LAPORAN PRAKTIKUM

SORTING ALGORITHMS 2



Oleh :

MUHAMMAD GALID AVERO

NIM 2311532008

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU : DR. WAHYUDI, S.T, M.T

ASISTEN LABORATORIUM : ZAKY ADIL HAKIM

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, MEI 2024

1. **PENDAHULUAN**

Sorting algorithms adalah prosedur atau metode yang digunakan untuk mengatur elemen-elemen dalam suatu daftar atau array dalam urutan tertentu, biasanya dalam urutan menaik (ascending) atau menurun (descending). Sorting adalah salah satu operasi fundamental dalam ilmu komputer dan penting dalam berbagai aplikasi, seperti pengolahan data, algoritma pencarian, dan optimasi. Sorting adalah salah satu operasi dasar dalam ilmu komputer yang memiliki banyak aplikasi, seperti:

1. **Pengorganisasian Data**: Mengurutkan data membantu dalam mengorganisir informasi sehingga lebih mudah dibaca dan dipahami.
2. **Peningkatan Efisiensi Pencarian**: Data yang telah diurutkan memungkinkan algoritma pencarian seperti binary search untuk bekerja lebih cepat.
3. **Optimasi Algoritma Lain**: Beberapa algoritma lain, seperti algoritma penggabungan data (merge), sering kali membutuhkan data yang sudah diurutkan untuk bekerja secara efisien.
4. **Pemrosesan Data**: Banyak operasi pemrosesan data, seperti menemukan median, memerlukan data yang sudah diurutkan.

Berikut adalah penjelasan rinci mengenai tiga algoritma penyortiran: Shell Sort, Quick Sort, dan Merge Sort.

**1. Shell Sort**

**Deskripsi:** Shell Sort adalah variasi dari Insertion Sort yang memperbaiki efisiensinya dengan membandingkan elemen yang berjauhan, bukan hanya elemen yang berdekatan. Algoritma ini menggunakan gap sequence (urutan celah) untuk membandingkan dan menukar elemen yang terpisah beberapa posisi.

**Langkah-langkah:**

1. Tentukan urutan celah (gap sequence). Misalnya, mulai dengan setengah panjang array, kemudian setengahnya lagi, dan seterusnya hingga celah menjadi 1.
2. Untuk setiap celah, lakukan Insertion Sort pada elemen-elemen yang terpisah oleh celah tersebut.
3. Ulangi proses ini hingga celah terakhir (celah 1), di mana dilakukan Insertion Sort pada seluruh array.

**Kompleksitas Waktu:**

* Worst-case: Bergantung pada gap sequence, umumnya antara O(n3/2)O(n^{3/2})O(n3/2) dan O(n2)O(n^2)O(n2).
* Best-case: O(nlog⁡n)O(n \log n)O(nlogn)
* Average-case: Bergantung pada gap sequence, umumnya lebih baik dari O(n2)O(n^2)O(n2).

**Kelebihan:**

* Mudah diimplementasikan.
* Lebih cepat dari Insertion Sort dan Bubble Sort untuk daftar besar.
* Dapat dioptimalkan dengan memilih gap sequence yang tepat.

**Kekurangan:**

* Kompleksitas waktu bergantung pada pilihan gap sequence.
* Tidak stabil.

**2. Quick Sort**

**Deskripsi:** Quick Sort adalah algoritma penyortiran berbasis divide-and-conquer yang memilih elemen pivot dan membagi array menjadi dua subarray: elemen yang lebih kecil dari pivot dan elemen yang lebih besar dari pivot. Proses ini diulang secara rekursif untuk setiap subarray.

**Langkah-langkah:**

1. Pilih elemen pivot dari array.
2. Partisi array sehingga elemen yang lebih kecil dari pivot berada di sebelah kiri dan elemen yang lebih besar dari pivot berada di sebelah kanan.
3. Sortir subarray kiri dan kanan secara rekursif menggunakan Quick Sort.
4. Gabungkan subarray yang sudah tersortir.

