# BERPIKIR KOMPUTASIONAL

# SEARCHING
# SORTING
# STACK DAN QUEUE

# SORTING (PENGURUTAN)

# Definisi Sorting

- Suatu Proses Penyusunan Kembali kumpulan objek menggunakan tata aturan tertentu.
- Proses pengurutan data baik dari nilai tertinggi maupun dari nilai terendah.

# SORTING

- **Pengurutan** (*sorting*) sangat penting saat memanipulasi data. Ketika kita mengurutkan data, data akan terlihat rapi dan mudah dibaca. Sehingga memudahkan juga dalam menganalisa.

# Teknik Sorting

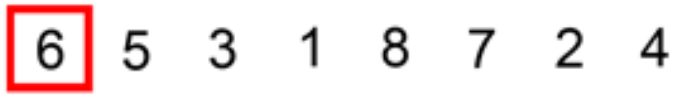
- INSERTION SORT
- MERGE SORT
- QUICK SORT
- SELECTION SORT
- BUBBLE SORT

# Insertion Sort

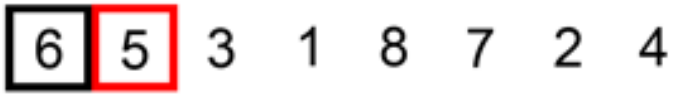
- **Insertion Sort** adalah algoritma yang melakukan pengurutan dengan membandingkan elemen satu dengan elemen lainnya dalam sebuah *list*. Elemen yang dibandingkan akan ditempatkan ke posisi yang sesuai (urut) pada *list*.
- Analoginya seperti mengurutkan kumpulan kartu. Setiap kartu yang kalian ambil, kalian bandingkan terlebih dahulu ke kumpulan kartu yang sudah diurutkan. Dan ketika tahu urutan ke berapa, kalian selipkan kartu itu ke tumpukan kartu agar urut.

Jika digambarkan secara singkat, maka algoritma Insertion sort ini dapat digambar sebagai berikut.

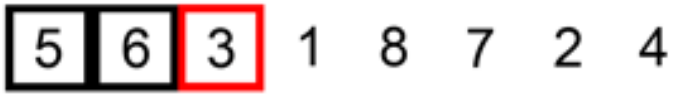

› angka diatas adalah angka yang akan kita urutkan dengan inserting sort.




› langkah pertama yang kita lakukan adalah ambil elemen pertama dari angka tersebut yaitu angka 6. karena angka 6 adalah angka pada elemen pertama maka tidak ada pertukaran. perhatikan gambar diatas.



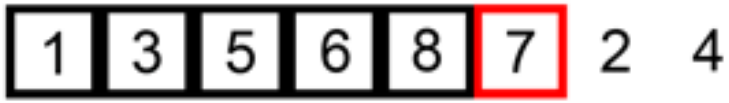

› kemudian lanjut ke element ke 2 yaitu angka 5. cek apakah angka lima lebih kecil dari angka sebelumnya (angka kiri) jika iya maka geser angka sebelumnya ke kanan. perhatikan gambar dibawah ini angka 5 berpindah ke element pertama.



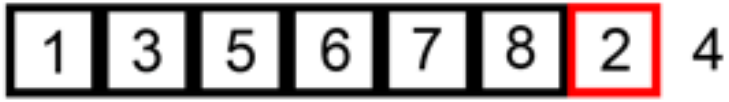

› lanjut ke element ke 3 yaitu angka 3. cek apakah angka 3 lebih kecil dari angka sebelumnya ? jika iya maka geser angka sebelumnya ke kanan.


› lanjut ke element ke 4 yaitu angka 1. cek apakah angka 1 lebih kecil dari angka sebelumnya? jika iya maka geser angka yang lebih besar darinya ke kanan.

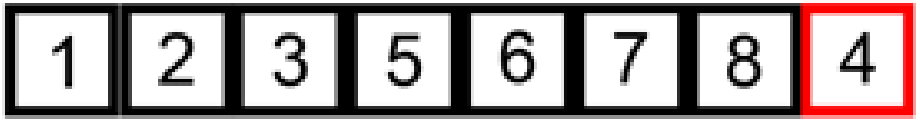

› lanjut ke element ke 5 yaitu angka 8. karena angka 8 tidak lebih kecil dari angka sebelumnya maka angka 8 tetap diposisi element ke 5.




› lanjut ke elemen ke 6 yaitu angka 7. cek apakah angka 7 lebih kecil dari sebelumnya. jika iya maka geser angka yang lebih besar darinya ke kanan.

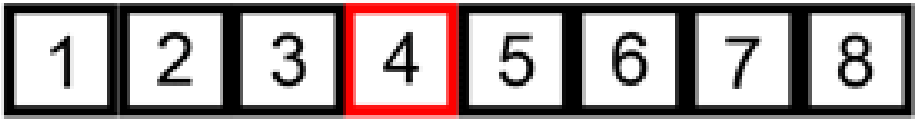



› lanjut ke element ke 8 yaitu angka 2. cek apakah angka 2 lebih kecil dari angka sebelumnya? jika iya geser angka yang lebih besar ke kanan.




› lanjut ke element ke 9 yaitu angka 4. cek apakah angka 4 lebih kecil dari angka sebelumnya? jika iya geser angka yang lebih besar dari angka 4 ke kanan.

› hasil akhir dari sorting di atas adalah sebagai berikut :



# Algoritma Insertion Sort

```python
def insertion_sort(array):
    # perulangan pertama
    for i in range(1, len(array)):
        # ini elemen yang akan kita posisikan
        key_item = array[i]
        # kunci index posisi
        j = i - 1
        # lakukan perulangan kedua
        while j >= 0 and array[j] > key_item:
            # menggeser elemen yang lain
            array[j + 1] = array[j]
            j -= 1
        # memposisikan elemen
        array[j + 1] = key_item

    return array
```

```python
unordered = [91, 21, 37, 77, 82]
print(insertion_sort(unordered))
```

Outputnya akan seperti ini :

```
[21, 37, 77, 82, 91]
```

# Alur Coding

- Kita mulai dari perulangan dari elemen kedua (1) sampai elemen terakhir. Kenapa tidak dari elemen pertama? karena elemen pertama adalah elemen yang dibandingkan oleh elemen yang lain.
- Variabel key_item, adalah tempat untuk menampung elemen yang akan diposisikan.
- Sedangkan variabel j dengan nilai i - 1 adalah tujuan index yang nantinya elemen key_item akan ditempatkan.
- Menggunakan perulangan while yang memiliki kondisi selama j >= 0 dan elemen index j lebih besar dari key_item, Maka elemen yang lain akan digeser dan index j diperbarui.

```
while j >= 0 and array[j] > key_item:
    array[j + 1] = array[j]
    j -= 1
```

- Setelah tidak ada lagi elemen yang lebih besar dari key_item, maka perulangan while akan berhenti.
- Lalu tempatkan elemen pada key_item ke index j + 1. Kenapa j + 1? karena sebelumnya kita memberikan nilai i - 1 yang seharusnya tidak sesuai.

# Merge Sort

- Merge sort atau pengurutan secara menggabungkan suatu data.

- Secara garis besar, suatu deret data yang akan diurutkan akan di bagi-bagi terlebih dahulu lalu dibandingkan masing-masing data dari masing-masing bagian lalu baru digabungkan kembali

# Kelebihan Merge Sort

- Dibanding dengan algoritma lain, merge sort ini termasuk algoritma yang sangat efisien dalam penggunaannya sebab setiap list selalu dibagi bagi menjadi list yang lebih kecil, kemudian digabungkan lagi sehingga tidak perlu melakukan banyak perbandingan.
- Cocok untuk sorting akses datanya lambat misalnya tape drive atau hard disk.
- Cocok untuk sorting data yang biasanya diakses secara sequentially (berurutan),
- misalnya linked list, tape drive, dan hard disk.

