

# GEGEVENSSTRUCTUREN EN ALGORITMEN

---

## Practicum 1: Sorteeralgoritmes

---

*Author:*

STEF TWEEPENNINCKX, R0677232

29 MAART 2018

## Inleiding

In dit verslag, dat ik moet maken voor het vak *Gegevensstructuren en algoritmen*, zoek ik het antwoord op de onderzoeksvraag "*Hoe efficiënt zijn selection sort, insertion sort, en quicksort op random data?*"

Om het antwoord op deze vraag heb ik de drie sorteeralgoritmes geïmplementeerd en een aantal experimenten uitgevoerd. Deze experimenten worden verder beschreven.

## Aantal vergelijkingen

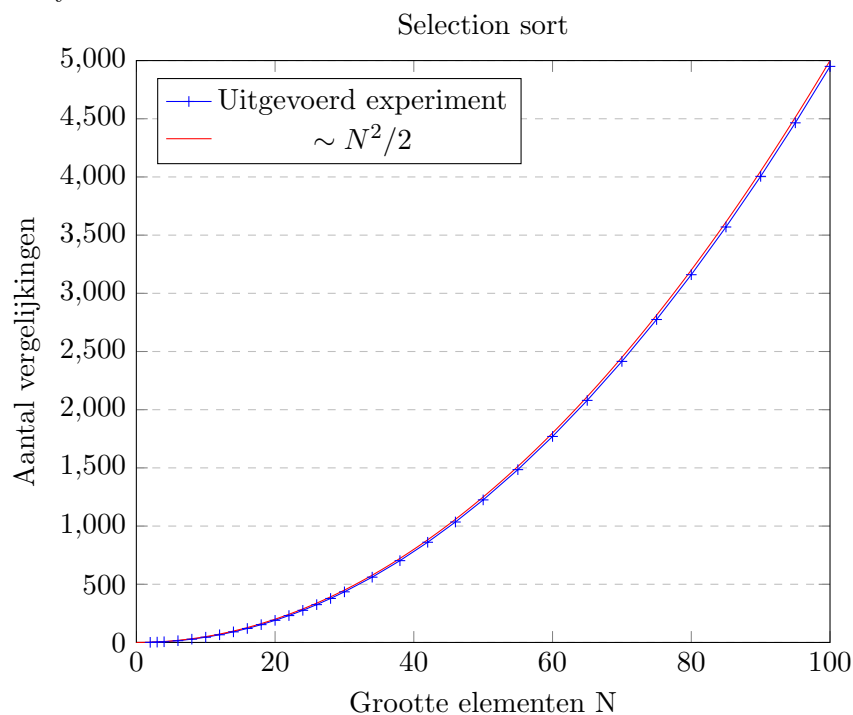
### Selection sort

**Werking selection sort** Selection sort is een van de meest eenvoudige sorteeralgoritmen. We doorlopen de array op zoek naar het kleinste element, waarna we dit element verwisselen met het eerste element van de array. Herhaal deze stappen tot we op het einde van de array zitten en de array gesorteerd is.

**Verwachtingen** Bij selection sort verwachten we dat het aantal vergelijkingen, voor een array van  $N$  elementen, groeit volgens  $\sim \frac{N^2}{2}$ .

### Experiment

De geplotte data (zie appendix) van de uitvoering van selection sort voor arrays met verschillend aantal elementen  $N$ :



**Conclusie experiment** Op bovenstaande grafiek kunnen we duidelijk zien dat de experimenten overeenkomen met de verwachte resultaten. De eventuele lichte afwijkingen kunnen we toeschrijven aan het feit dat we met random gegenereerde data zitten. Tussen deze random arrays kunnen arrays zitten die al (deels) gesorteerd zijn. Hierdoor hebben we dus minder vergelijkingen dan verwacht.

We kunnen ook zien dat het aantal vergelijkingen zeer snel stijgt met het

aantal elementen. Hieruit kunnen we afleiden dat selection sort inefficiënt is voor arrays met een groot aantal elementen.

