



# Divide and Conquer: Quicksort

Tim Olimpiade Komputer Indonesia

# Pengenalan

- Selain *merge sort*, ada algoritma pengurutan yang bekerja dalam  $O(N \log N)$ , salah satunya **quicksort**.
- *Quicksort* menggunakan prinsip *divide and conquer* dalam pengurutan.



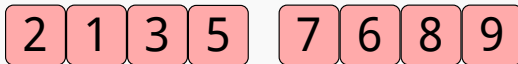
# Konsep

- Misalkan kita hendak mengurutkan *array* bilangan secara menaik.
- Pilih salah satu elemen, misalnya 5.



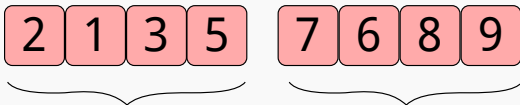
## Konsep (lanj.)

- Tempatkan seluruh elemen yang  $\leq 5$  di bagian kiri *array*, dan yang  $> 5$  di bagian kanan.
- Urutan elemennya setelah pemindahan tidak penting.



## Konsep (lanj.)

- Lakukan *quicksort* serupa untuk bagian kiri dan kanan secara rekursif.
- Suatu ketika, seluruh *array* menjadi terurut.



## Konsep (lanj.)

Jika dikaitkan dengan *divide and conquer*:

- *divide*: partisi *array* menjadi dua seperti yang dijelaskan sebelumnya.
- *conquer*: ketika *array* tinggal satu elemen, berarti sudah terurut.
- *combine*: tempelkan hasil *quicksort* bagian kiri dan kanan.



# Partisi

- Bagian utama dari *quicksort* adalah proses partisi (bagian *divide*).
- Sebelum melakukan partisi, pilih suatu elemen yang akan dijadikan *pivot* (= pijakan).
- Nantinya, akan dilakukan partisi supaya seluruh elemen yang  $\leq pivot$  berada di bagian kiri, dan yang  $> pivot$  di bagian kanan.
- Untuk saat ini, kita akan menggunakan elemen di tengah *array pivot*.



## Partisi (lanj.)

- Kini sudah ditentukan nilai *pivot*, bagaimana cara mempartisi *array*?
- Kita dapat menggunakan sebuah perulangan  $O(N)$  untuk menampung hasil partisi di suatu *array* sementara, lalu menempatkan kembali hasil partisi ke *array* sebenarnya.
- Namun cara ini agak merepotkan, kita perlu membuat *array* sementara dan memindahkan isi *array*.





# Partisi Hoare

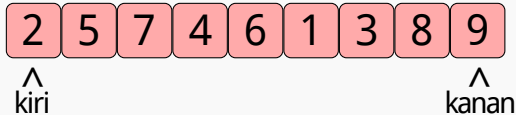
- Ada beberapa algoritma untuk melakukan partisi secara *in place*, yaitu tanpa *array* sementara.
- Kita akan menggunakan salah satunya, yaitu algoritma partisi **Hoare**.



## Partisi Hoare (lanj.)

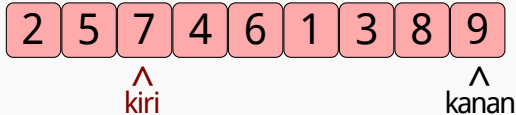
Misalkan  $pivot = 5$ .

Mulai dengan dua variabel penunjuk, *kiri* dan *kanan* di ujung-ujung *array*.



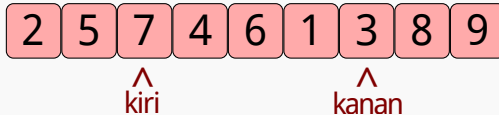
## Partisi Hoare (lanj.)

