Sorteer algoritmes

Welke van de volgende sorteeralgoritmes: Merge, Quick, Selection, Insertion en Bubble kan het beste gebruikt worden op verzamelingen met numerieke waardes?

Auteur:	Jesse van Bree
Studentnummer:	500801418
Opdrachtgever:	A.C.W. Suidman
Datum:	17-juni-2019

Inhoudsopgave

Samenvatting

De gemiddelde programmeur denkt niet lang genoeg na over het gebruik van sorteeralgoritmes. Zij gebruiken vaak de populairste variant die makkelijk te implementeren is. Als student Software engineer heb ik de mogelijkheid gekregen om sorteeralgoritmes te mogen onderzoeken.

Het doel van dit onderzoek is om uit te zoeken welk van de 5 gekozen sorteeralgoritmes een programmeer het beste kan gebruiken. De hoofdvraag van dit onderzoek is: “Welke van de volgende sorteeralgoritmes: Merge, Quick, Selection, Insertion en Bubble kan het beste gebruikt worden op verzamelingen met numerieke waardes?”.

Een sorteeralgoritme is een algoritme die een verzameling met data sorteert op een volgorde, die variëert per implementatie. Het sorteeralgoritme is geschreven aan de hand van een ontwerp. Hiernaast zijn er eigenschappen die onderscheid creëren tussen de verschillende sorteeralgoritmes.

Als er gekeken wordt naar de implementaties van de sorteeralgoritmes in Java. Is Bubble het meest simpele sorteeralgoritme om te gebruiken. De code van Bubble is niet complex en is best weinig. Hierna zijn Selection en Insertion wat complexer om te implementeren. Waarna Merge en Quick het meest complex zijn.

Stabiliteit in de context van sorteeralgoritmes betekent: De originele volgorde wordt bewaard bij het sorteren van gelijke elementen. Bubble, Insertion en Merge zijn stabiel en Quick en Selection niet.

Als het gaat om snelheid is Bubble snel bij kleine verzamelingen. Maar net zoals bij Selection en Insertion worden deze veel langzamer als, er grotere verzamelingen gesorteerd moeten worden. Quick is het snelste algoritme van de 5 en net zoals bij Merge zijn deze niet veel langzamer bij grotere verzamelingen.

Om te bepalen welk algoritme het snelst is. Moet er gekeken worden naar waarvoor het algoritme gebruikt moet worden. Als het algoritme snel gemaakt moet worden en het aantal waardes in de verzameling niet veel is. Dan is Bubble de beste optie. Zou de verzameling heel groot zijn dan zijn Quick en Merge een goede optie. Hier moet wel besloten worden of stabiliteit belangrijk is. Zoja dan is Merge de beste optie. Zo niet dan is Quick de betere keuze.

Inleiding

Als student Software Engineer heb ik de 2 stageperiodes in mijn vooropleiding kunnen vervullen bij het bedrijf NoNoA, in Heiloo. Het bedrijf ontwikkeld projectmanagement software, waarin ik de opdracht had gekregen om een aantal verschillende schermen te maken. In projecten worden veel verschillende soorten data gebruikt, waarin de meest voorkomende soorten tekstuele en numeriek zijn. Nu is het nut maken van deze data belangrijk en één van de aspecten van het nut maken van data is sorteren. Omdat ik dit aspect interessant vind. Heb ik gekozen voor sorteeralgoritmes als onderwerp voor dit onderzoeksrapport.

De hoofdvraag van dit onderzoek luidt
Welke van de volgende sorteeralgoritmes: Merge, Quick, Selection, Insertion en Bubble kan het beste gebruikt worden op verzamelingen met numerieke waardes?

Wat zorgt ervoor dat het ene sorteeralgoritme beter is dan de ander en wat zorgt voor de variatie in de algoritmes.

Het doel van dit onderzoeksrapport is, om vast te stellen welk van de 5 populaire sorteeralgoritmes. Een programmeur zou het best zou kunnen gebruiken voor verzamelingen met numerieke waardes. Hiernaast zullen ook de verschillen tussen de algoritmes in kaart worden gebracht.

