Universiteti Politeknik i Tiranës

Fakulteti i Teknologjisë së Informacionit

Dega: Inxhinieri Informatike

Grupi: III-B

Viti akademik 2024-2025

Punë laboratori nr 1

Lënda: Algoritmike dhe programim i avancuar

Punoi: Piro Gjidhima

Pranoi:Msc Alba Haveriku

Tema:Krahasimi i algoritmave të renditjes

Implementoni dhe ekzekutoni në Java,algoritmat në pikën A OSE në pikën B

A)Insertion Sort,Selection Sort dhe Shell Sort.

B)Merge Sort dhe Quick Sort

Për të krahasuar performancën e tyre do të përdoren si input të dhëna numerike të gjeneruara random dhe të ruajtura në një file.

- Gjeneroni numra random dhe ruajini në file;
- Provoni të krijoni 5 file të ndryshme me një sasi të ndryshme numrash. P.sh. 10, 100, 1000, 10000...
- Verifikoni deri në sa numra mund të renditen në një kohë të pranueshme nga secili algoritëm.
- Argumentoni në lidhje me kompleksitetin në kohë dhe në hapësire të secilit algoritëm.
- Sa është koha e ekzekutimit për secilin algoritëm në secilin file.
- Vendosini të dhënat e eksperimenteve në një tabelë dhe krijoni një grafik me këto të dhëna. Arsyetoni cili nga algoritmat është më i mirë.

Zgjidhje

Kodi i algoritmave , i shkrimit dhe leximit te numrave random nga skedar tekst

```
import java.io.File;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class lab1 {
   private static int partition(int[] a, int low, int high)
   {
      int pivot = a[high];
      int i = (low-1);
      for (int j=low; j<high; j++)
      {
         if (a[j] <= pivot)
         {
            i++;

            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }

      int temp = a[i+1];
      a[i+1] = a[high];
      a[high] = temp;

      return i+1;
   }
   private static void quicksort(int[] a, int l, int h)
   {
      if (l < h)
      {
         int pivot = partition(a, l, h);
         quicksort(a, l, pivot-1);
         quicksort(a, pivot+1, h);
      }
   }


public static void QuickSort(int[] a){
      quicksort(a,0,a.length-1);
```

```
}

private static void merge(int[] array, int left, int middle, int right) {
   int n1 = middle - left + 1;
   int n2 = right - middle;

   int[] leftArray = new int[n1];
   int[] rightArray = new int[n2];

   System.arraycopy(array, left, leftArray, 0, n1);
   System.arraycopy(array, middle + 1, rightArray, 0, n2);

   int i = 0, j = 0;
   int k = left;

   while (i < n1 && j < n2) {
      if (leftArray[i] <= rightArray[j]) {
         array[k] = leftArray[i];
         i++;
      } else {
         array[k] = rightArray[j];
         j++;
      }
      k++;
   }
   while (i < n1) {
      array[k++] = leftArray[i++];
   }
   while (j < n2) {
      array[k++] = rightArray[j++];
   }
}

private static void mergesort(int[] array, int left, int right) {
   if (left < right) {
      int middle = (left + right) / 2;

      mergesort(array, left, middle);
      mergesort(array, middle + 1, right);
      merge(array, left, middle, right);
   }
}
public static void MergeSort(int[] a){
   mergesort(a,0,a.length-1);
}

public static void create(int n,String path) throws FileNotFoundException {
   Random random  = new Random();
   int[] a = new int[n];
```

```
for (int i = 0;i<a.length;i++){
            a[i] = random.nextInt();
        }
        try (PrintWriter w = new PrintWriter(path)) {
            for (int i : a) {
                w.write(i + " ");
            }
        }
    }

    public static int[] read(String path){
        List<Integer> numbers = new ArrayList<>();

        try {
            File file = new File(path);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNext()) {
                if (scanner.hasNextInt()) {
                    numbers.add(scanner.nextInt());
                } else {
                    scanner.next();
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        }
        int[] a = new int[numbers.size()];
        for(int i = 0;i<a.length;i++) {
            a[i] = numbers.get(i);

        }
        return a;
    }

    public static void main(String[] args) throws FileNotFoundException {

        int j = 1;
        for (int i = 10 ; i<1000001;i*=10){
            create(i,"src/file"+ j + ".txt");
            j++;
        }
        for(int i = 1;i<7;i++) {

            int[] a = read("src/file" + i + ".txt");
            long start = System.nanoTime();
            MergeSort(a);
            long end = System.nanoTime();
            System.out.println("Iteration: " + i + ", MergeSort time " + (end - start) + " ns");
            int[] b = read("src/file" + i + ".txt");
            start = System.nanoTime();
            QuickSort(b);
```

```
            end = System.nanoTime();
            System.out.println("Iteration: " + i + ", QuickSort time " + (end - start) + " ns");
        }
    }
}
```

