



STRUKTUR DATA & ALGORITMA

BIG-O NOTATION & SEARCHING



Pengertian *Big-O Notation* adalah suatu cara atau metode untuk melakukan analisa terhadap kinerja sebuah algoritma untuk menghitung kompleksitas dari algoritma yang dibuat (“*Big-O*” dapat juga dibaca sebagai “*order of*”).

Contoh penulisan :

$O(n^2)$ → dibaca **order n^2** ;

$O(n!)$ → **order n faktorial**. aplikasi yang dibuat.



Ada dua dimensi dalam menghitung kompleksitas algoritma pemrograman:

- 1) *Space Complexity* (kompleksitas ruang) yang berkaitan dengan berapa banyak ruang yang digunakan seperti memori ataupun harddisk komputer.
- 2) *Time Complexity* (kompleksitas waktu) yang berkaitan berapa lama baris kode dijalankan.

Tujuan dari analisa algoritma tersebut adalah untuk menumbuhkan kesadaran pengguna/ pembuat algoritma untuk mencari alternatif yang lebih baik sebelum data semakin besar dan berdampak kepada performa aplikasi yang dibuat.



Contoh kasus:

Jumlah Data	3	5	10	100	n
Jumlah Operasi	9	25	100	1.000	n²

Pada tabel di atas, dapat diketahui perbedaan jumlah operasi yang harus dilakukan pada setiap data yang ada. Ketika jumlah data 3 dan 5, perbedaan jumlah operasi bisa dianggap tidak signifikan, tetapi pada saat jumlah data meningkat menjadi 100, jumlah operasi meningkat tajam menjadi 1000. Hal tersebut tentulah tidak optimal, karena semakin banyak datanya akan menyebabkan meningkatnya jumlah operasi yang harus dilakukan secara signifikan.

Dibutuhkan jumlah operasi **n²** untuk setiap jumlah data **n**, kita menuliskannya dengan notasi **O(n²)**.



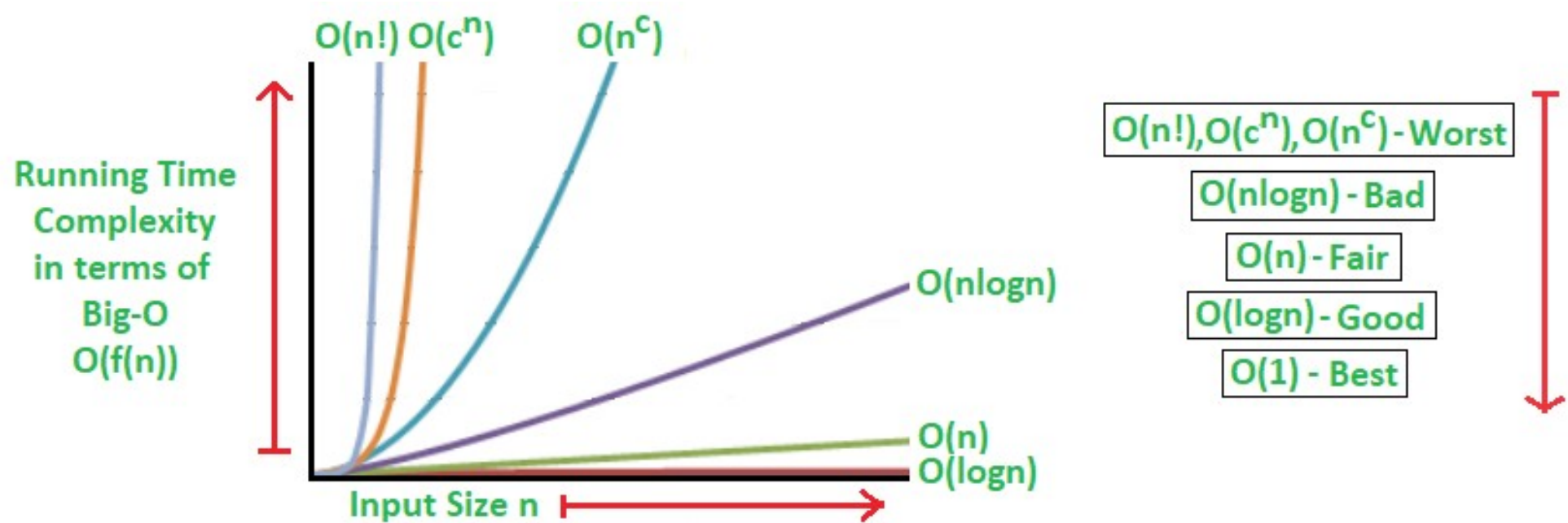
Ada 3 klasifikasi hasil pengukuran kompleksitas suatu algoritma, yaitu:

