## Pengantar Kumpulan Data

[](https://www.dicoding.com/academies/620/tutorials/35088?from=35083)

Dalam materi sebelumnya, Anda sudah memahami konsep dasar fungsi dan prosedur. Mulai dari mendefinisikan fungsi dan prosedur, mengenal parameter dan argumen, membandingkan antara fungsi dan prosedur, serta pembuatan fungsi dan prosedur pada bahasa pemrograman Python.

Kali ini, kita akan membahas mengenai kumpulan data secara mendalam. Anda perlu mempelajarinya secara komprehensif walaupun sudah dipelajari pada materi Tipe Data. Untuk itu, Anda akan belajar beberapa materi mengenai:

* menggambarkan konsep kumpulan data secara detail;
* mengidentifikasi kumpulan data yang bersarang;
* menguraikan proses pencarian data dalam kumpulan data; serta
* menyortir data dalam kumpulan data.

Apakah Anda mulai menantikan cerita dari anggota keluarga Bapak Fajar? Sabar. Kali ini kita akan mengamati aktivitas Bima terkait dengan kumpulan data. Kegiatan Bima akan membawa perspektif baru terhadap pemahaman kita mengenai kumpulan data. Sudah siap mengamati perjalanan Bima? Yuk, simak ceritanya.

Saat pertama kali masuk ke jenjang SMA, Bima mendapati teman satu kelas dengan jumlah 30 anak, termasuk dia. Mereka mulai menyapa dan berkenalan satu demi satu untuk mengenal lebih dekat antar teman.

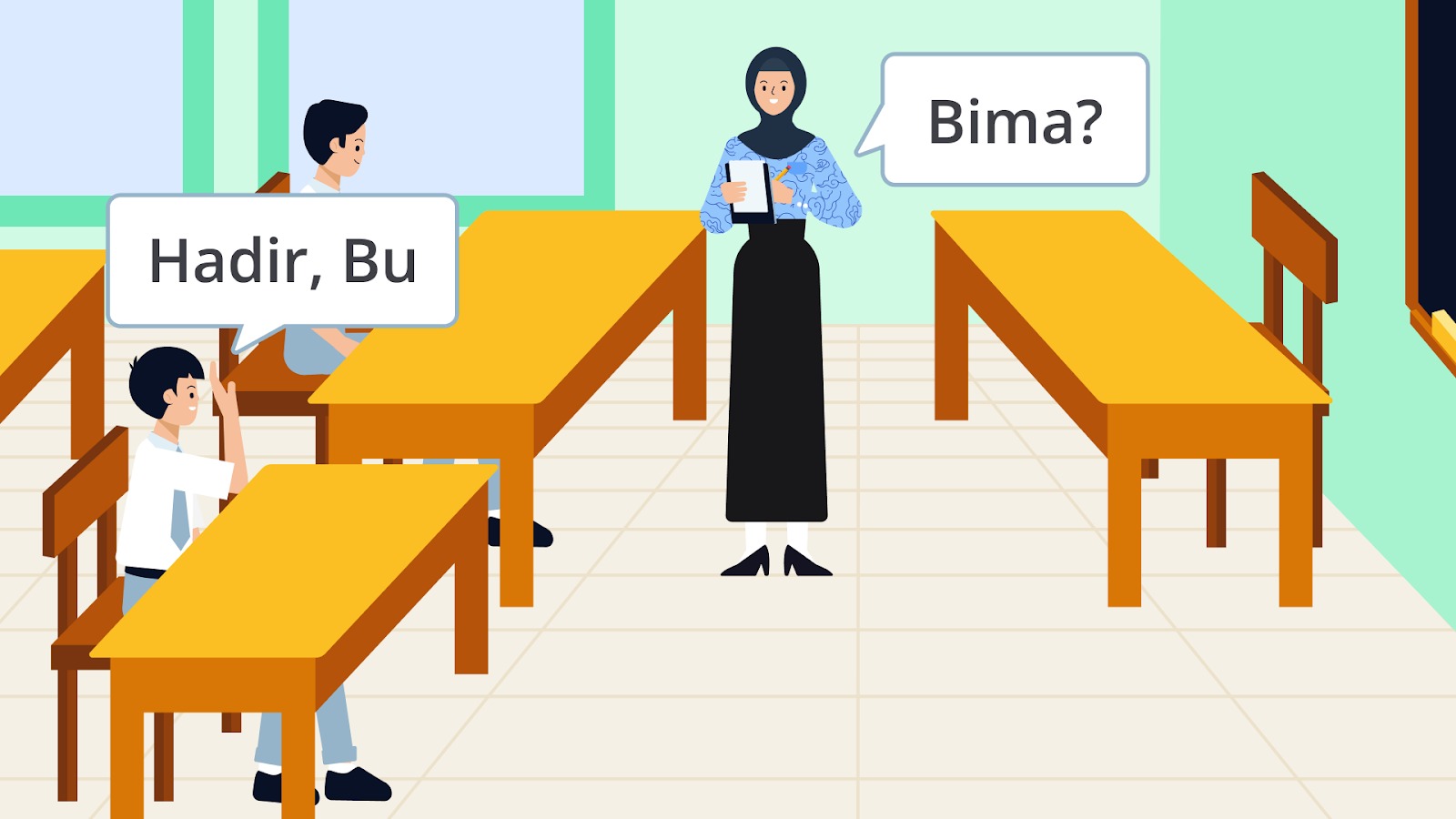
Setelah diberikan waktu untuk berkenalan, Bima menyadari bahwa mereka memiliki keberagaman latar belakang, hobi, dan minat secara kepribadian. Ada yang hobi bermain sepak bola, tertarik dengan pelajaran matematika, bahkan ada teman yang berasal dari luar negeri.

Secara administrasi, setiap siswa memiliki identitas yang membedakan satu dengan lainnya, seperti nama, nomor induk siswa, tanggal dan tempat lahir, serta jenis kelamin. Mereka mempunyai karakteristik berbeda dan berkumpul dalam satu lingkungan yang sama.

[](https://www.dicoding.com/academies/620/tutorials/35088?from=35083)

Keberagaman karakter dan identitas teman Bima yang ada di kelas adalah representasi dari kumpulan data. Layaknya pemrograman, setiap orang yang ada dalam kelas dapat dianggap sebagai elemen pada kumpulan data. Ia menyimpan informasi unik, seperti nama, hobi, minat, latar belakang, dan masih banyak lainnya. Jika mereka dikumpulkan dalam satu kelas, kita dapat menyebutkan kumpulan data siswa.

Ada kondisi ketika guru ingin memanggil salah satu dari murid kelas. Beliau dapat memanggil dengan cara menyebutkan informasi yang terikat dengan murid, seperti nama atau nomor induk siswa. Selain itu, ada kondisi ketika guru menyuruh murid untuk duduk sesuai nomor induknya, seperti saat ujian akhir. Ada banyak hal yang bisa dilakukan oleh guru dengan mengolah informasi melalui atribut murid.

[](https://www.dicoding.com/academies/620/tutorials/35088?from=35083)

Bermacam-macam kondisi tersebut adalah salah satu aksi yang dapat dicapai ketika kita memiliki informasi dengan jumlah yang banyak. Kita dapat melakukan pengurutan, pemanggilan, penambahan, pengurangan, pengelompokan, dan banyak hal lainnya yang terkait dengan informasi.

Dalam pemrograman pun memiliki konsep yang serupa. Pada sebuah kumpulan data, Anda dapat menjalankan berbagai macam aksi, seperti pengelompokan, pengurutan, pemanggilan, dan lainnya. Setiap aksi memiliki cara dan alur program yang beragam walaupun tujuannya sama. Dengan begitu, kita dapat menyelesaikan permasalahan dengan aksi yang cepat dan tepat.

Sudah terbayang apa yang akan dibahas pada materi kali ini? Kita akan mengeksplorasi seluk beluk pemrosesan kumpulan data. Maka dari itu, kencangkan sabuk, duduk yang tegap, dan hadirkan semangat yang tinggi karena kita akan mengungkap rahasia di balik kumpulan data. Yuk, pindah ke materi berikutnya.

## Kumpulan Data Bersarang

Sebelumnya, kita telah belajar dan memahami konsep tipe data pada bahasa pemrograman Python. Tipe data dibagi menjadi dua berdasarkan sifatnya, yaitu primitif dan non-primitif. Boolean, numerik, dan string termasuk dalam tipe data primitif, sedangkan non-primitif adalah kumpulan data yang mengandung unsur data primitif ataupun non-primitif.

Mengapa kita sedikit flashback ke materi sebelumnya? Ya, perlu Anda ketahui bahwa materi kumpulan data adalah tipe data non-primitif. Sebab, tipe data primitif tidak bisa dibuat bersarang. Jadi, kita perlu mendalami tentang tipe data non-primitif ini.

### Data yang Variatif

Umumnya, suatu kumpulan data mengandung elemen dengan tipe data yang sama. Apabila ada kumpulan data numerik, objek tersebut akan berisi angka saja. Begitu pun untuk kumpulan data boolean atau string, objek akan mengandung boolean atau teks saja. Akan tetapi, Anda dapat membuat kumpulan data yang berisi beragam tipe data pada bahasa pemrograman Python.

Seluruh jenis data non-primitif dalam bahasa pemrograman Python, seperti list, tuple, set, dan dictionary, dapat menyimpan beragam tipe data yang berbeda. Anda dapat mengisi boolean, string, dan numerik dalam satu variabel bertipe data non-primitif. Jadi, Anda mampu menyusun data yang kompleks dalam satu tempat untuk memudahkan pengaksesan nilai secara efisien.

|  |
| --- |
| Ada berbagai objek benda yang tersedia di kelas Bima. Tidak hanya murid saja yang ada dalam kelas, tetapi ada papan tulis, buku, tas, jam dinding, dll. Setiap objek benda memiliki informasi, seperti nama, warna, bahkan tanggal benda itu dipakai. Beragam benda tersebut berada dalam kelas yang notabene berisi murid.  [Beragam benda ada di dalam kelas.](https://www.dicoding.com/academies/620/tutorials/35093?from=35088)  Layaknya sebuah kelas, bahasa pemrograman Python pun memiliki cara untuk menyimpan kumpulan data yang beragam. Anda dapat mengisi berbagai jenis data dalam satu variabel bertipe data non-primitif. Tujuannya adalah mengelompokkan data pada satu wadah untuk memudahkan proses pengaksesan nilai. |

Berikut adalah contoh berbagai tipe data non-primitif yang memiliki beragam tipe data.

1. mejaKelas = [30, "cokelat", True]
2. infoBima = (17, "lulusan sekolah negeri")
3. dataSet = {1, 3.14, 'hello', True}
4. detailKelas = {"jumlahMeja": 30, "mejaGuru": True, "namaKelas": "11-A"}

Di balik keunggulan suatu nilai bertipe data non-primitif, ternyata ada risiko yang perlu Anda ketahui apabila mengisinya dengan beragam tipe data. Berikut adalah beberapa risikonya.

* **Data tidak konsisten**: suatu kumpulan data yang memiliki beragam tipe data berarti tidak memiliki konsistensi data. Anda perlu memastikan bahwa elemen data tersebut bernilai string, boolean, numerik, atau lainnya.
* **Kesulitan dalam pengelolaan**: apabila kumpulan data memiliki beragam tipe data, pengelolaannya akan semakin rumit. Anda harus cermat memanggil setiap elemen data supaya tidak terjadi kesalahan dalam memprosesnya.
* **Keterbatasan operasi**: operasi yang akan diterapkan pada setiap elemen akan berbeda-beda karena beragamnya tipe data. Ini akan menyulitkan pemrosesan karena operasi setiap elemen tidak bisa disamaratakan.

