Sorting Algorithms

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

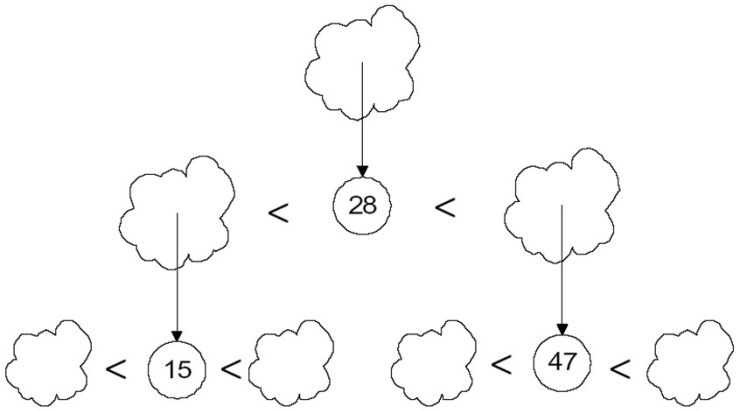
Sorting algorithms

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

O(nlogn)

- Mergesort dan Quicksort

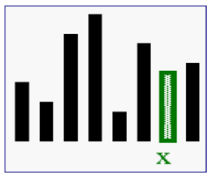
Idea of Quicksort



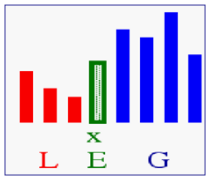
- Ambil sebuah “pivot”.
- Bagi menjadi 2 : bagian yang kurang dari dan bagian yang lebih dari pivot
- Urutkan masing-masing bagian secara rekursif

Idea of Quicksort

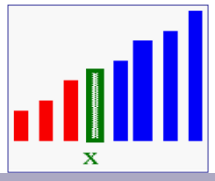
1. **Select**: pick an element



2. **Divide**: rearrange elements so that **x goes to its final position E**



3. **Recur and Conquer**: recursively sort



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 ≤ 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] <= pivot
    - Seluruh element dalam A[j..r] >= pivot

    (HOW ?)
  - Increment i until A[i] >= pivot
  - Decrement j until A[j] <= pivot
  - Jika i < j, maka Swap A[i] and A[j]
  - Jika tidak, return j
  - Repeat until i >= j

