2015

Verslag opdracht 1

VICTOR VAN DE RIET & ROB LOGTENBERG

De analyse

Legale permutaties

Algoritme 1, 2 en 3:

Bij algoritme 1 komen er legale permutaties voor, want in de tweede ‘for loop’ controleren we of het getal al in eerdere indexen zijn voorgekomen. Als in de tweede ‘for loop’ een getal al eerder is voorgekomen stopt die de ‘for loop’, voegt geen getal toe en wordt er een nieuwe getal gegenereerd en het proces opnieuw uitgevoerd. De kans op een zelfde permutatie is klein, want er wordt elke keer een willekeurig getal gegenereerd. Dit kan elke keer verschillen, omdat niet bekend is welke getal het systeem gaat genereren.

Bij algoritme 2 komen er ook legale permutaties voor, want in deze algoritme wordt in een tweede lijst bijgehouden of het getal al is voorgekomen. Bestaat het getal al, dan wordt het gegenereerde getal niet toegevoegd en wordt er opnieuw een getal gegenereerd. De kans op een zelfde permutatie is klein, want er wordt elke keer een willekeurig getal gegenereerd. Dit kan elke keer verschillen, omdat niet bekend is welke getal het systeem gaat genereren.

Bij algoritme 3 is er een legale permutatie. Een getal kan niet dubbel voorkomen, omdat je het waarde van de lijst wisselt met een eerdere index van de lijst met de waarde erin. Een getal komt ook maximaal één keer voor, omdat je de lijst één keer opvult van 0 tot en met de lengte van die lijst. Hier hangt de kans op een zelfde permutatie af van de lengte van de lijst. Is de lijst klein ,dan is de kans op het ruilen van getallen minder dan een lijst met meer getallen. Bij meer getallen wordt er meer gewisseld, dus is er een grotere kans dat er geen zelfde permutaties worden gecreëerd.

Big-Oh schatting

Algoritme 1:

```java
public int[] Algorithm1(int number) {
    int[] a = new int[number];
    boolean ignoreZero = false;
    for (int i = 0; i < number; i++) {
        int input = r.nextInt(number);
        boolean found = false;
        for (int j = 0; j < a.length; j++) {
            if (input == a[j]) {
                if (input == 0 && !ignoreZero) {
                    ignoreZero = true;
                    break;
                } else {
                    found = true;
                    break;
                }
            }
        }
        if (found) {
            i--;
        } else {
            a[i] = input;
        }

    }
    return a;
}
```

De Big-OH is: $O(n^2)$. Dit komt omdat we in het algoritme 2 for-loops gebruiken. De eerste for-loop gaat tot “n” en de tweede for-loop gaat in het slechtste geval tot “n”. De constante waarden mag je verwaarlozen.

Algoritme 2:

```
public void Algorithm2(int number) {
    int[] a = new int[number];
    boolean[] used = new boolean[number];
    for (int i = 0; i < a.length; i++) {
        int input = r.nextInt(number);
        if (used[input] == true) {
            i--;
        } else {
            used[input] = true;
            a[i] = input;
        }
    }
}
```

Bij algoritme 2 is de Big-Oh: O(n). Hier is maar 1 for-loop en dat levert maar één “n” op. Contante getallen mag je verwaarlozen.

Algoritme 3:

```
public int[] Algorithm3(int number) {
    int[] a = new int[number];
    int tempNumber;
    int random;
    for (int i = 0; i < number; i++) {
        random = r.nextInt(i + 1);
        a[i] = i;
        tempNumber = a[i];
        a[i] = a[random];
        a[random] = tempNumber;

    }
    return a;
}
```

Ook bij algoritme 3 is de Big-Oh: O(n). Ook hier is maar 1 for-loop en levert maar één “n” op. Constante getallen mag je verwaarlozen.

Ontwerp

Algoritme 1:

Bij dit algoritme wordt eerst een willekeurig getal gegenereerd hierna wordt gekeken of dit getal al in de array staat. Als dit niet zo is wordt het getal in de array gezet als dit wel zo is wordt er een nieuw getal gegenereerd en weer gekeken of hij er al instaat. Dit gaat zo door tot de array volledig gevuld is.

Algoritme 2:

Dit algoritme werkt bijna op dezelfde manier. Echter wordt er hier een array used bijgehouden. Als een getal in de hoofd array wordt geplaatst, wordt er in de array used op de plaats wat het getal was true gezet. Op deze manier kun je makkelijker controleren of het getal al een keer gebruikt is.

Algoritme 3:

Hierbij wordt het algoritme stap voor stap gevuld, zodra een plek is gevuld wordt deze direct verwisseld met een willekeurige reeds gevulde positie.