Dalam hal ini, Anda perlu memperhatikan konsep menyimpan kumpulan nilai. Anda dapat mengelompokkan data dengan tipe yang serupa dalam satu variabel non-primitif. Setiap tipe data non-primitif memiliki fungsi dan peran yang berbeda-beda. Gunakanlah struktur data yang sesuai dengan kebutuhan, seperti list untuk menyimpan data homogen atau dictionary untuk menyimpan data dengan kombinasi key-value.

### Data yang Bersarang

Kita perlu memahami bahwa kumpulan data seharusnya berisi nilai yang homogen untuk mengurangi risiko pengaksesan data. Homogen yang dimaksud adalah seragam atau sejenis. Misalnya, Anda ingin mengelompokkan nilai siswa pada lima murid. Jadi, Anda hanya menyimpan data nilai bertipe data numerik.

1. nilaiList = [97, 89, 75, 84, 93]

Adakalanya, kita perlu menyimpan data dengan jumlah yang cukup besar. Dalam satu kelas, mungkin hanya ada 30 nilai yang perlu disimpan. Namun, apa jadinya bila nilai satu sekolah dikumpulkan dalam satu variabel? Bagaimana cara membedakan antara nilai satu dengan lainnya?

Itulah salah satu alasan kita memerlukan kumpulan data secara bersarang. Kondisi bersarang memudahkan kita untuk **menyimpan data secara terstruktur**. Dengan begitu, setiap elemen **memiliki relasi antar elemen lainnya** dengan jelas dan mudah dipahami. Selain itu, berikut adalah beberapa alasan bahwa kita perlu menyimpan kumpulan data secara bersarang.

* **Mengelompokkan data yang serupa**: suatu kumpulan data yang bersarang berarti memiliki tipe data non-primitif yang bersusun sehingga isinya seragam.
* **Mempermudah akses dan proses manipulasi data**: apabila menyimpan kumpulan data yang bersarang, kita dapat mengakses dan memanipulasi elemennya dengan mudah.
* **Mampu menangani data yang kompleks**: beberapa kasus dapat diselesaikan dengan bantuan kumpulan data yang kompleks karena mampu menyimpan data secara terstruktur dan mudah diolah.

|  |
| --- |
| Bayangkan Bima bersekolah di SMA yang memiliki tiga tingkat, yaitu tingkat 10, 11, dan 12. Setiap tingkat setidaknya memiliki empat kelas, mulai dari kelas A hingga D. Setiap kelasnya berjumlah 30 murid. Jadi, total murid pada sekolah yang Bima tempati adalah 3 x 4 x 30 = 360 murid.  [Deretan pintu kelas di sekolah Bima.](https://www.dicoding.com/academies/620/tutorials/35093?from=35088)  Lalu, bagaimana guru mengumpulkan nilai keseluruhan murid? Apakah seluruh nilai digabung menjadi satu wadah atau buku? Tentu tidak. Guru akan mengelompokkan nilai berdasarkan mata pelajaran, kelas, dan tingkat. Dengan begitu, struktur nilai menjadi terorganisasi secara baik.  Sama halnya dengan kumpulan data pada pemrograman, Anda mampu mengelompokkan suatu data dalam satu nilai dengan cara bersarang. Setiap objek yang sama dikelompokkan menjadi satu, kemudian digabung dengan objek lainnya yang memiliki tingkatan sama hingga menghasilkan satu kumpulan data bersarang. |

Dalam bahasa pemrograman Python, setiap jenis data non-primitif memiliki kemampuan untuk mengisi data secara bersarang. Contohnya, list dalam list, tuple dalam tuple, dan dictionary dalam dictionary. Tidak hanya satu tingkatan saja, bahkan satu nilai dapat memiliki tingkatan dalam jumlah yang banyak.

Perlu diingat, banyaknya elemen kumpulan data berbanding lurus dengan banyaknya memori yang dipakai. Dengan kata lain, semakin banyak elemen tersimpan, baik bersarang maupun tidak, kapasitas memori akan semakin banyak dipakai. Batasan ukuran atau elemen kumpulan data pada Python bergantung dengan platform dan perangkat fisik yang dipakai.

#### List dan Tuple Bersarang

Lalu, bagaimana membuat list dalam list atau tuple dalam tuple? Secara struktur, membuat list atau tuple yang bersarang lebih mudah dibandingkan dengan dictionary. Mengapa demikian? Struktur penulisan dalam list atau tuple lebih ringkas dan proses pengaksesannya lebih mudah dengan memanggil indeks.

Berikut adalah contoh list bersarang pada bahasa pemrograman Python.

1. nilaiBimaList = [[98, 82], [78, 89], [84, 92]]

Anda bisa melakukan deklarasi tuple dengan cara yang sama seperti kode di atas.

1. nilaiBimaTuple = ((98, 82), (78, 89), (84, 92))

Variabel nilaiBimaList dan nilaiBimaTuple merupakan suatu kumpulan data yang bersarang. Dalam list atau tuple utama ada nilai yang memuat kumpulan data lagi. Apabila indeks ke-0 diakses, output yang ditampilkan adalah list atau tuple, begitu pun untuk indeks lainnya.

Setelah menentukan isi dari list atau tuple, bagaimana cara mengaksesnya? Sangatlah mudah. Misalnya kita ingin menampilkan angka 78 pada layar konsol. Angka 78 berada pada list/tuple berindeks ke-0 dari list/tuple utama berindeks ke-1. Terbayangkah? Mari kita coba pada kode di bawah ini.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run1-editor1)

1

# deklarasi list/tuple bersarang

2

nilaiBimaList = [[98, 82], [78, 89], [84, 92]]

3

nilaiBimaTuple = ((98, 82), (78, 89), (84, 92))

4

​

5

# mengakses angka 78 dari list/tuple bersarang

6

angka78DariList = nilaiBimaList[1][0]

7

angka78DariTuple = nilaiBimaTuple[1][0]

8

​

9

print(angka78DariList)

10

print(angka78DariTuple)

11

​

12

'''

13

Output:

14

78

15

78

16

'''

 Input  Reset

 Jalankan

Voilà, angka 78 dapat ditampilkan pada layar konsol, baik list ataupun tuple. Kita perlu mengakses list utama pada indeks ke-1, kemudian masuk lagi ke indeks ke-0 untuk mengambil angka 78. Itu adalah cara manual untuk menampilkan data ke konsol melalui pencacahan indeks. Namun, ada cara lain untuk menampilkan seluruh elemen dalam bentuk tabel.

Masih ingatkah Anda dengan perulangan berdasarkan pencacahan? Ya, kita dapat mencacah atau menguraikan elemen dalam suatu list dengan perulangan berdasarkan pencacahan. Kita memerlukan dua perulangan untuk menguraikannya karena ada dua tingkatan list. Jadi, perulangannya akan seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run2-editor1)

1

nilaiBimaList = [[98, 82], [78, 89], [84, 92]]

2

​

3

# Menampilkan setiap elemen dari list nilaiBimaList menggunakan for loop

4

for sublist in nilaiBimaList:

5

for nilai in sublist:

6

print(nilai, end=" ")

7

print()

8

​

9

'''

10

Output:

11

98 82

12

78 89

13

84 92

14

'''

 Input  Reset

 Jalankan

Perulangan pertama akan menguraikan list utama sehingga didapatkan elemen list [98, 82], [78, 89], dan [84, 92]. Perulangan kedua akan menguraikan elemen list dari proses pencacahan pertama dan ditampilkan dalam teks yang dipisahkan dengan spasi. Setelah elemen list indeks ke-0 sudah diuraikan, program akan mencetak teks kosong untuk membuat baris baru dan dilanjutkan ke indeks berikutnya. Hal ini juga berlaku untuk variabel bertipe data tuple.

#### Dictionary Bersarang

Bagaimana dengan dictionary? Tenang, tidak perlu khawatir. Untuk dictionary, tipe ini juga mampu menyimpan kumpulan data dalam kumpulan data lainnya. Berikut adalah contoh dictionary bersarang pada bahasa pemrograman Python.

1. keluargaFajar = {
2. "bapak": {
3. "nama": "Bapak Fajar",
4. "pekerjaan": "arsitek dan mandor"
5. },
6. "ibu": {
7. "nama": "Ibu Fika",
8. "pekerjaan": "ibu rumah tangga dan usaha katering"
9. },
10. "anak": {
11. "nama": "Bima",
12. "pekerjaan": "pelajar dan guru les"
13. }
14. }

Berdasarkan kode di atas, terdapat dictionary dalam dictionary. Bagian mana, ya? Perhatikan key bapak, ibu, dan anak, ia merupakan dictionary utama yang memiliki value bertipe data dictionary. Struktur data ini dikatakan bersarang karena variabel **keluargaFajar** memiliki beberapa tingkatan dictionary dalam satu dictionary utama.

Untuk mengakses nilai dari salah satu pekerjaan pada variabel **keluargaFajar**, Anda dapat memanggilnya dengan cara berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run3-editor1)

1

keluargaFajar = {

2

"bapak": {

3

"nama": "Bapak Fajar",

4

"pekerjaan": "arsitek dan mandor"

5

},

6

"ibu": {

7

"nama": "Ibu Fika",

8

"pekerjaan": "ibu rumah tangga dan usaha katering"

9

},

10

"anak": {

11

"nama": "Bima",

12

"pekerjaan": "pelajar dan guru les"

13

}

14

}

15

​

16

pekerjaanIbu = keluargaFajar["ibu"][“pekerjaan"]

17

print(pekerjaanIbu)

18

​

19

# Output: ibu rumah tangga dan usaha katering

 Input  Reset

 Jalankan

Sedikit lebih panjang dibandingkan dengan list atau tuple, ya? Betul. Dictionary ini mengandalkan key untuk mengaksesnya. Untuk itu, kita perlu mengetahui nama dari key yang akan diakses. Hal ini akan bermanfaat ketika data sudah cukup banyak dengan pemanggilan yang bersifat konsisten.

Adakah cara untuk menampilkan keseluruhan nilai dictionary bersarang menggunakan perulangan? Tentu saja ada. Namun, ada cara lebih optimal dengan menampilkan elemen pada perulangan dictionary. Kita perlu mengetahui nama dari key untuk menampilkan elemen dictionary. Berbeda dengan list/tuple yang mampu dicacah secara otomatis melalui perulangan for.

Jadi, berikut adalah proses pencacahan nilai dictionary bersarang menggunakan perulangan.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run4-editor1)

1

keluargaFajar = {

2

"bapak": {

3

"nama": "Bapak Fajar",

4

"pekerjaan": "arsitek dan mandor"

5

},

6

"ibu": {

7

"nama": "Ibu Fika",

8

"pekerjaan": "ibu rumah tangga dan usaha katering"

9

},

10

"anak": {

11

"nama": "Bima",

12

"pekerjaan": "pelajar dan guru les"

13

}

14

}

15

​

16

for anggotaKeluarga, data in keluargaFajar.items():

17

print(anggotaKeluarga)

18

for kategori, nilai in data.items():

19

print(f"{kategori}: {nilai}")

20

print()

 Input  Reset

 Jalankan

Kita dapat memanfaatkan **method** bawaan dari Python untuk mencacah key dan value dalam dua variabel berbeda, yaitu dengan method .items(). Method tersebut akan menguraikan dictionary menjadi dua variabel supaya dapat diulang oleh perulangan for. Variabel pertama bernama anggotaKeluarga yang berisi key, sedangkan variabel kedua bernama data yang berisi dictionary. Variabel tersebut akan diulang sebanyak kombinasi key-value dalam dictionary utama.

Untuk perulangan berikutnya, kita menerapkan konsep yang serupa seperti sebelumnya. Anda perlu menguraikan dictionary data supaya dapat mengambil key-value-nya. Kemudian, kita dapat mencetak key dan value menggunakan variabel kategori dan nilai. Selanjutnya, kita dapat menggunakan perintah print() untuk memisahkan antara anggota satu dengan lainnya.

#### Set Bersarang

Bagaimana dengan set bersarang? Apakah set dalam set dapat dilakukan pada Python? Mari kita coba terlebih dahulu.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run5-editor1)

1

set1 = {1, 2, 3}

2

set2 = {4, 5, 6}

3

set3 = {set1, set2}

4

​

5

print(set3)

6

​

7

'''

8

Output:

9

Traceback (most recent call last):

10

File "main.py", line 3, in <module>

11

TypeError: unhashable type: 'set'

12

'''

 Input  Reset

 Jalankan

Waduh, kok error? Ternyata, set dalam set tidak dapat dilakukan pada Python. Mengapa demikian?

Masih ingatkah Anda bahwa set memiliki sifat mutable? Ya, set memiliki sifat mutable yang bisa menambah dan menghapus elemen di dalamnya. Ada satu hal lain yang perlu diingat bahwa set tidak boleh diisi oleh nilai duplikasi antar elemennya.

Ketika kita memasukkan set ke dalam set, ada potensi perubahan yang dapat memengaruhi keunikan set. Hal ini dapat mengakibatkan perubahan yang tidak terduga dalam struktur set. Jadi, kumpulan data set dalam set tidak dapat dilakukan.

Lalu, apakah set pada Python tidak dapat dijadikan sebagai kumpulan data yang bersarang? Uniknya, tuple bersarang dapat dimasukkan ke set. Tuple memiliki sifat immutable yang nilainya tidak dapat diubah setelah dibuat. Hal ini membuat tuple menjadi objek yang aman digunakan sebagai elemen dalam set sehingga keunikan set tetap terjaga.

Berikut adalah contoh kumpulan data tuple dalam set pada Python.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run6-editor1)

1

set1 = (1, 2, 3)

2

set2 = (4, 5, 6)

3

set3 = {set1, set2}

4

​

5

print(set3)

6

​

7

# Output: {(4, 5, 6), (1, 2, 3)}

 Input  Reset

 Jalankan

Anda dapat menguraikan kumpulan data tersebut menggunakan perulangan layaknya list/tuple bersarang. Berikut contohnya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35093?from=35088#run7-editor1)

1

set1 = (1, 2, 3)

2

set2 = (4, 5, 6)

3

set3 = {set1, set2}

4

​

5

# Mencacah elemen set dalam set

6

for setItem in set3:

7

for value in setItem:

8

print(value, end=" ")

9

print()

10

​

11

'''

12

Output:

13

1 2 3

14

4 5 6

15

'''

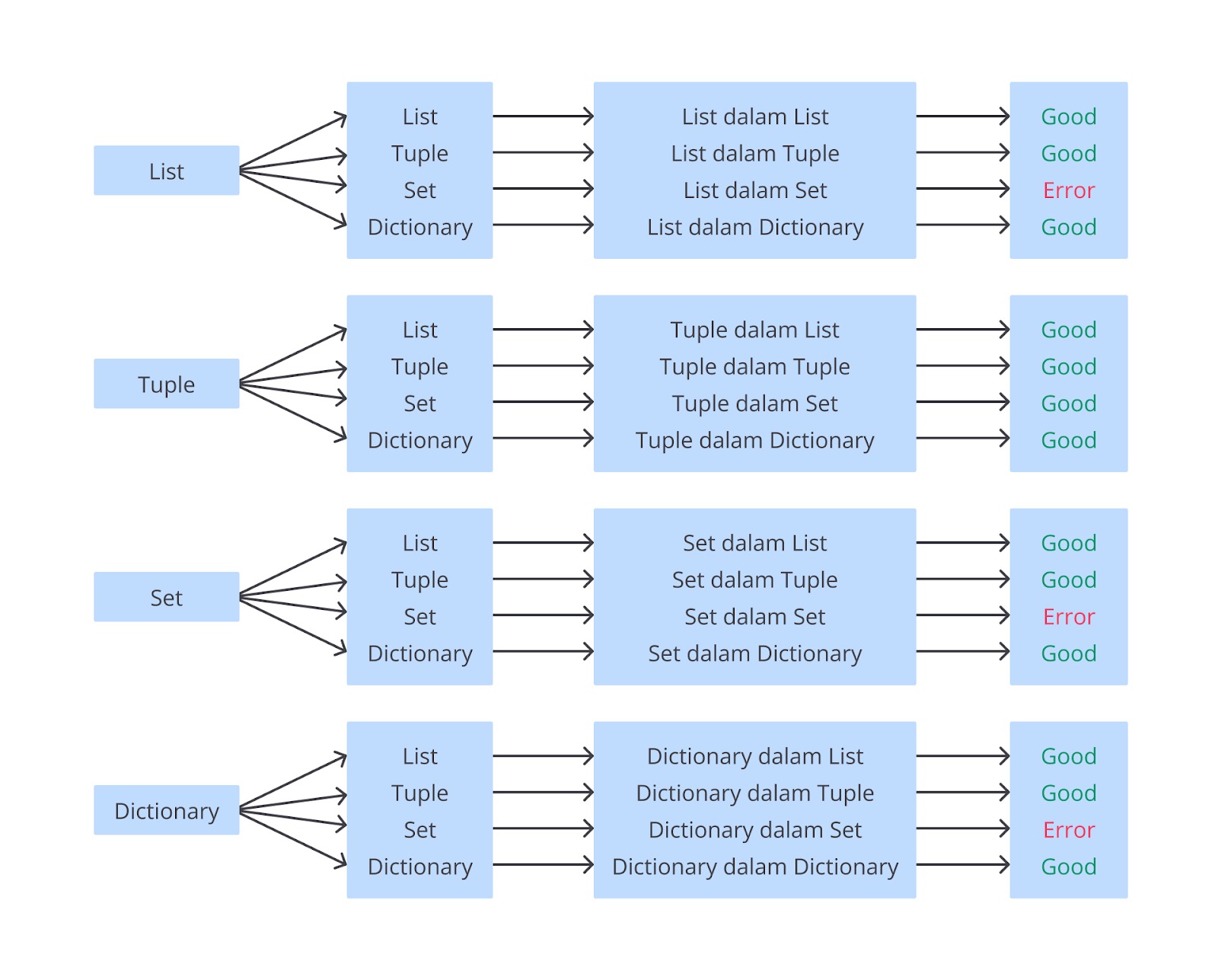
 Input  Reset

 Jalankan

#### Kombinasi Kumpulan Data yang Bersarang

Umumnya, data bertipe non-primitif dapat diubah menjadi kumpulan data bersarang. Bahkan, kita bisa crossover atau menyeberang antar kumpulan data, seperti list dalam dictionary, tuple dalam list, dictionary dalam tuple, dll. Akan tetapi, berdasarkan percobaan di atas, tidak semua tipe data non-primitif dapat diubah menjadi kumpulan data bersarang.

Bagaimana mengetahui kumpulan data bersarang yang kita buat tidak mengalami error? Nih, ada gambaran bagi Anda supaya tidak terjadi error saat membuat kumpulan data bersarang. Berikut adalah diagram yang menggambarkan error atau tidaknya kode saat membuat kumpulan data bersarang.

[](https://www.dicoding.com/academies/620/tutorials/35093?from=35088)

Bagaimana dengan materinya? Seru sekali, bukan? Saat ini kita mampu memahami kumpulan data yang variatif beserta risikonya dan data yang bersarang secara komprehensif. Apabila Anda ingin mengeksplorasi struktur data secara mendalam, silakan kunjungi tautan berikut.

* [Data Structures](https://docs.python.org/3/tutorial/datastructures.html#data-structures)

## Pencarian Data

Kita sebagai pengembang aplikasi terkadang menginginkan atau mengambil suatu data dari kumpulan data yang ada. Namun, dengan banyaknya data pada suatu kumpulan data, proses pencarian data menjadi tugas yang menantang. Oleh sebab itu, penting bagi kita untuk memahami cara yang tepat dalam mencari data agar lebih efisien dan tepat guna.

Pada materi ini, kita akan belajar mengenai teknik pencarian data, baik dalam kumpulan data yang terurut maupun tidak. Kita akan mengungkap metode pencarian secara linear untuk mencari data dalam cakupan kecil ataupun besar. Selain itu, kita perlu menilik lebih lanjut terkait metode pencarian lain yang lebih efisien untuk mendapatkan data berdasarkan susunan data yang terurut.

Secara umum, pencarian data dapat dilakukan ketika ada suatu kumpulan data yang memiliki beragam nilai. Kita dapat mencari satu atau lebih data yang ada di dalamnya untuk menemukan informasi spesifik atau memverifikasi keberadaan suatu nilai tertentu.

Kita perlu mendeskripsikan data yang ingin dicari agar sistem mampu mengomparasinya. Apabila data tidak dijelaskan ciri-cirinya atau kriterianya, sistem tidak akan mampu membandingkan antara data yang ingin dicari dengan yang ada di kumpulan data.

|  |
| --- |
| Ketika ingin memotong bayam, Bima ingin mengambilnya pada kumpulan sayur yang pernah dia kelompokkan. Bima perlu mencari tahu informasi tentang bayam supaya mampu membedakan antara bayam dengan sayur lainnya. Informasi yang perlu dia ketahui adalah berwarna hijau, memiliki daun yang cenderung bulat dan tipis, serta batang yang kecil.  [Bima sedang bingung memilih bayam di antara sayuran yang tersedia di meja dapur.](https://www.dicoding.com/academies/620/tutorials/35098?from=35093)  Hal tersebut terjadi ketika kita ingin mencari suatu data pada kumpulan data. Kita memerlukan informasi valid untuk membandingkan antara data yang ingin dicari dengan yang ada di kumpulan data. Ketika informasi dari keduanya serasi atau cocok, data itulah yang kita pilih dan dilanjutkan ke aksi berikutnya. |

Lalu, bagaimana cara menemukan data pada suatu kumpulan data? Ada beragam metode untuk mencari data. Penasaran? Yuk, simak materi ini hingga selesai.

### Pencarian Sekuensial

Dari penamaannya, pencarian ini dilakukan secara sekuensial dengan membandingkan antara data yang ingin dicari dan elemen kumpulan data. Ia akan membandingkan mulai dari elemen pertama hingga akhir. Umumnya, pencarian sekuensial ini disebut sebagai **linear search** karena ia akan mencari data dari awal hingga akhir elemen secara berurutan.

Gambaran dari proses pencarian sekuensial dapat Anda lihat pada ilustrasi berikut.

Ada beberapa cara untuk memahami konsep pencarian sekuensial pada kode. Mau tahu apa saja? Berikut adalah beberapa cara untuk mencari data secara sekuensial.

#### Pencarian Sekuensial Tanpa Boolean

Metode ini mengusung pencarian data secara sekuensial menggunakan perulangan tanpa memanfaatkan variabel boolean. Ia akan memeriksa seluruh elemen dari awal hingga akhir dan membandingkannya dengan data yang ingin dicari.

Lalu, bagaimana cara kita menghentikan perulangan tanpa adanya variabel boolean? Dalam perulangan, kita bisa memanfaatkan dua kondisi yang digabungkan, yaitu kondisi state untuk mengiterasi perulangan dan kondisi ekspresi perbandingan dua data yang bernilai sama. Perulangan akan berulang ketika kedua kondisi tersebut bernilai True. Sebaliknya, perulangan akan terhenti ketika salah satu dari kondisi tersebut bernilai False.