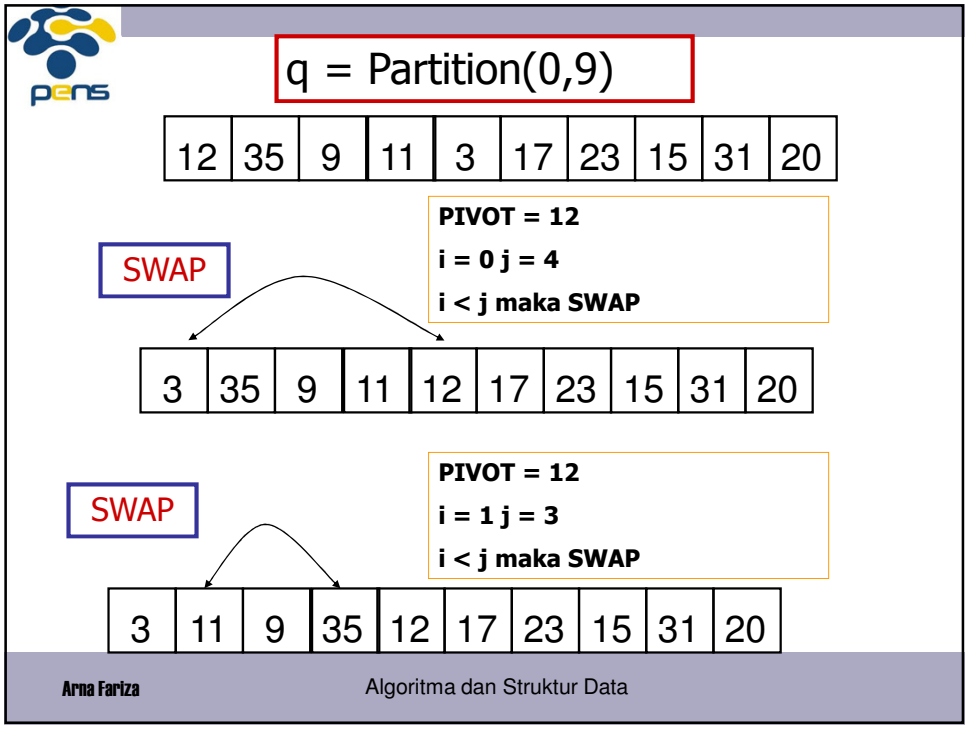
Partition Code

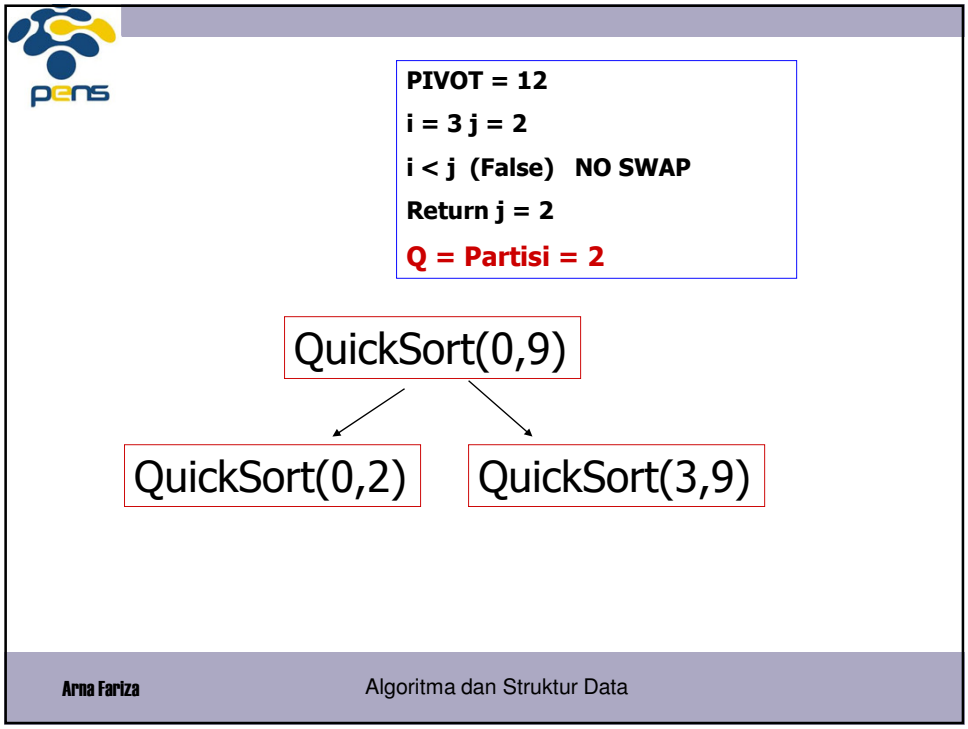
```
Partition(A, p, r)
    x = A[p];  //pivot=elemen posisi pertama
    i = p ;    //inisialisasi
    j = r ;
    repeat
        while(A[j] > x)
            j--;
         while(A[i] < x)
               i++;
         if (i < j){
             Swap(A, i, j);
            j--;
           i++
        }
  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

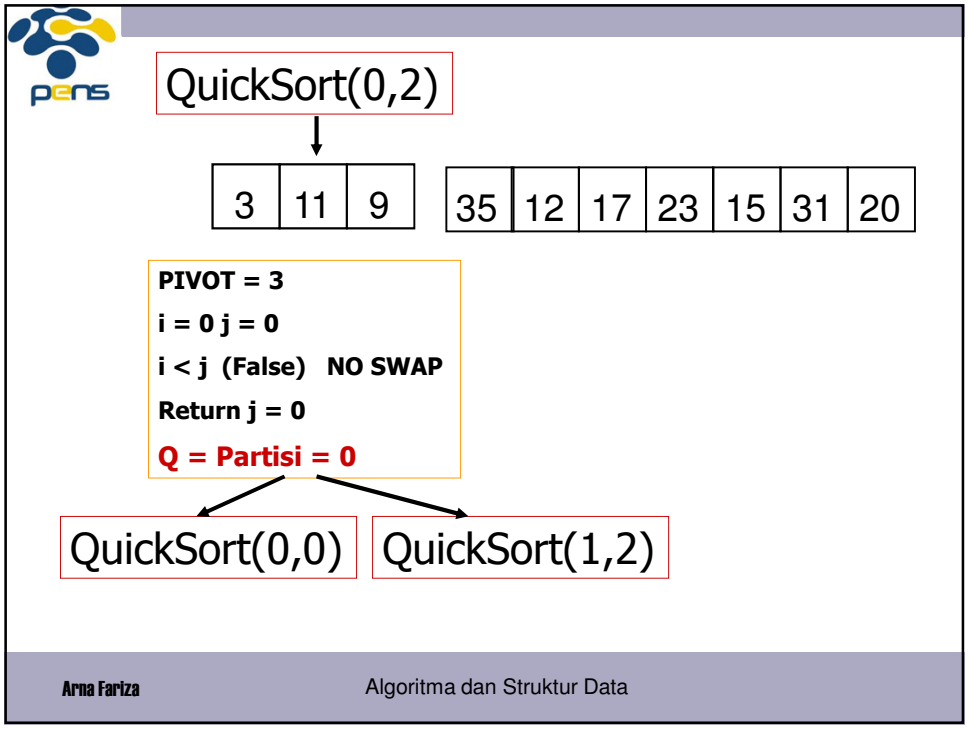

12	35	9	11	3	17	23	15	31	20

