# Algoritma Pengurutan

1. Seleksi/Selection
2. Gelembung/Bubble
3. Penyisipan/Insertion
4. Merge
5. Quick

# Metode Penyisipan

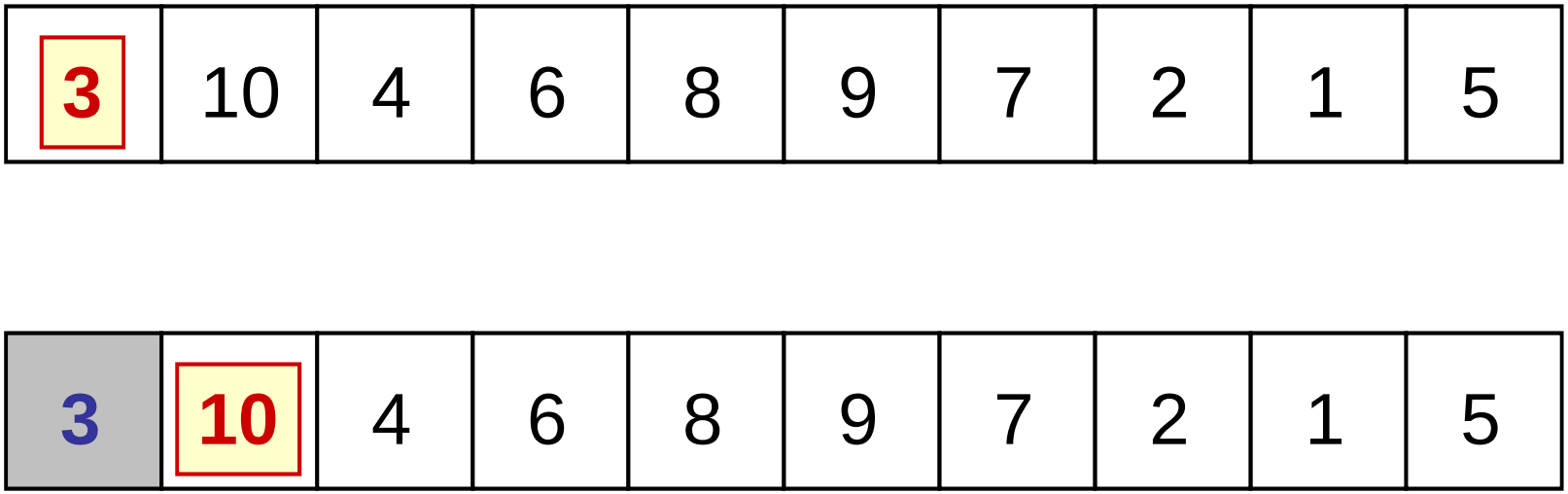
- Insertion sort menjadikan bagian kiri dari array terurut sampai keseluruhan dari array terurut.

- Algoritma ini diawali dengan membaca nilai-nilai di sebelah kanan dan membandingkan nilai-nilai di sebelah kiri, setelah ketemu tempatnya maka nilai yang dibaca ini disisipkan.

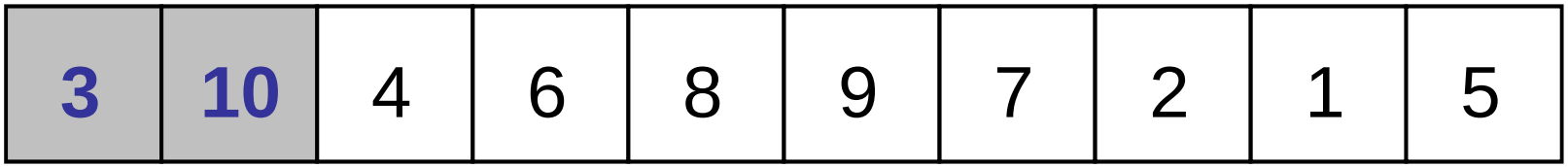
# Metode Penyisipan

| 3 | 10 | 4 | 6 | 8 | 9 | 7 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|

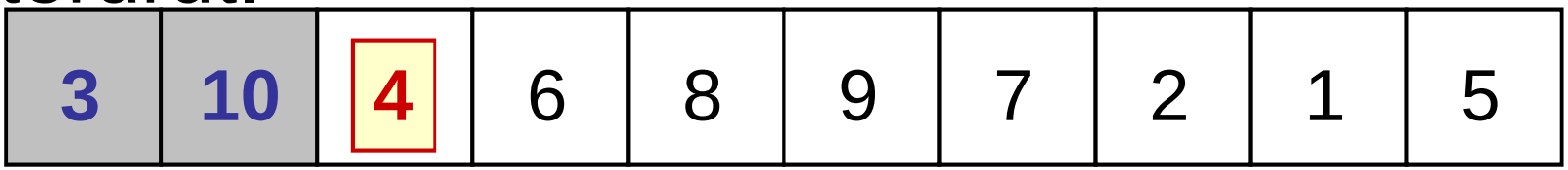
Nilai paling kiri(3) dapat dikatakan diurutkan dengan dirinya sendiri. Sehingga, kita tidak butuh untuk melakukan sesuatu terhadap nilai ini.




Cek dan lihat nilai kedua(10) apakah lebih kecil dari nilai pertama(3). Jika ya, tukar kedua nilai tersebut. Akan tetapi dalam hal ini kita tidak menukarnya.

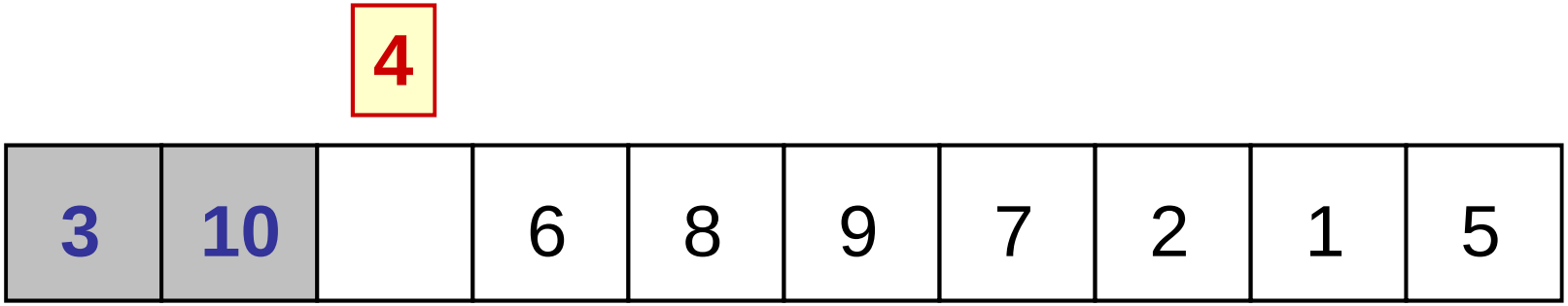



Angka biru/wilayah abu-abu (dua nilai pertama) secara relatif sudah dalam kondisi terurut.

