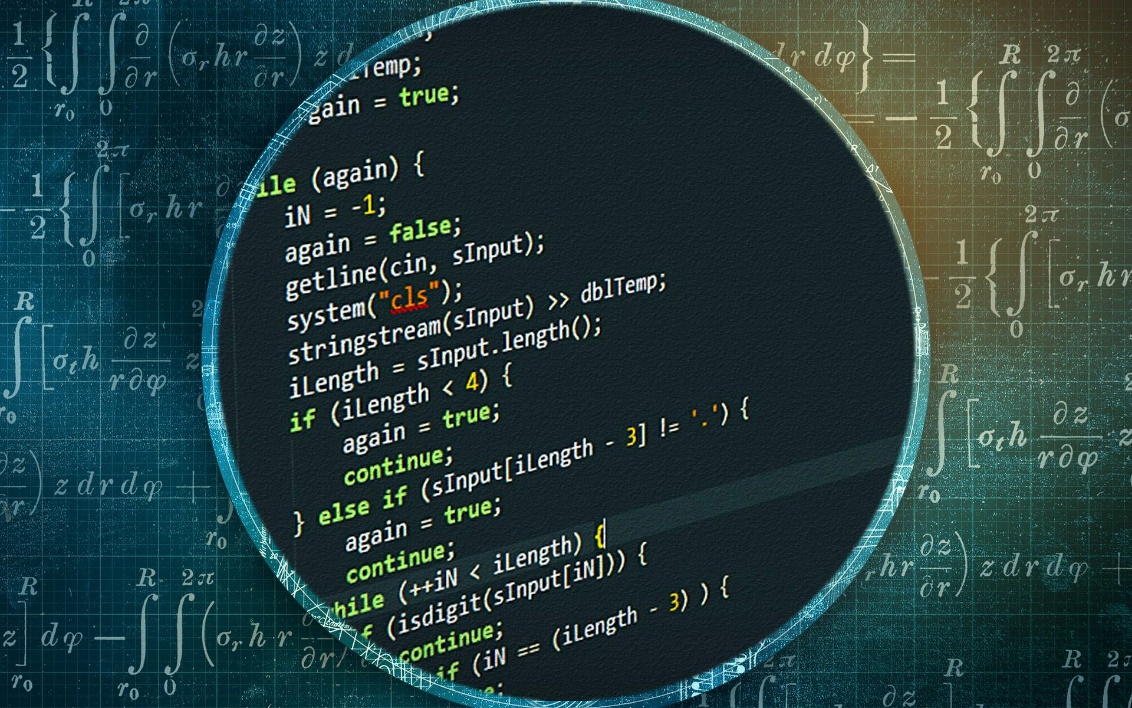
Onderzoek Algoritmiek

Door: Tiemon Steeghs



Inhoud

[1. Aanleiding 3](#_Toc138163274)

[2. Hoofdvraag: welke sorteer algoritmes zijn er en hoe kan je ze het best toepassen? 4](#_Toc138163275)

[2.1. Welke sorteer algoritmes zijn er? 4](#_Toc138163276)

[2.2. Hoe werkt het bubble sorteeralgoritme ? 4](#_Toc138163277)

[2.3. Hoe werkt het merge sorteeralgoritme? 5](#_Toc138163278)

[2.4. Hoe werkt het selection sorteeralgoritme? 5](#_Toc138163279)

[2.5. Wat zijn de verschillen tussen de onderzochte sorteeralgoritmes, welke is het snelst? 6](#_Toc138163280)

[2.5.1. Metingen 6](#_Toc138163281)

[2.5.2. Conclusie 7](#_Toc138163282)

[3. Conslusie 8](#_Toc138163283)

[4. Reflectie - Deel 2 & 4 9](#_Toc138163284)

[4.1. Deel 2 9](#_Toc138163285)

[4.2. Deel 4 10](#_Toc138163286)

[5. Bronvermelding 11](#_Toc138163287)

# Aanleiding

Om beter te bergrijpen wat voor sorteer algoritmes er bestaan, maak ik verschillende opdrachten die hier betrekking tot hebben. Ik ga hierbij kijken naar drie verschillende sorteer algoritmes om hun verschillen goed in kaart te brengen. Door drie algoritmes te onderzoeken kan ik voor een specifieke context het juiste algoritme uitkiezen.

# Hoofdvraag: welke sorteer algoritmes zijn er en hoe kan je ze het best 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 verschillende 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.

# Conslusie

Ik begon dit onderzoek met de vraag: Welke sorteeralgoritmes zijn er en hoe kan je zet het best toepassen? Ik heb in dit onderzoek drie algoritmes onderzocht: het bubble sorteeralgoritme, het merge sorteeralgoritme en het selection sorteeralgoritme.

Het bubble sorteeralgoritme vergelijkt steeds aangrenzende elementen en wisselt ze indien nodig om de array in oplopende volgorde te sorteren. Dit algoritme is eenvoudig maar niet efficiënt voor grote datasets, vanwege zijn tijdscomplexiteit van O(N^2).

Het merge sorteeralgoritme splitst de array op in kleinere arrays, sorteert deze afzonderlijk en voegt ze vervolgens samen om een gesorteerde array te verkrijgen. Dit algoritme is effectief voor het sorteren van grote hoeveelheden data en heeft een tijdscomplexiteit van O(n log n) in zijn meest slechte gevallen.