Om de hoofdvraag correct te kunnen beantwoorden, zullen de volgende deelvragen beantwoord moeten worden:

1. Wat is een sorteeralgoritme?
2. In hoeverre verschilt de snelheid van de sorteeralgoritmes bij variërende groottes data?
3. Hoe complex zijn de implementaties van de sorteeralgoritmes in de programmeertaal Java?
4. Wat betekent stabiliteit in de context van sorteeralgoritmes, en zijn de 5 sorteeralgoritmes ook stabiel?

Bij mijn onderzoek maak ik gebruik van rapportages, informatieve websites en data uit eerdere onderzoeken. Deze bronnen zijn te vinden aan het einde van dit onderzoeksrapport.

Dit rapport is primair geschreven voor software engineers. Die zich meer willen verdiepen in de 5 populaire sorteeralgoritmes.

De opbouw van dit rapport is als volgt:
In hoofdstuk 1 wordt gedefinieerd wat een sorteeralgoritme is. In hoofdstuk 2 wordt gekeken naar de implementaties van de algoritmes in de programmeertaal Java. Vervolgens in hoofdstuk 3 zal er gekeken worden naar de stabiliteit van de 5 populaire sorteeralgoritmes.

In hoofdstuk 4 wordt er gekeken naar de snelheid van de 5 populaire sorteeralgoritmes bij variërende groottes data. Tot slot zal in de conclusie de hoofdvraag beantwoord worden.

[1] Wat is een sorteeralgoritme?

Wat is een sorteeralgoritme eigenlijk? Een sorteeralgoritme is een algoritme wat een verzameling met data sorteert op een volgorde, die variëert per implementatie. Deze volgorde is bijna altijd numeriek of alfabetisch en is vaak van laag naar hoog. Een sorteeralgoritme wordt niet alleen gebruikt in het programmeren maar ook in algebra. In dit rapport wordt er alleen gefocust op sorteeralgoritmes bij het programmeren. Hierin zijn er veel verschillende uitwerkingen van sorteeralgoritmes. Elk algoritme lost het probleem van sorteren net iets anders op. Wel komt ieder algoritme op een gesorteerde verzameling uit. (Donald E. Knuth, 1998)

[1.1] Ontwerp

Ieder algoritme wat een probleem moet oplossen volgt een ontwerp. Dit ontwerp wordt algoritme ontwerp paradigma genoemd. Een sorteeralgoritme is hier geen uitzondering. Een aantal belangrijke ontwerpen die dit rapport van pas komen zijn:

- Divide and conquer
- Binary search
- Min-Max

Bij divide and conquer algoritmes wordt het probleem opgelost, door de verzameling op te splitsen in kleinere verzamelingen. Het breekpunt waar de verzameling gesplitst wordt staat niet vast. Hier kan gekozen worden voor het eerste element, het laatste element, de mediaan van de verzameling of het centerpunt. Dit verschilt per implementatie. Deze kleinere verzamelingen worden apart gesorteerd. Wanneer alle kleinere verzamelingen gesorteerd zijn. Voegt het algoritme de kleinere verzamelingen samen. Hier ligt het aan de implementatie of het algoritme de totale verzameling nog moet sorteren, of dat het algoritme rekening houdt met de inhoud van de kleinere verzamelingen en deze gelijk goed samenvoegt.

Bij binary search worden er meerdere iteraties gedaan door de meegegeven verzameling. Tijdens deze iteraties wordt er gezocht naar twee elementen. Hierna wordt bepaald of deze omgewisseld moeten worden. Bij dit ontwerp kan het voorkomen dat er een tweede verzameling wordt aangemaakt. Deze tweede verzameling wordt gebruikt als resultaat en is dan een gesorteerde verzameling met de elementen van de eerste verzameling.

