

# **PRAKTIKUM STRUKTUR DATA**

**“2 & 3”**



**Disusun oleh :**

**Diana Eka Permata Sari**

**TRPL 2C**

**(362458302090)**

**PROGAM STUDI TEKNOLOGI REKAYASA PERANKAT LUNAK  
JURUSAN BISNIS DAN INFORMATIKA**

**2025**

Jalan Raya Jember No.KM13, Kawang, Labanasem, Kec. Kabat, Kabupaten  
Banyuwangi, Jawa Timur 68461

## **ALGORITMA PENGURUTAN**

### **(INSERTION, SELECTION, BUBBLE DAN SHELL)**

#### Tujuan Praktikum :

1. Memahami mengenai algoritma pengurutan insertion sort dan selection sort
2. Mampu mengimplementasikan algoritma pengurutan insertion sort dan selection sort secara ascending dan descending
3. Memahami mengenai algoritma pengurutan bubble sort dan shell sort
4. Mampu mengimplementasikan algoritma pengurutan bubble sort dan shell sort secara ascending dan descending.

#### Dasar Teori :

##### 1. Algoritma Insertion Sort

Salah satu algoritma sorting yang paling sederhana adalah insertion sort. Algoritma insertion sort pada dasarnya memilah data yang akan urutkan menjadi 2 bagian, yang belum diurutkan dan yang sudah diurutkan. Elemen pertama diambil dari bagian array yang belum diurutkan dan kemudian diletakkan sesuai posisinya pada bagian lain dari array yang telah diurutkan. Langkah ini dilakukan secara berulang hingga tak ada lagi elemen tersisa pada bagian array yang belum diurutkan. Metode insertion sort merupakan metode yang mengurutkan bilangan-bilangan yang telah terbaca, dan berikutnya secara berulang akan menyisipkan bilangan-bilangan dalam array yang belum terbaca ke sisi kiri array yang telah terurut. Kita mengambil pada bilangan yang paling kiri. Bilangan tersebut dikatakan urut terhadap dirinya sendiri karena bilangan yang di bandingkan baru 1.

##### 2. Algoritma Selection Sort

Ide utama dari algoritma selection sort adalah memilih elemen dengan nilai paling rendah dan menukar elemen yang terpilih dengan elemen ke-i. Nilai dari i dimulai dari 1 ke n, dimana n adalah jumlah total elemen dikurangi 1. Cek seluruh array dan cari array yang mempunyai nilai terkecil index 8 (1). Setelah ketemu tukar dengan array yang berada di pojok kiri (3).

##### 3. Algoritma Bubble Sort

Bubble sort (metode gelembung) adalah metode/algoritma pengurutan dengan dengan cara melakukan penukaran data dengan tepat disebelahnya secara terus menerus sampai bisa dipastikan dalam satu iterasi tertentu tidak ada lagi perubahan. Jika tidak ada perubahan berarti data sudah terurut. Disebut pengurutan gelembung karena masing-masing kunci akan dengan lambat menggelembung ke posisinya yang tepat.

##### 4. Algoritma Shell Sort

Metode shell sort dikembangkan oleh Donald L. Shell pada tahun 1959. Dalam metode ini jarak antara dua elemen yang dibandingkan dan ditukarkan tertentu. Secara singkat metode ini dijelaskan sebagai berikut. Pada langkah pertama, kita ambil elemen pertama dan kita bandingkan dan kita bandingkan dengan elemen pada jarak tertentu dari elemen pertama tersebut. Kemudian elemen kedua kita bandingkan dengan elemen lain dengan jarak yang sama seperti jarak yang sama seperti diatas. Demikian seterusnya sampai seluruh elemen dibandingkan. Pada langkah kedua proses diulang dengan langkah yang lebih kecil, pada langkah ketiga jarak tersebut diperkecil lagi seluruh proses dihentikan jika jarak sudah sama dengan satu.

Tugas Pendahuluan :

1. Tuliskan algoritma pengurutan insertion sort secara ascending

Jawab :

- a. mulai mengambil array yang diurutkan
- b. mulai dari elemen kedua (indeks 1) hingga elemen terakhir array
- c. membandingkan 'j' lebih besar atau sama dengan 0 dan elemen pada indeks 'j' lebih besar pada 'key'
- d. menyisipkan setelah keluar dari loop, tempatkan 'key' pada indeks 'j +1'
- e. ulangi Kembali ke Langkah 2
- f. selesai

2. Tuliskan algoritma pengurutan selection sort secara ascending

jawab :

- a. Mengambil array yang akan diurutkan
- b. Setiap elemen array anggap elemen saat ini sebagai terkecil dan simpan indeksnya, bandingkan dengan elemen yang tersisa, jika ada elemen terkecil perbarui indeks elemen terkecil
- c. Menukar elemen terkecil dengan elemen posisi saat ini
- d. Ulangi Langkah 2 dan 3
- e. Selesai

3. Tuliskan algoritma pengurutan bubble sort secara ascending

Jawab ;