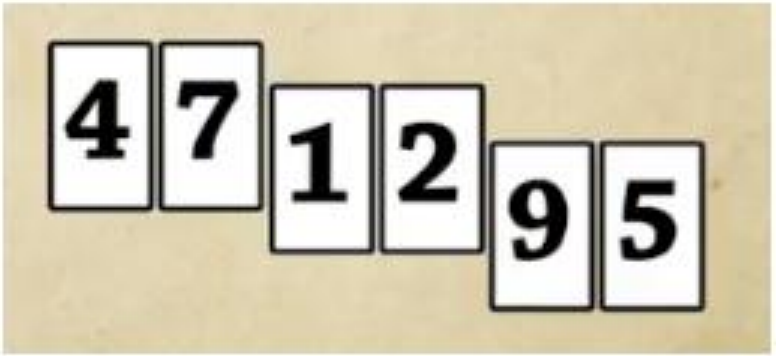
# Kekurangan Merge Sort

- Kekurangan Merge Sort yaitu terlalu banyak menggunakan ruang pada memori.
- Merge Sort membutuhkan lebih banyak ruang daripada jenis sorting lainnya.

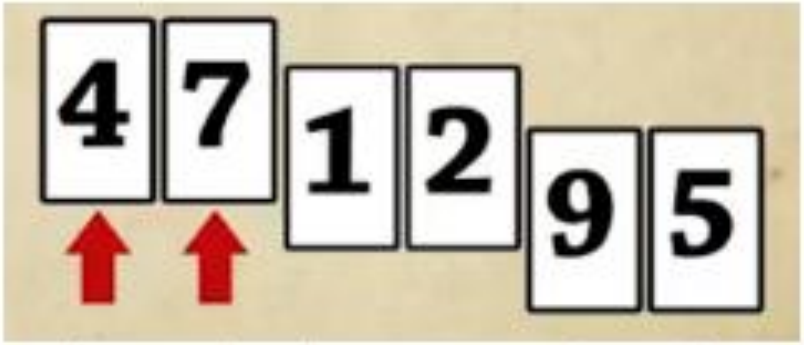
# Contoh Penerapan Merge Sort.

Data sumber yang akan diurutkan adalah sebagai berikut:

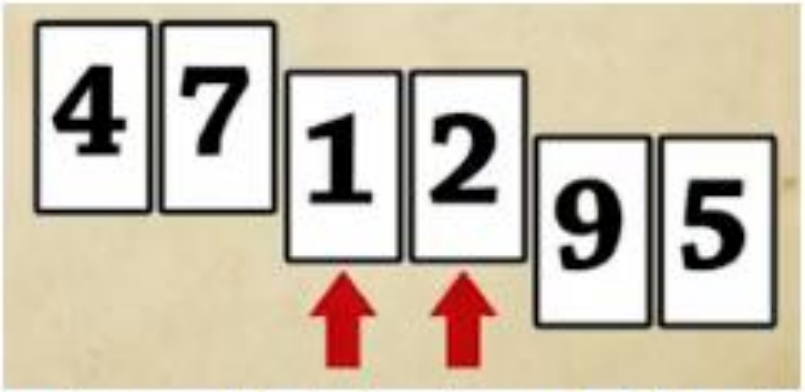

langkah pertama kita bagi menjadi 3 bagian susunan elemen data yaitu 4 dengan 7, 1 dengan 2 dan 9 dengan 5.



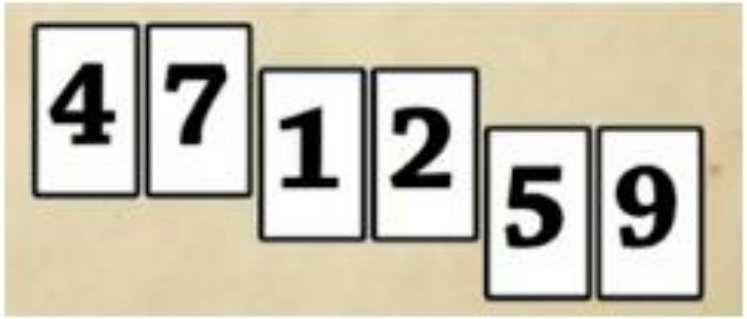

kemudian kita bandingan elemen pertama yaitu 4 dan 7. karena 4 sudah lebih kecil dari 7 maka posisi tetap.



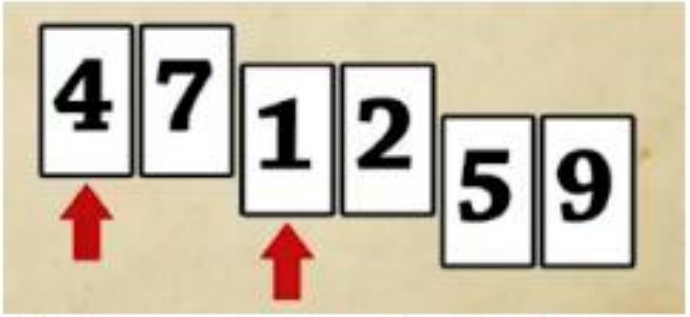

kemudian bandingkan elemen data ke 2 yaitu angka 1 dan 2. karena angka 1 lebih kecil dari 2 maka posisi tetap.



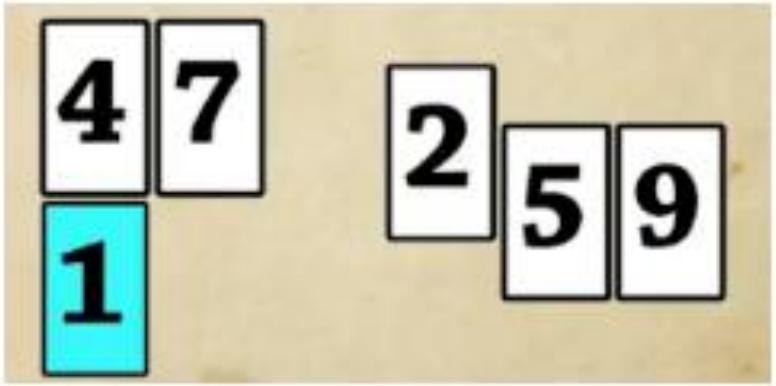

kemudian bandingkan elemen data ke 3 yaitu angka 9 dan 5. karena angka 5 lebih kecil dari 9 maka posisi 9 ditukar dengan angka 5.



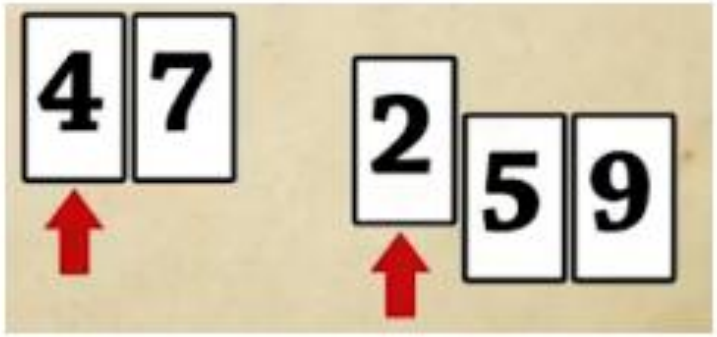

langkah selanjutnya adalah kita bandingkan angka pertama pada elemen pertama dengan elemen 2 dan elemen 3 pertama kita bandingkan angka 4 dengan element 2 yaitu 1.



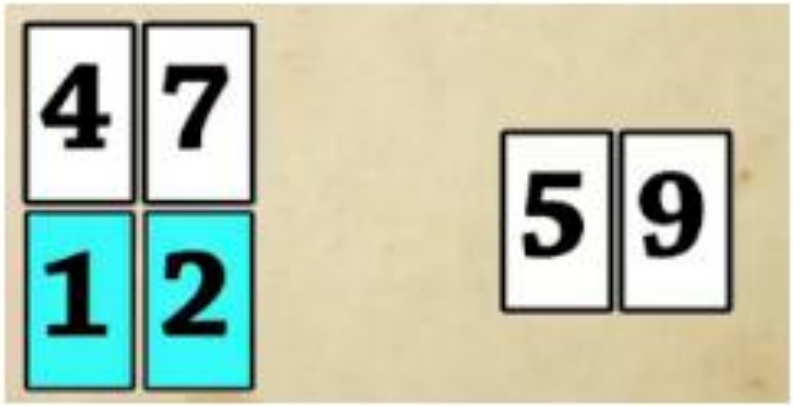

cek angka 4 dan 1. lebh besar mana angka 4 dibandingkan dengan angka 1. karena angka 1 lebih kecil dari angka 4 maka angka 1 akan menempati posisi baru di elemen pertama.



