Anthony & Ezra

500695495  500600102

Datastructures OPdracht 2

Inhoud

[Klassendiagram en Code snippets 2](#_Toc400537209)

[Klassendiagram 2](#_Toc400537210)

[Codesnippets 3](#_Toc400537211)

[Insertion sort 3](#_Toc400537212)

[BucketSort 3](#_Toc400537213)

[Output van de Sorts 4](#_Toc400537214)

[Output insertionSort 4](#_Toc400537215)

[Output BucketSort 4](#_Toc400537216)

[Bepaling van de big O 5](#_Toc400537217)

[Resultaten 5](#_Toc400537218)

[Insertion 5](#_Toc400537219)

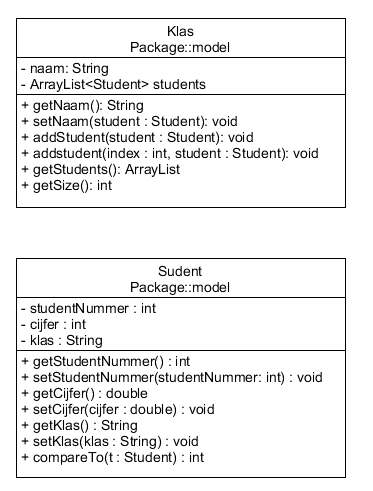
[Bucket 5](#_Toc400537220)

[Bijlage 6](#_Toc400537221)

[BucketSort Code 6](#_Toc400537222)

# Klassendiagram en Code snippets

## Klassendiagram



De applicatie bevat twee klassen een student klasse en een klas klasse.

De student klasse bevat 3 attributen Studentnummer(int) een studentcijfer(double) en een klas(String). Daarnaast implementeert Student de interface “Comparable” dus de bevat Student een CompareTo methode. De compareTo gebruiken we in voor de insertion Sort om de studentcijfers met elkaar te vergelijken.

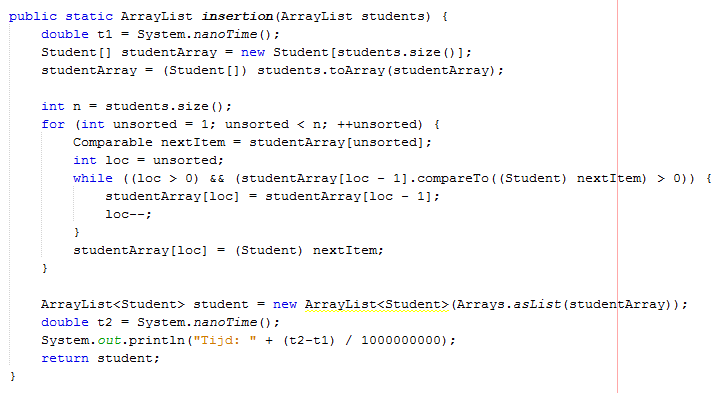
De klas klasse bevat een String Naam en een ArrayList student. De klasse Klas wordt gebruikt in de bucket methode. Klas bevat de normale getters en setters voor Naam en een add- en getStudent voor de Arraylist.

## Codesnippets

### Insertion sort

De insertion sort werkt als volgt. In de insertion sort wordt eerst de meegegeven Arraylist gekopieerd naar een Array. Vervolgens maken we een for-loop waarin we allereerst van een studentenobject(uit de Array) een comperable maken. In de while-loop worden de cijfers van de studenten met elkaar vergeleken en worden de student met het hogere cijfer een plek opgeschoven in de array. Daarna wordt de array weer omgezet naar de student arraylist en deze wordt dan meegegeven in de return.

(onderstaande afbeelding is van de insertion sort)



