Algoritma dan Struktur Data

Quick Sort

Umi Sa'adah
Tita Karlita
Entin Martiana Kusumaningtyas
Arna Fariza
2021

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Sorting Algorithms

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

Sorting algorithms

- Metode insertion, selection dan bubble sort memiliki worst-case performance yang bernilai quadratic.
- Apakah algoritma berbasis comparison yang tercepat ?

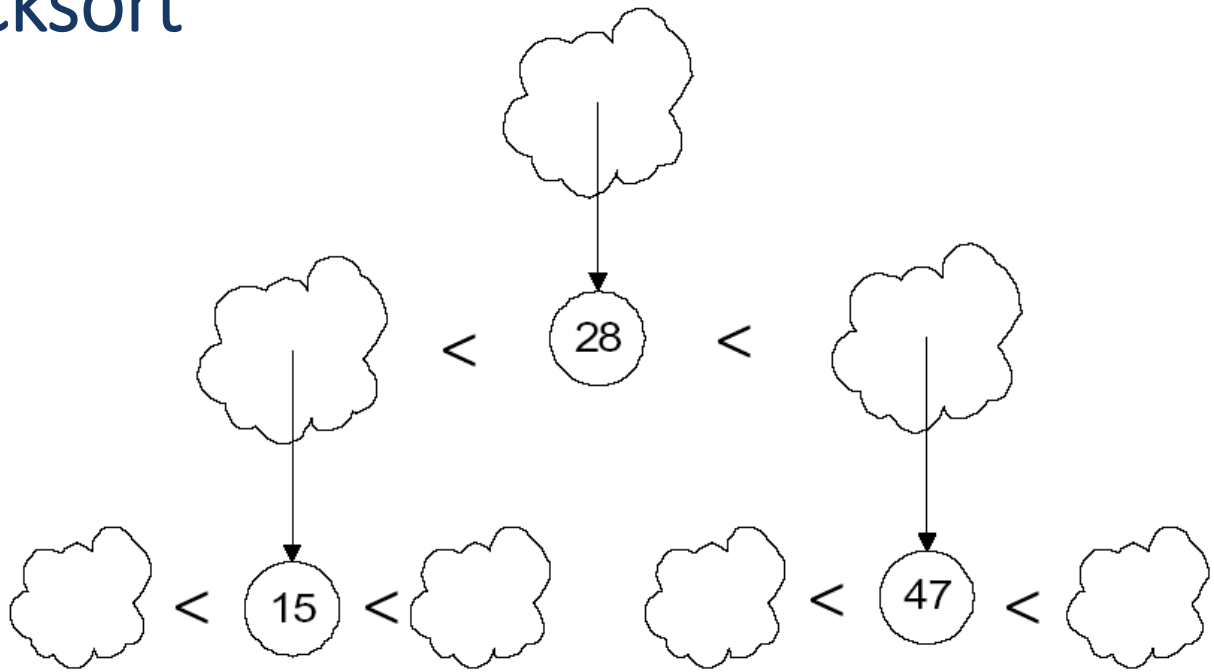
O(nlogn)

- Kompleksitas O(nlogn) dimiliki oleh Merge sort dan Quick sort

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

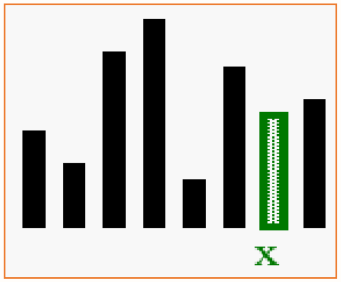
Idea of Quicksort



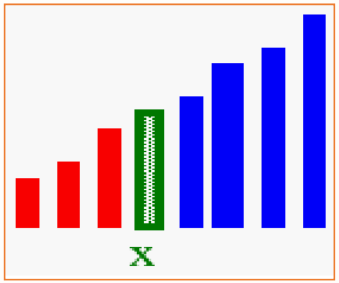
- Ambil sebuah “pivot”.
- Bagi menjadi 2: bagian yang kurang dari dan bagian yang lebih dari pivot
- Urutkan masing-masing bagian secara rekursif

Idea of Quicksort

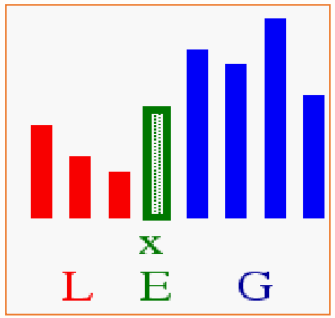
1. **Select**: pick an element (x)



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



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

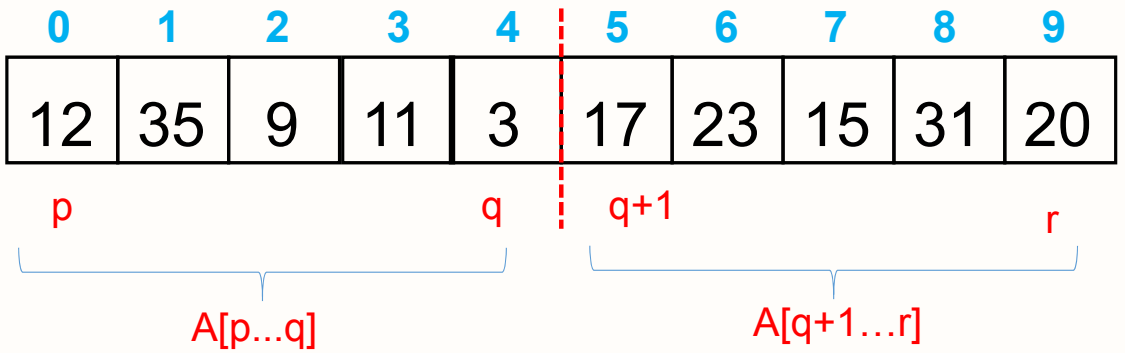


Quick sort algorithm

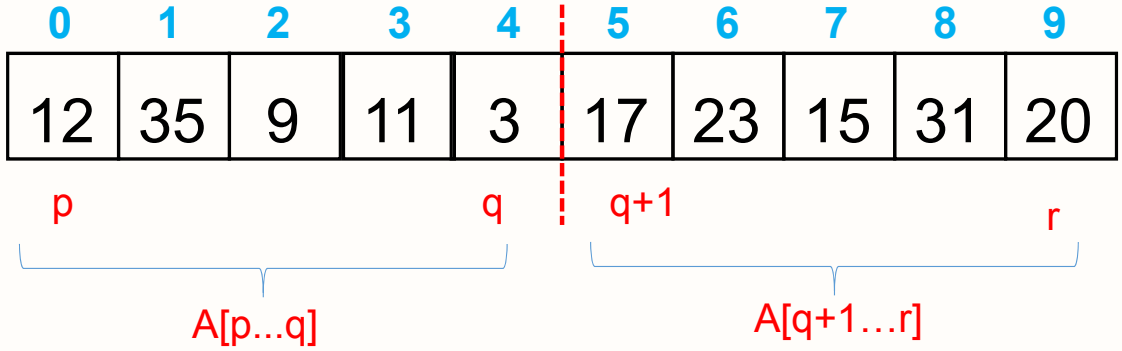
Misal diberikan sebuah array A memiliki sejumlah n elemen (integer)