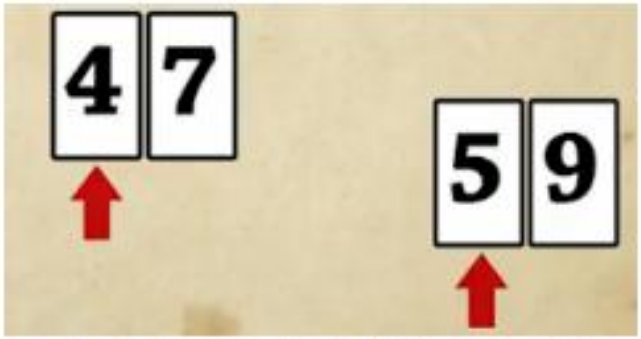

kemudian bandingkan angka 4 dengan 2. cek lebih kecil mana antara angka 4 dan 2?



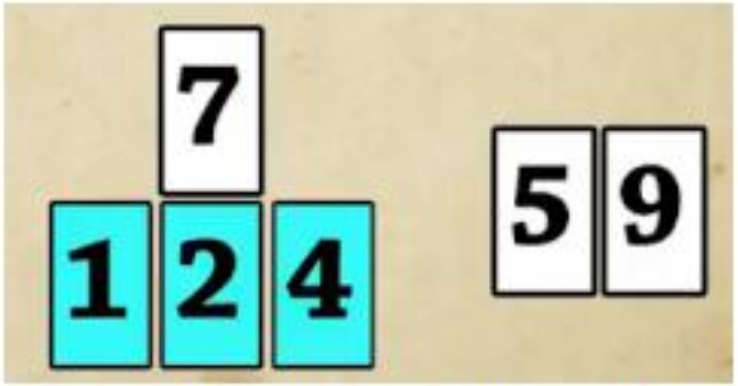

karena angka 2 lebih kecil dari 4 maka angka 2 ke elemen baru di posisi ke 2



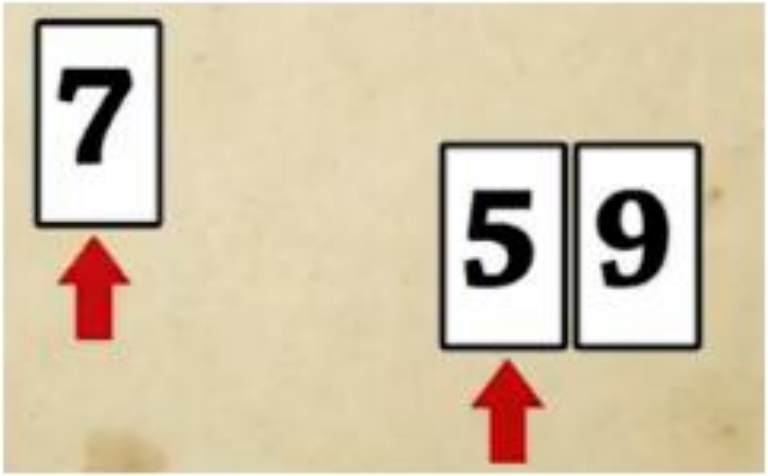

kemudian bandingkan angka 4 dengan elemen ke 3 yaitu 5.



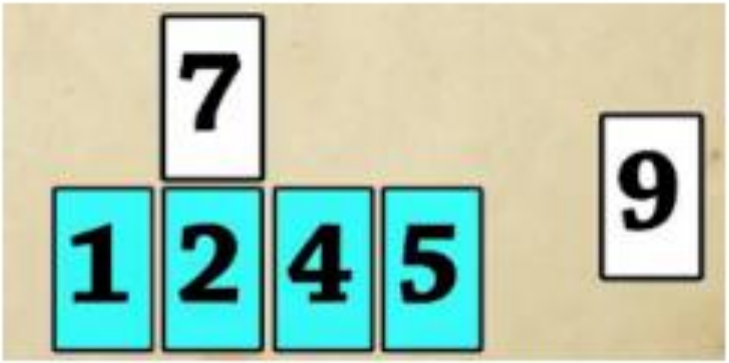

cek lebih kecil mana angka 4 dengan angka 5. karena angka 4 lebih kecil dari 5 maka angka 4 menempati posisi ke 3.




lanjut ke urutan angka ke 2 pada elemen pertama yaitu angka 7. bandingkan angka 7 dan 5. lebih kecil mana antara angka 7 dan 5.




karena angka 5 lebih kecil dari 7 maka angka 5 menempati posisi ke 4.




lanjut ... bandingkan angka 7 dengan angka 9. lebih kecil mana antara angka 7 dan 9?


karena angka 7 lebih kecil dari 9 maka angka 7 menempati urutan ke 5.


karena angka 9 adalah angka terakhir maka angka 9 menempati posisi ke 6


pengurutan dengan merge telah selesai.

# QUICK SORT

- Quick Sort merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort.

- Algoritma ini mengambil salah satu elemen secara acak (biasanya dari tengah) yang disebut dengan pivot lalu menyimpan semua elemen yang lebih kecil di sebelah kiri pivot dan semua elemen yang lebih besar di sebelah kanan pivot. Hal ini dilakukan secara rekursif terhadap elemen di sebelah kiri dan kanannya sampai semua elemen sudah terurut.

# KEUNGGULAN QUICK SORT

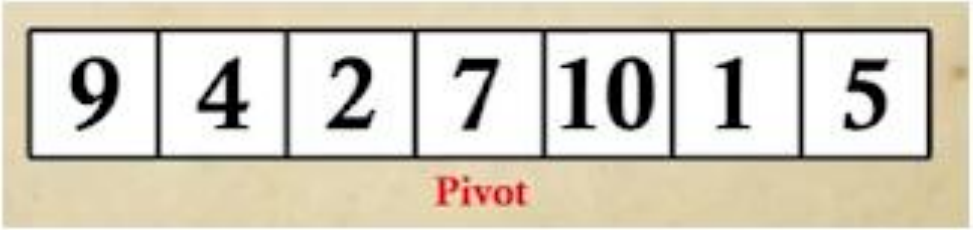
- Secara umum memiliki kompleksitas O(n log n).
- Algoritmanya sederhana dan mudah diterapkan pada berbagai bahasa pemrograman dan arsitektur mesin secara efisien.
- Dalam prakteknya adalah yang tercepat dari berbagai algoritma pengurutan dengan perbandingan, seperti mergesort dan heapsort.
- Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.
- Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

# KEKURANGAN QUICK SHORT

- Sedikit kesalahan dalam penulisan program membuatnya bekerja tidak beraturan (hasilnya tidak benar atau tidak pernah selesai).
- Memiliki ketergantungan terhadap data yang dimasukkan, yang dalam kasus terburuk memiliki kompleksitas O(n2).
- Secara umum bersifat tidak stable, yaitu mengubah urutan input dalam hasil akhirnya (dalam hal inputnya bernilai sama).
- Pada penerapan secara rekursif (memanggil dirinya sendiri) bila terjadi kasus terburuk dapat menghabiskan stack dan memacetkan program.
- Pada bahasa pemrograman, quicksort ada dalam pustaka stdlib.h untuk bahasa C, dan class TList dan TStringList dalam Delphi (Object Pascal) maupun FreePascal.
-

