studywithstudent

ANALISIS & STRATEGI ALGORITMA

Ibu Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

#1 PENGANTAR

youtube.com/studywithstudentkuy

Beberapa Hal ?

01 Saya dapat di Semester 3 (Informatika)

02 Lanjutan dari Matkul DDP (Algoritma 1) & Algoritma 2

03 Full Teori (tidak ada praktikum)

04 Terdapat rumus, perhitungan dan pengujian

youtube.com/studywithstudentkuy

Pra-Syarat ?

01 Sudah finish DDP (Algoritma 1) & Algoritma 2

02 Paham notasi algoritma Kalimat Deskriptif, FlowChart & Pseudocode

03 Sudah bisa membuat algoritma dengan notasi tersebut (terutama Pseudocode)

04 Sudah implementasi algoritma ke minimal 1 Bahasa Pemrograman

05 Siap, semangat & tekun

youtube.com/studywithstudentkuy

Pengenalan Singkat ?

adalah mata kuliah untuk mempelajari teknik pembuatan algoritma, menganalisa algoritma, meng-efektifkan dan menguji algoritma.

Kenapa ?

01 Mengetahui algoritma yang bagus

02 Membuat program yang efektif dan efisien

03 Meminimalkan kebutuhan waktu dan ruang

04 Peduli terhadap spesifikasi perangkat user

05 Program jangka panjang

06 Membedakan anak informatika atau bukan

Materi ?

01 Kompleksitas Algoritma

02 Big O

03 Strategi Algoritma

04 Brute Force

05 Greedy

06 Backtracking

07 Branch and Bound

08 Divide and Conquer

09 Dynamic Programming

youtube.com/studywithstudentkuy

NEXT

#2 KOMPLEKSITAS ALGORITMA

THANK YOU

KEEP LEARNING & KEEP SPIRITS

studywithstudent

ANALISIS & STRATEGI ALGORITMA

Ibu Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

#2 KOMPLEKSITAS ALGORITMA

youtube.com/studywithstudentkuy

Kompleksitas Algoritma ?

Kompleksitas : kepelikan, kekusutan, kerumitan, keruwetan, kesulitan.

Algoritma : urutan langkah logis untuk menyelesaikan suatu masalah komputer.

Kompleksitas Algoritma : tingkat kerumitan yang terjadi dari algoritma yang dibuat.

youtube.com/studywithstudentkuy

Good Algorithm ?

- Sebuah algoritma tidak saja harus benar , tetapi juga harus mangkus (efisien).
- Algoritma yang bagus adalah algoritma yang mangkus.
- Kemangkusan algoritma diukur dari berapa jumlah waktu dan ruang (space) memori yang dibutuhkan untuk menjalankannya.
- Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang.
- Kemangkusan algoritma dapat digunakan untuk menilai algoritma yang terbaik.

youtube.com/studywithstudentkuy

Perhitungan Kebutuhan Waktu/Ruang ?

Menghitung Rata-rata

a_0	a_1	a_2	$a_{..}$	a_n

```
integer length = length(a), sum = 0, i = 0
while i < length do
  sum = sum + a_i
  i++
endwhile
integer avg = sum / length
```

Hitung Pengisian Nilai

length=length(a), sum=0, i=0,
sum=sum+a_i, i++, avg=sum/length

$t_1 = n + 1 + 1 + n + n + 1 = 3 + 3n$

Hitung Penjumlahan

sum=sum+a_i, i++

$t_2 = n + n = 2n$

Hitung Pembagian

avg=sum/length

$t_3 = 1$

Total Perhitungan

$t = t_1 + t_2 + t_3 = (3+3n)a + (2n)b + c$ detik

Kurang Dapat Diterima ?

- Dalam implementasi tidak mempunyai berapa waktu sesungguhnya untuk melaksanakan suatu operasi tertentu.
- Komputer dengan arsitektur yang berbeda akan berbeda pula lama waktu untuk setiap jenis operasinya.
- Selain bergantung pada komputer, kebutuhan waktu sebuah program juga ditentukan oleh compiler bahasa yang digunakan.
- Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan compiler apapun.

youtube.com/studywithstudentkuy

+ Informasi ?

- Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah kompleksitas algoritma.
- Ada dua macam kompleksitas algoritma, yaitu kompleksitas waktu dan kompleksitas ruang.