→ p = 0; r = n-1

- Array A[p.. r] dipartisi menjadi dua non-empty sub-array : A[p...q] and A[q+1...r]
  - Goal: 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



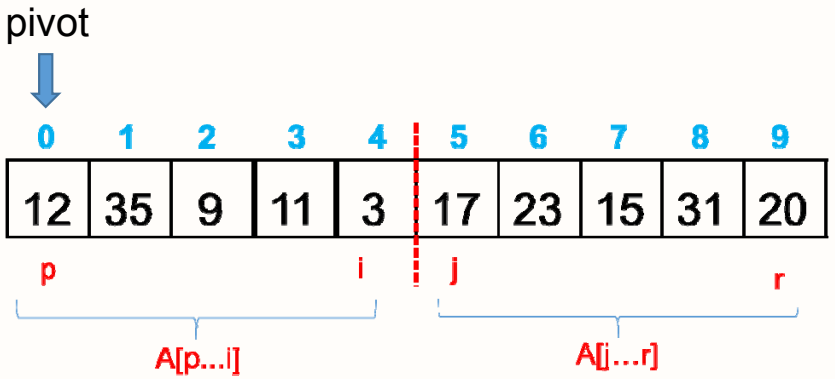

```
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r);
        Quicksort(A, p, q);
        Quicksort(A, q+1, r);
    }
}
```

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

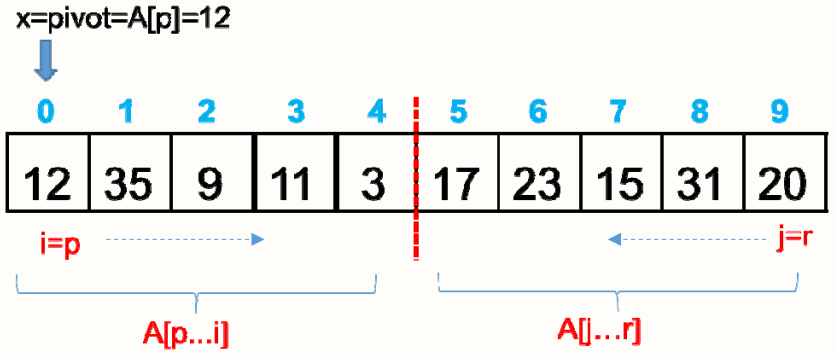
- *How do you suppose we implement this?*

Partition in Words



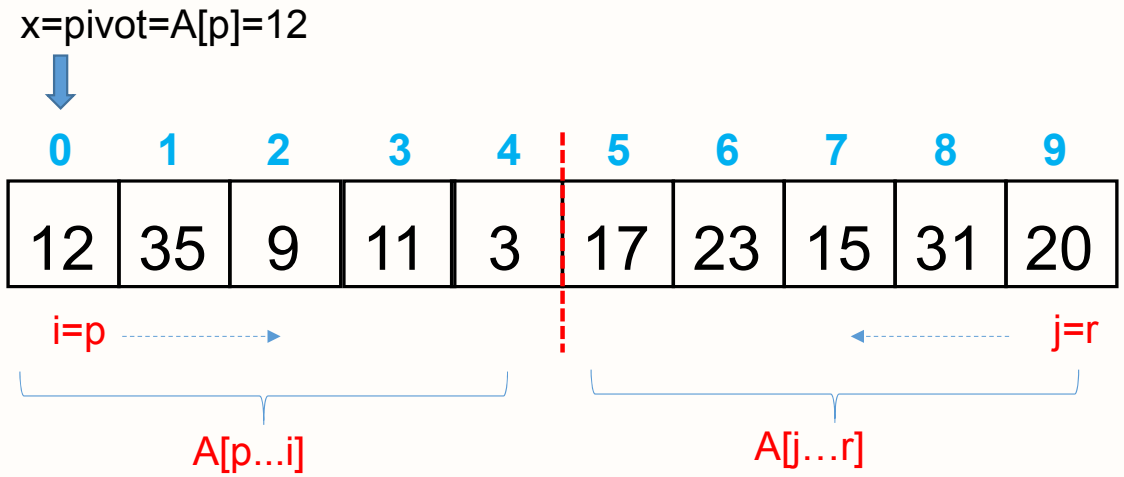
- Partition(A, p, r):
  - Pilih sebuah elemen yang bertindak sebagai "pivot" (*which?*)
  - Pecah array menjadi dua bagian, A[p...i] and A[j...r]
    - Seluruh element dalam A[p...i] <= pivot
    - Seluruh element dalam A[j...r] >= pivot