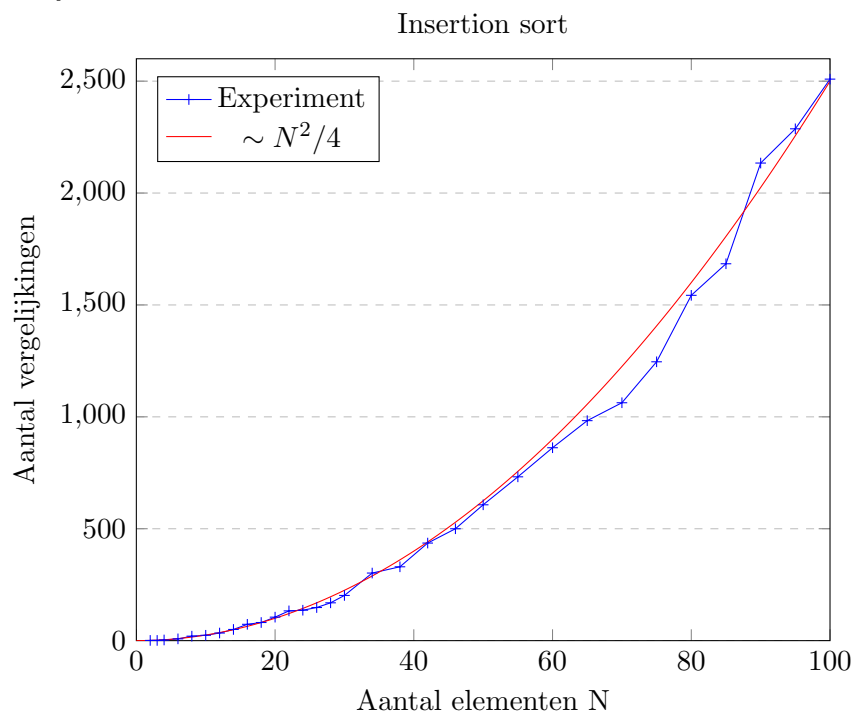
## Insertion sort

**Werking insertion sort** Insertion sort is net als selection sort een eenvoudig sorteeralgoritme. We doorlopen de array van links naar rechts. Als het huidige element kleiner is dan een van de voorgaand elementen, verschuiven we het huidige element naar voor tot er geen kleiner element voor het huidige staat.

**Verwachtingen** Bij insertion sort verwachten we dat het aantal vergelijkingen, voor een array van  $N$  elementen, groeit volgens  $\sim \frac{N^2}{4}$ . In het worst-case geval (een array die gesorteerd is van groot naar klein) groeit het aantal vergelijkingen volgens  $\sim \frac{N^2}{2}$

## Experiment

De geplote data (zie appendix) van de uitvoering van insertion sort voor arrays met verschillend aantal elementen  $N$ :



**Conclusie experiment** Op bovenstaande grafiek kunnen we opnieuw zien dat de experimenten overeenkomen met de verwachte resultaten. De resultaten wijken wel meer af van de verwachte resultaten dan bij selection sort. Dit komt omdat het eventuele al (deels) gesorteerd zijn van de arrays, een grotere invloed heeft op het aantal vergelijkingen dan bij selection sort. Bij selection sort vergelijken we namelijk enkel met het eerste element, waar

we bij insertion sort met elk voorgaand element vergelijken. Het aantal vergelijkingen stijgt ook bij insertion sort snel met het aantal elementen, maar minder dan bij selection sort. Hieruit kunnen we afleiden dat insertion sort een zeer goed algoritme is voor arrays met weinig elementen. Daarom wordt insertion sort vaak gebruikt als cutoff bij quicksort indien er weinig elementen te sorteren zijn.