NEXT

#2A KOMPLEKSITAS WAKTU & RUANG

THANK YOU

KEEP LEARNING & KEEP SPIRITS



Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 2A

KOMPELSITAS WAKTU & RUANG

Kompleksitas Waktu & Ruang ?

Komplesitas Waktu

Kompleksitas Waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n.

Komplesitas Ruang

Kompleksitas Ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat didalam algoritma sebagai fungsi dari ukuran masukan n.

Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, kita dapat menentukan laju peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan n.

Kompleksitas Waktu ?

- Kasus Terburuk

$T_{max}(n)$: (kebutuhan waktu maksimum), Kompleksitas waktu untuk kasus terburuk (Worst Case)

- Kasus Terbaik

$T_{min}(n)$: (kebutuhan waktu minimum), Kompleksitas waktu untuk kasus terbaik (Best Case)

- Kasus Rata-rata

$T_{avg}(n)$: (kebutuhan waktu secara rata-rata), Kompleksitas waktu untuk kasus rata-rata (Average Case)

Contoh ?

Kasus Sequential Search

```
procedure sequentialSearch (var ar
as array, var key)
 status = false

 for  i from 0 to length(arr)-1
   if (ar[i] == key )
     print( key + " ada di index ke"
+ i);
     status=true;
     break;
   endif
 endfor

 if(status==false)
   print( key + " tidak ada");
 endif
endprocedure
```

- Kasus Terbaik

Kasus terbaik bila ar[0] == key

$T_{min}(n) = 1$

- Kasus Terburuk

Kasus terburuk bila ar[n] == key atau key tidak ditemukan

$T_{max}(n) = n$

- Kasus Rata-rata

$$T_{avg}(n) = \frac{(1 + 2 + 3 + \ldots + n)}{n}$$

$$= \frac{\frac{1}{2} n\ (1+n)}{n} = \frac{1}{2}\ (1 + n)$$

$$= \frac{1+n}{2}$$

Video Selanjutnya

#3 Komplesitas Waktu Asimptotik / Big O

Thank you

#KEEPLEARNING
#KEEPSPIRITS



Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 3

KOMPLEKSITAS WAKTU ASIMPTOTIK / BIG O

Pertumbuhan T(n) dengan n^2

- Untuk n yang besar, pertumbuhan T(n) sebanding dengan n^2. Pada kasus ini, T(n) tumbuh seperti n^2 tumbuh.

- T(n) tumbuh seperti n^2 tumbuh saat n bertambah. Kita katakan bahwa T(n) berorde n^2 dan kita tuliskan
  $T(n) = O(n^2)$

- Notasi "O" disebut notasi "O-Besar" (Big-O) yang merupakan notasi kompleksitas waktu asimptotik.

- T(n) = O(f(n)) (dibaca "T(n) adalah O(f(n)" yang artinya T(n) berorde paling besar f(n)) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C(f(n))$$

untuk $n \geq n_0$

- f(n) adalah batas atas (upper bound) dari T(n) untuk n yang besar.

Pertumbuhan T(n) dengan n^2

Tinjau $T(n) = 2n^2 + 6n + 1$

N	$T(n) = 2n^2 + 6n + 1$	n^2
10	261	100
100	20.601	10.000
1000	2.006.001	1.000.000
10.000	200.060.001	100.000.000

Notasi Asimptotik ?

- Notasi asimtotik merupakan himpunan fungsi yang dibatasi oleh suatu fungsi $n \in N$ yang cukup besar.

- Fungsi : $N \rightarrow R$ (sering R+)

- Notasi Asimtotik digunakan untuk menentukan kompleksitas suatu algoritma dengan melihat waktu tempuh algoritma. Waktu tempuh algoritma merupakan fungsi : $N \rightarrow R+$

Kompleksitas Waktu ?

- Kasus Terbaik

$T_{min}(n)$: (kebutuhan waktu minimum), Kompleksitas waktu untuk kasus terbaik (Best Case)
Dilambangkan $\theta(\ldots)$ dibaca Theta

- Kasus Rata-rata

$T_{avg}(n)$: (kebutuhan waktu secara rata-rata), Kompleksitas waktu untuk kasus rata-rata (Average Case)
Dilambangkan $\Omega(\ldots)$ dibaca Omega

- Kasus Terburuk

$T_{max}(n)$: (kebutuhan waktu maksimum), Kompleksitas waktu untuk kasus terburuk (Worst Case)
Dilambangkan $O(\ldots)$ dibaca Big O

+ Information ?

- Kinerja sebuah algoritma biasanya diukur dengan menggunakan patokan keadaan terburuk (worst case) yang dinyatakan dengan Big O.

- Big O dan Kompleksitas Waktu (Time Complexity) itu sama. Karena sama-sama analisis waktu dengan indikator waktu akan bertambah seiring dengan banyaknya masukkan data.

Pembacaan Big O ?

Kelompok Algoritma	Nama Algoritma
O(1)	Algoritma Konstan
O(log *n*)	Algoritma Logaritmik
O(*n*)	Algoritma Lanjar
O(*n* log *n*)	Algoritma *n* log *n*
O(n^2)	Algoritma Kuadratik
O(n^3)	Algoritma Kubik
O(2^n)	Algoritma Eksponensial
O(*n*!)	Algoritma Faktorial

* Urutan menunjukkan big O dari kecil ke semakin besar.

Teorema ?

Merupakan suatu pernyataan matematika yang masih memerlukan pembuktian serta pernyataanya dapat ditunjukkan nilai kebenarannya atau juga bernilai benar.

Aturan Teorema ?

- $T(n) = a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n + a_0$ then
  $T(n) = O(n^m)$

- if $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$ then

  - $T_1(n) + T_2(n) = O(f(n)) + O(g(n))$
    $= O(\max(f(n), g(n))$

  - $T_1(n) . T_2(n) = O(f(n)) . O(g(n))$
    $= O(f(n) . g(n))$

- $O(cf(n)) = O(f(n))$, c adalah konstanta

- $f(n) = O(f(n))$

- $C = O(1)$

- ...???

- if $T_1(n) = O(n)$ and $T_2(n) = O(n^2)$ then

  - $T_1(n) + T_2(n) = O(n) + O(n^2)$
    $= O(\max(n, n^2)) = O(n^2)$

  - $T_1(n) . T_2(n) = O(n) . O(n^2)$
    $= O(n . n^2) = O(n^3)$

- $O(5n^2) = O(n^2)$

- $n^2 = O(n^2)$

- $3 = O(1)$

Contoh Soal ?

Now or Next Video .. ??

Contoh Soal ?

1. Buktikan : $T(n) = (n^2 - 1) + 3 = O(n^2)$

2. Buktikan : $T(n) = n^3 + 2 = O(n^3)$

3. Tentukan : $5n^2 = \ldots???$

4. Tentukan : $12 = \ldots???$

5. Jumlahkan : $T_1(n) = n/2$ dengan $T_2(n) = n^2/4 \ldots???$

6. Kalikan : $T_1(n) = n^2$ dengan $T_2(n) = 3 \ldots???$

7. Perhitungan Big 0 Pseudocode Konstan

8. Perhitungan Big O Pseudocode Lanjar

Jawaban Soal ?

1. Buktikan : $T(n) = (n^2 - 1) + 3 = O(n^2)$

Ketentuan

$T(n) \leq C(f(n))$

Penyelesaian

$(n^2 - 1) + 3 \leq (n^2 - 1n^2) + 3n^2 = O(n^2)$

$(n^2 - 1) + 3 \leq (n^2 - n^2) + 3n^2 = 3(n^2)$

Konstanta dan n_0

$C = 3$ & $n_0 = \ldots???$

Percobaan kalo $n_0 = 1$

$(1^2 - 1) + 3 \leq (1^2 - 1^2) + 3.1^2 = 3(n^2)$

$(0) \quad + 3 \leq (0) \quad + 3$

Percobaan kalo $n_0 = 2$

$(2^2 - 1) + 3 \leq (2^2 - 2^2) + 3.2^2 = 3(n^2)$

$(3) \quad + 3 \leq (0) \quad + 6$

Jadi $(n^2 - 1) + 3 = O(n^2)$ karena $(n^2 - 1) + 3 \leq (n^2 - n^2) + 3n^2 = 3(n^2)$ untuk semua $n \geq 1$ ($C = 3$ dan $n0 = 1$)

Jawaban Soal ?

