LAPORAN PRATIKUM STRUKTUR DATA PEKAN 8
VISUALISASI ALGORITMA PENGURUTAN (SORTING) STEP-BY-STEP

MATA KULIAH

DOSEN PENGAMPU :

DR. WAHYUDI, S.T, M,T

OLEH:

AUFAN TAUFIQURRAHMAN

NIM 2411532011

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

BAB 1
PENDAHULUAN

A. LATAR BELAKANG

Dalam ilmu komputer, pengurutan (sorting) adalah salah satu operasi paling fundamental dan sering digunakan. Algoritma pengurutan adalah prosedur untuk menyusun kembali elemen-elemen dari sebuah daftar atau array dalam urutan tertentu, seperti numerik (menaik atau menurun) atau leksikografis. Efisiensi sebuah program seringkali bergantung pada pemilihan algoritma pengurutan yang tepat, karena setiap algoritma memiliki karakteristik, kelebihan, dan kekurangan yang berbeda-beda dalam hal kecepatan (kompleksitas waktu) dan penggunaan memori (kompleksitas ruang).

Memahami cara kerja internal dari berbagai algoritma pengurutan seperti Bubble Sort, Merge Sort, Quick Sort, dan Shell Sort terkadang menjadi tantangan. Konsep-konsep seperti iterasi, rekursi, partisi, dan perbandingan bisa bersifat abstrak. Oleh karena itu, visualisasi menjadi alat bantu pembelajaran yang sangat efektif. Dengan memvisualisasikan setiap langkah—setiap perbandingan dan pertukaran elemen—secara grafis, proses yang kompleks menjadi lebih mudah dipahami dan dianalisis.

Praktikum ini bertujuan untuk mengimplementasikan empat algoritma pengurutan yang berbeda ke dalam sebuah aplikasi Java dengan antarmuka pengguna grafis (GUI). Aplikasi ini dirancang untuk tidak hanya mengurutkan sekumpulan data, tetapi juga menampilkan proses pengurutan tersebut langkah demi langkah, sehingga memberikan pemahaman yang mendalam tentang mekanisme internal setiap algoritma.

B. TUJUAN

Adapun tujuan dari pelaksanaan praktikum ini adalah sebagai berikut:

1. Memahami konsep dan prinsip kerja dari algoritma Bubble Sort, Merge Sort, Quick Sort, dan Shell Sort.
2. Mampu mengimplementasikan keempat algoritma pengurutan tersebut menggunakan bahasa pemrograman Java.
3. Mampu membangun antarmuka pengguna grafis (GUI) menggunakan Java Swing untuk memvisualisasikan proses kerja algoritma.

4. Menganalisis dan membandingkan proses setiap algoritma secara visual untuk memahami perbedaan, kelebihan, dan kekurangan masing-masing.

BAB 2

DASAR TEORI

A. BUBBLE SORT

Bubble Sort adalah algoritma pengurutan paling sederhana. Algoritma ini bekerja dengan berulang kali menelusuri daftar, membandingkan setiap pasang elemen yang bersebelahan, dan menukarnya jika urutannya salah. Penelusuran ini diulangi sampai tidak ada lagi pertukaran yang perlu dilakukan, yang menandakan bahwa daftar telah diurutkan. Meskipun sederhana, Bubble Sort sangat tidak efisien untuk daftar besar dan kinerjanya dianggap buruk ($O(n^2)$).

B. MERGE SORT

Merge Sort adalah algoritma pengurutan yang menggunakan pendekatan Divide and Conquer (Bagi dan Kuasai). Cara kerjanya adalah:

1. Divide: Membagi array menjadi dua bagian (sub-array) yang sama besar secara rekursif hingga setiap sub-array hanya berisi satu elemen.
2. Conquer: Menggabungkan (merge) kembali sub-array yang telah diurutkan. Proses penggabungan membandingkan elemen dari kedua sub-array dan menempatkannya kembali ke array utama dalam urutan yang benar.

Merge Sort memiliki kompleksitas waktu yang stabil, yaitu O(n log n) dalam semua kasus (terbaik, rata-rata, dan terburuk), namun membutuhkan ruang memori tambahan untuk proses penggabungan.

C. QUICK SHORT

Quick Sort juga merupakan algoritma *Divide and Conquer*. Algoritma ini memilih satu elemen sebagai 'pivot' dan mempartisi array menjadi dua sub-array. Elemen-elemen yang lebih kecil dari pivot dipindahkan ke sebelah kiri pivot, dan elemen-elemen yang lebih besar dipindahkan ke sebelah kanan. Proses ini kemudian diulangi secara rekursif untuk sub-array di sebelah kiri dan kanan pivot. Kompleksitas waktu rata-ratanya adalah O(n log n), menjadikannya salah satu algoritma tercepat.

Namun, pada kasus terburuk (misalnya saat pivot selalu elemen terkecil atau terbesar), kompleksitasnya bisa mencapai $O(n^2)$.

