Hogeschool van Amsterdam

Sorting & Searching

Efficiëntie van geavanceerde sorteeralgoritmes

Practicum 1

Author:
Robert Bakker

Studentnummer:
500689284

Klas:
IVSE4

Author:
Robert Bakker

Studentnummer:
500689284

Klas:
IVSE4

Blok 2, 2016 - 2017

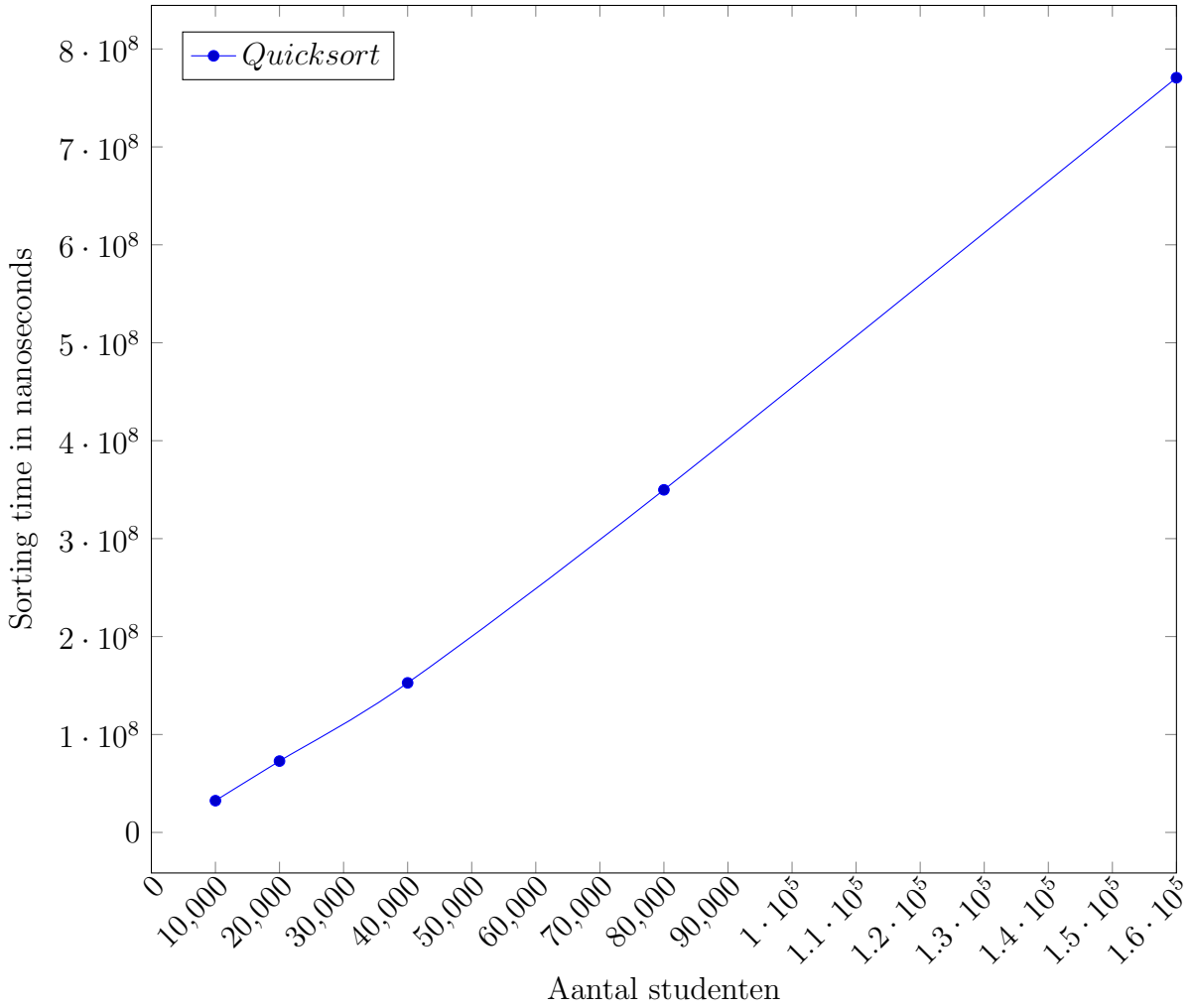
Inhoudsopgave

a Resultaten van studenten sorteren met een advanced sort

a.1 Advanced sort toevoegen

```
// De quicksort accepteert een lijst van objecten met een comparable
// interface, het beginpunt van links, en het beginpunt van rechts
private void quicksort(Comparable[] list, int low, int high) {

    // Neem het middelpunt van de array als spil (draaipunt)
    Comparable pivot = list[low + (high - low) / 2];

    int i = low; // linkerkant
    int j = high; // rechterkant

    while (i <= j) {
        // Wanneer object vanaf links kleiner is dan de spil
        // Verschuif naar de volgende in de linkerlijst
        while (list[i].compareTo(pivot) < 0) i++;

        // Wanneer object vanaf rechts groter is dan de spil
        // Verschuif naar de volgende in de rechterlijst
        while (list[j].compareTo(pivot) > 0) j--;

        // Als er een index van de linkerlijst is gevonden, met een waarde
        // die groter is dan de spil, en een index in de rechterlijst met
        // een waarde die kleiner is dan de spil, moeten de 2 waarden
        // worden omgedraaid
        if (i <= j) {
            Comparable temp = list[i];
            list[i] = list[j];
            list[j] = temp;
            i++;
            j--;
        }
    }
    // Hetzelfde voor de rest van de linkerlijst
    if (low < j) {
        quicksort(list, low, j);
    }
    // en voor de rechterlijst
    if (high > i) {
        quicksort(list, i, high);
    }
}
```