- 1) **Best Case** (kasus terbaik): jumlah operasi dan waktu yang dibutuhkan untuk menyelesaikan permasalahan sangat optimal/ efisien/ cepat.
- 2) **Average Case** (kasus rerata/ rata-rata): jumlah operasi dan waktu untuk menyelesaikan permasalahan membutuhkan tahapan/ sumber daya yang cukup.
- 3) **Worst Case** (kasus terburuk): jumlah operasi dan waktu yang dibutuhkan untuk menyelesaikan permasalahan sangat banyak/ lama.

Pada pemrograman, yang dimaksud dengan kasus terbaik, rerata atau kasus terburuk suatu algoritma adalah besar kecilnya atau banyak sedikitnya sumber-sumber yang digunakan oleh suatu algoritma. Makin sedikit makin baik, makin banyak makin buruk. Biasanya sumber-sumber yang paling dipertimbangkan tak hanya waktu eksekusi tetapi bisa juga besar memori, catu-daya dan sumber-sumber lain.



Menuliskan Big O



<https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/>



Beberapa fungsi *Big-O* yang biasa digunakan untuk menganalisa algoritma:

Notasi	Nama	Contoh Fungsi
$O(1)$	Konstan	Menentukan apakah suatu bilangan ganjil atau genap (tidak memerlukan pengulangan).
$O(\log n)$	Logaritmik	Pencarian dalam list terurut dengan <i>Binary Search Algorithm</i> (membagi permasalahan menjadi 2 bagian).
$O(n)$	Linear	Pencarian dalam list tidak terurut (kompleksitas bertumbuh selaras dengan pertumbuhan data).
$O(n \log n)$	Linearitmik	Mengurutkan list dengan Heapsort
$O(n^2)$	Kuadratik	Mengurutkan list dengan Insertion Sort
$O(nm)$	Polinomial	Pencarian <i>shortest path</i> dengan algoritma <i>Floyd-Warshall</i>
$O(n!)$	Faktorial	Menyelesaikan <i>traveling salesman problem</i> dengan menggunakan <i>brute force</i>.



Pencarian (*Searching*)



- ❖ Pencarian merupakan proses penting yang banyak dilakukan terhadap data.
- ❖ Selain mencari data, proses pencarian juga dapat mengetahui posisi data.
- ❖ Pengguna (*user*) dapat mencari data dalam sekumpulan data bertipe sama.



Pada banyak kasus, sering dijumpai pencarian tanpa kata kunci, pencarian hanya dilakukan dengan menggunakan kata yang ada atau kata yang diingatnya



Jika seorang *Customer Service* (CS) pada sebuah bank mencari data nasabah dengan menginput nama nasabahnya, misalnya “Budi”, maka yang akan ditampilkan oleh komputer bisa jadi semua nama yang mengandung kata “Budi”, seperti Budi Hermawan, Budiman, Budiarto, Sapta Budi dsb.

Hal tersebut kurang efektif dan mungkin akan menyita banyak waktu untuk mendapatkan orang yang dicarinya dengan tepat.



Pencarian (*Searching*)

- ✓ Untuk mempercepat pencarian data, pencarian dapat dilakukan dengan menggunakan kata kunci (*keyword*), dimana kata kunci tersebut merupakan kata yang unik (tidak ada kesamaannya).
- ✓ Jika seorang CS sebuah bank memasukkan nomor rekening nasabah, semua data tentang nasabah tersebut dapat dilihat olehnya.
- ✓ Pegawai administrasi kampus dapat mencari data mahasiswa dengan menginput Nomor Induk Mahasiswa (NIM).





