10/14/2016

Eloy Maduro Clement / Serdar Gürler

IS-204

Practicum 2

Efficientie van sorteeralgoritmes

Table of Contents

[Inleiding 3](#_Toc464314802)

[Opdracht 2 – Sorting Efficiency 4](#_Toc464314803)

[Classes 4](#_Toc464314804)

[1. Genereer en sorteer 5](#_Toc464314805)

[Genereer klassen 5](#_Toc464314806)

[Genereer studenten 6](#_Toc464314807)

[Nummering van studenten en klassen, bepaling van cijfer 6](#_Toc464314808)

[Sorteren op cijfer 2](#_Toc464314809)

[Sorteren op klas en studentnummer 1](#_Toc464314810)

[2. Meet de efficientie 2](#_Toc464314811)

[Conclusie 3](#_Toc464314812)

# Inleiding

Dit verslag dient als meetinstrument voor de faculteit om ons kennen en kunnen op gebied van sorteermethoden te pijlen. In dit verslag zullen we aantonen, niet alleen dat we de stof beheersen, maar ook dat we weten hoe de methoden in elkaar steken en hoe het (waar mogelijk) beter kan. Verder bekijken we de verschillen tussen de sorteermethoden in de big O notatie en bekijken we de mogelijke overeenkomsten tussen de methoden.

# Opdracht 2 – Sorting Efficiency

## Classes

**Packages: Classes: Methods:**

1. Main

* GroupSet
  + - CreateStudents() maakt x aantal studenten aan
    - getStudents() geeft aangemaakte lijst van studenten weer
    - CreateGroups() maakt x aantal klassen aan afhankelijk van aantal studenten
    - getGroups() geeft aangemaakte array van klassen terug
    - sortByGrade() sorteert studenten aan de hand van hun cijfer
* Main
  + - Main() wordt gebruikt om de applicatie te testen

1. Model

* Group
  + - Add() voegt een student toe aan de klas
    - Group() maakt een instantie van een klas
    - getGroup() geeft een klas terug
    - setGroupID() wijst ID waarde aan een bepaalde klas toe
    - toString() geeft de ID terug van een bepaalde klas
* Student
  + - SecureRandom() geeft een random waarde terug
    - Student() maakt een instantie van een student
    - getStudentNum() geeft studentnummer terug
    - setStudentNum() wijst studentnummer aan een bepaalde student toe
    - getGroup() geeft klas van een bepaalde student terug
    - getGrade() geeft cijfer van een bepaalde student terug
    - setGrade() wijst cijferwaarde aan een bepaalde student toe
    - toString() geeft studentnr, cijfer en klas terug in de vorm van tekst

1. Sort

* GroupComparator
  + - Compare() vergelijkt van twee klassen het tweede character in de groupID
* StudentNoComparator
  + - Compare() vergelijkt studentnummers met elkaar

## Genereer en sorteer

In dit deel van de opdracht maken we stapsgewijs studenten aan, gevolgd door klassen op basis van de hoeveelheid aangemaakte studenten.

### Genereer klassen

Voordat we het aantal aangevraagde studenten kunnen aanmaken, moeten we eerst (op basis van de aanvraag) een bepaald aantal klassen aanmaken. Dit doen we door middel van de methode createGroups(). Deze methode bevindt zich in de GroupSet.

public void createGroups() {

if (totalStudents <= 0) {

System.err.println("Because no humans.");

} else {

studentsPerRoute = (int) Math.ceil(totalStudents / (double) ROUTE\_MAX);

groupsPerRoute = (int) Math.ceil((double) studentsPerRoute / MAX\_SIZE);

totalGroups = groupsPerRoute \* ROUTE\_MAX;

groups = new Group[totalGroups];

String[] route = new String[]{"B", "G", "N", "S", "T"};

int routeCurrent = 0;

int groupNum = 0;

for (int i = 0; i < groups.length; i++) {

if (routeCurrent == ROUTE\_MAX) {

routeCurrent = 0;

}

if (i % 5 == 0) {

groupNum++;

}

groups[i] = new Group("I" + route[routeCurrent++] + (GROUP\_START + groupNum));

