Divide and Conquer

Tim Olimpiade Komputer Indonesia

Perkenalan

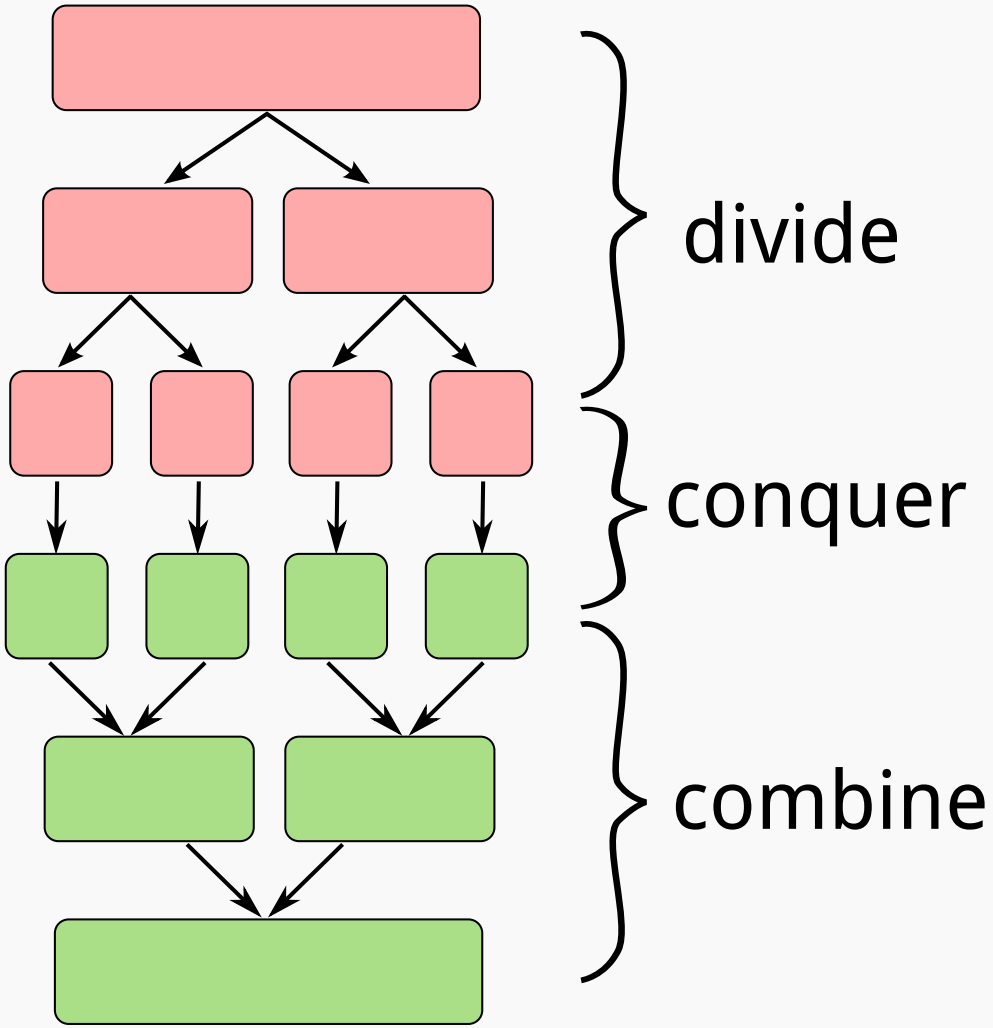
- Terkadang, permasalahan lebih mudah diselesaikan jika dibagi menjadi beberapa masalah yang lebih kecil.
- Masalah lebih kecil kemudian diselesaikan secara independen.
- Hasilnya digabungkan menjadi solusi untuk masalah yang lebih besar.
- Strategi ini digunakan Belanda pada masa penjajahan, biasa dipelajari pada sejarah sebagai *devide et impera*.

Konsep

Secara umum, **divide and conquer** terdiri dari tiga tahap:

- **divide**: membagi masalah yang besar menjadi masalah-masalah yang lebih kecil.
- **conquer**: ketika sebuah masalah sudah cukup kecil untuk diselesaikan, langsung selesaikan.
- **combine**: menggabungkan solusi dari masalah-masalah yang lebih kecil menjadi solusi untuk masalah yang besar.

Ilustrasi Konsep



Studi Kasus 1: Merge Sort

Merge sort: algoritma pengurutan $O(N \log N)$.

Prinsip kerja algoritma ini adalah:

- *divide*: jika *array* yang akan diurutkan berukuran besar, bagi menjadi dua *array* sama besar.
- *conquer*: ketika *array* sudah cukup kecil untuk diurutkan, lakukan pengurutan.
- *combine*: dari dua *array* yang telah terurut, gabungkan menjadi sebuah *array* terurut.

Contoh Eksekusi Merge Sort

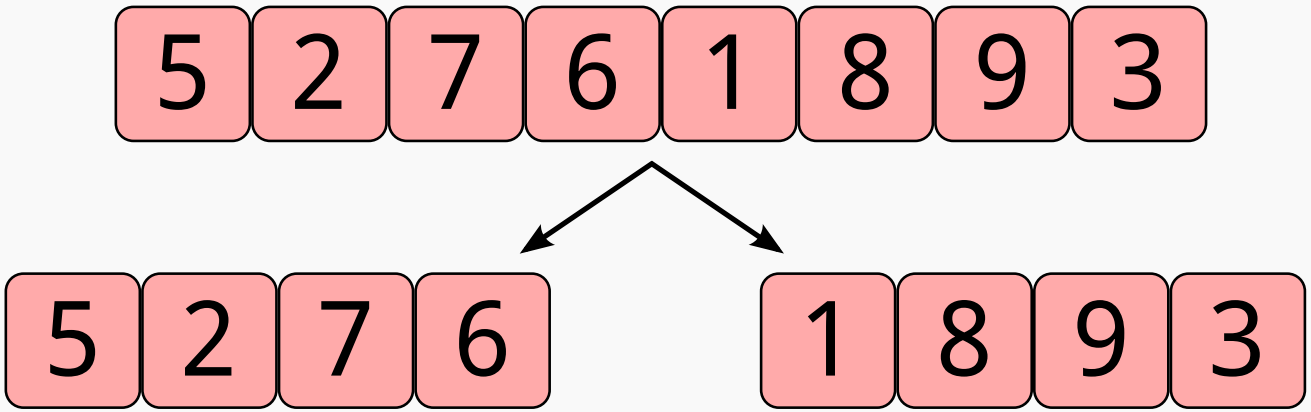
Kapan suatu *array* dianggap cukup kecil untuk dapat diurutkan secara langsung?

- Sederhana, yaitu ketika panjang *array* itu tinggal *satu*.
- Tentu saja, *array* dengan panjang satu sudah pasti terurut.

Jadi selama *array* yang hendak diurutkan masih memiliki panjang lebih dari satu, bagi *array* itu menjadi dua.

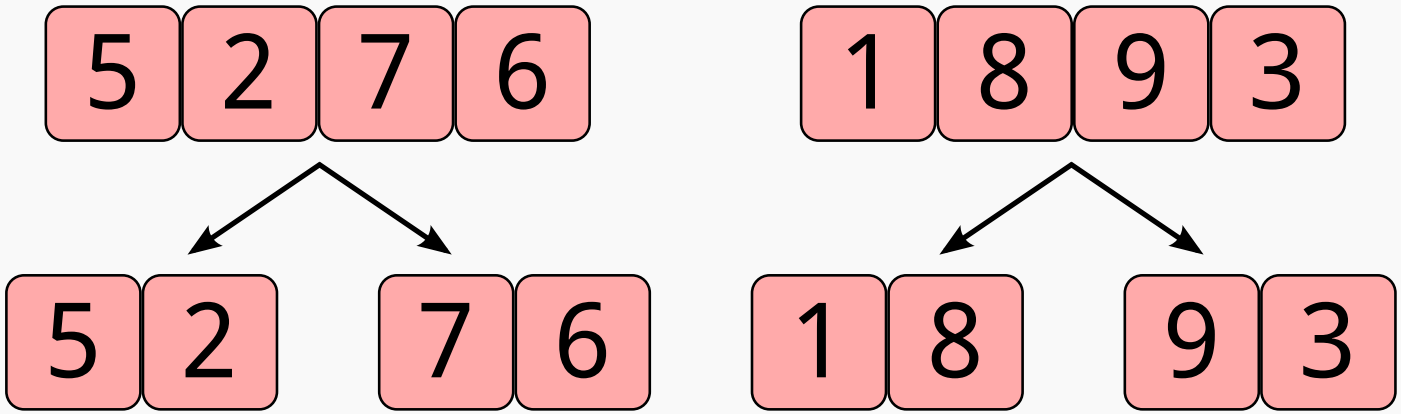
Contoh Eksekusi Merge Sort (lanj.)

Array yang dimiliki masih terlalu besar, belah menjadi dua.



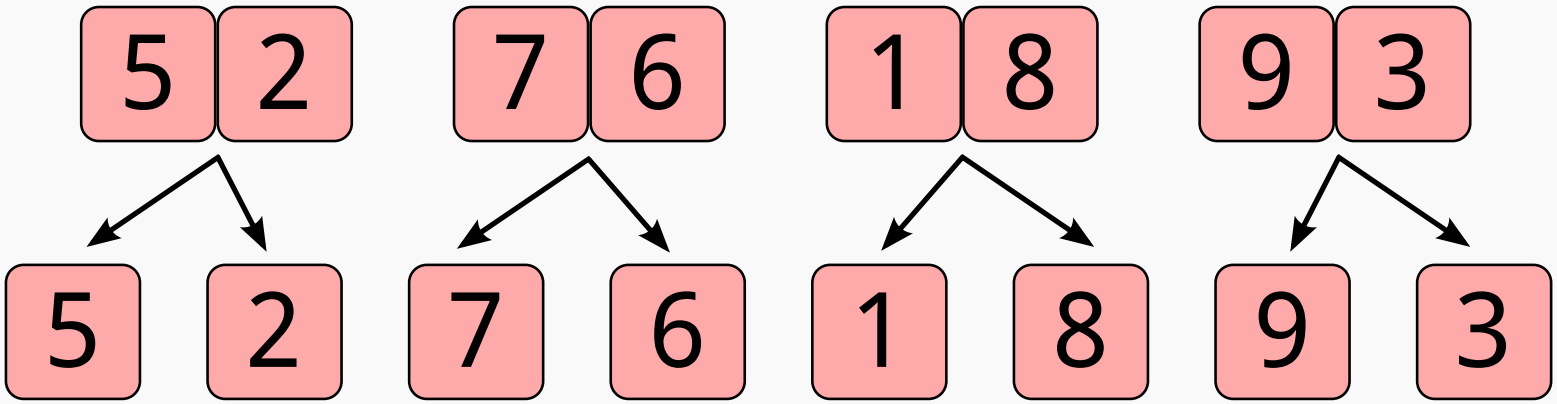
Contoh Eksekusi Merge Sort (lanj.)

Masih belum cukup kecil, belah lagi.



Contoh Eksekusi Merge Sort (lanj.)