Sudah terbayang konsep metode ini? Untuk lebih jelasnya, mari simak langkah-langkah berikut.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [23,99,55,49,89,72,9,13,42,62]
2. Tentukan nilai yang ingin dicari.
   1. number\_what\_we\_looking\_for = 42
3. Definisikan state awal bernilai 0. State ini digunakan untuk mengiterasikan perulangan dari 0 hingga n-1 dan n adalah jumlah elemen pada kumpulan data.
   1. i = 0
4. Lakukan perulangan menggunakan metode evaluasi berhenti (while loop) dengan dua kondisi, yaitu iterasi state dan ekspresi perbandingan dua nilai. Isi perulangan hanyalah penambahan state supaya proses iterasi berjalan.
   1. while i < len(numbers) and numbers[i] != number\_what\_we\_looking\_for:
   2. i = i + 1

State dan ekspresi perbandingan berfungsi sebagai stopper atau penghenti perulangan. Ketika salah satu kondisi bernilai False, perulangan akan terhenti. Artinya, data yang ingin dicari sudah ditemukan pada indeks ke-i.

1. Saat perulangan berakhir, state i menyimpan nilai indeks kumpulan data. Jadi, kumpulan data yang berindeks ke-**i**adalah data yang kita cari. Kita dapat mencetaknya pada layar konsol.
   1. if numbers[i] == number\_what\_we\_looking\_for:
   2. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")
   3. else:
   4. print("Angka tidak ditemukan dalam kumpulan data.")
2. Mantap! Proses pencarian data tanpa menggunakan state boolean sudah selesai. Apabila keseluruhan kode dijalankan, output yang dihasilkan program akan seperti berikut.
   * [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run1-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 42

3

​

4

i = 0

5

​

6

while i < len(numbers) and numbers[i] != number\_what\_we\_looking\_for:

7

i = i + 1

8

​

9

if numbers[i] == number\_what\_we\_looking\_for:

10

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")

11

else:

12

print("Angka tidak ditemukan dalam kumpulan data.")

13

​

14

# Output: Angka 42 ditemukan pada indeks ke-8.

 Input  Reset

 Jalankan

Kode di atas bertujuan untuk mencari angka 42 pada kumpulan data numbers. Dalam kode, kita bisa melihat bahwa angka tersebut berada di urutan ke-9 atau indeks ke-8 dari kumpulan data. Ketika keseluruhan kode dijalankan, keluaran program akan seperti berikut. Bagaimana menurut kalian? Keren, bukan? Saat ini Anda sudah mampu mencari data yang diinginkan dalam suatu kumpulan data.

Metode ini akan mencari satu per satu data sesuai dengan yang kita inginkan. Ia akan mengiterasi dari awal hingga akhir kumpulan data maksimal 10 kali (karena ada 10 elemen) untuk mencari data yang sama. Jika nilai yang dicari ditemukan, pencarian berhenti dan elemen yang sesuai akan teridentifikasi.

Lalu, bagaimana jika data yang ingin kita cari tidak ditemukan? Apa yang terjadi pada program? Coba kita ubah nilai pada variabel number\_what\_we\_looking\_for menjadi **52**. Angka tersebut tidak tersedia pada kumpulan data. Mari kita lihat apa hasilnya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run2-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 52 # nilai diganti menjadi 52

3

​

4

i = 0

5

​

6

while i < len(numbers) and numbers[i] != number\_what\_we\_looking\_for:

7

i = i + 1

8

​

9

if numbers[i] == number\_what\_we\_looking\_for:

10

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")

11

else:

12

print("Angka tidak ditemukan dalam kumpulan data.")

13

​

14

'''

15

Output:

16

Traceback (most recent call last):

17

File "main.py", line 9, in <module>

18

IndexError: list index out of range

19

'''

 Input  Reset

 Jalankan

Mengapa program menghasilkan error? Berdasarkan kondisi berhenti pada perulangan, ia memanfaatkan dua kondisi, iterasi dan ekspresi perbandingan. Ketika ekspresi perbandingan dua nilai selalu bernilai True atau tidak ada data yang sama, perulangan akan mengandalkan state iterasi untuk menghentikannya.

Kondisi perulangan while akan terhenti saat nilai i telah melebihi jumlah elemen kumpulan data (dalam hal ini melebihi indeks 9). Ketika mencari data menggunakan pengondisian, numbers dengan indeks ke-10 akan mengalami error karena jumlah indeksnya dimulai dari 0 hingga 9.

Ada beberapa cara untuk menanggulangi error tersebut. Berikut caranya.

* **Menggunakan Fungsi**  
  Anda bisa membuat fungsi untuk mencari data menggunakan perulangan dan mengembalikan nilai berupa indeks kumpulan data. Jadi, tujuan fungsi ini untuk mengembalikan nilai indeks baik menemukan data maupun tidak. Ia akan memangkas beberapa langkah sehingga menghasilkan kode seperti berikut.
  + [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run3-editor1)

1

def sequential\_search\_without\_boolean(arr: list, target: any) -> int:

2

i = 0

3

while i < len(numbers):

4

if numbers[i] == number\_what\_we\_looking\_for:

5

return i

6

i = i + 1

7

return -1

8

​

9

numbers = [23,99,55,49,89,72,9,13,42,62]

10

number\_what\_we\_looking\_for = 52

11

​

12

output = sequential\_search\_without\_boolean(numbers, number\_what\_we\_looking\_for)

13

if output != -1:

14

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{output}.")

15

else:

16

print("Angka tidak ditemukan dalam kumpulan data.")

17

​

18

# Output: Angka tidak ditemukan dalam kumpulan data.

 Input  Reset

 Jalankan

Algoritma pada kode di atas memangkas langkah 3 sampai 5 menjadi satu fungsi. Ia mengembalikan indeks apabila elemen pada kumpulan data sama dengan data yang ingin kita cari. Ketika tidak ada yang sama, fungsi akan mengembalikan nilai -1 sebagai penanda bahwa data yang ingin dicari tidak tersedia di kumpulan data.

* **Menggunakan Metode Lain**  
  Beberapa metode lain dalam pencarian data mampu menanggulangi kesalahan yang terjadi pada algoritma sebelumnya. Ketika data yang diinginkan tidak ditemukan pada kumpulan data, program akan mengatasinya dengan keunikan masing-masing metode.

#### Pencarian Sekuensial dengan Boolean

Metode ini mengusung pencarian data secara sekuensial dengan memanfaatkan variabel boolean sebagai kondisi berhenti. Ia akan memeriksa elemen satu per satu menggunakan perulangan dan membandingkan elemen dengan data yang ingin dicari. Apabila nilai elemen sama dengan data yang ingin dicari, terjadi perubahan nilai boolean yang akan mengakibatkan perulangan berhenti.

Apakah Anda sudah bisa membayangkan cara kerja metode ini? Yuk, kita perinci proses pencarian sekuensial dengan boolean.

1. Definisikan kumpulan data homogen atau elemen yang bertipe sama.
   1. numbers = [23,99,55,49,89,72,9,13,42,62]
2. Tentukan nilai yang ingin dicari.
   1. number\_what\_we\_looking\_for = 42
3. Deklarasikan variabel boolean bernilai False dan state awal bernilai 0. Kedua variabel ini digunakan untuk menghentikan perulangan saat kondisi terpenuhi.
   1. found = False
   2. i = 0
4. Lakukan perulangan menggunakan while dengan menerapkan dua kondisi, yaitu iterasi state sesuai dengan jumlah elemen pada kumpulan data dan memanfaatkan perubahan variabel boolean. Dalam perulangan, ada pengondisian untuk membandingkan elemen dengan data yang ingin dicari. Variabel boolean berubah menjadi True ketika pengondisian bernilai True. Sebaliknya, state akan bertambah satu ketika pengondisian bernilai False.
   1. while i < len(numbers) and not found:
   2. if numbers[i] == number\_what\_we\_looking\_for:
   3. found = True
   4. else:
   5. i = i + 1

State dan variabel boolean pada perulangan adalah kunci untuk menemukan data yang ingin dicari. Apabila salah satu dari kedua kondisi tersebut bernilai False, perulangan akan terhenti.

1. Ketika perulangan berakhir dengan variabel boolean bernilai True, state akan merepresentasikan indeks kumpulan data yang berisi nilai pencarian. Apabila variabel boolean masih bernilai False, data yang ingin kita cari tidak tersedia di kumpulan data.
   1. if found:
   2. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")
   3. else:
   4. print("Angka tidak ditemukan dalam kumpulan data.")
2. Selesai! Anda telah berhasil membuat proses pencarian sekuensial menggunakan data boolean.

Ketika keseluruhan kode dijalankan, keluaran program akan seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run4-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 42

3

​

4

found = False

5

i = 0

6

​

7

while i < len(numbers) and not found:

8

if numbers[i] == number\_what\_we\_looking\_for:

9

found = True

10

else:

11

i = i + 1

12

​

13

if found:

14

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")

15

else:

16

print("Angka tidak ditemukan dalam kumpulan data.")

17

​

18

# Output: Angka 42 ditemukan pada indeks ke-8.

 Input  Reset

 Jalankan

Apakah output-nya benar? Tentu saja. Ia akan menyusuri seluruh elemen dari awal hingga menemukan data yang ingin dicari.

Metode ini menggunakan variabel boolean untuk mengindikasikan data yang dicari telah ditemukan atau tidak. Apabila variabel boolean bernilai True, otomatis data telah ditemukan dan perulangan berakhir. Sebaliknya, perulangan akan berlanjut sampai state memenuhi kondisi sehingga iterasi terhenti. Kondisi tersebut merepresentasikan data yang dicari tidak tersedia pada kumpulan data.

Bagaimana kalau data yang kita inginkan tidak tersedia pada kumpulan data? Variabel found yang menjadi kunci dari metode ini. Pada awal sebelum perulangan, ada proses assignment pada variabel bernilai False. Ketika tidak ada data yang diinginkan dalam kumpulan data, variabel found masih bernilai False. Jadi, pada pengondisian terakhir, program akan menampilkan teks yang menunjukkan bahwa angka tidak tersedia dalam kumpulan data.

Melihat kode di atas, prosesnya sederhana, bukan? Pencarian dengan metode ini dapat menjadi solusi saat kumpulan data tidak terlalu besar atau jumlah elemen sedikit. Perlu Anda ingat, proses pencarian ini tidak hanya berjalan pada kumpulan data numerik saja, tetapi pada kumpulan data dengan tipe data lainnya, seperti string atau boolean.

#### Pencarian Sekuensial dengan Sentinel

Metode ini memanfaatkan suatu elemen fiktif yang disebut “sentinel” untuk menggantikan elemen terakhir pada kumpulan data. Elemen fiktif ini nilainya sama dengan data yang ingin dicari. Sentinel dianggap sebagai penjaga perulangan dengan harapan proses pencarian selalu berhasil ditemukan walaupun di akhir iterasi.

Lalu, bagaimana kita mengetahui bahwa data yang dicari berada di kumpulan data, padahal ada elemen fiktif di dalamnya? Ada dua kondisi untuk memastikan data yang dicari berada di kumpulan data.

* Data berada di kumpulan data mulai indeks ke 0 hingga n-1 dan **n**adalah jumlah elemen pada kumpulan data; atau
* Nilai pada elemen terakhir kumpulan data sama dengan data yang dicari.

Apabila salah satu dari kedua kondisi tersebut bernilai benar, data yang Anda cari berada di kumpulan data. Jika tidak, kumpulan data tidak menyediakan data yang Anda cari.

Bagaimana menurut Anda? Masih belum terbayang, ya, konsep dari pencarian sekuensial dengan sentinel? Mari kita coba perdalam langkahnya satu per satu.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [23,99,55,49,89,72,9,13,42,62]
2. Tentukan nilai yang ingin dicari.
   1. number\_what\_we\_looking\_for = 42
3. Deklarasikan state awal bernilai 0. Tidak menjadi kondisi berhenti pada perulangan, state ini berguna untuk menyimpan indeks saat perulangan berlangsung.
   1. i = 0
4. Kita perlu menyimpan elemen terakhir pada variabel dan menggantikannya dengan data yang dicari untuk mengubahnya menjadi sentinel. Gunakan method **pop()** untuk menghilangkan elemen terakhir pada kumpulan data dan simpanlah elemen terakhir pada variabel last. Jangan lupa untuk menambahkan elemen terakhir dengan data yang ingin dicari menggunakan method **append()**.
   1. last = numbers.pop()
   2. numbers.append(number\_what\_we\_looking\_for)
5. Lakukan perulangan menggunakan while dengan ekspresi perbandingan dua nilai sebagai kondisi berhenti. Isi perulangan hanyalah penambahan state supaya kita dapat menyimpan indeks.
   1. while numbers[i] != number\_what\_we\_looking\_for:
   2. i = i + 1
6. Kembalikan elemen terakhir pada kumpulan data.
   1. numbers[-1] = last
7. Saat ini, state i menyimpan indeks kumpulan data tempat data yang dicari itu berada. Lakukan pengondisian dengan dua kondisi untuk memastikan data yang dicari berada di kumpulan data seperti pada penjelasan sebelumnya. Jika kedua kondisi tersebut tidak terpenuhi, angka tidak ada di kumpulan data.
   1. if i < len(numbers) - 1 or numbers[-1] == number\_what\_we\_looking\_for:
   2. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")
   3. else:
   4. print("Angka tidak ditemukan dalam kumpulan data.")
8. Voilà! Proses pencarian data menggunakan sentinel sudah selesai.

Cukup panjang untuk menyusun algoritma pencarian data menggunakan sentinel. Ketika keseluruhan kode dijalankan, keluaran program akan seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run5-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 42

3

​

4

i = 0

5

​

6

last = numbers.pop()

7

numbers.append(number\_what\_we\_looking\_for)

8

​

9

while numbers[i] != number\_what\_we\_looking\_for:

10

i = i + 1

11

​

12

numbers[-1] = last

13

​

14

if i < len(numbers) - 1 or numbers[-1] == number\_what\_we\_looking\_for:

15

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")

16

else:

17

print("Angka tidak ditemukan dalam kumpulan data.")

18

​

19

# Output: indeks data = 8

 Input  Reset

 Jalankan

Konsepnya sentinel dipasang pada elemen terakhir suatu kumpulan data. Namun, tidak menutup kemungkinan sentinel dapat diletakkan di awal atau akhir elemen. Mengapa bisa? Iya, tergantung kebutuhannya. Jika pencarian data dilakukan secara maju atau dari indeks terkecil ke besar, sentinel diletakkan di akhir elemen, berlaku untuk kondisi sebaliknya.

Penerapan metode ini bertujuan untuk menghindari pengujian kondisi yang banyak pada setiap iterasi. Cukup sentinel yang berperan untuk menghentikan perulangan. Tingkat efisiensi ini menjadi keunggulan metode ini untuk mencari data pada kumpulan data.

Perlu diingat, proses pencarian dengan metode ini masih kurang efektif ketika kita dihadapkan dengan kumpulan data yang memiliki banyak elemen. Saat kita mencari angka 42, ia berada pada indeks terakhir. Kita perlu menyusuri satu per satu untuk mencapai indeks tersebut. Bagaimana dengan kumpulan data yang memiliki jumlah elemen lebih banyak? Nah, jawabannya akan dibahas dalam metode berikutnya.

#### Pencarian Sekuensial dengan Sorted List

Sorted list, artinya kumpulan data perlu diurutkan terlebih dahulu sebelum melakukan pencarian data. Setelah itu, proses pencarian data dapat dilakukan menggunakan metode yang sudah kita pelajari sebelumnya, yaitu tanpa boolean, dengan boolean, ataupun dengan sentinel.

Sangat mudah, bukan? Untuk memperjelas konsep metode ini, mari kita coba menggabungkan metode sorted list dengan penerapan sentinel pada akhir elemen. Yuk, simak penjelasannya.

* Definisikan kumpulan data homogen atau bertipe sejenis.
  1. numbers = [23,99,55,49,89,72,9,13,42,62]
* Tentukan nilai yang ingin dicari.
  1. number\_what\_we\_looking\_for = 42
* Urutkan kumpulan data dari nilai yang terkecil hingga terbesar dengan memanfaatkan fungsi **sorted()**. Jadi, kumpulan data akan tersusun secara rapi dari awal hingga akhir. Anda bisa memperbarui nilai variabel numbersdengan kumpulan data telah berurutan.
  1. numbers = sorted(numbers)
* Sisanya, lakukan hal yang sama seperti metode pencarian data dengan sentinel, mulai dari deklarasi state, mengganti elemen terakhir dengan sentinel, hingga mencetak indeks kumpulan data.
  1. # deklarasi state awal
  2. i = 0
  4. # mengganti elemen akhir dengan sentinel
  5. last = numbers.pop()
  6. numbers.append(number\_what\_we\_looking\_for)
  8. # melakukan perulangan untuk mencari data yang sama
  9. while numbers[i] != number\_what\_we\_looking\_for:
  10. i = i + 1
  12. # mengembalikan elemen terakhir dengan data yang sebelumnya
  13. numbers[-1] = last
  15. # mencetak indeks keberadaan data yang dicari apabila tersedia di kumpulan data
  16. if i < len(numbers) - 1 or numbers[-1] == number\_what\_we\_looking\_for:
  17. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")
  18. else:
  19. print("Angka tidak ditemukan dalam kumpulan data.")
* Ta-da! Proses pencarian data dengan sorted list telah selesai.

Apabila keseluruhan kode dijalankan, program akan menghasilkan output seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run6-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 42

3

​

4

numbers = sorted(numbers)

5

​

6

i = 0

7

​

8

last = numbers.pop()

9

numbers.append(number\_what\_we\_looking\_for)

10

​

11

while numbers[i] != number\_what\_we\_looking\_for:

12

i = i + 1

13

​

14

numbers[-1] = last

15

​

16

if i < len(numbers) - 1 or numbers[-1] == number\_what\_we\_looking\_for:

17

print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{i}.")

18

else:

19

print("Angka tidak ditemukan dalam kumpulan data.")

20

​

 Input  Reset

 Jalankan

Coba perhatikan output yang dihasilkan program. Mengapa angka 42 berada pada indeks ke-3, padahal metode sebelumnya selalu menghasilkan indeks ke-8? Ya, dugaan Anda benar. Program diawali dengan proses sorted list atau mengurutkan dari nilai terkecil hingga terbesar. Alhasil, angka 42 tidak berada pada indeks ke-8, melainkan pada indeks ke-3.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run7-editor1)

1

# sebelum diurutkan

2

numbers = [23,99,55,49,89,72,9,13,42,62]

3

print(numbers)

4

​

5

# setelah diurutkan

6

numbers = sorted(numbers)

7

print(numbers)

8

​

9

'''

10

Output:

11

[23, 99, 55, 49, 89, 72, 9, 13, 42, 62]

12

[9, 13, 23, 42, 49, 55, 62, 72, 89, 99]

13

'''

 Input  Reset

 Jalankan

Berdasarkan output di atas, kita dapat menarik kesimpulan bahwa pencarian angka 42 pada kumpulan data numbers menggunakan metode sorted list bisa mempersingkat waktu. Metode sebelumnya membutuhkan sembilan kali iterasi, tetapi pencarian angka 42 dengan sorted list hanya membutuhkan empat kali iterasi. Cukup mempersingkat waktu, bukan?

Eits, tunggu dulu. Apakah benar sorted list dapat mempersingkat waktu komputasi? Coba bayangkan Anda memiliki kumpulan data yang berisi angka 1 hingga 100 secara berurutan sehingga ada 100 elemen di dalamnya. Lalu, Anda ingin mencari angka 42. Apa yang terjadi pada proses perulangan? Perulangan akan beriterasi sebanyak 42 kali untuk menemukan angka yang sama. Ternyata, algoritma sorted list masih kurang efisien, ya.

Sorted list memang mengunggulkan pencarian data dengan kondisi data yang sudah berurutan. Data akan lebih mudah dicari dengan tingkat komputasi waktu yang setara dengan jumlah elemennya. Namun, hal ini akan kurang efisien ketika kita memiliki jumlah elemen yang banyak. Jadi, proses pencariannya akan membutuhkan waktu yang cukup banyak pula. Lalu, apakah ada cara lainnya? Lanjut ke metode berikutnya, ya.

### Binary Search

Metode binary search adalah salah satu solusi bagi Anda yang ingin mencari data pada kumpulan data yang cukup banyak. Konsepnya adalah mengurutkan data menjadi berurutan dan menemukan data yang ingin dicari. Lalu, apa bedanya dengan metode sebelumnya? Metode ini menggunakan pendekatan pembagian dan pengecekan pada elemen tengah untuk mencari data yang dicari.

Sudah bisa membayangkan cara kerjanya? Mungkin perlu diperinci lagi. Sebelum masuk ke kode, simak cara kerja binary search berikut, ya.

1. Tentukan nilai yang ingin dicari. Pada konteks ini, nilai yang dicari kita sebut sebagai **X**.
2. Pertama, urutkan kumpulan data dari yang terkecil hingga terbesar atau sebaliknya. Kondisi ini tidak menjadi masalah karena kumpulan data tetap terurut walaupun secara ascending maupun descending. Akan tetapi, algoritma akan lebih mudah dipahami apabila kumpulan data diurutkan secara ascending (dari kecil ke besar).
3. Tentukan batas awal dan akhir indeks untuk mengetahui range (rentang) pencarian datanya. Range ini akan berubah setiap iterasi perulangan berjalan.
4. Kemudian, tentukan elemen tengah berdasarkan indeks dari kumpulan data. Anda dapat menghitungnya dengan cara mengambil hasil bagi saja tanpa menghiraukan sisa bagi.
5. Selanjutnya, bandingkan elemen tengah tersebut dengan nilai X.
   1. Apabila elemen tengah sama dengan nilai X, proses pencarian berakhir. Anda telah menemukan data pada indeks tersebut.
   2. Apabila elemen tengah lebih besar dari nilai X, artinya nilai X berada di antara elemen indeks ke-0 hingga elemen tengah. Jadi, proses pencarian berlanjut pada setengah kumpulan data sebelum elemen tengah. Di sinilah batas akhir berubah nilai sehingga nilainya adalah indeks elemen tengah dikurang satu.
   3. Apabila elemen tengah lebih kecil dari nilai X, artinya nilai X berada di antara elemen tengah hingga elemen indeks ke-n. Jadi, proses pencarian berlanjut pada setengah kumpulan data setelah elemen tengah. Tahap ini batas awal berubah nilai sehingga nilainya adalah indeks elemen tengah ditambah satu.
6. Ulangi langkah keempat hingga data yang dicari telah ditemukan. Ada dua kondisi yang mengakibatkan perulangan berakhir, yaitu ketika elemen tengah sama dengan nilai X atau range pencarian data hanya menyisakan satu elemen saja. Opsi kedua akan terjadi ketika data yang diinginkan tidak ditemukan pada kumpulan data.

Supaya lebih paham lagi, Anda dapat melihat ilustrasi berikut. Resapi pola perubahan gambarnya agar Anda semakin mengerti konsep tentang binary search.

Setelah itu, mari kita berpindah ke kode untuk memahami konsep binary search ketika diterapkan pada program.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [23,99,55,49,89,72,9,13,42,62]
2. Tentukan nilai yang ingin dicari.
   1. number\_what\_we\_looking\_for = 42
3. Urutkan kumpulan data dari nilai terkecil hingga terbesar dengan memanfaatkan fungsi **sorted()**. Anda dapat memperbarui nilai variabel numbers dengan kumpulan data yang telah berurutan.
   1. numbers = sorted(numbers)
4. Tentukan batas awal dan akhir mula-mula untuk mengetahui range pencarian data.
   1. low = 0
   2. high = len(numbers) - 1
5. Deklarasikan beberapa state, seperti state boolean, untuk memberi tanda bahwa data sudah ditemukan dan state middle atau miduntuk menyimpan indeks elemen tengah.
   1. found = False
   2. mid = 0
6. Lakukan perulangan menggunakan while dengan menerapkan dua kondisi, yaitu pengondisian batas bawah kurang dari sama dengan batas atas dan perubahan variabel boolean. Ketika salah satu ada yang bernilai False, perulangan akan terhenti. Isi dari perulangan adalah berikut.
   1. Tentukan elemen tengah berdasarkan indeks dari kumpulan data.
   2. Pengondisian if dengan membandingkan elemen tengah dengan data yang ingin dicari.
      1. Jika hasilnya sama dengan data yang dicari, data telah ditemukan dan mengubah variabel boolean menjadi True.
      2. Jika hasilnya kurang dari data yang dicari, iterasi perulangan akan berlanjut dengan mengubah batas bawah menjadi mid - 1.
      3. Jika hasilnya lebih dari data yang dicari, iterasi perulangan akan berlanjut dengan mengubah batas atas menjadi mid + 1.
         1. while low <= high and not found:
         2. mid = (low + high) // 2
         3. mid\_value = numbers[mid]
         5. if mid\_value == number\_what\_we\_looking\_for:
         6. found = True
         7. elif mid\_value < number\_what\_we\_looking\_for:
         8. low = mid + 1
         9. else:
         10. high = mid - 1
7. Ketika perulangan berakhir dengan variabel boolean bernilai True, state mid akan merepresentasikan indeks kumpulan data berisi data yang dicari. Apabila variabel boolean masih bernilai False atau range pencarian menyisakan satu elemen, data yang ingin dicari tidak tersedia di kumpulan data.
   1. if found:
   2. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{mid}.")
   3. else:
   4. print("Angka tidak ditemukan dalam kumpulan data.")
8. Voilà! Proses pencarian data menggunakan binary search sudah selesai.

Apabila keseluruhan kode dijalankan, program akan menghasilkan output seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35098?from=35093#run8-editor1)

1

numbers = [23,99,55,49,89,72,9,13,42,62]

2

number\_what\_we\_looking\_for = 42

3

​

4

numbers = sorted(numbers)

5

​

6

low = 0

7

high = len(numbers) - 1

8

​

9

found = False

10

mid = 0

11

​

12

while low <= high and not found:

13

mid = (low + high) // 2

14

mid\_value = numbers[mid]

15

​

16

if mid\_value == number\_what\_we\_looking\_for:

17

found = True

18

elif mid\_value < number\_what\_we\_looking\_for:

19

low = mid + 1

20

else:

 Input  Reset

 Jalankan

Berdasarkan output program, angka 42 telah ditemukan pada indeks ke-3. Namun, apakah angka tersebut didapatkan pada perulangan dalam iterasi ke-4? Mari kita lihat ilustrasi berikut.

Ternyata, program melakukan iterasi sebanyak empat kali untuk mencari angka 42. Walaupun jumlah iterasinya sama dengan metode sorted list, cara untuk mencari angka sedikit berbeda.

Bayangkan ketika Anda memiliki kumpulan data numerik mulai angka 1 hingga 100. Kemudian, Anda ingin mencari angka 42. Apakah program akan melakukan iterasi sebanyak 42 kali? Tentu saja tidak. Dengan binary search, angka 42 pada kumpulan data tersebut dapat ditemukan pada iterasi ke-7. Ia melewati serangkaian elemen yang dianggap tidak perlu dibandingkan dengan data yang ingin dicari. Dengan begitu, eksekusi program akan lebih efisien dan cepat.

Wow, materinya sangat menarik , bukan? Saat ini, Anda sudah memahami beberapa konsep pencarian data mulai dari dasar hingga menghasilkan komputasi yang efisien. Mulai dari pencarian data tanpa boolean, dengan boolean, dengan sentinel, dengan sorted list, hingga binary search telah Anda kuasai.

Namun, jangan hanya belajar sampai tahap ini saja. Ternyata, ada banyak metode pencarian lainnya yang bisa Anda pelajari. Ada metode hash-based, binary tree, dan masih banyak lainnya. Apabila Anda tertarik untuk memahami beberapa metode tersebut, silakan kunjungi tautan berikut.

* [Algorithms in a Nutshell - Searching](https://learning.oreilly.com/library/view/algorithms-in-a/9780596516246/ch05.html).
* [Linear, Binary, and Hashing Search in JavaScript](https://javascript.plainenglish.io/linear-binary-and-hashing-search-in-javascript-908a54ca7658).

## Pengurutan Data

Proses pengurutan merupakan runtutan aksi yang dilakukan untuk menyusun elemen secara teratur pada suatu kumpulan data berdasarkan kriteria tertentu. Proses pengurutan data sering kali kita temui pada kehidupan sehari-hari, seperti daftar kontak dalam ponsel berdasarkan abjad, pengelompokan pakaian berdasarkan warna, pengurutan uang berdasarkan nominal, dan masih banyak lainnya.

|  |
| --- |
| Pada suatu hari, Bima datang ke perpustakaan sekolah untuk meminjam buku matematika. Dia perlu meminjam buku karena ingin mencari referensi terkait materi fungsi pada matematika. Bima pun mengambil beberapa buku yang ada di rak pada bagian matematika. Dia hanya perlu meminjam satu dari sekian buku yang diambil olehnya dari rak. Alhasil, Bima meninggalkan sisa buku yang dia ambil dalam troli khusus dan mengunjungi meja administrasi untuk meminjam buku tersebut kepada pustakawan.  [Troli perpustakaan yang penuh dengan tumpukan buku.](https://www.dicoding.com/academies/620/tutorials/35103?from=35098)  **Catatan:** Sebagai informasi, troli ini khusus digunakan untuk mengumpulkan buku-buku yang harus dikembalikan ke rak semula. Pengunjung perpustakaan tidak boleh sembarangan mengembalikan buku ke rak walaupun mereka mengetahui tempat asalnya. Hanya pustakawan yang boleh mengembalikan buku-buku tersebut.  Pada akhir hari jam kerja, pustakawan melakukan “ritual” mengembalikan buku dari troli ke masing-masing rak. Pustakawan perlu mengurutkan buku berdasarkan nomor, urutan abjad, dan kategori lainnya yang sesuai dengan kebijakan perpustakaan. Tujuan dari pekerjaan tersebut adalah untuk menciptakan sistematisasi dan organisasi dalam mengelompokkan serta mengurutkan buku perpustakaan.  Berdasarkan aksi tersebut, penyortiran data menjadi hal yang diperlukan untuk seorang pustakawan. Ia perlu mengelola sistem koleksi buku secara baik agar pengunjung mudah mencarinya. Hal ini memberikan daya tarik tersendiri bagi para pengunjung perpustakaan supaya lebih nyaman dan terbantu dalam mencari buku yang diminati. |

Penting bagi kita untuk memahami konsep pengurutan data. Tidak hanya mengurutkannya berdasarkan abjad atau angka, tetapi cara menyeleksi dan menyusun ulang kumpulan data menjadi hal yang perlu diperhatikan. Tingkat kompleksitas pengurutan data pun juga menjadi parameter penting supaya proses berjalan dengan efisien.

Anda harus memiliki keterampilan untuk mengurutkan data pada kasus-kasus tertentu. Tugas yang diemban sebagai pengembang aplikasi bukan mengurutkan 10 data saja, tetapi puluhan bahkan ratusan ribu data yang perlu diproses. Ketika pengelolaan data yang kurang efektif, tingkat pengalaman pengguna akan semakin turun. Oleh sebab itu, kita perlu mengetahui metode yang cocok untuk mengurutkan data dengan efektif.

Ada beberapa algoritma yang perlu Anda ketahui dalam mengurutkan data. Setiap algoritma memiliki tingkat kompleksitas dan performa yang berbeda-beda, bergantung pada jumlah data yang diurutkan. Anda mulai penasaran? Yuk, simak materi ini hingga selesai.

### Counting Sort

Counting sort adalah salah satu algoritma sederhana yang mampu mengurutkan kumpulan data dengan memanfaatkan rentang nilai terkecil dan terbesar. Algoritma ini bekerja dengan membuat slot kosong, menghitung frekuensi kemunculan data, memisahkan data satu dengan lainnya, kemudian diurutkan berdasarkan nilai yang terkecil.

Untuk memudahkan Anda memahami konsep algoritma counting sort, silakan perhatikan ilustrasi berikut.

Sudah mulai paham cara kerja counting sort? Saatnya kita pelajari cara algoritma counting sort bekerja dalam program Python. Simak baik-baik, ya!

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [1,3,2,5,1,6,1,3,2,1]
2. Tentukan rentang nilai elemen dari kumpulan data. Untuk mencari nilai tersebut, Anda dapat mencari nilai terkecil dan terbesar dari kumpulan data. Selanjutnya, hitunglah selisih kedua nilai dan tambahkan angka satu. Perhitungan tersebut menghasilkan rentang nilai elemen dari kumpulan data.
   1. max\_value = max(numbers)
   2. min\_value = min(numbers)
   3. range\_values = max\_value - min\_value + 1
3. Selanjutnya, Anda dapat membuat list kosong untuk menampung frekuensi kemunculan setiap elemen dalam kumpulan data. List ini memiliki total elemen sebanyak nilai rentang yang didapatkan sebelumnya.
   1. count = [0] \* range\_values
4. Berikutnya adalah menghitung frekuensi kemunculan data. Pada tahap ini, list kosong akan berisi frekuensi elemen yang ada di kumpulan data. Misalnya, angka 1 pada kumpulan data memiliki frekuensi sebanyak 4 data. Angka tersebut akan masuk dalam list kosong pada indeks ke-0. Begitu pun untuk frekuensi lainnya.
   1. for num in numbers:
   2. count[num - min\_value] += 1
5. Setelah dihitung frekuensinya, saatnya menghitung akumulasi jumlah list tersebut. Tujuannya adalah untuk mengetahui indeks awal dan akhir elemen tersebut berada. Misalnya, angka 1 memiliki frekuensi sebanyak 4 data. Jadi, kita dapat mengetahui alokasi slot untuk angka 1 mulai dari indeks ke-0 hingga ke-3. Begitu pun untuk elemen yang lainnya.
   1. for i in range(1, range\_values):
   2. count[i] += count[i - 1]
6. Kemudian, buat list kosong baru untuk menyimpan susunan list yang berurutan. Anda dapat menggunakan cara yang sama seperti langkah sebelumnya, tetapi total elemennya sebanyak list yang ingin diurutkan.
   1. sorted\_numbers = [0] \* len(numbers)
7. Sentuhan terakhir, Anda perlu mengurutkan kumpulan data. Kita bisa memanfaatkan perulangan, kumpulan data asli, dan frekuensi kumulatif yang sudah dibuat sebelumnya.  
   Pertama, lakukan perulangan dengan mengiterasi elemen kumpulan data. Kemudian, carilah indeks elemen sebagai slot untuk memasukkan data yang terurut. Beri assignment pada list kosong untuk meletakkan elemen berdasarkan indeks. Lalu, lakukan proses decrement indeks pada frekuensi kumulatif yang sudah terpakai. Lakukan secara berulang hingga elemen terakhir telah tereksekusi.
   1. for num in arr:
   2. index = count[num - min\_value] - 1
   3. sorted\_arr[index] = num
   4. count[num - min\_value] -= 1
8. Ta-da! Proses pengurutan menggunakan counting sort telah selesai.

Apabila keseluruhan kode dijalankan, program akan menghasilkan output seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35103?from=35098#run1-editor1)

1

numbers = [1,3,2,5,1,6,1,3,2,1]

2

​

3

max\_value = max(numbers) # Menentukan nilai maksimum dalam data

4

min\_value = min(numbers) # Menentukan nilai minimum dalam data

5

range\_values = max\_value - min\_value + 1 # Mencari rentang nilai data

6

​

7

# Membuat list kosong untuk menyimpan frekuensi kemunculan setiap nilai

8

count = [0] \* range\_values

9

​

10

# Menghitung frekuensi kemunculan setiap nilai dalam data

11

for num in numbers:

12

count[num - min\_value] += 1

13

​

14

# Mengakumulasi jumlah dalam count list

15

for i in range(1, range\_values):

16

count[i] += count[i - 1]

17

​

18

# Membuat list kosong baru untuk menyimpan data yang sudah diurutkan

19

sorted\_numbers = [0] \* len(numbers)

20

​

 Input  Reset

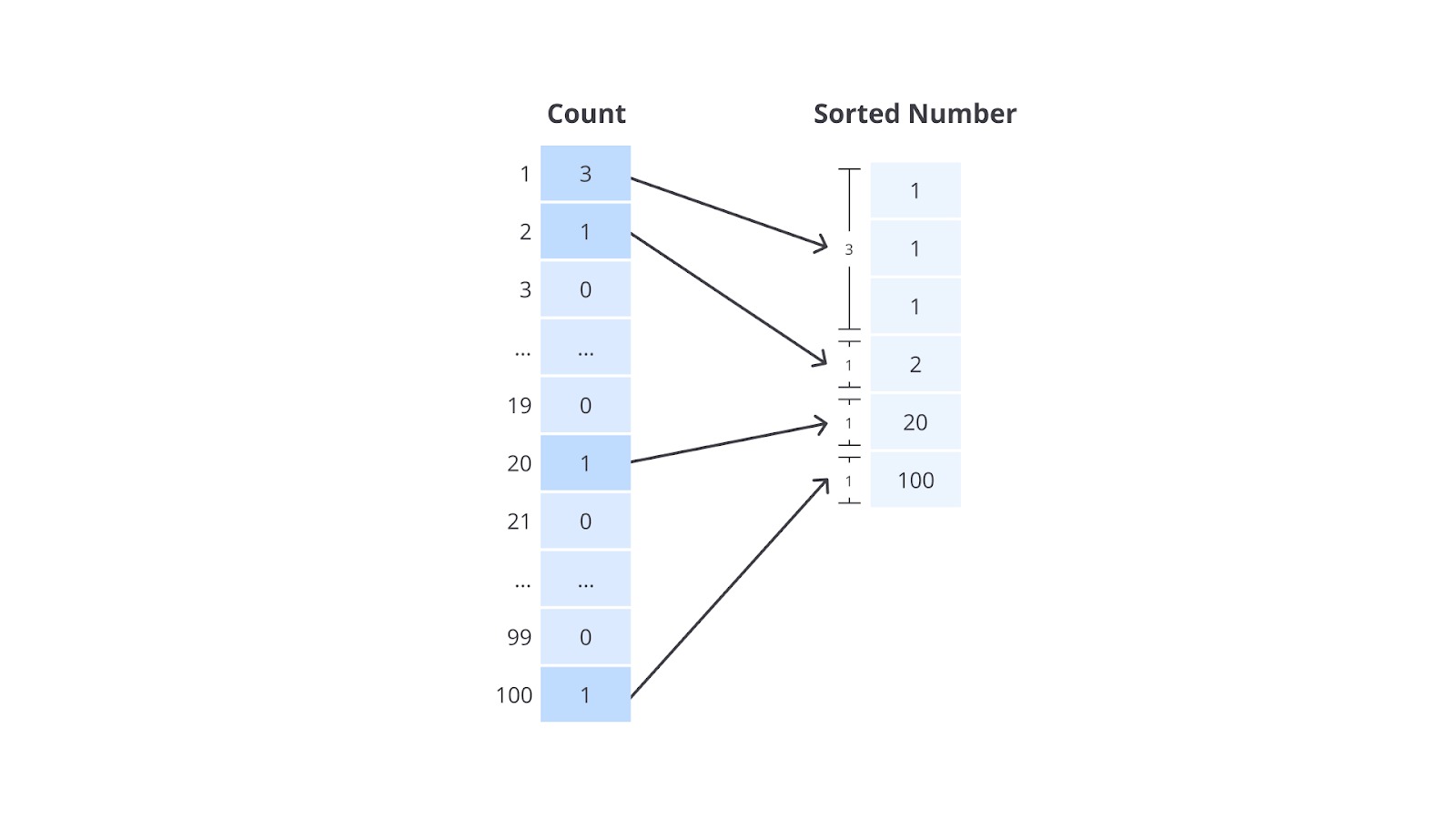
 Jalankan

Untuk mempermudah pembacaan kode, kita bisa mengelompokkan beberapa aksi menjadi satu fungsi bernama **counting\_sort**. Fungsi tersebut memiliki parameter list numerik dan mengembalikan list numerik juga. Maksud dari pembuatan fungsi ini adalah mengelompokkan serangkaian aksi yang memiliki tujuan, yaitu menghasilkan list yang berurutan, dan memudahkan pemanggilan apabila dibutuhkan untuk mengurutkan kembali. Jadi, kode program akan seperti berikut.

1. **def counting\_sort(numbers: list) -> list:**
2. **max\_value = max(numbers)  # Menentukan nilai maksimum dalam data**
3. **min\_value = min(numbers)  # Menentukan nilai minimum dalam data**
4. **range\_values = max\_value - min\_value + 1  # Mencari rentang nilai data**
5. **# Membuat list kosong untuk menyimpan frekuensi kemunculan setiap nilai**
6. **count = [0] \* range\_values**
7. **# Menghitung frekuensi kemunculan setiap nilai dalam data**
8. **for num in numbers:**
9. **count[num - min\_value] += 1**
10. **# Mengakumulasi jumlah dalam count list**
11. **for i in range(1, range\_values):**
12. **count[i] += count[i - 1]**
13. **# Membuat list kosong baru untuk menyimpan data yang sudah diurutkan**
14. **sorted\_numbers = [0] \* len(numbers)**
15. **# Mengatur data yang akan diurutkan berdasarkan count list**
16. **for num in numbers:**
17. **index = count[num - min\_value] - 1**
18. **sorted\_numbers[index] = num**
19. **count[num - min\_value] -= 1**
20. **return sorted\_numbers**
22. # main program
23. numbers = [1,3,2,5,1,6,1,3,2,1]
25. **sorted\_data\_counting\_sort = counting\_sort(numbers)**
26. print("Data sebelum diurutkan:", numbers)
27. print("Data setelah diurutkan:", sorted\_data\_counting\_sort)
29. '''
30. Output:
31. Data sebelum diurutkan: [1, 3, 2, 5, 1, 6, 1, 3, 2, 1]
32. Data setelah diurutkan: [1, 1, 1, 1, 2, 2, 3, 3, 5, 6]
33. '''

Berdasarkan cara kerjanya, algoritma counting sort ini memanfaatkan rentang nilai dari kumpulan data. Ia memberikan slot kosong untuk menyimpan frekuensi kumulatif.

Lalu, bagaimana jika rentang nilai dari kumpulan data sangat besar, tetapi total elemen hanya sedikit? Misalnya, kumpulan data berisi 6 angka, yakni [1, 20, 1, 100, 1]. Kemudian, rentang nilainya adalah 100. Alhasil, kita memerlukan 100 kumpulan data untuk menyimpan frekuensi kumulatif tersebut. Supaya Anda mendapatkan gambaran, mari tengok ilustrasi berikut.

[](https://www.dicoding.com/academies/620/tutorials/35103?from=35098)

Frekuensi elemen akan membuat kumpulan data sebanyak 100, tetapi yang berisi hanya 5 elemen. Sisanya hanyalah nilai kosong. Hal ini yang menyebabkan algoritma ini kurang efisien. Ia membutuhkan memori yang cukup banyak apabila rentang nilainya cukup besar. Oleh sebab itu, algoritma ini lebih cocok untuk data yang relatif kecil dan rentang nilainya tidak besar.

### Selection Sort

Selection sort adalah algoritma pengurutan data dengan memilih dan membandingkan dua elemen data yang berurutan, menemukan nilai terkecil atau terbesar, dan menukarnya dengan elemen yang belum diurutkan. Proses iterasi tersebut akan berlanjut hingga seluruh data berurutan, baik ascending maupun descending.

Untuk memudahkan Anda memahami konsep algoritma selection sort, perhatikan ilustrasi berikut.

Bagaimana dengan ilustrasi di atas? Cukup menggambarkan cara kerja selection sort, bukan? Untuk mengetahui langkah demi langkah, yuk, simak cara kerja algoritma selection sort berikut.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [1,3,2,5,1,6,1,3,2,1]
2. Untuk membedakan antara kumpulan data sebelum dan sesudah diurutkan, kita bisa menduplikasi kumpulan data pada variabel baru. Gunakan method **copy()**untuk menduplikasi kumpulan data.
   1. sorted\_numbers = numbers.copy()
3. Hitung total elemen dari kumpulan data. Nilai ini akan dipakai untuk melakukan perulangan pada setiap elemen kumpulan data.
   1. n = len(sorted\_numbers)
4. Lakukan perulangan untuk setiap elemen kumpulan data menggunakan for-in loop. Gunakan state iuntuk melakukan iterasi pada perulangan ini.
   1. Simpanlah indeks pertama pada variabel baru untuk menandakan bahwa indeks tersebut berisi data minimal.
   2. Lakukan perulangan lagi dengan state j untuk membandingkan dua elemen dalam kumpulan data. Apabila elemen berindeks j kurang dari elemen berindeks min\_indeks, program akan menyimpan indeks j sebagai indeks minimal. Iterasi akan terus berulang hingga seluruh elemen telah dibandingkan dengan elemen terakhir.
   3. Lakukan swap atau pergantian elemen antara indeks ke-idengan indeks minimum.
      1. for i in range(n):
      2. min\_index = i
      4. for j in range(i+1, n):
      5. if sorted\_numbers[j] < sorted\_numbers[min\_index]:
      6. min\_index = j

      9. sorted\_numbers[i], sorted\_numbers[min\_index] = sorted\_numbers[min\_index], sorted\_numbers[i]
5. Selesai! Anda telah berhasil menyelesaikan algoritma selection sort pada kode Python.

