

Python Programming

Array Searching



01

Pencarian (Searching)

Pendahuluan Searching

- Pencarian array adalah proses penting dalam pengolahan data di dunia pemrograman.
- Saat bekerja dengan kumpulan data, sering kali kita perlu menentukan apakah suatu nilai tertentu ada dalam array atau tidak.
- Proses ini dikenal sebagai "searching" dan dapat dilakukan dengan berbagai metode tergantung pada sifat data dan kebutuhan aplikasi.
- Dalam konteks Python, array dapat direpresentasikan menggunakan struktur data seperti list atau array.
- Setiap elemen dalam array memiliki indeks yang mempermudah pengaksesan nilai secara spesifik.
- Pencarian array menjadi kunci ketika kita ingin mengetahui keberadaan atau lokasi suatu nilai dalam kumpulan data tersebut.

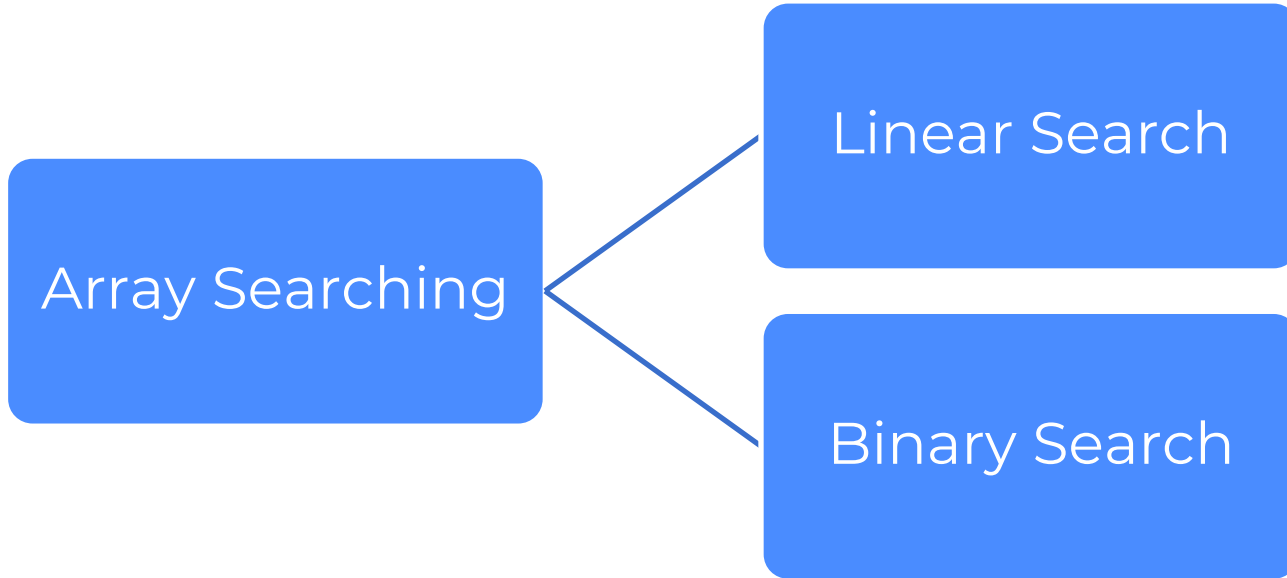
Array Sorting vs Array Searching (1/2)

	Array Sorting	Array Searching
Definisi	Proses menyusun elemen-elemen array dalam urutan tertentu, seperti dari terkecil hingga terbesar atau sebaliknya.	Proses mencari nilai tertentu di dalam array, untuk menentukan apakah nilai tersebut ada dalam array dan dimana letaknya.
Tujuan	Membantu membuat data lebih terstruktur, memudahkan pemrosesan data, dan memungkinkan penerapan metode pencarian yang efisien.	Menemukan keberadaan dan lokasi suatu nilai dalam array, memberikan informasi tentang keberadaan atau ketiadaan elemen tertentu.
Proses	Melibatkan pengaturan ulang elemen-elemen dalam array berdasarkan aturan tertentu, seperti nilai numerik atau urutan alfabet.	Melibatkan pengecekan setiap elemen dalam array atau penerapan algoritma yang lebih canggih untuk menemukan nilai yang dicari.