}

System.out.println("Total students: " + totalStudents);

System.out.println("Students per route: " + studentsPerRoute);

System.out.println("Groups per route: " + groupsPerRoute);

System.out.println("Total groups of students: " + totalGroups);

System.out.println("");

}

}

### Genereer studenten

Om studenten aan te maken in ons programma roepen we de methode CreateStudents() aan in main. Deze methode bevindt zich in GroupSet.

public void createStudents() {

int groupIndex = 0;

for (int i = 0; i < totalStudents; i++) {

if (groupIndex == groups.length) {

groupIndex = 0;

}

Group currentGroup = groups[groupIndex++];

Student s = new Student(currentGroup);

students.add(s);

currentGroup.add(s);

}

}

In de for-loop van createStudents() worden de studenten bij aanmaken gelijk in de bijbehorende groep gestopt.

### Nummering van studenten en klassen, bepaling van cijfer

Wanneer een student wordt aangemaakt door middel van de student constructor (in de methode createStudents()), krijgt elke student een studentnummer met dezelfde waarde als de voorgaande student +1. Dit doen we door middel van een setterMethode, die een waarde aan de studentNum attribuut meegeeft.

Hierbij de gepaste code snippet:

private void setStudentNum() {

studentNum = ++studentNumStart;

}

Hetzelfde geldt voor het toewijzen van een willekeurig cijfer. Bij het aanmaken van een student wordt de methode setGrade() aangeroepen. Deze methode geeft een random float terug door middel van java.security.SecureRandom.

Hierbij de gepaste code snippet:

public static SecureRandom getSEC() {

return SEC;

}

Hieronder volgt de output bij het aanmaken van 165 studenten:

Total students: 165

Students per route: 33

Groups per route: 2

Total groups of students: 10

Studentnr: 50060001

Grade: 7.9

Group: IB20001

Studentnr: 50060011 Grade: 8.7 Group: IB20001

Studentnr: 50060021 Grade: 4.0 Group: IB20001

Studentnr: 50060031 Grade: 5.5 Group: IB20001

Studentnr: 50060041 Grade: 9.1 Group: IB20001

Studentnr: 50060051 Grade: 8.1 Group: IB20001

Studentnr: 50060061 Grade: 4.0 Group: IB20001

Studentnr: 50060071 Grade: 9.0 Group: IB20001

Studentnr: 50060081 Grade: 1.2 Group: IB20001

Studentnr: 50060091 Grade: 4.0 Group: IB20001

Studentnr: 50060101 Grade: 6.0 Group: IB20001

Studentnr: 50060111 Grade: 5.5 Group: IB20001

Studentnr: 50060121 Grade: 6.7 Group: IB20001

Studentnr: 50060131 Grade: 9.6 Group: IB20001

Studentnr: 50060141 Grade: 8.5 Group: IB20001

Studentnr: 50060151 Grade: 2.9 Group: IB20001

Studentnr: 50060161 Grade: 9.8 Group: IB20001

**STUDENTS PER GROUP: 17**

Studentnr: 50060006 Grade: 4.9 Group: IB20002

Studentnr: 50060016 Grade: 2.1 Group: IB20002

Studentnr: 50060026 Grade: 3.9 Group: IB20002

Studentnr: 50060036 Grade: 4.9 Group: IB20002

Studentnr: 50060046 Grade: 1.0 Group: IB20002

Studentnr: 50060056 Grade: 7.7 Group: IB20002

Studentnr: 50060066 Grade: 7.5 Group: IB20002

Studentnr: 50060076 Grade: 6.5 Group: IB20002

Studentnr: 50060086 Grade: 6.6 Group: IB20002

Studentnr: 50060096 Grade: 2.0 Group: IB20002

Studentnr: 50060106 Grade: 3.0 Group: IB20002

Studentnr: 50060116 Grade: 8.8 Group: IB20002

Studentnr: 50060126 Grade: 2.0 Group: IB20002

Studentnr: 50060136 Grade: 8.9 Group: IB20002

Studentnr: 50060146 Grade: 6.4 Group: IB20002

Studentnr: 50060156 Grade: 4.3 Group: IB20002

**STUDENTS PER GROUP: 16**

Studentnr: 50060002 Grade: 2.1 Group: IG20001

Studentnr: 50060012 Grade: 8.7 Group: IG20001

Studentnr: 50060022 Grade: 7.4 Group: IG20001

Studentnr: 50060032 Grade: 4.2 Group: IG20001

Studentnr: 50060042 Grade: 9.8 Group: IG20001

Studentnr: 50060052 Grade: 9.6 Group: IG20001

Studentnr: 50060062 Grade: 2.0 Group: IG20001

Studentnr: 50060072 Grade: 2.4 Group: IG20001

Studentnr: 50060082 Grade: 4.2 Group: IG20001

Studentnr: 50060092 Grade: 3.3 Group: IG20001

Studentnr: 50060102 Grade: 6.8 Group: IG20001

Studentnr: 50060112 Grade: 9.5 Group: IG20001

Studentnr: 50060122 Grade: 7.5 Group: IG20001

Studentnr: 50060132 Grade: 2.2 Group: IG20001

Studentnr: 50060142 Grade: 4.6 Group: IG20001

Studentnr: 50060152 Grade: 2.3 Group: IG20001

Studentnr: 50060162 Grade: 7.8 Group: IG20001

**STUDENTS PER GROUP: 17**

Studentnr: 50060007 Grade: 5.8 Group: IG20002

Studentnr: 50060017 Grade: 2.2 Group: IG20002

Studentnr: 50060027 Grade: 5.4 Group: IG20002

Studentnr: 50060037 Grade: 6.1 Group: IG20002

Studentnr: 50060047 Grade: 7.0 Group: IG20002

Studentnr: 50060057 Grade: 6.1 Group: IG20002

Studentnr: 50060067 Grade: 2.9 Group: IG20002

Studentnr: 50060077 Grade: 6.9 Group: IG20002

Studentnr: 50060087 Grade: 2.1 Group: IG20002

Studentnr: 50060097 Grade: 4.4 Group: IG20002

Studentnr: 50060107 Grade: 6.7 Group: IG20002

Studentnr: 50060117 Grade: 9.8 Group: IG20002

Studentnr: 50060127 Grade: 6.6 Group: IG20002

Studentnr: 50060137 Grade: 9.8 Group: IG20002

Studentnr: 50060147 Grade: 8.1 Group: IG20002

Studentnr: 50060157 Grade: 9.4 Group: IG20002

**STUDENTS PER GROUP: 16**

Studentnr: 50060003 Grade: 6.9 Group: IN20001

Studentnr: 50060013 Grade: 1.0 Group: IN20001

Studentnr: 50060023 Grade: 9.1 Group: IN20001

Studentnr: 50060033 Grade: 8.0 Group: IN20001

Studentnr: 50060043 Grade: 7.2 Group: IN20001

Studentnr: 50060053 Grade: 4.7 Group: IN20001

Studentnr: 50060063 Grade: 4.3 Group: IN20001

Studentnr: 50060073 Grade: 9.1 Group: IN20001

Studentnr: 50060083 Grade: 9.6 Group: IN20001

Studentnr: 50060093 Grade: 6.2 Group: IN20001

Studentnr: 50060103 Grade: 6.1 Group: IN20001

Studentnr: 50060113 Grade: 2.5 Group: IN20001

Studentnr: 50060123 Grade: 1.1 Group: IN20001

Studentnr: 50060133 Grade: 6.2 Group: IN20001

Studentnr: 50060143 Grade: 2.7 Group: IN20001

Studentnr: 50060153 Grade: 7.8 Group: IN20001

Studentnr: 50060163 Grade: 5.1 Group: IN20001

**STUDENTS PER GROUP: 17**

Studentnr: 50060008 Grade: 3.4 Group: IN20002

Studentnr: 50060018 Grade: 6.6 Group: IN20002

Studentnr: 50060028 Grade: 8.6 Group: IN20002

Studentnr: 50060038 Grade: 10.0 Group: IN20002

Studentnr: 50060048 Grade: 3.4 Group: IN20002

Studentnr: 50060058 Grade: 3.7 Group: IN20002

Studentnr: 50060068 Grade: 7.7 Group: IN20002

Studentnr: 50060078 Grade: 7.1 Group: IN20002

Studentnr: 50060088 Grade: 2.0 Group: IN20002

Studentnr: 50060098 Grade: 2.8 Group: IN20002

Studentnr: 50060108 Grade: 5.0 Group: IN20002

Studentnr: 50060118 Grade: 3.8 Group: IN20002

Studentnr: 50060128 Grade: 5.6 Group: IN20002

Studentnr: 50060138 Grade: 9.0 Group: IN20002

Studentnr: 50060148 Grade: 2.7 Group: IN20002

Studentnr: 50060158 Grade: 2.4 Group: IN20002

**STUDENTS PER GROUP: 16**

Studentnr: 50060004 Grade: 6.6 Group: IS20001

Studentnr: 50060014 Grade: 7.0 Group: IS20001

Studentnr: 50060024 Grade: 7.6 Group: IS20001

Studentnr: 50060034 Grade: 1.5 Group: IS20001

Studentnr: 50060044 Grade: 6.5 Group: IS20001

Studentnr: 50060054 Grade: 2.4 Group: IS20001

Studentnr: 50060064 Grade: 8.9 Group: IS20001

Studentnr: 50060074 Grade: 6.0 Group: IS20001

Studentnr: 50060084 Grade: 7.0 Group: IS20001

Studentnr: 50060094 Grade: 3.9 Group: IS20001

Studentnr: 50060104 Grade: 9.5 Group: IS20001

Studentnr: 50060114 Grade: 2.4 Group: IS20001

Studentnr: 50060124 Grade: 2.6 Group: IS20001

Studentnr: 50060134 Grade: 9.5 Group: IS20001

Studentnr: 50060144 Grade: 6.2 Group: IS20001

Studentnr: 50060154 Grade: 9.2 Group: IS20001

Studentnr: 50060164 Grade: 8.8 Group: IS20001

**STUDENTS PER GROUP: 17**

Studentnr: 50060009 Grade: 8.5 Group: IS20002

Studentnr: 50060019 Grade: 3.7 Group: IS20002

Studentnr: 50060029 Grade: 6.0 Group: IS20002

Studentnr: 50060039 Grade: 9.0 Group: IS20002

Studentnr: 50060049 Grade: 8.4 Group: IS20002

Studentnr: 50060059 Grade: 1.1 Group: IS20002

Studentnr: 50060069 Grade: 7.1 Group: IS20002

Studentnr: 50060079 Grade: 4.6 Group: IS20002

Studentnr: 50060089 Grade: 3.7 Group: IS20002

Studentnr: 50060099 Grade: 8.0 Group: IS20002

Studentnr: 50060109 Grade: 6.4 Group: IS20002

Studentnr: 50060119 Grade: 3.8 Group: IS20002

Studentnr: 50060129 Grade: 3.7 Group: IS20002

Studentnr: 50060139 Grade: 3.5 Group: IS20002

Studentnr: 50060149 Grade: 2.9 Group: IS20002

Studentnr: 50060159 Grade: 4.6 Group: IS20002

**STUDENTS PER GROUP: 16**

Studentnr: 50060005 Grade: 4.2 Group: IT20001

Studentnr: 50060015 Grade: 9.3 Group: IT20001

Studentnr: 50060025 Grade: 4.9 Group: IT20001

Studentnr: 50060035 Grade: 9.1 Group: IT20001

Studentnr: 50060045 Grade: 9.2 Group: IT20001

Studentnr: 50060055 Grade: 4.3 Group: IT20001

Studentnr: 50060065 Grade: 9.4 Group: IT20001

Studentnr: 50060075 Grade: 9.9 Group: IT20001

Studentnr: 50060085 Grade: 4.2 Group: IT20001

Studentnr: 50060095 Grade: 3.0 Group: IT20001

Studentnr: 50060105 Grade: 3.8 Group: IT20001

Studentnr: 50060115 Grade: 2.3 Group: IT20001

Studentnr: 50060125 Grade: 1.0 Group: IT20001

Studentnr: 50060135 Grade: 1.1 Group: IT20001

Studentnr: 50060145 Grade: 4.6 Group: IT20001

Studentnr: 50060155 Grade: 1.4 Group: IT20001

Studentnr: 50060165 Grade: 7.7 Group: IT20001

**STUDENTS PER GROUP: 17**

Studentnr: 50060010 Grade: 1.6 Group: IT20002

Studentnr: 50060020 Grade: 5.2 Group: IT20002

Studentnr: 50060030 Grade: 5.0 Group: IT20002

Studentnr: 50060040 Grade: 2.5 Group: IT20002

Studentnr: 50060050 Grade: 2.3 Group: IT20002

Studentnr: 50060060 Grade: 5.8 Group: IT20002

Studentnr: 50060070 Grade: 1.8 Group: IT20002

Studentnr: 50060080 Grade: 5.9 Group: IT20002

Studentnr: 50060090 Grade: 5.2 Group: IT20002

Studentnr: 50060100 Grade: 8.3 Group: IT20002

Studentnr: 50060110 Grade: 4.7 Group: IT20002

Studentnr: 50060120 Grade: 4.1 Group: IT20002

Studentnr: 50060130 Grade: 2.9 Group: IT20002

Studentnr: 50060140 Grade: 1.0 Group: IT20002

Studentnr: 50060150 Grade: 3.9 Group: IT20002

Studentnr: 50060160 Grade: 2.2 Group: IT20002

**STUDENTS PER GROUP: 16**

Cijfers worden door middel van Java.Security.SecureRandom gegenereerd en zoals is te zien aan de grafiek zijn nergens duidelijke clusters te vinden. Hierbij de gepaste code snippet:

private double setGrade() {

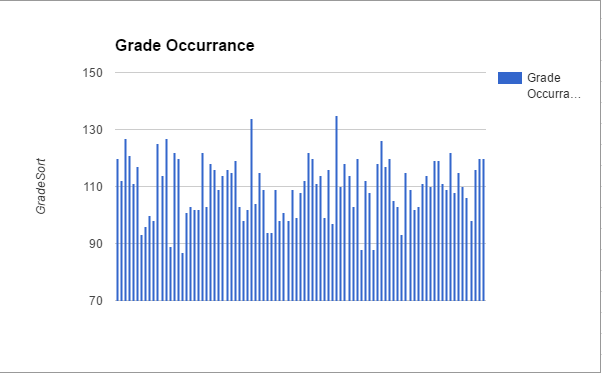
//this.grade = (double) Math.round(1 + (Math.random() \* 10) \* 10d) / 10d;

return SEC.nextInt(91) / 10.0 + 1;

}

Hierbij de grafiek:

(de y-axis geeft de frequentie van een cijfer, en de x-axis geeft de cijfers)



### Sorteren op cijfer

Voor het sorteren van studenten op cijfers hebben we (met tegenzin) gebruik gemaakt van bubble-sort. Onze voorkeur gaat uit naar mergesort omdat in deze use case de big O voor mergesort (n log n) is, wat ook niet bijster snel is, maar zeker wel sneller dan bubble sort, insertion sort, of selection sort. Hierbij de code snippet voor bubble sort:

public void sortByGrade(List<Student> students) {

for (int i = 0; i < students.size() - 1; i++) {

for (int k = i + 1; k < students.size(); k++) {

// Compare left to right, not right to left.

amountCompared++;

if (students.get(i).getGrade() < students.get(k).getGrade()) {

Student temp = students.get(i);

students.set(i, students.get(k));

students.set(k, temp);

}

}

}

}

De output bij het sorteren van 50 studenten op cijfer, van hoog naar laag is als volgt:

Studentnr: 50060011 Grade: 10.0 Group: IB20001

Studentnr: 50060016 Grade: 9.9 Group: IB20001

Studentnr: 50060010 Grade: 9.8 Group: IT20001

Studentnr: 50060039 Grade: 9.8 Group: IS20001

Studentnr: 50060004 Grade: 9.6 Group: IS20001

Studentnr: 50060036 Grade: 9.5 Group: IB20001

Studentnr: 50060022 Grade: 9.3 Group: IG20001

Studentnr: 50060037 Grade: 9.2 Group: IG20001

Studentnr: 50060030 Grade: 9.2 Group: IT20001

Studentnr: 50060005 Grade: 9.0 Group: IT20001

Studentnr: 50060045 Grade: 8.8 Group: IT20001

Studentnr: 50060041 Grade: 8.6 Group: IB20001

Studentnr: 50060003 Grade: 8.6 Group: IN20001

Studentnr: 50060034 Grade: 8.3 Group: IS20001

Studentnr: 50060031 Grade: 8.1 Group: IB20001

Studentnr: 50060050 Grade: 7.9 Group: IT20001

Studentnr: 50060038 Grade: 6.8 Group: IN20001

Studentnr: 50060013 Grade: 6.6 Group: IN20001

Studentnr: 50060044 Grade: 6.5 Group: IS20001

Studentnr: 50060026 Grade: 6.4 Group: IB20001

Studentnr: 50060012 Grade: 6.3 Group: IG20001

Studentnr: 50060002 Grade: 6.0 Group: IG20001

Studentnr: 50060025 Grade: 5.8 Group: IT20001

Studentnr: 50060048 Grade: 5.7 Group: IN20001

Studentnr: 50060027 Grade: 5.4 Group: IG20001

Studentnr: 50060001 Grade: 5.3 Group: IB20001

Studentnr: 50060035 Grade: 5.2 Group: IT20001

Studentnr: 50060032 Grade: 4.7 Group: IG20001

Studentnr: 50060018 Grade: 4.6 Group: IN20001

Studentnr: 50060009 Grade: 4.6 Group: IS20001

Studentnr: 50060033 Grade: 4.4 Group: IN20001

Studentnr: 50060006 Grade: 3.8 Group: IB20001

Studentnr: 50060008 Grade: 3.4 Group: IN20001

Studentnr: 50060043 Grade: 3.3 Group: IN20001

Studentnr: 50060020 Grade: 3.3 Group: IT20001

Studentnr: 50060028 Grade: 3.2 Group: IN20001

Studentnr: 50060029 Grade: 3.1 Group: IS20001

Studentnr: 50060019 Grade: 3.0 Group: IS20001

Studentnr: 50060024 Grade: 2.7 Group: IS20001

Studentnr: 50060023 Grade: 2.3 Group: IN20001

Studentnr: 50060015 Grade: 2.1 Group: IT20001

Studentnr: 50060017 Grade: 2.1 Group: IG20001

Studentnr: 50060049 Grade: 1.9 Group: IS20001

Studentnr: 50060047 Grade: 1.8 Group: IG20001

Studentnr: 50060046 Grade: 1.6 Group: IB20001

Studentnr: 50060042 Grade: 1.5 Group: IG20001

Studentnr: 50060007 Grade: 1.4 Group: IG20001

Studentnr: 50060021 Grade: 1.3 Group: IB20001

Studentnr: 50060014 Grade: 1.2 Group: IS20001

Studentnr: 50060040 Grade: 1.2 Group: IT20001

### Sorteren op klas en studentnummer

Voor het sorteren op klas en studentnummer hebben we, zoals de opdracht beschrijft gebruik gemaakt van een bucket sort. We hebben voor elke sorteermethode een aparte klasse geschreven. Dit gaat om StudentNoComparator en GroupComparator. Beide klassen implemeteren de Comparator interface en overriden de compare methode van deze interface. Bij het comparen van studentnummers hebben we simpelweg de int attribuut studentNum vergeleken. Bij het vergelijken van klassen moesten we echter specifieke characters binnen de string attribuut groupID vergelijken. Dit hebben we gedaan door middel van charAt. Hierbij de gepaste code snippets:

StudentNoComparator()

@Override

public int compare(Student a, Student b) {

amountCompared++;

if (b.getStudentNum() < a.getStudentNum()) {

return 1;

} else if (b.getStudentNum() > a.getStudentNum()) {

return -1;

} else {

return 0;

}

}

(

GroupComparator()

@Override

public int compare(Group a, Group b) {

amountCompared++;

if (b.getGroupID().charAt(1) < a.getGroupID().charAt(1)) {

return 1;

} else if (b.getGroupID().charAt(1) > a.getGroupID().charAt(1)) {

return -1;

} else {

return 0;

}

}

Hierbij een lijst van 50 studenten gesorteerd op basis van klassen en studentnummer:

Total students: 50

Students per route: 10

Groups per route: 1

Total groups of students: 5

Studentnr: 50060001 Grade: 2.7 Group: IB20001

Studentnr: 50060006 Grade: 5.6 Group: IB20001

Studentnr: 50060011 Grade: 6.6 Group: IB20001

Studentnr: 50060016 Grade: 1.8 Group: IB20001

Studentnr: 50060021 Grade: 3.6 Group: IB20001

Studentnr: 50060026 Grade: 1.6 Group: IB20001

Studentnr: 50060031 Grade: 7.2 Group: IB20001

Studentnr: 50060036 Grade: 3.8 Group: IB20001

Studentnr: 50060041 Grade: 9.8 Group: IB20001

Studentnr: 50060046 Grade: 6.6 Group: IB20001

STUDENTS PER GROUP: 10

Studentnr: 50060002 Grade: 1.3 Group: IG20001

Studentnr: 50060007 Grade: 7.8 Group: IG20001

Studentnr: 50060012 Grade: 1.9 Group: IG20001

Studentnr: 50060017 Grade: 2.7 Group: IG20001

Studentnr: 50060022 Grade: 8.8 Group: IG20001

Studentnr: 50060027 Grade: 4.2 Group: IG20001

Studentnr: 50060032 Grade: 5.6 Group: IG20001

Studentnr: 50060037 Grade: 10.0 Group: IG20001

Studentnr: 50060042 Grade: 8.4 Group: IG20001

Studentnr: 50060047 Grade: 4.5 Group: IG20001

STUDENTS PER GROUP: 10

Studentnr: 50060003 Grade: 1.5 Group: IN20001

Studentnr: 50060008 Grade: 9.7 Group: IN20001

Studentnr: 50060013 Grade: 2.0 Group: IN20001

Studentnr: 50060018 Grade: 9.9 Group: IN20001

Studentnr: 50060023 Grade: 9.0 Group: IN20001

Studentnr: 50060028 Grade: 8.9 Group: IN20001

Studentnr: 50060033 Grade: 2.7 Group: IN20001

Studentnr: 50060038 Grade: 5.7 Group: IN20001

Studentnr: 50060043 Grade: 8.6 Group: IN20001

Studentnr: 50060048 Grade: 8.9 Group: IN20001

STUDENTS PER GROUP: 10

Studentnr: 50060004 Grade: 8.7 Group: IS20001

Studentnr: 50060009 Grade: 2.6 Group: IS20001

Studentnr: 50060014 Grade: 7.4 Group: IS20001

Studentnr: 50060019 Grade: 6.5 Group: IS20001

Studentnr: 50060024 Grade: 3.8 Group: IS20001

Studentnr: 50060029 Grade: 6.6 Group: IS20001

Studentnr: 50060034 Grade: 8.2 Group: IS20001

Studentnr: 50060039 Grade: 6.9 Group: IS20001

Studentnr: 50060044 Grade: 5.3 Group: IS20001

Studentnr: 50060049 Grade: 4.3 Group: IS20001

STUDENTS PER GROUP: 10

Studentnr: 50060005 Grade: 4.4 Group: IT20001

Studentnr: 50060010 Grade: 2.1 Group: IT20001

Studentnr: 50060015 Grade: 10.0 Group: IT20001

Studentnr: 50060020 Grade: 7.6 Group: IT20001

Studentnr: 50060025 Grade: 8.6 Group: IT20001

Studentnr: 50060030 Grade: 3.9 Group: IT20001

Studentnr: 50060035 Grade: 6.5 Group: IT20001

Studentnr: 50060040 Grade: 4.1 Group: IT20001

Studentnr: 50060045 Grade: 4.8 Group: IT20001

Studentnr: 50060050 Grade: 2.7 Group: IT20001

STUDENTS PER GROUP: 10

Amount of grade comparisons: 0

Amount of student comparisons: 45

Amount of group comparisons: 4

GroupComparator groupC = new GroupComparator();

Arrays.sort(gen.getGroups(), groupC);

for (Group g : gen.getGroups()) {

// comparator to sort studentnr

StudentNoComparator compStudentNo = new StudentNoComparator();

Collections.sort(g, compStudentNo);

for (int i = 0; i < g.size(); i++) {

System.out.println(g.get(i));

}

System.out.println("STUDENTS PER GROUP: " + g.size());

}

De bovenstaande code snippet staat in main en wordt gebruikt om methoden aan te roepen die 1) klassen sorteren, en 2) de studenten per klas op studentnummer sorteren.

# Meet de efficiëntie

## Sorteren

We hebben onderzoek gedaan naar het sorteren op cijfer en het groeperen per klas met sortering op studentnummer. Er is getest op 10.000, 20.000, 40.000, 80.000 en 160.000 studenten met hun resultaten. Om een goed beeld te krijgen is uit verscheidene testcycli de gemiddelde meetwaarde gebruikt. Hiervoor hebben we System.nanoTime gebruikt. Vervolgens hebben we de nanoTime omgezet naar milliseconden om een wat leesbaardere output terug te krijgen. Starttijd en een eindtijd werden bijgevoegd voor en na de uitvoering van een sorteer methode, respectievelijk. De runtime van het proces hebben we berekend door de starttijd van de eindtijd af te trekken.

Meting van het sorteren op cijfer:

long startT = System.nanoTime();

gen.sortByGrade(gen.getStudents()); // Sort by high to low grades

long endT = System.nanoTime();

long diff = endT - startT;

long nanoToMill = TimeUnit.NANOSECONDS.toMillis(diff);

Bovenstaande code is ook voor het groeperen per klas met sortering op studentnummer toegepast.

Resultaten in grafieken en tabel:

|  |  |
| --- | --- |
|  | C:\Users\Astrophe\AppData\Local\Microsoft\Windows\INetCache\Content.Word\GroupSortChart.png |
|  | |

Zoals is te zien in de linker grafiek in de figuur hierboven groeit de tijd kwadratisch. Verdubbeling van data vereist ruwweg twee keer zoveel handelingen. Hieruit kunnen we opmaken dat de bubblesort voor het sorteren van studenten op basis van de cijfers, een big O(n²) heeft.

Anderzijds heeft mergesort, de sorteermethode die we voor groupsort gebruiken, een tijdscomplexiteit van O(nLog n)

## Vergelijkingen

Het aantal vergelijkingen tussen twee studenten en twee klassen worden gemeten door een opgehoogde int variabele. Dit wordt gedaan in de methode compare() van de comparators: StudentNo en Group.

In ons onderzoek is gebleken wanneer de hoeveelheid studenten verdubbeld wordt, zijn het aantal vergelijkingen ook verdubbeld. O(n^2)

public static int amountCompared = 0;

@Override

public int compare(Group a, Group b) {

amountCompared++;

if (b.getGroupID().charAt(1) < a.getGroupID().charAt(1)) {

return 1;

} else if (b.getGroupID().charAt(1) > a.getGroupID().charAt(1)) {

return -1;

} else {

return 0;

}

}

|  |  |
| --- | --- |
| C:\Users\Astrophe\AppData\Local\Microsoft\Windows\INetCache\Content.Word\GroupComparion.png | C:\Users\Astrophe\AppData\Local\Microsoft\Windows\INetCache\Content.Word\StudentComparison.png |
|  | |

# Conclusie

Uit de statistieken / onderzoek blijkt dat het sorteren op cijfer veel langer duurt dan het groeperen per klas met gesorteerd op studentnummer.

Daarentegen is de implementatie van een comparator (mergesort) makkelijker te implementeren dan het toepassen van een eigen sorteer algoritme, in dit geval de (werking van een) bubblesort.

Efficiënte manier? Kies voor mergesort!