Algoritmiek

Tiemon Steeghs

Inhoud

[1. Aanleiding 3](#_Toc135138497)

[2. Hoofdvraag 4](#_Toc135138498)

[2.1. Welke sorteer algoritmes zijn er? 4](#_Toc135138499)

[2.2. Hoe werkt het bubble sorteeralgoritme ? 4](#_Toc135138500)

[3. Conslusie 5](#_Toc135138501)

[4. Bronvermelding 6](#_Toc135138502)

# Aanleiding

Om beter te bergrijpen wat voor sorteer algoritmes er bestaan, maak ik verschillende opdrachten die hier betrekking tot hebben.

# Hoofdvraag: welke sorteer algoritmes zijn er en hoe zou je ze toepassen?

## Welke sorteer algoritmes zijn er?

Er zijn veel soorten sorteeralgoritmes. Een lijst kan je hieronder zien. Elk algoritme heeft zijn eigen voor en nadelen waardoor ze in verschillende contexten toepasbaar zijn. Algoritmes hebben allemaal hun eigen tijd complexiteit statistiek. Deze worden genoteerd in de zogeheten “big O” notatie. Het bubble sorteer algoritme heeft een big O notatie van O(N^2), wat inhoud dat bij complexere situaties het algoritme een stuk slomer wordt.

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

(GeeksforGeeks, 2023)

## Hoe werkt het bubble sorteeralgoritme ?

Het bubble sorteeralgoritme is een van de simpelste sorteer algoritmes. Bij het bubble sorteeralgoritme vergelijk je steeds een array element met de twee elementen die er naast zitten. Bij een array van bijvoorbeeld [5, 0, 8, 3] zou het element 8 dan vergeleken kunnen worden met 0 en 3. Hierbij zou je kunnen kijken wel getal groter is zodat de array van klein naar groot gesorteerd kan worden.

Het bubble sorteeralgoritme is niet super efficient waardoor het niet heel geschikt is voor grote hoeveelheden data. Dit zie je ook terug in zijn “big O” rating van O(N^2), wat inhoud dat hoe complexer de data, hoe langer het algoritme over het sorteren doet.

## Hoe werkt het merge sorteeralgoritme?

Het merge sorteeralgoritme werkt aan de hand van het splitsen van de gegeven array in twee lossen arrays. Door vervolgens deze twee arrays te sorteren en weer samen te voegen krijg je een gesorteerde array. Op grotere arrays wordt dit principe meerdere keren toegepast op dezelfde array. Bij een array van 8 elementen wordt de data meerdere keren gesplits.

Het merge sorteeralgoritme heeft een tijdscomplexiteit score van O(n log n) in zijn meest slechte gevallen, wat inhoud dat het algoritme een goede optie is voor het sorteren van grote hoeveelheden data.

## Hoe werkt het selection sorteeralgoritme?

Het selection sorteeralgoritme is een relatief eenvoudig algoritme wat ook gebruikt kan worden om arrays te sorteren. Het algoritme werkt doormiddel van steeds het kleinste element in de array te pakken en deze dan vervolgens vooraan zetten van het ongesorteerde gedeelte van de array. Dus bij de array [9,4,3,6,7,2] zal als eerste het getal 2 worden gepakt en vooraan gezet zodat je [2,9,4,3,6,7] hebt. Daarna komt 3 en dan 4 etc.

De tijdscomplexiteit score van dit algoritme is O (N^2) net als die van de bubbelsort. Dit betekent dus dat dit algoritme makkelijk is om te gebruiken maar dat het tegenvalt bij complexere data.

## Wat zijn de verschillen tussen de onderzochte sorteeralgoritmes, welke is het snelst?

Om de verschillen tussen de drie sorteeralgoritmes nog verder inkaart te brengen heb ik verschillende metingen gedaan. In de tabel hieronder zie je de drie algoritmes die getest zijn op 10 testdata bestanden. Voor elk testbestand zijn het aantal swaps, comparisions en de tijd genoteerd.