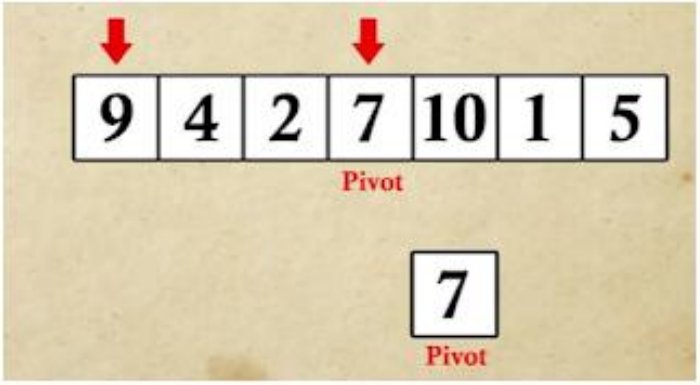
# Contoh Ilustrasi Pengurutan Quick Sort

› langkah pertama adalah tentukan pivotnya. dalam hal ini adalah saya memilih angka 7

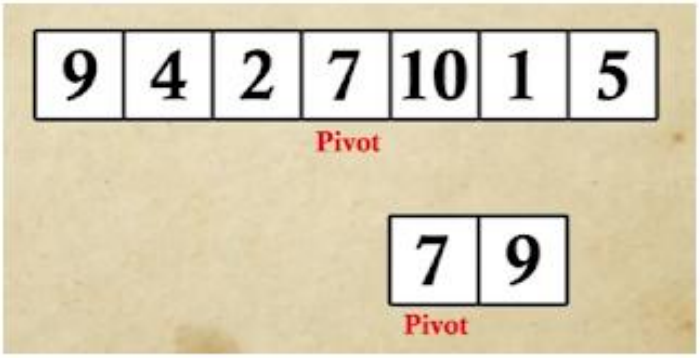



› kemudian buatpartisi buat masing2 angka sebelah kanan dan kiri

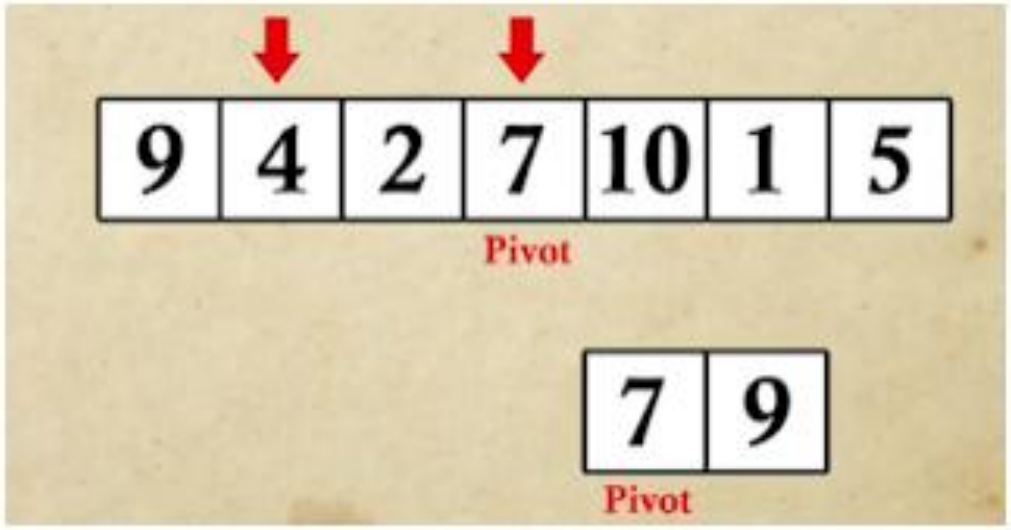
› kemudian gunakan algoritma quicksort yang ada diatas. jika angka lebih kecil dari pivot maka akan diletakan sebelah kiri dan jika lebih besar maka letakan disebelah kanan. langkah pertama adalah bandingkan angka 9 dengan pivot apakah lebih kecil atau lebih besar.



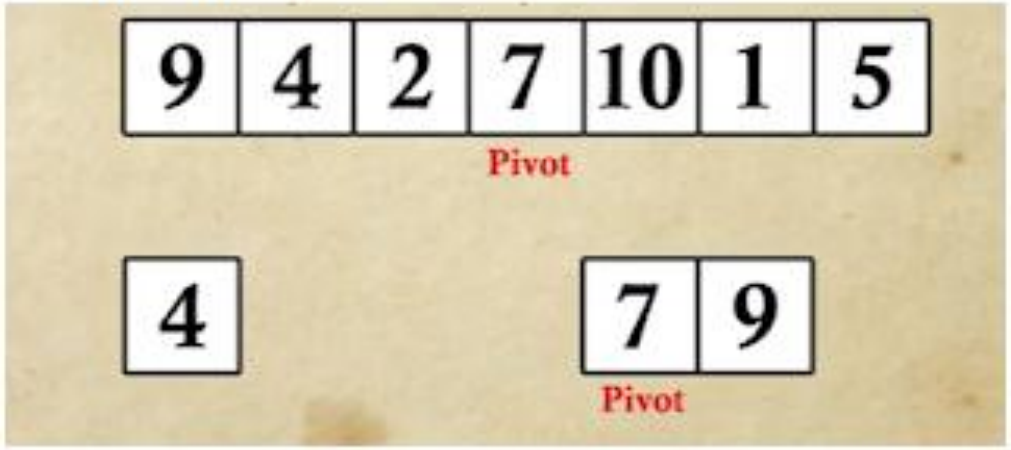

› karena angka 9 lebih besar maka letakan angka 9 setelah pivot.



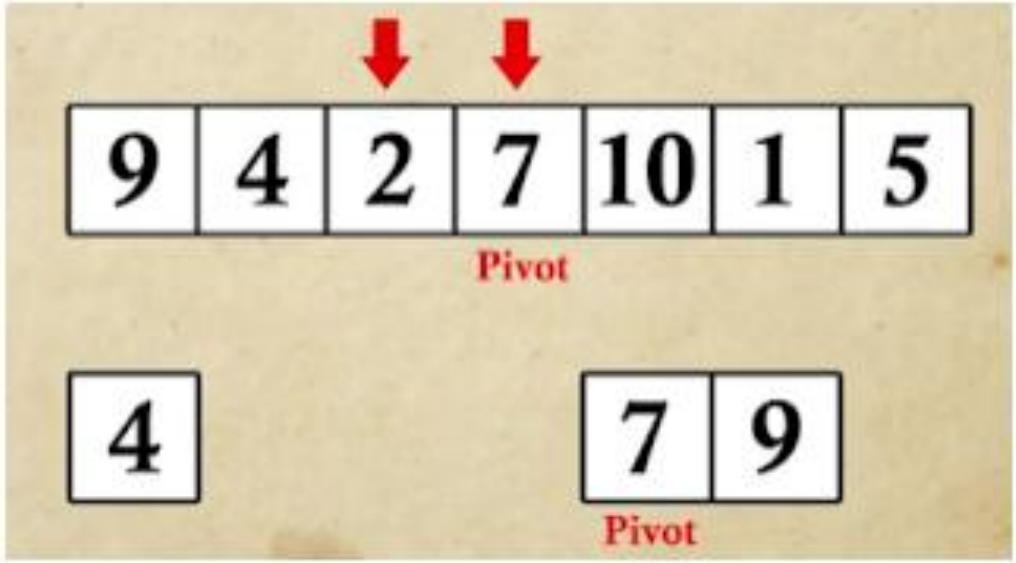
› lanjut ke angka 4. bandingkan angka 4 dengan pivot.