Min-Max algoritmes kunnen geïmplementeerd worden in combinatie met andere ontwerpen. Er zijn algoritmes die het min-max ontwerp volgen. Maar tegelijkertijd ook een divide and conquer of binary search algoritme zijn. Wat Min-Max algoritmes uniek maakt is, dat zij de verzameling sorteren. Op basis van de laagste en/of hoogste waarde in de verzameling. Hoe de algoritmes uitkomen met een gesorteerd resultaat verschillend per implementatie. (Tutorialspoint, N.D.), (GeeksforGeeks, 2019)

[1.2] Eigenschappen

Sorteer algoritmes zijn te onderscheiden aan de hand van hun specifieke eigenschappen. De belangrijkste eigenschappen zijn:

- Snelheid
- Geheugengebruik
- Omgang slechtste scenario's
- Stabiliteit
- Implementatie

Hieronder zal in het kort uitgelicht worden wat deze eigenschappen inhouden en hoe deze beïnvloed worden. (Gautam Ghosh, 2016 & Donald E. Knuth, 1998)

Bij snelheid wordt gekeken naar hoe snel een algoritme een lijst kan sorteren. De twee aspecten die het meeste invloed hebben op de snelheid zijn: de grote van de data en het soort data. Hier is te denken aan soorten zoals nummers, tekst of datums. Meer informatie hierover in het hoofdstuk "Snelheid".

Bij het geheugengebruik wordt gemeten hoeveel systeemgeheugen het algoritme ineemt om zijn berekeningen te maken. Sommige algoritmes maken een tweede lijst aan bij het sorteren. Wat als resultaat heeft dat het algoritme meer geheugen gebruikt. Ook heeft hier de grote van de data invloed.

Soms kan het voorkomen dat een lijst al gesorteerd is of dat de lijst in de verkeerde volgorde gesorteerd staat. Dit zijn een paar slechte scenario's waar een algoritme rekening mee moet houden. Sommige algoritmes kunnen dit beter dan de ander.

In de context van sorteeralgoritmes betekent stabiliteit. Dat er rekening gehouden wordt met de originele volgorde van gelijke elementen. Het kan voorkomen dat er duplicaten elementen zijn in de originele lijst. Hier houden sommige sorteeralgoritmes rekening mee en sommige niet. Meer informatie hierover in het hoofdstuk "Stabiliteit"

Tot slot is implementatie heel belangrijk. Er is een duidelijk verschil in de implementatie van sorteeralgoritmes. Het verschil ligt in de complexiteit en lengte van de code die geschreven moet worden. Ook heeft hier de programmeertaal invloed op deze twee aspecten. Meer informatie hierover in het hoofdstuk "Implementatie".

(GeeksforGeeks, 2019), (Gautam Ghosh, 2016), (Donald E. Knuth, 1998)

[2] Werking en implementatie in Java

De uiteindelijke implementatie van een sorteeralgoritme verschilt per programmeertaal. Omdat er volgens (Diarmuid P, 2011) duizenden programmeertalen zijn en dit niet past in 1 rapport. Wordt er in dit rapport alleen gefocust op de programmeertaal Java. In dit hoofdstuk zal er gekeken worden naar hoe de 5 gekozen algoritmes werken. En hoe deze geïmplementeerd kunnen worden in Java.

De volgende termen zullen in dit hoofdstuk tevoorschijn komen:

1. Methode: een stuk code met een invoer en een resultaat
2. Array: een verzameling van data
3. Variabele: Een waarde die gekoppelt is aan een naam. Hiermee kunnen bepaalde waardes tijdelijk opgeslagen worden. Totdat het programma sluit.
4. Recursie: Een methode die zichzelf aanroept. Om hiermee een probleem op te lossen door het te verdelen in kleine stukken.

(GeeksforGeeks, 2019)

[2.1] Bubble

Het sorteeralgoritme Bubble is één van de simpelste sorteeralgoritmes. Dit is omdat het algoritme de tussentijdse status van de verzameling niet onthoud. Het algoritme loopt meerdere keren door de verzameling heen en doet dit totdat hij geen elementen meer kan omdraaien. Het kan dus voorkomen dat de verzameling naar een aantal iteraties al gesorteerd is, maar omdat het algoritme dit niet weet. Moet er nog een laatste iteratie gemaakt worden. Om er zeker van te zijn dat de verzameling gesorteerd is. Wel is het aantal code wat geschreven moet worden niet veel en ook niet complex. Wat Bubble populair maakt bij niet zo ervaren programmeurs. (Owen Astrachan, 2003), (GeeksforGeeks, 2019)

[2.1.1] Implementatie Java

```
void bubbleSort(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            // Kijk of de waarde arr[j] groter is dan de opvolgende waarde
            if (arr[j] > arr[j+1])
            {
                // Wissel arr[j+1] en arr[i] om
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

Hierboven is de implementatie te zien van Bubble in Java. Er wordt een array meegegeven aan de methode bubbleSort. De methode stelt eerst vast hoeveel elementen er in de array

zitten. Hierna worden er 2 opeenvolgende iteraties gedaan, waarin er gekeken wordt of het huidige element wat bekeken wordt. Groter is dan het volgende element. Zoja dan worden deze 2 elementen omgewisseld. Zoniet dan gaat de methode naar de volgende iteratie totdat de array gesorteerd is. (Owen Astrachan, 2013)

[2.2] Quick

Het sorteeralgoritme Quick staat bekend om zijn snelheid. Het algoritme volgt het Min-Max ontwerp en is hiernaast ook een divide and conquer algoritme. Zoals besproken in hoofdstuk 1. Splitst dit algoritme de verzameling op in kleinere stukken. Om deze apart weer op te lossen. De code van het algoritme Quick is complexer dan bij Bubble, omdat Quick werkt met recursie.

(Charles Knessl & Wojciech Szpankowski, 1999), (GeeksforGeeks, 2019)

[2.2.1] Implementatie Java

```java
/* De hoofdmethode die QuickSort() implementeerd
arr[] --> Die gesorteerd moet worden,
low --> Index waar het sorteren begint,
high --> Index van laatste element*/
void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
        now at right place */
        // pi is de index
        int pi = partition(arr, low, high);

        // Sorteer recursief de eerste en tweede array
        // voor en na de partitie
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```

```java
/* Deze functie neemt het laatste element als breekpunt
 * en plaatst het breekpunt element op de juiste plek in
 * de gesorteerde array. Alle kleinere waardes worden
 * naar links verplaatst en de grotere waardes naar rechts
 op basis van het breekpunt*/
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // positie van het kleinere element
    for (int j=low; j<high; j++)
    {
        // Als het huidige element kleiner of gelijk is aan het breekpunt
        if (arr[j] <= pivot)
        {
            i++;

            // Wissel de waarde van arr[i] om met arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Wissel de waarde van arr[i+1] om
    // met de waarde van arr[high] (of het breekpunt)
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
}
```

Hierboven is de implementatie te zien van Quick in Java. Bij de implementatie van Quick zijn er 2 verschillende methodes. De methode sort implementeert het algoritme en de methode partition verplaatst de elementen naar hun juiste plek. De methode sort roept zichzelf recursief aan. Nadat alle kleine arrays zijn gesorteerd worden deze samengevoegd. Hierna geeft de methode een gesorteerde array terug. (GeeksforGeeks, 2019)

[2.3] Selection

Selection is een Min-Max sorteeralgoritme. Het algoritme itereert door de meegegeven array heen en zoekt vervolgens bij iedere iteratie de laagste waarde. Deze laagste waarde zet hij aan het begin van de ongesorteerde array. Wat speciaal is aan Selection is, dat het algoritme 2 arrays bijhoudt in 1 array. eén hiervan wordt gevuld met gesorteerde data. De tweede zijn de overige elementen die nog gesorteerd moeten worden. Hoe het algoritme dit doet is redelijk simpel. In de methode van Selection wordt een grens bijgehouden die bij iedere iteratie verplaatst. Totdat de tweede array met overige elementen leeg is. Hierdoor is in de code 1 array. Dit bespaart geheugen tegenover meerdere losse arrays in de code hebben zitten. (GeeksforGeeks, 2019)

