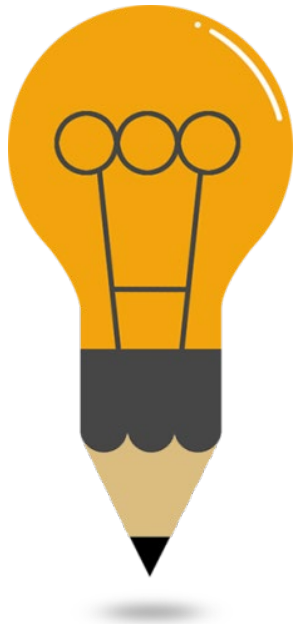
STRUKTUR DATA

Pertemuan 12

Agenda Pertemuan



1. Konsep Sorting (Pengurutan)
2. Insertion Sort, Selection Sort, Merge Sort,Bubble Sort

Konsep Sorting (Pengurutan)

- Dapat diartikan sebagai proses untuk menyusun elemen-elemen/data dalam urutan tertentu
- Ingat pertemuan sebelumnya tentang pencarian (searching), jika data telah terurut maka proses pencarian dapat **lebih efisien.**

- Data yang diurutkan dapat berupa numerik ataupun karakter
- Jenis pengurutan data:
  - **Ascending (dari nilai terkecil ke terbesar) >> kita akan membahas ini**
  - Descending (dari nilai terbesar ke terkecil)
- Untuk memudahkan pemahaman konsep, dalam pembahasan materi ini, kita akan mengambil array sebagai input, melakukan operasi tertentu pada array dan menampilkan array yang telah diurutkan.

Metode Sorting

Beberapa metode pengurutan yang akan kita bahas sebagai berikut:

- Insertion Sort (Metode Penyisipan)
- Selection Sort (Metode Seleksi)
- Merge Sort (Metode Penggabungan)
- Bubble Sort(Metode Gelembung)