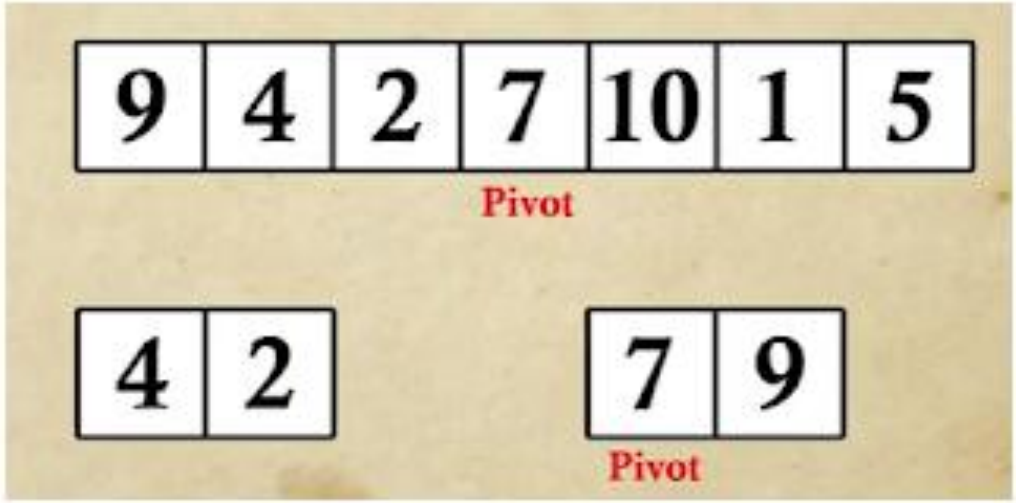

› karena angka 4 lebih kecil dari 7 maka posisi tetap.



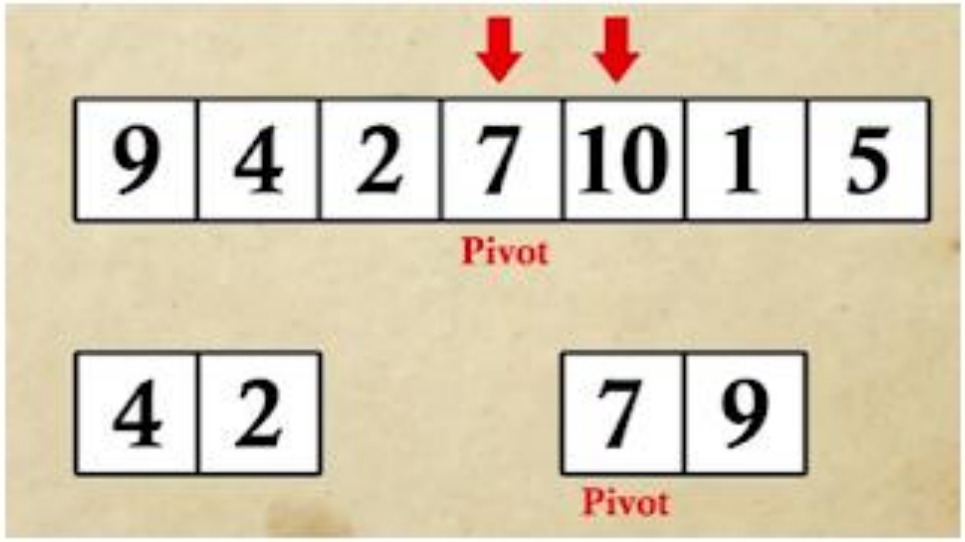
› lanjut ke angka 2. cek apakah angka 2 lebih kecil atau lebih besar dari pivot.




› karena angka 2 lebih kecil dari pivot maka letaknya tetap

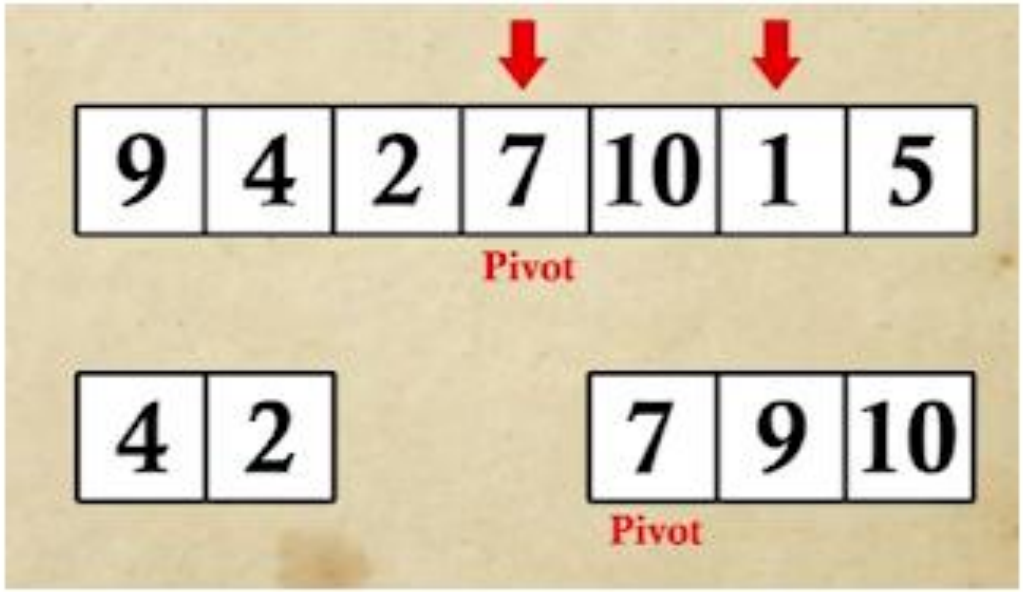


› bandingkan pivot dengan angka 10. cek angka 10 ebih besar atau lebih kecil dari pivot.

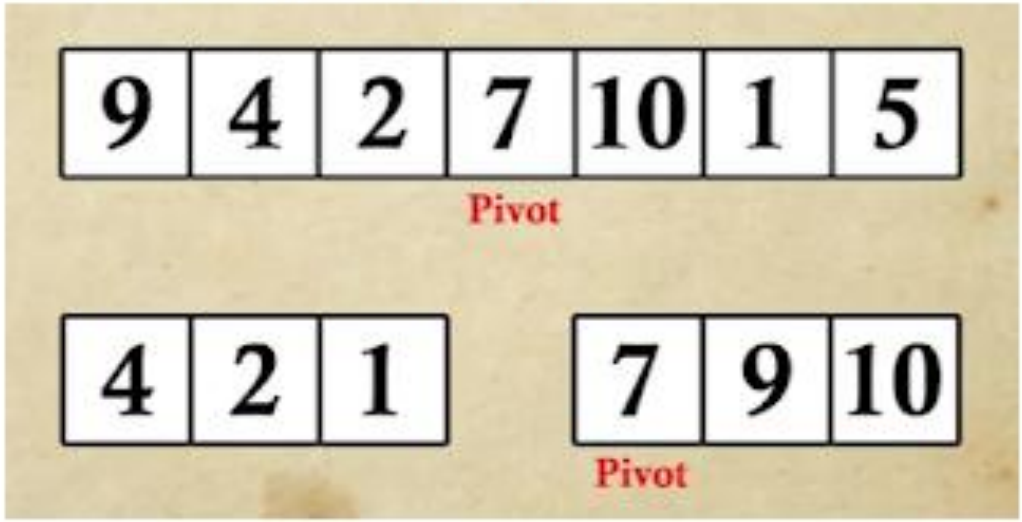



› karena angka 10 lebih besar maka posisi tetap sebelah kanan

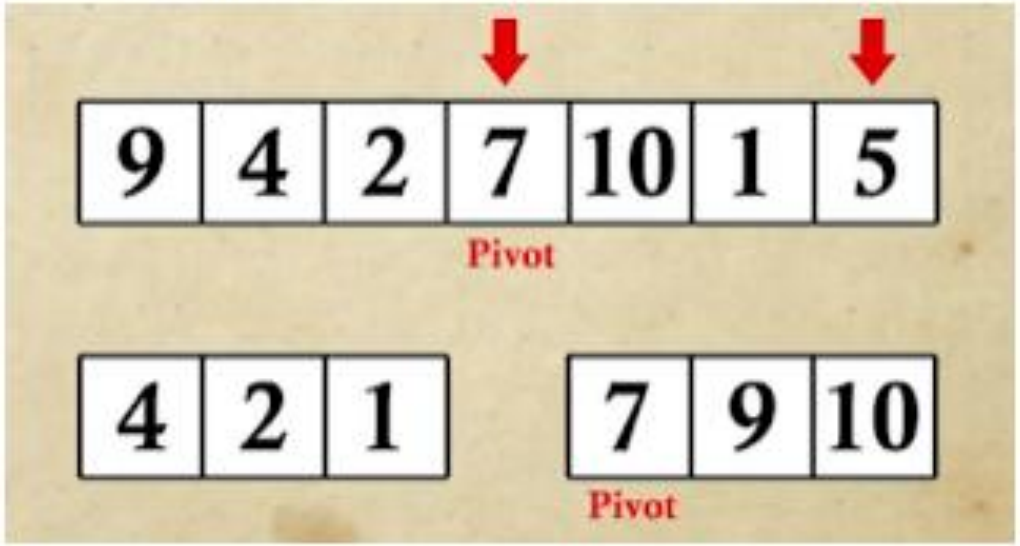
› lanjut ke angka 1. cek angka 1 lebih kecil atau lebih besar dari pivot.