- a. Ambil array yang diurutkan
- b. Lakukan perulangan indeks dari 0 hingga n-1
- c. Ulangi Langkah 2
- d. Selesai

4. Tuliskan algoritma pengurutan shell sort secara ascending

Jawab :

- a. Ambil array yang akan diurutkan
- b. Tentukan nilai awal untuk gap

- c. Lakukan insertion sort untuk setiap gap: Bandingkan elemen dengan elemen lain yang berjarak gap, Jika elemen sebelumnya lebih besar, lakukan pertukaran.
- d. Selesai

Percobaan :

1. Insertion short secara ascending

```
void insertionSort<T extends Comparable<T>>(List<T> arr) {
    for (int i = arr.length - 1; i >= 0; i--) {
        for (int j = i + 1, k = i; j < arr.length; j++) {
            if (arr[j].compareTo(arr[k]) > 0) {
                break;
            } else {
                T temp = arr[k];
                arr[k] = arr[j];
                arr[j] = temp;
                k = j;
            }
        }
    }
}

void display<T>(List<T> data) {
    for (T obj in data) {
        print('$obj ');
    }
    print('');
}

void main() {
    List<num> data = List<num>.generate(10, (index)=>(index + 1) * 2);
    data.shuffle();
    print('Data Sebelum Pengurutan:');
    display(data);

    DateTime start = DateTime.now();
    insertionSort<num>(data);
    Duration elapsedTime = DateTime.now().difference(start);
    print('Data Setelah Pengurutan:');
    display(data);
    print('Waktu: $elapsedTime');
}
```

Hasil run :

```
PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Pengurutan:
8
4
20
10
14
6
2
18
12
16

Data Setelah Pengurutan:
2
4
6
8
10
12
14
16
18
20

Waktu: 0:00:00.004934

Exited.
```

## 2. Selection sort secara ascending

```
3. void selectionSort<T extends Comparable<T>>(List<T> arr) {
4.     T temp;
5.     for (int i = arr.length - 1; i >= 0; i--) {
6.         int max = i;
7.         for (int j = i - 1; j >= 0; j--) {
8.             if (arr[j].compareTo(arr[max]) > 0) max = j;
9.         }
10.        temp = arr[i];
11.        arr[i] = arr[max];
12.        arr[max] = temp;
13.    }
14.}
15.
16.void display<T>(List<T> data) {
```

```

17.  for (T objek in data) {
18.      print('$objek ');
19.  }
20.  print('');
21.}
22.
23.void main() {
24.    List<num> data = List<num>.generate(10, (index)=>(index + 1) * 2);
25.    data.shuffle();
26.    print('Data Sebelum Pengurutan:');
27.    display(data);
28.
29.    DateTime start = DateTime.now();
30.    selectionSort(data);
31.    Duration elapsedTime = DateTime.now().difference(start);
32.
33.    print('Data Setelah Pengurutan:');
34.    display(data);
35.    print('Waktu: ${elapsedTime.inMilliseconds} ms');
36.}

```

Hasil run :

```

PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Pengurutan:
12
16
4
8
2
10
18
20
6
14

Data Setelah Pengurutan:
2
4
6
8
10
12
14
16
18
20

Waktu: 0 ms

Exited.

```

### 3. Buble short secara ascending

```
void bubbleSort<T extends Comparable<T>>(List<T> arr) {
    int i, j;
    T temp;
    for (i = 0; i < arr.length - 1; i++) {
        for (j = 0; j < arr.length - i - 1; j++) {
            if (arr[j].compareTo(arr[j + 1]) > 0) {
                temp = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

void display<T>(List<T> data) {
    for (T objek in data) {
        print('$objek ');
    }
    print('');
}

void main() {
    List<num> data = List<num>.generate(10, (index)=>(index + 1) * 2);
    data.shuffle();
    print('Data Sebelum Pengurutan:');
    display(data);

    DateTime start = DateTime.now();
    bubbleSort(data);
    Duration elapsedTime = DateTime.now().difference(start);

    print('Data Setelah Pengurutan:');
    display(data);
    print('Waktu: ${elapsedTime.inMilliseconds} ms');
}
```

Hasil run :

```
PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Data Sebelum Pengurutan:
4
8
14
20
18
12
2
16
10
6

Data Setelah Pengurutan:
2
4
6
8
10
12
14
16
18
20

Waktu: 0 ms

Exited.

> Please start a debug session to evaluate expressions
```

