Verslag practicum 1: sorteeralgoritme

Inleiding

In dit verslag zullen we onderzoeken hoe efficiënt selection sort, insertion sort en quicksort zijn met willekeurig gekozen data. Ik heb dit onderzocht door middel van een aantal experimenten.

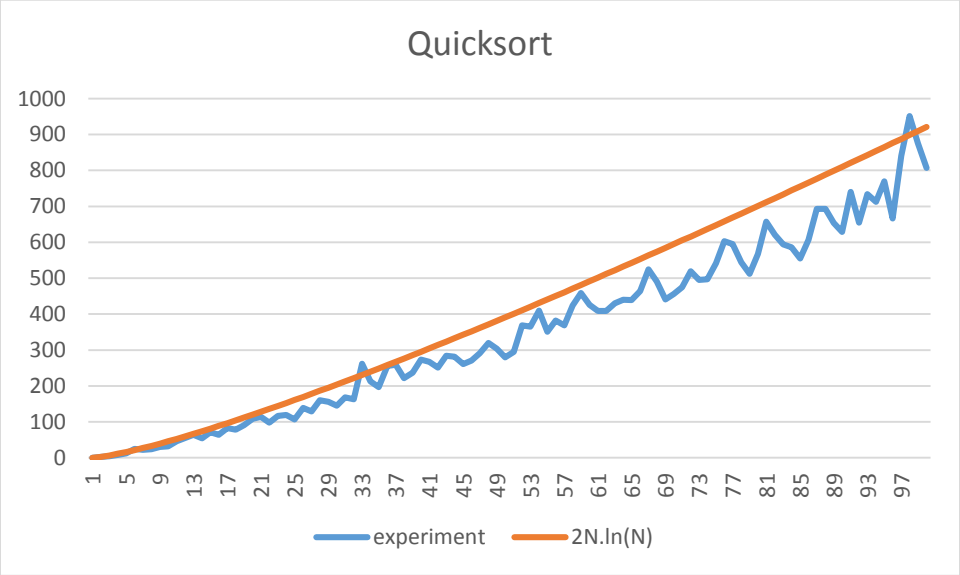
Quicksort

Dit is een sorteer algoritme dat in vergelijking met de andere die we hier onder bespreken veel sneller is. Zeker als de data al bijna gesorteerd of omgekeerd gesorteerd is.

Dit komt omdat quicksort een pivot gebruikt om mee te vergelijken in plaats van elk element in de te sorteren rij of de al gesorteerde rij.

Hieronder kan u een grafiek zien dat het # vergelijkingen weergeeft(op de y-as) bij het sorteren van een rij van 1 t.e.m. 100 elementen (weergegeven op de x as).

Er werd telkens willekeurige data gesorteerd met het originele quicksort algoritme(met '2-way sort'). Het blauwe lijnstuk geeft weer wat de conclusie van het experiment was en het rode lijnstuk geeft weer hoeveel het theoretisch gemiddeld # vergelijkingen zou moeten zijn.



Bij dit experiment kwam ik tot de conclusie dat Quicksort minder vergelijkingen gebruikt dan onder vermelde sorteeralgoritme in vergelijkbare omstandigheden.

Louis Roebben
r0589423

Selection sort

Dit is een sorteer algoritme dat in vergelijking met de andere die we hier bespreken veel trager is. De prestatie van selection sort is wel onafhankelijk van hoe de data verdeeld is over de rij.

Dit komt omdat selection sort een element vergelijkt met alle elementen in de te sorteren rij.

Hieronder kan u een grafiek zien dat het aantal vergelijkingen weergeeft(op de y-as) bij het sorteren van een rij van 1 t.e.m. 100 elementen (weergegeven op de x as).

Er werd telkens willekeurige data gesorteerd met het selection sort algoritme. Het blauwe lijnstuk geeft weer wat de conclusie van het experiment was en het rode lijnstuk geeft weer hoeveel het theoretisch gemiddeld # vergelijkingen zou moeten zijn.

Zoals u misschien ziet is het experiment bijna gelijk aan de theoretische verwachtingen. Dit komt natuurlijk vanwege het boven vermelde kenmerk van selection sort zijnde dat dit algoritme onafhankelijk is van data verdeling.