Insertion Sort

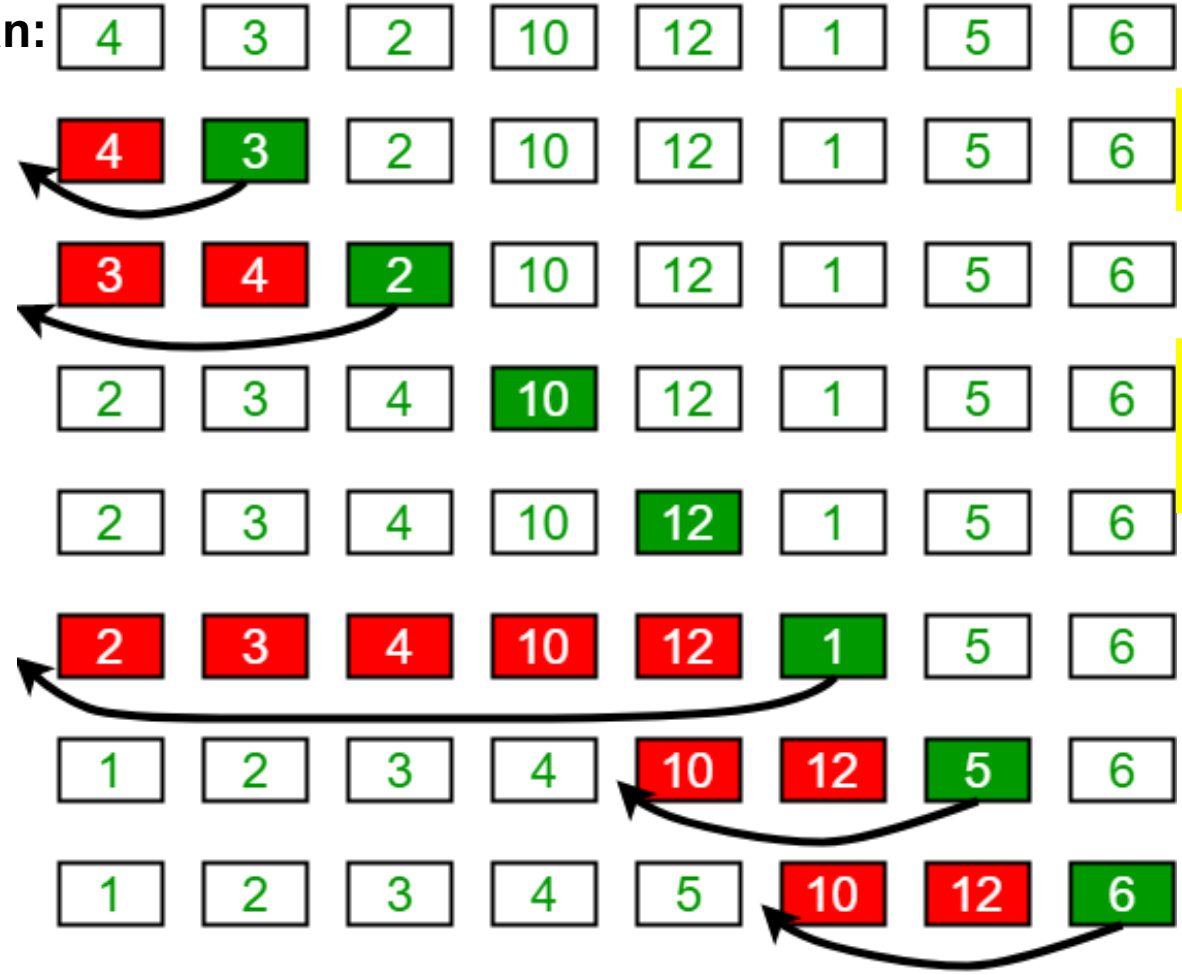
- Pengurutan yang dilakukan dengan cara menyisipkan elemen pada posisi yang sudah ditentukan atau yang seharusnya.
- Data dicek satu per satu mulai **dari yang kedua sampai dengan yang terakhir**. Apabila ditemukan elemen yang lebih kecil daripada elemen-elemen sebelumnya, maka **elemen tersebut disisipkan pada posisi yang sesuai**.

6 5 3 1 8 7 2 4

**slide show untuk menampilkan gif*

Ilustrasi Insertion Sort

Data yang akan diurutkan:



Cek elemen ke-2 dan tempatkan pada posisi yang sesuai

Cek elemen ke-4, posisi sudah tepat jadi tidak perlu disisipkan di posisi lain

Data yang telah urut:

1 2 3 4 5 6 10 12

Contoh Program dengan Prosedur Insertion Sort

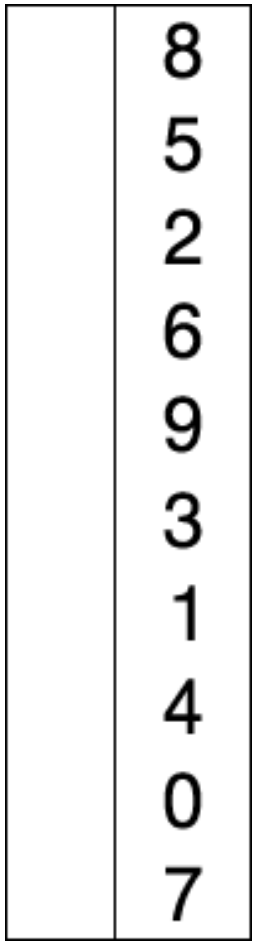
```
void insertionSort(int array[], int size)
{
    int i, key, j;
    for (i = 1; i < size; i++) {
        key = array[i];
        j = i - 1;

        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
```