Array Sorting vs Array Searching (2/2)

	Array Sorting	Array Searching
Waktu Eksekusi	Memiliki kompleksitas waktu yang lebih tinggi, terutama untuk algoritma pengurutan yang lebih canggih seperti merge sort atau quicksort.	Bergantung pada metode pencarian yang digunakan.
Urgensi	Penting untuk memfasilitasi operasi pencarian yang efisien dan meningkatkan keterbacaan data.	Penting untuk menemukan informasi spesifik tentang keberadaan atau ketiadaan suatu nilai dalam array.
Contoh Penggunaan	Mengurutkan nilai mahasiswa dari yang tertinggi ke terendah.	Mencari apakah suatu nilai tertentu ada dalam daftar nilai mahasiswa.

Array Sorting vs Array Searching (2/2)



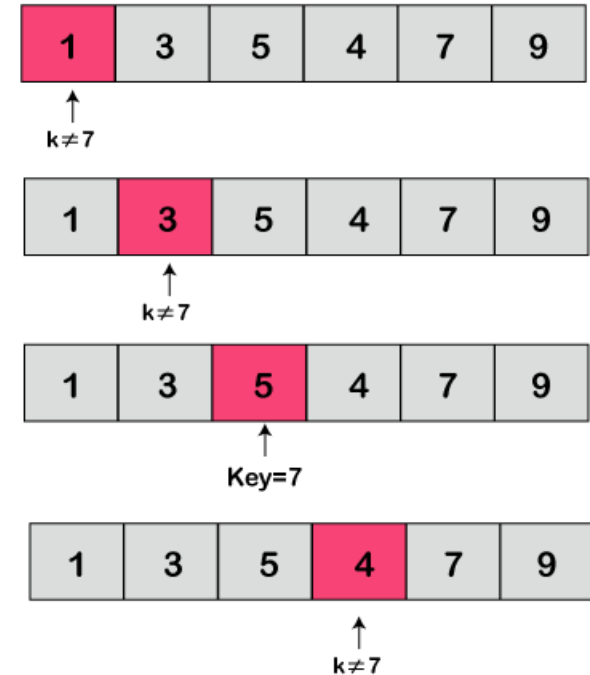


02

Algoritma Searching: Linear Search

Linear Search

- Linear search dikenal sebagai Sequential Search (Pencarian Berurutan).
- Merupakan metode atau teknik yang digunakan untuk menemukan elemen dalam array atau list
- Linear Search dikenal sebagai algoritma pencarian yang paling sederhana dan termudah dari semua algoritma pencarian lainnya.
- Prinsipnya adalah membandingkan setiap elemen dengan nilai yang kita cari.
- Jika keduanya cocok, maka elemen ditemukan, dan algoritma akan mengembalikan posisi key index.
- Kekurangan dari algoritma ini adalah apabila data yang di cari letaknya pada elemen terakhir, maka pencarian lebih memakan waktu yang cukup lama pula. Karena Linear Search mengunjungi setiap elemen data yang ada.



Alur Linear Search

- **Langkah 1:** Mencari elemen secara berurutan dengan membandingkan elemen yang diinginkan dengan setiap elemen yang ada dalam array tertentu.
- **Langkah 2:** Jika elemen yang diinginkan ditemukan, maka index atau posisi elemen tersebut akan ditampilkan kepada user.
- **Langkah 3:** Jika elemen tidak ada dalam array, maka user akan diberitahu bahwa elemen tersebut tidak ditemukan.

Contoh Code Linear Search (1/3)

```
1 # Linear Search Function
2 def linearSearch(array, cariNilai):
3     for i in range(len(array)):
4         if array[i] == cariNilai:
5             return i # index elemen yang ditemukan
6     return -1 # jika elemen tidak ditemukan
7
8 # Contoh penggunaan
9 array = [0, 1, 2, 3, 4, 5]
10 nilai = 3 # nilai yang dicari
11 result = linearSearch(array, nilai)
12
13 if (result != -1):
14     print(f"Nilai {nilai} ditemukan pada index ke-{result}.")
15 else:
16     print(f"Nilai {nilai} tidak ditemukan dalam array.")
```

✓ 0.0s

Nilai 3 ditemukan pada index ke-3.