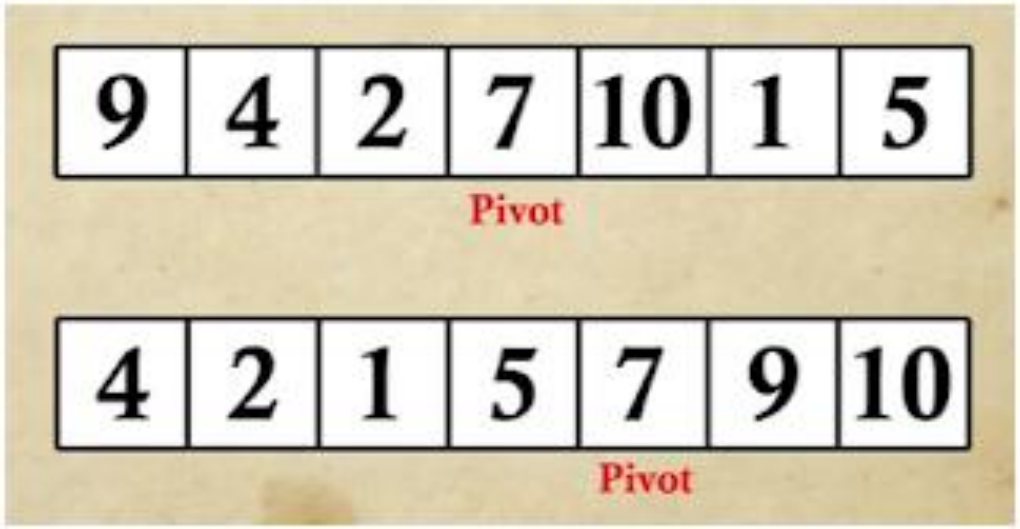

› karena lebih kecil maka pindah ke sebelah kiri pivot.



› lanjut ke angka 5. cek apakah angka 5 lebih kecil atau lebih besar dari angka pivot.

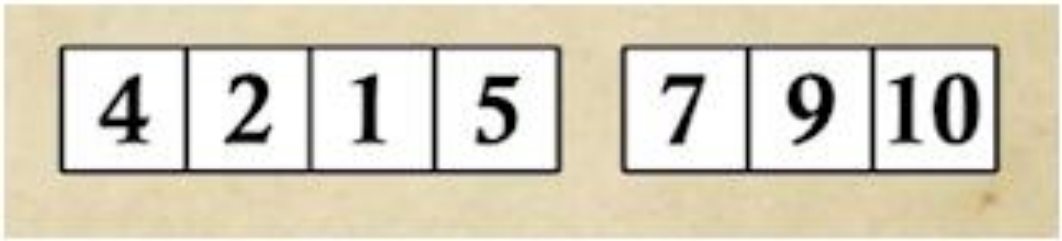


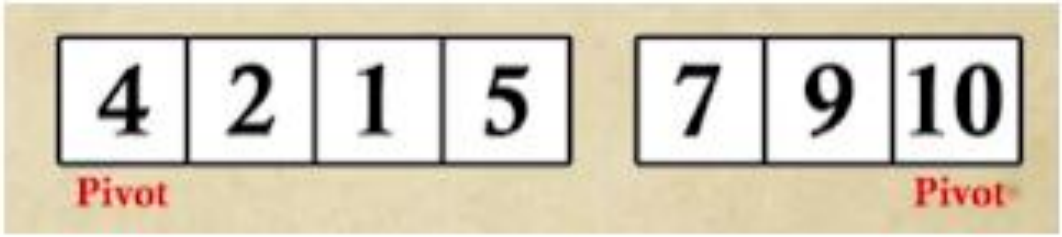

› karena angka 5 lebih kecil maka pindah ke sebelah kiri pivot.



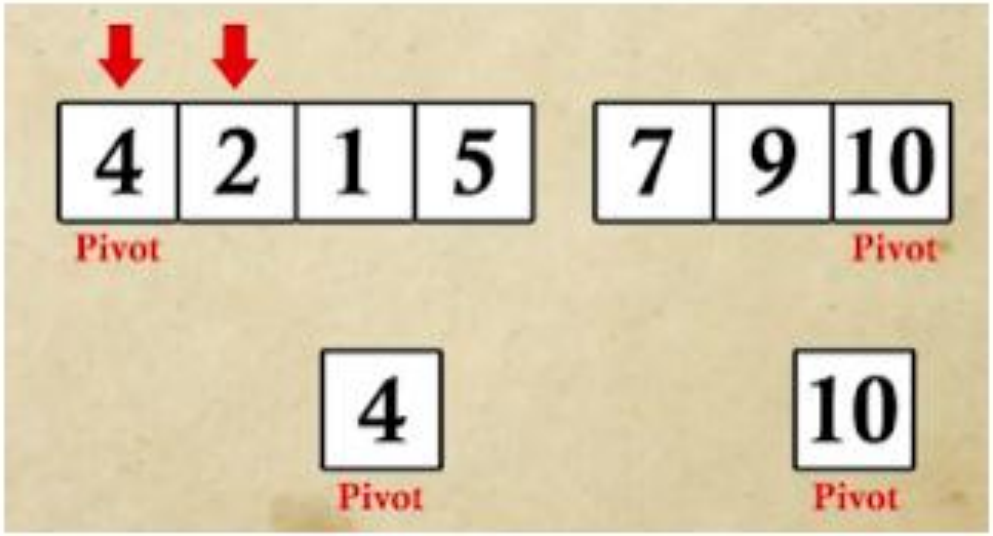
› setelah itu masuk ke dalam partisi baru. sampai sini proses belum selesai.