[2.3.1] Implementatie Java

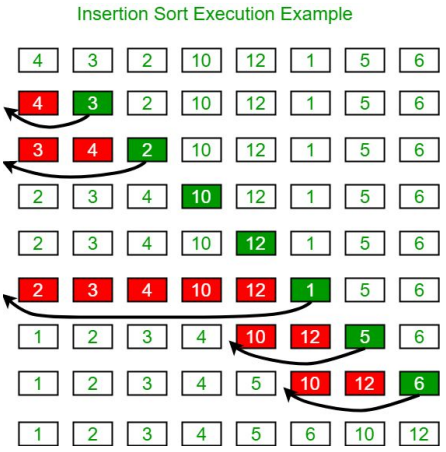
```java
void sort(int arr[])
{
    int n = arr.length;

    // Verplaats de grens van de 2 arrays eem voor een
    for (int i = 0; i < n-1; i++)
    {
        // vind het laagste element in de ongesorteerde array
        int min_idx = i;
        for (int j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Wissel het gevonden element om  met het eerste element
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}
```

Hierboven is de implementatie te zien van Selection in Java. Zoals bij de andere sorteeralgoritmes, wordt er een array meegegeven. Hiervan wordt de lengte opgeslagen in de variabele n. Vervolgens wordt de grens bepaald in de bovenste iteratie. Hierna wordt de laagste waarde opgezocht en wordt deze in op de juiste plaats gezet. (GeeksforGeeks, 2019)

[2.4] Insertion



Insertion sorteert een array op een manier zoals mensen dat ook doen. Insertion begint altijd bij de 2de waarde in de array. Vervolgens kijkt Insertion of de vorige waarde in de array groter is dan de 2de waarde. Zo ja dan kijk Insertion of dit ook geldt voor de waarde hiervoor. Dit doet het algoritme totdat de juiste plek is gevonden. Hierna zet het algoritme de waarde op de goed plek. Stel dat de 2 waardes goed staan slaat hij deze over. Insertion herhaalt deze stappen totdat het algoritme bij het einde komt. Hierdoor is insertion redelijk efficiënt in kleinere arrays en ook simpel om te implementeren. (GeeksforGeeks, 2019)

[2.4.1] Implementatie Java

```java
void sort(int arr[])
{
    int n = arr.length;
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;

        // Verplaats de elementen van arr[0..i-1]
        // die groter zijn dan de variabele key
        // 1 positie hoger dan hun huidige positie
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Hierboven is de implementatie te zien van Insertion in Java. In deze methode sort wordt een array meegegeven. Eerst wordt in de variabele “n” de lengte van de array opgeslagen. Hierna worden er meerdere iteraties gedaan. Tijdens de iteraties wordt er gekeken of het vorige element groter is dan het element waar de methode nu mee bezig is. En wisselt de elementen waar nodig is. (GeeksforGeeks, 2019)

[2.5] Merge

Net zoals bij het algoritme Quick is Merge een divide and conquer algoritme. En lost in plaats van meerdere iteraties te maken. Wordt het probleem recursief opgelost. (GeeksforGeeks, 2019)

[2.5.1] Implementatie Java

```
// Hoofd methode die arr[l..r] sorteert
// met de methode merge
void sort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Zoek het middelste punt in de array
        int m = (l+r)/2;

        // sorteer de de 2 halven
        sort(arr, l, m);
        sort(arr , m+1, r);