    (HOW ?)
  - Set i=p, j=r
  - 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
```



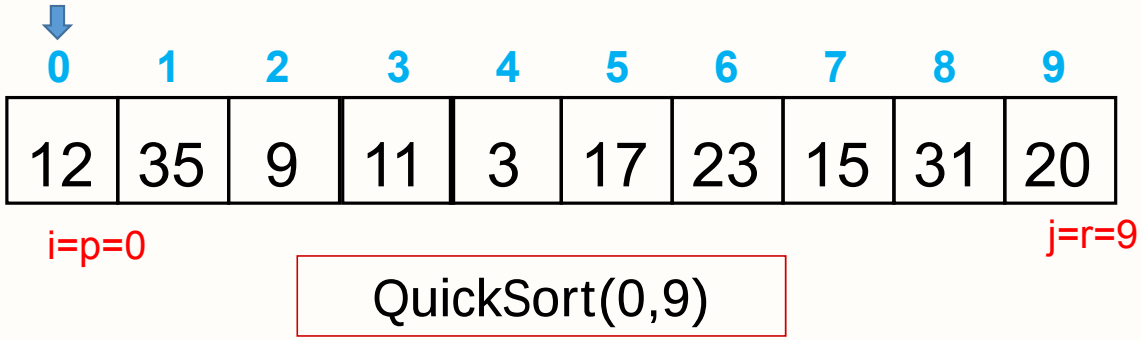
Pemilihan pivot

Pemilihan pivot menentukan apakah algoritma quick sort akan memberikan performa terbaik atau terburuk.

Cara pemilihan pivot

1. Pivot adalah elemen pertama, elemen terakhir, atau elemen tengah array. Cara ini hanya bagus jika elemen array tersusun secara acak, tetapi tidak bagus jika elemen array semula sudah terurut.
2. Pivot dipilih secara acak dari salah satu elemen array, Cara ini baik, tetapi belum tentu maksimal, sebab memerlukan prosedur khusus untuk menentukan pivot secara acak.
3. Pivot adalah elemen median array . Cara ini paling bagus, karena hasil partisi menghasilkan dua bagian array yang berukuran seimbang. Cara ini memberikan kompleksitas waktu yang minimum. Masalahnya, mencari median dari elemen array yang belum terurut adalah persoalan tersendiri.

x=pivot=A[p]=12



- X = PIVOT merupakan indeks ke –0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih besar dari atau sama dengan PIVOT.

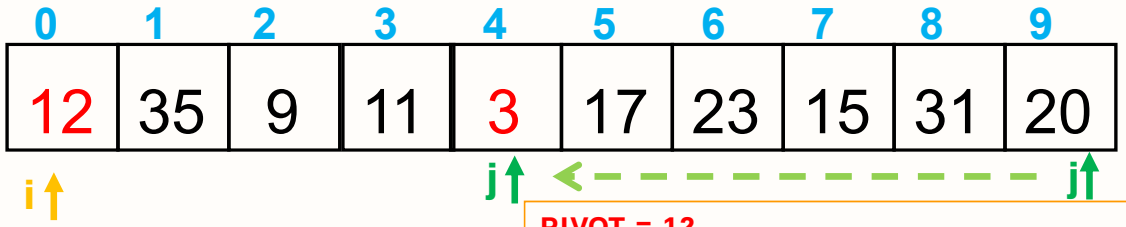
  Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari atau sama dengan PIVOT.

  Cara kerjanya : selama Data[j] > PIVOT maka nilai j dikurangi

```
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
```

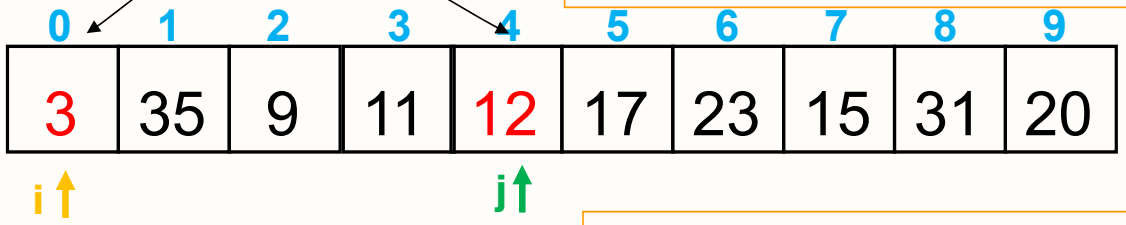
q = Partition(0,9)

PIVOT = 12
i = 1, j = 9



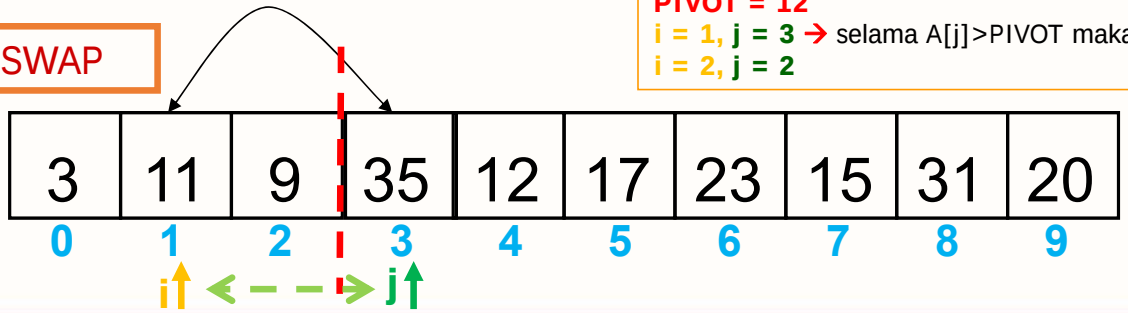
PIVOT = 12
i = 0, j = 9 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 0, j = 4
if i < j ? YES maka SWAP dan i++, j--

SWAP



PIVOT = 12
i = 1, j = 3 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 2, j = 2

SWAP



```
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
```

PIVOT = 12

i = 2 j = 2

i < j ? (NO) STOP LOOPING

Return j = 2

Q = Partisi = 2

3	11	9	35	12	17	23	15	31	20
0	1	2	3	4	5	6	7	8	9

i j

QuickSort(0,9)

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