› tentukan pivot untuk masing-masing partisi

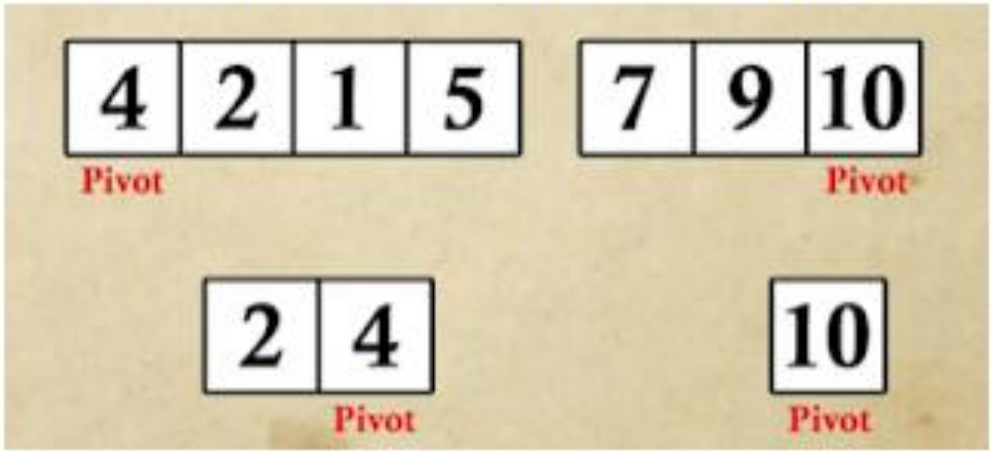


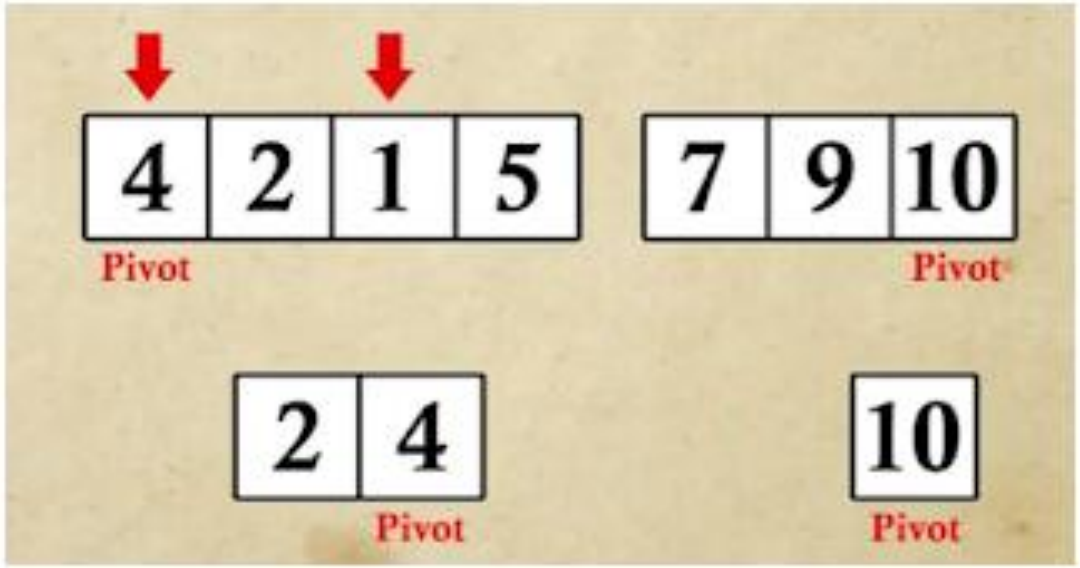
› perbandingan untuk pivot pertama. angka 2. cek apakah angka 2 lebih kecil atau lebih besar dari pivot.



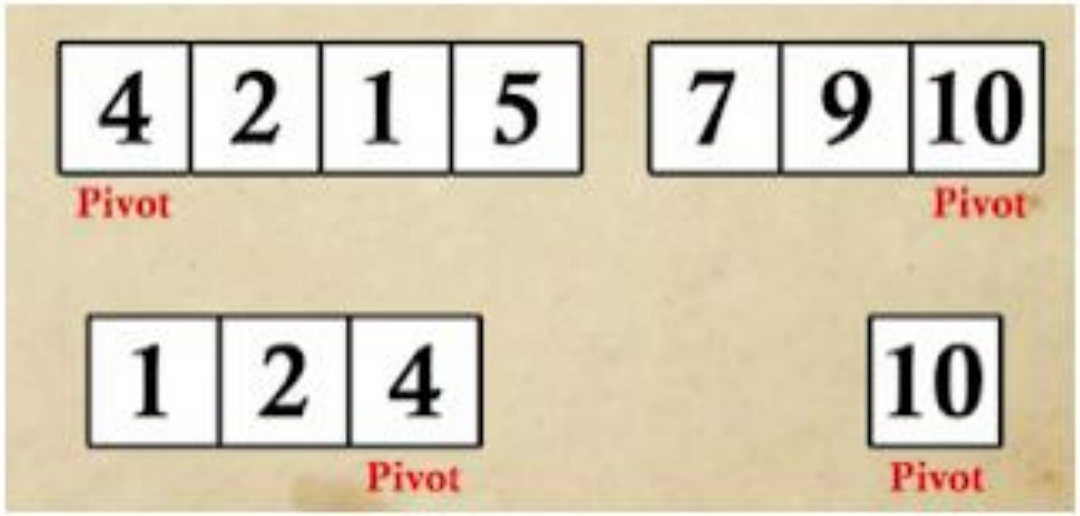

› karena angka 2 lebih kecil dari pivot maka pindahkan ke kiri pivot.



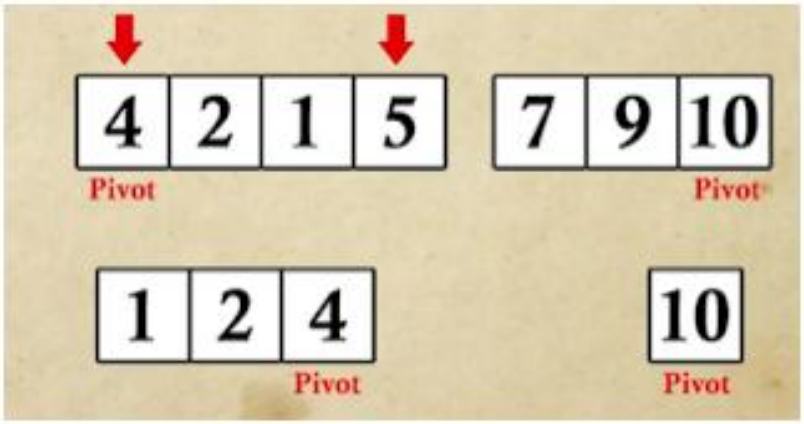
› lanjut ke angka 1. cek apakah angka lebih besar atau lebih kecil dari pivot.



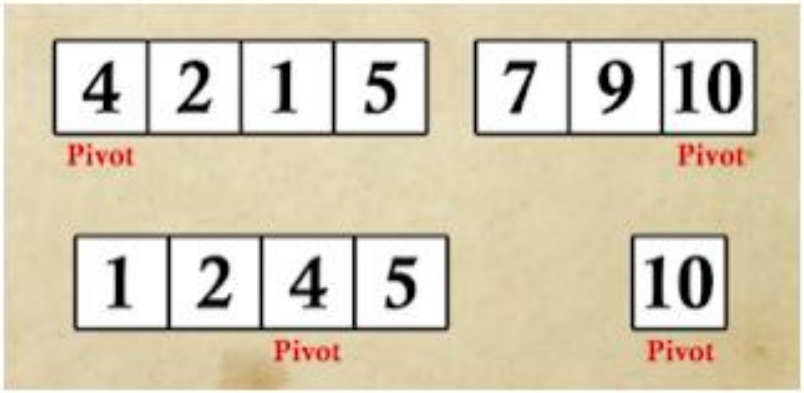

› karena angka 1 lebih kecil maka pindahkan disebelah kiri angka pivot.



› lanjut ke angka 5. cek apakah angkanya lebih besar atau lebih kecil dari pivot.




› karena lebih besar maka posisinya tetap. untuk partisi pertama selesai




› ulangi langkah2 seperti sebelumnya untuk pivot partisi ke 2. dan hasil akhir dari quick sort ini adalah seperti ini :

# Bubble sort

- Algoritma bubble sort termasuk ke dalam kategori algoritma comparison sort, karena menggunakan perbandingan pada operasi antar elemen nya.

