

## Sorting Algorithms

1. Insertion
2. Selection
3. Bubble
4. Shell
5. Quick
6. Merge

## Sorting algorithms

- Metode Insertion, selection dan bubble sort memiliki worst-case performance yang bernilai kuadratik
- Apakah algoritma berbasis comparison yang tercepat ?

$O(n \log n)$

- Mergesort dan Quicksort

**ITS** Institut Teknologi Sepuluh Nopember

# Idea of Quicksort

- Ambil sebuah "pivot".
- Bagi menjadi 2 : bagian yang kurang dari dan bagian yang lebih dari pivot
- Urutkan masing-masing bagian secara rekursif

Arna Fariza                      Algoritma dan Struktur Data                      3

**ITS** Institut Teknologi Sepuluh Nopember

# Idea of Quicksort

- Select:** pick an element
- Divide:** rearrange elements so that **x goes to its final position**  
**E**
- Recur and Conquer:** recursively sort

Arna Fariza                      Algoritma dan Struktur Data                      4

## Quicksort Algorithm

Misal diberikan sebuah array  $A$  memiliki  $n$  elemen (integer)

→  $p = 0; r = n-1$

- Array  $A[p..r]$  dipartisi menjadi dua non-empty subarray :  
 $A[p..q]$  and  $A[q+1..r]$ 
  - Seluruh elemen dalam array  $A[p..q]$  lebih kecil dari seluruh elemen dalam array  $A[q+1..r]$
- Seluruh sub array diurutkan secara rekursif dengan cara memanggil fungsi `quicksort()`

## Quicksort Code

```
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r);
        Quicksort(A, p, q);
        Quicksort(A, q+1, r);
    }
}
```

## Partition

- Terlihat bahwa, seluruh aksi terjadi dalam fungsi `partition()`
  - Rearranges subarray secara in place
  - Hasil akhir:
    - Dua subarray
    - Seluruh elemen pada subarray pertama  $\leq$  seluruh elemen pada subarray kedua
  - Return value berupa index dari elemen “pivot” –yang memisahkan kedua subarray tsb
- *How do you suppose we implement this?*

## Partition In Words

- `Partition(A, p, r)`:
  - Pilih sebuah elemen yang bertindak sebagai “pivot” (*which?*)
  - Pecah array menjadi dua bagian, `A[p..i]` and `A[j..r]`
    - Seluruh element dalam `A[p..i]`  $\leq$  pivot
    - Seluruh element dalam `A[j..r]`  $\geq$  pivot
  - (HOW ?)
    - Increment `i` until `A[i]  $\geq$  pivot`
    - Decrement `j` until `A[j]  $\leq$  pivot`
    - Jika `i < j`, maka Swap `A[i]` and `A[j]`
    - Jika tidak, return `j`
    - Repeat until `i  $\geq$  j`

## Partition Code

```
Partition(A, p, r)
  x = A[p]; //pivot=elemen posisi pertama
  i = p ;   //inisialisasi
  j = r ;
  repeat
    while(A[j] > x)
      j--;
    while(A[i] < x)
      i++;
    if (i < j){
      Swap(A, i, j);
      j--;
      i++;
    }
  else
    return j;
  until i >= j
```

12	35	9	11	3	17	23	15	31	20
----	----	---	----	---	----	----	----	----	----

### QuickSort(0,9)

- X = PIVOT merupakan indeks ke -0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih dari atau sama dengan PIVOT. Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari atau sama dengan PIVOT. Cara kerjanya : selama Data[j] > PIVOT maka nilai j dikurangi

