratio 4 nadert.

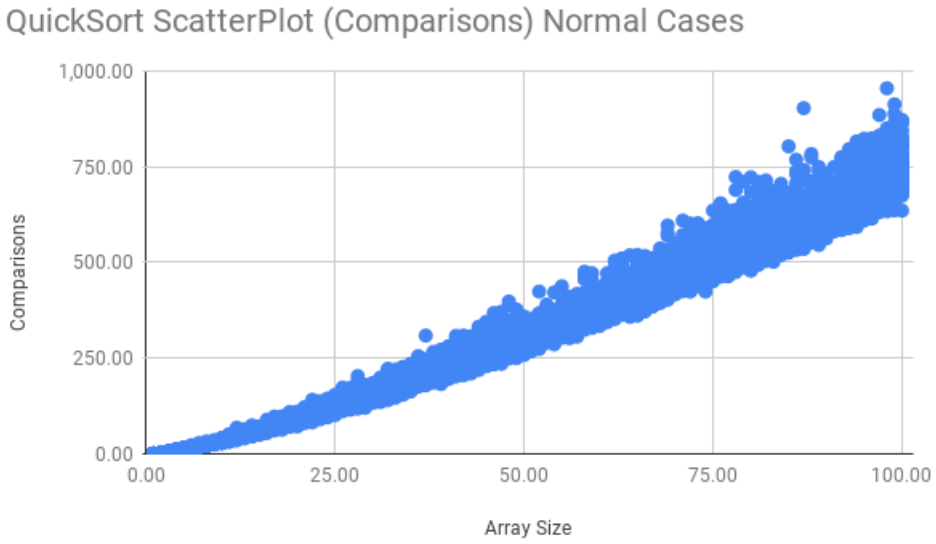
AS(N)	**PT(N)**	**PR(N)**	**MT(N)**	**MR(N)**
40 960	1 314.60 ms	4.00	2 162.25 ms	6.58
81 920	5 258.40 ms	4.00	8 426.65 ms	3.90
163 840	21 033.60 ms	4.00	40 994.95 ms	4.86

Tabel 2: InsertionSort Doubling Ratio Predictions

Voor de voorspelling is het doubling ratio 4 gebruikt zoals hierboven uitgelegt.We zien uit de tabel dat de voorspelling ongeveer correct is. Als we nu een voorspelling maken voor een problem size N die 8 keer groter is dan onze grootste meting, volgt dat: $40994.95 * 4^8 = 2686645043.2$ of 1 maand 16 uur 12 minuten en 37 seconden.

5 QuickSort

5.1 Algemeen



QuickSort is een divide-and-conquer algoritme. Het werkt door een *pivot* te kiezen uit de array en het array daar te partitioneren in twee sub-arrays gebaseerd op of dat het element groter of kleiner is als de *pivot*, de twee sub-arrays worden dan recursief gesorteerd door hetzelfde algoritme.

Theoretisch gezien is QuickSort een $\sim 1,39nlog(n)$ algoritme.

We kunnen zien aan de grafiek dat in de normale gevallen, net zoals InsertionSort, dat er een grote verspreiding is in de vergelijkingen van het algoritme. Omdat het best-case scenario voor QuickSort niet de echt best-case scenario is, krijgen we niet de theoretisch berekende $nlog(n)$ vergelijkingen. Hetzelfe voor het worst-case scenario, we krijgen niet de theoretisch berekende n^2.

5.2 Doubling Ratio

AS(N)	**MT(N)**	**MR(N)**
20 000	1.75 ms	1.94
40 000	3.85 ms	2.20
80 000	8.60 ms	2.23
160 000	20.70 ms	2.41
320 000	38.35 ms	1.85
640 000	88.10 ms	2.30

Tabel 3: QuickSort Doubling Ratio Measurements

Het doubling ratio experiment levert de bovenstaande tabel op. We verwachten dat het doubling ratio 2 nadert want, $T(2) = 2\log_2(2) = 2$ en als N 2 keer zo groot was zou $T(N)$ ook ongeveer 2 moeten zijn. We zien dan ook dat het doubling ratio snel deze theoretische limiet nadert.