**Kompleksitas Waktu:**

* Worst-case: O(n2)O(n^2)O(n2) (terjadi jika pivot dipilih secara buruk, misalnya elemen terkecil atau terbesar secara konsisten)
* Best-case: O(nlog⁡n)O(n \log n)O(nlogn) (terjadi jika pivot selalu membagi array secara seimbang)
* Average-case: O(nlog⁡n)O(n \log n)O(nlogn)

**Kelebihan:**

* Umumnya lebih cepat dalam praktik dibandingkan algoritma penyortiran lain seperti Merge Sort dan Heap Sort.
* Tidak membutuhkan ruang tambahan yang signifikan (in-place).

**Kekurangan:**

* Kompleksitas waktu terburuk yang buruk jika pivot dipilih secara tidak optimal.
* Tidak stabil.

**3. Merge Sort**

**Deskripsi:** Merge Sort adalah algoritma penyortiran berbasis divide-and-conquer. Algoritma ini membagi array menjadi dua bagian, menyortir setiap bagian secara rekursif, dan kemudian menggabungkan (merge) kedua bagian tersebut menjadi satu array tersortir.

**Langkah-langkah:**

1. Bagi array menjadi dua subarray yang sama besar.
2. Sortir setiap subarray secara rekursif menggunakan Merge Sort.
3. Gabungkan kedua subarray yang sudah tersortir.

**Kompleksitas Waktu:**

* Worst-case: O(nlog⁡n)O(n \log n)O(nlogn)
* Best-case: O(nlog⁡n)O(n \log n)O(nlogn)
* Average-case: O(nlog⁡n)O(n \log n)O(nlogn)

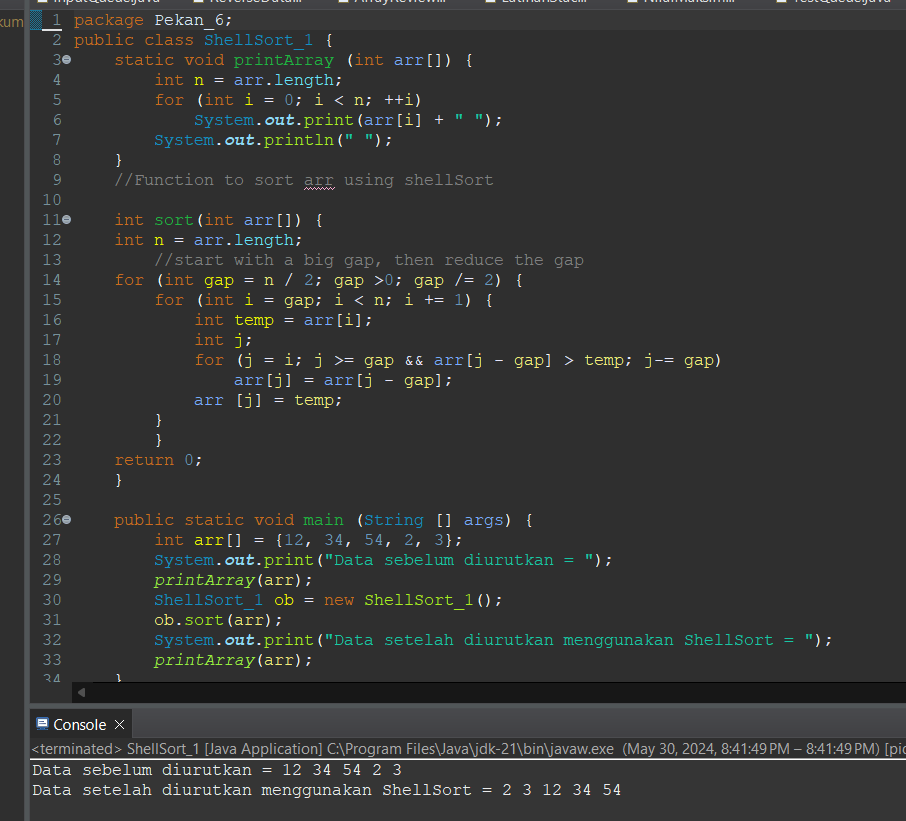
**Kelebihan:**

* Stabil (tidak mengubah urutan elemen dengan nilai yang sama).
* Kompleksitas waktu yang konsisten dan terjamin O(nlog⁡n)O(n \log n)O(nlogn).
* Cocok untuk daftar besar.