ITS Institut Teknologi Sepuluh Nopember

**q = Partition(0,9)**

12	35	9	11	3	17	23	15	31	20
----	----	---	----	---	----	----	----	----	----

**SWAP**

**PIVOT = 12**  
*i* = 0 *j* = 4  
*i* < *j* maka **SWAP**

3	35	9	11	12	17	23	15	31	20
---	----	---	----	----	----	----	----	----	----

**SWAP**

**PIVOT = 12**  
*i* = 1 *j* = 3  
*i* < *j* maka **SWAP**

3	11	9	35	12	17	23	15	31	20
---	----	---	----	----	----	----	----	----	----

Arna Fariza      Algoritma dan Struktur Data      11

ITS Institut Teknologi Sepuluh Nopember

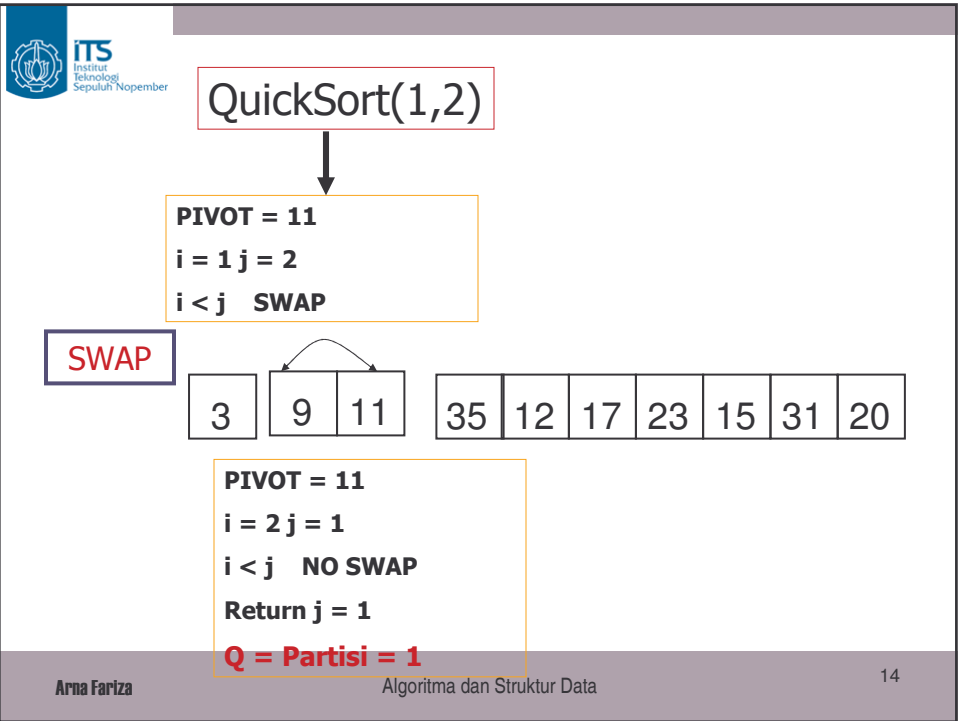
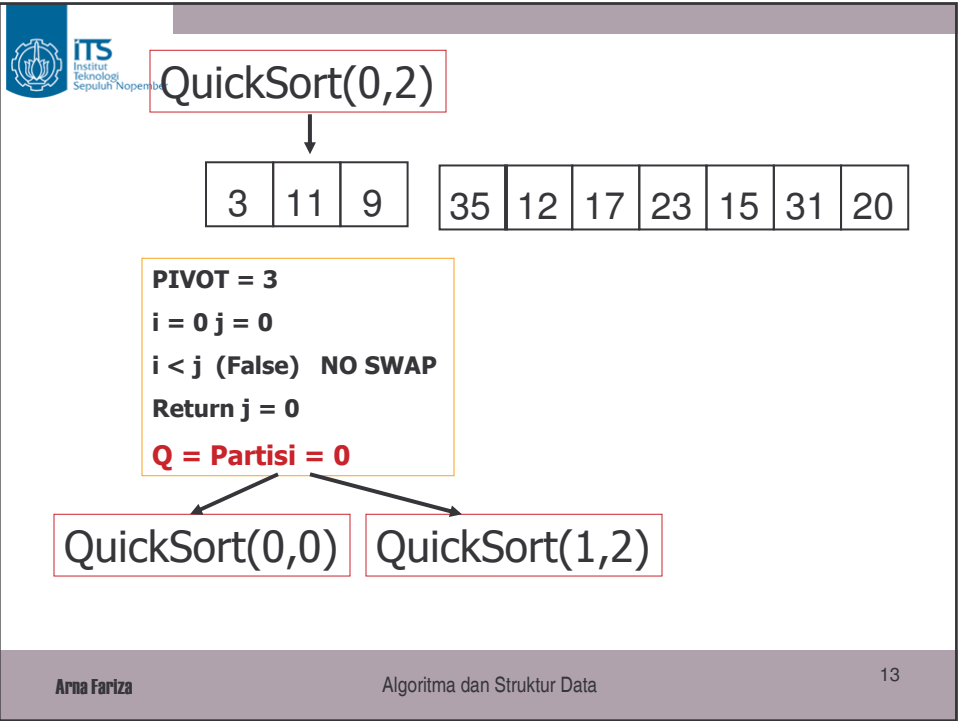
**PIVOT = 12**  
*i* = 3 *j* = 2  
*i* < *j* (False) **NO SWAP**  
 Return *j* = 2  
**Q = Partisi = 2**