```
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
```

QuickSort(0,2)

0	1	2	3	4	5	6	7	8	9
3	11	9	35	12	17	23	15	31	20

i j

PIVOT = 3

i = 0, j = 2 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++

i = 0, j = 0

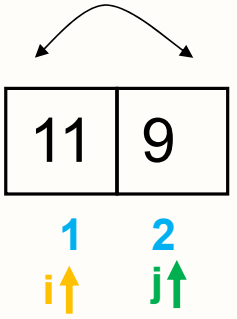
if i < j ? (NO) NO SWAP and STOP LOOPING

Return j = 0

Q = Partisi = 0

QuickSort(0,0) QuickSort(1,2)

QuickSort(1,2)

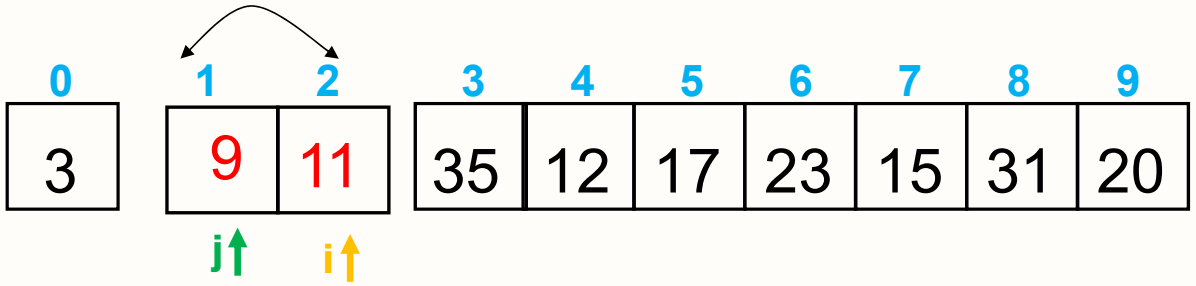


PIVOT = 11

i = 1, j = 2 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++

if i < j ? (YES) maka SWAP , j-- dan i++

SWAP



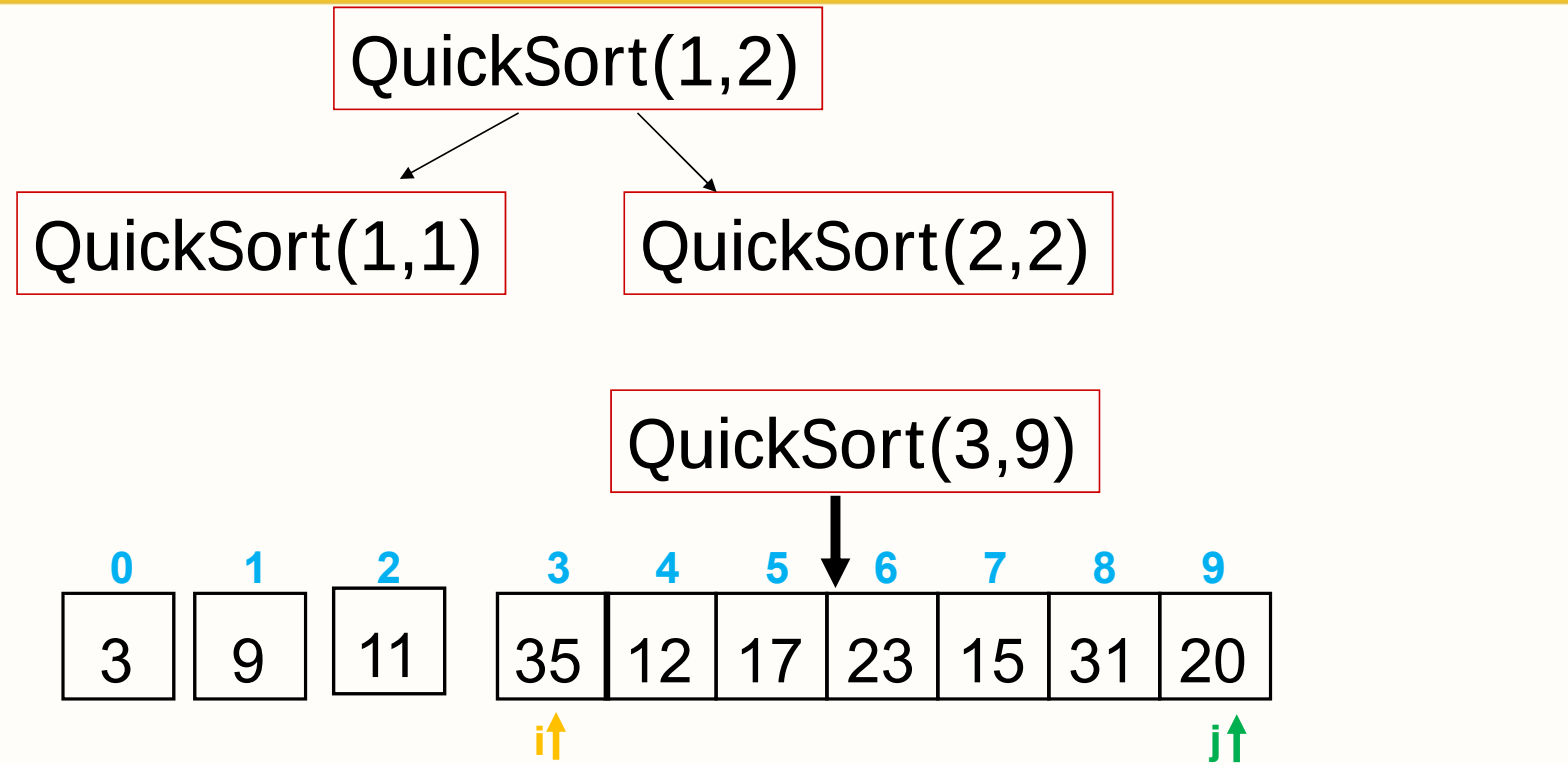
PIVOT = 11

i = 2, j = 1

if i < j ? (NO) STOP LOOPING

Return j = 1

Q = Partisi = 1



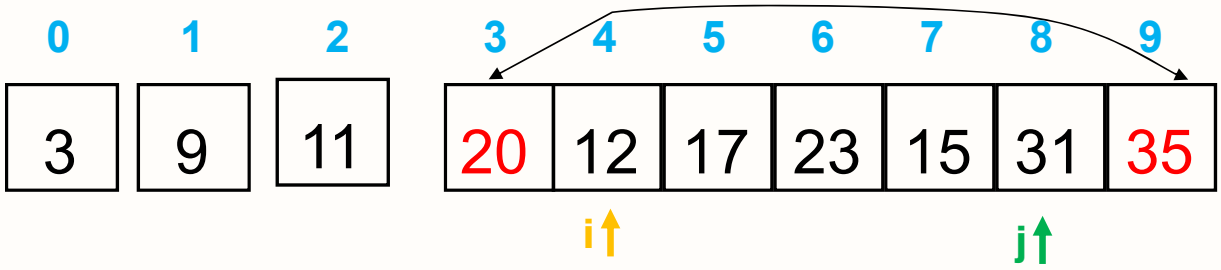
PIVOT = 35

i = 3, j = 9 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++

i = 3, j = 9

if i < j ? (YES) SWAP, i++ dan j—

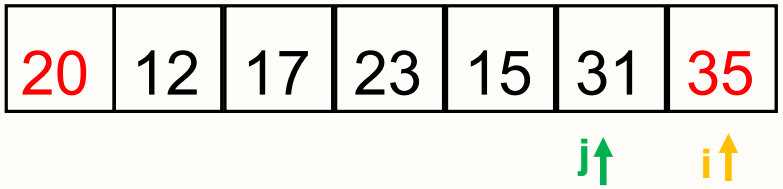
i = 4, j = 8