D. SHELL SHORT

Shell Sort adalah pengembangan dari Insertion Sort. Ide utamanya adalah memungkinkan pertukaran elemen yang posisinya berjauhan. Algoritma ini bekerja dengan mengurutkan elemen-elemen yang terpisah oleh jarak (gap) tertentu. Proses ini diulang dengan mengurangi nilai gap secara bertahap hingga gap menjadi 1. Ketika gap sama dengan 1, algoritma ini pada dasarnya melakukan Insertion Sort biasa pada array yang sudah "hampir terurut", membuatnya jauh lebih cepat daripada Insertion Sort standar.

E. JAVA SWING

Java Swing adalah *toolkit* (kumpulan perangkat) untuk membuat antarmuka pengguna grafis (GUI) untuk aplikasi Java. Swing menyediakan komponen-komponen seperti jendela (JFrame), panel (JPanel), tombol (JButton), label (JLabel), dan area teks (JTextArea, JTextField) yang independen dari platform, memungkinkan aplikasi memiliki tampilan dan nuansa yang konsisten di berbagai sistem operasi.

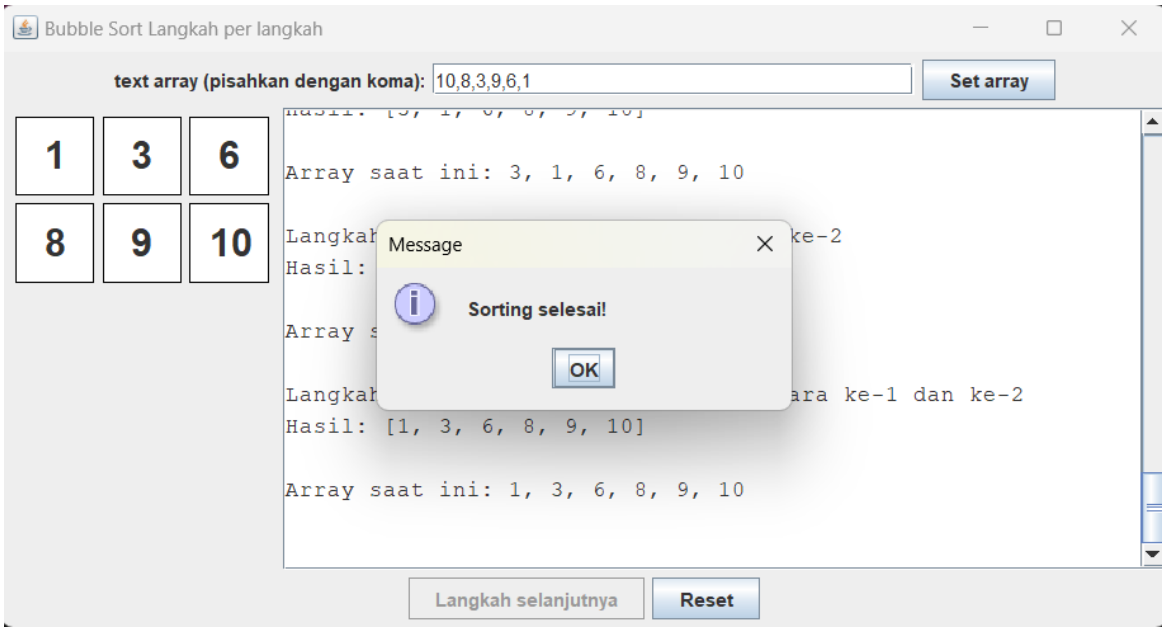
BAB 3
PROSEDUR PRAKTIKUM

Prosedur praktikum ini adalah membuat empat program Java terpisah untuk setiap algoritma pengurutan. Setiap program memiliki GUI untuk memasukkan data array, serta tombol kontrol untuk menjalankan proses pengurutan langkah demi langkah.

A. Class BubbleSort.java

Program ini memvisualisasikan algoritma Bubble Sort. Pengguna memasukkan deret angka yang dipisahkan koma, lalu menekan tombol "Langkah

selanjutnya" untuk melihat setiap perbandingan dan pertukaran yang terjadi.



1. Package dan Import

```
import java.awt.*;
import java.util.Arrays;
import javax.swing.*;
```

- `package pekan8;`: Mendeklarasikan bahwa kelas ini berada dalam paket `pekan8`.
- `import java.awt.*;`: Mengimpor semua kelas dari paket AWT (Abstract Window Toolkit) untuk komponen dasar GUI dan layout.
- `import java.util.Arrays;`: Mengimpor kelas `Arrays` untuk fungsionalitas terkait array, seperti `Arrays.toString()`.
- `import javax.swing.*;`: Mengimpor semua kelas dari paket Swing untuk komponen GUI yang lebih modern.