        // voeg de 2 halve arrays samen
        merge(arr, l, m, r);
    }
}
```

In de afbeelding hierboven en hiernaast is de implementatie te zien van Merge in Java. Net zoals bij Quick is dit algoritme opgesplitst in 2 methodes. In de methode sort wordt deze recursief aangeroepen. Met als doel om de array op te splitsen in kleinere stukken en deze los te sorteren. Aan het einde wordt de methode merge aangeroepen om de 2 halve arrays weer samen te voegen. Deze methode gaat er van uit dat de halve arrays gesorteerd zijn. (GeeksforGeeks, 2019)

```
// Voegt de tweehalve arrays van arr[].
// Eerste half is arr[l..m]
// Tweede half is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    // Sla de 2 lengtes op van de halve arrays
    int n1 = m - l + 1;
    int n2 = r - m;

    /* creer tijdelijke arrays*/
    int L[] = new int [n1];
    int R[] = new int [n2];

    /*Kopieer de halve arrays in de tijdelijke arrays */
    for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
    for (int j=0; j<n2; ++j)
        R[j] = arr[m + 1+ j];


    /* Voeg de 2 halve arrays samen*/

    // Eerste index van de 2 halve arrays
    int i = 0, j = 0;

    // Eerste index van samengevoegde array
    int k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Kopieer de overgebleven elementen als deze er zijn van L[] */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Kopieer de overgebleven elementen als deze er zijn van R[] */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

[3] Stabiliteit

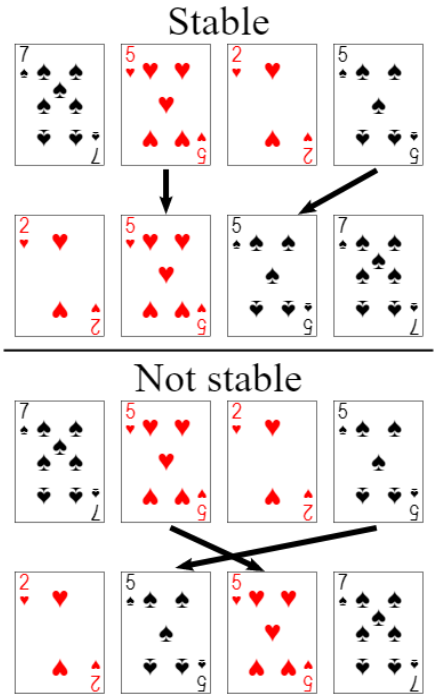
Stabiliteit kan heel belangrijk zijn bij het sorteren van verzamelingen. Maar wat betekent stabiliteit nou? En hoezo is dit belangrijk?

[3.1] Betekenis

Stabiliteit in de context van sorteeralgoritmes betekent: Het behouden van de originele volgorde van gelijke elementen in de gesorteerde verzameling.

[3.1.1] Voorbeeld

Als voorbeeld nemen we de afbeelding rechts. Er is een verzameling met 4 kaarten. In deze verzameling zit een zwarte 7, rode 5, rode 2 en een zwarte 5 in deze volgorde. De verzameling moet gesorteerd worden op volgorde van de nummers. Stel als het sorteeralgoritme dat gebruikt wordt niet stabiel is. Wat zou kunnen voorkomen is dat in de gesorteerde verzameling de zwarte 5 voor de rode 5 komt te staan. Stel het algoritme is welk stabiel dan zou de rode 5 altijd voor de zwarte 5 komen.



(Jehad Alnihoud & Rami Mansi, 2008)

[3.2] Vergelijking

Sorteeralgoritme	**Stabiel**
Bubble	Ja
Quick	Nee
Selection	Nee
Insertion	Ja
Merge	Ja

Hoe zit het nu met de 5 populaire algoritmes? Zoals te zien in de tabel links. Zijn Bubble, Insertion en Merge stabiel. Hier zijn Quick en Selection daarin tegen niet stabiel. Wel zijn er varianten van de algoritmes Quick en Selection beschikbaar die wel stabiel zijn. Deze variaties zijn in hun kern gelijk aan het origineel maar in praktijk zijn ze anders genoeg om een nieuw algoritme te zijn. Wat de reden is waarom deze variaties niet in dit rapport worden opgenomen. Een voorbeeld voor zo'n variatie is: "Enhanced Quicksort". (Jehad Alnihoud & Rami Mansi, 2008), (GeeksforGeeks, 2019), (Rami Mansi, 2008)

[4] Snelheid

[4.1] Methodiek

Om te bepalen welk sorteeralgoritme het snelste numerieke waardes kan sorteren. Zullen de resultaten uit de volgende onderzoeken gebruikt worden.