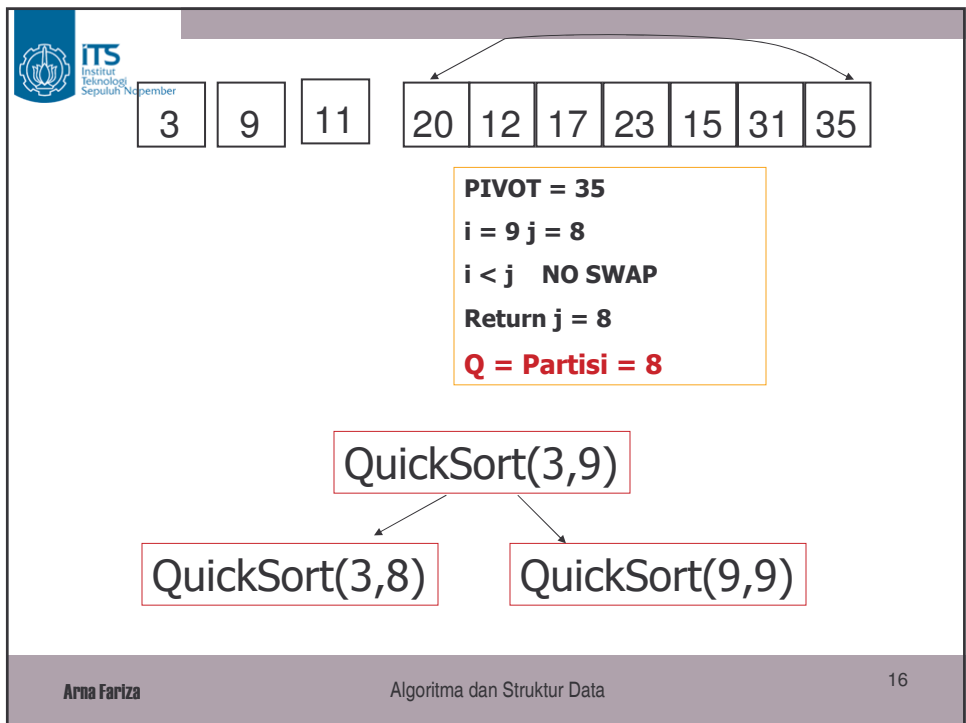
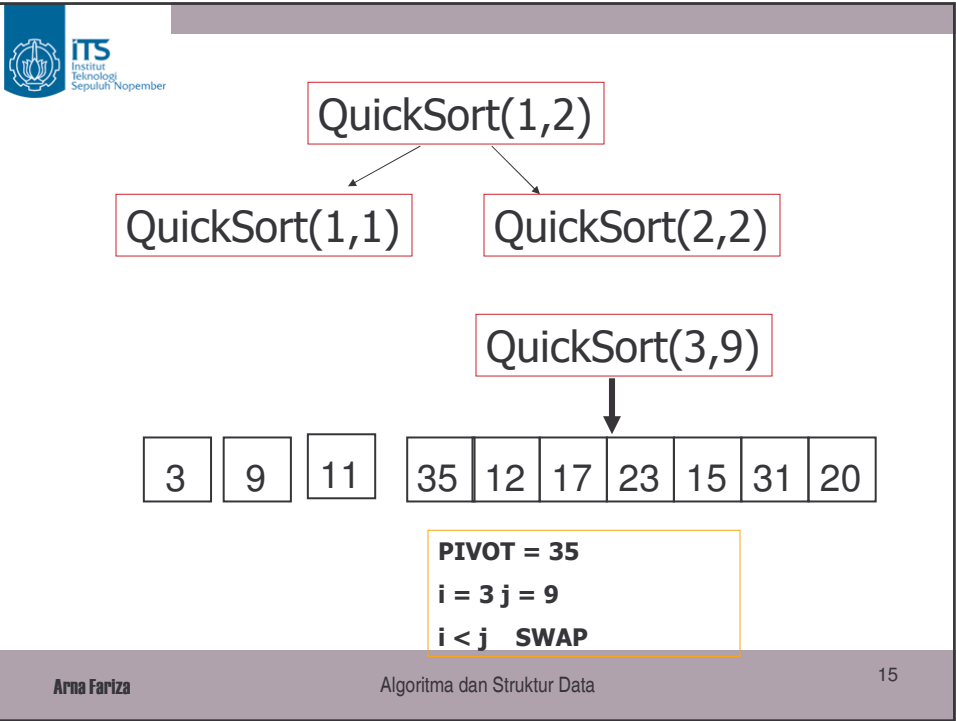
**QuickSort(0,9)**