2. Deklarasi Kelas dan Variabel Instance

```
public class BubbleSortGUI extends JFrame {

    // Aufan_Taufiqurrahman
    // 2411532011
    private static final long serialVersionUID = 1L;
    private int[] array;
    private JLabel[] labelArray;
    private JButton stepButton, resetButton, setButton;
    private JTextField inputField;
    private JPanel panelArray;
    private JTextArea stepArea;

    private int i = 0, j;
    private boolean sorting = false;
    private int stepCount = 1;
```

- `public class BubbleSortGUI extends JFrame`: Mendefinisikan kelas `BubbleSortGUI` yang merupakan turunan dari `JFrame`, artinya kelas ini adalah sebuah jendela aplikasi.
- `private int[] array;`: Array integer untuk menyimpan data yang akan diurutkan.
- `private JLabel[] labelArray;`: Array `JLabel` untuk memvisualisasikan setiap elemen dari `array`.
- `stepButton, resetButton, setButton`: Tombol untuk kontrol (langkah berikutnya, reset, dan set array).
- `inputField`: Kolom teks untuk memasukkan data array.
- `panelArray`: Panel untuk menampung `labelArray`.
- `stepArea`: Area teks untuk menampilkan log atau penjelasan dari setiap langkah.
- `i, j`: Variabel iterator untuk loop Bubble Sort.
- `sorting`: Flag boolean untuk menandakan apakah proses pengurutan sedang aktif.
- `stepCount`: Penghitung langkah untuk ditampilkan di log.

3. Konstruktor BubbleSortGUI()

```
public BubbleSortGUI() {
    setTitle(title:"Bubble Sort Langkah per langkah");
    setSize(width:750, height:400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(c:null);
    setLayout(new BorderLayout());

    JPanel textPanel = new JPanel(new FlowLayout());
    inputField = new JTextField(columns:30);
    setButton = new JButton(text:"Set array");
    textPanel.add(new JLabel(text:"text array (pisahkan dengan koma):"));
    textPanel.add(inputField);
    textPanel.add(setButton);

    panelArray = new JPanel();
    panelArray.setLayout(new FlowLayout());

    JPanel controlPanel = new JPanel();
    stepButton = new JButton(text:"Langkah selanjutnya");
    resetButton = new JButton(text:"Reset");
    stepButton.setEnabled(b:false);
    controlPanel.add(stepButton);
    controlPanel.add(resetButton);

    stepArea = new JTextArea(rows:8, columns:60);
    stepArea.setEditable(b:false);
    stepArea.setFont(new Font(name:"Monospaced", Font.PLAIN, size:14));
    JScrollPane scrollPane = new JScrollPane(stepArea);

    add(textPanel, BorderLayout.NORTH);
    add(panelArray, BorderLayout.CENTER);
    add(controlPanel, BorderLayout.SOUTH);
```

- setTitle, setSize, setDefaultCloseOperation: Mengatur properti dasar jendela aplikasi.
- Di dalam konstruktor, semua komponen GUI (panel, tombol, field teks) dibuat, dikonfigurasi, dan ditambahkan ke JFrame.
- addActionListener(...): Menambahkan event listener ke setiap tombol. Ketika tombol diklik, metode yang sesuai akan dieksekusi (misalnya, klik setButton akan memanggil setArrayFromText()).

4. Method setArrayFromText()

```
private void setArrayFromText() {
    String text = inputField.getText().trim();
    if (text.isEmpty()) return;
    String[] parts = text.split(regex:",");
    array = new int[parts.length];

    try {···
    } catch (NumberFormatException e) {···

    i = 0;
    j = 0;
    stepCount = 1;
    sorting = true;
    stepButton.setEnabled(b:true);
    stepArea.setText(t:"");
    labelArray = new JLabel[parts.length];
    panelArray.removeAll();
    resetButton.setEnabled(b:true);

    for (int k = 0; k < array.length; k++) {···

    panelArray.revalidate();
    panelArray.repaint();
}
```

- Metode ini dipanggil saat tombol "Set array" diklik.
- Ia mengambil teks dari inputField, memisahkannya berdasarkan koma (split(",")), dan mengonversinya menjadi array integer.
- Jika input tidak valid, sebuah pesan error akan ditampilkan.
- Selanjutnya, metode ini mereset variabel-variabel state seperti i, j, dan stepCount.
- Terakhir, ia membuat dan menampilkan JLabel untuk setiap elemen array di panelArray. revalidate() dan repaint() digunakan untuk memastikan panel diperbarui secara visual.

