



# Pencarian

Tim Olimpiade Komputer Indonesia

# Pendahuluan

Melalui dokumen ini, kalian akan:

- Mempelajari konsep algoritma sederhana.
- Memahami berbagai algoritma pencarian.



## Pendahuluan (lanj.)

- Pencarian merupakan hal sederhana dan sering digunakan dalam pemrograman.
- Terdapat berbagai macam cara untuk melakukan pencarian.
- Membahas beberapa cara tersebut dapat memberikan gambaran bagaimana sebuah persoalan diselesaikan dengan berbagai algoritma.



# Soal: Sepatu Untuk Bebek

Deskripsi:

- Kwek, salah satu bebek Pak Dengklek akan segera merayakan ulang tahunnya. Pak Dengklek akan memberikan Kwek hadiah ulang tahun berupa sepatu.
- Terdapat  $N$  sepatu di toko. Sepatu ke- $i$  memiliki ukuran sebesar  $h_i$ .
- Pak Dengklek tahu bahwa ukuran kaki Kwek adalah sebuah bilangan bulat  $X$ .
- Karena  $N$  bisa jadi sangat besar, Pak Dengklek meminta bantuan kalian untuk mencari sepatu beberapa yang cocok dengan ukuran kaki Kwek.
- Bantulah dia!



## Soal: Sepatu Untuk Bebek (lanj.)

Format masukan:

- Baris pertama berisi dua bilangan bulat, yaitu  $N$  dan  $X$ .
- Baris kedua berisi  $N$  bilangan bulat. Bilangan ke- $i$  menyatakan  $h_i$ .

Format keluaran:

- Jika ada ukuran sepatu yang cocok, cetak sebuah baris yang menyatakan nomor sepatu yang cocok dengan ukuran kaki Kwek.
- Jika tidak ada ukuran sepatu yang cocok, cetak "beri hadiah lain".

Batasan:

- $1 \leq N \leq 100.000$
- $1 \leq X \leq 100.000$
- $1 \leq h_i \leq 100.000$ , untuk  $1 \leq i \leq N$



# Solusi

- Kita dihadapkan pada persoalan pencarian: diberikan  $N$  angka. Cari apakah  $X$  ada di antara angka-angka tersebut.
- Jika ditemukan, cetak di urutan seberapa angka tersebut berada.
- Salah satu ide yang muncul adalah:
  - Periksa satu per satu dari setiap pertama, kedua, ketiga, dan seterusnya.
  - Jika ditemukan, langsung laporkan.
  - Jika sampai akhir belum juga ditemukan, artinya angka yang dicari tidak ada pada daftar.



## Contoh Solusi: cari\_1.pas

Implementasinya cukup sederhana:

---

```
(* pencarian *)
hasil := 0; (* artinya belum ditemukan *)
for i := 1 to N do begin
    if (h[i] = X) then begin
        hasil := i;
        break;
    end;
end;

if (hasil = 0) then begin
    writeln('beri hadiah lain');
end else begin
    writeln(hasil);
end;
```

---



## Solusi (lanj.)

- Algoritma pencarian dengan membandingkan elemen satu per satu semacam ini disebut dengan *sequential search* atau *linear search*.
- Jika ada  $N$  elemen pada daftar yang perlu dicari, maka paling banyak diperlukan  $N$  operasi perbandingan.
- Dalam kompleksitas waktu, performa algoritma *sequential search* bisa dinyatakan dalam  $O(N)$ .





# Adakah yang Lebih Cepat?

- Misalkan terdapat 90.000 kata pada Kamus Besar Bahasa Indonesia (KBBI).
- Bayangkan jika sesekali kita butuh mencari arti kata pada kamus, dan dengan *sequential search* perlu dilakukan paling banyak 90.000 operasi.
- Apakah kita sebagai manusia melakukan perbandingan sampai 90.000 kata?



## Adakah yang Lebih Cepat? (lanj.)

- Pada kehidupan nyata, sebagai manusia kita tidak mencari kata satu per satu, halaman demi halaman, sampai 90.000 kata.
- Bahkan, kita dapat mencari arti suatu kata cukup dalam beberapa perbandingan, yang jauh lebih sedikit dari 90.000.
- Bagaimana hal ini bisa terjadi?



