2015

Verslag opdracht 1

VICTOR VAN DE RIET & ROB LOGTENBERG

De analyse

Legale permutaties

Big-Oh schatting

Algoritme 1:

```
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.

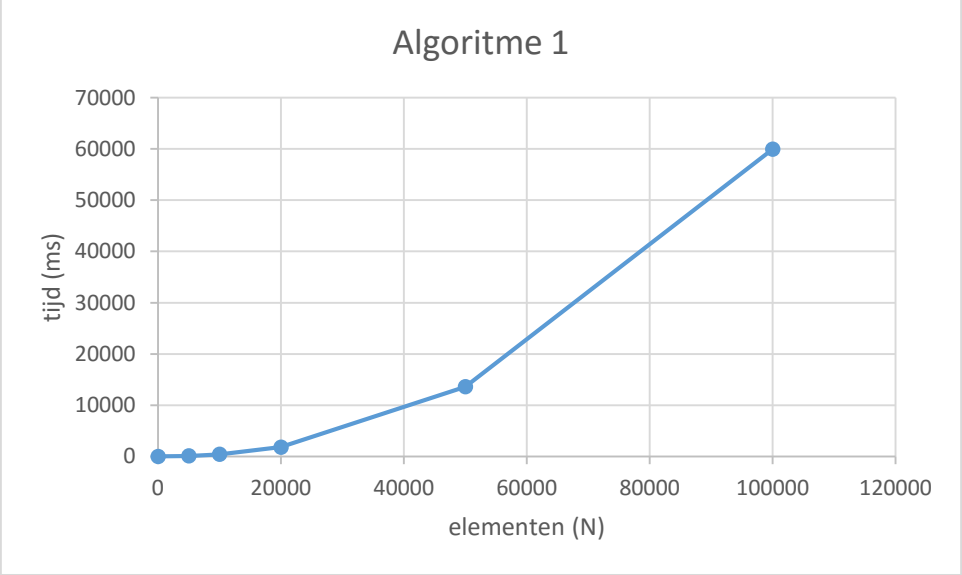
Implementatie:

Zie code

Resultaten

Meetresultaten

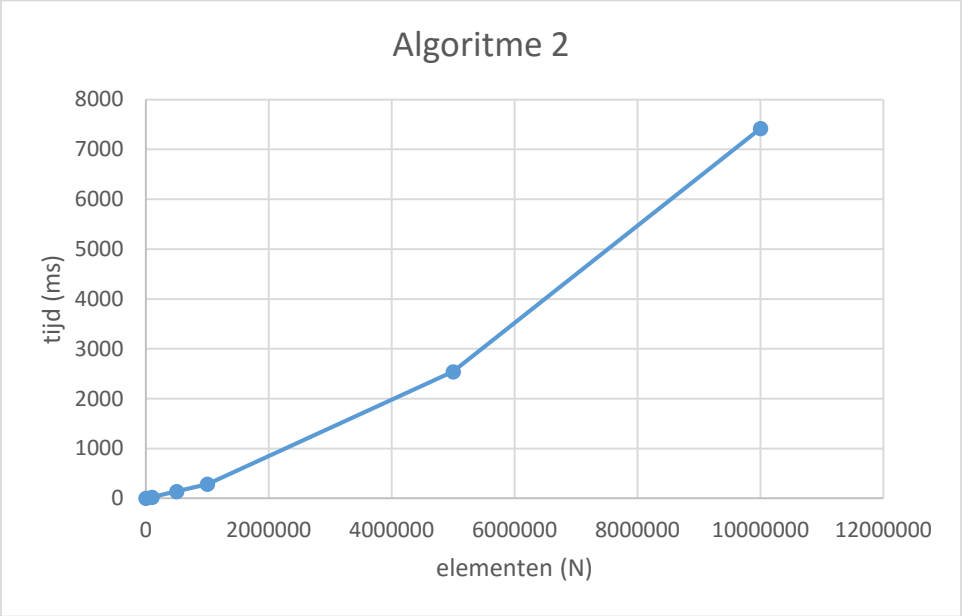
Elementen (N)	Tijd (ms)
0	0
100000	22,5
500000	140
1000000	283
5000000	2542,5
10000000	7416,5



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

Algoritme 2:

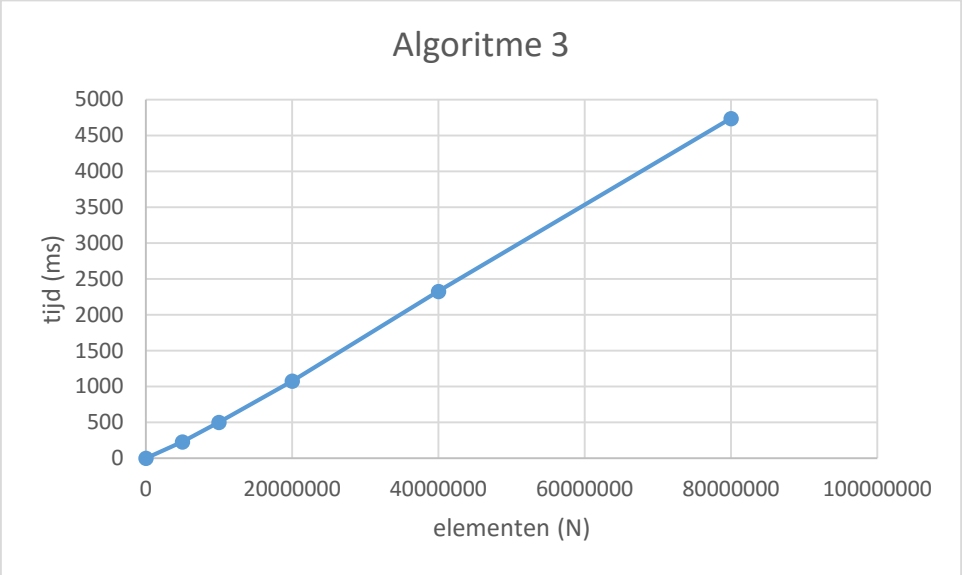
Elementen (N)	Tijd (ms)
0	0
100000	22,5
500000	140
1000000	283
5000000	2542,5
10000000	7416,5



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.

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.