Ky kod është përgjegjës për krijimin , shtimit dhe leximin e File-ve të krijuar me numra random, po ashtu këtu janë implementuar algoritmet e MergeSort (ky implementim krijon array ndihmes) dhe QuickSort. Përmes njohurive teorike dihet qe kompleksiteti ne rastin mesatar ne te dy algoritmat është $O(nlogn)$.Mergjithate nga mënyra e implementimit,makina ku ekzekutohet dhe optimizimet e mundshme ne praktike vihet re nje diference ne kohen e ekzekutimit.

(Shënim: ne metodën randInt() nuk kam vendosur nje parametër me qellim qe numrat e gjeneruar te jene thuajse te ndryshëm dhe te ulim rastin e vlerave te përsëritura.)

Kodi I paraqitjes grafike te te dhenave me ate te librarise JFreeChart

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.text.DecimalFormat;

public class SortingTimeChart extends JFrame {

    public SortingTimeChart() {

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();


        Object[][] data = {
                {"Iteration 1", 8800, 7800},
                {"Iteration 2", 42900, 24800},
                {"Iteration 3", 630000, 439700},
                {"Iteration 4", 1839900, 1320700},
                {"Iteration 5", 17609800, 9115600},
                {"Iteration 6", 145100800, 86644400}
        };

        for (Object[] row : data) {
            String iteration = (String) row[0];
```

```
            dataset.addValue((Number) row[1], "MergeSort", iteration);
            dataset.addValue((Number) row[2], "QuickSort", iteration);
        }

        JFreeChart lineChart = ChartFactory.createLineChart(
                "MergeSort vs QuickSort Execution Time",
                "Iteration",
                "Time (ns)",
                dataset,
                PlotOrientation.VERTICAL,
                true, true, false);

        CategoryPlot plot = lineChart.getCategoryPlot();
        NumberAxis yAxis = (NumberAxis) plot.getRangeAxis();
        yAxis.setNumberFormatOverride(new DecimalFormat("#,###"));


        ChartPanel chartPanel = new ChartPanel(lineChart);
        chartPanel.setPreferredSize(new Dimension(800, 400));


        String[] columnNames = {"Iteration", "MergeSort Time (ns)", "QuickSort Time (ns)"};
        DefaultTableModel tableModel = new DefaultTableModel(data, columnNames);
        JTable table = new JTable(tableModel);
        JScrollPane tableScrollPane = new JScrollPane(table);
        table.setFillsViewportHeight(true);
        table.setPreferredScrollableViewportSize(new Dimension(800, 150));


        JPanel contentPanel = new JPanel(new BorderLayout());
        contentPanel.add(chartPanel, BorderLayout.CENTER);
        contentPanel.add(tableScrollPane, BorderLayout.SOUTH);
        setContentPane(contentPanel);


