Opgave 1: Tijdscomplexiteit van algoritmen

Wat wordt er berekend?

Het doel van deze opdrachten is het verschil tussen twee algoritmen te bepalen. De beide algoritmen worden een x aantal keren opgeroepen. Wanneer het programma door het algoritme gaat gaan we kijken hoelang het programma er heeft over gedaan om zijn taak te vervullen. Door dit meerdere malen te doen kunnen we nagaan hoe lang het algoritme er gemiddeld over doet. We kunnen hierdoor zien hoe efficiënt het algoritme werkt. Nu we weten hoe efficiënt het ene algoritme werkt kunnen we het ook vergelijken met anderen.

De algoritmen bekijkt naar een array van willekeurige getallen. Hij gaat elke plaats van de array af en neemt de som van alle vorige getallen uit deze array. Deze som zet hij in een andere array A[i]. Dit gebeurt zoveel keer als de grootte van de array met willekeurige getallen.

Tijdscomplexiteit van de algoritmen

```
package com.company;

public class PrefixAverage1 {

    private int[] numbers;

    public PrefixAverage1(int[] numbers) { this.numbers = numbers; }

    public int[] run() {

        int[] A = new int[this.numbers.length]; //1

        for (int i = 0; i < this.numbers.length; i++) { //2

            int a = 0; //3

            for (int j = 0; j < i; j++) { //4

                a = a + numbers[j]; //5
            }

            A[i] = a / (i + 1); //6
        }
        return A;
    }

}
```

Bij het eerste algoritme berekenen we:

- lijn 1: kost toekenning = O(1)

- lijn 2 en 4: kost ingenestelde lus

O(n)·n =O(n²)

- lijn 3: kost toekenning = O(1)

- lijn 5: kost toekenning = O(1)

- lijn 6: kost toekenning = O(1)

Big O = O(1)+O(n²)+O(1) +O(1)+ O(1)

=O(n²)

```java
package com.company;

public class PrefixAverage2 {

    private int[] numbers;

    public PrefixAverage2(int[] numbers) { this.numbers = numbers; }

    public int[] run() {

        int[] A = new int[this.numbers.length]; //1

        int s=0; //2

        for (int i=0;i<this.numbers.length;i++) //3
        {
            s = s + numbers[i]; //4

            A[i] = s / (i + 1); //5
        }
        return A;
    }
}
```

Bij het tweede algoritme zijn we dat er geen genestelde for loop voorkomt. Dit zal al voor een beter resultaat zorgen.

-lijn 1: kost toekenning = O(1)

-lijn 2: kost toekenning = O(1)

-lijn 3: kost lus = O(n)

-lijn 4: kost toekenning = O(1)

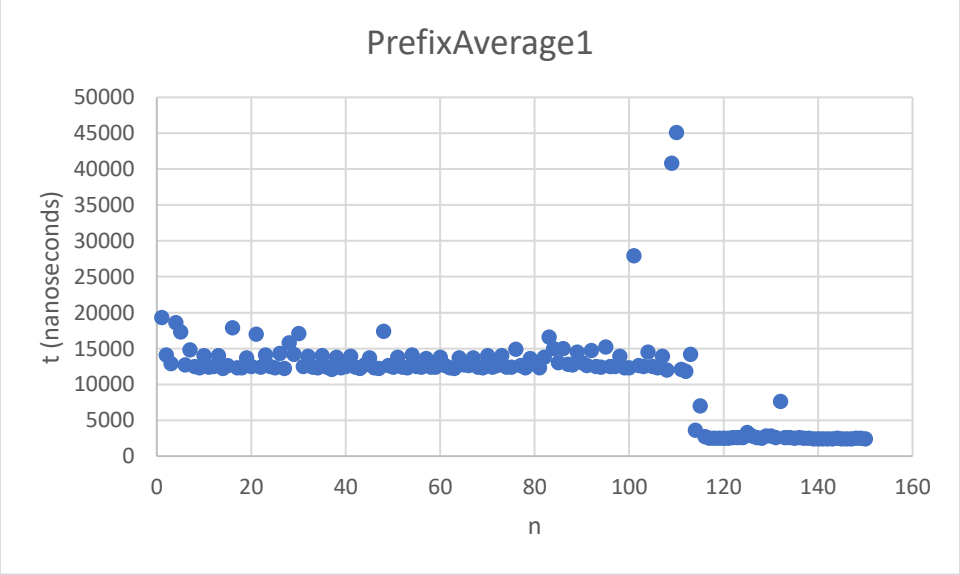
-lijn 5: kost toekenning = O(1)

Big O= O(1)+O(n)+O(1)+ O(1)