Gerakkan variabel *kiri* ke arah kanan, sampai elemen yang ditunjuk tidak  $<$  *pivot*



## Partisi Hoare (lanj.)

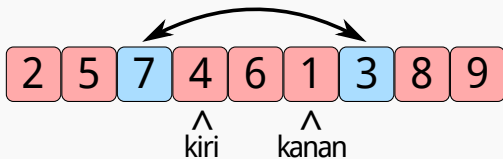
Gerakkan variabel *kanan* ke arah kiri, sampai elemen yang ditunjuk tidak  $>$  *pivot*



## Partisi Hoare (lanj.)

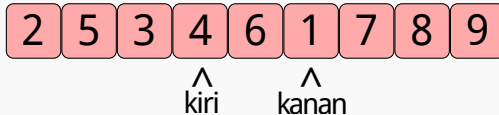
Tukar elemen yang ditunjuk *kiri* dan *kanan*, lalu gerakkan:

- *kiri* ke kanan satu langkah
- *kanan* ke kiri satu langkah



## Partisi Hoare (lanj.)

Karena  $kiri \leq kanan$ , artinya partisi belum selesai.  
Kita akan mengulangi hal yang serupa.



## Partisi Hoare (lanj.)

Gerakkan variabel *kiri*.



## Partisi Hoare (lanj.)

Gerakkan variabel *kanan*.

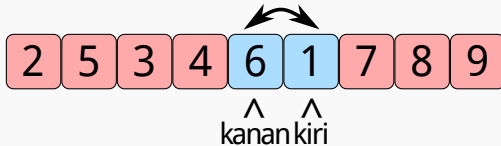
Kebetulan, elemen yang ditunjuk sudah tidak  $>$  *pivot*.





## Partisi Hoare (lanj.)

Tukar dan gerakkan variabel *kiri* dan *kanan* satu langkah.



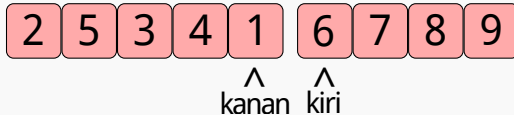
## Partisi Hoare (lanj.)

Kini sudah tidak  $kiri \leq kanan$ , artinya partisi selesai.



## Partisi Hoare (lanj.)

Perhatikan bahwa seluruh elemen yang  $\leq pivot$  berada di kiri, dan sisanya di kanan.



## Implementasi Partisi Hoare

```
PARTITION(arr[], left, right, pivot)  
1  pLeft = left  
2  pRight = right  
3  while pLeft ≤ pRight  
4      while arr[pLeft] < pivot  
5          pLeft = pLeft + 1  
6      while arr[pRight] > pivot  
7          pRight = pRight - 1  
8      if pLeft ≤ pRight  
9          swap(arr[pLeft], arr[pRight])  
10         pLeft = pLeft + 1  
11         pRight = pRight - 1
```



# Analisis Algoritma Partisi Hoare

- Terdapat dua variabel penunjuk, yang setiap langkahnya selalu bergerak ke satu arah tanpa pernah mundur.
- Algoritma berhenti ketika variabel *kiri* dan *kanan* bertemu.
- Artinya, setiap elemen *array* dikunjungi tepat satu kali.
- Kompleksitasnya adalah  $O(N)$ .



## Integrasi ke Quicksort

Setelah kita mengimplementasikan algoritma partisi, mengintegrasikan ke *quicksort* cukup mudah.

```
QUICKSORT(arr[], left, right)
```

```
1  if left  $\geq$  right
2      // Tidak ada elemen yang perlu diurutkan
3  else
4      pivot = arr[(left + right) div 2]

5      // ... sisipkan isi algoritma Hoare di sini ...
6      // Sampai saat ini, dipastikan pRight < pLeft

7      QUICKSORT(left, pRight)
8      QUICKSORT(pLeft, right)
```



# Analisis Algoritma Quicksort

- Pada setiap kedalaman rekursi, *array* hasil partisi belum tentu memiliki ukuran yang sama.
- Hasil partisi bergantung pada nilai pivot yang kita pilih.
- Kita anggap dulu hasil partisi selalu membelah *array* menjadi dua *subarray* sama besar.