# Properti Khusus: Terurut

- KBBI memiliki sebuah properti yang khusus, yaitu **terurut berdasarkan abjad**.
- Dengan cara ini, kita bisa membuka halaman tengah dari KBBI, lalu periksa apakah kata yang kita cari ada pada halaman tersebut.



## Properti Khusus: Terurut (lanj.)

- Misalkan kata yang kita cari berawalan 'W', lalu halaman tengah hanya menunjukkan daftar kata dengan awalan 'G'. Jelas bahwa kata yang kita cari tidak mungkin berada di **separuh pertama** kamus.
- Ulangi hal serupa dengan separuh belakang kamus, buka bagian tengahnya dan bandingkan.
- Dengan cara ini, setiap perbandingan akan mengeleminasi separuh rentang pencarian!



# Analisis Kinerja

- Misalkan terdapat sebuah daftar berisi  $N$  elemen, dan kita perlu mencari salah satu elemen.
- Banyaknya operasi maksimal yang dibutuhkan sampai suatu elemen bisa dipastikan keberadaannya sama dengan panjang dari barisan  $[N, \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \dots, 2, 1]$ .
- Yakni sebesar  $\lceil \log N \rceil$ , sehingga kompleksitasnya  $O(\log N)$ .
- Pencarian seperti ini disebut dengan **binary search**.



## Perbandingan dengan Sequential Search

Perhatikan tabel banyaknya operasi yang dibutuhkan untuk nilai  $N$  tertentu:

N	Sequential	Binary
50	50	6
100	100	7
150	150	8
200	200	8
250	250	8
300	300	9

Seperti yang terlihat, *binary search* jauh lebih cepat!

Bahkan untuk  $N = 10^6$ , *binary search* hanya butuh maksimal 20 operasi!



## Implementasi (cari\_2.pas)

Terdapat bermacam cara implementasi *binary search*. Salah satunya:

---

```
(* pencarian *)
hasil := 0; (* artinya belum ditemukan *)
kiri := 1;
kanan := N;
while ((kiri <= kanan) and (hasil = 0)) do begin
    tengah := (kiri + kanan) div 2;

    if (X < h[tengah]) then begin
        kanan := tengah - 1;
    end else if (X > h[tengah]) then begin
        kiri := tengah + 1;
    end else begin
        hasil := tengah;
    end;
end;
```

```
end;
```



## Implementasi (cari\_2.pas) (lanj.)

---

```
if (hasil = 0) then begin
    writeln('beri hadiah lain');
end else begin
    writeln(hasil);
end;
```

---





## Penjelasan Implementasi

- Variabel **kiri** dan **kanan** menyatakan bahwa rentang pencarian kita ada di antara [kiri, kanan]. Nilai  $X$  *mungkin* ada di dalam sana.
- Kita mengambil nilai tengah dari kiri dan kanan, lalu periksa apakah  $X$  sama dengan  $h[tengah]$ .
- Jika ternyata  $X$  kurang dari  $h[tengah]$ , artinya  $X$  tidak mungkin berada pada rentang [tengah, kanan].
- Dengan demikian, rentang pencarian kita yang baru adalah [kiri, tengah-1].



## Penjelasan Implementasi (lanj.)

- Hal yang serupa juga terjadi ketika  $X$  lebih dari  $h[\text{tengah}]$ , rentang pencarian kita menjadi  $[\text{tengah}+1, \text{kanan}]$ .
- Tentu saja jika  $X$  sama dengan  $h[\text{tengah}]$ , catat posisinya dan pencarian berakhir.
- Untuk kasus  $X$  tidak ditemukan, pencarian akan terus dilakukan sampai **kiri** > **kanan**, yang artinya rentang pencarian **sudah habis**.