Contoh Code Linear Search (2/3)

```
1 # Linear Search Function
2 def linearSearch(array, cariNilai):
3     for i in range(len(array)):
4         if array[i] == cariNilai:
5             return i # index elemen yang ditemukan
6     return -1 # jika elemen tidak ditemukan
7
8 # Contoh penggunaan
9 array = [0, 1, 2, 3, 4, 5]
10 nilai = 10 # nilai yang dicari
11 result = linearSearch(array, nilai)
12
13 if (result != -1):
14     print(f"Nilai {nilai} ditemukan pada index ke-{result}.")
15 else:
16     print(f"Nilai {nilai} tidak ditemukan dalam array.")
```

✓ 0.0s

Nilai 10 tidak ditemukan dalam array.

Contoh Code Linear Search (3/3)

```
1 # Linear Search Function
2 def linearSearch(array, cariNilai):
3     for i in range(len(array)):
4         if array[i] == cariNilai:
5             return i # index elemen yang ditemukan
6     return -1 # jika elemen tidak ditemukan
7
8 # Contoh penggunaan
9 unsortedArray = [5, 3, 0, 4, 1, 2]
10 nilai = 4 # nilai yang dicari
11 result = linearSearch(unsortedArray, nilai)
12
13 if (result != -1):
14     print(f"Nilai {nilai} ditemukan pada index ke-{result}.")
15 else:
16     print(f"Nilai {nilai} tidak ditemukan dalam array.")
```

✓ 0.0s

Nilai 4 ditemukan pada index ke-3.

Penjelasan Code Linear Search (1/2)

```
1 # Linear Search Function
2 def linearSearch(array, cariNilai):
3     for i in range(len(array)):
4         if array[i] == cariNilai:
5             return i # index elemen yang ditemukan
6     return -1 # jika elemen tidak ditemukan
```

- **linearSearch:** function untuk melakukan pencarian linear. Fungsi ini menerima dua parameter: **array** (array yang akan dicari) dan **cariNilai** (nilai yang dicari).
- Function menggunakan looping **for** untuk iterasi melalui setiap elemen dalam array.
- Pada setiap iterasi, function memeriksa apakah **array[i]** (elemen saat ini) sama dengan **cariNilai** (nilai yang dicari).
- Jika nilai ditemukan, function mengembalikan index elemen tersebut dengan menggunakan pernyataan **return i**.
- Jika seluruh array telah diperiksa namun nilai tidak ditemukan, function mengembalikan **-1** untuk menandakan bahwa nilai tidak ada dalam array.

Penjelasan Code Linear Search (1/2)

```
8 # Contoh penggunaan
9 array = [0, 1, 2, 3, 4, 5]
10 nilai = 3 # nilai yang dicari
11 result = linearSearch(array, nilai)
12
13 if (result != -1):
14     print(f"Nilai {nilai} ditemukan pada index ke-{result}.")
15 else:
16     print(f"Nilai {nilai} tidak ditemukan dalam array.")
```

- Contoh penggunaan fungsi linearSearch diberikan dengan **array = [0, 1, 2, 3, 4, 5]** dan **nilai yang dicari adalah 3**.
- Hasil pencarian disimpan dalam variabel **result**.
- Pada bagian kondisional **if (result)**, program akan memeriksa apakah nilai yang dicari ditemukan.
- Jika iya (**result != -1**), print pesan bahwa "Nilai ditemukan pada index ke-n".
- Sedangkan jika tidak (**result == -1**), print pesan bahwa "Nilai tidak ditemukan dalam array".




02

Algoritma Searching: Binary Search

Binary Search

- Algoritma Binary Search sangat berbeda dengan algoritma Linear Search.
- Algoritma ini mengikuti prosedur yang sangat berbeda untuk mencari elemen dari array karena array harus diurutkan secara umum terlebih dahulu.
- Jika array tersebut belum terurut (unsorted array), maka array tersebut harus diurutkan terlebih dahulu, lalu kemudian pencarian menggunakan algoritma Binary Search akan dimulai.
- Binary search merupakan algoritma searching yang relatif cepat dan efisien walaupun ada banyak data sekalipun. Karena data dicari dari depan, tengah dan belakang. Tetapi algoritma dan code-nya sedikit lebih rumit.