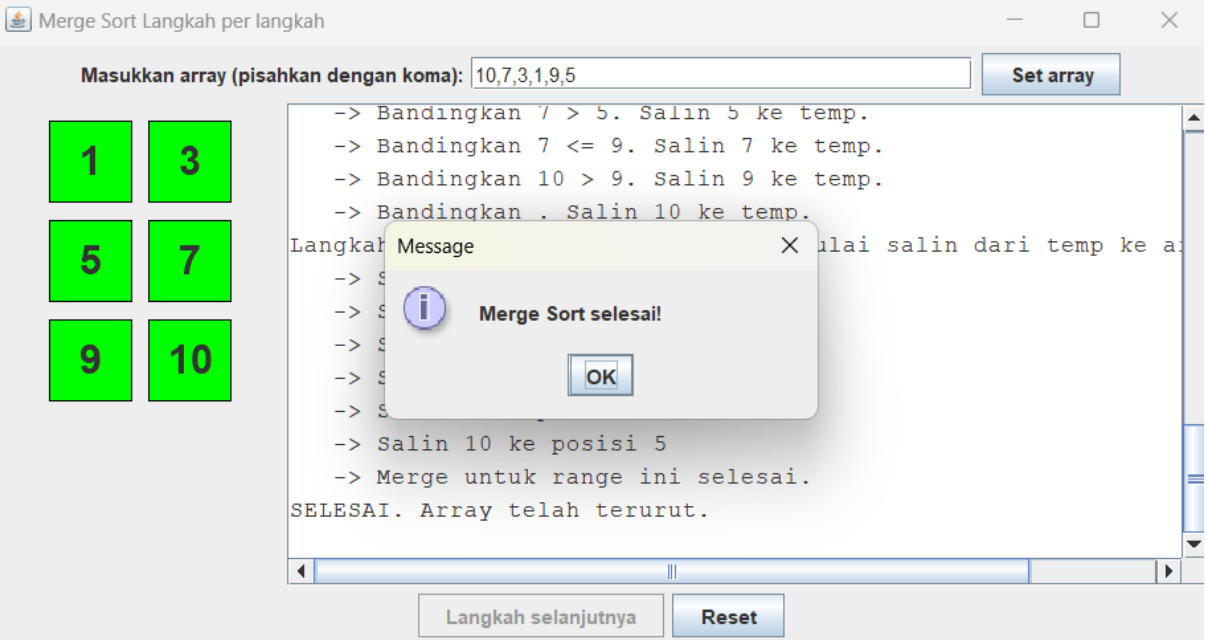
5. Metode performStep() (Logika Inti Bubble Sort)

```
private void performStep() {
    if (!sorting || i >= array.length) {
        sorting = false;
        stepButton.setEnabled(b:false);
        resetHighlights(); // Tambahkan ini
        JOptionPane.showMessageDialog(this, message:"Sorting selesai!");
        return;
    }

    resetHighlights();
    StringBuilder stepLog = new StringBuilder();
    labelArray[j].setBackground(Color.CYAN);
    labelArray[j + 1].setBackground(Color.CYAN);

    if (array[j] > array[j + 1]) {
        int temp = array[j];
        array[j] = array[j + 1];
        array[j + 1] = temp;
        labelArray[j].setBackground(Color.RED);
        labelArray[j + 1].setBackground(Color.RED);
        stepLog.append(str:"Langkah ").append(stepCount).append(str:": Menukar elemen ke-")
               .append(j + 1).append(str:" dan ke-").append(j + 2).append(str:"\n")
               .append(str:"Hasil: ").append(Arrays.toString(array)).append(str:"\n\n");
    } else {
        stepLog.append(str:"Langkah ").append(stepCount).append(str:": Tidak ada pertukaran antara ke-")
               .append(j + 1).append(str:" dan ke-").append(j + 2).append(str:"\n")
               .append(str:"Hasil: ").append(Arrays.toString(array)).append(str:"\n\n");
    }

    stepLog.append(str:"Array saat ini: ").append(arrayToString()).append(str:"\n\n");
    stepArea.append(stepLog.toString());
    } else {
        stepLog.append(str:"Langkah ").append(stepCount).append(str:": Tidak ada pertukaran antara ke-")
               .append(j + 1).append(str:" dan ke-").append(j + 2).append(str:"\n")
               .append(str:"Hasil: ").append(Arrays.toString(array)).append(str:"\n\n");
    }

    stepLog.append(str:"Array saat ini: ").append(arrayToString()).append(str:"\n\n");
    stepArea.append(stepLog.toString());
    updateLabels();

    j++;
    if (j >= array.length - i - 1) {
        j = 0;
        i++;
    }
    stepCount++;

    if (i >= array.length) {
        sorting = false;
        stepButton.setEnabled(b:false);
        resetHighlights(); // Tambahkan ini
        JOptionPane.showMessageDialog(this, message:"Sorting selesai!");
    }
}
```

- Ini adalah jantung dari visualisasi. Setiap klik "Langkah selanjutnya" memanggil metode ini.
- resetHighlights(): Menghapus warna latar belakang dari langkah sebelumnya.
- labelArray[j].setBackground(Color.CYAN): Menyorot dua elemen yang sedang dibandingkan dengan warna sian.

- if (array[j] > array[j + 1]): Logika inti Bubble Sort. Jika elemen kiri lebih besar dari kanan, mereka ditukar.
- Jika pertukaran terjadi, elemen yang ditukar disorot dengan warna merah.
- stepArea.append(...): Setiap tindakan (membandingkan, menukar, atau tidak) dicatat di area teks.
- updateLabels(): Memperbarui teks pada JLabel agar sesuai dengan kondisi array saat ini.
- j++ dan i++: Logika untuk memajukan iterator j (posisi saat ini) dan i (jumlah elemen yang sudah berada di posisi akhir yang benar).