# Analisis Algoritma Quicksort (lanj.)

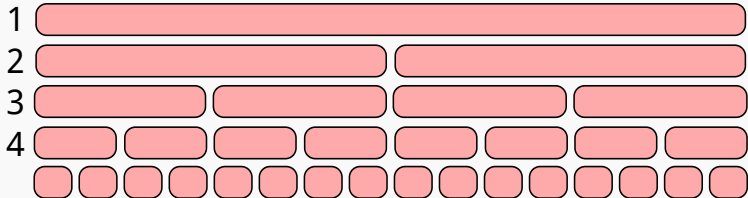
- Ternyata analisisnya mirip seperti *merge sort*.
- Kompleksitasnya adalah  $O(N \log N)$





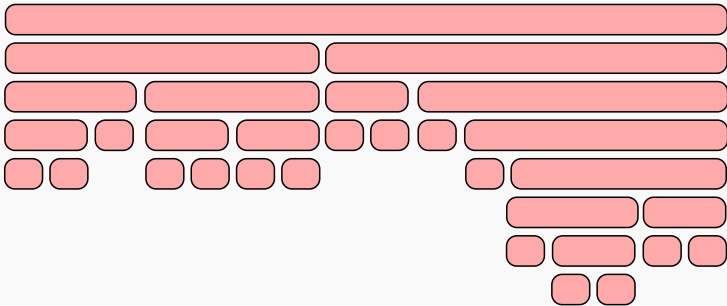
# Analisis Algoritma Quicksort: Kasus Terbaik

- Pembelahan menjadi dua *subarray* sama besar menjamin kedalaman rekursif **sedangkal mungkin**.
- Sehingga untuk kasus terbaik, jalannya algoritma menjadi seperti *merge sort* dan bekerja dalam  $O(N \log N)$ .



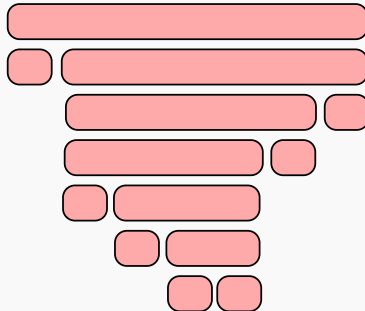
# Analisis Algoritma Quicksort: Kasus Rata-Rata

- Pada kebanyakan kasus, ukuran hasil partisi berbeda.
- Secara rata-rata kompleksitasnya masih dapat dianggap  $O(N \log N)$ .



# Analisis Algoritma Quicksort: Kasus Terburuk

- Kasus paling buruk: ukuran hasil partisi sangat timpang.
- Akibatnya, kedalaman rekursif mendekati  $N$ .
- Kompleksitasnya menjadi  $O(N^2)$ .



## Analisis Algoritma Quicksort (lanj.)

- Tidak perlu khawatir, peluang terjadinya kasus terburuk sangat-sangat-amat-lah kecil.
- Artinya, pada sebagian besar kasus, *quicksort* akan berjalan dengan sangat cepat.



# Pemilihan Pivot

Terdapat beberapa strategi pemilihan *pivot* untuk mencegah hasil partisi yang terlalu timpang:

- Pilih salah satu elemen secara acak, ketimbang selalu memilih elemen di tengah.
- Pilih median dari elemen paling depan, tengah, dan paling belakang.



# Stable Sort

- *Quicksort* memiliki sifat **tidak stable**.
- Artinya jika dua elemen  $a_1$  dan  $a_2$  memenuhi:
  - memiliki yang nilai sama, dan
  - sebelum diurutkan  $a_1$  terletak sebelum  $a_2$ ,maka setelah diurutkan **tidak** dijamin  $a_1$  tetap terletak sebelum  $a_2$ .



# Penutup

- Terdapat jiwa *divide and conquer* pada algoritma *quicksort*.
- Kita telah mempelajari algoritma pengurutan yang efisien lainnya.