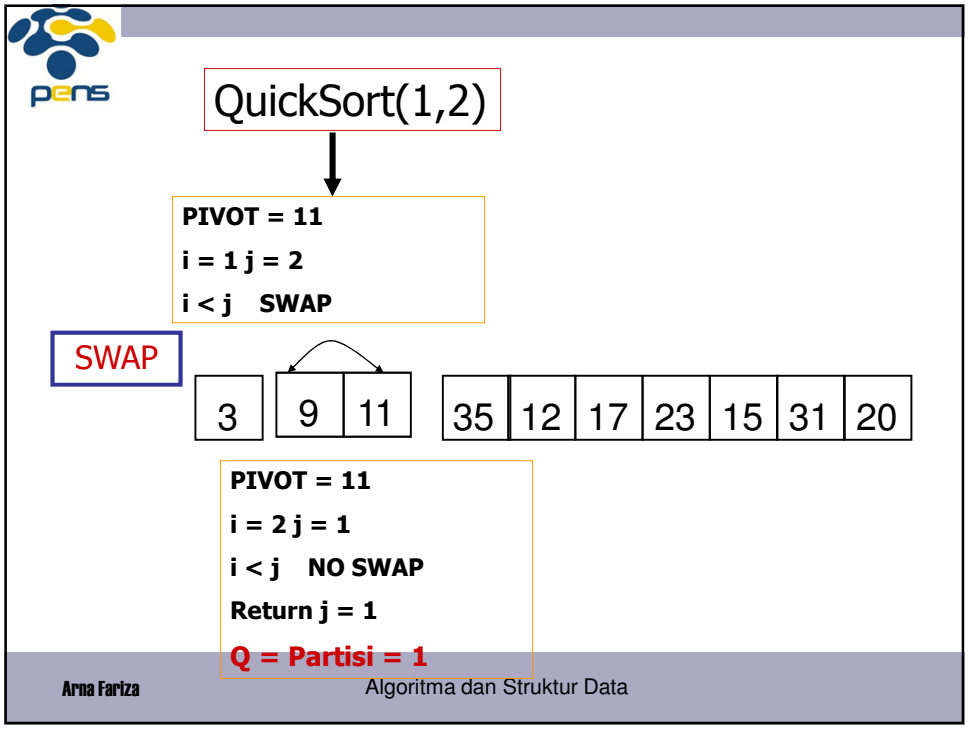
QuickSort(0,9)

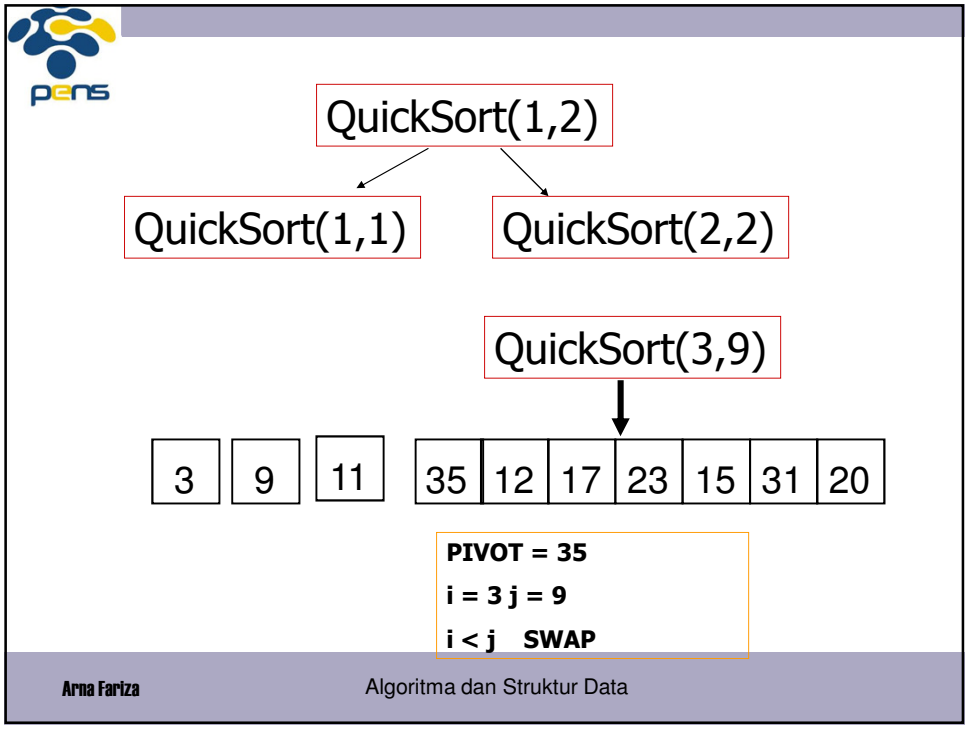
- X = PIVOT merupakan indeks ke –0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih dari atau sama dengan PIVOT. Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari atau sama dengan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

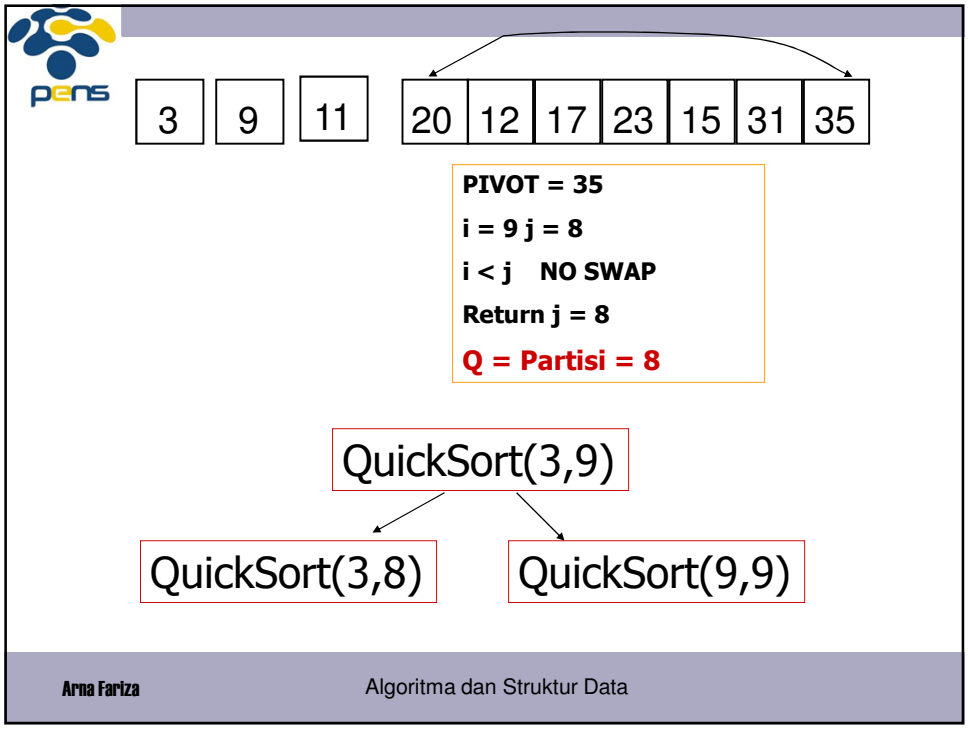

q = Partition(0,9)
12 35 9 11 3 17 23 15 31 20
PIVOT = 12
i = 0 j = 4
i < j maka SWAP
SWAP
3 35 9 11 12 17 23 15 31 20