Mengecek elemen satu per satu dari ke-2 sampai terakhir

Tempatkan pada posisi yang seharusnya

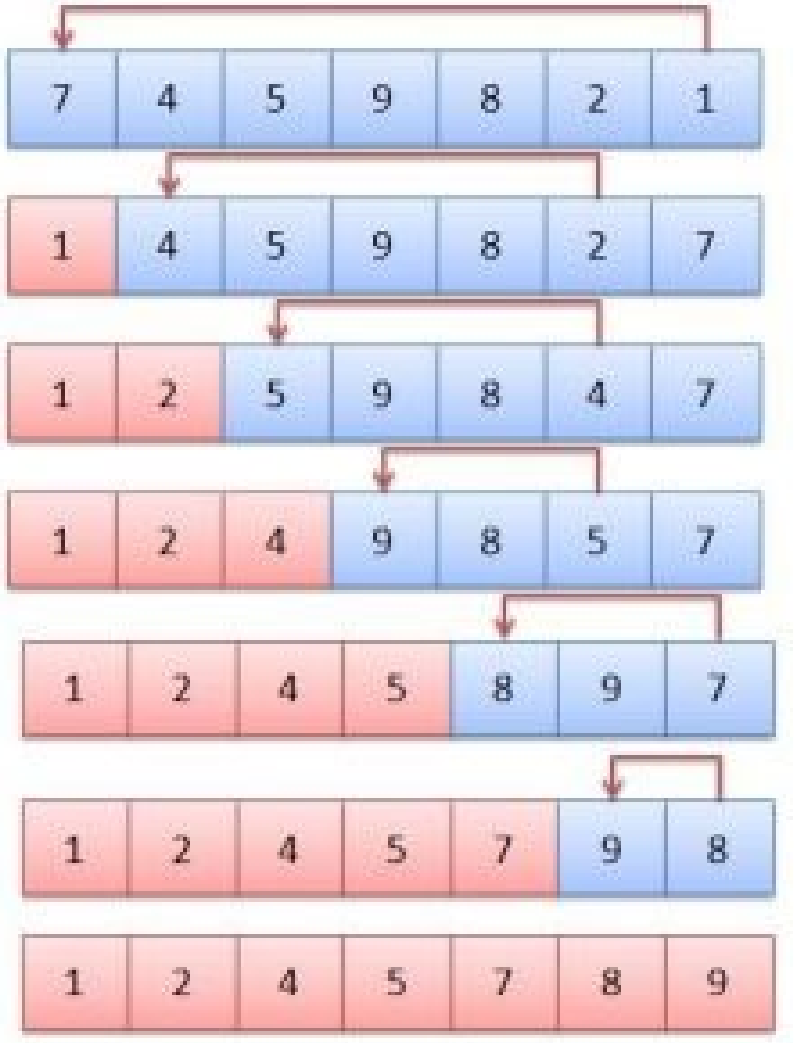
Selection Sort

- Dalam prosesnya, algoritma ini akan mempunyai 2 sub array yang menyimpan:
  - Bagian yang sudah diurutkan
  - Bagian yang belum diurutkan
- Pengurutan dilakukan dengan memilih elemen dengan nilai paling rendah dari bagian yang belum diurutkan dan menempatkan di awal bagian yang belum diurutkan.



**slide show untuk menampilkan gif*

Ilustrasi Selection Sort



Bagian yang sudah diurutkan

Bagian yang belum diurutkan

1. Langkah pertama dicari data terkecil dari data pertama sampai data terakhir. Kemudian data terkecil ditukar dengan data pertama. Dengan demikian, data pertama sekarang mempunyai nilai paling kecil dibanding data yang lain.
2. Langkah kedua, kita cari data terkecil mulai dari data kedua sampai terakhir. Data terkecil yang kita peroleh ditukar dengan data kedua dan demikian seterusnya sampai semua elemen dalam keadaan terurutkan.