#### 4. Bubble Sort secara ascending dengan flag

```
void bubbleSortFlag<T extends Comparable<T>>(List<T> arr) {
    int i = 0, j;
    bool didSwap = true;
    T temp;
    while (i < arr.length - 1 && didSwap) {
        didSwap = false;
        for (j = 0; j < arr.length - i - 1; j++) {
            if (arr[j].compareTo(arr[j + 1]) > 0) {
                temp = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = temp;
                didSwap = true;
            }
        }
    }
}
```



```

        i++;
    }
}

void display<T>(List<T> data) {
    for (T objek in data) {
        print('$objek ');
    }
    print('');
}

void main() {
    List<num> data = List<num>.generate(10, (index)=>(index + 1) * 2);
    data.shuffle();
    print('Data Sebelum Pengurutan:');
    display(data);

    DateTime start = DateTime.now();
    bubbleSortFlag(data);
    Duration elapsedTime = DateTime.now().difference(start);
    print('Data Setelah Pengurutan:');
    display(data);
    print('Waktu: ${elapsedTime.inMilliseconds} ms');
}

```

Hasil run :

```

PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Pengurutan:
12

Exited.
6
16
4
10
20
18
8
14
2

Data Setelah Pengurutan:
2
4
6
8
10
12
14
16
18
20

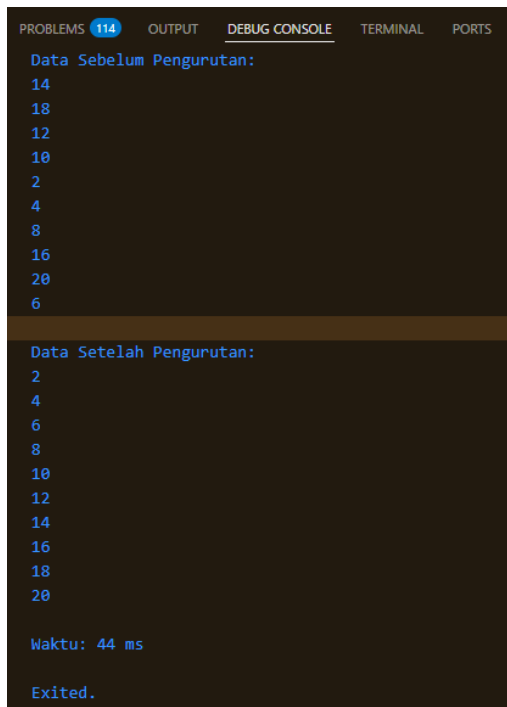
Waktu: 0 ms
>

```

## 5. Shell shrot secara ascending

```
6. void shellSort<T extends Comparable<T>>(List<T> arr) {
7.     int i, jarak;
8.     bool didSwap = true;
9.     T temp;
10.    jarak = arr.length;
11.    while (jarak > 1) {
12.        jarak = (jarak / 2).floor();
13.        didSwap = true;
14.        while (didSwap) {
15.            didSwap = false;
16.            i = 0;
17.            while (i < (arr.length - jarak)) {
18.                if (arr[i].compareTo(arr[i + jarak]) > 0) {
19.                    temp = arr[i];
20.                    arr[i] = arr[i + jarak];
21.                    arr[i + jarak] = temp;
22.                    didSwap = true;
23.                }
24.                i++;
25.            }
26.        }
27.    }
28.}
29.
30.void display<T>(List<T> data) {
31.    for (T objek in data) {
32.        print('$objek ');
33.    }
34.    print('');
35.}
36.
37.void main() {
38.    List<num> data = List<num>.generate(10, (index)=>(index + 1) * 2);
39.    data.shuffle();
40.    print('Data Sebelum Pengurutan:');
41.    display(data);
42.
43.    DateTime start = DateTime.now();
44.    shellSort(data);
45.    Duration elapsedTime = DateTime.now().difference(start);
46.    print('Data Setelah Pengurutan:');
47.    display(data);
48.    print('Waktu: ${elapsedTime.inMilliseconds} ms');
49.}
50.
```

Hasil run :



```
PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Pengurutan:
14
18
12
10
2
4
8
16
20
6

Data Setelah Pengurutan:
2
4
6
8
10
12
14
16
18
20

Waktu: 44 ms
Exited.
```

Latihan :