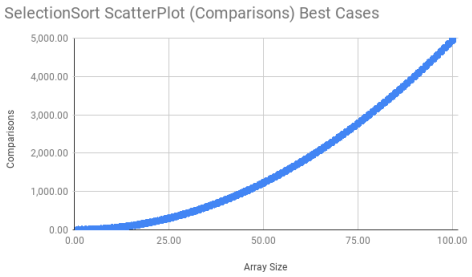
AS(N)	**PT(N)**	**PR(N)**	**MT(N)**	**MR(N)**
1 280 000	176.20 ms	2.00	250.70 ms	2.85
2 560 000	352.40 ms	2.00	561.30 ms	2.24
5 120 000	704.80 ms	2.00	1 325.20 ms	2.36

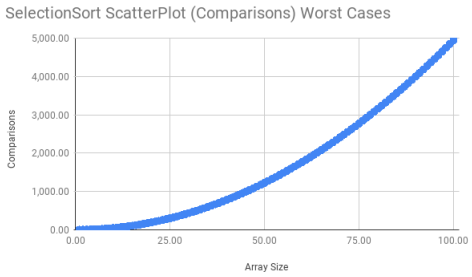
Tabel 4: QuickSort Doubling Ratio Predictions

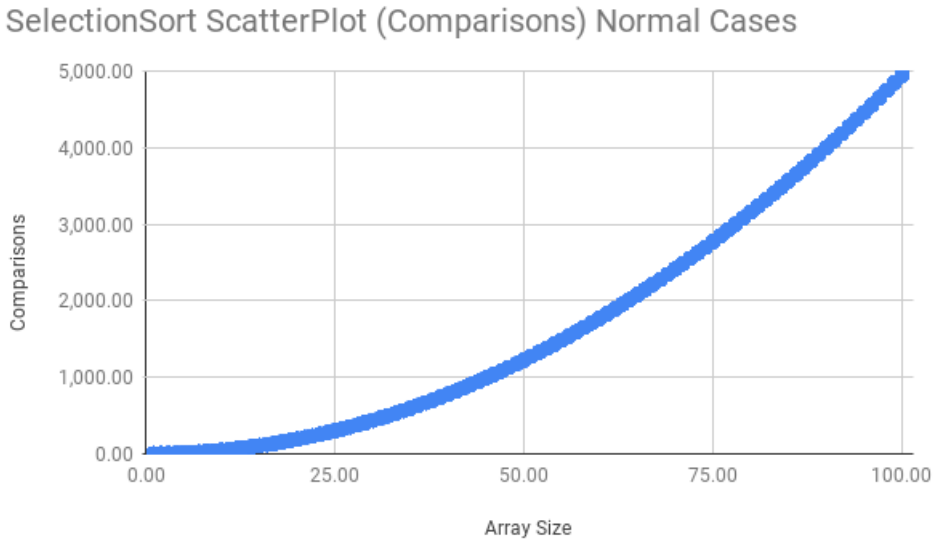
Voor de voorspelling is als doubling ratio 2 genomen zoals hierboven besproken. We zien uit de tabbel dat de voorspelling zich ongeveer richten met de metingen. Als we nu een voorspelling maken voor een problem size N die 8 keer groter is aals onze grootste meeting, dan bekomen we: $1325,20 * 2^8 = 338995,2$ of $\sim$ 338 seconden. Dit is trouwens veel sneller dan bijde InsertionSort en SelectionSort.

6 SelectionSort

6.1 Algemeen







SelectionSort is in-place vergelijkingssorteeralgoritme, het heeft een theoretische tijdscomplexiteit van $\sim n^2/2$. Het algoritme overloopt heel de array van start tot einde, dan van start + 1 to einde, ..., enzovoort. Hieruit kunnen we een logisch besluit maken dat het aantal vergelijkingen gelijk zal zijn voor eenzelfde problem size.

Het eerste dat opvalt bij de data is dat de resultaten van eenzelfde problem size allemaal exact evenveel vergelijkingen gebruiken. Dit is omdat SelectionSort een constant algoritme is. Omdat het geen spreiding heeft is het een heel betrouwbaar algoritme.

7 Conclusie

Als we nu terug komen tot de originele vraag: *Hoe efficiënt zijn selection sort, insertion sort, en quicksort op willekeurig verdeelde inputrijen?*. Uit de data dat er is verzameld blijkt dat QuickSort het meest efficiënt is van

de sorteeralgoritmen getest. Tussen InsertionSort en SelectionSort is InsertionSort het meest efficiënt. Het besluit van dit artikel is dus: QuickSort > InsertionSort > SelectionSort.

8 Github

Excel en source code staan op GitHub