B. CLASS MERGESORT.JAVA



1. Variabel Kunci dan Struktur Data

```
private boolean isMerging = false;
private int[] temp;
private int left, mid, right;
private boolean copying = false;
private int k = 0;
private java.util.Queue<int[]> mergeQueue = new java.util.LinkedList<>();
```

- `isMerging`: Flag untuk menandakan apakah sedang dalam fase penggabungan sub-array.
- `temp`: Array sementara yang dibutuhkan untuk proses penggabungan.
- `left, mid, right`: Menandai batas-batas sub-array yang sedang digabungkan.
- `copying`: Flag untuk menandakan fase penyalinan dari array `temp` kembali ke array utama.
- `mergeQueue`: Sebuah antrian (`Queue`) untuk menyimpan semua rentang `[left, mid, right]` yang perlu digabungkan. Ini adalah cara cerdas untuk mengubah sifat rekursif Merge Sort menjadi proses iteratif (langkah demi langkah).

2. Metode generateMergeSteps()

```
private void generateMergeSteps(int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        generateMergeSteps(l, m);
        generateMergeSteps(m + 1, r);
        mergeQueue.add(new int[]{l, m, r});
    }
}
```

- Metode rekursif ini tidak melakukan pengurutan, tetapi ia mensimulasikan panggilan rekursif dari Merge Sort.
- Setiap kali sebuah "merge" seharusnya terjadi dalam algoritma rekursif standar, metode ini malah menambahkan rentang `[left, mid, right]` dari merge tersebut ke dalam `mergeQueue`.
- Ini mempersiapkan semua langkah penggabungan yang akan dieksekusi satu per satu oleh `performStep`.

3. Metode performStep() (Logika Inti Merge Sort)
   Metode ini dibagi menjadi beberapa fase yang dikontrol oleh flag isMerging dan copying.
   1. Fase 1: Memulai Merge Baru

```
if (!isMerging && !mergeQueue.isEmpty()) {
    int[] range = mergeQueue.poll();
    left = range[0];
    mid = range[1];
    right = range[2];
    temp = new int[right - left + 1];
    i = left;
    j = mid + 1;
    k = 0;
    copying = false;
    isMerging = true;
    stepArea.append("Langkah " + stepCount++ + ": Mulai merge antara indeks " + left + " dan " + right + "\n");
    for(int idx = left; idx <= right; idx++) {
        labelArray[idx].setBackground(Color.YELLOW);
    }
    return;
}
```

Jika tidak sedang dalam proses merge dan antrian tidak kosong, mulai proses merge baru dengan mengambil parameter dari mergeQueue.

2. Fase 2: Membandingkan dan Menyalin ke temp

```
if (isMerging && !copying) {
    if(i <= mid) labelArray[i].setBackground(Color.CYAN);
    if(j <= right) labelArray[j].setBackground(Color.CYAN);

    if (i <= mid && (j > right || array[i] <= array[j])) {
        stepArea.append("   -> Bandingkan " + (j > right ? "" : array[i] + " <= " + array[j]) + ". Salin " + array[i] + " k
        temp[k++] = array[i++];
    } else if (j <= right) {
        stepArea.append("   -> Bandingkan " + array[i] + " > " + array[j] + ". Salin " + array[j] + " ke temp.\n");
        temp[k++] = array[j++];
    } else {
        copying = true;
        k = 0;
        stepArea.append("Langkah " + stepCount++ + ": Selesai membandingkan. Mulai salin dari temp ke array utama.\n");
    }
    return;
}
```

Ini adalah inti dari proses "merge". Elemen dari dua sub-array dibandingkan, dan yang lebih kecil disalin ke temp.