5 Buatlah class Mahasiswa dengan variable nrp dan nama yang memiliki tipe String! Class Mahasiswa mengimplementasikan interface Comparable, selanjutnya implementasikan fungsi abstract compareTo(), untuk membandingkan dua objek mahasiswa berdasarkan nrp.

```
class Mahasiswa implements Comparable<Mahasiswa> {
    String nrp;
    String nama;

    // Konstruktor untuk inisialisasi nrp dan nama
    Mahasiswa(this.nrp, this.nama);

    // Override metode toString untuk representasi objek
    @Override
    String toString() {
        return 'Mahasiswa{nrp: $nrp, nama: $nama}';
    }

    // Implementasi metode compareTo
    @Override
    int compareTo(Mahasiswa other) {
        // Membandingkan berdasarkan NRP
        return this.nrp.compareTo(other.nrp);
    }
}
```

```

void main() {
  // Contoh penggunaan
  Mahasiswa mhs1 = Mahasiswa('123456', 'Nandita');
  Mahasiswa mhs2 = Mahasiswa('654321', 'Dewangga');
  Mahasiswa mhs3 = Mahasiswa('123456', 'Cani');

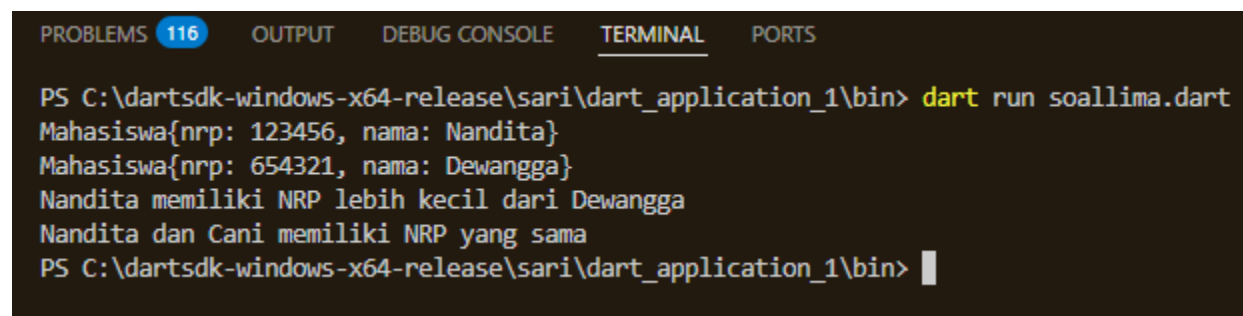
  // Membandingkan dua objek
  print(mhs1); // Output: Mahasiswa{nrp: 123456, nama: nandita}
  print(mhs2); // Output: Mahasiswa{nrp: 654321, nama: dewangga}

  // Perbandingan
  if (mhs1.compareTo(mhs2) < 0) {
    print('${mhs1.nama} memiliki NRP lebih kecil dari ${mhs2.nama}');
  } else if (mhs1.compareTo(mhs2) > 0) {
    print('${mhs1.nama} memiliki NRP lebih besar dari ${mhs2.nama}');
  } else {
    print('${mhs1.nama} dan ${mhs2.nama} memiliki NRP yang sama');
  }

  // Contoh perbandingan yang sama
  if (mhs1.compareTo(mhs3) == 0) {
    print('${mhs1.nama} dan ${mhs3.nama} memiliki NRP yang sama');
  }
}

```

Hasil run :



```

PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin> dart run soallima.dart
Mahasiswa{nrp: 123456, nama: Nandita}
Mahasiswa{nrp: 654321, nama: Dewangga}
Nandita memiliki NRP lebih kecil dari Dewangga
Nandita dan Cani memiliki NRP yang sama
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin>

```

Penjelasan : code ini menyediakan struktur untuk mengeloladata dari mahasiswa dan membandingkannya dengan nrp, kode ini juga dapat melihat bagaimana objek mahasiswa berperilaku dalam perandingan

6. Pada soal nomor 5, lakukan untuk algoritma selction sort

```

class Mahasiswa {
  String nrp;
  String nama;

  // Konstruktor untuk inisialisasi nrp dan nama

```

```

Mahasiswa(this.nrp, this.nama);

// Override metode toString untuk representasi objek
@Override
String toString() {
    return 'Mahasiswa{nrp: $nrp, nama: $nama}';
}
}

void main() {
    // Membuat daftar mahasiswa
    List<Mahasiswa> mahasiswaList = [
        Mahasiswa('778909', 'Nandita'),
        Mahasiswa('778908', 'Dzikra'),
        Mahasiswa('778907', 'Alisa'),
        Mahasiswa('778904', 'Jason'),
        Mahasiswa('778901', 'Gumilang')
    ];

    // Menampilkan data sebelum diurutkan
    print("Data sebelum diurutkan:");
    print(mahasiswaList);

    // Menggunakan Selection Sort untuk mengurutkan mahasiswaList berdasarkan
    NRP
    selectionSort(mahasiswaList);

    // Menampilkan data setelah diurutkan
    print("\nData setelah diurutkan dengan Selection Sort:");
    print(mahasiswaList);
}

void selectionSort(List<Mahasiswa> list) {
    int n = list.length;

    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            // Mencari indeks elemen terkecil
            if (list[j].nrp.compareTo(list[minIndex].nrp) < 0) {
                minIndex = j;
            }
        }
        // Menukar elemen terkecil dengan elemen di posisi i
        if (minIndex != i) {
            var temp = list[i];
            list[i] = list[minIndex];
            list[minIndex] = temp;
        }
    }
}

```

```

    }
  }
}

```

Hasil run :

```

PROBLEMS 116 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin> dart run soalno6.dart
Data sebelum diurutkan:
[Mahasiswa{nrp: 778909, nama: Nandita}, Mahasiswa{nrp: 778908, nama: Dzikra}, Mahasiswa{nrp: 778907, nama: Alisa}, Mahasiswa{nrp: 778904, nama: Jason}, Mahasiswa{nrp: 778901, nama: Gumilang}]

Data setelah diurutkan dengan Selection Sort:
[Mahasiswa{nrp: 778904, nama: Jason}, Mahasiswa{nrp: 778907, nama: Alisa}, Mahasiswa{nrp: 778908, nama: Dzikra}, Mahasiswa{nrp: 778909, nama: Nandita}, Mahasiswa{nrp: 778901, nama: Gumilang}]
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin>

```

Penjelasan : kode ini mengatur dan mengurutkan daftar mahasiswa berdasarkan nrp menggunakan algoritma selection short dan mrlihat bagaimana data mahasiswa diurutkan.