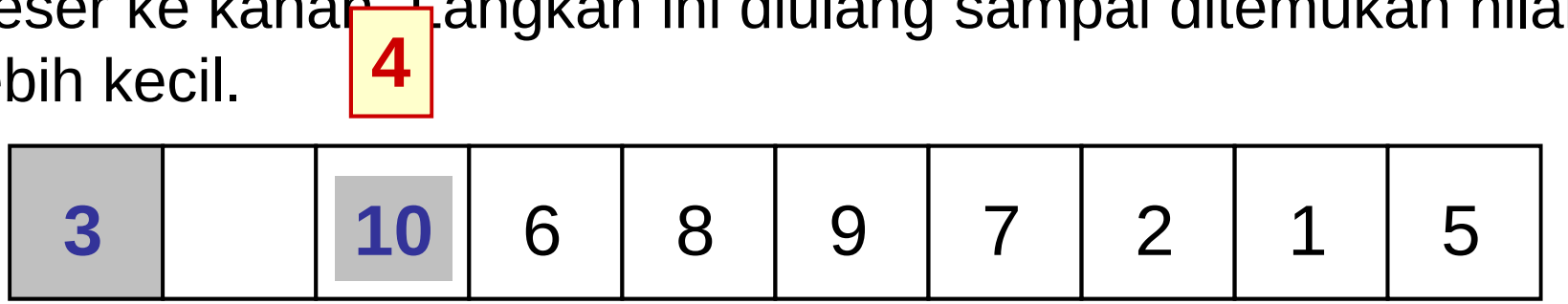



Selanjutnya, kita butuh menyisipkan nilai ketiga(4) dalam daerah abu-abu agar setelah penyisipan, daerah abu-abu tetap secara relatif dalam kondisi terurut.

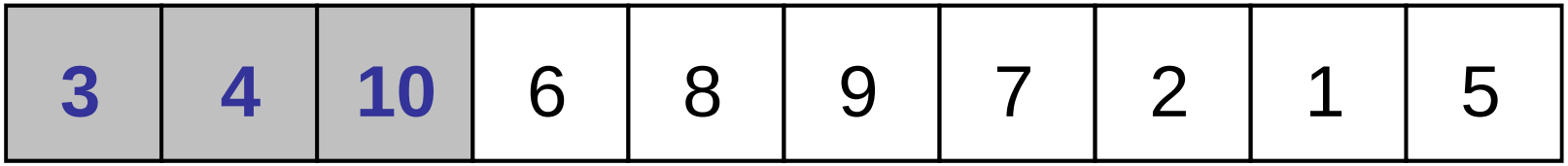
Pertama : Simpan nilai ketiga dalam sebuah variabel.




Kedua : Lihat nilai di sebelah kirinya (10), jika lebih besar geser ke kanan. Langkah ini diulang sampai ditemukan nilai lebih kecil.




Ketiga : Sisipkan nilai 4 pada posisi yang sesuai.



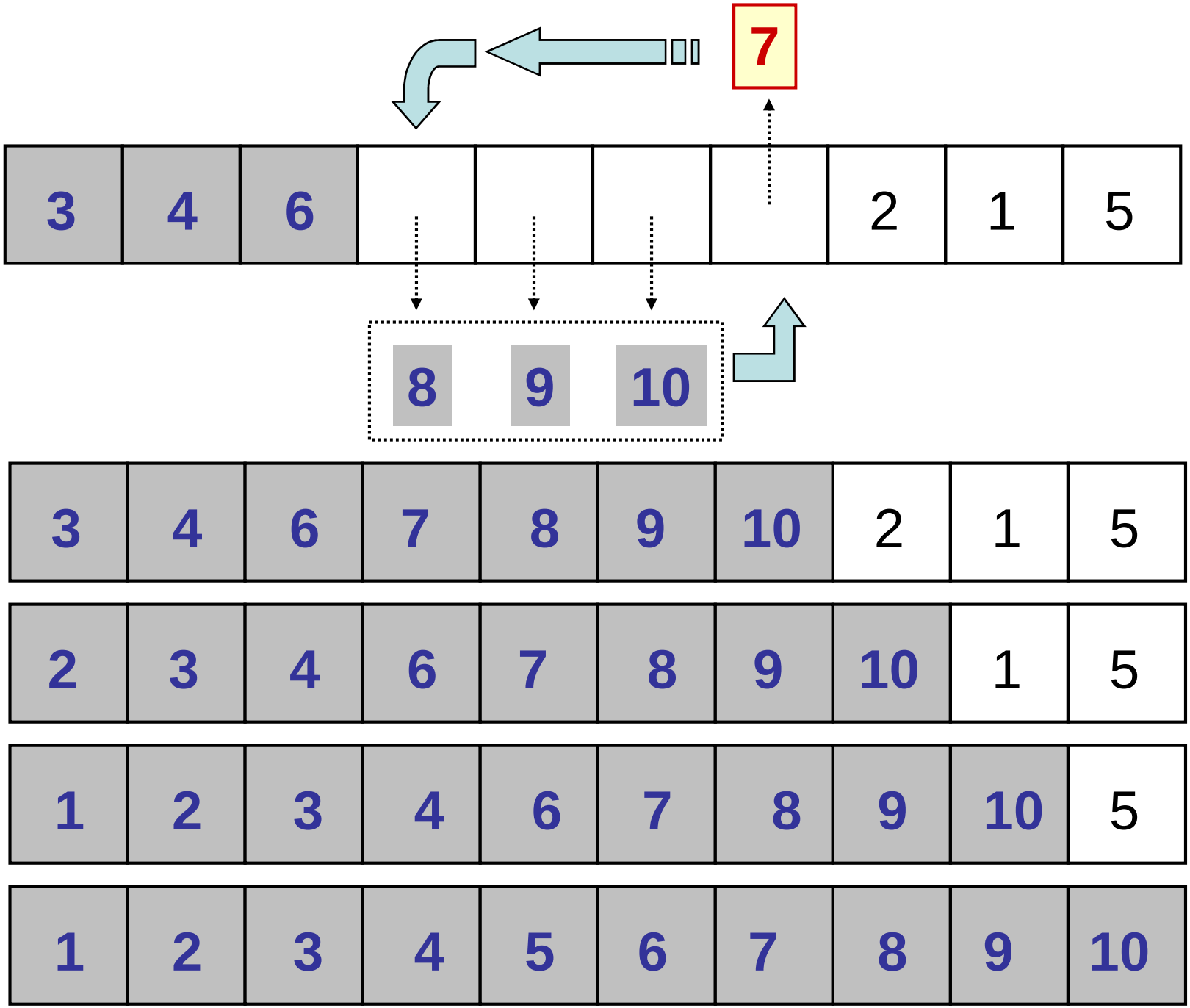
Sekarang ketiga nilai pertama secara relatif dalam kondisi terurut, dan kita telah menyisipkan nilai keempat dalam ketiga nilai tsb, sehingga keempat nilai pertama secara relatif telah terurut.

| **3** | **4** | **6** | **10** | 8 | 9 | 7 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|

Ulangi proses sampai nilai terakhir telah disisipkan.

| **3** | **4** | **6** | **8** | **10** | 9 | 7 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|

| **3** | **4** | **6** | **8** | **9** | **10** | 7 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|


7
3 4 6 2 1 5
8 9 10
3 4 6 7 8 9 10 2 1 5
2 3 4 6 7 8 9 10 1 5
1 2 3 4 6 7 8 9 10 5
1 2 3 4 5 6 7 8 9 10

# Algoritma

1. Ulangi langkah 2-6 untuk j=1 s/d n-1
2. i <- j – 1
3. tmp <- data[j]
4. Ulangi langkah 5-6 selama i>=0 dan tmp<data[i]
5. data[i+1]=data[i]
6. i--
7. data[i+1]=tmp

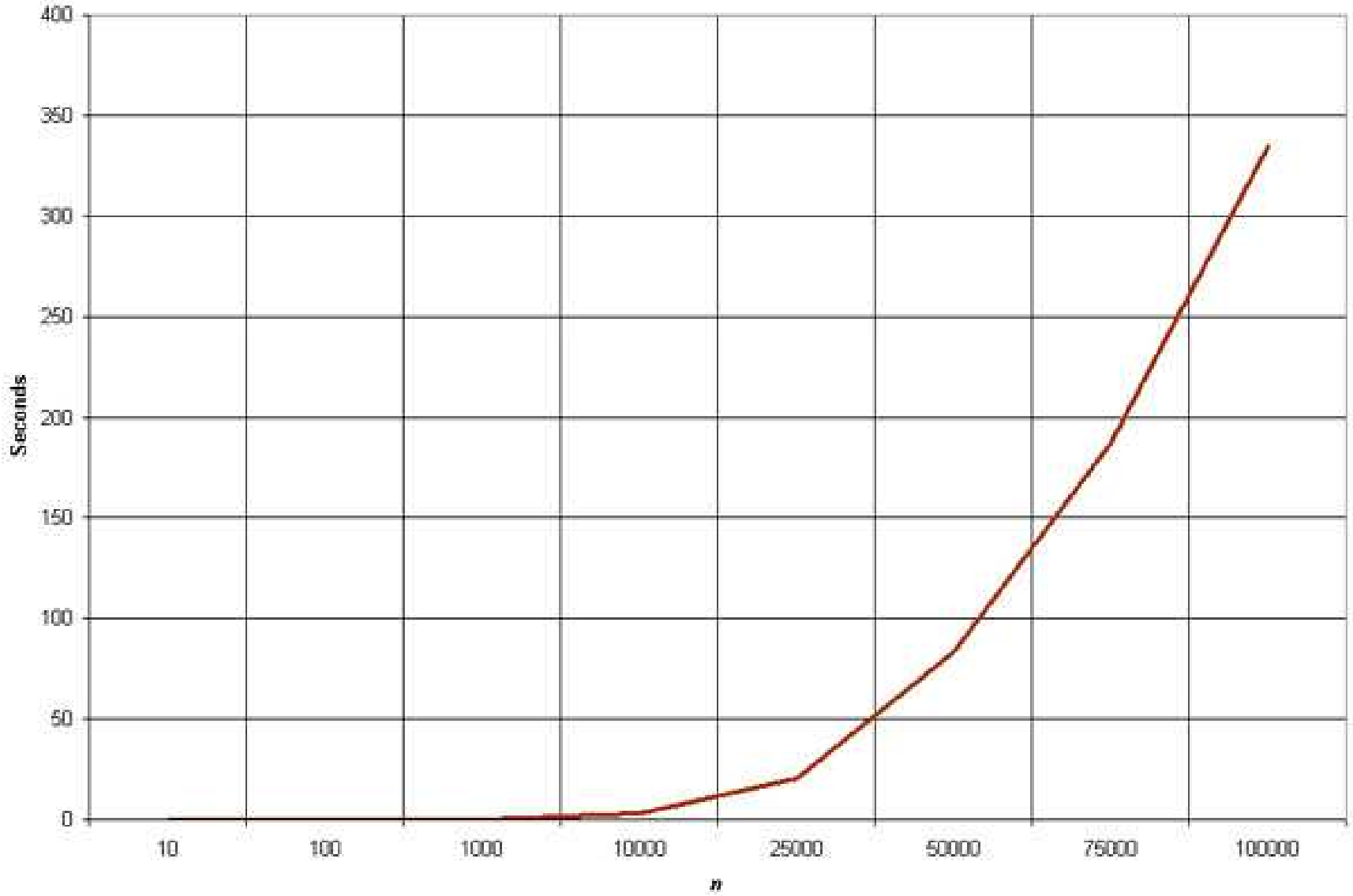
# Insertion Sort → Analysis

- Assuming there are *n* elements in the array, we must index through *n* - 1 entries.
- For each entry, we may need to examine and shift up to *n* - 1 other entries, resulting in a $\boldsymbol{O}(n^2)$ algorithm.
- The insertion sort is an *in-place* sort. That is, we sort the array in-place. No extra memory is required.
- The insertion sort is also a *stable* sort. Stable sorts retain the original ordering of keys when identical keys are present in the input data.

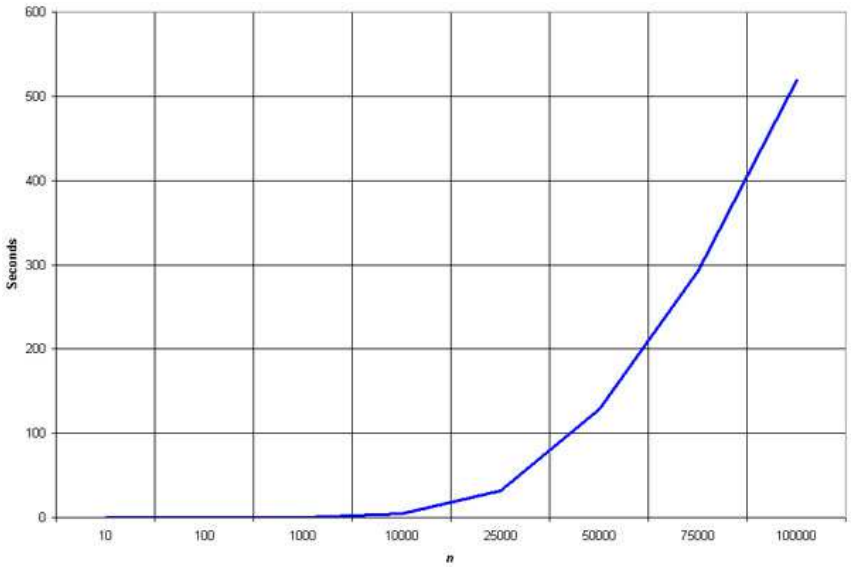
# Insertion Sort → Analysis

- **best case:**
  - about N comparisons, 0 moves, (input already sorted)
- **worst case:**
  - about $N^2/2$ comparisons, $N^2/2$ moves, (input sorted in reverse order)
- **average case:**
  - about $N^2/4$ comparisons, $N^2/4$ moves
- **notes:**
  - very efficient when input is "almost sorted";
  - moving a record is about half the work of exchanging two records, so average case is equivalent to roughly $N^2/8$ exchanges.

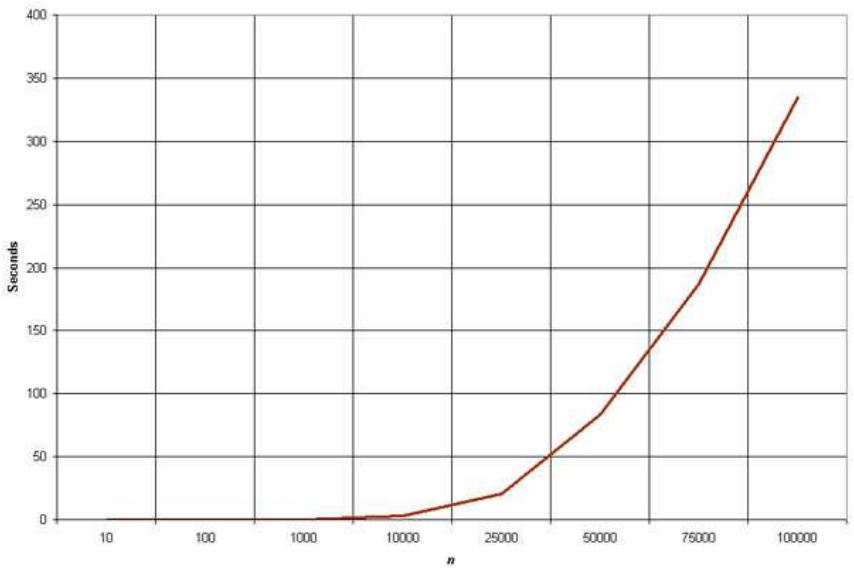
# Insertion Sort → Empirical Analysis

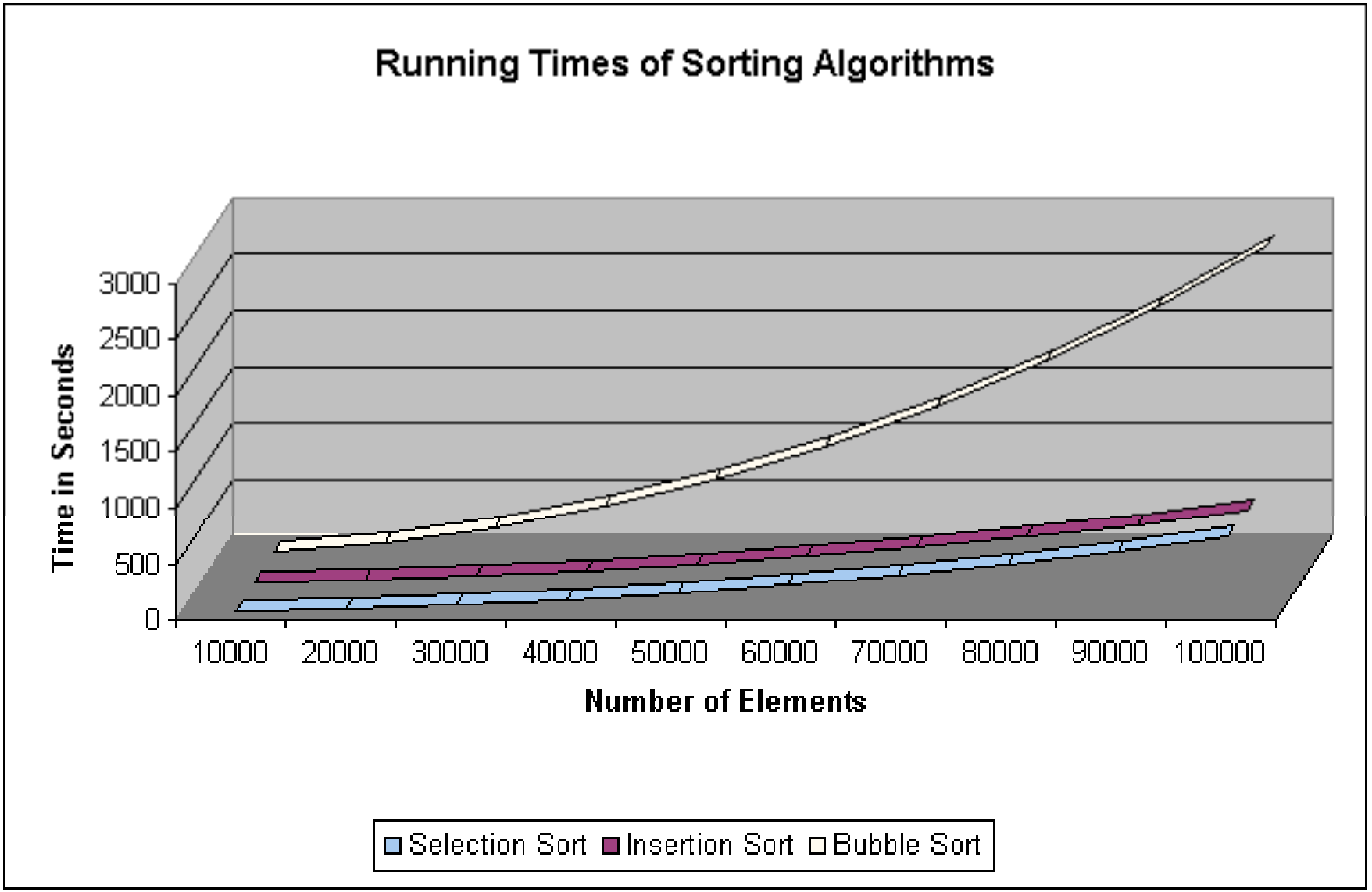


## Selection




## Insertion






**Selection Sort yields a 60% performance improvement over the Bubble Sort**

- Bubble sort uses about $N^2/2$ comparisons and $N^2/2$ exchanges on the average and in the worst case.

- Selection sort uses about $N^2/2$ comparisons and N exchanges on the average, twice as many in the worst case.

- Insertion sort uses about $N^2/4$ comparisons and $N^2/8$ exchanges and is linear for "almost sorted" files.