**Kekurangan:**

* Membutuhkan ruang tambahan untuk array sementara (tidak in-place).
* Implementasi lebih kompleks dibandingkan dengan algoritma sederhana lainnya.

1. **TUJUAN PRAKTIKUM**
2. Mengetahui cara membuat ShellSort
3. Mengetahui cara membuat QuickSort
4. Mengetahui cara membuat MergeSort
5. **LANGKAH PRAKTIKUM**
6. ShellSort



Metode printArray

Metode ini digunakan untuk mencetak isi array ke console. Metode ini menerima array integer sebagai parameter dan mencetak setiap elemen array dengan spasi di antaranya.

int n = arr.length; : Mendapatkan panjang array dan menyimpannya dalam variabel n.

for (int i = 0; i < n; ++i) : Looping untuk mencetak setiap elemen array.

System.out.print(arr[i] + " "); : Mencetak elemen array dengan spasi di antaranya.

System.out.println(" "); : Mencetak baris baru setelah mencetak semua elemen array.

Metode sort

Metode ini adalah inti dari algoritme Shell Sort. Metode ini menerima array integer sebagai parameter dan melakukan sorting menggunakan Shell Sort.

int n = arr.length; : Mendapatkan panjang array dan menyimpannya dalam variabel n.

for (int gap = n / 2; gap > 0; gap /= 2) : Looping untuk mengurangi gap (jarak) secara dinamis. Gap awalnya adalah setengah dari panjang array, kemudian dikurangi setengahnya pada setiap iterasi.

for (int i = gap; i < n; i += 1) : Looping untuk melakukan sorting pada setiap elemen array, dimulai dari elemen ke-gap hingga elemen terakhir.

int temp = arr[i]; : Menyimpan elemen array ke-i dalam variabel temp.

int j; : Deklarasi variabel j yang akan digunakan untuk melakukan perbandingan.

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) : Looping untuk melakukan perbandingan elemen array ke-j dengan elemen array ke-j-gap. Jika elemen array ke-j-gap lebih besar dari temp, maka elemen array ke-j-gap akan digeser ke kanan.

arr[j] = temp; : Menyimpan temp ke elemen array ke-j setelah perbandingan selesai.

Metode main

int arr[] = {12, 34, 54, 2, 3}; : Deklarasi array integer dengan isi awal.

System.out.print("Data sebelum diurutkan = "); : Mencetak pesan "Data sebelum diurutkan = ".

printArray(arr); : Mencetak isi array awal menggunakan metode printArray.

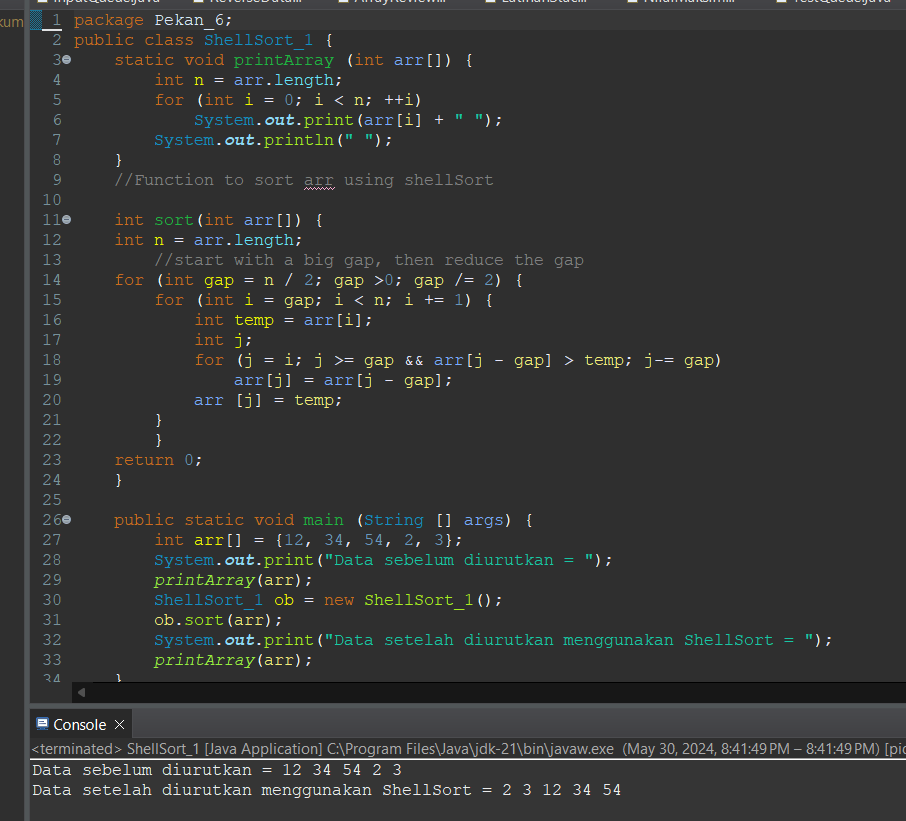
ShellSort\_1 ob = new ShellSort\_1(); : Membuat objek dari kelas ShellSort\_1.

ob.sort(arr); : Memanggil metode sort untuk melakukan sorting pada array.

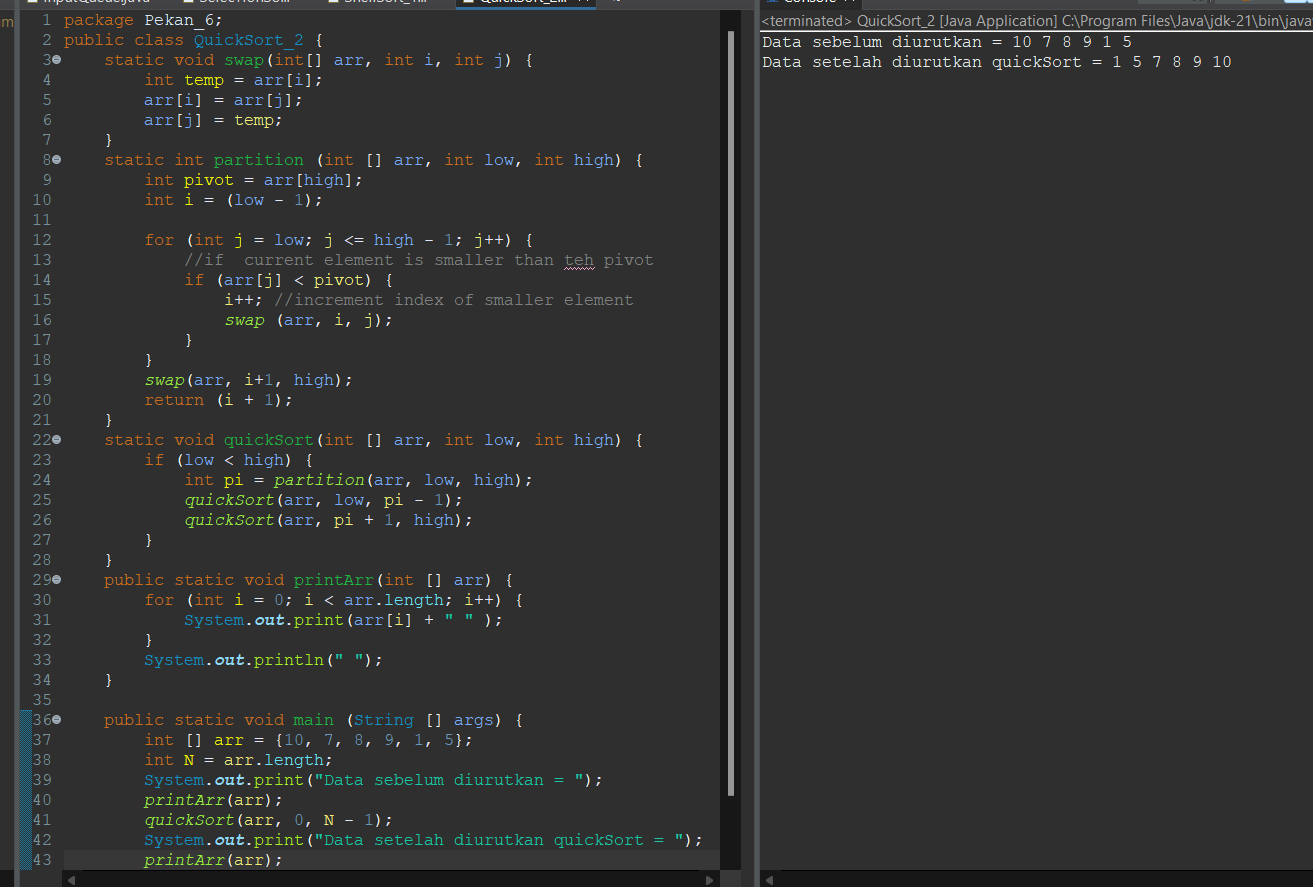
System.out.print("Data setelah diurutkan menggunakan ShellSort = "); : Mencetak pesan "Data setelah diurutkan menggunakan ShellSort = ".

printArray(arr); : Mencetak isi array yang telah diurutkan menggunakan metode printArray.

Untuk hasil dari program adalah sebagai berikut :



1. QuickSort



Metode swap

Metode ini digunakan untuk menukar dua elemen dalam array.

int temp = arr[i]; : Menyimpan elemen array ke-i dalam variabel temp.

arr[i] = arr[j]; : Menggantikan elemen array ke-i dengan elemen array ke-j.

arr[j] = temp; : Menggantikan elemen array ke-j dengan temp.

Metode partition

Metode ini digunakan untuk mempartisi array menjadi dua bagian: elemen-elemen yang lebih kecil dari pivot dan elemen-elemen yang lebih besar dari pivot.

int pivot = arr[high]; : Menyimpan elemen array ke-high sebagai pivot.

int i = (low - 1); : Inisialisasi variabel i sebagai indeks elemen yang lebih kecil dari pivot.

for (int j = low; j <= high - 1; j++) : Looping untuk mempartisi array.

if (arr[j] < pivot) : Jika elemen array ke-j lebih kecil dari pivot, maka:

i++; : Increment indeks i.

swap(arr, i, j); : Menukar elemen array ke-i dengan elemen array ke-j.

swap(arr, i+1, high); : Menukar elemen array ke-i+1 dengan pivot.

return (i + 1); : Mengembalikan indeks i+1 sebagai indeks pivot.

Metode quickSort

Metode ini digunakan untuk melakukan sorting menggunakan QuickSort

if (low < high) : Jika indeks low lebih kecil dari indeks high, maka:

int pi = partition(arr, low, high); : Memanggil metode partition untuk mempartisi array.

quickSort(arr, low, pi - 1); : Rekursif memanggil metode quickSort untuk sorting bagian kiri.

quickSort(arr, pi + 1, high); : Rekursif memanggil metode quickSort untuk sorting bagian kanan.

Metode printArr

Metode ini digunakan untuk mencetak isi array ke console.

for (int i = 0; i < arr.length; i++) : Looping untuk mencetak setiap elemen array.

System.out.print(arr[i] + " "); : Mencetak elemen array dengan spasi di antaranya.

System.out.println(" "); : Mencetak baris baru setelah mencetak semua elemen array.

Metode main

Metode ini adalah metode utama yang akan dijalankan ketika program dijalankan.

int [] arr = {10, 7, 8, 9, 1, 5}; : Deklarasi array integer dengan isi awal.

int N = arr.length; : Mendapatkan panjang array dan menyimpannya dalam variabel N.

System.out.print("Data sebelum diurutkan = "); : Mencetak pesan "Data sebelum diurutkan = ".

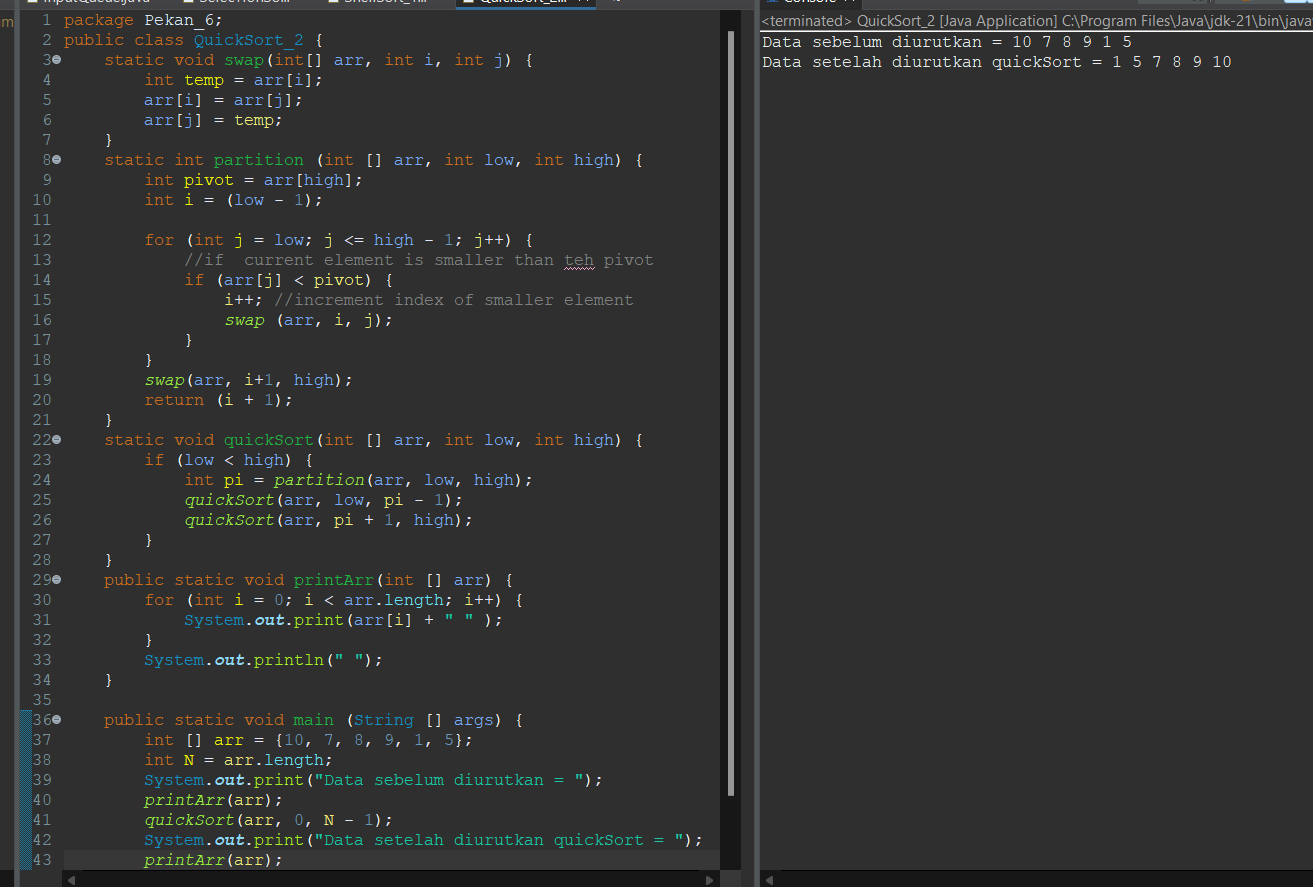
printArr(arr); : Mencetak isi array awal menggunakan metode printArr.