## ALGORITMA PENGURUTAN

### (QUICK DAN MERGE)

Tujuan praktikum :

1. Memahami mengenai algoritma pengurutan quick sort dan merge sort
2. Mampu mengimplementasikan algoritma pengurutan quick sort dan merge sort secara ascending dan descending.

Dasar Teori :

#### 1. Algoritma Quick Sort

Metode Quick Sort dikembangkan oleh C.A.R Hoare. Secara garis besar metode ini dijelaskan sebagai berikut. Misalnya kita ingin mengurutkan data A yang mempunyai N elemen. Kita pilih sembarang elemen dari data tersebut, biasanya elemen pertama, misalnya X. kemudian semua elemen tersebut disusun dengan menempatkan X pada posisi J sedemikian rupa sehingga elemen ke 1 sampai ke J-1 mempunyai nilai lebih kecil dari X dan elemen J+1 sampai ke N mempunyai nilai lebih besar dari X. Sampai saat ini kita sudah mempunyai dua sub data (kiri dan kanan).

#### 2. Algoritma Merge Sort

Algoritma merge ini disesuaikan untuk mesin drive tape. Penggunaannya dalam akses memori acak besar yang terkait telah menurun, karena banyak aplikasi algoritma merge yang mempunyai alternatif lebih cepat ketika kamu memiliki akses memori acak yang menjaga semua data mu. Hal ini disebabkan algoritma ini membutuhkan setidaknya ruang atau memori dua kali lebih besar karena dilakukan secara rekursif dan memakai dua tabel. Algoritma merge sort membagi tabel menjadi dua tabel yang sama besar. Masing-masing tabel diurutkan secara rekursif, dan kemudian digabungkan kembali untuk membentuk tabel yang terurut. Implementasi dasar dari algoritma merge sort memakai tiga buah tabel, dua untuk menyimpan elemen dari tabel yang telah di bagi dua dan satu untuk menyimpan elemen yang telah terurut. Namun algoritma ini dapat juga dilakukan langsung pada dua tabel, sehingga menghemat ruang atau memori yang dibutuhkan. Algoritma Merge umumnya memiliki satu set pointer  $p_0 \dots p_n$  yang menunjuk suatu posisi di dalam satu set daftar  $L_0 \dots L_n$ . Pada awalnya mereka menunjuk item yang pertama pada setiap daftar. Algoritmanya sebagai berikut: Selama  $p_0 \dots p_n$  masih menunjuk data yang di dalam sebagai pengganti pada akhirnya:

- Melakukan sesuatu dengan data item yang menunjuk daftar mereka masing-masing.
- Menemukan pointers points untuk item dengan kunci yang paling rendah; membantu salah satu pointer untuk item yang berikutnya dalam daftar.

Tugas Pendahuluan :

1. Tuliskan algoritma pengurutan quick sort secara ascending.

Jawab :

- a. Mengambil array yang akan diurutkan
  - b. Jika Panjang array kurang dari atau sama dengan 1 kembalikan array
  - c. Pilih elemen pivot dari array
  - d. Bagi array menjadi 2 sub : lebih kecil dari pivot & lebih besar dari pivot
  - e. Terapkan Langkah 2 hingga 4 secara rekursif pada kedua sub array
  - f. Gabungkan dari kedua sub array dan pivot membentuk array terurut
  - g. Selesai
2. Tuliskan algoritma pengurutan merge sort secara ascending

Jawab :

- a. Mengambil array yang akan diurutkan
- b. Jika panjang array kurang dari atau sama dengan satu kembalikan
- c. Bagi menjadi 2 sub : sub array kiri ( dari indeks awal hingga Tengah) & sub array kanan (dari tengah + 1 hingga akhir)
- d. Terapkan Langkah 2 & 3 secara rekursif pada kedua sub array
- e. Gabungkan ke2 sub menjadi satu
- f. Selesai

Percobaan :

1. Quick Sort secara ascending

```

2. void main() {
3.     List<num> data = List.generate(10, (index) => (index +
4.         1)..toDouble());
5.     printData(data);
6.     DateTime startTime = DateTime.now();
7.     quickSort(data, 0, data.length - 1);
8.     Duration elapsedTime = DateTime.now().difference(startTime);
9.
10.    printData(data);
11.    print('Waktu ${elapsedTime.inMilliseconds}');
12.}
13.
14. void quickSort<T extends Comparable<T>>(List<T> arr, int p, int r) {
15.     int q;
16.     if (p < r) {
17.         q = partition(arr, p, r);
18.         quickSort(arr, p, q);
19.         quickSort(arr, q + 1, r);
20.     }
21.}
22.
23. int partition<T extends Comparable<T>>(List<T> arr, int p, int r) {
24.     int i, j;
25.     T pivot, temp;
26.     pivot = arr[p];