↙      ↘

**QuickSort(0,2)**      **QuickSort(3,9)**

Arna Fariza      Algoritma dan Struktur Data      12







ITS  
Institut Teknologi  
Sepuluh Nopember

3 9 11 20 12 17 23 15 31 35

**QuickSort(3,8)**

PIVOT = 20  
i = 3 j = 7  
i < j SWAP

**SWAP**

3 9 11 15 12 17 23 20 31 35

PIVOT = 20  
i = 6 j = 5  
i < j NO SWAP  
Return j = 5  
**Q = Partisi = 5**

Arna Fariza      Algoritma dan Struktur Data      17

ITS  
Institut Teknologi  
Sepuluh Nopember

**QuickSort(3,8)**

**QuickSort(3,5)**      **QuickSort(6,8)**

↓

PIVOT = 15  
i = 3 j = 4  
i < j SWAP

**SWAP**

3 9 11 12 15 17 23 20 31 35

Arna Fariza      Algoritma dan Struktur Data      18

ITS  
Institut Teknologi Sepuluh Nopember

**PIVOT = 15**  
**i = 4 j = 3**  
**i < j NO SWAP**  
**Return j = 3**  
**Q = Partisi = 3**

q = 3

QS(3,5)

QS(3,3)      QS(4,5)

3	9	11	12	15	17	23	20	31	35
---	---	----	----	----	----	----	----	----	----

Arna Fariza      Algoritma dan Struktur Data      19

ITS  
Institut Teknologi Sepuluh Nopember

**PIVOT = 15**  
**i = 4 j = 4**  
**i < j NO SWAP**  
**Return j = 4**  
**Q = Partisi = 4**

q = 4

QS(4,5)

QS(4,4)      QS(5,5)