Implementatie:

Zie code

Resultaten

Meetresultaten

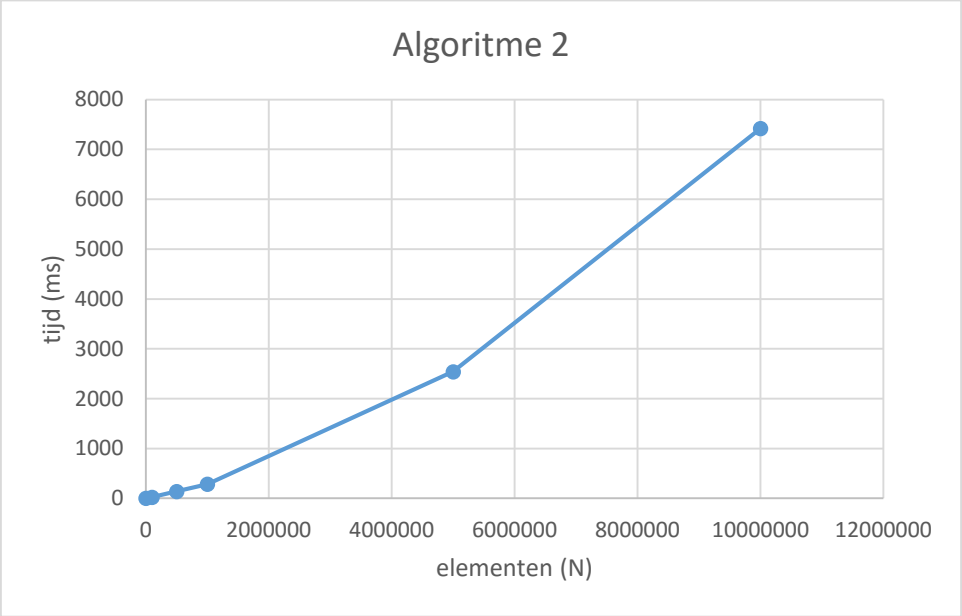
Elementen (N)	Tijd (ms)
0	0
5000	92
10000	409
20000	1838
50000	13604
100000	59948

De grafiek neemt kwadratisch toe, dus de Big-Oh schatting klopt wel

Als we naar N = 10000 kijken t.o.v. N = 5000 dan is N verdubbelt maar omdat de schatting kwadratisch is moet T 4x zo groot zijn. Als we naar de bijbehorende tijden kijken klopt dit. Als we verder gaan vergelijken dan klopt dit ook voor de rest van de Elementen met de bijbehorende Tijd. Alleen hoe groter de N hoe groter de afwijking t.o.v. de N in de rij daarboven.

Algoritme 2:

Elementen (N)	Tijd (ms)
0	0
100000	22,5
500000	140
1000000	283
5000000	2542,5
10000000	7416,5

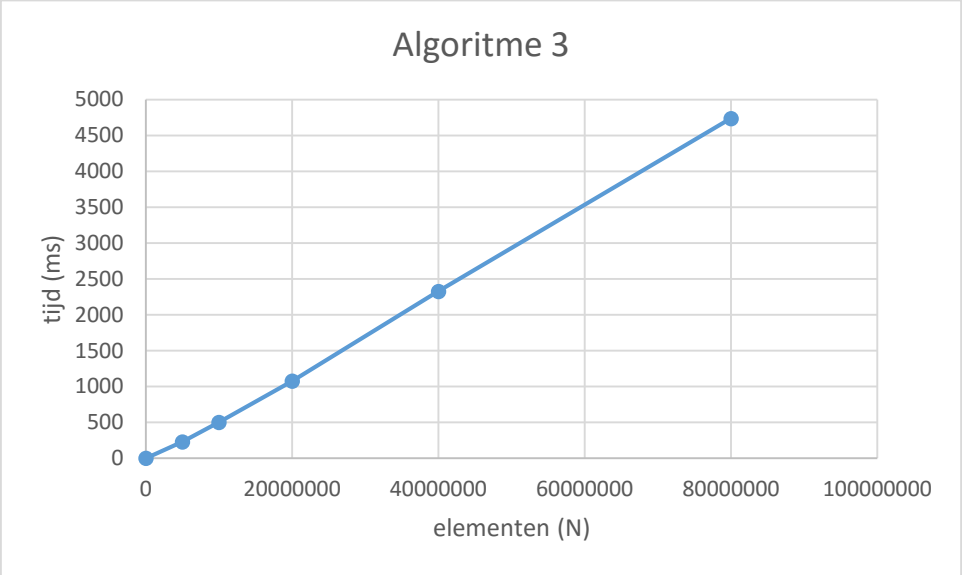


De grafiek is kwadratisch, dus de Big-Oh schatting klopt niet. Waarschijnlijk komt dit omdat het algoritme ook moet checken of het getal al een keer gebruikt is. Dit neemt veel tijd in beslag.

Als we hier naar deze tabel kijken zien we dat 500000, t.o.v. 10000, 5x zo groot is als 100000 dus moet de tijd ook 5x zo groot zijn. Want we hebben immers geschat dat de grafiek lineair zou zijn. Dit gaat ook goed als we naar 1000000 kijken t.o.v. 500000 maar vanaf 5000000 is er geen touw meer aan vast te knopen. De tijd bij 5000000 t.o.v. 1000000 is (3542,5 / 283) ≈ 12 en dat is niet (ongeveer) 566 zoals we verwacht hadden. Dus hier kunnen we vrij weinig mee.

Algoritme 3:

Elementen (N)	Tijd (ms)
0	0
5000000	231,5
10000000	502
20000000	1076,5
40000000	2328
80000000	4737



De grafiek is lineair, dus onze geschatte Big-Oh komt wel overeen met de praktijk.

Als we kijken naar 10000000 t.o.v. 5000000 is dat een verdubbeling en als we daarna kijken naar de bijbehorende tijd zien we ook een verdubbeling (ongeveer), dit is ook het geval bij 20000000, 40000000 en 80000000 dus de grafiek klopt bij de geschatte Big-Oh.