1. (K. Al-Kharabsheh & I. AlTurani & A. AlTurani & N. Zanoon, 2013)
2. (Jehad Alnihoud & Rami Mansi, 2008)[2]
3. (Owen Astrachan, 2013)
4. (Yingxu Wang, 1996)

[4.2] Vergelijking

	Sorteeralgoritme **Snelheid**				
Aantal elementen	**Bubble**	**Quick**	**Selection**	**Insertion**	**Merge**
10000 elementen[1]	1133 ms	489 ms	2227 ms	1605 ms	728 ms
20000 elementen[1]	3103 ms	1084 ms	5058 ms	3678 ms	1509 ms
30000 elementen[1]	5730 ms	1648 ms	8254 ms	6125 ms	2272 ms
12500 elementen[2]	506 ms	N.D.	346 ms	N.D.	N.D.
10000 elementen[3]	16800 ms	N.D.	11500 ms	5400 ms	N.D.
100 elementen[4]	2555 ms	525 ms	N.D.	N.D.	N.D.
1000 elementen[4]	10466 ms	1154 ms	N.D.	N.D.	N.D.

Belangrijk: De oorzaak van de variatie in tijden bij gelijke elementen. Komt door de verschillen in hardware bij de verschillende onderzoeken. Dit heeft geen grote invloed op de conclusie, omdat deze waardes nog steeds relatief per onderzoek vergeleken kunnen worden.

[1234]: Verwijzen naar lijst met onderzoeken in [4.1]

Zoals te zien in de tabel hierboven. Is Quick in ieder onderzoek waar dit algoritme getest is het snelste. Hier is ook te zien dat Bubble redelijk snel is in het sorteren van kleinere verzamelingen. Wel is te zien bij de algoritmes Selection, Insertion en Bubble deze erg langzaam worden bij grotere verzamelingen. Merge daarentegen blijft net zo consistent snel als Quick bij de variërende groottes van de verzamelingen.

(K. Al-Kharabsheh & I. AlTurani & A. AlTurani & N. Zanoon, 2013), (Jehad Alnihoud & Rami Mansi, 2008), (Owen Astrachan, 2013), (Yingxu Wang, 1996)

Conclusie

In dit rapport is gezocht naar het antwoord op de onderzoeksvraag: “*Welke van de volgende sorteeralgoritmes: Merge, Quick, Selection, Insertion en Bubble kan het beste gebruikt worden op verzamelingen met numerieke waardes?”.*

Zoals besproken in hoofdstuk 1 is een sorteeralgoritme: Een algoritme die een verzameling met data sorteert op een volgorde die kan variëren per implementatie. Het sorteeralgoritme is geschreven aan de hand van een ontwerp. Hiernaast zijn er eigenschappen die onderscheid creëert tussen de verschillende sorteeralgoritmes.

In hoofdstuk 2 is er gekeken naar de implementaties van de 5 algoritmes. Hier is uitgekomen dat de divide and conquer algoritmes Quick en Merge complexer zijn om op te zetten, dan de algoritmes Selection, Insertion en Bubble. Van deze 3 is bubble het simpelste om te gebruiken.

In hoofdstuk 3 is besproken dat stabiliteit in sorteeralgoritmes betekent, dat de originele volgorde wordt bewaard bij het sorteren van gelijke elementen. Hier was uitgekomen dat Bubble, Insertion en Merge stabiel zijn en Quick en Selection niet stabiel zijn.