a.2 Efficiëntie - Experiment

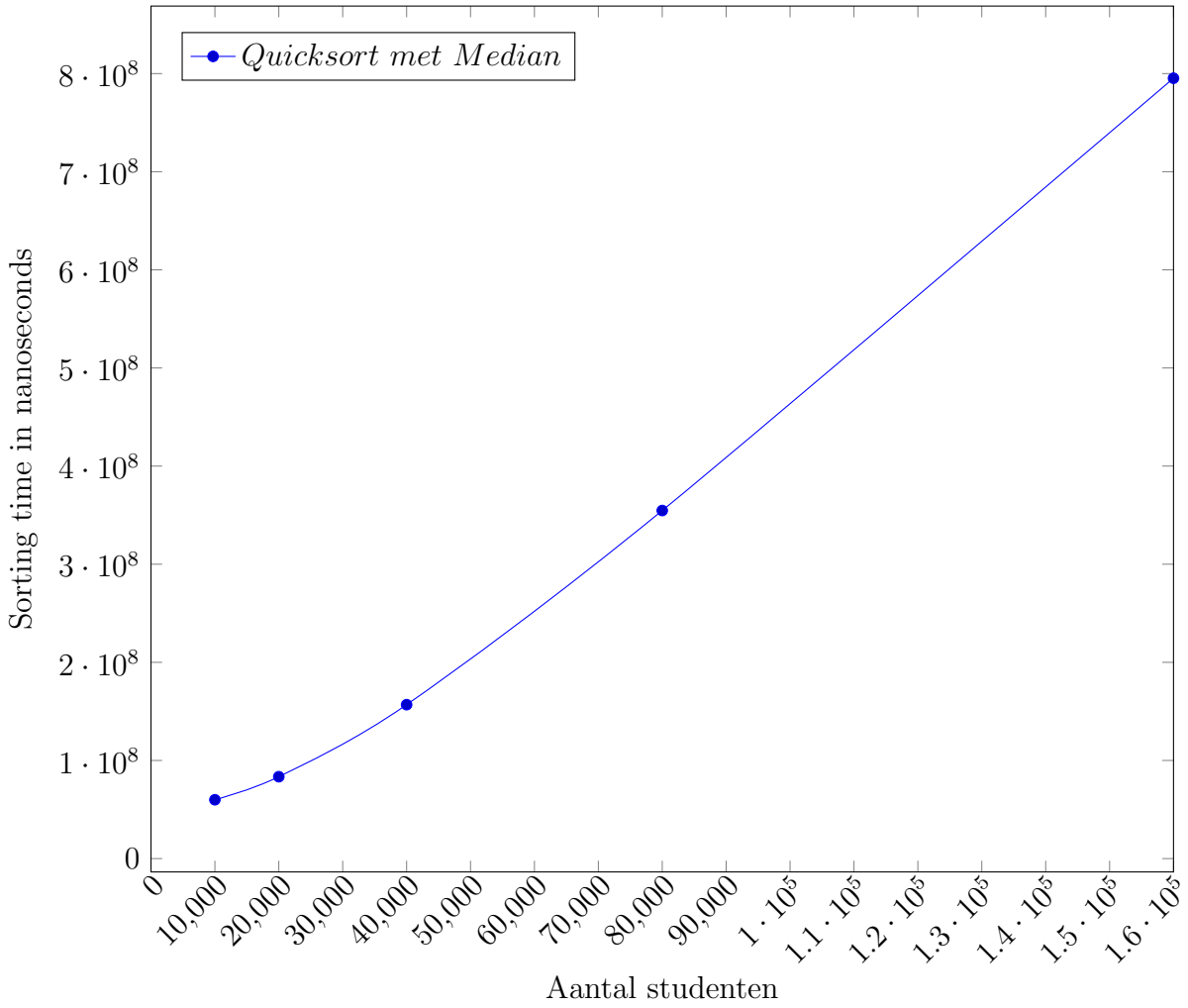


a.3 Big O

2n(log(n))

b Verbetering toevoegen aan algoritme

b.1 Experiment



b.2 Big O

c Resultaten in een Binary Search Tree en implementatie van rank()

De input is een lijst van studenten bestaand uit een cijfer en studentnummer. De input set bestaat uit 10000 studenten. Op deze set voeren wij de rank method uit. om te tellen welke hoeveel studenten lager dan een bepaald cijfer hebben behaald.

c.1 BST implementatie rank

```
private int rank(Key key, Node x) {
       if (x == null) {
           return 0;
       }
       int cmp = key.compareTo(x.key);
       if (cmp < 0) {
           return rank(key, x.left);
       } else if (cmp > 0) {
           return x.val.size() + size(x.left) + rank(key, x.right);
       } else {
           return size(x.left);
       }
   }
```

c.2 output rank

Grade: 1, rank: 0
Grade: 2, rank: 1111
Grade: 3, rank: 2154
Grade: 4, rank: 3297
Grade: 5, rank: 4471
Grade: 6, rank: 5564
Grade: 7, rank: 6713
Grade: 8, rank: 7826
Grade: 9, rank: 8900
Grade: 10, rank: 10000