Setelah SWAP

PIVOT = 35

i = 4 j = 8, → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++



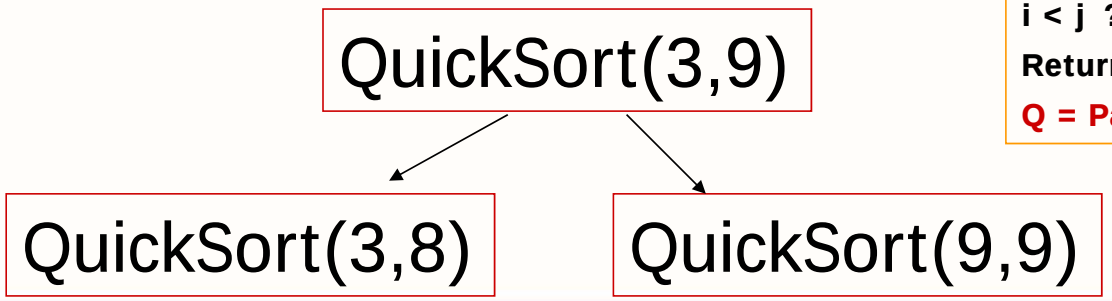
PIVOT = 35

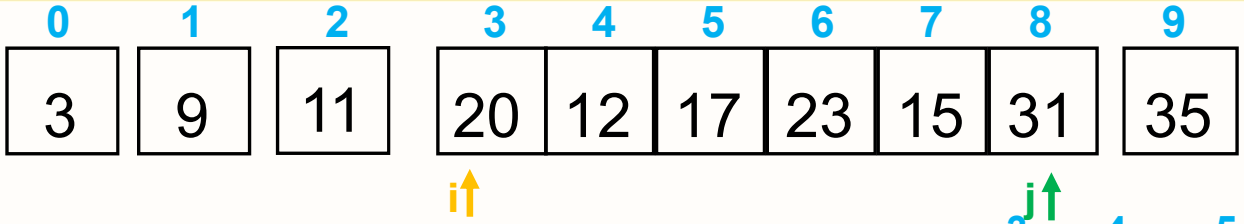
i = 9 j = 8

i < j ? (NO) NO SWAP and STOP LOOPING

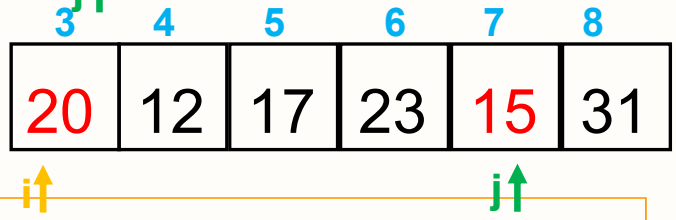
Return j = 8

Q = Partisi = 8



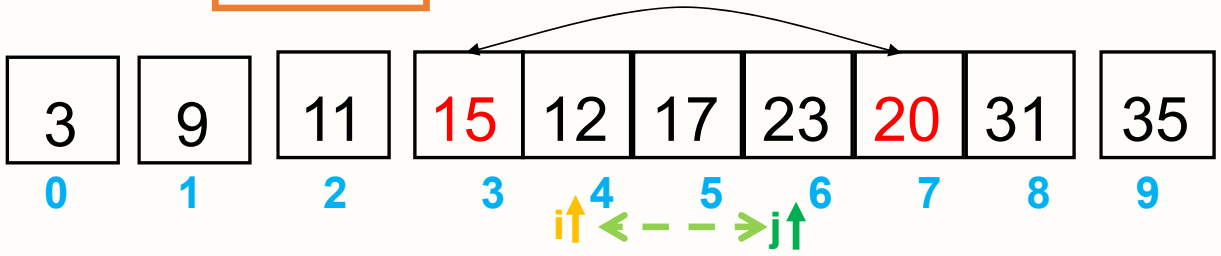


QuickSort(3,8)



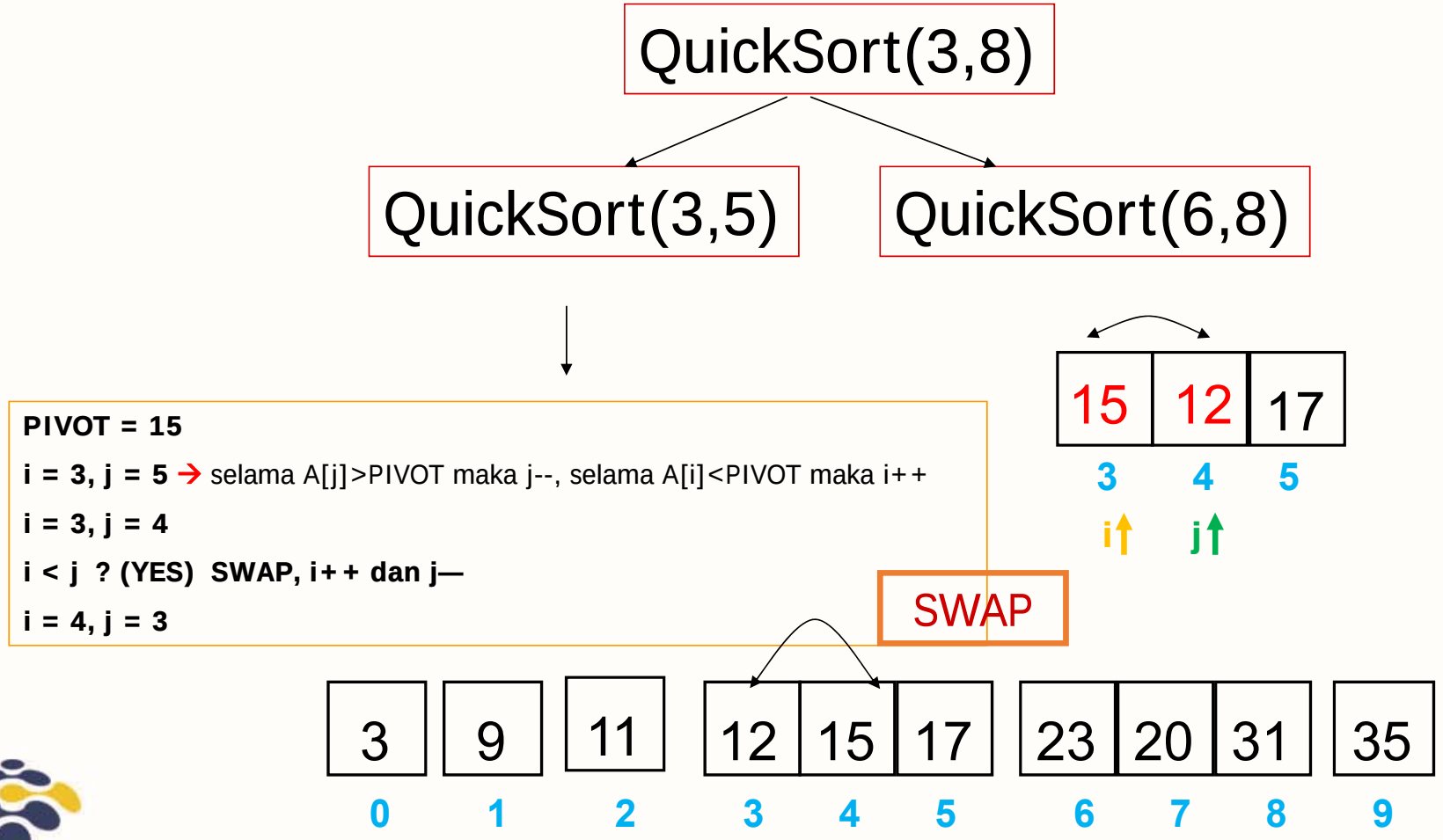
PIVOT = 20
i = 3, j = 8 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 3, j = 7
i < j ? (YES) SWAP, i++ dan j--

SWAP



Setelah SWAP
PIVOT = 20
i = 4, j = 6 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++

PIVOT = 20
i = 6, j = 5
i < j ? (NO), NO SWAP and STOP LOOPING
Return j = 5
Q = Partisi = 5


QuickSort(3,8)
QuickSort(3,5)
QuickSort(6,8)
PIVOT = 15
i = 3, j = 5 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 3, j = 4
i < j ? (YES) SWAP, i++ dan j—
i = 4, j = 3
15 12 17
3 4 5
i j