Louis Roebben
r0589423

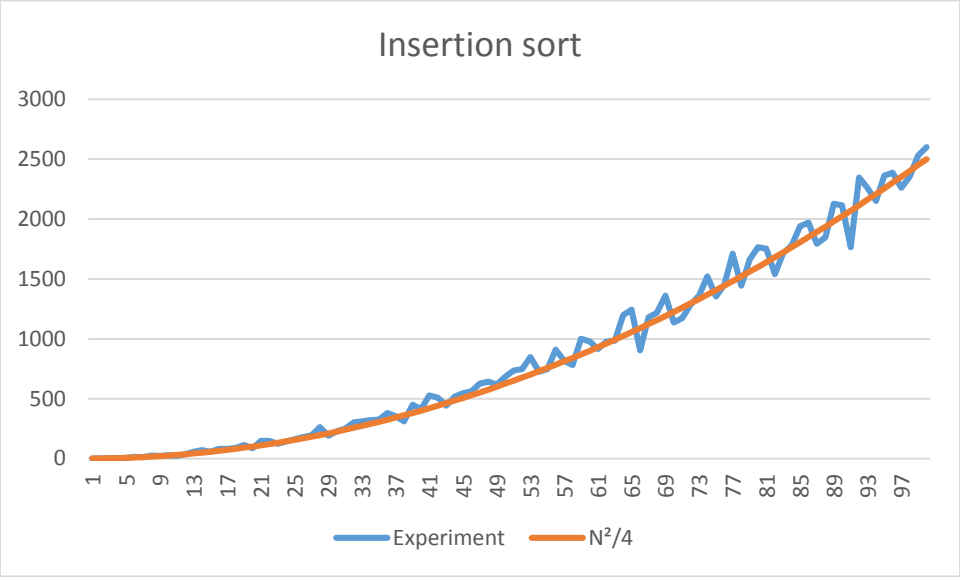
Insertion sort

Dit is een sorteer algoritme dat in vergelijking met de andere die we hier bespreken veel trager is. De prestatie van insertion sort is wel onafhankelijk van hoe de data verdeeld is over de rij.

Dit komt omdat insertion sort een element vergelijkt met alle elementen in de te sorteren rij.

Hieronder kan u een grafiek zien dat het aantal vergelijkingen weergeeft(op de y-as) bij het sorteren van een rij van 1 t.e.m. 100 elementen (weergegeven op de x as).

Er werd telkens willekeurige data gesorteerd met het selection sort algoritme. Het blauwe lijnstuk geeft weer wat de conclusie van het experiment was en het rode lijnstuk geeft weer hoeveel het theoretisch gemiddeld # vergelijkingen zou moeten zijn.



Hoewel er veel afwijkingen zijn naar maten onze rij groter word is het gemiddeld # vergelijkingen ongeveer wel gelijk aan de theoretische verwachting van $N^2/4$ met N zijnde het # elementen in de rij.

Louis Roebben
r0589423

Doubling ratio experiment

- ***Quicksort***

  In dit experiment hebben zijn we opzoek gegaan naar het groei ratio.

  Als we het gemiddelde nemen van de 7 laatste groei ratio's dan komen we uit op een gemiddelde van ~2,4.

  Daaruit volgt dat b gelijk is aan $\log_2(\sim 2{,}4)$ of 1,26

  Als we kijken naar de laatste rij in onze tabel dan kunnen we a berekenen als volgt: a is gelijk aan 31,980s/67108864^b of ~4.39*10^-9

  Vervolgens kunnen we T(N) berekenen: T(N)= ~4.39 * 10^-9 * N^b

  Deze formule laat ons toe om de runtime te berekenen met een veel grotere N.

  T(1 000 000 000)= ~4.39 * 10^-9
  *1 000 000 000^b = ~ 960,43s

N	runtime	growth ratio
4096	2	0
8192	6	3
16384	10	1,6666667
32768	4	0,4
65536	9	2,25
131072	18	2
262144	34	1,8888889
524288	85	2,5
1048576	252	2,9647059
2097152	586	2,3253968
4194304	1082	1,8464164
8388608	2577	2,3817006
16777216	5792	2,2475747
33554432	13717	2,3682666
67108864	31980	2,3314136

  Voor N zijnde het aantal elementen in de rij; runtime zijnde hoe lang de berekening duurde in ms en growth ratio de verhoudig tussen de runtime van $\sqrt{N}$ en N.

  Het gene wat we uit dit al kunnen afleiden is dat quicksort relatief geschikt is voor het behandelen van rijen met veel elementen.

Louis Roebben
r0589423

- ***Insertion sort***

  Zoals u kan zien in de tabel hier rechts is bij insertion sort het groei ratio niet consistent, hierdoor kunnen we ook niet gebreken hoe lang het zou duren voor een grotere n. Voor N zijnde het aantal elementen in de rij; runtime zijnde hoe lang de berekening duurde in ms en growth ratio de verhoudig tussen de runtime van √N en N.

N	runtime	growth ratio	
1024	2	0	
2048	25	12,5	
4096	20	0,8	
8192	80	4	
16384	387	4,8375	
32768	1312	3,390181	
65536	5302	4,041159	
131072	22095	4,167295	
262144	62808	2,842634	
524288	533706	8,497421	

  Wat we wel al kunnen zien uit dit kleiner experiment is dat insertion sort een slecht alternatief is ten opzichten van quicksort.

Louis Roebben
r0589423