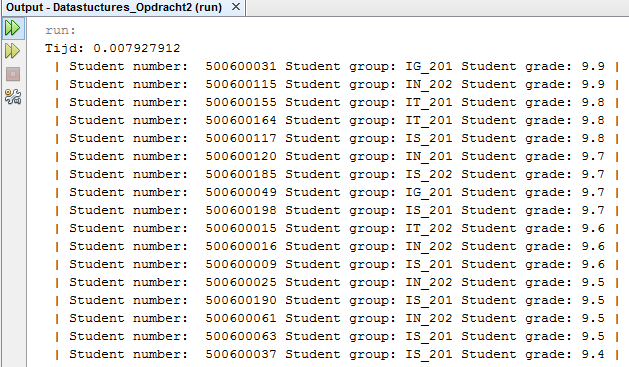
### BucketSort

Bij de bucket sort beginnen we eerst met het aanmaken van een arraylist genaamd klassen. Hier worden de klassen ingezet. Vervolgens loopen we door de alle studenten in de arraylist die is meegegeven en kijken we bij iedere student naar de klas. Zo kijken we eerst of deze klas al bestaat, zo niet dan maken we de klas aan en kijken we op welke plek in de arraylist hij moet staan, zodat deze dan meteen gesorteerd is. Als de klas wel bestaat, dan voegen we deze student toe in die klas en ook op dezelfde manier als de hoe de klas in de klassen array wordt gezet. Op het laatst lezen we achter elkaar de klassen uit en zetten deze terug in één arraylist en geven we deze mee als output.

# Output van de Sorts

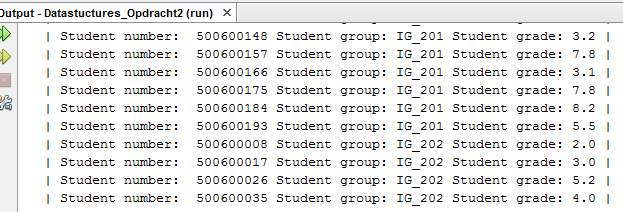
## Output insertionSort

Hieronder de output als de student arraylist wordt gesorteerd op Cijfer (met insertion sort)



## Output BucketSort

Hieronder de output als de student arraylist wordt gesorteerd met BucketSort



# Bepaling van de big O

## Resultaten

### Insertion

|  |  |
| --- | --- |
| Aantal Studenten | Tijd (in seconde) |
| 200 | 0.015 |
| 400 | 0.023 |
| 800 | 0.033 |
| 1600 | 0.118 |
| 3200 | 0.174 |

De insertion sort maakt gebruik van een for loop en een while-loop wat dit n^2 maakt.

### Bucket

|  |  |
| --- | --- |
| Aantal Studenten | Tijd (in Seconde |
| 200 | 0.014 |
| 400 | 0.027 |
| 800 | 0.099 |
| 1600 | 0.422 |
| 3200 | 1.687 |

In de bucket maken we gebruik van 3 for-loops wat dit O(n^3) zou maken.

# Bijlage

## BucketSort Code

public static ArrayList bucket(ArrayList students) {

double t1 = System.nanoTime();

Student[] studentArray = new Student[students.size()];

studentArray = (Student[]) students.toArray(studentArray);

ArrayList<Klas> klassen = new ArrayList<Klas>();

for (Student student : studentArray) {

if (klassen.isEmpty()) {

Klas newKlas = new Klas();

newKlas.setNaam(student.getKlas());

newKlas.addStudent((Student) student);

klassen.add(newKlas);

} else {

boolean zitErin = false;

boolean appel = false;

int index = -1;

for (int j = 0; j < klassen.size(); j++) {

if (klassen.get(j).getNaam().equals(student.getKlas())) {

boolean kiwi = false;

int indx = -1;

for (int k = 0; k < klassen.get(j).getSize(); k++) {

Student[] studentInKlas = new Student[students.size()];

studentInKlas = (Student[]) klassen.get(j).getStudents().toArray(studentArray);

if ((student.getStudentNummer() < studentInKlas[k].getStudentNummer()) && (kiwi == false)) {

indx = k;

kiwi = true;

}

}

if(indx == -1){

klassen.get(j).addStudent((Student) student);

} else{

klassen.get(j).addStudent(indx, (Student) student);

}

zitErin = true;

}

if ((appel == false) && (student.getKlas().compareTo(klassen.get(j).getNaam()) < 0)) {

index = j;

appel = true;

}

}

if (!zitErin) {

Klas newKlas = new Klas();

newKlas.setNaam(student.getKlas());

newKlas.addStudent((Student) student);

if (index == -1) {

klassen.add(newKlas);

} else {

klassen.add(index, newKlas);

}

}

}

}

ArrayList<Student> outputArray = new ArrayList<Student>();

for (int u = 0; u < klassen.size(); u++) {

for (int y = 0; y < klassen.get(u).getStudents().size(); y++) {

outputArray.add((Student) klassen.get(u).getStudents().get(y));

}

}

double t2 = System.nanoTime();

System.out.println("Tijd: " + (t2-t1) / 1000000000);

return outputArray;

}