PIVOT = 12
i = 1 j = 3
i < j maka SWAP
SWAP
3 11 9 35 12 17 23 15 31 20

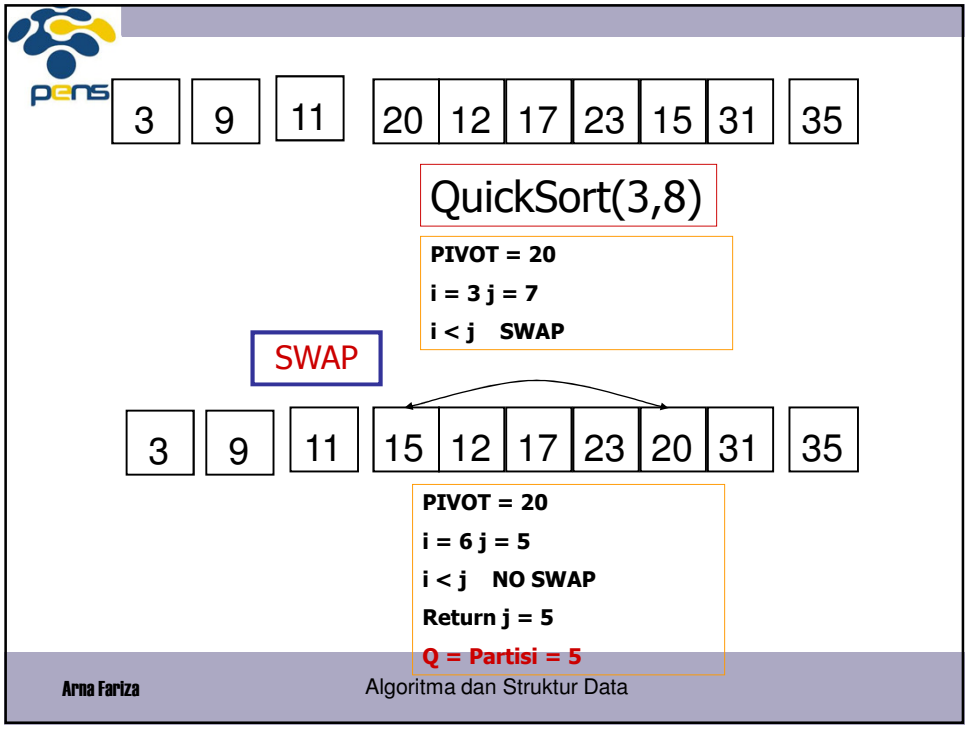

PIVOT = 12
i = 3 j = 2
i < j (False) NO SWAP
Return j = 2
Q = Partisi = 2
QuickSort(0,9)
QuickSort(0,2)
QuickSort(3,9)

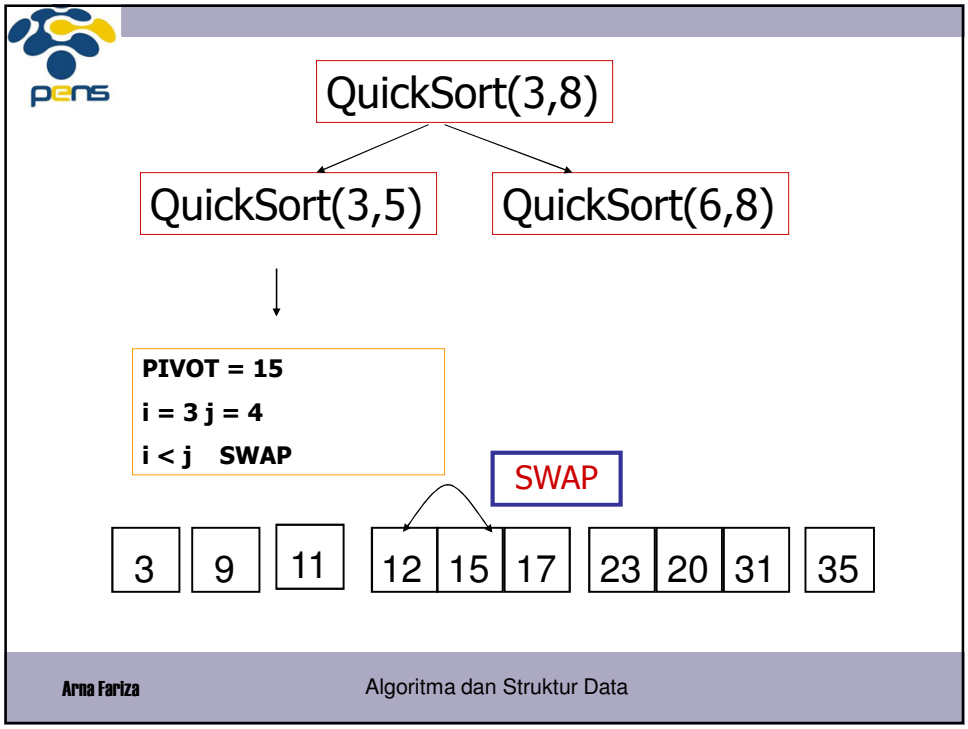

QuickSort(0,2)
3 11 9
35 12 17 23 15 31 20
PIVOT = 3
i = 0 j = 0
i < j (False) NO SWAP
Return j = 0
Q = Partisi = 0
QuickSort(0,0)
QuickSort(1,2)
Arna Fariza
Algoritma dan Struktur Data

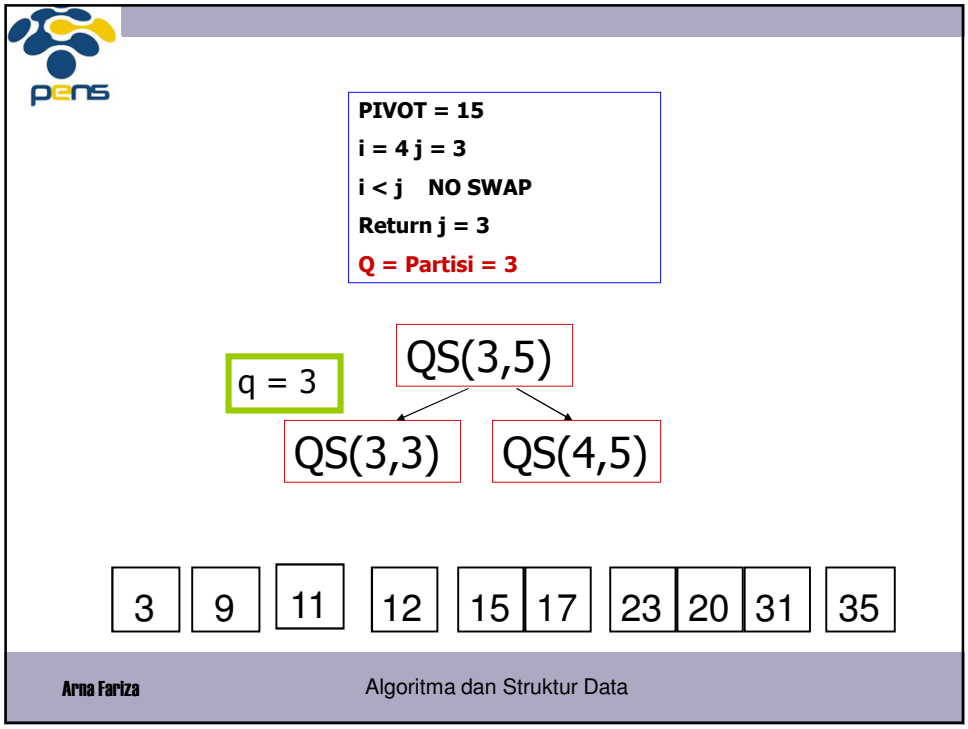