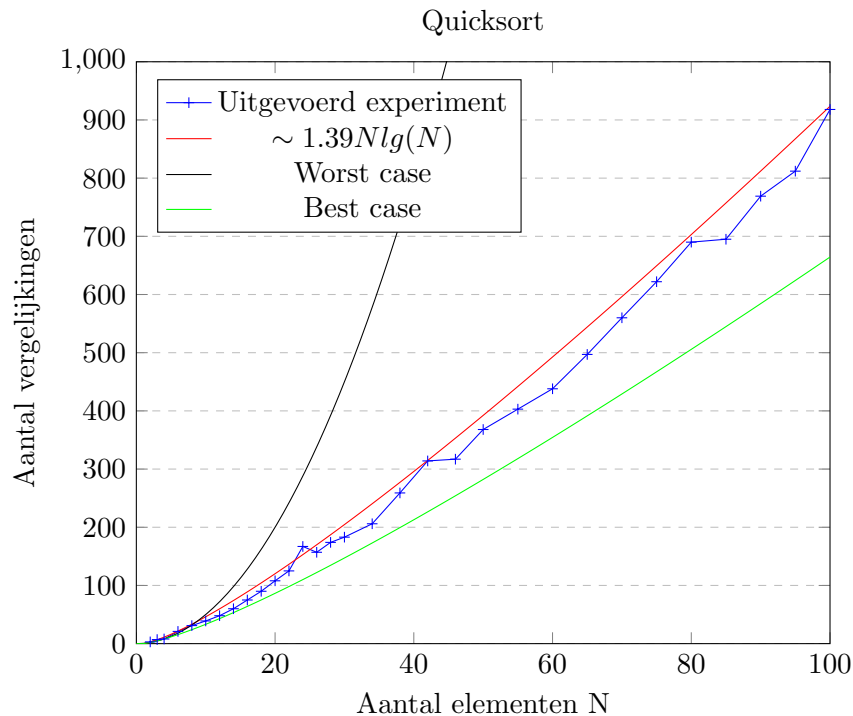
## Quicksort

**Werking quicksort** Quicksort is een recursief sorteeralgoritme en daardoor iets minder eenvoudig dan voorgaande. We kiezen een pivot, meestal het 1e element, en vergelijken alle andere elementen uit de array hiermee. Zo delen we de array op in 3 delen: de pivot, elementen kleiner dan de pivot, en elementen groter dan de pivot. Dit herhalen we dan op de 2 deelarrays, enz.

**Verwachtingen** Bij quicksort verwachten we dat het aantal vergelijkingen, voor een array van  $N$  elementen, groeit volgens  $\sim 1.39n \lg(n)$ .

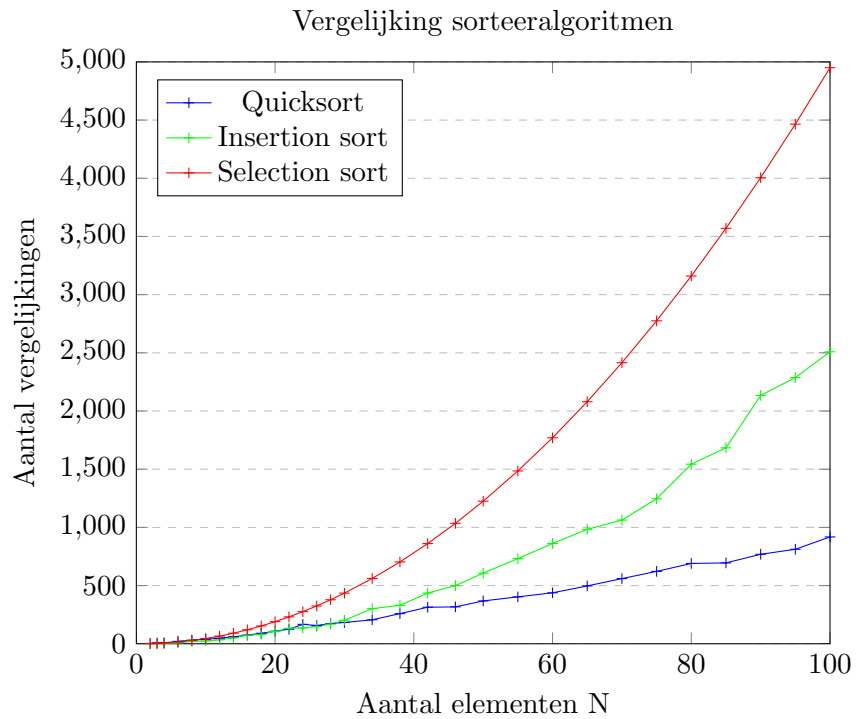
## Experiment

De geplotte data (zie appendix) van de uitvoering van quicksort voor arrays met verschillend aantal elementen  $N$ :



**Conclusie experiment** De resultaten van dit experiment komen overeen met de verwachte waarden. Opnieuw zijn er aantal lichte afwijkingen, maar ze bevinden zich mooi tussen de best en worst case van quicksort. De reden van afwijking bij quicksort is de keuze van de pivot, deze deelt de array niet altijd perfect in twee en dit kan leiden tot meer of minder vergelijkingen dan verwacht.