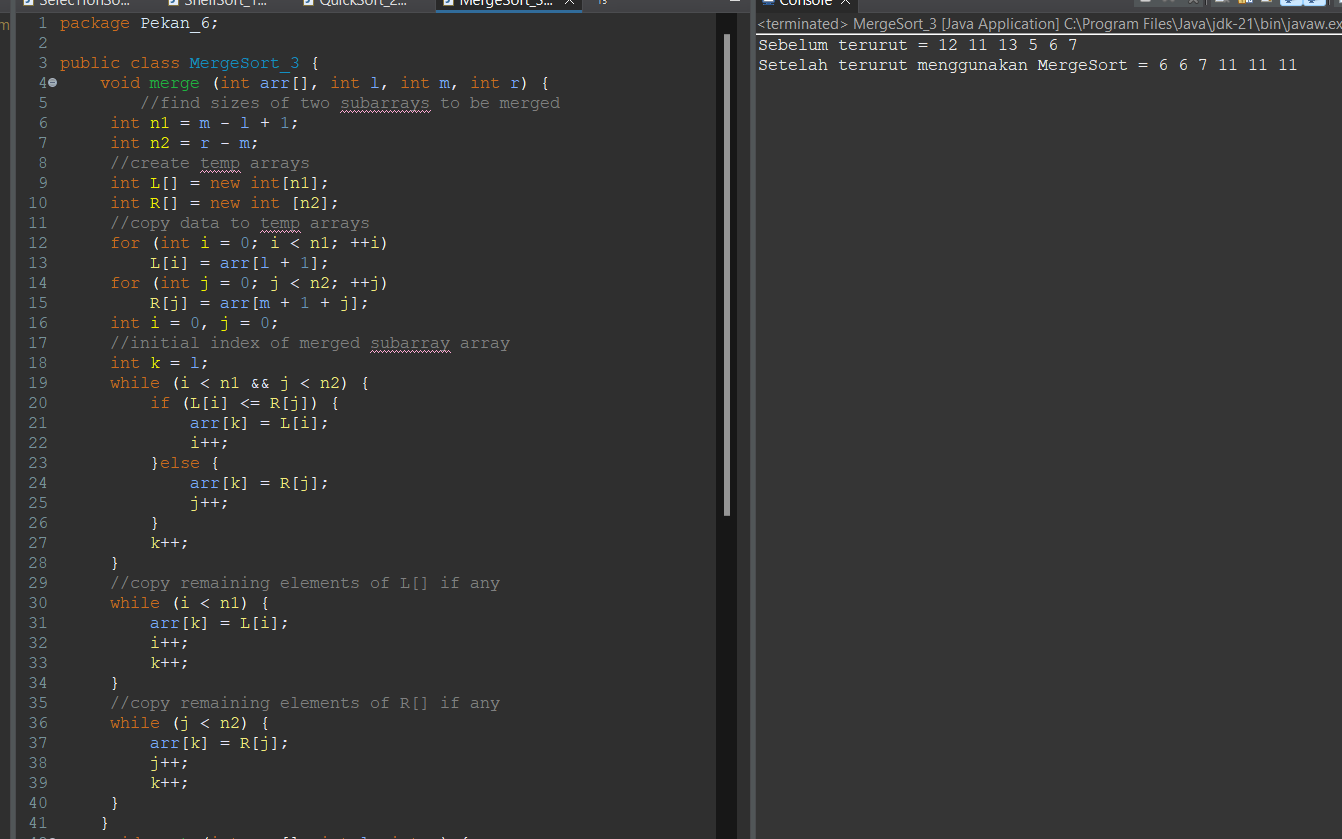
quickSort(arr, 0, N - 1); : Memanggil metode quickSort untuk melakukan sorting pada array.