3. Fase 3: Menyalin dari temp kembali ke array

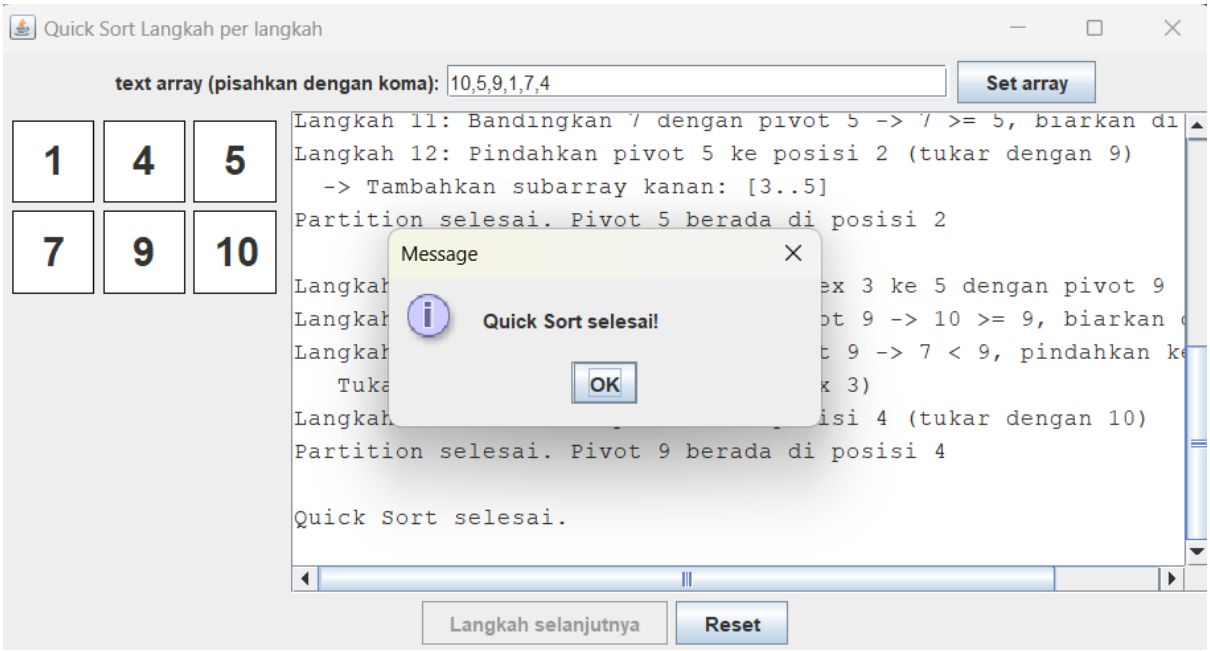
```
if (copying && k < temp.length) {
    array[left + k] = temp[k];
    labelArray[left + k].setText(String.valueOf(temp[k]));
    labelArray[left + k].setBackground(Color.GREEN);
    stepArea.append("   -> Salin " + temp[k] + " ke posisi " + (left + k) + "\n");
    k++;
    return;
}
```

Setelah temp terisi penuh, elemen-elemennya disalin kembali ke array utama di posisi yang benar.

4. Fase 4: Menyelesaikan Merge

```
if (copying && k == temp.length) {
    isMerging = false;
    copying = false;
    stepArea.append(str:"   -> Merge untuk range ini selesai.\n");
}
```

C. CLASS QUICKSORTGUI.JAVA



1. Variabel Kunci dan Struktur Data

```
private Stack<int[]> stack = new Stack<>();
private int i = 0, j;
private boolean sorting = false;
private int stepCount = 1;
private int low, high, pivot;
private boolean partitioning = false;
private int pivotIndex;
```

- `stack`: Sama seperti `Queue` pada Merge Sort, `Stack` digunakan untuk mengubah sifat rekursif Quick Sort menjadi iteratif. Stack menyimpan rentang `[low, high]` dari sub-array yang perlu diurutkan.
- `partitioning`: Flag untuk menandakan apakah sedang dalam fase partisi.
- `low, high`: Batas sub-array saat ini.
- `pivot, pivotIndex`: Nilai dan indeks dari pivot yang dipilih.
- `i, j`: Iterator untuk proses partisi. `j` memindai array, dan `i` menandai batas elemen yang lebih kecil dari pivot.

2. Metode setArrayFromInput()

```
private void setArrayFromInput() {
    String text = inputField.getText().trim();
    if (text.isEmpty()) return;

    String[] parts = text.split(regex:",");
    array = new int[parts.length];

    try {···
    } catch (NumberFormatException e) {···

    labelArray = new JLabel[array.length];
    panelArray.removeAll();
    for (int k = 0; k < array.length; k++) {···

    stack.clear();
    if (array.length > 1) {
        stack.push(new int[]{0, array.length - 1});
    }
    sorting = true;
    partitioning = false;
    stepCount = 1;
    stepArea.setText(t:"");
    stepButton.setEnabled(b:true);

    panelArray.revalidate();
    panelArray.repaint();
}
```

   - Setelah array di-set, rentang awal dari seluruh array (0 sampai length - 1) dimasukkan ke dalam stack untuk memulai proses.
3. **Metode performStep() (Logika Inti Quick Sort)** Metode ini dikontrol oleh flag partitioning.

- Fase 1: Memulai Partisi Baru

```
if (!partitioning) {
    while (!stack.isEmpty()) {
        int[] range = stack.pop();
        low = range[0];
        high = range[1];

        if (low < high) {
            pivotIndex = high;
            pivot = array[high];
            i = low - 1;
            j = low;
            partitioning = true;

            stepArea.append("Langkah " + stepCount++ + ": Mulai partition dari index "
                + low + " ke " + high + " dengan pivot " + pivot + " (index " + pivotIndex + ")\n");
            highlightPivot(pivotIndex);
            return;
        }
    }
```

Jika tidak sedang partisi, ambil sub-array baru dari stack dan siapkan untuk dipartisi.