Het selection sorteeralgoritme selecteert steeds het kleinste element in de array en plaatst het vooraan in het ongesorteerde deel van de array. Hoewel dit algoritme eenvoudig te implementeren is, heeft het ook een tijdscomplexiteit van O(N^2), vergelijkbaar met het bubble sorteeralgoritme.

Uit de metingen blijkt dat het merge sorteeralgoritme over het algemeen het meest geschikt is voor grote datasets, omdat het minder swaps en comparisons vereist. Voor kleinere datasets kunnen de bubble en selection sorts echter efficiënter zijn in bepaalde gevallen.

Het is belangrijk op te merken dat de daadwerkelijke uitvoeringstijd van de algoritmes ook afhankelijk is van andere factoren, zoals de specificaties van de computer. Daarom kan de gemeten tijd niet als een nauwkeurige indicator worden beschouwd om het meest efficiënte algoritme te bepalen.

Kortom, bieden de verschillende sorteeralgoritmes verschillende voordelen en nadelen, en hun toepassing hangt af van de specifieke context en de grootte van de dataset die gesorteerd moet worden. Het is essentieel om de eigenschappen en complexiteit van elk algoritme te begrijpen om een weloverwogen keuze te maken bij het selecteren van het meest geschikte sorteeralgoritme.

# Reflectie - Deel 2 & 4

## Deel 2

In algoritme part 2 heb ik een Memory Manager geimplementeerd met behulp van linked lists Tijdens het uitvoeren van deze opdracht heb ik veel geleerd over linked lists en hun toepassingen. Ik kreeg een beter begrip van hoe linked lists werken en hoe ik ze kan gebruiken om dynamisch geheugenbeheer mogelijk te maken.

Het eerste deel van de opdracht ging over het implementeren van een proof of concept van een linked list. Ik moest de functionaliteit van het toevoegen en verwijderen van elementen in de linked list implementeren. Dit hielp me met het leren van de basic principes van het werken met linked lists.

Vervolgens moest ik de linked list module verbeteren om deze algemeen te maken, zodat deze meerdere lijsten kan ondersteunen. Dit vereiste het ontwerpen van een goed gestructureerde module, waarbij de linked list geen kennis had van het specifieke datatype dat werd opgeslagen en de hogere niveaus van de code, zoals de Memory Manager, geen kennis hadden van de werking van de linked list.

Een ander belangrijk aspect dat ik heb geleerd tijdens het uitvoeren van deze opdracht, is het belang van het vermijden van memory leaks. Ik moest ervoor zorgen dat ik de gealloceerde geheugenblokken op de juiste manier vrijgaf en geen ongebruikte geheugenruimte achterliet. Door gebruik te maken van Valgrind kon ik mijn implementatie testen en controleren op memory leaks.

Al met al was deze opdracht een waardevolle leerervaring. Ik heb mijn kennis over linked lists vergroot en begrijp nu beter hoe ik ze kan toepassen in situaties waar dynamisch geheugenbeheer vereist is. Daarnaast heb ik geleerd hoe ik aandacht kan besteden aan het voorkomen van memory leaks, wat essentieel is voor het ontwikkelen van betrouwbare software. Ik ben tevreden met mijn voortgang en in de toekomst denk ik dat dit zeer waardevolle kennis gaat zijn.

## Deel 4

Voor algoritme 4 heb ik veel geleerd over verschillende zoekalgoritmes, met name de Breadth-First Search (BFS) en Depth-First Search (DFS). Deze algoritmes waren essentieel bij het maken van de twee delen.

In het eerste deel moest ik de kortste route vinden van knooppunt 1 naar knooppunt N in een netwerk van N knooppunten. Om dit te bereiken, heb ik het BFS-algoritme toegepast. Door het implementeren van BFS heb ik beter inzicht gekregen in het werken met zoekalgoritmes.

Voor het tweede deel moest ik de diepte van een tree met N elementen bepalen. Hier heb ik gebruik gemaakt van het DFS-algoritme. Met behulp van recursie kon ik de diepte van de boom berekenen door de diepte van de verschillende onderdelen van de tree te vergelijken. Het toepassen van DFS heeft me geholpen om vertrouwd te raken met het concept van recursie en hoe ik het kan gebruiken om complexe taken op te lossen.

Door het implementeren van zowel BFS als DFS heb ik een beter begrip gekregen van de werking en toepassing van verschillende zoekalgoritmes. Ik heb geleerd hoe ik graphs en trees kan doorzoeken, hoe ik efficiënte routes kan vinden en hoe ik de diepte van een tree kan bepalen. Daarnaast heb ik mijn programmeervaardigheden verbeterd door het toepassen van recursie in de tweede opdracht.

Ik ben tevreden met mijn prestaties in deze opdracht en met de progressie die ik heb gemaakt wat betreft mijn kennis van dit onderwerp. Ik ben ervan overtuigd dat de kennis en vaardigheden die ik heb opgedaan bij het implementeren van deze algoritmes van grote waarde zullen zijn bij het aanpakken van toekomstige programmeeruitdagingen.

# Bronvermelding

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

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

Edureka. (2020). Everything You Need To Know About Sorting Algorithms In C. *Edureka*. <https://www.edureka.co/blog/sorting-algorithms-in-c/>