In hoofdstuk 4 zijn de snelheden in kaart gebracht van de 5 algoritmes. Hier werd duidelijk dat bubble snel is met kleinere verzamelingen. Maar net zoals bij Selection en Insertion deze veel langzamer worden bij grotere verzamelingen. Hiernaast was Quick bij iedere test het snelste en werd dit algoritme net, zoals bij Merge niet veel langzamer bij grotere verzamelingen.

Om te bepalen welk algoritme het snelst is. Moet er gekeken worden naar waarvoor het algoritme gebruikt moet worden. Als het algoritme snel gemaakt moet worden en het aantal waardes in de verzameling niet veel is. Dan is Bubble de beste optie. Zou de verzameling heel groot zijn dan zijn Quick en Merge een goede optie. Hier moet wel besloten worden of stabiliteit belangrijk is. Zoja dan is Merge de beste optie. Zo niet dan is Quick de betere keuze.

Bronvermelding

Donald E. Knuth, *The art of computer programming*, volume 3: Sorting and searching. Second edition (Reading, Massachusetts: Addison-Wesley, 1998), ISBN 0-201-89685-0

GeeksforGeeks, (2019), Sorting algorithms, Opgehaald uit: https://www.geeksforgeeks.org/sorting-algorithms/

GeeksforGeeks, (2019), Selection sort, Opgehaald uit: https://www.geeksforgeeks.org/selection-sort/

GeeksforGeeks, (2019), Bubble sort, Opgehaald uit: https://www.geeksforgeeks.org/bubble-sort/

GeeksforGeeks, (2019), Insertion sort, Opgehaald uit: https://www.geeksforgeeks.org/insertion-sort/

GeeksforGeeks, (2019), Merge sort, Opgehaald uit: https://www.geeksforgeeks.org/merge-sort/

GeeksforGeeks, (2019), Quick sort, Opgehaald uit: https://www.geeksforgeeks.org/quick-sort/

Gautam Ghosh, (2016), Sorting Techniques in Data Structures, Opgehaald uit: https://www.w3schools.in/data-structures-tutorial/sorting-techniques/

Bentley, Jon (2000), Programming Pearls, ACM Press/Addison–Wesley, pp. 116, 121

Black, Paul E. (24 August 2009). "bubble sort". Dictionary of Algorithms and Data Structures. National Institute of Standards and Technology. Retrieved 1 October 2014

Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison–Wesley, 1997. ISBN 0-201-89685-0. Pages 138–141 of Section 5.2.3: Sorting by Selection.

Anany Levitin. Introduction to the Design & Analysis of Algorithms, 2nd Edition. ISBN 0-321-35828-7. Section 3.1: Selection Sort, pp 98–100.

Owen Astrachan, (2003), Bubble Sort: An Archaeological Algorithmic Analysis, http://cygnus-x1.cs.duke.edu/courses/cps182s/compsci342s/cps182s/fall03/latex/checkout/bubble182.pdf

Tutorialspoint, (N.D.), Design and analysis of algorithms, opgehaald uit: https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_divide_conquer.htm

Jehad Alnihoud and Rami Mansi, (2008), An Enhancement of Major Sorting Algorithms, Opgehaald uit: http://ccis2k.org/iajit/PDF/vol.7,no.1/9.pdf

K. Al-Kharabsheh & I. AlTurani & A. AlTurani & N. Zanoon, (2013), Review on Sorting Algorithms A Comparative Study, Opgehaald uit: https://www.researchgate.net/publication/259911982_Review_on_Sorting_Algorithms_A_Comparative_Study

Rami Mansi, (2008), Enhanced Quicksort Algorithm, Opgehaald uit http://iajit.org/PDF/vol.7%2Cno.2/740.pdf

Yingxu Wang, (1996), A New Sort Algorithm: Self-Indexed Sort, Opgehaald uit: https://www.researchgate.net/profile/Yingxu_Wang/publication/220178394_A_New_Sort_Algorithm_Self-Indexed_Sort/links/545c72ed0cf249070a7aa1e9.pdf