QuickSort(1,2)
PIVOT = 11
i = 1 j = 2
i < j SWAP
SWAP
3 9 11
35 12 17 23 15 31 20
PIVOT = 11
i = 2 j = 1
i < j NO SWAP
Return j = 1
Q = Partisi = 1
Arna Fariza
Algoritma dan Struktur Data

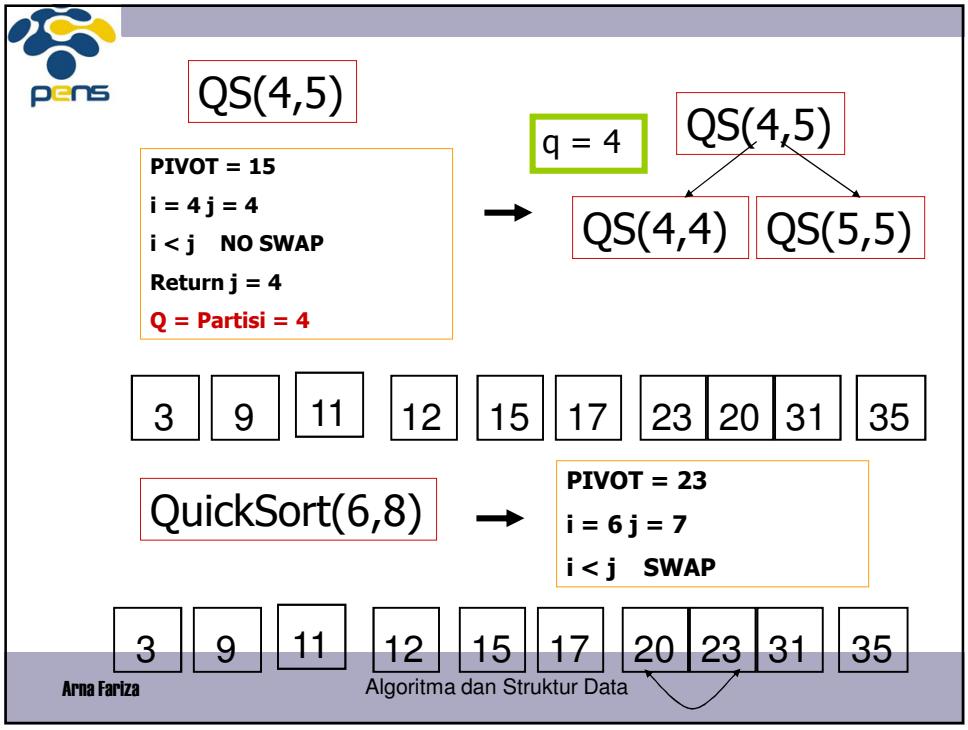

QuickSort(1,2)
QuickSort(1,1)
QuickSort(2,2)
QuickSort(3,9)
3
9
11
35
12
17
23
15
31
20
PIVOT = 35
i = 3 j = 9
i < j SWAP

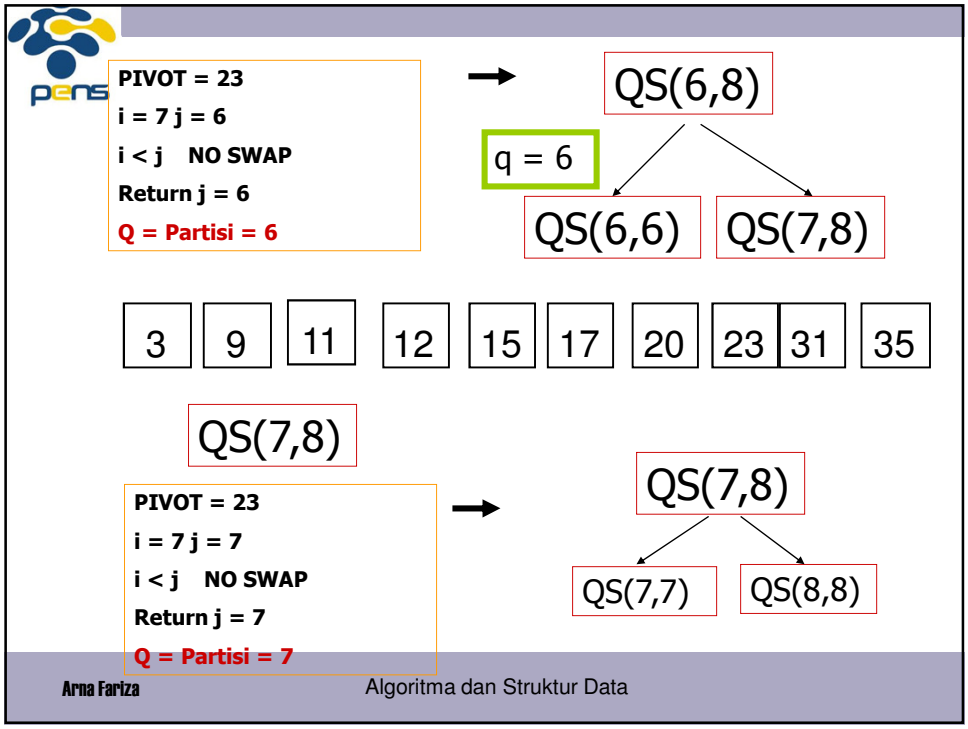

3
9
11
20
12
17
23
15
31
35
PIVOT = 35
i = 9 j = 8
i < j NO SWAP
Return j = 8
Q = Partisi = 8
QuickSort(3,9)
QuickSort(3,8)
QuickSort(9,9)

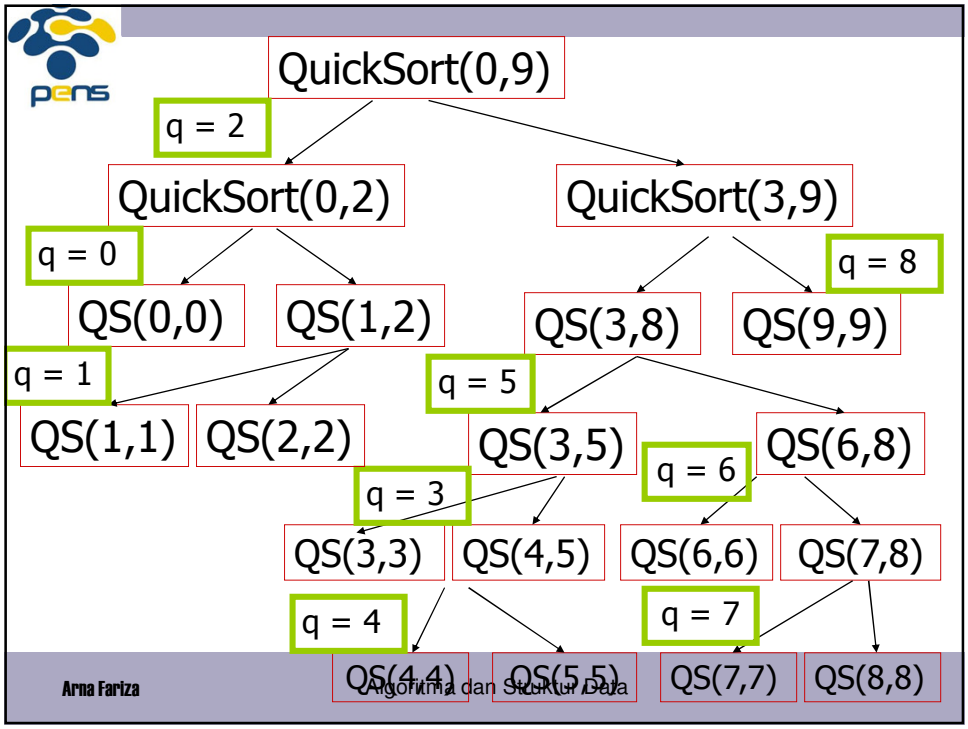

3 9 11 20 12 17 23 15 31 35
QuickSort(3,8)
PIVOT = 20
i = 3 j = 7
i < j SWAP
SWAP
3 9 11 15 12 17 23 20 31 35
PIVOT = 20
i = 6 j = 5
i < j NO SWAP
Return j = 5
Q = Partisi = 5


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


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

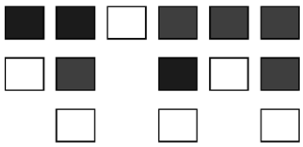

QS(4,5)
PIVOT = 15
i = 4 j = 4
i < j NO SWAP
Return j = 4
Q = Partisi = 4
q = 4
QS(4,5)
QS(4,4)
QS(5,5)