```

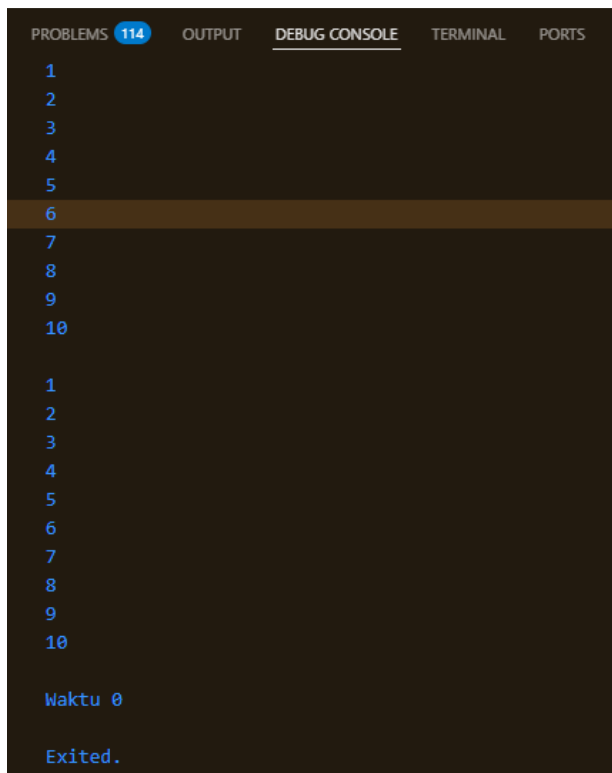


```

27. i = p;
28. j = r;
29. while (i <= j) {
30.     while (pivot.compareTo(arr[j]) < 0) j--;
31.     while (pivot.compareTo(arr[i]) > 0) i++;
32.     if (i < j) {
33.         temp = arr[i];
34.         arr[i] = arr[j];
35.         arr[j] = temp;
36.         j--;
37.         i++;
38.     } else {
39.         return j;
40.     }
41. }
42. return j;
43.}
44.
45.void printData<T>(List<T> data) {
46.    for (T objek in data) {
47.        print('$objek ');
48.    }
49.    print('');
50.}
51.

```

Hasil run :



```

PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
1
2
3
4
5
6
7
8
9
10

1
2
3
4
5
6
7
8
9
10

Waktu 0
Exited.

```

## 2. Merge Sort secara ascending

```
void main() {
    List<num> data = List.generate(10, (index) => (index + 1)..toDouble());
    printData(data);

    DateTime startTime = DateTime.now();
    mergeSort(data, 0, data.length - 1);
    Duration elapsedTime = DateTime.now().difference(startTime);

    printData(data);
    print('\nwaktu : ${elapsedTime.inMilliseconds}');
}

void mergeSort<T extends Comparable<T>>(List<T> arr, int l, int r) {
    int med;
    if (l < r) {
        med = (l + r) ~/ 2;
        mergeSort(arr, l, med);
        mergeSort(arr, med + 1, r);
        merge(arr, l, med, r);
    }
}

void merge<T extends Comparable<T>>(
    List<T> arr, int left, int median, int right) {
    List<T?> temp = List.filled(arr.length, null);

    int kiri1, kanan1, kiri2, kanan2, i, j;
    kiri1 = left;
    kanan1 = median;
    kiri2 = median + 1;
    kanan2 = right;
    i = left;

    while ((kiri1 <= kanan1) && (kiri2 <= kanan2)) {
        if (arr[kiri1].compareTo(arr[kiri2]) <= 0) {
            temp[i] = arr[kiri1];
            kiri1++;
        } else {
            temp[i] = arr[kiri2];
            kiri2++;
        }
        i++;
    }

    while (kiri1 <= kanan1) {
        temp[i] = arr[kiri1];
        kiri1++;
    }
}
```

```

        i++;
    }

    while (kiri2 <= kanan2) {
        temp[i] = arr[kiri2];
        kiri2++;
        i++;
    }

    j = left;
    while (j <= right) {
        arr[j] = temp[j]!;
        j++;
    }
}

void printData<T>(List<T> data) {
    for (T objek in data) {
        print('$objek ');
    }
    print('');
}

```

Hsil run :

```

PROBLEMS 114 OUTPUT DEBUG CONSOLE TERMINAL PORTS
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
2
waktu : 1
Exited.