SWAP
3 9 11 12 15 17 23 20 31 35
0 1 2 3 4 5 6 7 8 9

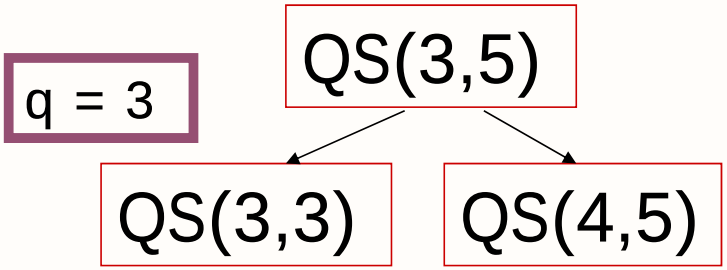
PIVOT = 15

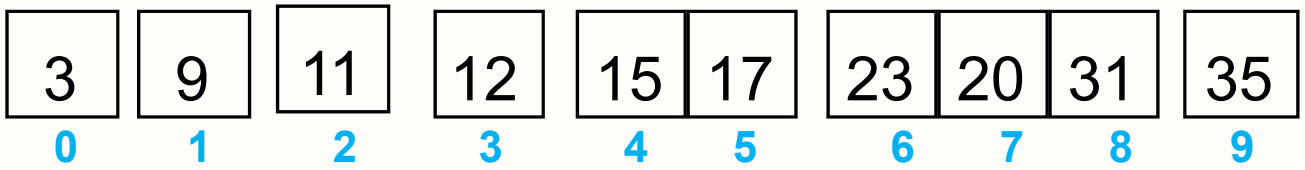
i = 4 j = 3

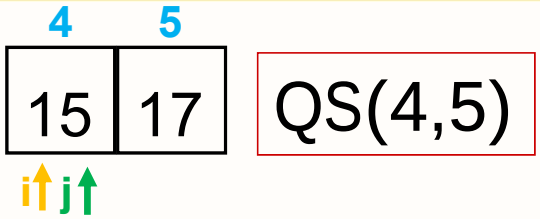
i < j ? (NO) NO SWAP and STOP LOOPING

Return j = 3

Q = Partisi = 3







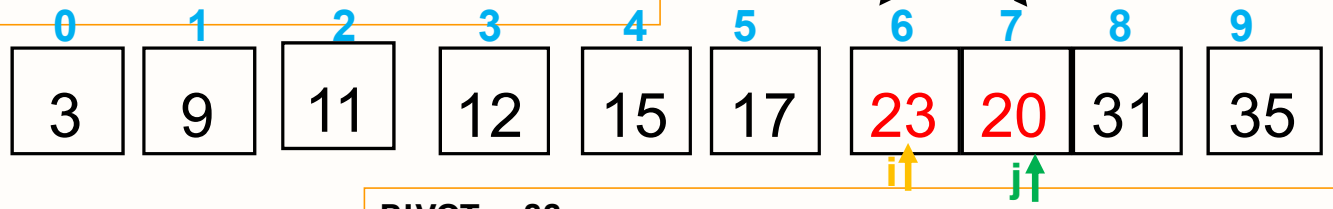
QS(4,5)

q = 4

QS(4,5)

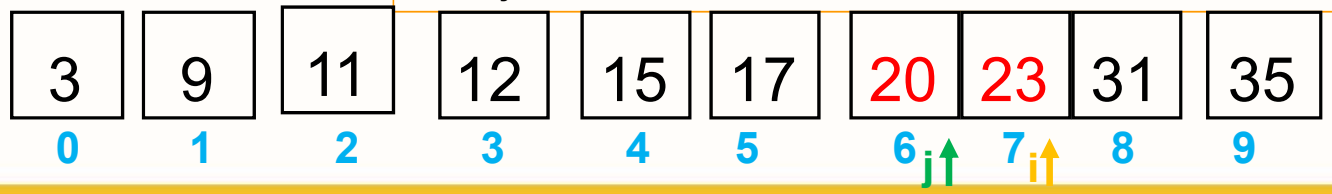
QS(4,4) QS(5,5)

PIVOT = 15
i = 4, j = 5 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 4, j = 4
i < j ? (NO) NO SWAP and STOP LOOPING
Return j = 4
Q = Partisi = 4

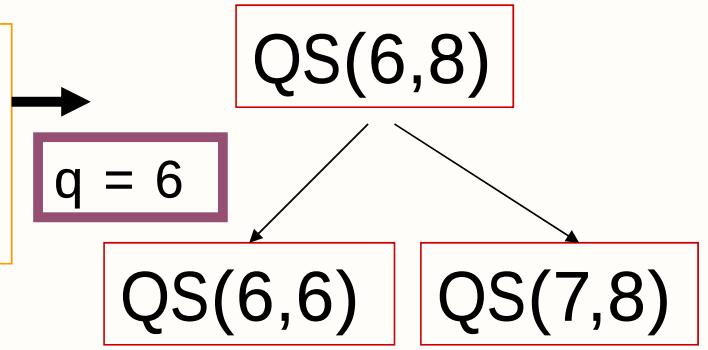


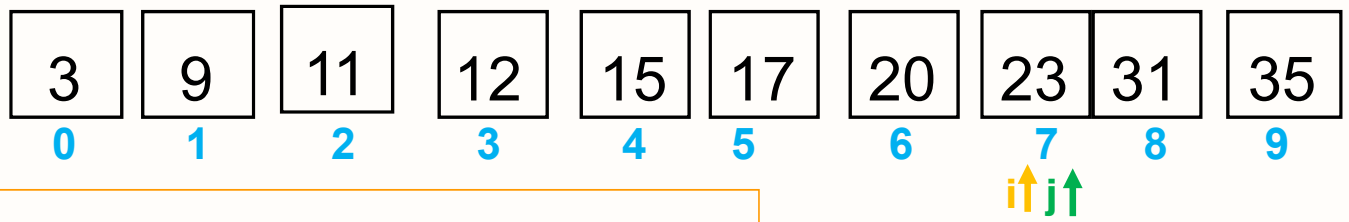
QuickSort(6,8)

PIVOT = 23
i = 6, j = 8 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 6, j = 7
i < j ? (YES) SWAP, i++ dan j—
i = 7, j = 6

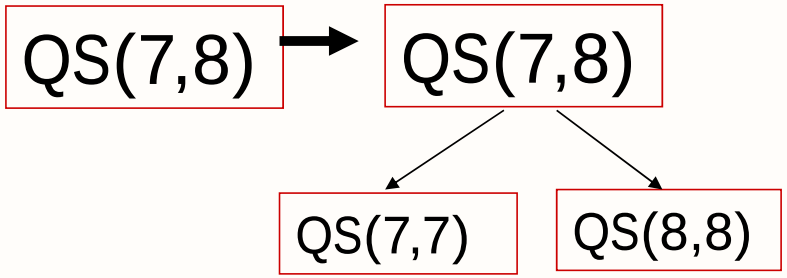


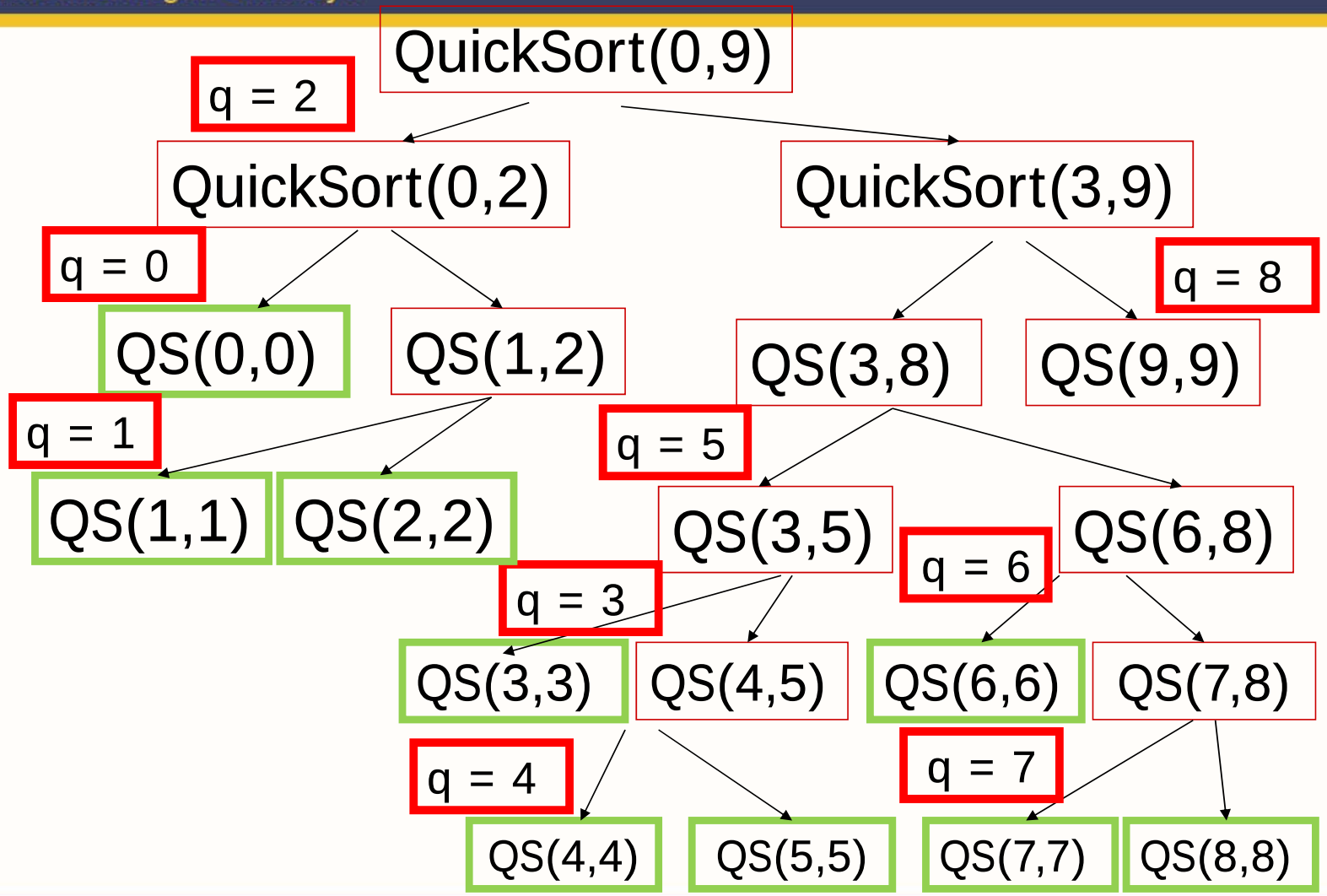
Setelah SWAP
PIVOT = 23
i = 7 j = 6
i < j ? (NO) NO SWAP and STOP LOOPING