- Fase 2: Proses Partisi (Iterasi)

```
if (j < high) {
    highlightCompare(j, pivotIndex);

    stepArea.append("Langkah " + stepCount++ + ": Bandingkan " + array[j] + " dengan pivot " + pivot);

    if (array[j] < pivot) {
        i++;
        stepArea.append(" -> " + array[j] + " < " + pivot + ", pindahkan ke kiri\n");
        if (i != j) {
            int valueJ = array[j];
            int valueI = array[i];
            swap(i, j);
            stepArea.append("   Tukar " + valueJ + " (index " + j + ") dengan "
                + valueI + " (index " + i + ")\n");
            updateLabels();
        } else {
            stepArea.append("   " + array[j] + " sudah di posisi yang benar\n");
        }
    } else {
        stepArea.append(" -> " + array[j] + " >= " + pivot + ", biarkan di kanan\n");
    }
    j++;
    return;
}
```

Langkah ini diulang-ulang. j berjalan dari low ke high-1. Jika array[j] lebih kecil dari pivot, ia ditukar dengan elemen di posisi i+1, dan i dimajukan. Ini secara efektif memindahkan semua elemen yang lebih kecil ke sisi kiri.

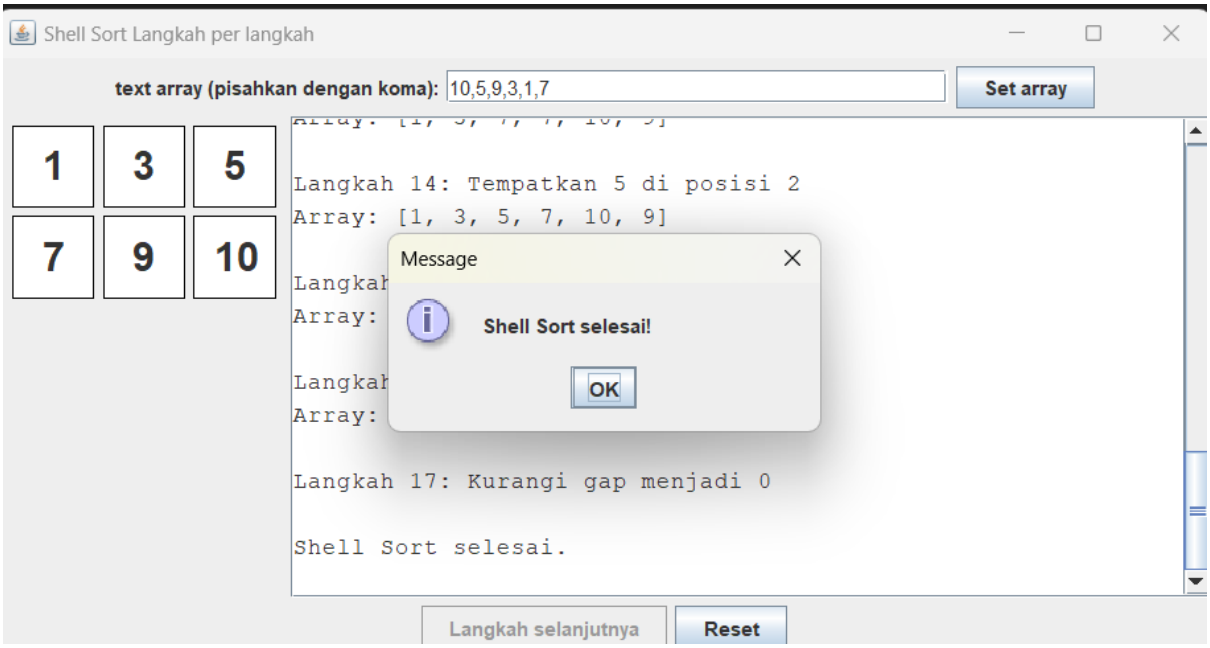
- **Fase 3: Menempatkan Pivot dan Mendorong Sub-array Baru ke Stack**

Java

```
partitioning = false;

if (low < finalPivotPos - 1) {
    stack.push(new int[]{low, finalPivotPos - 1});
    stepArea.append("  -> Tambahkan subarray kiri: [" + low + ".." + (finalPivotPos - 1) + "]\n");
}
if (finalPivotPos + 1 < high) {
    stack.push(new int[]{finalPivotPos + 1, high});
    stepArea.append("  -> Tambahkan subarray kanan: [" + (finalPivotPos + 1) + ".." + high + "]\n");
}
```

Setelah j mencapai high, partisi selesai. Pivot ditempatkan di posisi akhirnya (i+1). Kemudian, rentang sub-array di kiri dan kanan pivot yang baru ini dimasukkan ke dalam stack untuk diproses di langkah-langkah berikutnya.

D. CLASS SHELLSORT.JAVA



1. Variabel kunci

```
private boolean isSwapping = false;
private int temp;
private int gap;
private int i = 0, j;
private boolean sorting = false;
private int stepCount = 1;
```

- gap: Menyimpan jarak antar elemen yang sedang dibandingkan. Dimulai dari n/2 dan terus dibagi dua.
- i, j: Iterator untuk proses gapped insertion sort.
- temp: Menyimpan elemen array[i] yang akan disisipkan.
- isSwapping: Flag untuk mengelola sub-langkah dalam proses penyisipan.