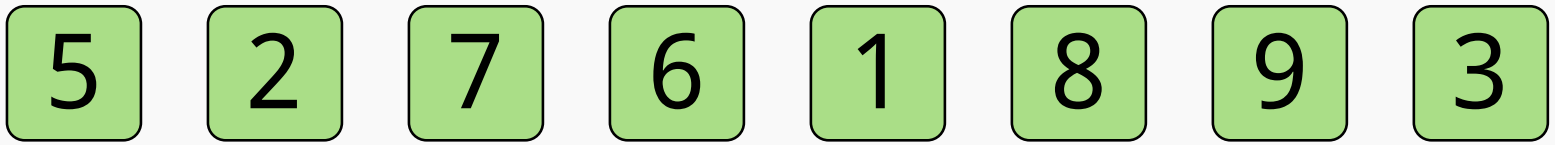
Masih belum cukup kecil, belah lagi.



Contoh Eksekusi Merge Sort (lanj.)

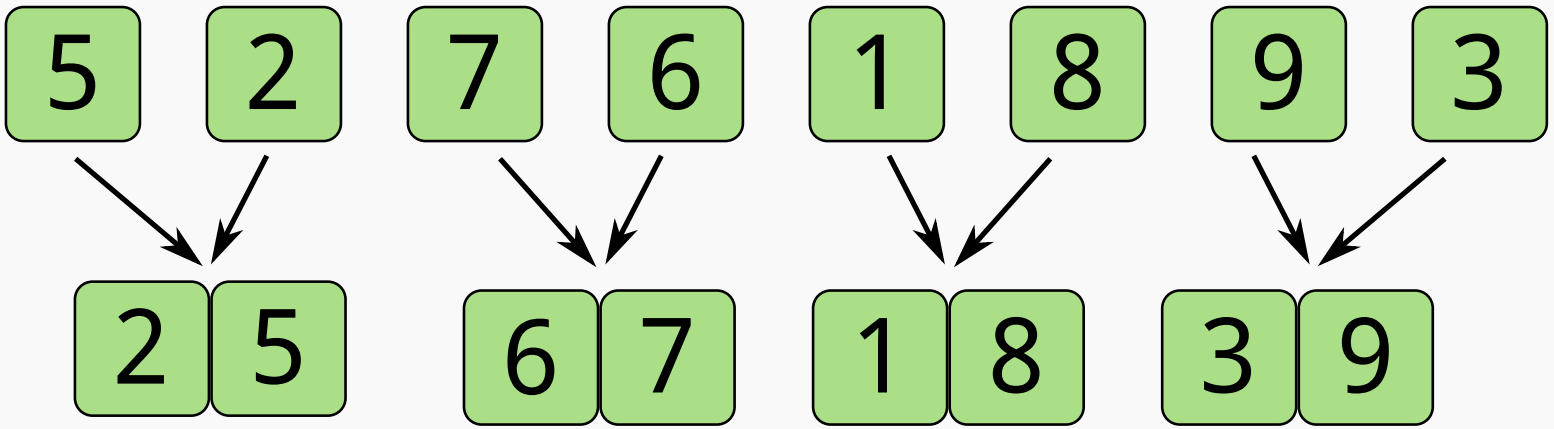
Kini kita memiliki *array-array* yang panjangnya hanya satu.

Secara definisi, masing-masing *array* telah terurut.



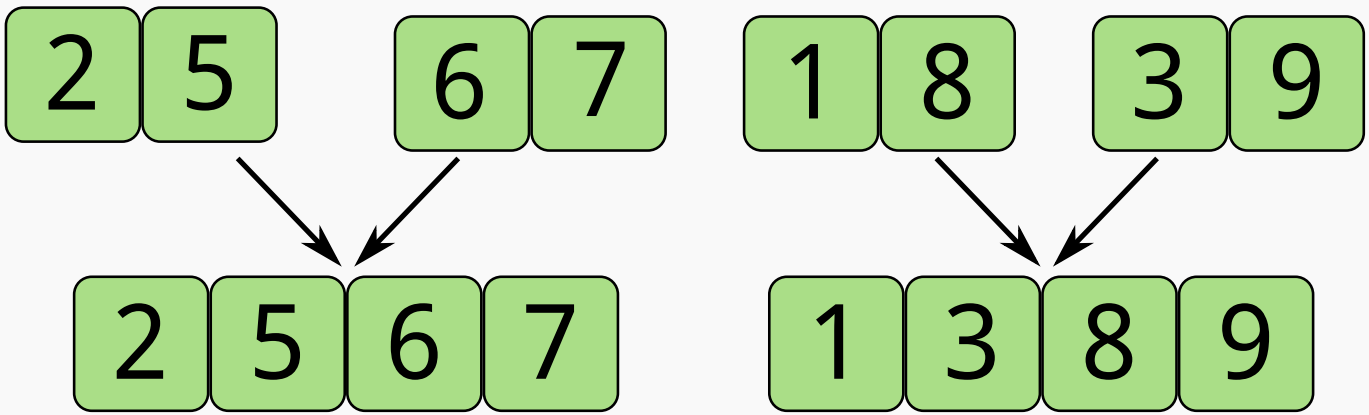
Contoh Eksekusi Merge Sort (lanj.)

Gabungkan hasil pembelahan sebelumnya.



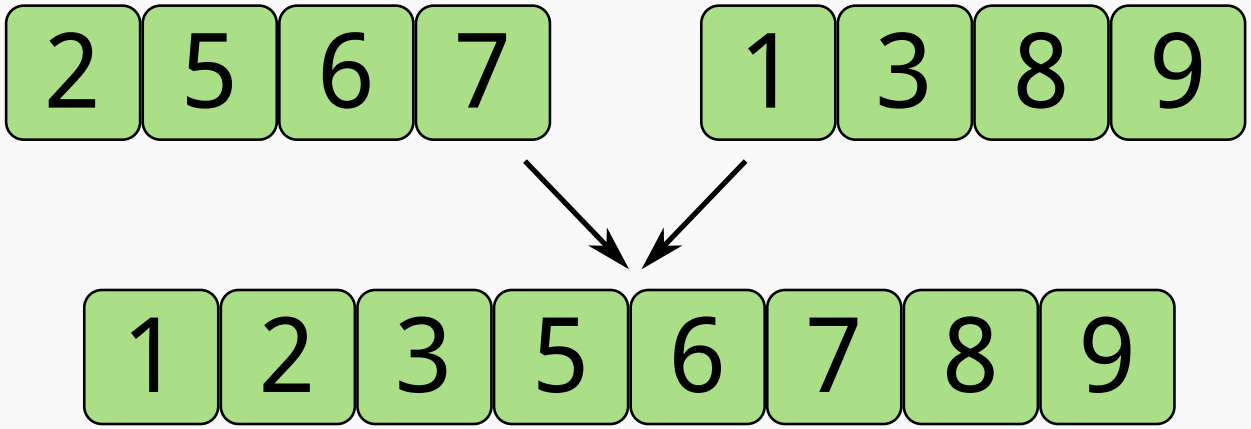
Contoh Eksekusi Merge Sort (lanj.)

Gabungkan lagi.



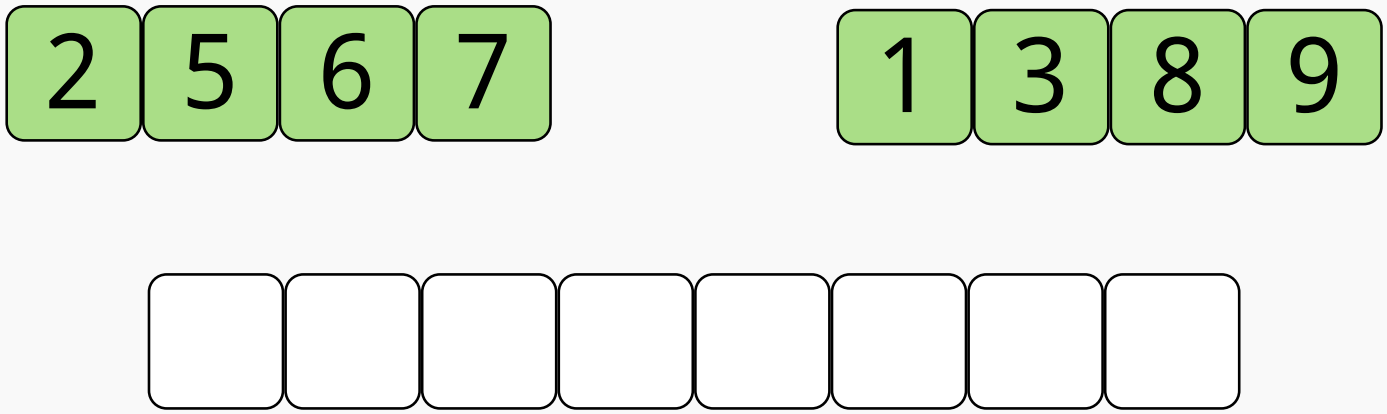
Contoh Eksekusi Merge Sort (lanj.)

Gabungkan lagi dan akhirnya didapatkan *array* terurut.



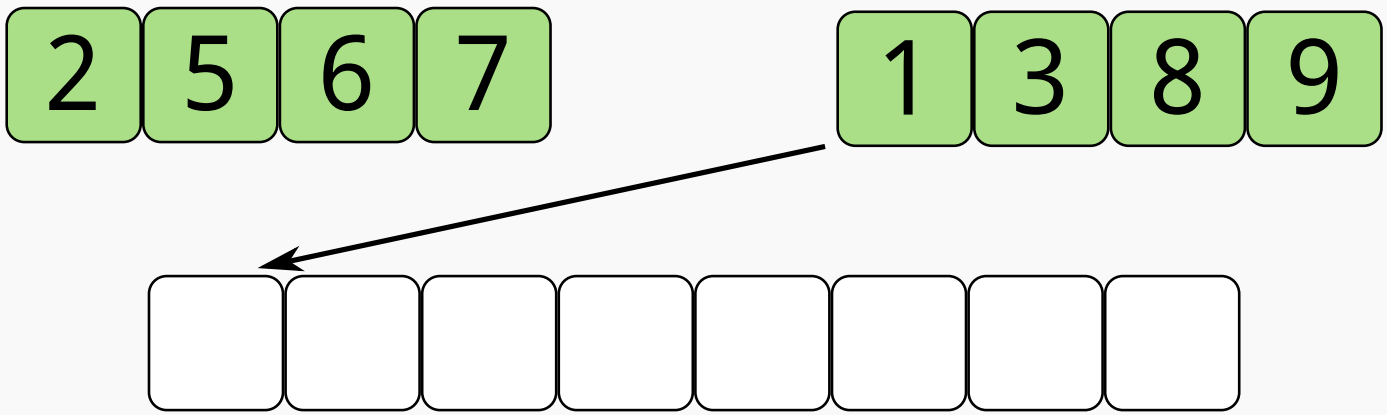
Menggabungkan Dua Array Terurut

Bagaimana cara menggabungkan dua *array* yang telah terurut?



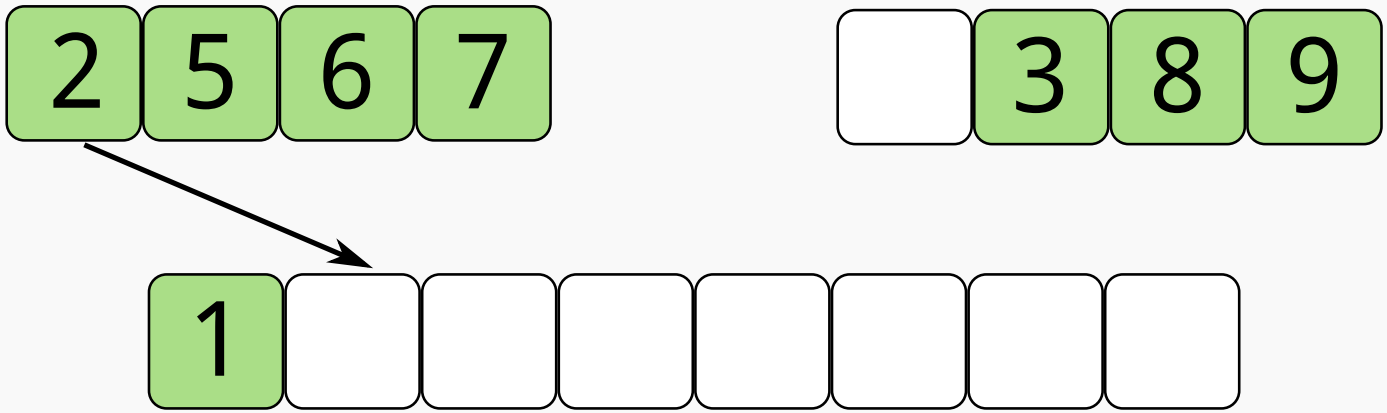
Menggabungkan Dua Array Terurut (lanj.)

Observasi: elemen terkecil dari *array* gabungan pasti salah satu dari elemen terkecil *array* yang terurut. Lebih tepatnya, yang memiliki nilai lebih kecil.

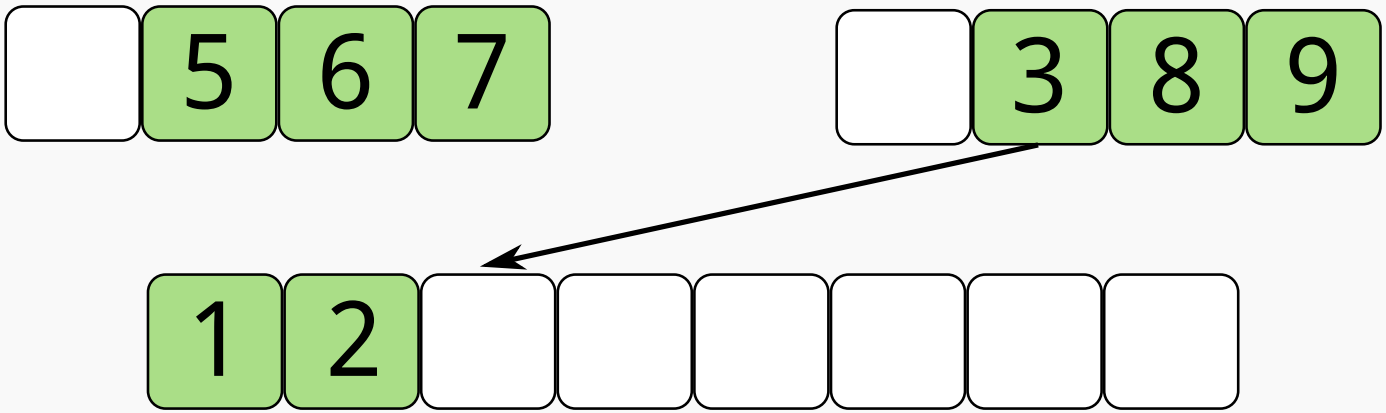


Menggabungkan Dua Array Terurut (lanj.)

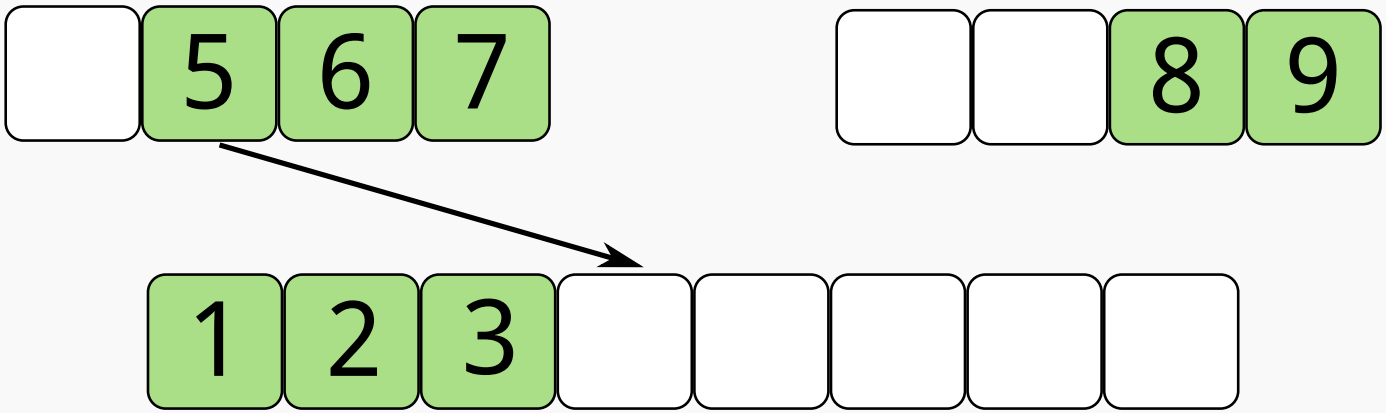
Ulangi hal serupa sampai salah satu atau kedua *array* habis.



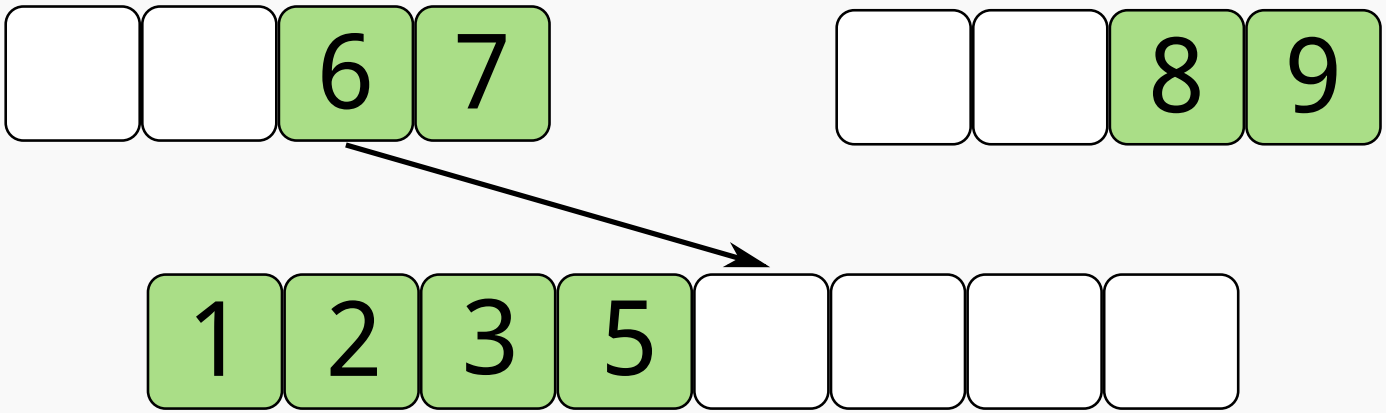
Menggabungkan Dua Array Terurut (lanj.)



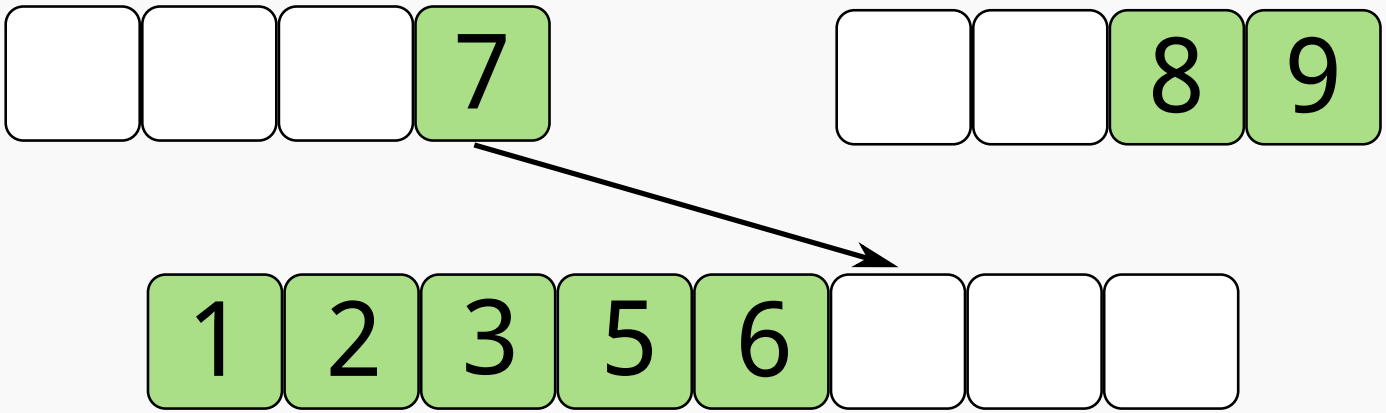
Menggabungkan Dua Array Terurut (lanj.)



Menggabungkan Dua Array Terurut (lanj.)

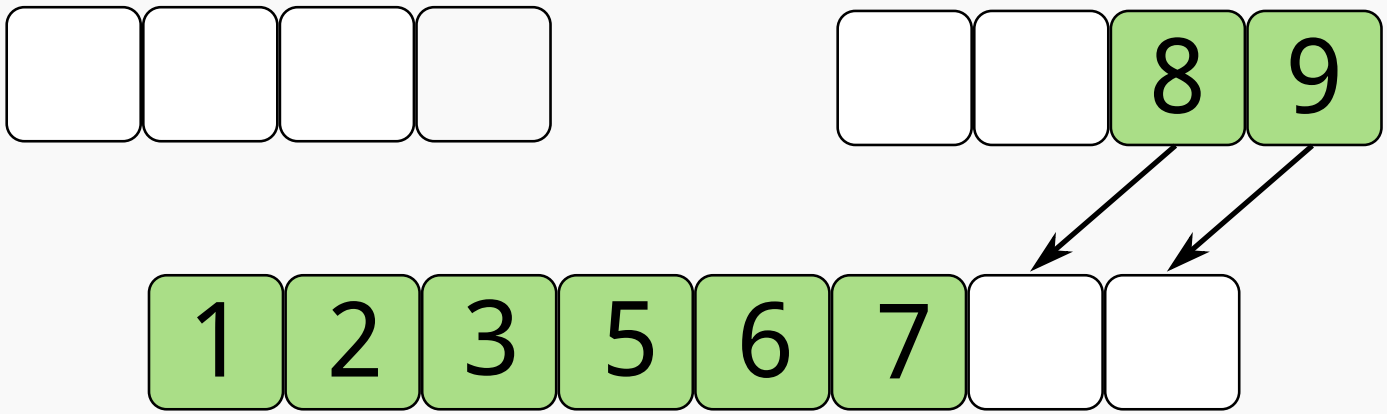


Menggabungkan Dua Array Terurut (lanj.)



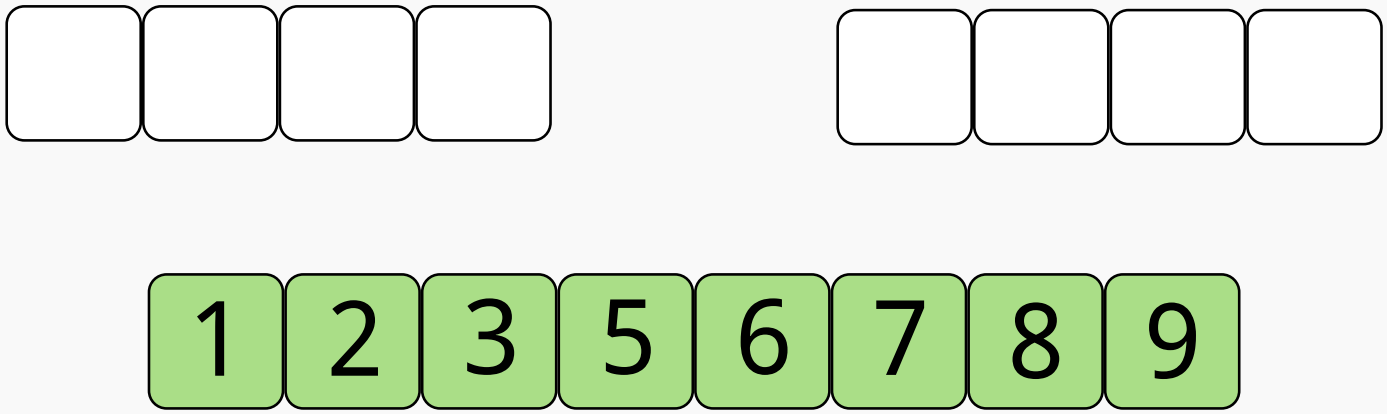
Menggabungkan Dua Array Terurut (lanj.)

Ketika salah satu *array* telah habis, *array* yang masih bersisa tinggal ditempelkan di akhir *array* gabungan.



Menggabungkan Dua Array Terurut (lanj.)

Selesai proses menggabungkan.



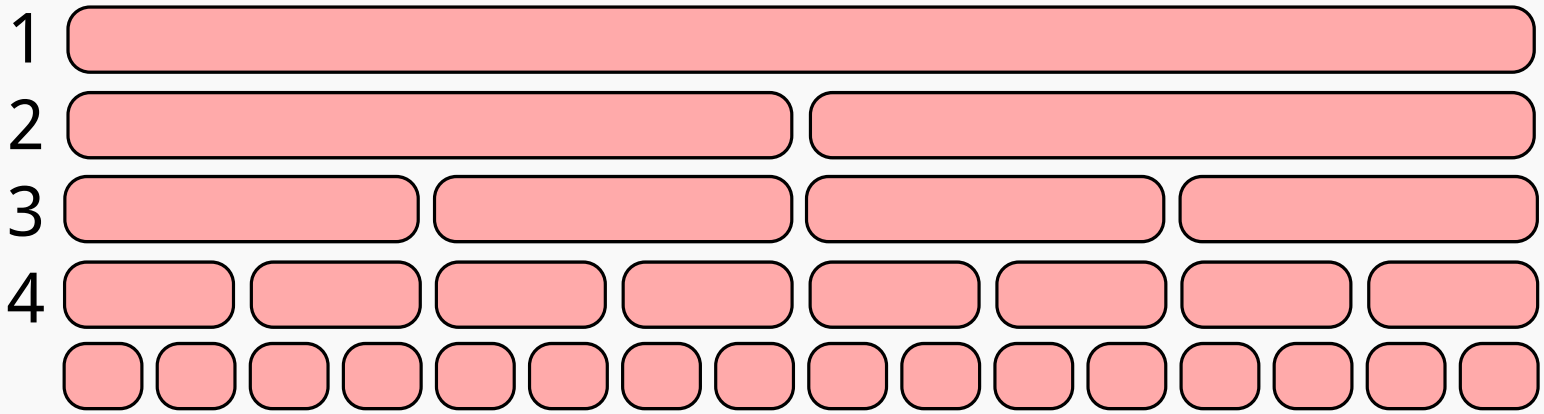
Analisis Menggabungkan Dua Array Terurut

- Misalkan kedua *array* terurut yang akan digabung adalah A dan B.
- Pada setiap langkah, salah satu dari elemen A atau B dipindahkan.
- Total terdapat $|A| + |B|$ proses, sehingga kompleksitasnya $O(|A| + |B|)$.

Analisis Algoritma Merge Sort

- Misalkan N menyatakan ukuran dari *array*.
- Dengan sifat membagi dua secara terus-menerus, kedalaman rekursif dari *merge sort* adalah $O(\log N)$
- Untuk setiap kedalaman, dilakukan aktivitas *divide* dan *combine*.
- Proses *divide* dan proses *conquer* selalu bekerja dalam $O(1)$.

Analisis Algoritma Merge Sort (lanj.)



- Kedalaman 1, proses *combine* bekerja dalam $O(2 \times \frac{N}{2})$
- Kedalaman 2, proses *combine* bekerja dalam $O(4 \times \frac{N}{4})$
- Kedalaman 3, proses *combine* bekerja dalam $O(8 \times \frac{N}{8})$
- ...

Analisis Algoritma Merge Sort (lanj.)

- Mudah untuk disadari bahwa keseluruhan proses untuk setiap kedalaman bekerja dalam $O(N)$.
- Karena kedalaman maksimal adalah $O(\log N)$, kompleksitas akhir *merge sort* adalah $O(N \log N)$.
- Jauh lebih cepat dari algoritma pengurutan $O(N^2)$ seperti *bubble sort*.
- *Merge sort* mampu mengurutkan *array* dengan ratusan ribu elemen dalam waktu singkat.

Contoh Implementasi

Mengurutkan $arr[left..right]$:

---

mergeSort($arr[], left, right$)

```
1   if left == right
2         // Tinggal 1 elemen, sudah pasti terurut
3   else
4         mid = (left + right) div 2
5         mergeSort(arr, left, mid)
6         mergeSort(arr, mid + 1, right)
7         merge(arr, left, mid, mid + 1, right)
```

Contoh Implementasi (lanj.)

Menggabungkan $arr[aLeft..aRight]$ dengan $arr[bLeft..bRight]$ yang telah terurut:

MERGE($arr[], aLeft, aRight, bLeft, bRight$)

```
1  // Buat array penampungan sementara bernama temp
2  // Isikan temp[aLeft .. bRight] dengan nilai dari arr[aLeft .. bRight]
3  tIndex = 1
4  aIndex = aLeft
5  bIndex = bLeft
6  ...
```

Contoh Implementasi (lanj.)

```
 5   ...
 6   // Selama kedua subarray masih ada isinya, ambil yang terkecil
 7   while (aIndex ≤ aRight) and (bIndex ≤ bRight)
 8         if temp[aIndex] < temp[bIndex]
 9               arr[tIndex] = temp[aIndex]
10               aIndex = aIndex + 1
11         else
12               arr[tIndex] = temp[bIndex]
13               bIndex = bIndex + 1
14         tIndex = tIndex + 1
15   ...
```

Contoh Implementasi (lanj.)

```
14  ...
15  // Masukkan subarray yang masih bersisa
16  // Hanya salah satu dari kedua while ini yang akan dieksekusi
17  while (aIndex ≤ aRight)
18        arr[tIndex] = temp[aIndex]
19        aIndex = aIndex + 1
20        tIndex = tIndex + 1
21  while (bIndex ≤ bRight)
22        arr[tIndex] = temp[bIndex]
23        bIndex = bIndex + 1
24        tIndex = tIndex + 1
25  // selesai penggabungan
```

Catatan Tentang Merge Sort

- *Merge sort* memiliki sifat **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 a_1 tetap terletak sebelum a_2.

Studi Kasus 2: Mencari Nilai Terbesar

- Diberikan sebuah *array* A yang memiliki N bilangan.
- Cari nilai terbesar yang ada pada A!

- Masalah ini mudah diselesaikan dengan perulangan biasa.
- Namun, coba kita selesaikan dengan *divide and conquer*.

Studi Kasus 2: Mencari Nilai Terbesar (lanj.)

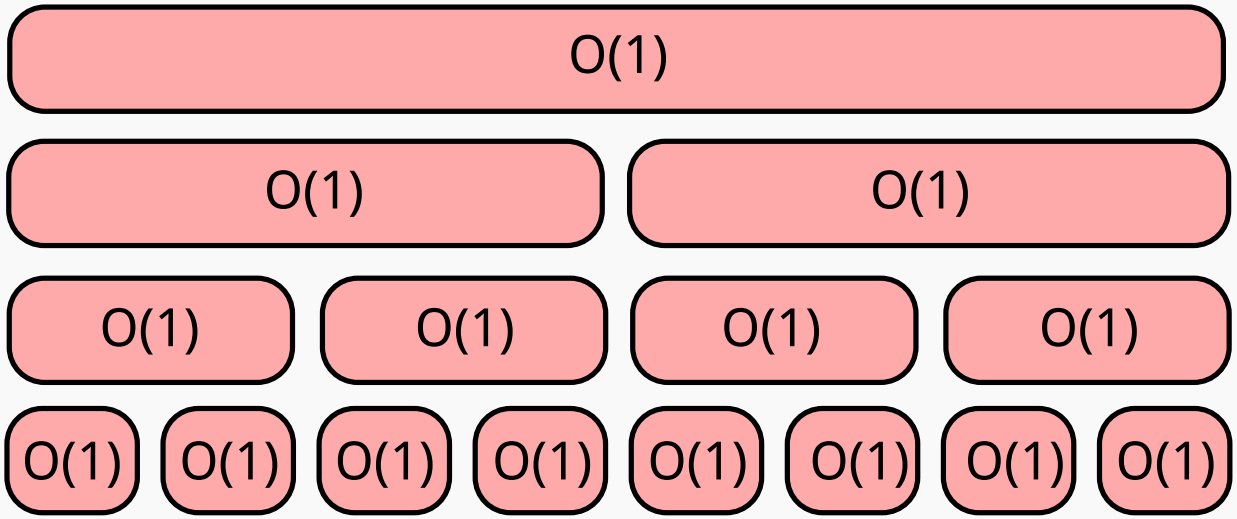
Pertama kita definisikan tahap-tahapnya:

- *Divide*: jika *array* berukuran besar, bagi menjadi dua *subarray*.
- *Conquer*: ketika *array* hanya berisi satu elemen, nilai terbesarnya pasti elemen tersebut.
- *Combine*: nilai terbesar dari *array* adalah maksimum dari nilai terbesar di *subarray* pertama dan nilai terbesar di *subarray* kedua.

Contoh Implementasi

```
FINDMAX(arr[], left, right)
1   if left == right
2         return arr[left]
3   else
4         mid = (left + right) div 2
5         leftMax = FINDMAX(arr, left, mid)
6         rightMax = FINDMAX(arr, mid + 1, right)
7         return max(leftMax, rightMax)
```

Analisis Algoritma Mencari Nilai Terbesar



- Setiap operasi *divide*, *conquer*, dan *combine* bekerja dalam $O(1)$.
- Ketiga operasi tersebut dilaksanakan sebanyak $1 + 2 + 4 + 8 + ... + 2^L$ kali, dengan 2^L mendekati N.
- Sehingga totalnya dilaksanakan sekitar $2^{L+1} - 1 = 2N$ operasi.

Analisis Algoritma Mencari Nilai Terbesar (lanj.)

- Kompleksitas akhirnya adalah $O(N)$.
- Ternyata, strategi ini tidak lebih baik dari mencari nilai maksimum satu per satu.
- Kesimpulannya, *divide and conquer* tidak selalu dapat mengurangi kompleksitas solusi naif.

Penutup

- *Divide and conquer* merupakan salah satu strategi dalam penyelesaian masalah.
- Konsep ini banyak digunakan pada struktur data lanjutan dan geometri komputasional.
- Jika suatu masalah dapat dibelah menjadi beberapa masalah yang lebih kecil dan serupa, kemudian hasil dari masing-masing penyelesaiannya dapat digabungkan, maka *divide and conquer* dapat digunakan.