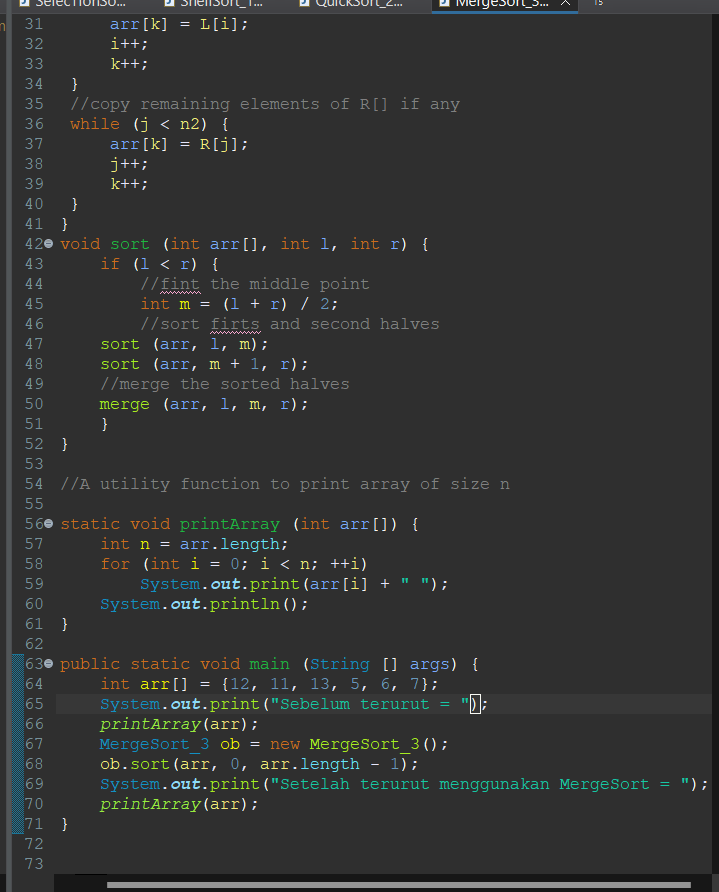
System.out.print("Data setelah diurutkan quickSort = "); : Mencetak pesan "Data setelah diurutkan quickSort = ".

printArr(arr); : Mencetak isi array yang telah diurutkan menggunakan metode printArr.

Untuk hasil dari kode tersebut adalah sebagai berikut :



1. MergeSort



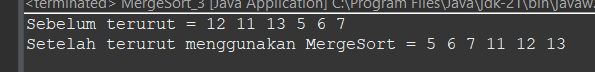
merge(int arr[], int l, int m, int r): Fungsi ini menggabungkan dua subarray dari arr[]. Subarray pertama adalah arr[l..m] dan subarray kedua adalah arr[m+1..r]. Fungsi ini bekerja dengan membuat array sementara L[] dan R[], menyalin data ke dalamnya, dan kemudian menggabungkan kembali ke dalam arr[] dengan cara yang terurut.

sort(int arr[], int l, int r): Fungsi ini adalah fungsi utama yang memanggil dirinya sendiri secara rekursif untuk mendapatkan subarray yang lebih kecil, dan kemudian memanggil fungsi merge() untuk menggabungkan subarray yang telah diurutkan.

printArray(int arr[]): Fungsi ini digunakan untuk mencetak elemen array pada konsol.

main(String[] args): Ini adalah fungsi utama yang menjalankan algoritma Merge Sort. Array arr[] didefinisikan dengan elemen yang belum diurutkan, dan fungsi sort() dipanggil untuk mengurutkan array tersebut. Sebelum dan sesudah pemanggilan sort(), fungsi printArray() dipanggil untuk menampilkan kondisi array.

Untuk hasil dari kode tersebut adalah sebagai berikut :



1. **KESIMPULAN**

Sorting algorithms berguna dalam proses pengurutan data yang dilakukan oleh seorang programmer untuk membuat data yang lebih rapi dan lebih teratur, Algoritma Sorting seperti Shell Sort, Quick Sort, dan Merge Sort adalah salah satu metode dasar untuk mengurutkan data. Memilih algoritma yang tepat tergantung pada kebutuhan spesifik dan karakteristik data yang akan diurutkan.