2. Metode setArrayFromInput()

```
private void setArrayFromInput() {
    String text = inputField.getText().trim();
    if (text.isEmpty()) return;

    String[] parts = text.split(regex:",");
    array = new int[parts.length];
    try {···
    } catch (NumberFormatException e) {···

    gap = array.length / 2;
    i = gap;
    sorting = true;
    stepCount = 1;
    stepArea.setText(t:"");
    stepButton.setEnabled(b:true);
    panelArray.removeAll();

    labelArray = new JLabel[array.length];
    for (int k = 0; k < array.length; k++) {···

    panelArray.revalidate();
    panelArray.repaint();
}
```

Menginisialisasi gap pertama kali menjadi setengah dari panjang array. i juga dimulai dari posisi gap.

3. **Metode performStep() (Logika Inti Shell Sort)**

   Metode ini mensimulasikan gapped insertion sort secara bertahap.

   - **Pemeriksaan Kondisi Akhir atau Perubahan Gap**

```
    if (!sorting || gap == 0) {
        stepArea.append(str:"Shell Sort selesai.\n");
        resetHighlights();
        stepButton.setEnabled(b:false);
        JOptionPane.showMessageDialog(this, message:"Shell Sort selesai!");
        stepArea.append("Hasil akhir: " + java.util.Arrays.toString(array) + "\n\n");
        return;
    }

if (i < array.length) {
    if (!isSwapping) {···
    if (j >= gap && array[j - gap] > temp) {···
    } else {
        array[j] = temp; // letakkan nilai temp
        updateLabels();
        logStep("Tempatkan " + temp + " di posisi " + j);
        i++;
        isSwapping = false;
    }
```

Logika ini memeriksa apakah seluruh proses sort telah selesai (gap == 0). Jika iterasi untuk gap saat ini selesai (i >= array.length), maka gap diperkecil dan i direset.

- **Proses Penyisipan (Insertion)**

```
if (i < array.length) {
    if (!isSwapping) {

        temp = array[i];
        j = i;
        isSwapping = true;
    }
    if (j >= gap && array[j - gap] > temp) {
        array[j] = array[j - gap];
        labelArray[j].setBackground(Color.GREEN);

        labelArray[j - gap].setBackground(Color.CYAN);
        updateLabels();
        logStep("Geser elemen " + array[j] + " ke kanan");
        j -= gap;
        return;
    } else {
        array[j] = temp; // letakkan nilai temp
        updateLabels();
        logStep("Tempatkan " + temp + " di posisi " + j);
        i++;
        isSwapping = false;
    }
```

Bagian ini adalah inti dari *insertion sort* dengan jarak gap.

- temp = array[i] menyimpan nilai yang akan disisipkan.
- Loop while (disimulasikan dengan if dan return) membandingkan temp dengan elemen-elemen sebelumnya yang berjarak gap (array[j - gap]).
- Jika elemen sebelumnya lebih besar, ia digeser ke kanan (array[j] = array[j - gap]).
- Proses ini berlanjut hingga ditemukan posisi yang tepat, lalu temp ditempatkan di sana (array[j] = temp).
- Setelah satu elemen selesai disisipkan, i akan bertambah untuk memproses elemen berikutnya.

BAB 4

KESIMPULAN

Berdasarkan praktikum yang telah dilaksanakan, dapat ditarik beberapa kesimpulan:

1. Keempat algoritma pengurutan—Bubble Sort, Merge Sort, Quick Sort, dan Shell Sort—berhasil diimplementasikan menggunakan bahasa Java dan divisualisasikan menggunakan GUI dari Java Swing.
2. Visualisasi langkah demi langkah terbukti sangat efektif dalam memberikan pemahaman intuitif tentang cara kerja setiap algoritma.
    - **Bubble Sort** secara visual sangat lambat, dengan banyak sekali pertukaran kecil untuk memindahkan satu elemen ke posisi yang benar.
    - **Merge Sort** menunjukkan proses penggabungan yang teratur dan sistematis. Kebutuhan akan array temp terlihat jelas selama fase penggabungan.
    - **Quick Sort** memperlihatkan kekuatan partisi, di mana dalam satu putaran banyak elemen dapat berpindah mendekati posisi akhirnya. Namun, kinerjanya sangat bergantung pada pemilihan pivot.
    - **Shell Sort** menunjukkan efisiensinya dengan melakukan pertukaran elemen jarak jauh pada awalnya, yang dengan cepat mengurangi ketidakteraturan array secara keseluruhan, sebelum akhirnya disempurnakan dengan gap yang lebih kecil.
3. Implementasi state machine (menggunakan flag seperti isMerging, partitioning, dan struktur data seperti Queue atau Stack) adalah teknik yang krusial untuk mengubah algoritma yang bersifat kontinu atau rekursif menjadi serangkaian langkah diskrit yang dapat dikontrol oleh pengguna melalui tombol.