Binary Search 

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	9
23 > 16 take 2 nd half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 < 56 take 1 st half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

Alur Binary Search

- **Langkah 1:** Proses array sorting adalah langkah pertama yang dilakukan.
- **Langkah 2:** Setelah array diurutkan, array dianggap sebagai dua bagian. Setengahnya dimulai dari elemen pertama hingga elemen tengah dari array yang diurutkan (elemen paruh pertama) dan setengah lainnya dimulai dari elemen setelah elemen tengah hingga elemen terakhir dari array yang diurutkan (elemen paruh kedua).
- **Langkah 3:** Key elemen (nilai yang dicari) dibandingkan dengan elemen tengah dari array yang diurutkan.
- **Langkah 4:** Jika key elemen kurang dari atau sama dengan elemen tengah dari array yang diurutkan, elemen paruh kedua akan diabaikan lebih lanjut karena key elemen lebih kecil dari elemen tengah. Jadi yang pasti elemen tersebut harus ada di antara elemen pertama atau elemen tengah.
- **Langkah 5:** Jika key elemen lebih besar dari elemen tengah, maka elemen paruh pertama array yang diurutkan diabaikan dan elemen dari elemen tengah hingga elemen terakhir akan dipertimbangkan.
- **Langkah 6:** Dari elemen tersebut, key elemen dibandingkan lagi dengan elemen tengah dari array yang dibelah dua dan mengulangi prosedur yang sama. Jika key elemen lebih besar dari elemen tengah array yang dibelah dua, maka paruh pertama diabaikan.
- **Langkah 7:** Jika key elemen kurang dari atau sama dengan elemen tengah dari array yang dibelah dua, elemen paruh kedua dari array yang dibelah dua akan diabaikan. Dengan cara ini, elemen-elemen dicari di bagian mana pun dari array yang sesuai.

Contoh Code Binary Search (1/3)

```
1 def binarySearch(array, cariNilai):
2     indexLow, indexHigh = 0, len(array) - 1 # menentukan index pertama (selalu 0) dan index terakhirnya (n - 1)
3
4     while indexLow <= indexHigh:
5         middle = (indexLow + indexHigh) // 2 # untuk membagi 2 elemen array
6         middleValue = array[middle]
7
8         if middleValue == cariNilai:
9             return middle # indeks elemen yang ditemukan
10        elif middleValue < cariNilai:
11            indexLow = middle + 1
12        else:
13            indexHigh = middle - 1
14    return -1 # jika elemen tidak ditemukan
15
16 # Contoh penggunaan
17 sortedArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # array harus sudah terurut
18 cari = 4
19 result = binarySearch(sortedArray, cari)
20
21 if (result != -1):
22     print(f"Nilai {cari} ditemukan pada index ke-{result}.")
23 else:
24     print(f"Nilai {cari} tidak ditemukan dalam array.")
✓ 0.0s
```

Nilai 4 ditemukan pada index ke-4.

Contoh Code Binary Search (2/3)

```
1 def binarySearch(array, cariNilai):
2     indexLow, indexHigh = 0, len(array) - 1 # menentukan index pertama (selalu 0) dan index terakhirnya (n - 1)
3
4     while indexLow <= indexHigh:
5         middle = (indexLow + indexHigh) // 2 # untuk membagi 2 elemen array
6         middleValue = array[middle]
7
8         if middleValue == cariNilai:
9             return middle # indeks elemen yang ditemukan
10        elif middleValue < cariNilai:
11            indexLow = middle + 1
12        else:
13            indexHigh = middle - 1
14    return -1 # jika elemen tidak ditemukan
15
16 # Contoh penggunaan
17 sortedArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # array harus sudah terurut
18 cari = 10
19 result = binarySearch(sortedArray, cari)
20
21 if (result != -1):
22     print(f"Nilai {cari} ditemukan pada index ke-{result}.")
23 else:
24     print(f"Nilai {cari} tidak ditemukan dalam array.")
✓ 0.0s
```

Nilai 10 tidak ditemukan dalam array.

Contoh Code Binary Search (3/3)

```
1 def binarySearch(array, cariNilai):
2     indexLow, indexHigh = 0, len(array) - 1 # menentukan index pertama (selalu 0) dan index terakhirnya (n - 1)
3
4     while indexLow <= indexHigh:
5         middle = (indexLow + indexHigh) // 2 # untuk membagi 2 elemen array
6         middleValue = array[middle]
7
8         if middleValue == cariNilai:
9             return middle # indeks elemen yang ditemukan
10        elif middleValue < cariNilai:
11            indexLow = middle + 1
12        else:
13            indexHigh = middle - 1
14    return -1 # jika elemen tidak ditemukan
15
16 # Contoh penggunaan
17 unsortedArray = [3, 6, 1, 9, 2, 7, 5, 8, 4] # jika array belum terurut
18 cari = 3
19 result = binarySearch(unsortedArray, cari)
20
21 if (result != -1):
22     print(f"Nilai {cari} ditemukan pada index ke-{result}.")
23 else:
24     print(f"Nilai {cari} tidak ditemukan dalam array.")
✓ 0.0s
```

Nilai 3 tidak ditemukan dalam array.