3 9 11 12 15 17 23 20 31 35
QuickSort(6,8)
PIVOT = 23
i = 6 j = 7
i < j SWAP
3 9 11 12 15 17 20 23 31 35
Arna Fariza
Algoritma dan Struktur Data


PIVOT = 23
i = 7 j = 6
i < j NO SWAP
Return j = 6
Q = Partisi = 6
QS(6,8)
q = 6
QS(6,6)
QS(7,8)
3
9
11
12
15
17
20
23
31
35
QS(7,8)
PIVOT = 23
i = 7 j = 7
i < j NO SWAP
Return j = 7
Q = Partisi = 7
QS(7,8)
QS(7,7)
QS(8,8)
Arna Fariza
Algoritma dan Struktur Data


QuickSort(0,9)
q = 2
QuickSort(0,2)
QuickSort(3,9)
q = 0
q = 8
QS(0,0)
QS(1,2)
QS(3,8)
QS(9,9)
q = 1
q = 5
QS(1,1)
QS(2,2)
QS(3,5)
QS(6,8)
q = 6
q = 3
QS(3,3)
QS(4,5)
QS(6,6)
QS(7,8)
q = 4
q = 7
QS(4,4)
QS(5,5)
QS(7,7)
QS(8,8)
Arna Fariza
Algoritma dan Struktur Data

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	$O(n^2)$	• in-place • slow (good for small inputs)
insertion-sort	$O(n^2)$	• in-place • slow (good for small inputs)
quick-sort	$O(n \log n)$ expected	• in-place, randomized • fastest (good for large inputs)
merge-sort	$O(n \log n)$	• sequential data access • fast (good for huge inputs)