```

Latihan :

3. Buatlah class Mahasiswa dengan variable nrp dan nama yang memiliki tipe String! Lakukan perbandingan dua objek mahasiswa berdasarkan nrp.

```
4. class Mahasiswa {
5.     String nrp;
6.     String nama;
7.
8.     // Konstruktor untuk inisialisasi nrp dan nama
9.     Mahasiswa(this.nrp, this.nama);
10.
11.    // Override metode toString untuk representasi objek
12.    @Override
13.    String toString() {
14.        return 'Mahasiswa{nrp: $nrp, nama: $nama}';
15.    }
16.
17.    // Metode untuk membandingkan dua objek Mahasiswa berdasarkan NRP
18.    int compareTo(Mahasiswa other) {
19.        return this.nrp.compareTo(other.nrp);
20.    }
21.}
22.
23.void main() {
24.    // Membuat beberapa objek Mahasiswa
25.    Mahasiswa mhs1 = Mahasiswa('443230', 'Andini');
26.    Mahasiswa mhs2 = Mahasiswa('443240', 'Reyhan');
27.    Mahasiswa mhs3 = Mahasiswa('443210', 'Dimas');
28.
29.    // Menampilkan objek Mahasiswa
30.    print(mhs1); // Output: Mahasiswa{nrp: 443230, nama: Andini}
31.    print(mhs2); // Output: Mahasiswa{nrp: 443240, nama: Reyhan}
32.
33.    // Melakukan perbandingan
34.    if (mhs1.compareTo(mhs2) < 0) {
35.        print('${mhs1.nama} memiliki NRP lebih kecil dari ${mhs2.nama}');
36.    } else if (mhs1.compareTo(mhs2) > 0) {
37.        print('${mhs1.nama} memiliki NRP lebih besar dari ${mhs2.nama}');
38.    } else {
39.        print('${mhs1.nama} dan ${mhs2.nama} memiliki NRP yang sama');
40.    }
41.
42.    // Contoh perbandingan yang sama
43.    if (mhs1.compareTo(mhs3) == 0) {
44.        print('${mhs1.nama} dan ${mhs3.nama} memiliki NRP yang sama');
45.    }
46.}
```

Hasil run :

```
PROBLEMS 117 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin> dart run nomor3.dart
Mahasiswa{nrp: 443230, nama: Andini}
Mahasiswa{nrp: 443240, nama: Reyhan}
Andini memiliki NRP lebih kecil dari Reyhan
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin> []
```

Kesimpulan : kode ini mengelola data mahasiswa dan melakukan perbandingan dengan nrp yang menentukan lebih besar atau lebih kecil dan bisa melihat hasil dari perbandingannya.

6. Lakukan percobaan untuk melihat waktu running untuk setiap algoritma sorting dengan data acak yang sama seperti table di bawah ini dan gambarkan dalam bentuk grafik (boleh menggunakan Excel atau aplikasi untuk visualisasi lainnya). Tabel 1. Analisa Perbandingan metode pengurutan berdasarkan waktu running

```
import 'dart:math';

void main() {
  List<int> datasetSizes = [
    50000, 100000, 150000, 200000, 250000,
    300000, 350000, 400000, 450000, 500000
  ];

  for (var size in datasetSizes) {
    List<int> data = generateRandomArray(size);

    print("\n--- Ukuran Data: $size ---");

    measureSortTime("Insertion", insertionSort, List<int>.from(data));
    measureSortTime("Selection", selectionSort, List<int>.from(data));
    measureSortTime("Bubble", bubbleSort, List<int>.from(data));
    measureSortTime("Shell", shellSort, List<int>.from(data));
    measureSortTime("Quick", quickSort, List<int>.from(data));
    measureSortTime("Merge", mergeSort, List<int>.from(data));
  }
}

List<int> generateRandomArray(int size) {
  Random random = Random();
  return List<int>.generate(size, (_) => random.nextInt(1000000));
}

void measureSortTime(String name, Function(List<int>) sortFunction, List<int> data) {
  DateTime start = DateTime.now();
  sortFunction(data);
  Duration elapsedTime = DateTime.now().difference(start);
}
```

```

    print("$name Sort: ${elapsedTime.inMilliseconds} ms");
}

// Implementasi Sorting

void insertionSort(List<int> arr) {
    for (int i = 1; i < arr.length; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void selectionSort(List<int> arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        int temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }
}

void bubbleSort(List<int> arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void shellSort(List<int> arr) {
    int n = arr.length;
    for (int gap = n ~/ 2; gap > 0; gap ~/= 2) {

```

```

        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

```

```

void quickSort(List<int> arr, [int left = 0, int? right]) {
    right ??= arr.length - 1;
    if (left < right) {
        int pivotIndex = partition(arr, left, right);
        quickSort(arr, left, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, right);
    }
}

```

```

int partition(List<int> arr, int left, int right) {
    int pivot = arr[right];
    int i = left - 1;
    for (int j = left; j < right; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[right];
    arr[right] = temp;
    return i + 1;
}

```

```

void mergeSort(List<int> arr, [int left = 0, int? right]) {
    right ??= arr.length - 1;
    if (left < right) {
        int mid = (left + right) ~/ 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