Penjelasan Code Binary Search (1/2)

```
1 def binarySearch(array, cariNilai):
2     indexLow, indexHigh = 0, len(array) - 1 # menentukan index pertama (selalu 0) dan index terakhirnya (n - 1)
3
4     while indexLow <= indexHigh:
5         middle = (indexLow + indexHigh) // 2 # untuk membagi 2 elemen array
6         middleValue = array[middle]
7
8         if middleValue == cariNilai:
9             return middle # indeks elemen yang ditemukan
10        elif middleValue < cariNilai:
11            indexLow = middle + 1
12        else:
13            indexHigh = middle - 1
14    return -1 # jika elemen tidak ditemukan
```

- **binarySearch**: function untuk melakukan pencarian biner. Function ini menerima dua parameter: **array** (array yang sudah terurut) dan **cariNilai** (nilai yang dicari).
- Dua indeks, **indexLow** dan **indexHigh**, diinisialisasi untuk menunjukkan batas awal dan akhir dari array yang akan dicari.
- Looping **while** digunakan untuk melakukan binary searching. Selama **indexLow kurang dari atau sama dengan indexHigh**, pencarian akan terus dilakukan.
- Di setiap iterasi, **middle** (nilai tengah) dari array dihitung, dan nilai tengah tersebut dibandingkan dengan **cariNilai** (nilai yang dicari).
- Jika nilai yang dicari ditemukan, function mengembalikan indeks elemen tersebut.
- Jika nilai yang dicari lebih besar dari nilai tengah, **indexLow** diatur menjadi **middle + 1** untuk mencari di setengah kanan array. Jika lebih kecil, **indexHigh** diatur menjadi **middle - 1** untuk mencari di setengah kiri array.

Penjelasan Code Binary Search (2/2)

```
16 # Contoh penggunaan
17 sortedArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # array harus sudah terurut
18 cari = 4
19 result = binarySearch(sortedArray, cari)
20
21 if (result != -1):
22     print(f"Nilai {cari} ditemukan pada index ke-{result}.")
23 else:
24     print(f"Nilai {cari} tidak ditemukan dalam array.")
```

- Dalam contoh ini, kita menggunakan array yang sudah terurut yaitu **sortedArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]** dan **mencari nilai 4** menggunakan function `binarySearch`.
- Hasil searching disimpan dalam variabel **result**.
- Pada bagian kondisional **if (result)**, program akan memeriksa apakah nilai yang dicari ditemukan.
- Jika iya (**result != -1**), print pesan bahwa "Nilai ditemukan pada index ke-n".
- Namun, jika tidak (**result == -1**), print pesan bahwa "Nilai tidak ditemukan dalam array".



05

Studi Kasus

Studi Kasus

1. Sebuah toko memiliki stok barang yang tersedia. Buatlah program Python untuk mencari apakah suatu barang tersedia dalam stok toko.. (Buatlah array dummy dimana jumlah elemen ada 15 barang)
2. Universitas Pendidikan Indonesia memiliki daftar mahasiswa yang terdaftar di prodi RPL. Buatlah program Python untuk mencari apakah nama seorang mahasiswa tertentu ada dalam daftar tersebut? (Buat nama tersebut berdasarkan inputan dari user dan buat array dummy 20 elemen untuk daftar nama mahasiswanya)
3. Dari algoritma Linear Search dan Binary Search yang telah Anda pelajari, buatlah perbandingan running program (execution time) dengan 2 algoritma tersebut. Buatlah dengan menggunakan sortedArray berikut:

```
array = [1, 2, 5, 7, 8, 10, 16, 18, 19, 23, 24, 26, 28, 29, 32, 33, 34, 35, 36, 38, 40, 41, 42, 43, 44, 46, 48, 49, 51, 55, 57,  
58, 59, 60, 63, 65, 66, 69, 74, 75, 76, 77, 78, 79, 81, 82, 85, 90, 93, 100]
```

Tentukan manakah dari ke 2 algoritma tersebut yang paling cepat untuk mencari nilai 60?

Terimakasih

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution