

# Sorting Algorithms

1. Selection

2. Bubble

3. Insertion

4. Merge

5. Quick

6. Shell

# Buble Sort

- Metode gelembung (*bubble sort*) disebut dengan metode penukaran (*exchange sort*) adalah metode yang mengurutkan data dengan cara membandingkan masing-masing elemen, kemudian melakukan penukaran bila perlu.
- Metode ini mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien.

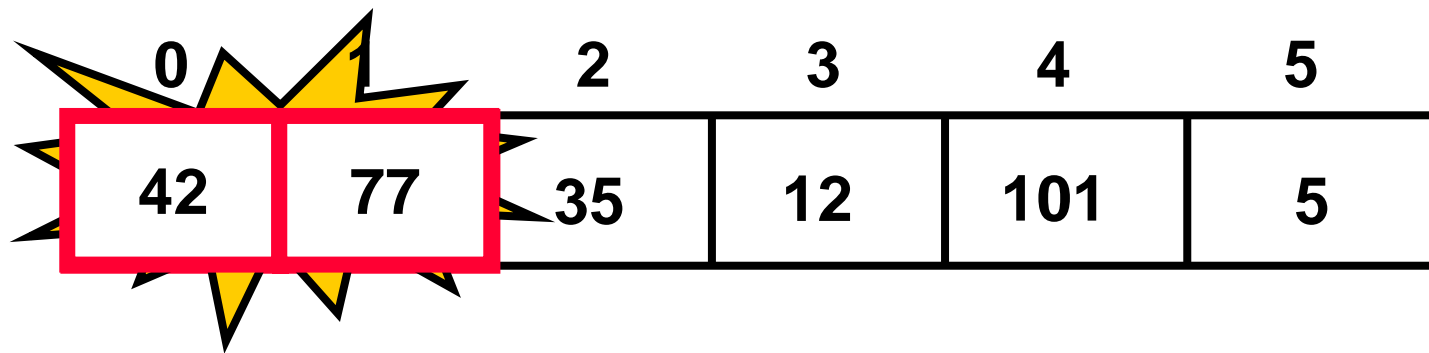
# "Bubbling Up" the Largest Element

- **Mengecek sekumpulan elemen**
  - Memindahkannya dari posisi awal ke akhir
  - “Menggelembungkan” **nilai terbesar** ke bagian akhir menggunakan metode **pembandingan sepasang dan penukaran**

0	1	2	3	4	5
77	42	35	12	101	5

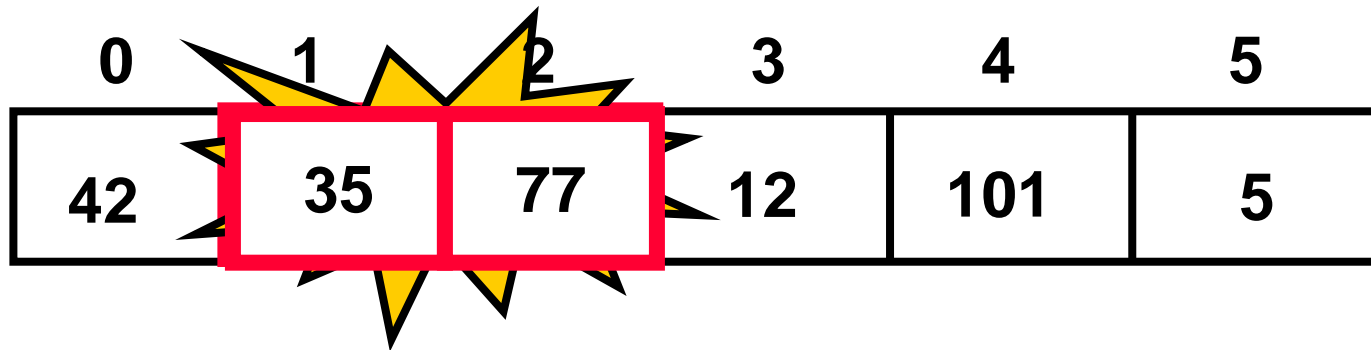
# "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping



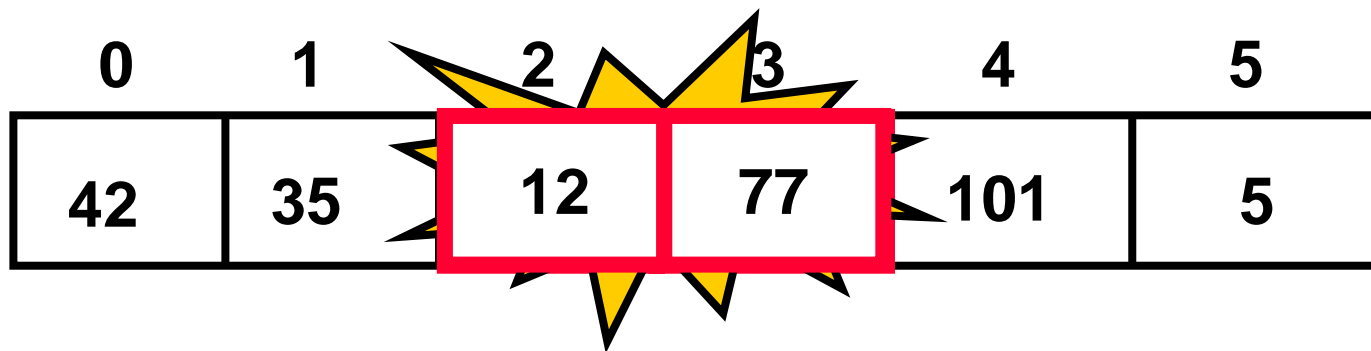
# "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping



# "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping



# "Bubbling Up" the Largest Element

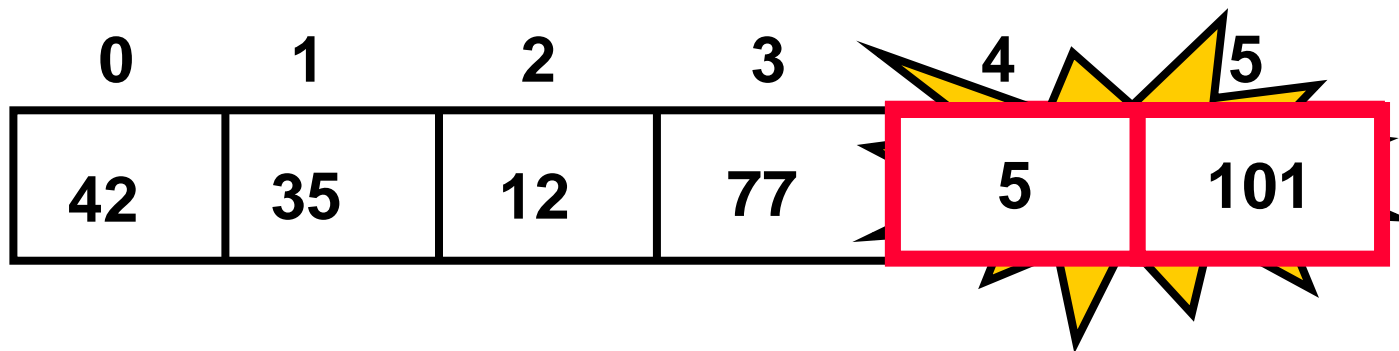
- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping

0	1	2	3	4	5
42	35	12	77	101	5

No need to swap

# "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping





# "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pairwise comparisons and swapping

0	1	2	3	4	5
42	35	12	77	5	101

Nilai terbesar telah menempati posisinya

# Algoritma Metode Bubble Sort

untuk satu kali iterasi

1. `pos_akhir`  $\leftarrow$  `n-2`
2. `index`  $\leftarrow$  0
3. selama `index`  $\leq$  `pos_akhir` kerjakan  
baris 4 dan 5
4. Jika `A[index] > A[index+1]`  
    `Swap(A[index], A[index+1])`
5. `index`  $\leftarrow$  `index + 1`

# Algoritma Metode Bubble Sort

versi 2

```
pos_akhir <- n - 2
```

```
index <- 0
```

```
loop
```

```
    exitif(index > pos_akhir)
```

```
    if(A[index] > A[index + 1]) then
```

```
        Swap(A[index], A[index + 1])
```

```
    endif
```

```
    index <- index + 1
```

```
endloop
```

# Yang perlu diperhatikan....

- Perhatikan bahwa hanya nilai terbesar yang sudah menempati posisinya
- Seluruh nilai yang lain masih belum terurutkan
- Sehingga kita perlu **mengulang proses ini**

0	1	2	3	4	5
42	35	12	77	5	101

Nilai terbesar telah menempati posisinya

# Repeat “Bubble Up” How Many Times?

- Jika kita punya  $N$  elemen...
- Dan jika setiap kali kita menggelembungkan sebuah elemen, kita menempatkannya pada posisi yang tepat...
- Berarti, kita **mengulang proses “bubble up” sebanyak  $N - 1$  kali.**
- Hal ini **menjamin kita akan menempatkan seluruh  $N$  elemen secara tepat.**

# “Bubbling” All the Elements

Diagram illustrating the bubble sort process across five iterations. A blue bracket on the left indicates the first pass (1-1) and the second pass (2).

0	1	2	3	4	5
42	35	12	77	5	101
35	12	42	5	77	101
12	35	5	42	77	101
12	5	35	42	77	101
5	12	35	42	77	101

# Mengurangi Jumlah Pembandingan

0	1	2	3	4	5
77	42	35	12	101	5

0	1	2	3	4	5
42	35	12	77	5	101

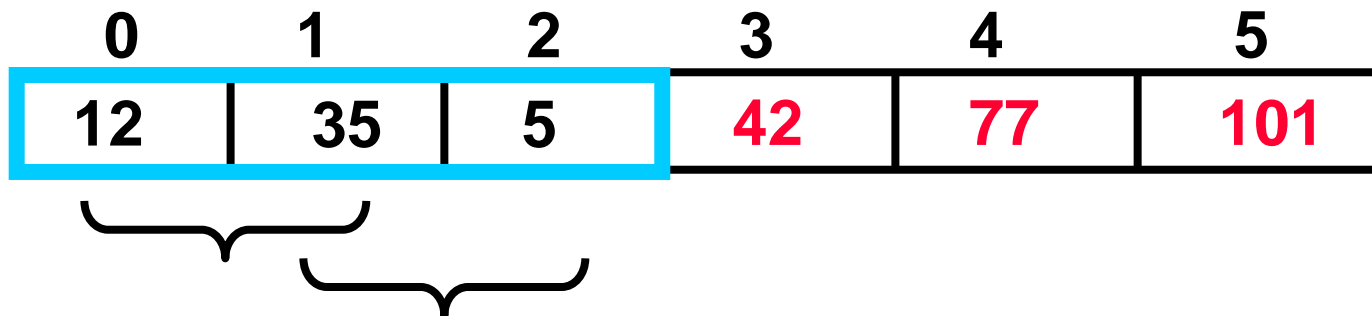
0	1	2	3	4	5
35	12	42	5	77	101

0	1	2	3	4	5
12	35	5	42	77	101

0	1	2	3	4	5
12	5	35	42	77	101

# Mengurangi Jumlah Perbandingan

- Pada proses “bubble up” ke-N, kita hanya butuh untuk melakukan sebanyak **MAX-N perbandingan**.
- Contoh:
  - Ini adalah proses “bubble up” ke-4
  - MAX adalah 6
  - So, kita punya **2 perbandingan** yang harus dilakukan





Putting It All Together

N adalah ... //Ukuran array

Arr\_Type define as Array[0..N-1] of Num

Procedure Swap(n1, n2 isoftype in/out Num)

temp isoftype Num





temp <- n1

n1 <- n2

n2 <- temp

endprocedure //Swap

```
procedure Bubblesort(A isoftype in/out Arr_Type)
  pos_akhir, index isoftype Num
  pos_akhir <- N - 2
```

```
  loop ← 
    exitif(pos_akhir < 0)
    index <- 0
    loop ← 
      exitif(index > pos_akhir)
      if(A[index] > A[index + 1]) then
        Swap(A[index], A[index + 1])
      endif
      index <- index + 1
    endloop ← 
    pos_akhir <- pos_akhir - 1
  endloop ← 
endprocedure // Bubblesort
```

**Inner loop**

**Outer loop**

# Apakah seluruh elemen telah terurut?

- Bagaimana jika seluruh elemen telah terurut?
- Bagaimana jika hanya sedikit elemen yang tidak pada posisinya, dan setelah beberapa operasi “bubble up,” seluruh elemen telah terurut?
- Kita menginginkan untuk bisa mendeteksi kondisi ini dan “stop early”!

0	1	2	3	4	5
5	12	35	42	77	101

# Gunakan sebuah “Flag” Boolean

- Kita bisa menggunakan sebuah variabel boolean untuk menentukan apakah terjadi operasi swapping selama proses “bubble up.”
- Jika tidak terjadi, maka kita akan mengetahui bahwa seluruh elemen telah terurut!
- “flag” boolean ini perlu di-RESET setiap kali selesai satu kali operasi “bubble up.”

```
did_swap isoftype Boolean
did_swap <- true

loop
  exitif ((pos_akhir < 0) OR NOT(did_swap))
  index <- 0
  did_swap <- false           //did_swap diRESET
  loop
    exitif(index > pos_akhir)
    if(A[index] > A[index + 1]) then
      Swap(A[index], A[index + 1])
      did_swap <- true
    endif
    index <- index + 1
  endloop
  pos_akhir <- pos_akhir - 1
endloop
```

# Summary

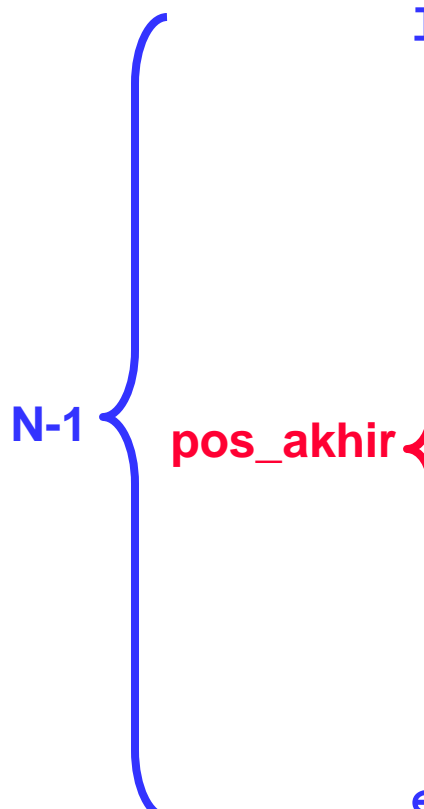
- Algoritma “Bubble Up” akan **memindahkan nilai terbesar ke posisinya yang tepat** (di sebelah kanan)
- Ulangi proses “Bubble Up” sampai seluruh elemen telah menempati posisinya yang tepat:
  - **Maximum sebanyak  $N-1$  kali**
  - Bisa berakhir lebih cepat jika **tidak lagi terjadi swapping (penukaran)**
- Kita mengurangi jumlah perbandingan elemen setiap kali satu elemen berhasil diletakkan pada posisinya yang tepat.

# Analysis of Bubble Sort

- **Berapa kali perbandingan pada inner loop?**
  - `pos_akhir` dimulai dari nilai  $N-2$  turun sampai dengan  $0$  , sehingga perbandingan pada inner loop adalah  $N-1$  kali
  - Average:  $N/2$  untuk setiap kali “pass” outer loop.
- **Berapa kali “pass” outer loop?**
  - $N - 1$



```
procedure BubbleSort(A isoftype in/out Arr_Type)
  pos_akhir, index isoftype Num
  pos_akhir <- N - 2
```



A blue brace on the left side of the code block, spanning from the first 'loop' to the 'endloop', is labeled 'N-1'. A red brace on the right side of the code block, spanning from the inner 'loop' to the 'endloop', is labeled 'pos\_akhir'.

```
  loop
    exitif(pos_akhir < 0)
    index <- 0
    loop
      exitif(index > pos_akhir)
      if(A[index] > A[index + 1]) then
        Swap(A[index], A[index + 1])
      endif
      index <- index + 1
    endloop
    pos_akhir <- pos_akhir - 1
  endloop
endprocedure // Bubblesort
```

# Bubble Sort → Analysis

- BEST CASE:
  - Array sudah dalam keadaan terurut naik
  - Jumlah perbandingan key (C) :  $n-1$
  - Jumlah swap = 0
  - Jumlah pergeseran (M) : 0
- WORST CASE
  - Array dalam urutan kebalikannya
  - Jumlah perbandingan key (C) :  
$$(n-1) + (n-2) + \dots + 1 = n * (n-1) / 2$$
  - Jumlah swap =  $(n-1) + (n-2) + \dots + 1 = n * (n-1) / 2$
  - Jumlah pergeseran (M) :  $3 * n * (n-1) / 2$

# Kompleksitas Bubble Sort

**Perhatikan pada hubungan antara 2 loop yang ada:**

- Inner loop bersarang di dalam outer loop**
- Inner loop akan dieksekusi untuk setiap iterasi dari outer loop**

# Bubble Sort

- Mirip dengan Selection, setiap kali proses “Bubble Up” akan memilih nilai maksimum dari elemen yang ada pada sisi unsorted
- Wastes time imparting some order to the unsorted part of the array

# $O(N^2)$ Runtime Example

- Assume you are sorting 250,000,000 items
- $N = 250,000,000$
- $N^2 = 6.25 \times 10^{16}$
- If you can do one operation per nanosecond ( $10^{-9}$  sec) *which is fast!*
- It will take  $6.25 \times 10^7$  seconds
- So  $6.25 \times 10^7$   
60 x 60 x 24 x 365  
= 1.98 years