## Ilustrasi Eksekusi Algoritma

1    1    3    5    5    6    7

↑    ↑

kiri                                      kanan

Misalkan hendak dicari bilangan 3 dari [1, 1, 3, 5, 5, 6, 7]



## Ilustrasi Eksekusi Algoritma (lanj.)



Bilangan yang terletak di tengah adalah 5, yang lebih besar daripada 3. Artinya, jika bilangan 3 ada, pasti berada di antara *kiri* sampai dengan *tengah* - 1.



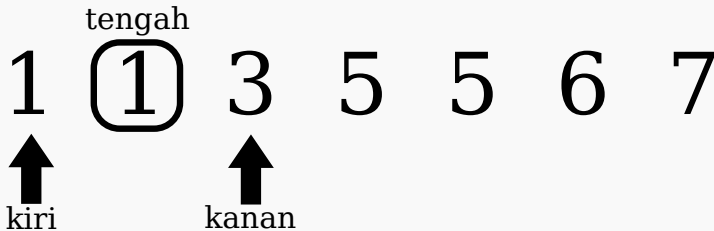
## Ilustrasi Eksekusi Algoritma (lanj.)



Sekarang rentang pencarian menjadi hanya separuh dari rentang awalnya.



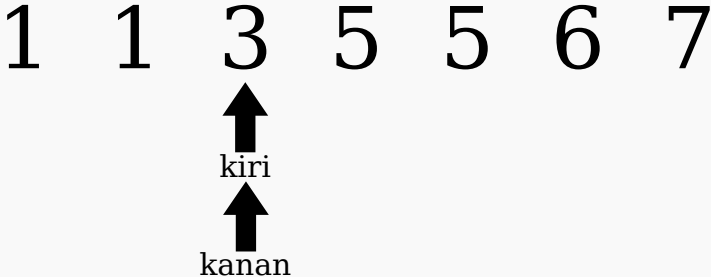
## Ilustrasi Eksekusi Algoritma (lanj.)



Bilangan yang terletak di tengah adalah 1, yang lebih kecil daripada 3. Artinya, jika bilangan 3 ada, pasti berada di antara *tengah* + 1 sampai dengan *kanan*.

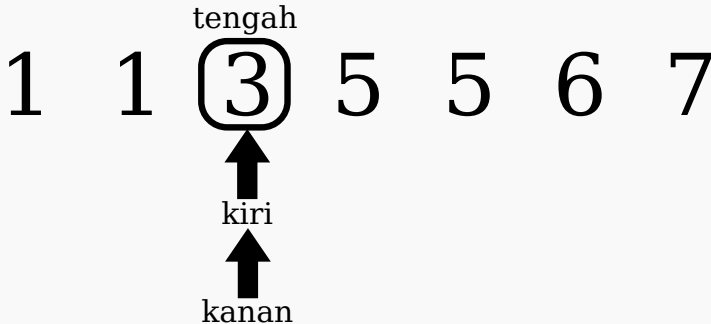


## Ilustrasi Eksekusi Algoritma (lanj.)



Kini rentang pencariannya menjadi tinggal satu elemen.

## Ilustrasi Eksekusi Algoritma (lanj.)



Ternyata elemen di tengah adalah bilangan 3, yaitu bilangan yang dicari.  
Dengan demikian, bilangan 3 ditemukan.



# Rangkuman

## Sequential Search

- Data untuk pencarian tidak perlu terurut.
- Kompleksitasnya  $O(N)$ , dengan  $N$  adalah ukuran data.
- Baik diimplementasikan jika pencarian hanya dilakukan sesekali.

## Binary Search

- Data untuk pencarian harus terurut.
- Kompleksitasnya  $O(\log N)$ , dengan  $N$  adalah ukuran data.
- Baik diimplementasikan jika pencarian perlu dilakukan berkali-kali.



# Data Tidak Terurut?

- Bagaimana jika kita memiliki data yang sangat banyak, tidak terurut, dan butuh pencarian yang efisien?
- Salah satu solusinya adalah dengan **mengurutkan** data tersebut, lalu mengaplikasikan *binary search* untuk setiap pencarian.
- Pengurutan akan dipelajari pada materi berikutnya.