Contoh Program dengan Prosedur Selection Sort

```
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void selectionSort(int array[], int size)
{
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {
      if (array[i] < array[min_idx]){
        min_idx = i;
      }
    }

    swap(&array[min_idx], &array[step]);
  }
}
```

Memilih elemen dengan nilai paling rendah di bagian yang belum diurutkan

Menempatkan nilai terendah (`array[min_idx]`) di awal bagian yang belum diurutkan

Merge Sort

- Algoritma ini dirancang untuk memenuhi kebutuhan pengurutan data yang tidak memungkinkan untuk ditampung dalam memori komputer karena ukurannya yang terlalu besar.
- Menggunakan cara ***divide and conquer*** yaitu dengan memecah, kemudian menyelesaikan setiap bagian, kemudian menggabungkannya kembali.
    1. Pertama data **dipecah menjadi 2 bagian** dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data,
    2. Kemudian dilakukan pemecahan kembali untuk masing-masing blok **sampai hanya terdiri dari satu data tiap blok**.

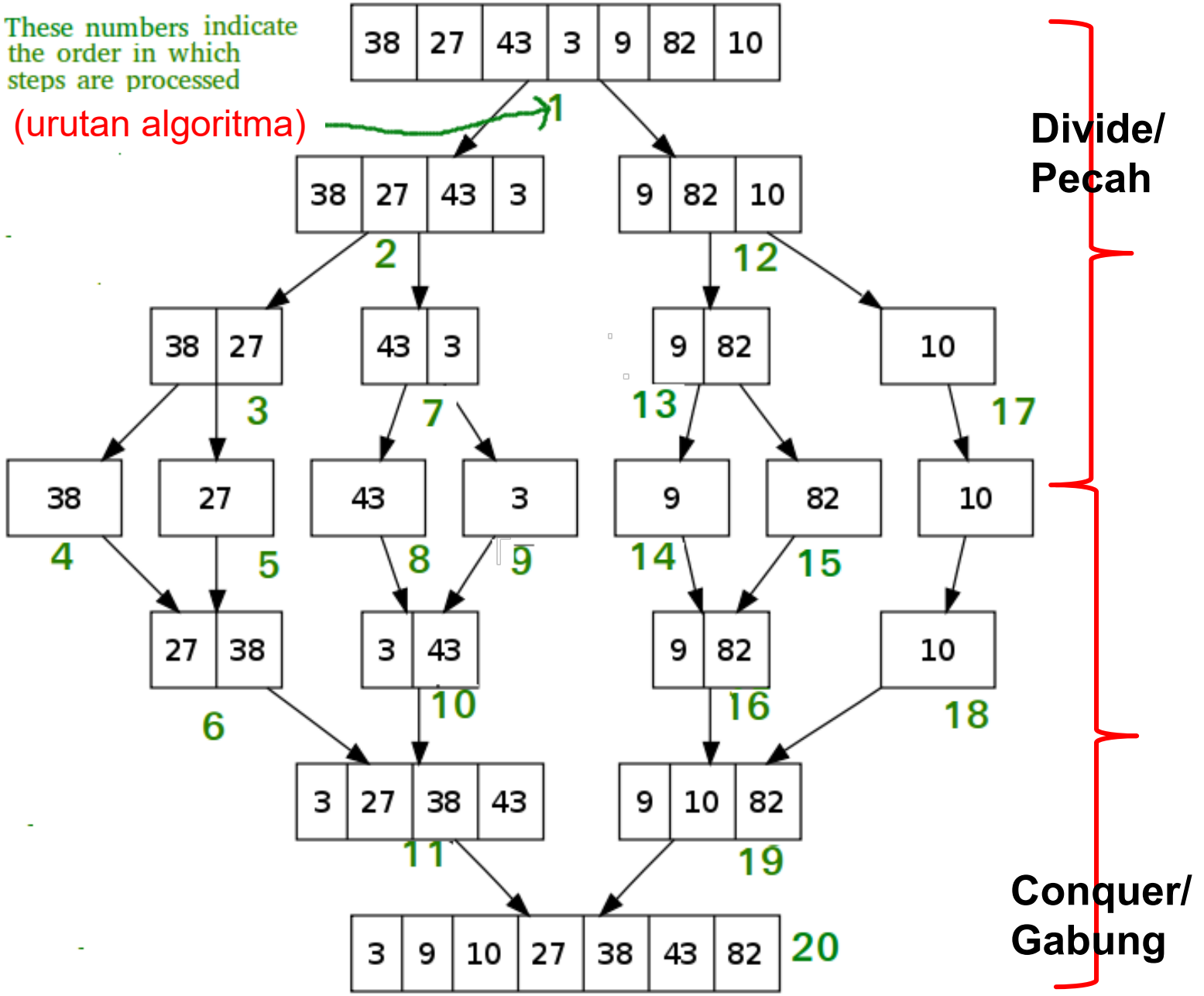
Merge Sort

3. Setelah itu digabungkan kembali dengan **membandingkan data pada blok i** apakah data **lebih besar daripada data di blok setelahnya i+1**, jika **ya** maka data blok setelahnya dipindah sebagai data pertama di blok yang baru (digabungkan). Demikian seterusnya sampai menjadi satu blok utuh seperti awalnya.

6 5 3 1 8 7 2 4

**slide show untuk menampilkan gif*

Ilustrasi Merge Sort



Contoh Program dengan Prosedur Merge Sort

```
void mergeSort(int arr[], int l, int r) {
  if (l < r) {

    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);

    merge(arr, l, m, r);
  }
}
```

Data dipecah menjadi 2 bagian sub array

Memecah secara rekursif bagian kiri: awal sampai tengah

Memecah secara rekursif bagian kanan: tengah sampai akhir

Menggabungkan 2 sub array yang telah terurut

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0; // Initial index of first sub array
    j = 0; // Initial index of second sub array
    k = l; // Initial index of merged sub array
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Buat temporary arrays

Copy data arr ke temporary array L dan R

Gabungkan 2 temporary array ke arr[1…r]