Contoh Kasus Pencarian (*Searching*)

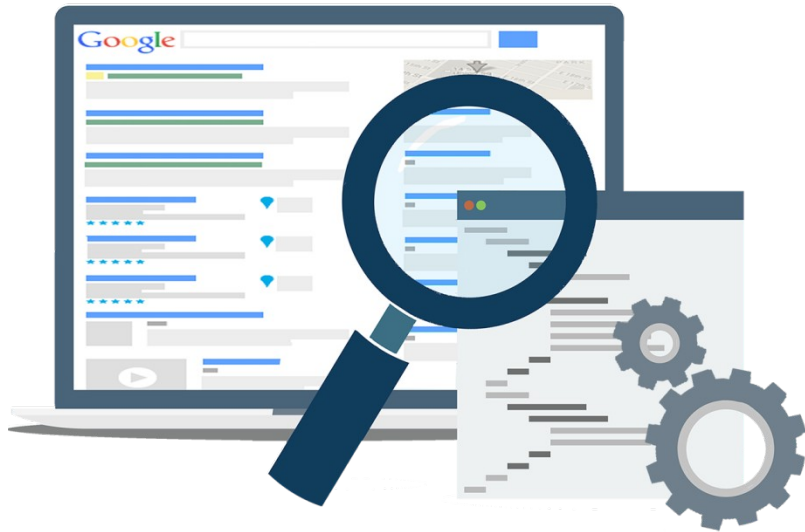


Digunakan untuk mencari kata atau data yang memiliki karakter yang sama



Ada dua teknik pencarian yang umum dilakukan, yaitu:

- 1) **Pencarian Linier/ Sekuensial (*Linear Search*)**: merupakan teknik yang efisien untuk mencari pada data yang acak (tidak urut).
- 2) **Pencarian Biner (*Binary Search*)**: merupakan teknik yang efisien untuk mencari pada data yang telah berurutan.



Linear Search



1. Pencarian Linier (*Linear Search*)

Pencarian Linier/ Sekuensial (*Linear Search*): merupakan teknik yang efisien untuk mencari pada data yang acak (tidak urut).

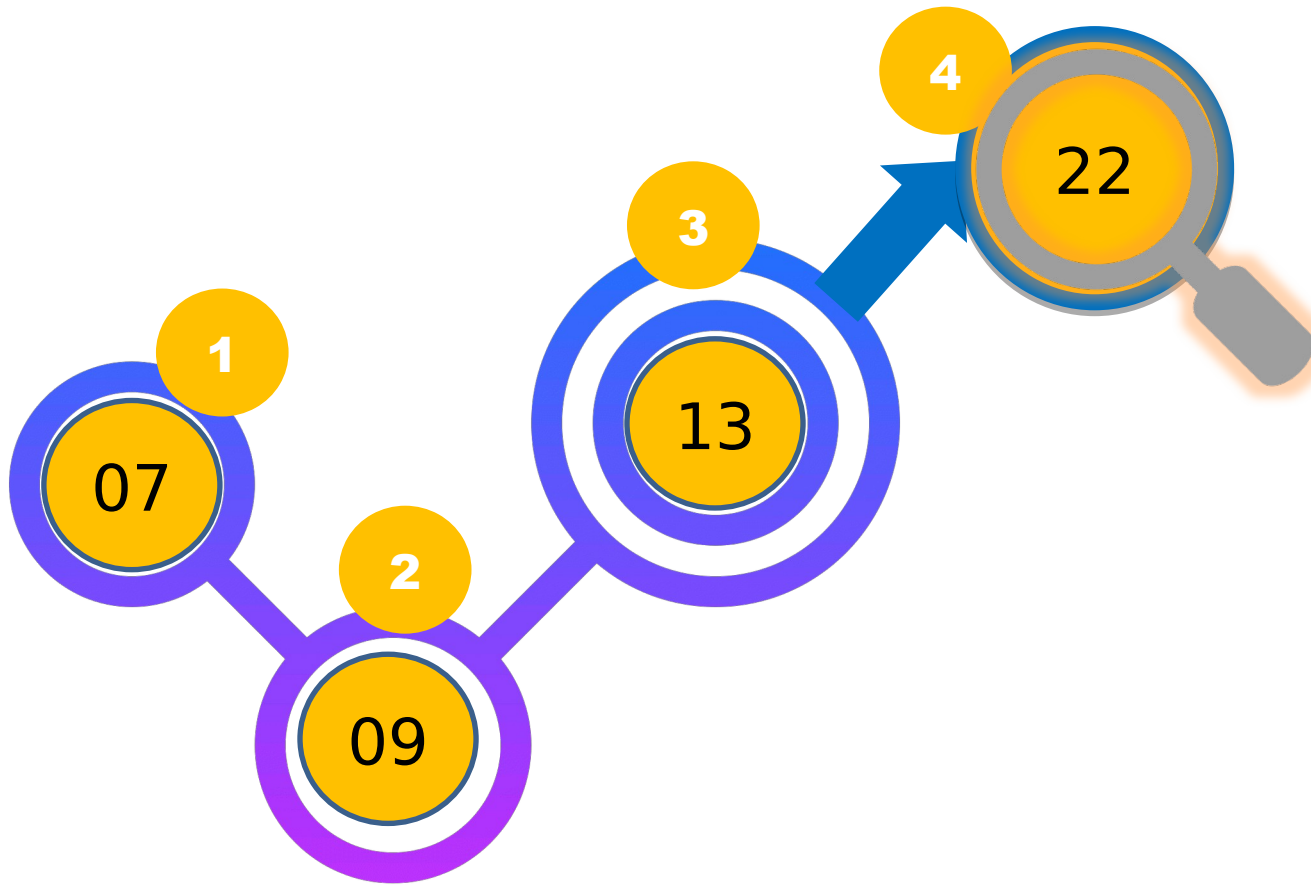
Pencarian dilakukan dari data pertama, lalu dibaca satu persatu hingga data yang dicari ditemukan atau sampai data terakhir.



Mencari sebuah nama dalam daftar nama yang tidak diurutkan menurut abjad nama orang

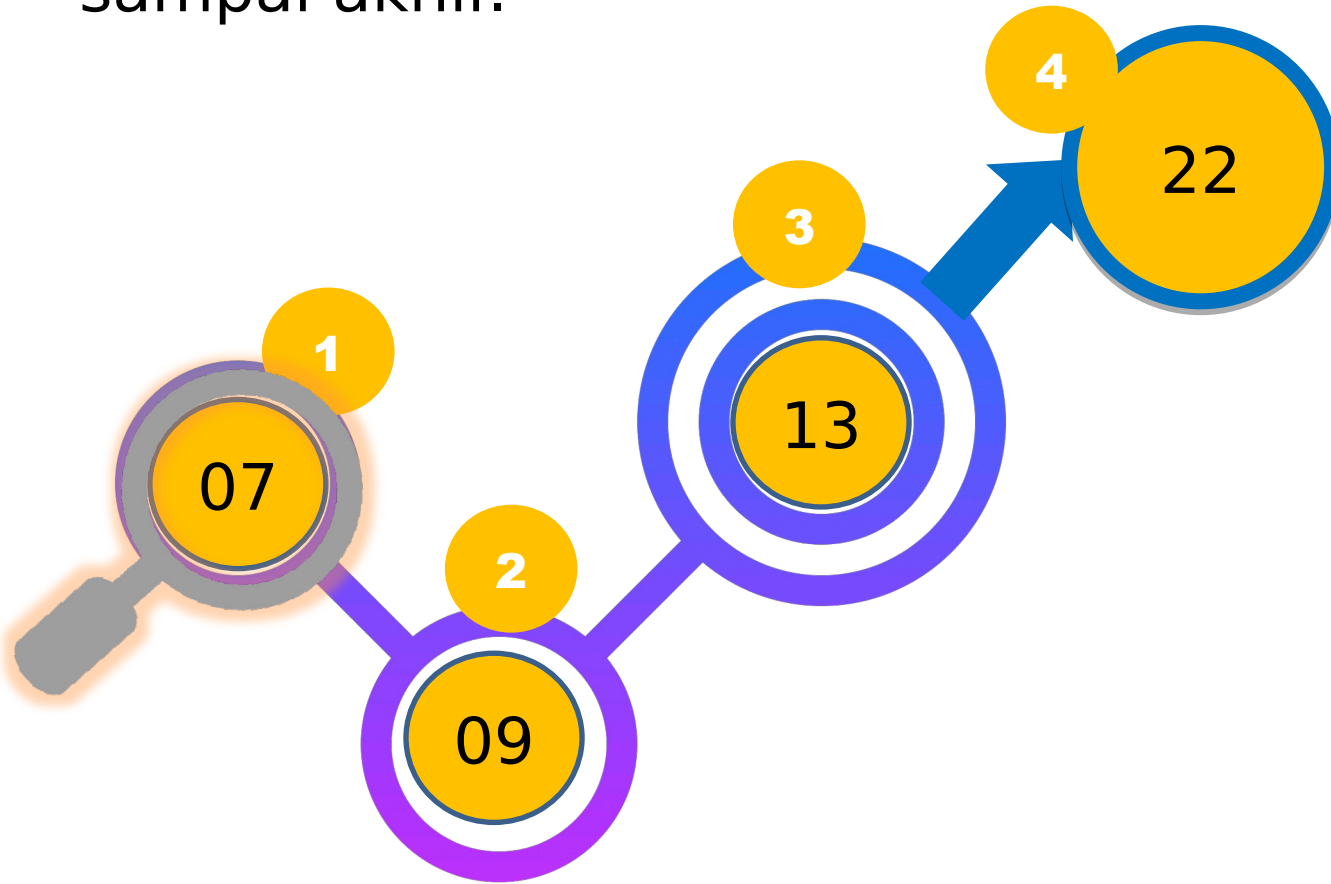


Θ Pencarian akan dilakukan dari awal daftar dan kemudian dibaca satu persatu hingga data yang di cari ditemukan.



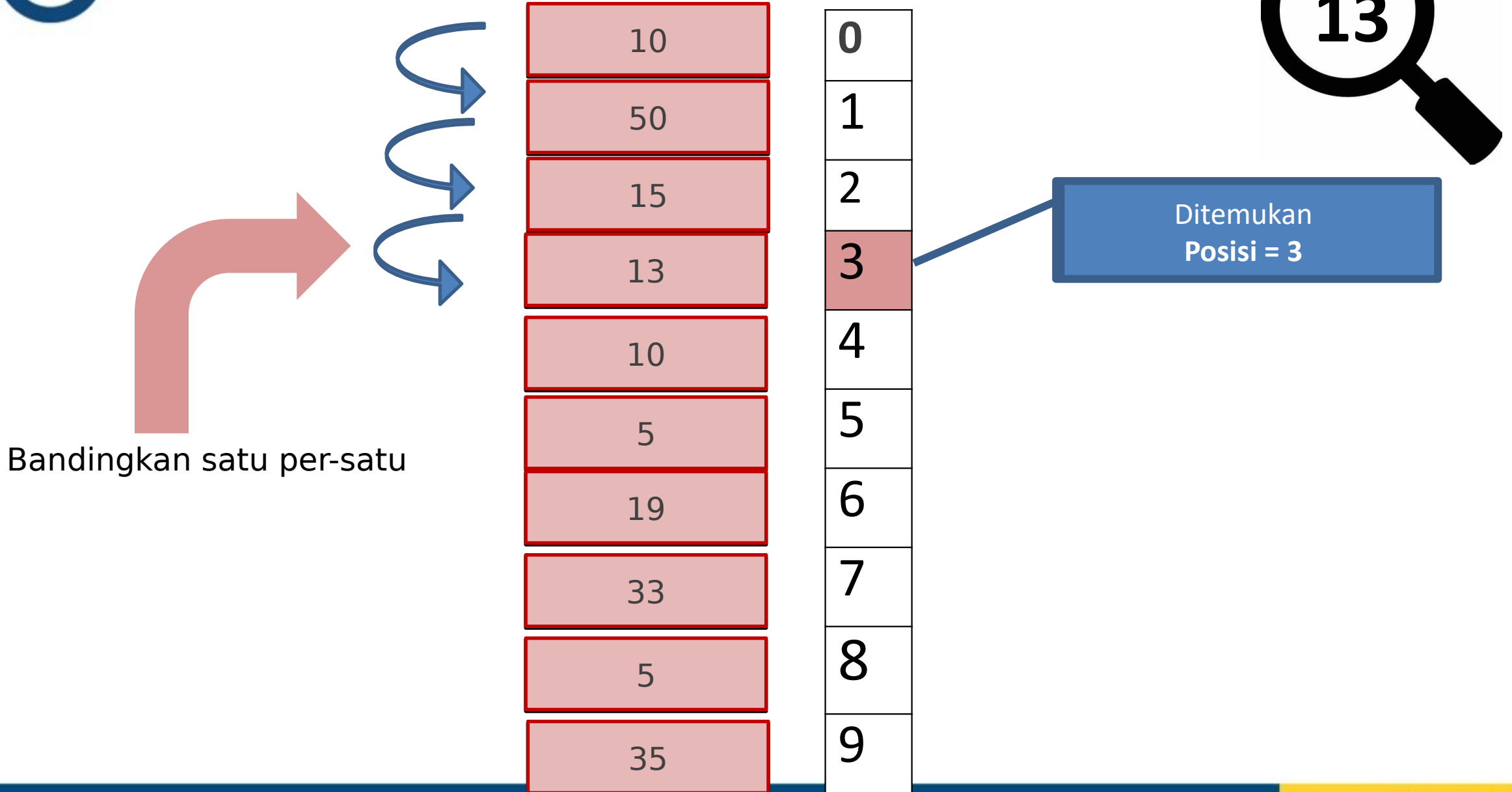


Θ **Linear search** adalah pencarian yang dilakukan secara berurutan dari awal data hingga data tersebut di temukan atau dilakukan sampai akhir.






1. Pencarian Linier (Linear Search)





1. Pencarian Linier (*Linear Search*)

Contoh mencari data **bernilai 25**:



<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]
<i>Data</i>	8	65	21	94	25	19	37

Cara kerja pencarian linier :

- 1) Membandingkan data bernilai 25 dengan data pertama, yaitu data index [0] yang bernilai 8 → data tidak sama, lanjutkan ke data berikutnya.
- 2) Membandingkan data bernilai 25 dengan data index [1] yang bernilai 65 → data tidak sama, lanjutkan ke data berikutnya sampai data ditemukan atau hingga data terakhir telah dibandingkan.
- 3) Pada posisi index [4] data bernilai 25 → data sama, proses pencarian selesai.

PSEUDOCODE Linear Search

Pada awal program akan dilakukan inisialisasi indek pertama pada array

Jika data dalam array tersebut sama dengan data Yang dicari, maka pencarian akan berhenti dan akan mengembalikan nilai sesuai yang diminta.

Jika data tidak sama, maka indeks akan terus bertambah hingga data di temukan atau array sudah habis

```
Vertikal: ar[L, N, X]:
```

```
Indek ← -1
```

```
Indek ←
```

```
While Indek < N
```

```
    If ar[Indek] = X
```

```
        Return Indek
```

```
    Indek ← Indek + 1
```

```
Return -1
```

```
End While
```

```
Return Indek
```



```
CariLinear(L,N,X) :
```

```
Posisi ← -1
```

```
J ← 0
```

```
WHILE J < N
```

```
  IF L[J] = X
```

```
    Posisi ← J
```

```
    Keluar dari WHILE
```

```
  END-IF
```

```
  J++
```

```
END-WHILE
```

```
RETURN Posisi
```