### Metingen

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Bubble Sort** | **Merge Sort** | **Selection Sort** |
| Testdata In1\_1 | Swaps: 5  Comparisons: 10  Tijd: 0m0.412s | Swaps: 12  Comparisons: 7  Tijd: 0m0.022s | Swaps: 3  Comparisons: 10  Tijd: 0m0.023s |
| Testdata In1\_2 | Swaps: 15  Comparisons: 45  Tijd: 0m0.033s | Swaps: 34  Comparisons: 25  Tijd: 0m0.018s | Swaps: 8  Comparisons: 45  Tijd: 0m0.023s |
| Testdata In1\_3 | Swaps: 15  Comparisons: 45  Tijd: 0m0.033s | Swaps: 34 Comparisons: 19  Tijd: 0m0.023s | Swaps: 0  Comparisons: 45  Tijd: 0m0.018s |
| Testdata In1\_4 | Swaps: 0  Comparisons: 4999950000  Tijd: 0m4.696s | Swaps: 1668928  Comparisons: 853904  Tijd: 0m0.076s | Swaps: 0  Comparisons: 4999950000  Tijd: 0m3.354s |
| Testdata In1\_5 | Swaps: 0  Comparisons: 4999950000  Tijd: 0m4.026s | Swaps: 1668928 Comparisons: 853904  Tijd: 0m0.071s | Swaps: 0 Comparisons: 4999950000  Tijd: 0m3.643s |
| Testdata In1\_6 | Swaps: 0  Comparisons: 4999950000  Tijd: 0m4.549s | Swaps: 1668928  Comparisons: 853904  Tijd: 0m0.036s | Swaps: 0 Comparisons: 4999950000  Tijd: 0m2.686s |
| Testdata In2\_1 | Swaps: 1593  Comparisons: 3570  Tijd: 0m0.028s | Swaps: 552 Comparisons: 440  Tijd: 0m0.023s | Swaps: 81 Comparisons: 3570  Tijd: 0m0.017s |
| Testdata In2\_2 | Swaps: 302  Comparisons: 780  Tijd: 0m0.030s | Swaps: 216  Comparisons: 161  Tijd: 0m0.021s | Swaps: 34 Comparisons: 780  0m0.018s |
| Testdata In2\_3 | Swaps: 2438 Comparisons: 4950  Tijd: 0m0.028s | Swaps: 672 Comparisons: 540  Tijd: 0m0.022s | Swaps: 98 Comparisons: 4950  Tijd: 0m0.020s |
| Testdata In2\_4 | Swaps: 247086 Comparisons: 498501  Tijd: 0m0.384s | Swaps: 9965 Comparisons: 8664 | Swaps: 991 Comparisons: 498501  Tijd: 0m0.358s |

### Conclusie

Als je kijkt naar de metingen die ik heb gemaakt is het erg duidelijk te zien dat merge sort een stuk geschikter algoritme is voor grote data sets. Dit zie je terug in het veel kleinere aantal swaps en comparisons bij dataset 1\_4, 1\_5 en 1\_6. Bij kleinere data sets daarentegen, zijn de bubbel en selection sort helemaal prima om te gebruiken, ze zijn zelfs efficiënter in sommige gevallen. In mijn metingen komt ook een duidelijk tijdsverschil naar voren. Maar omdat de tijd ook afhankelijk is van veel andere factoren, zoals de specificaties van de computer, is deze data niet een erg accurate meting voor het bepalen van het meest efficiënte algoritme.

## Wat is een Linked list?

Part 1.1: Dit probleem los je op door gebruik te maken van een generiek gegevenstype, zoals een pointer naar “void\*”. Door een pointer naar void te gebruiken, kan de gelinkte lijst elk type gegevens opslaan zonder dat de gelinkte lijstmodule specifieke kennis heeft over het opgeslagen datatype.

Part 1.2: Als je meerdere lijsten wilt kunnen aanmaken moet je dus eigenlijk meerdere lijstobjecten kunnen aanmaken. Hiervoor kan je dus een lijststruct aanmaken waarin het voorste element wordt opgeslagen.

Part 1.3: Dit probleem los je op door dus een size variabele toe te voegen aan de struct waarin wordt opgeslagen hoeveel memory nodig is voor de data.

Part 1.4: Door dus alles variabele te maken zou het niet uit moeten maken voor de geheugenmanager als de linkedlist wordt aangepast.

## Hoe en wanneer merge je elementen in de free list van een memory manager?

Om te weten of er data gemerged kan worden kan je de volgende berekening uitvoeren:

*Adress – size (in bytes) = address van free list element*

Dit werkt omdat stel je hebt:

*Free list*

*1000 - 20*

*1040 - 60*

*Alloc list*

*1020 – 20*

En je geeft 1020 – 20 vrij dan krijg je:

1020-20 = 1000, het address van de eerste element van de free list is ook 1000 dus er kan gemerged worden naar:

*Free list*

*1000 - 40*

*1040 - 60*

*Alloc list*

En nu omdat 1040 – 40 = 1000 wat ook een element in de list is kan je nog een keer mergen:

*Free list*

*1000 – 100*

*Alloc list*

# Conslusie

# Bronvermelding

GeeksforGeeks. (2023). Sorting Algorithms. *GeeksforGeeks*. <https://www.geeksforgeeks.org/sorting-algorithms/>

*Introducing ChatGPT*. (z.d.). <https://openai.com/blog/chatgpt>