## Algemene conclusie 1



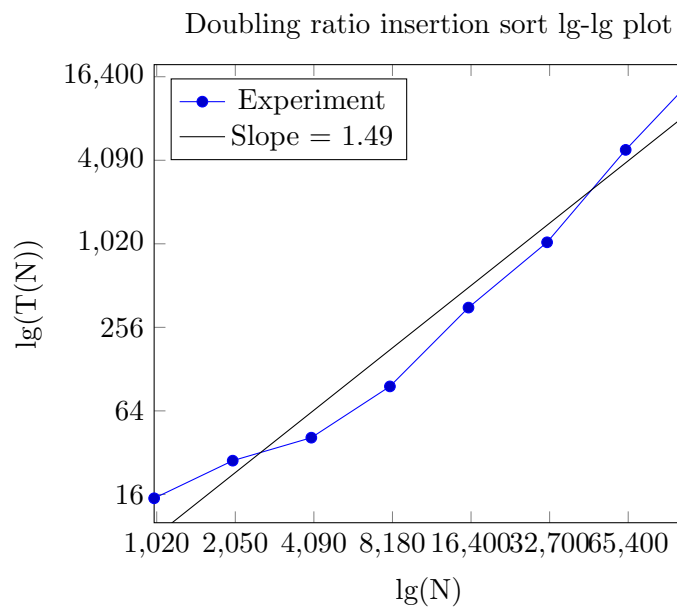
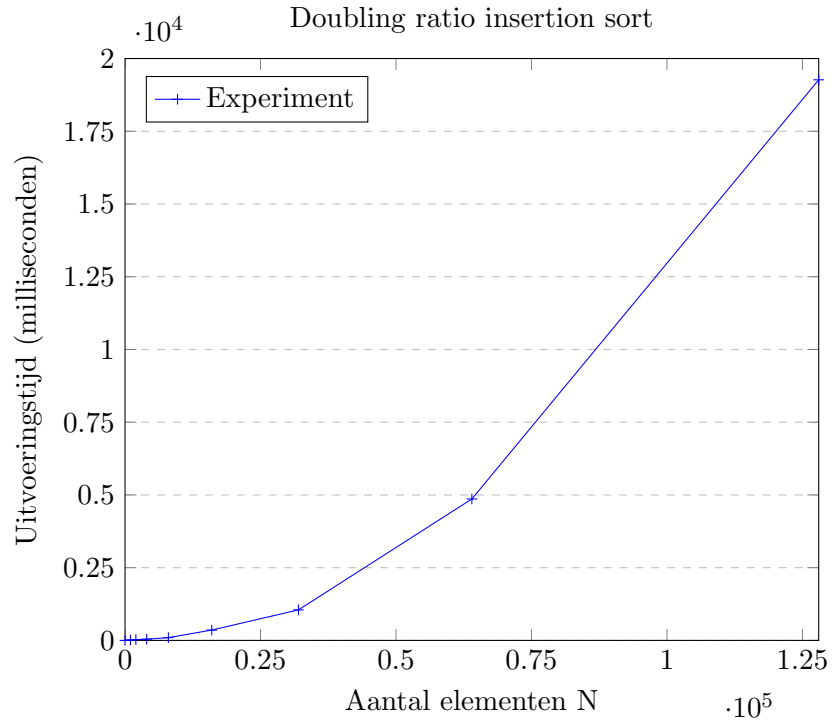
Zoals we duidelijk kunnen zien is quicksort het meest efficiënte algoritme voor arrays met een groot aantal elementen. Bij een klein aantal elementen,  $< 20$ , ligt het aantal vergelijkingen echter heel dicht bij elkaar. Vooral bij insertion sort en quicksort is het verschil zeer klein. Zoals eerder gezegd is dit een van de redenen dat insertion sort soms gebruikt wordt als cutoff bij quicksort voor een klein aantal elementen.



## Doubling ratio

### Insertion sort

We voeren het doubling ratio experiment uit en plotten de resultaten:



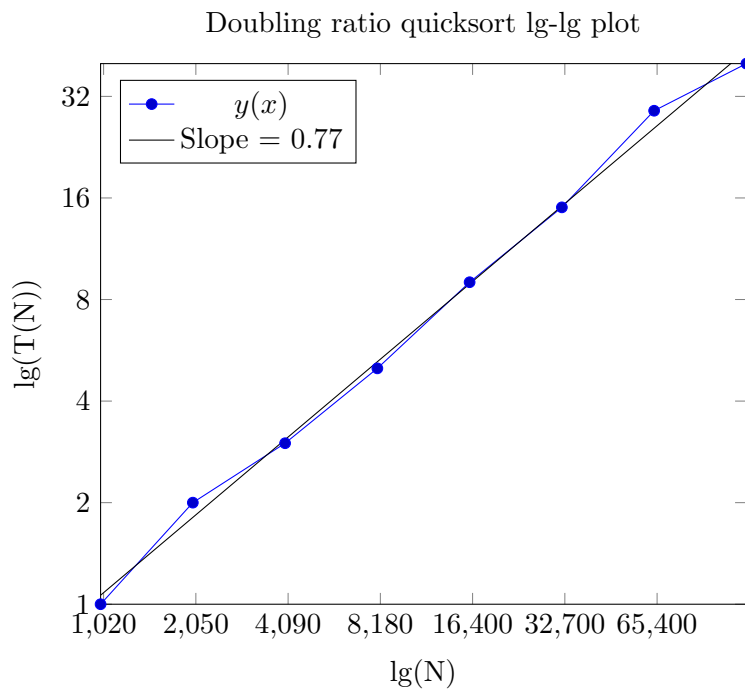
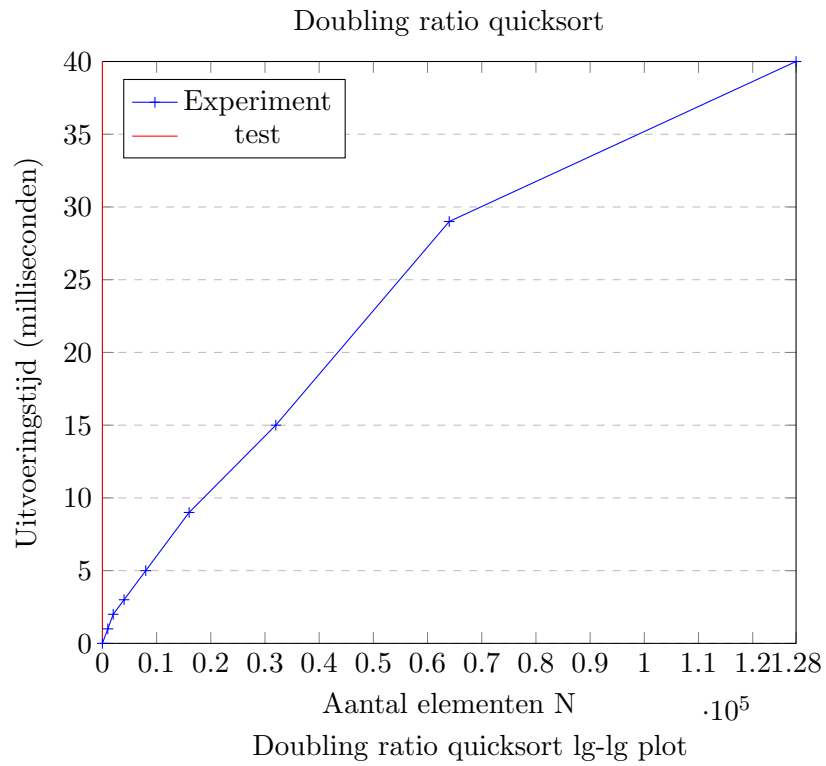
We krijgen een rechte met een slope van 1.49,  $T(N) = aN^b$  en  $b = 1.49$ . Om de voorspelling te kunnen maken voor de uitvoeringstijd met  $N = 1024000$ , moeten we dus eerst  $a$  berekenen. Dit doen we door de vergelijking  $1049 = a \times 32000^{1.49}$  op te lossen en dit geeft ons  $a = 0.203 \times 10^{-3}$ . Nu kunnen we een voorspelling maken voor  $N = 1024000$ , immers  $T(N) = 0.203 \times 10^{-3} N^{1.49}$ . Als we  $N = 1024000$  invullen, krijgen we  $T(1024000) = 183165$  milliseconden.

<b>N</b>	<b>Tijd (milliseconden)</b>	<b>ratio</b>	<b>lg ratio</b>
1000	15	-	-
2000	28	1.9	0.93
4000	41	1.5	0.58
8000	96	2.3	1.20
16000	355	3.7	1.89
32000	1049	3	1.58
64000	4861	4.6	2.20
128000	19269	4	2

Tabel 1: Doubling ratio insertion sort

## Quicksort

We voeren het doubling ratio experiment uit en plotten de resultaten:



We krijgen een rechte met een slope van 0.77,  $T(N) = aN^b$  en  $b = 0.77$ . Om de voorspelling te kunnen maken voor de uitvoeringstijd met  $N = 1024000$ , moeten we dus eerst  $a$  berekenen. Dit doen we door de vergelijking  $15 = a \times 32000^{0.77}$  op te lossen en dit geeft ons  $a = 0.509 \times 10^{-2}$ . Nu kunnen we een voorspelling maken voor  $N = 1024000$ , immers  $T(N) = 0.509 \times 10^{-2} N^{0.77}$ . Als we  $N = 1024000$  invullen, krijgen we  $T(1024000) = 216$  milliseconden.

N	Tijd (milliseconden)	ratio	lg ratio
1000	1	-	-
2000	2	2	1
4000	3	1.5	0.58
8000	5	1.7	0.77
16000	9	1.8	0.85
32000	15	1.7	0.77
64000	29	1.9	0.93
128000	40	1.4	0.49

Tabel 2: Doubling ratio quicksort

## 1 roemel