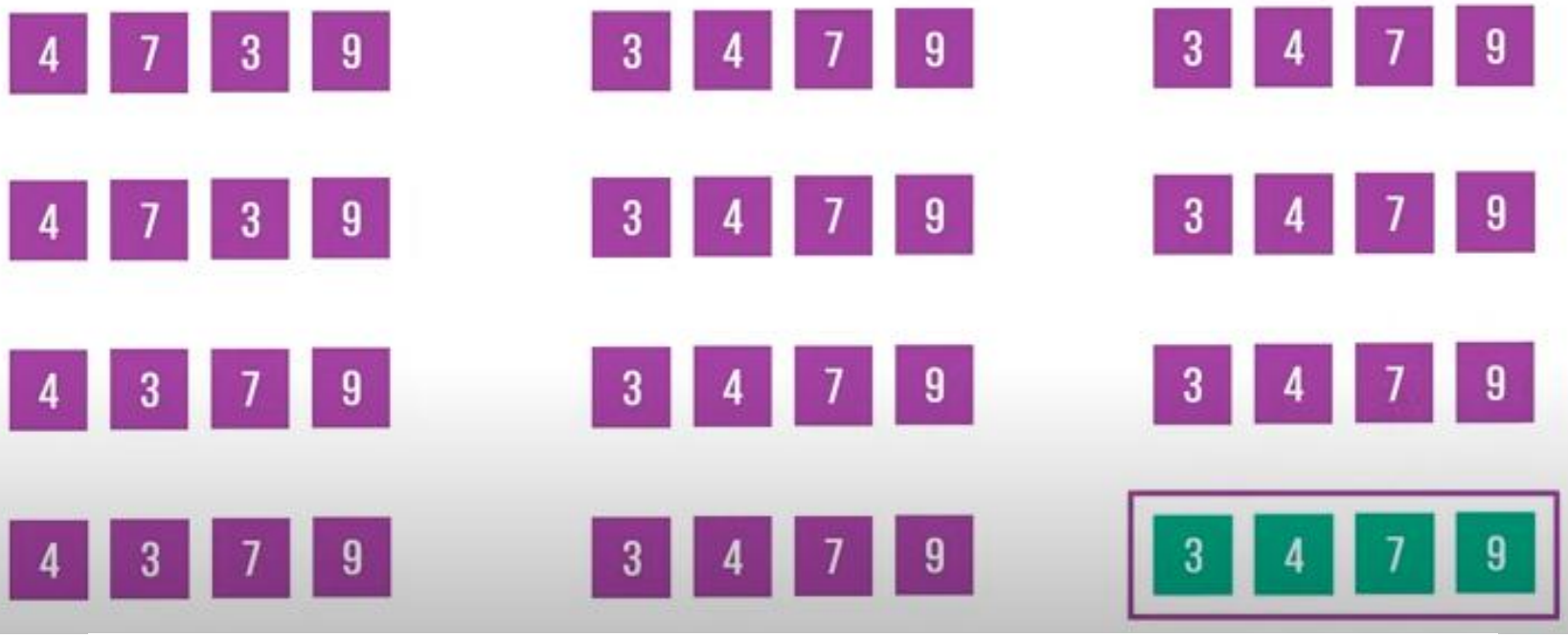
- Proses pengurutan pada algoritma ini dengan membandingkan masing - masing elemen secara berpasangan dan akan terus diulang sampai elemen terakhir atau sampai tidak ada lagi elemen yang dapat ditukar.

# Analogi algoritma bubble sort :

- Bandingkan nilai pada data ke satu dengan data ke dua
- Apabila nilai data ke satu lebih besar dari data ke dua maka tukar posisinya
- Kemudian data yang lebih besar tersebut dibandingkan lagi dengan data ketiga
- Apabila data ke tiga lebih kecil dari data ke dua maka tukar posisinya
- Dan begitu seterusnya hingga semua data yang ada menjadi terurut

# CARA KERJA BUBBLE SORT

ASCENDING




Banyak Perbandingan

= ( jumlahData – 1 ) x ( jumlahData – 1 )

```python
def bubble_sort(array):
    n = len(array) # jumlah list
    # perulangan pertama
    for i in range(n):
        # perulangan kedua
        for j in range(n - i - 1):
            # bandingkan masing" elemen
            if array[j] > array[j + 1]:
                # jika lebih besar, tukar.
                array[j], array[j + 1] = array[j + 1], array[j]
    return array
```

```python
unordered = [5, 3, 4, 8, 1, 2, 9, 6]
print(bubble_sort(unordered))
```

Outputnya seperti ini :

```
[1, 2, 3, 4, 5, 6, 8, 9]
```

# Alur kode

1. Kita hitung jumlah *list* menggunakan `len(array)` sebagai parameter perulangan.
2. Buat dua perulangan untuk membandingkan elemen pada *list*.
3. Bandingkan elemen pertama dengan elemen kedua menggunakan `if`.
4. Jika elemen pertama lebih besar daripada elemen kedua maka tukar posisinya.

```
if array[j] > array[j + 1]:
    array[j], array[j + 1] = array[j + 1], array[j]
```

5. Jika elemen kedua lebih besar dari pada elemen pertama maka biarkan saja.
6. Langkah 3 - 5 diulang sampai elemen terakhir (atau perulangan selesai).

Lalu, untuk mengurutkan secara ***descending*** (dari terbesar ke terkecil) bagaimana? Mudah, tinggal kita ubah saja perbandingannya ( `if` ).

```
# descending
if array[j] < array[j + 1]:
    ...
```

```
[9, 8, 6, 5, 4, 3, 2, 1]
```

# Selection Sort

- **Selection Sort** adalah algoritma *sorting* yang mengurutkan data dengan cara mencari elemen paling kecil dari *list*, lalu menukar elemen tersebut ke urutan paling awal.

- Dalam algoritma ini memiliki konsep yang sama dengan *bubble sort*,
  yaitu **membandingkan** dan **menukar**. Tetapi, dalam selection sort ia akan
  mencari index dengan elemen paling kecil, ketika sudah ketemu, elemen pada index itu akan ditukar dengan elemen pada index pertama.

# Algoritma selection sort

```
def selection_sort(arr):
    n = len(arr)
    # perulangan list elemen
    for i in range(n):
        # cari nilai elemen terendah
        # yang masih tersedia
        min_idx = i
        for j in range(i+1, n):
            if arr[min_idx] > arr[j]:
                min_idx = j

        # tukar dengan nilai elemen ke-i
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

```
listku = [64, 25, 12, 22, 11]
print(selection_sort(listku))
```

```
[11, 12, 22, 25, 64]
```

# Alur Kode :

- Kita cari dulu jumlah elemen pada *list* dengan len(arr).
- Lalu lakukan perulangan pertama yang didalamnya terdapat kode untuk mencari nilai minimium dan menukar nilai.
- Jika kalian perhatikan terdapat min_idx yang berperan untuk menampung index dengan nilai terendah.
- Lalu pada perulangan kedua, setiap elemen akan terus dibandingkan menggunakan if untuk mendapatkan index dengan nilai terkecil.

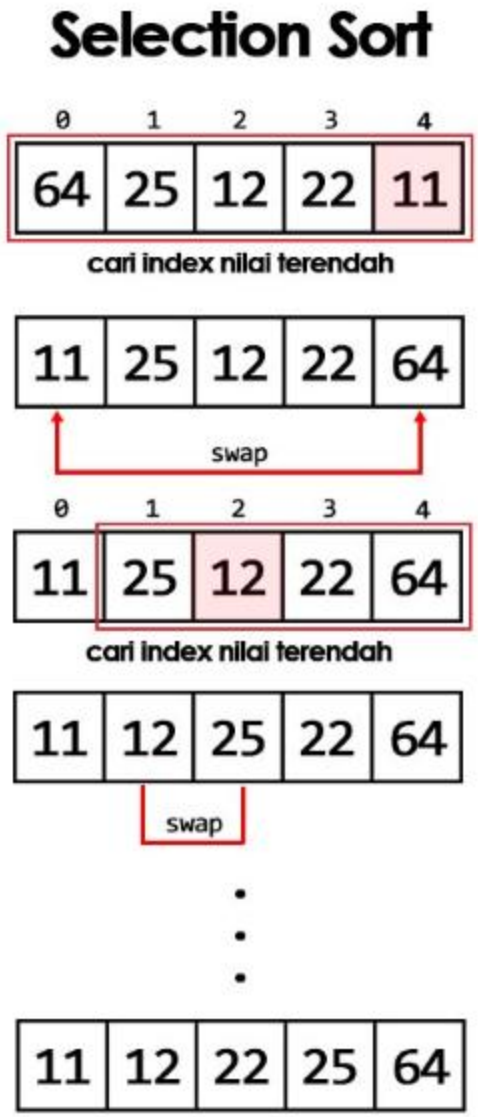
```
for j in range(i+1, n):
    if arr[min_idx] > arr[j]:
        min_idx = j
```

- Pada perulangan kedua, semua elemen setelah elemen ke-i atau i+1, saling dibandingkan untuk mencari nilai terkecil.
- Setelah menemukan elemen dengan nilai terkecil, index tersebut ditukar dengan nilai ke-i.

```
arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

- Hal tersebut diulang sampai perulangan pertama selesai (atau tidak elemen tuk diulang).

# Ilustrasi gamba selection sort



# ASCENDING