



# ALGORITMA PENCARIAN (SEARCHING)

Dwi Ratna

# Topik

1. Linear Search
2. Binary Search

# Searching pada data

- Proses mendapatkan (*retrieve*) informasi berdasarkan kunci (*key*) tertentu dari sejumlah informasi yang telah disimpan.
- Proses pencarian data yang ada pada suatu deret data dengan cara menelusuri data-data tersebut.
- Kunci (*key*) digunakan untuk melakukan pencarian data yang diinginkan
- Tahapan paling penting pada searching: memeriksa jika data yang dicari sama dengan data yang ada pada deret data.

# Jenis searching :

- Single match : Pencarian yang menghasilkan satu data.

Contoh : mencari mahasiswa dengan nim "120651234"

- Multiple match : Pencarian yang memungkinkan menghasilkan beberapa data.

Contoh : mencari mahasiswa dengan ipk  $\geq 3.5$



# LINEAR SEARCH SEQUENTIAL SEARCH

# Linear Search

- Metode pencarian beruntun atau linear atau sequential search.
- Adalah suatu teknik pencarian data yang akan menelusuri tiap elemen satu per-satu dari awal sampai akhir.
- Suatu deret data dapat disimpan dalam bentuk array maupun linked list.

# Case

- **Best case** : jika data yang dicari terletak di indeks array terdepan (elemen array pertama) sehingga waktu yang dibutuhkan untuk pencarian data sangat sebentar (minimal).
- **Worst case** : jika data yang dicari terletak di indeks array terakhir (elemen array terakhir) sehingga waktu yang dibutuhkan untuk pencarian data sangat lama (maksimal).

# Contoh

- Misalnya terdapat array satu dimensi sebagai berikut:

0	1	2	3	4	5	6	7	indeks
8	10	6	-2	11	7	1	100	value

- Kemudian program akan meminta data yang akan dicari, misalnya **6**.
- Iterasi :
  - 6 = 8 (tidak!)
  - 6 = 10 (tidak!)
  - 6 = 6 (Ya!) => output : 2 (index)



# Algoritma

1.  $i \leftarrow 0$  ✓
2.  $ketemu \leftarrow \text{false}$
3. Selama (tidak  $ketemu$ ) dan ( $i < N$ ) kerjakan baris 4
4. Jika ( $Data[i] = key$ ) maka  
     $ketemu \leftarrow \text{true}$   
    jika tidak  
         $i \leftarrow i + 1$
5. Jika ( $ketemu$ ) maka  
     $i$  adalah indeks dari data yang dicari  
    jika tidak  
        data tidak ditemukan ✓

$ketemu \leftarrow \text{false}$  total data

loop  $i = 0 : N-1$   
indeks  $\leftarrow i$

# Q & A

- **Problem:** Apakah cara di atas efisien? Jika datanya ada 10000 dan semua data dipastikan unik?
- **Solution:** Untuk meningkatkan efisiensi, seharusnya jika data yang dicari sudah ditemukan maka perulangan harus dihentikan!
  - **Hint:** Gunakan **break**!
- **Question:** Bagaimana cara menghitung ada berapa data dalam array yang tidak unik, yang nilainya sama dengan data yang dicari oleh user?
  - **Hint:** Gunakan variabel counter yang nilainya akan selalu bertambah jika ada data yang ditemukan!

```

public class SequentialSearch {
    private String [] allData = new String[]{"A", "B", "C", "D", "E", "F"};
    private void tampilkanData(){
        for (String data : allData) {
            System.out.print(data + " ");
        }
        System.out.println();
    }

    private void searching(String karakter){
        int x = 0;
        boolean ketemu = false;
        for (int i = x; i < allData.length; i++) {
            if(karakter.equals(allData[i])){
                ketemu = true;
                x = i;
            }
        }
        if(ketemu){
            System.out.println("Data berada pada urutan ke - "+(x+1));
        } else {
            System.out.println("Data Tidak Ditemukan");
        }
    }
}

```

# Contoh

```

public static void main(String[] args) {
    SequentialSearch obj = new
    SequentialSearch();

    // Untuk menampilkan data pada Array
    obj.tampilkanData();

    // Melakukan pencarian data
    obj.searching("C");
}

```



# BINARY SEARCH

- Pencarian data dimulai dari pertengahan data yang telah terurut.
- Jika kunci pencarian lebih kecil daripada kunci posisi tengah, maka kurangi lingkup pencarian pada separuh data pertama.
- Begitu juga sebaliknya jika kunci pencarian lebih besar daripada kunci tengah, maka pencarian ke separuh data kedua.
- Teknik Binary Search hanya dapat digunakan pada sorted array.

# Binary Search

- Menggunakan Binary Search, jika :
  - Nilai-nilai tersebut sudah berurutan (ascending). Disimpan dalam bentuk larik (*array*) atau struktur data sejenis.

# Ilustrasi

## Contoh Data:

Misalnya data yang dicari **17**

• 0	1	2	3	4	5	6	7	8
• 3	9	11	12	15	17	23	31	35
• A				B				C

- Karena  $17 > 15$  (data tengah), maka: awal = tengah + 1

• 0	1	2	3	4	5	6	7	8
• 3	9	11	12	15	17	23	31	35
•				A B			C	

- Karena  $17 < 23$  (data tengah), maka: akhir = tengah - 1

• 0	1	2	3	4	5	6	7	8
• 3	9	11	12	15	17	23	31	35
•					A=B=C			

- Karena  $17 = 17$  (data tengah), maka KETEMU!

# Algoritma Binary Search

1. Data diambil dari posisi 1 sampai posisi akhir N
2. Kemudian cari posisi data tengah dengan rumus: **(posisi awal + posisi akhir) / 2**
3. Kemudian data yang dicari dibandingkan dengan data yang di tengah, apakah sama atau lebih kecil, atau lebih besar?
4. Jika lebih besar, maka proses pencarian dicari dengan posisi awal adalah **posisi tengah + 1**
5. Jika lebih kecil, maka proses pencarian dicari dengan posisi akhir adalah **posisi tengah - 1**
6. Jika data sama, berarti ketemu.



# CONTOH

```
public class BinarySearch {  
    private final int [] data = {5, 9, 12, 15, 17, 23, 27, 38, 42, 54, 64, 78, 90};  
  
    public String pencarianBinary(int key) {  
        int bawah = 0;  
        int atas = data.length - 1;  
  
        while (atas >= bawah) {  
            int tengah = (bawah + atas) / 2;  
            if (key < data[tengah]){  
                atas = tengah - 1;  
            } else if (key == data[tengah]){  
                return "Nomor "+key+" Berada Pada Urutan Ke - "+(tengah+1);  
            } else{  
                bawah = tengah + 1;  
            }  
        }  
        return "Data Tidak Ditemukan";  
    }  
}
```

# Contoh (Lanjut.....)

```
private void tampilData(){
    for (int i : data) {
        System.out.print(i+" ");
    }
    System.out.println();
}

public static void main(String args []){
    BinarySearch obj = new BinarySearch();
    obj.tampilData();
    System.out.println(obj.pencarianBinary(8));
}
```