Copy sisa elemen L dan R jika ada ke arr

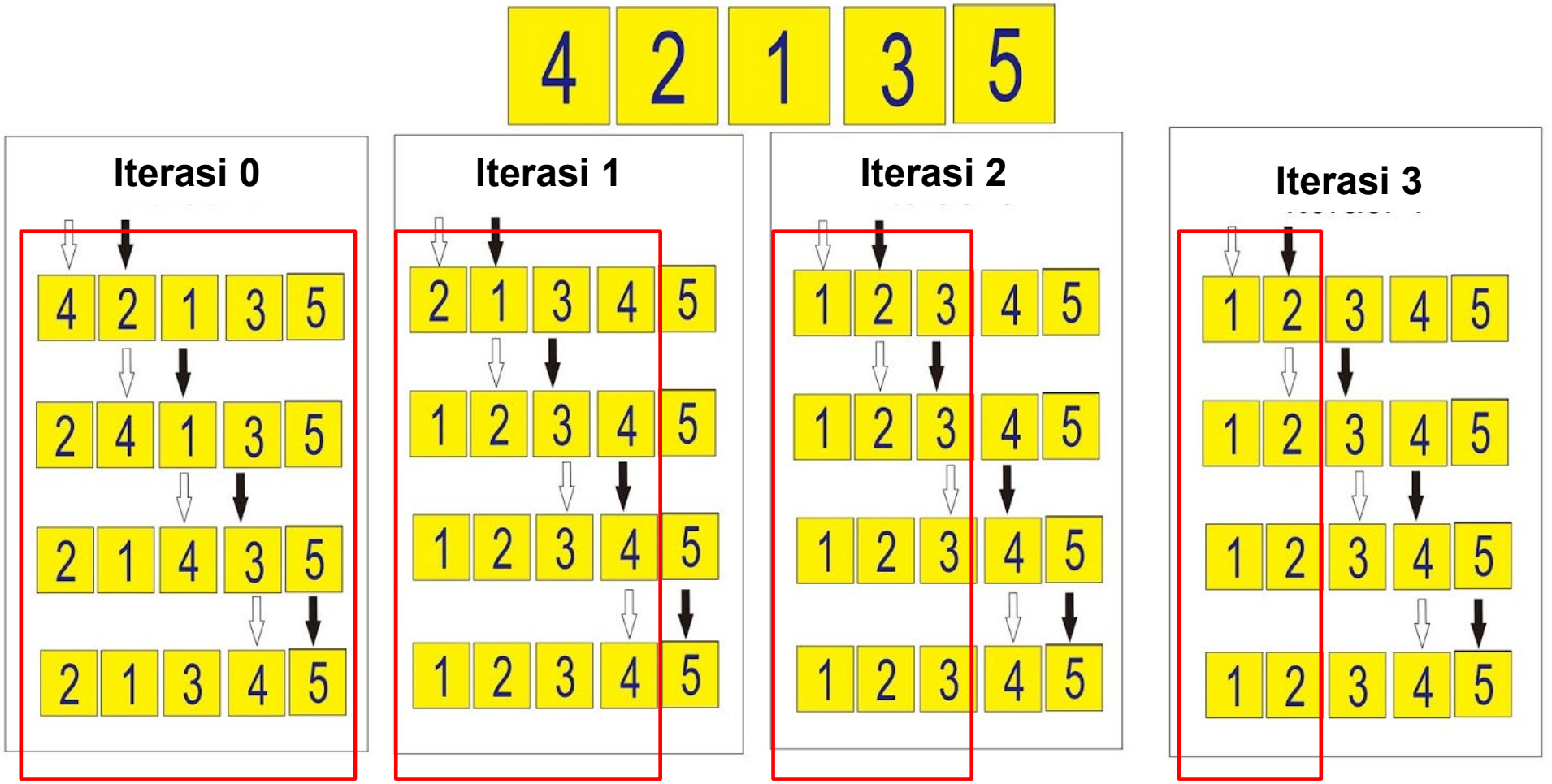
Bubble Sort

- Metode untuk mengurutkan data dengan cara membandingkan masing-masing elemen, apakah data sebelum dengan sesudahnya mana yang lebih besar atau kecil lalu ditukarkan bila perlu, secara terus menerus sampai data tersebut terurut.
- Mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien

6 5 3 1 8 7 2 4

**slide show untuk menampilkan gif*

Ilustrasi Bubble Sort



Contoh Program dengan Prosedur Bubble Sort

```
void bubbleSort(int array[], int size)
{
  for (int step = 0; step < size - 1; ++step) {
    for (int i = 0; i < size - step - 1; ++i) {
      if (array[i] > array[i + 1]) {
        swap(&array[i], &array[i+1]);
      }
    }
  }
}
```

Iterasi elemen-elemen di dalam array. Kenapa sampai size-step-1 saja? Karena elemen di size-step-1 sampai size pasti sudah urut

Membandingkan elemen sebelum dan sesudahnya

Perbandingan Algoritma Sorting

- Tidak ada algoritma terbaik untuk setiap situasi, semuanya tergantung situasi dari setiap masalah. Banyak penelitian membandingkan beberapa algoritma sorting, hasilnya pun berbeda-beda.
- Dilihat dari algoritmanya/langkah-langkahnya, berikut perbandingan pada kasus terbaik, rata-rata, dan terburuk setiap metode:

Algorithm	Data structure	Time complexity:Best	Time complexity:Average	Time complexity:Worst
Merge sort	Array	O(n log(n))	O(n log(n))	O(n log(n))
Bubble sort	Array	O(n)	O(n^2)	O(n^2)
Insertion sort	Array	O(n)	O(n^2)	O(n^2)
Selection sort	Array	O(n^2)	O(n^2)	O(n^2)

*Time complexity artinya jumlah langkah-langkah(lama waktu) yang dibutuhkan untuk menjalankan suatu algoritma dengan input sejumlah n. Cara membacanya: misal O(n^2) = misal jumlah elemen adalah n, maka algoritma membutuhkan sebanyak n^2 langkah.

TERIMA KASIH