Return j = 6
Q = Partisi = 6





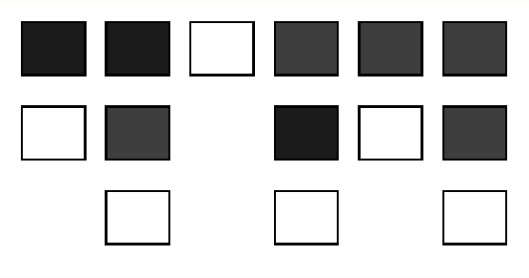
PIVOT = 23
i = 7, j = 8 → selama A[j]>PIVOT maka j--, selama A[i]<PIVOT maka i++
i = 7, j = 7
i < j ? (NO) NO SWAP and STOP LOOPING
Return j = 7
Q = Partisi = 7




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)

Quicksort Analysis

- Jika diasumsikan pivot diambil secara random
- **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, menjadi bagian array dengan elemen-elemen yang lebih kecil dan bagian array dengan elemen-elemen yang lebih besar dari pivot.
  - Bagian sebelah kiri pivot berisi elemennya lebih kecil dari pivot.
  - Bagian sebelah kanan pivot berisi elemennya 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 kosong

Quicksort Analysis

- **Best** case running time: **$O(n \log_2 n)$**
- **Average** case running time: **$O(n \log_2 n)$**
- **Worst** case running time: **$O(n^2)$**