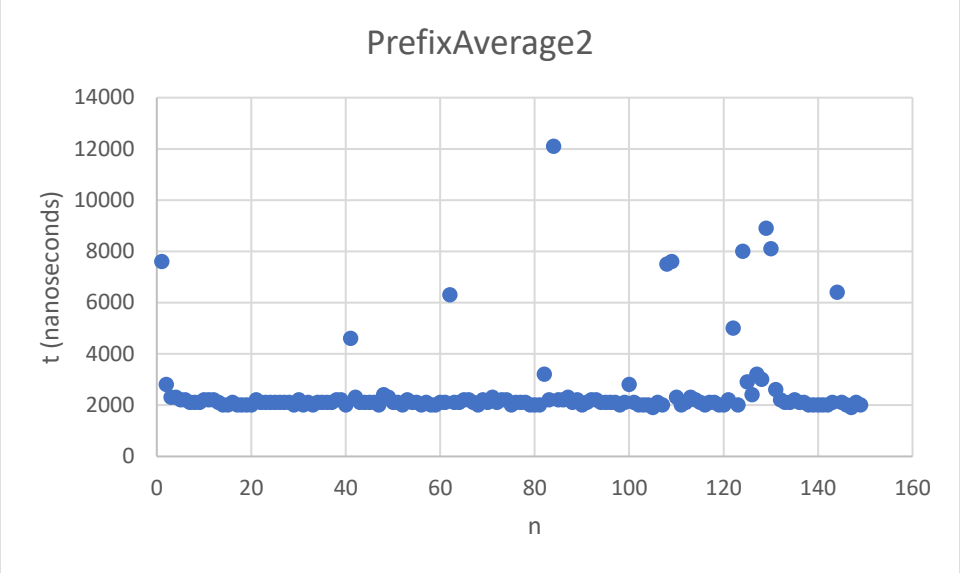
=O(n)

Resultaten:

We hebben een grafiek laten plotten van het eerste algoritme, op de y-as is de tijd te zien hoelang elke cycle duurde. Op de x-as zijn de cycles af te lezen in dit geval zijn dat er 150. We zien dat het een vrij lineair verband vertoont.

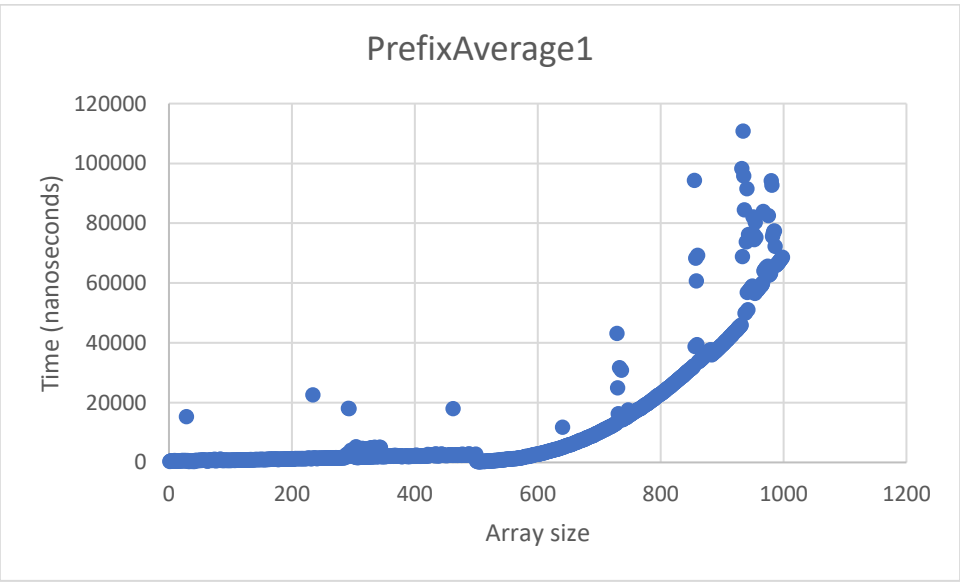


Bij het tweede algoritme hebben we ook 150 keer door het algoritme gegaan en zien we ook dat elke cycle er ongeveer even lang over doet.



Als we de twee grafieken met elkaar vergelijken zien we wel degelijk een verschil. Wanneer we gaan kijken zien we dat het eerste algoritme er gemiddeld tussen de 10000 en 15000 nanoseconden over doet, terwijl het tweede algoritme er maar rond de 2000 nanoseconden over doet. Het tweede is dus meer als 5 keer sneller wanneer we met een array van 25 integers meegeven.

Op onderstaande grafiek hebben we de grootte van de eerste algoritme laten veranderen van 1 tot 1000 integers. We kunnen nu duidelijk het kwadratisch verband terugvinden zoals in de berekening van de big O.



Het tweede algoritme heeft een sporadischere grafiek, al kunnen we nog wel zien dat het een lineair verband volgt. We zien duidelijk dat het tweede algoritme efficiënter werkt dan het eerste. Het heeft voor elke grootte van array een snellere tijd dan het eerste.