        setTitle("Sorting Algorithm Comparison");
        setSize(800, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SortingTimeChart example = new SortingTimeChart();
            example.setVisible(true);
        });
    }
}
```

Me siper është paraqitur kodi I cili krijon grafikun dhe tabelen ku jave hedhur te dhenat e nxjerra nga ekzekutimi I kodit.Eshte perdorur libraria JFreeChart bashkë me Java Swing.

Rezultatet

Kam bërë ekzekutimin e kodit 2 herë me qellim qe te shihja a kishin perputhshmeri me njera-tjetren rezultatet e perftuara

Hera 1

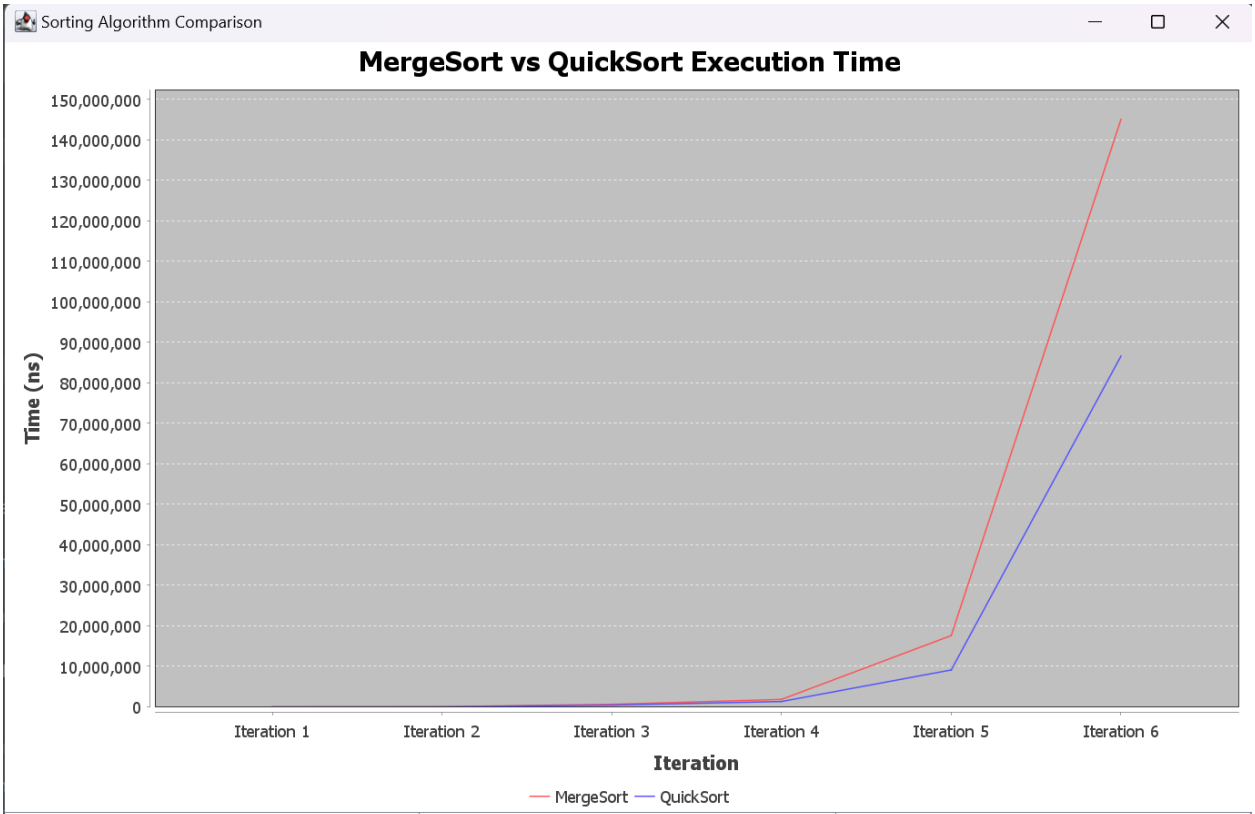
```
Iteration: 1, MergeSort time 8800 ns
Iteration: 1, QuickSort time 7800 ns
Iteration: 2, MergeSort time 42900 ns
Iteration: 2, QuickSort time 24800 ns
Iteration: 3, MergeSort time 630000 ns
Iteration: 3, QuickSort time 439700 ns
Iteration: 4, MergeSort time 1839900 ns
Iteration: 4, QuickSort time 1320700 ns
Iteration: 5, MergeSort time 17609800 ns
Iteration: 5, QuickSort time 9115600 ns
Iteration: 6, MergeSort time 145100800 ns
Iteration: 6, QuickSort time 86644400 ns
```

Hera 2