```

```

void merge(List<int> arr, int left, int mid, int right) {

```

```

List<int> leftArr = arr.sublist(left, mid + 1);
List<int> rightArr = arr.sublist(mid + 1, right + 1);

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

while (i < leftArr.length && j < rightArr.length) {
    if (leftArr[i] <= rightArr[j]) {
        arr[k++] = leftArr[i++];
    } else {
        arr[k++] = rightArr[j++];
    }
}

while (i < leftArr.length) {
    arr[k++] = leftArr[i++];
}

while (j < rightArr.length) {
    arr[k++] = rightArr[j++];
}
}

```

Hasil run :

Jumlah Data	Insertion Short	Selection Short	Buble Short	Shell Short	Quick Short	Merge Short
<b>50000</b>	906 ms	2651 ms	7538 ms	17 ms	14 ms	30 ms
<b>100000</b>	4134 ms	9771 ms	29570 ms	19 ms	10 ms	31 ms
<b>150000</b>	8038 ms	16973 ms	84460 ms	53 ms	39 ms	73 ms
<b>200000</b>	15867 ms	27979 ms	104394 ms	53 ms	32 ms	70 ms
<b>250000</b>	21236 ms	44720 ms	191288 ms	70 ms	36 ms	78 ms
<b>300000</b>	33117 ms	67374 ms	247895 ms	95 ms	53 ms	95 ms
<b>350000</b>	471327 ms	107557 ms	343411 ms	181 ms	61 ms	113 ms
<b>400000</b>	56667 ms	115554 ms	450801 ms	177ms	72 ms	114 ms
<b>450000</b>	83390 ms	163924 ms	535500 ms	162 ms	99 ms	171 ms
<b>500000</b>	104406	202205 ms	836525 ms	139 ms	87ms	151 ms

REFERENSI



```
--- Ukuran Data: 50000 ---
Insertion Sort: 906 ms
Selection Sort: 2651 ms
Bubble Sort: 7538 ms
Shell Sort: 17 ms
Quick Sort: 14 ms
Merge Sort: 30 ms

--- Ukuran Data: 100000 ---
Insertion Sort: 4134 ms
Selection Sort: 9771 ms
Bubble Sort: 29570 ms
Shell Sort: 19 ms
Quick Sort: 10 ms
Merge Sort: 31 ms

--- Ukuran Data: 150000 ---
Insertion Sort: 8038 ms
Selection Sort: 16973 ms
Bubble Sort: 84460 ms
Shell Sort: 53 ms
Quick Sort: 39 ms
Merge Sort: 73 ms

--- Ukuran Data: 200000 ---
Insertion Sort: 15867 ms
Selection Sort: 27979 ms
Bubble Sort: 104394 ms
Shell Sort: 53 ms
Quick Sort: 32 ms
Merge Sort: 70 ms

--- Ukuran Data: 250000 ---
Insertion Sort: 21236 ms
Selection Sort: 44720 ms
Bubble Sort: 191288 ms
Shell Sort: 70 ms
Quick Sort: 36 ms
Merge Sort: 78 ms

--- Ukuran Data: 300000 ---
Insertion Sort: 33117 ms
Selection Sort: 67374 ms
Bubble Sort: 247895 ms
Shell Sort: 95 ms
Quick Sort: 53 ms
Merge Sort: 95 ms
```

```
--- Ukuran Data: 350000 ---  
Insertion Sort: 47137 ms  
Selection Sort: 107557 ms  
Bubble Sort: 343411 ms  
Shell Sort: 181 ms  
Quick Sort: 61 ms  
Merge Sort: 113 ms  
  
--- Ukuran Data: 400000 ---  
Insertion Sort: 56667 ms  
Selection Sort: 115554 ms  
Bubble Sort: 450801 ms  
Shell Sort: 117 ms  
Quick Sort: 72 ms  
Merge Sort: 114 ms  
  
--- Ukuran Data: 450000 ---  
Insertion Sort: 83390 ms  
Selection Sort: 163924 ms  
Bubble Sort: 535500 ms  
Shell Sort: 162 ms  
Quick Sort: 99 ms  
Merge Sort: 171 ms  
  
--- Ukuran Data: 500000 ---  
Insertion Sort: 104406 ms  
Selection Sort: 202205 ms  
Bubble Sort: 836525 ms  
Shell Sort: 139 ms  
Quick Sort: 87 ms  
Merge Sort: 151 ms  
PS C:\dartsdk-windows-x64-release\sari\dart_application_1\bin> --
```

<https://medium.com/@antonio.tioypedro1234/5-sorting-algorithms-in-dart-c6aad21d5132>

<https://media.neliti.com/media/publications/126448-ID-analisis-perbandingan-algoritma-bubble-s.pdf>

<https://kumparan.com/how-to-teknologi/contoh-merge-sort-pengertian-beserta-cara-kerjanya-20Qu12Ui8J7>