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)

Merge sort Vs. Quick sort

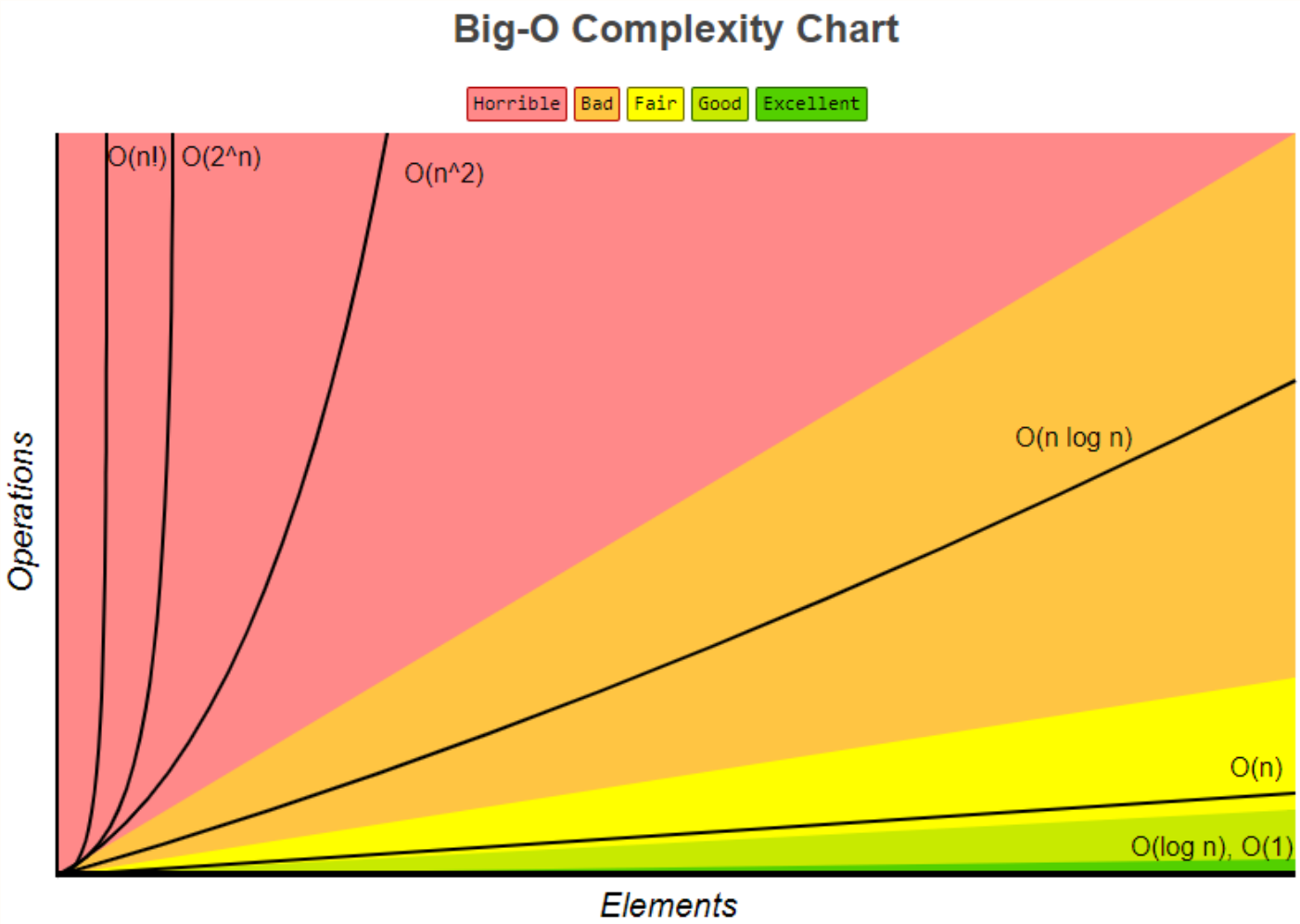
- Merge sort
  - Splits partitions in half
  - Merges smaller lists back into larger list
  - Requires overhead when sorting arrays
- Quick sort
  - Relies on a pivot point for sorting
  - Smaller sets are sorted based on pivot point
  - Can perform slowly if a bad pivot point is used

Merge sort Vs. Quick sort

- **Mergesort**
  - Use extra space
  - Is guaranteed to have O(n log n) performance in the worst case
- **Quicksort**
  - Does **not** use extra space
  - Is **not** guaranteed to have O(n log n) performance in the worst case, unlike merge sort

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Kesimpulan

Cara Kerja Quick Sort

- Tentukan “pivot”.
- Bagi data menjadi 2 Bagian yaitu:
  - Data kurang dari pivot dan
  - Data lebih besar dari pivot.
- Urutkan tiap bagian tersebut secara rekursif.