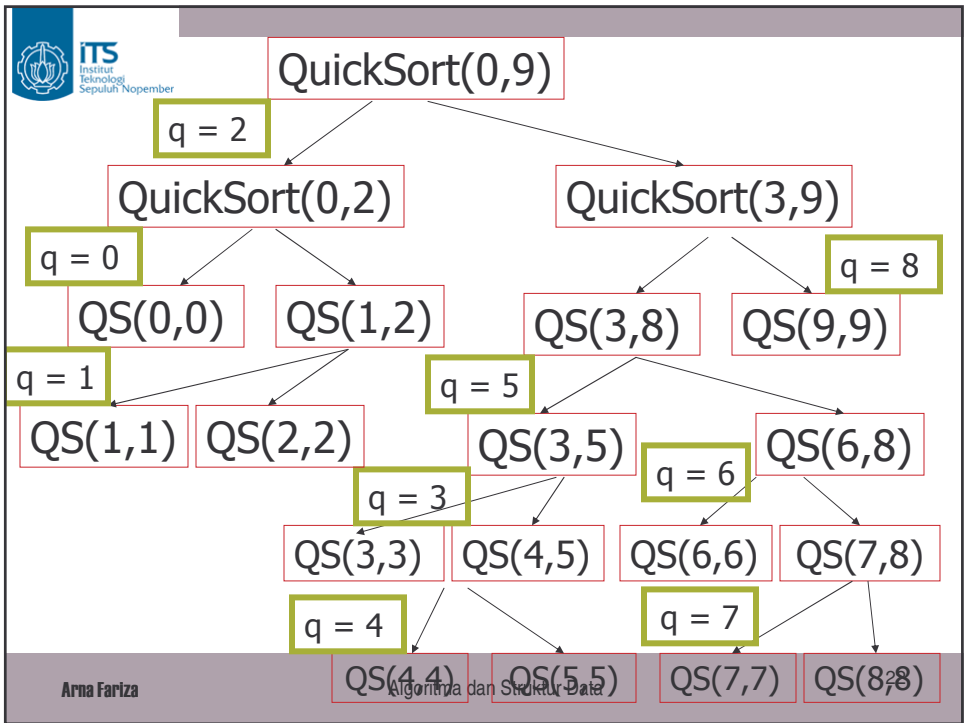
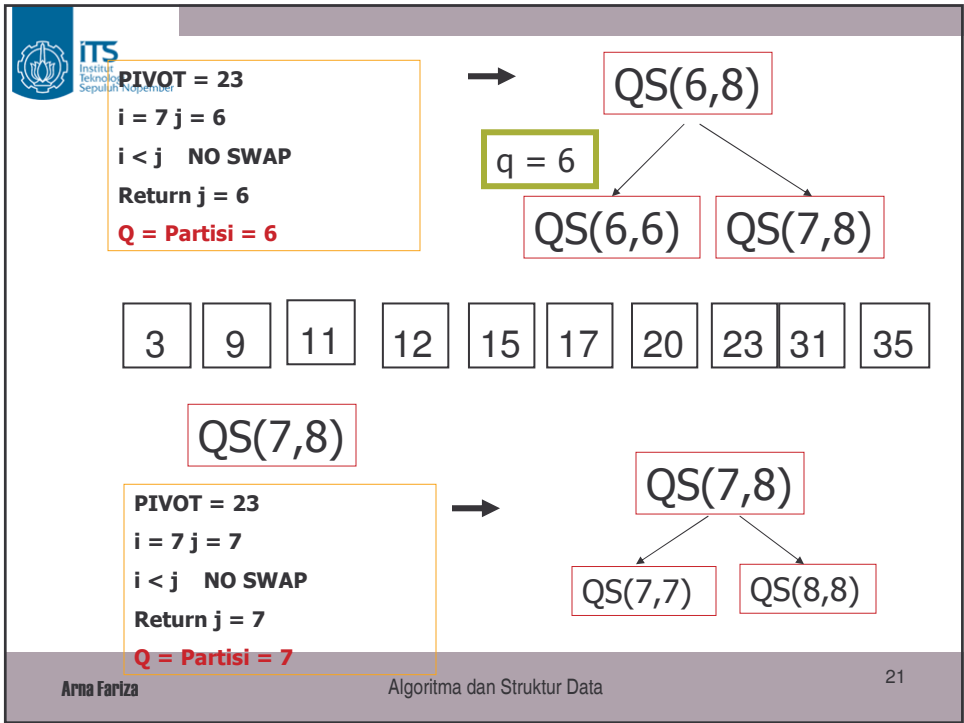
3	9	11	12	15	17	23	20	31	35
---	---	----	----	----	----	----	----	----	----

QuickSort(6,8) →

**PIVOT = 23**  
**i = 6 j = 7**  
**i < j SWAP**

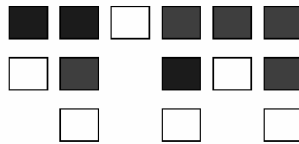
3	9	11	12	15	17	20	23	31	35
---	---	----	----	----	----	----	----	----	----

Arna Fariza      Algoritma dan Struktur Data      20



## Quicksort Analysis

- Jika diasumsikan pivot diambil secara random, terdistribusi secara uniform
- **Best case running time:  $O(n \log_2 n)$** 
  - Pada setiap pemanggilan rekursif, posisi elemen pivot selalu persis di tengah, array dipecah menjadi dua bagian yang sama, elemen-elemen yang lebih kecil dan yang lebih besar dari pivot



## Quicksort Analysis

### Worst case: $O(N^2)$

- Pada setiap pemanggilan rekursif, pivot selalu merupakan elemen terbesar (atau terkecil); array dipecah menjadi satu bagian yang semua elemennya lebih kecil dari pivot, pivot, dan sebuah bagian lagi array yang empty

## Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	<ul style="list-style-type: none"><li>• in-place</li><li>• slow (good for small inputs)</li></ul>
insertion-sort	$O(n^2)$	<ul style="list-style-type: none"><li>• in-place</li><li>• slow (good for small inputs)</li></ul>
quick-sort	$O(n \log n)$ expected	<ul style="list-style-type: none"><li>• in-place, randomized</li><li>• fastest (good for large inputs)</li></ul>
merge-sort	$O(n \log n)$	<ul style="list-style-type: none"><li>• sequential data access</li><li>• fast (good for huge inputs)</li></ul>