Dianggap data belum di temukan

Indek pada array nilai 0 | Nilai awal pencarian

N adalah jumlah total dari elemen array (kondisi per

Data telah ditemukan

Jika IF tidak ditemukan, maka proses akan di hentikan pada saat index/array sudah dibandingkan semua

Kondisi pencarian atau pembadingan nilai yg akan dicari

While akan berhenti jika nilai J tidak lebih kecil dari N



1. Pencarian Linier (*Linear Search*)

Berdasarkan teknik pencarian linier, terlihat bahwa kompleksitasnya berupa $O(n)$, yaitu :

- ✓ $O(1)$ → kondisi dalam keadaan terbaik (***best case***) apabila data yang dicari langsung dapat ditemukan pada posisi pertama atau index [0].
- ✓ $O(n)$ → kondisi dalam keadaan terburuk (***worst case***) apabila data yang dicari berada pada posisi terakhir.



2. Pencarian BINER (*BINARY Search*)



2. Pencarian BINER (*BINARY Search*)

Pencarian Biner (*Binary Search*): merupakan teknik yang efisien untuk mencari pada data yang telah berurutan.

Jika data berurutan, maka pencarian menggunakan ***linear Search*** tidaklah efisien



linear Search : mencari data satu persatu dari awal hingga akhir



2. Pencarian BINER (*BINARY Search*)

Pencarian Biner (*Binary Search*): merupakan teknik yang efisien untuk mencari pada data yang telah berurutan.

Jika data berurutan, maka pencarian menggunakan ***linear Search*** tidaklah efisien



Misal, ketika mencari kata diksi dalam KBBI, tentu tidak memulai dari halaman pertama yang memuat abjad “A” untuk pencariannya



2. Pencarian BINER (*BINARY Search*)

Misalnya, ketika kita mencari arti kata **“*River*”** pada buku kamus bahasa Inggris, maka kita tidak perlu memulai pencarian dari awal huruf “A”, karena isi kamusnya telah berurutan. Kita dapat langsung menuju pada halaman yang diawali dengan huruf “R”.





Binary Search



Binary Search adalah sebuah model pencarian data atau elemen dalam sebuah array dengan syarat Kondisi data dalam keadaan yang sudah berurutan



```
1 CariBiner(L,N,X):
2 //L adalah array dengan N elemen
3 //X data yang dicari
4 Bawah ← 0
5 Atas ← N-1
6 Posisi ← -1 //Berarti data tidak ditemukan
7
8 WHILE Atas ≥ Bawah
9     Tengah ← (Atas + Bawah)/2
10    IF X > L [Tengah]
11        //Kemungkinan data ada pada bagian kanan
12        Bawah ← Tengah +1
13
14    ELSE
15        IF X < L [Tengah]
16            //Kemungkinan data ada di bagian kiri
17            Atas ← Tengah -1
18
19        ELSE
20            //Berarti data telah ditemukan
21            Posisi ← Tengah
22            Bawah ← Atas + 1 //Untuk mengakhiri WHILE
23        END-IF
24    END-IF
25 END-WHILE
26 CariLinear ← Posisi
```

Berikut adalah
algoritme dalam
binary search.



2. Pencarian BINER (*BINARY Search*)

	B (batas bawah)				T (batas tengah)					A (batas atas)
	↓				↓					↓
Index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
Data	10	20	30	40	50	60	70	80	90	100

Langkah-langkah pencarian biner (*binary search algorithm*):

- Mencari index dalam array yang terletak di tengah-tengah dari batas bawah hingga batas atas: **$T = ([B] + [A]) / 2$** .
- Contoh: $T = (0+9)/2 = 4,5 \rightarrow T = 4$
- Bandingkan data yang dicari dengan data yang berada di tengah tersebut.



2. Pencarian BINER (*BINARY Search*)

	B ↓				T ↓					A ↓
<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<i>Data</i>	10	20	30	40	50	60	70	80	90	100

Dalam melakukan perbandingan data, akan terdapat 3 kemungkinan sbb:

1. Jika kedua data yang dibandingkan bernilai sama, maka data yang dicari telah ditemukan.
2. Jika data yang dicari bernilai lebih kecil dari elemen yang berada di tengah array, maka pencarian dilakukan kembali pada data yang berada di sisi bagian kiri dari elemen tengah.
3. Jika data yang dicari bernilai lebih besar dari elemen yang berada di tengah array, maka pencarian dilakukan kembali pada data yang berada di sisi bagian kanan dari elemen tengah.