Apabila keseluruhan kode digabung dan dijalankan, program akan menghasilkan output seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35103?from=35098#run2-editor1)

1

def selection\_sort(numbers: list) -> list:

2

sorted\_numbers = numbers.copy()

3

n = len(sorted\_numbers)

4

for i in range(n):

5

# Anggap elemen pertama dari sisa data yang belum diurutkan adalah elemen terkecil

6

min\_index = i

7

8

# Mencari elemen terkecil dari sisa data yang belum diurutkan

9

for j in range(i+1, n):

10

if sorted\_numbers[j] < sorted\_numbers[min\_index]:

11

min\_index = j

12

13

# Menukar elemen terkecil dengan elemen pertama dari data yang belum diurutkan

14

sorted\_numbers[i], sorted\_numbers[min\_index] = sorted\_numbers[min\_index], sorted\_numbers[i]

15

16

return sorted\_numbers

17

​

18

# main program

19

numbers = [1,3,2,5,1,6,1,3,2,1]

20

​

 Input  Reset

 Jalankan

Alur program yang cukup mudah, bukan? Secara baris kode dan alur program memang cukup sederhana. Selain itu, kita tidak perlu menyimpan banyak variabel atau data yang terbuang sehingga proses perhitungan tidak terlalu kompleks.

Akan tetapi, proses perulangan yang bersarang membuat komputasi semakin lama. Ketika kumpulan data memiliki total elemen yang banyak, waktu eksekusi program akan semakin meningkat. Hal ini menjadikan algoritma selection sort tidak cocok untuk kumpulan data yang besar.

### Bubble Sort

Bubble sort adalah salah satu algoritma penyortiran sederhana yang akan bekerja dengan membandingkan dua elemen berdekatan dan menukarnya apabila mereka berada pada urutan yang salah. Kedua hal tersebut akan berlangsung mulai dari indeks terkecil hingga terbesar sampai tersusun kumpulan data yang berurutan.

Konsep bubble sort ini seperti gelembung air yang “mengapung” ke permukaan dan “tenggelam” ke dasar air. Elemen dengan nilai yang kecil akan “diapungkan” melalui proses pertukaran nilai. Sebaliknya, elemen dengan nilai yang besar akan diberi beban agar tenggelam ke dasar.

Anda dapat mengamati proses pertukaran nilai pada bubble sort melalui ilustrasi berikut.

Bagaimana dengan ilustrasi di atas? Cukup menggambarkan cara kerja bubble sort, bukan? Saatnya kita mengulik cara kerja bubble sort pada kode Python. Yuk, simak langkah-langkahnya hingga akhir.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [1,3,2,5,1,6,1,3,2,1]
2. Untuk membedakan antara kumpulan data sebelum dan sesudah diurutkan, kita bisa menduplikasi kumpulan data pada variabel baru. Gunakan method **copy()**untuk menduplikasi kumpulan data.
   1. sorted\_numbers = numbers.copy()
3. Hitung total elemen dari kumpulan data. Nilai ini akan dipakai untuk melakukan perulangan pada setiap elemen kumpulan data.
   1. n = len(sorted\_numbers)
4. Lakukan perulangan dengan state i untuk menjalankan iterasi dari indeks awal sampai akhir. Anda dapat menggunakan perulangan for-in loop untuk melakukan perulangan tersebut. Selanjutnya, lakukan perulangan bersarang dengan state j untuk mengulang indeks dari 0 hingga n-i-1. Tujuan dari perulangan bersarang adalah supaya mampu membandingkan data dengan elemen yang belum berurutan.
   1. Dalam perulangan bersarang ada pengondisian untuk membandingkan elemen j kurang dari elemen j+1. Apabila bernilai True, kedua elemen tersebut akan bertukar untuk mengurutkan dari terkecil ke terbesar.
      1. for i in range(n):
      3. for j in range(0, n - i - 1):
      5. if sorted\_numbers[j] > sorted\_numbers[j + 1]:
      6. sorted\_numbers[j], sorted\_numbers[j+1]= sorted\_numbers[j+1], sorted\_numbers[j]
5. Selesai! Anda sudah berhasil membuat program menyusun kumpulan data menggunakan bubble sort.

Apabila keseluruhan kode digabung dan dijalankan, program akan menghasilkan output seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35103?from=35098#run3-editor1)

1

def bubble\_sort(numbers: list) -> list:

2

sorted\_numbers = numbers.copy()

3

n = len(sorted\_numbers)

4

​

5

for i in range(n):

6

​

7

for j in range(0, n - i - 1):

8

​

9

if sorted\_numbers[j] > sorted\_numbers[j + 1]:

10

sorted\_numbers[j], sorted\_numbers[j+1]= sorted\_numbers[j+1], sorted\_numbers[j]

11

​

12

return sorted\_numbers

13

​

14

# main program

15

numbers = [1,3,2,5,1,6,1,3,2,1]

16

​

17

sorted\_data\_bubble\_sort = bubble\_sort(numbers)

18

print("Data sebelum diurutkan:", numbers)

19

print("Data setelah diurutkan:", sorted\_data\_bubble\_sort)

20

​

 Input  Reset

 Jalankan

Secara kode, bubble sort memiliki perulangan bersarang yang mengakibatkan proses komputasi membutuhkan waktu cukup lama. Sama halnya dengan selection sort yang memiliki perulangan bersarang. Ia akan membandingkan data dengan elemen yang belum diurutkan sehingga menemukan nilai terendah.

Namun, secara stabilitas data, bubble sort unggul dibandingkan dengan selection sort. Bubble sort mampu mempertahankan letak urutan data apabila ada dua atau lebih nilai yang sama dalam satu kumpulan data. Dengan begitu, susunan data akan selalu terjaga walaupun ada data baru yang ditambahkan.

### Insertion Sort

Konsep algoritma insertion sort ini unik. Idenya adalah mencari tempat yang “tepat” untuk setiap elemen kumpulan data secara sekuensial. Ia membandingkan elemen satu dengan elemen sebelahnya untuk mencari nilai minimal. Apabila elemen tersebut tidak berada di tempat yang tepat, ia akan disisipkan ke elemen sebelumnya hingga tersusun secara berurutan.

Supaya Anda memiliki pandangan terkait algoritma insertion sort, perhatikan ilustrasi berikut.

Konsep ini sama seperti cara manusia ketika menyusun kartu yang tidak berurutan. Ketika ingin memindahkan kartu dari tempat yang salah, kita menyisipkannya ke tempat yang sesuai. Alhasil, susunan kartu menjadi berurutan.

Anda sudah bisa membayangkan bagaimana cara kerjanya? Selanjutnya, kita akan beralih memahami cara kerja algoritma insertion sort pada kode Python.

1. Definisikan kumpulan data homogen atau bertipe sejenis.
   1. numbers = [1,3,2,5,1,6,1,3,2,1]
2. Untuk membedakan antara kumpulan data sebelum dan sesudah diurutkan, kita bisa menduplikasi kumpulan data pada variabel baru. Gunakan method **copy()**untuk menduplikasi kumpulan data.
   1. sorted\_numbers = numbers.copy()
3. Hitung total elemen dari kumpulan data. Nilai ini akan dipakai untuk melakukan perulangan pada setiap elemen kumpulan data.
   1. n = len(sorted\_numbers)
4. Lakukan perulangan dengan state iuntuk mengulang setiap elemen kumpulan data menggunakan for-in loop. Kita bisa mulai state perulangan dari indeks ke-1 hingga akhir untuk mempermudah proses perbandingan dua nilai.
   1. Siapkan nilai state untuk menyimpan proses decrement data dan nilai temporer key untuk menyimpan elemen yang akan dipindahkan.
   2. Lakukan perulangan dengan state j untuk mengetahui elemen yang dibandingkan memenuhi syarat atau tidak. Apabila key lebih kecil dari data pembanding, nilai data pembanding akan berpindah pada indeks ke-j+1. Kemudian, lakukan decrement j pada akhir perulangan. Proses ini akan berlangsung hingga nilai key lebih besar dari data pembanding atau state j lebih besar sama dengan 0.
   3. Setelah perulangan state j berakhir, kita bisa memasukkan key ke dalam kumpulan data berindeks ke-j+1.
5. Voilà! Proses mengurutkan kumpulan data menggunakan insertion sort sudah selesai.

Saat ini, Anda dapat menjalankan keseluruhan kode insertion sort. Jadi, output yang dihasilkan oleh program tersebut akan seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35103?from=35098#run4-editor1)

1

def insertion\_sort(numbers: list) -> list:

2

sorted\_numbers = numbers.copy()

3

n = len(sorted\_numbers)

4

​

5

for i in range(1, n):

6

key = sorted\_numbers[i]

7

j = i - 1

8

​

9

while j >= 0 and key < sorted\_numbers[j]:

10

sorted\_numbers[j + 1] = sorted\_numbers[j]

11

j -= 1

12

13

sorted\_numbers[j + 1] = key

14

​

15

return sorted\_numbers

16

​

17

# main program

18

numbers = [1,3,2,5,1,6,1,3,2,1]

19

​

20

sorted\_data\_insertion\_sort = insertion\_sort(numbers)

 Input  Reset

 Jalankan

Secara keseluruhan kode, program insertion sort dan selection sort tidak jauh berbeda. Mereka memiliki perulangan bersarang yang mengakibatkan proses komputasi akan semakin lama.

Akan tetapi, proses komputasi pada insertion sort lebih cepat dibandingkan dengan selection sort. Mengapa demikian? Hal ini disebabkan karena algoritma hanya membandingkan data dengan elemen yang sudah diurutkan sampai menemukan posisi yang tepat. Sebaliknya, selection sort akan membandingkan data dengan elemen yang belum diurutkan untuk mencari elemen terkecil. Alhasil, komputasi pada selection sort semakin banyak.

Di sisi lain, insertion sort adalah salah satu algoritma yang sangat mudah dipahami. Mulai dari gambaran hingga kode pun sangat sederhana. Berdasarkan urutan data, insertion sort adalah algoritma yang stabil dengan mempertahankan letak urutan apabila ada dua atau lebih nilai serupa dalam satu kumpulan data. Dengan begitu, susunan data akan selalu terjaga walaupun ada data baru yang ditambahkan.

Materinya sangat insightful, bukan? Sampai saat ini, Anda telah belajar cara kerja hingga kode untuk mengurutkan suatu kumpulan data. Ada empat algoritma yang sudah Anda pelajari, yaitu counting sort, selection sort, bubble sort, dan insertion sort.

Namun, pembelajaran Anda tidak boleh putus sampai keempat algoritma itu saja. Masih banyak algoritma lainnya yang perlu Anda perdalam, mulai dari quick sort, merge sort, heap sort, dan lainnya. Dengan mempelajari berbagai algoritma pengurutan data, Anda dapat menyimpulkan algoritma yang cocok saat mendapati permasalahan tertentu.

Untuk mempertajam ilmu Anda terkait algoritma pengurutan data, silakan kunjungi tautan berikut.

* [Algorithms in a Nutshell: Sorting Algorithms](https://learning.oreilly.com/library/view/algorithms-in-a/9781491912973/ch04.html)
* [Beginning Algorithms: Basic Sorting and Advanced Sorting](https://learning.oreilly.com/library/view/beginning-algorithms/9780764596742/ch06.html)
* [Understanding Sorting Algorithms - John Long](https://medium.com/jl-codes/understanding-sorting-algorithms-af6222995c8)
* [Sorting Algorithms Demystified - Taniya Gupta](https://medium.com/swlh/sorting-algorithms-demystified-efc6f0a5bf8f)
* [Cracking the Code of Sorting and Searching: 11 Algorithms for Enhanced Efficiency - Armin Norouzi](https://levelup.gitconnected.com/cracking-the-code-of-sorting-and-searching-11-algorithms-for-enhanced-efficiency-data-structure-466ad0bf4509)