```
Iteration: 1, MergeSort time 13300 ns
Iteration: 1, QuickSort time 20000 ns
Iteration: 2, MergeSort time 80400 ns
Iteration: 2, QuickSort time 38400 ns
Iteration: 3, MergeSort time 817800 ns
Iteration: 3, QuickSort time 539300 ns
Iteration: 4, MergeSort time 1446900 ns
Iteration: 4, QuickSort time 957500 ns
Iteration: 5, MergeSort time 17832200 ns
Iteration: 5, QuickSort time 8131000 ns
Iteration: 6, MergeSort time 160768100 ns
Iteration: 6, QuickSort time 82676000 ns
```

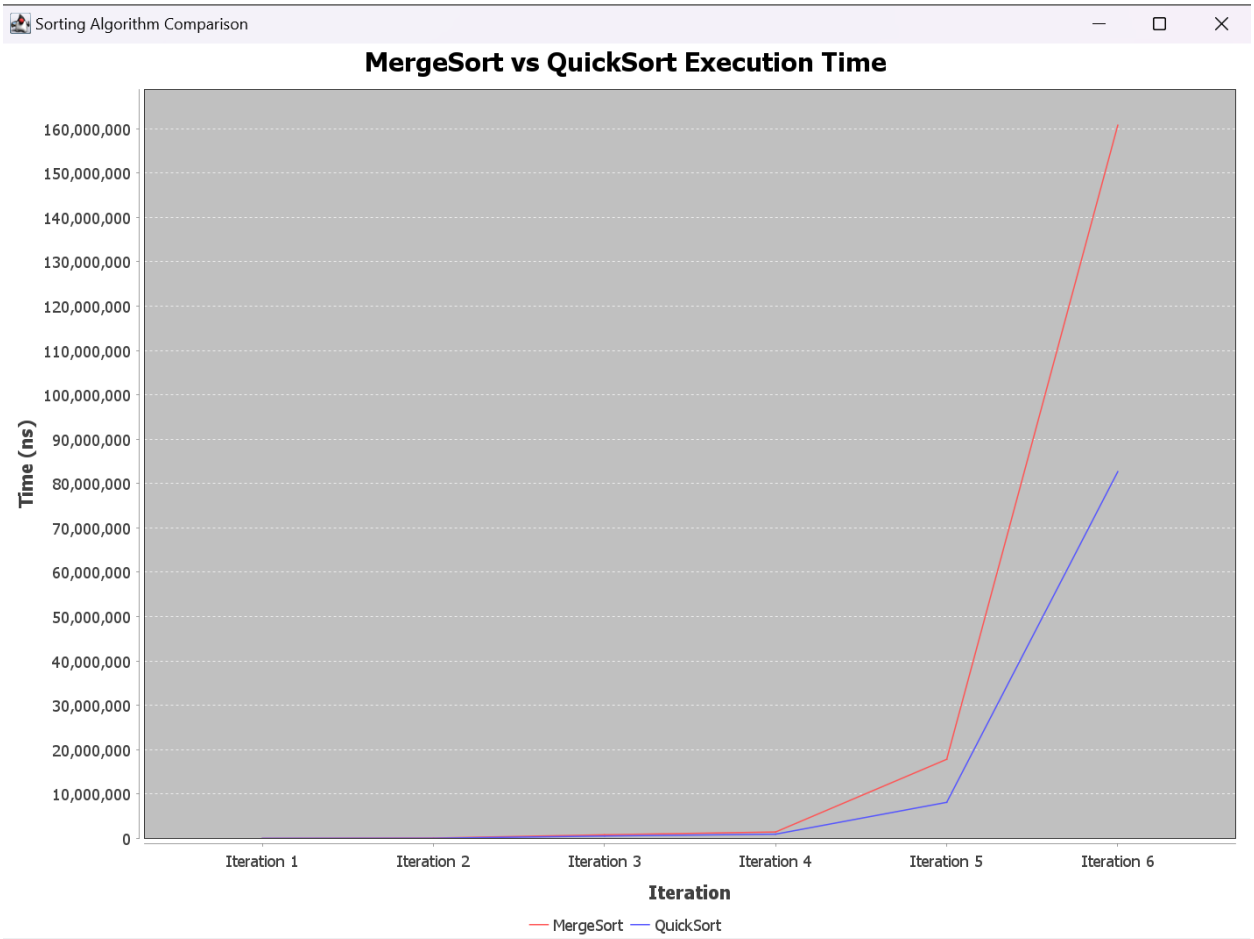
Praqitja grafike

Hera 1



Iteration	MergeSort Time (ns)	QuickSort Time (ns)
Iteration 1	8800	7800
Iteration 2	42900	24800
Iteration 3	630000	439700
Iteration 4	1839900	1320700
Iteration 5	17609800	9115600
Iteration 6	145100800	86644400

Hera 2

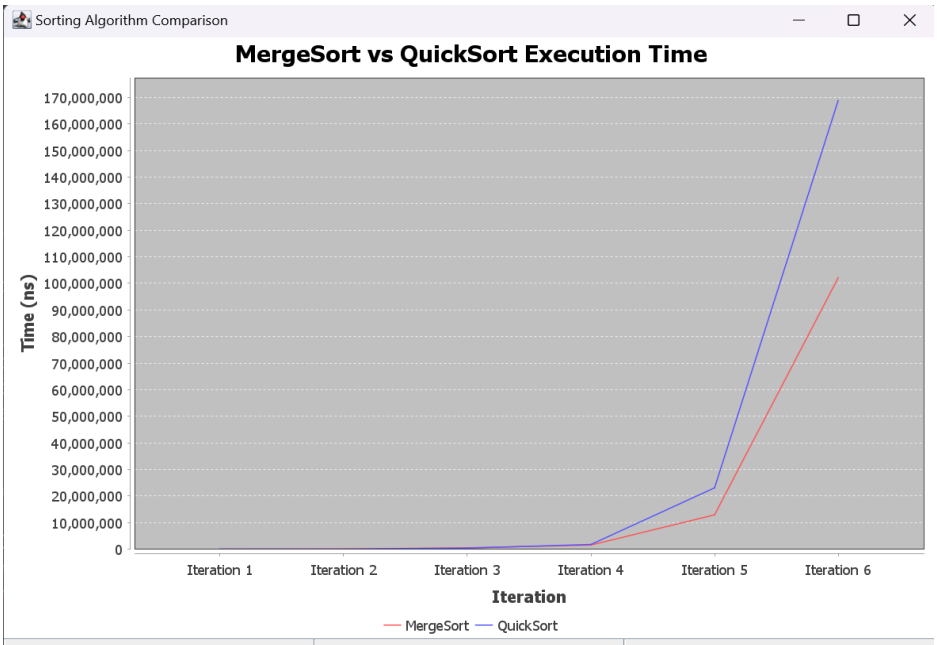


Iteration	MergeSort Time (ns)	QuickSort Time (ns)
Iteration 1	13300	20000
Iteration 2	80400	38400
Iteration 3	817800	539300
Iteration 4	1446900	957500
Iteration 5	17832200	8131000
Iteration 6	160768100	82676000

Dallohet prej rezultateve qe algoritmi QuickSort eshte me i shpejte se MergeSort megjithese nga ana teorike kane kompleksitet te njejte ne rastin mesatar.Dallohet prej kurbes qe QuickSort eshte me I shpjete sa me shume rritet madhesia e inputit. Mirepo si garanci kam vendosur qe mos te kemi interval numrash, ne menyre qe te kemi sa me pak perseritje, ne mos fare. Po nese do te kishte perseritje? Versioni I implementuar I QuickSort nuk eshte optimizuar per ate rast.Presim qe te kete kohe ekzekutimi me te keqe sesa MergeSort sepse ka me shume gjasa qe elementet qe kapim si pivot mund te jene vlera ekstreme te vektorit me numra dhe ka kosto te larte ne kohe.Per te provuar kete gje kam bere nje ekzekutim te trete me interval numrash deri ne 200. Per file me 1000000 elemente do te kete shume vlera te perseritura.

Hera 3

```
Iteration: 1, MergeSort time 8200 ns
Iteration: 1, QuickSort time 13100 ns
Iteration: 2, MergeSort time 41500 ns
Iteration: 2, QuickSort time 27000 ns
Iteration: 3, MergeSort time 491500 ns
Iteration: 3, QuickSort time 366700 ns
Iteration: 4, MergeSort time 1516500 ns
Iteration: 4, QuickSort time 1739400 ns
Iteration: 5, MergeSort time 12889000 ns
Iteration: 5, QuickSort time 23058200 ns
Iteration: 6, MergeSort time 102137300 ns
Iteration: 6, QuickSort time 1688342000 ns
```



Iteration	MergeSort Time (ns)	QuickSort Time (ns)
Iteration 1	8200	13100
Iteration 2	41500	27000
Iteration 3	491500	366700
Iteration 4	1516500	1739400
Iteration 5	12889000	23058200
Iteration 6	102137300	168834200

Nga grafiku dhe tabela dallojme qe QuickSort eshte I avashte se MergeSort ne rastin kur kemi shume vlera te perseritura sidomos ne vektoret me me shume se 1000 elemente. Prandaj eshte e nevojshme implementimi I optimizimeve si 3-way QuickSort apo dhe per MergeSort.

Si perfundim te dyja algoritmet jane te shpejta, por per te dhena random dhe pa perseritje QuickSort del me I shpjete sesa MergeSort.