2. Pencarian BINER (*BINARY Search*)

Contoh mencari data **bernilai 17**:

	B (batas bawah)		T (batas tengah)				A (batas atas)			
	↓		↓				↓			
<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<i>Data</i>	10	11	12	13	14	16	17	18	19	20

Cara kerja pencarian biner:

- 1) Bagilah jumlah data menjadi 2 bagian ($n/2$) $\rightarrow (10 / 2) = 5$, dalam array data ke lima adalah index[4], maka T[4] adalah batas tengah, B[0] adalah batas bawah dan A[9] adalah batas atas.
- 2) Bandingkan nilai 17 dengan nilai T[4] $\rightarrow 17 \neq 14$ dan $17 > 14 \rightarrow$ lanjutkan pencarian kembali dengan menggeser B dan T ke arah kanan.
- 3) $B = T[4] + 1 \rightarrow B[5]$ dan $T = (B + A) / 2 = ([5] + [9]) / 2 \rightarrow T[7]$.



2. Pencarian BINER (*BINARY Search*)

Contoh mencari data **bernilai 17**:

<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<i>Data</i>	10	11	12	13	14	16	17	18	19	20

Diagram illustrating the binary search process for finding the value 17. The array is divided into two halves by a dashed line. The left half (indices [0] to [4]) is shaded gray. The right half (indices [5] to [9]) is shaded light orange. The value 17 is highlighted in cyan at index [6]. The value 18 is highlighted in yellow at index [7]. Arrows indicate the search path: a red dashed arrow from index [2] to index [5] (labeled B), a blue dashed arrow from index [4] to index [7] (labeled T), and a blue dashed arrow from index [9] to index [9] (labeled A).

- 4) Data array yang digunakan saat ini, index[5] s.d. index[9] → B[5], T[7] dan A[9].
- 5) Bandingkan nilai 17 dengan nilai T[7] → $17 \neq 18$ dan $17 < 18$ → lanjutkan pencarian kembali dengan menggeser A dan T ke arah kiri.
- 6) $A = T[7] - 1 \rightarrow A[6]$ dan $T = (B + A) / 2 = ([5] + [6]) / 2 \rightarrow T[5]$



2. Pencarian BINER (*BINARY Search*)

Contoh mencari data **bernilai 17**:

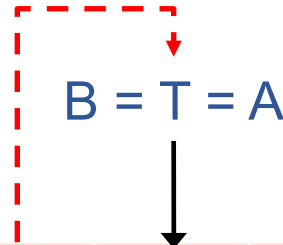
<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<i>Data</i>	10	11	12	13	14	16	17	18	19	20

- 7) Data array yang digunakan saat ini, index[5] s.d. index[6] → B[5], T[5] dan A[6].
- 8) Bandingkan nilai 17 dengan nilai T[5] → $17 \neq 16$ dan $17 > 16$ → lanjutkan pencarian kembali dengan menggeser B ke arah kanan.



2. Pencarian BINER (*BINARY Search*)

Contoh mencari data bernilai **17**:



<i>Index</i>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<i>Data</i>	10	11	12	13	14	16	17	18	19	20

9) Karena data yang tersisa hanya index[6] $\rightarrow B=T=A=6$.

10) Bandingkan nilai 17 dengan nilai $T[6] \rightarrow 17 = 17 \rightarrow$ data sama, $T[6]$ adalah lokasi dari data yang dicari.



2. Pencarian BINER (*BINARY Search*)

Binary search selalu mencari data pada setengah jumlah data.

- ✓ Pada keadaan terburuk (*worst case*) akan terjadi **$\text{Log}_2 (n)$** perbandingan.
- ✓ Kompleksitas dalam waktu *average case* akan sama dengan kondisi *worst case*.



SELESAI