2. Buktikan : $T(n) = n^3 + 2 = O(n^3)$

Ketentuan

$T(n) \leq C(f(n))$

Penyelesaian

$n^3 + 2 \leq n^3 + 2n^3 = O(n^3)$

$n^3 + 2 \leq n^3 + 2n^3 = 3(n^3)$

Konstanta dan n_0

$C = 3$ & $n_0 = \ldots???$

Percobaan kalo $n_0 = 1$

$1^3 + 2 \leq 1^3 + 2.1^3 = 3(n^3)$

$3 \leq 3$

Percobaan kalo $n_0 = 2$

$2^3 + 2 \leq 2^3 + 2.2^3 = 3(n^3)$

$10 \leq 18$

Jadi $n^3 + 2 = O(n^3)$ karena $n^3 + 2 \leq n^3 + 2n^3 = 3(n^3)$ untuk semua $n \geq 1$ ($C = 3$ dan $n0 = 1$)

Jawaban Soal ?

***catatan :**

- ada 2 versi mengerjakan jenis soal nomor 1 dan 2. Tapi menurut saya cara mengerjakan soal seperti ini yang masuk akal. Karena tidak merubah konstanta (konstanta bersifat tetap)

- Jika n0 = 1 tidak benar coba untuk menguji ke 2,3,dst. Namun jika tetap salah, maka tidak terbukti

Jawaban Soal ?

3. Tentukan : $5n^2$ = …???

$5n^2 = O(5n^2) = O(n^2)$ <= Algoritma Kuadratik

4. Tentukan : 12 = …???

12 = O(12) = O(1) <= Algoritma Konstan

Jawaban Soal ?

5. Jumlahkan : $T_1(n) = n/2$ dengan $T_2(n) = n^2/4$ …???

Jawaban

$T_1(n) = n/2 = O(n)$
$T_2(n) = n/4 = O(n)$

$T_1 + T_2 = \max(O(n,n)) = O(n)$ <= Algoritma Lanjar

6. Kalikan : $T_1(n) = n^2$ dengan $T_2(n) = 3$ …???

Jawaban

$T_1(n) = n^2 = O(n^2)$
$T_2(n) = 3 = O(1)$

$T_1 \times T_2 = O(n^2,1) = O(n^2)$ <= Algoritma Kuadratik

Jawaban Soal ?

7. Perhitungan Big O Pseudocode Konstan

```
INPUT number1
INPUT number2
SET total = number1 + number2

if total > 100 then

  DISPLAY("too large a total");

endif
```

***Catatan :** Untuk semua operasi penugasan, aritmatik, baca, tulis, pengkondisian, akses array (baca, tulis, hapus, edit) masing-masing bernilai O(1).
Untuk perulangan bernilai O(n)

```
INPUT number1 <= O(1)
INPUT number2 <= O(1)
SET total = number1 + number2 <= O(1)

if total > 100 then <= O(1)

  DISPLAY("too large a total"); <= O(1)

endif
```

Jawaban

T(n) = O(1) + O(1) + O(1) + (O(1) . O(1))

= max(O(1), O(1), O(1), (O(1) . O(1)))

= max(O(1), O(1), O(1), O(1))

= O(1)

Jawaban Soal ?

8. Perhitungan Big O Pseudocode Lanjar

```
INPUT number

for i = number to 1 do

  DISPLAY( number * i );

endfor
```

***Catatan :** Untuk semua operasi penugasan, aritmatik, baca, tulis, pengkondisian, akses array (baca, tulis, hapus, edit) masing-masing bernilai O(1).
Untuk perulangan bernilai O(n)

```
INPUT number <= O(1)

for i = number to 1 do <= O(n)

  DISPLAY( number * i ); <= O(1)

endfor
```

Jawaban

T(n) = O(1) + (O(n) . O(1))

= max(O(1), (O(n) . O(1)))

= max(O(1), O(n))

= O(n)

Video Selanjutnya

#4 Strategi Algoritma

Thank you

#KEEPLEARNING
#KEEPSPIRITS



Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 4

STRATEGI ALGORITMA

Strategi Algoritma ?

- **Strategi :** adalah rencana yang cermat mengenai kegiatan untuk mencapai sasaran khusus (KBBI).

- **Algoritma :** adalah urutan langkah-langkah untuk memecahkan suatu masalah.

- **Strategi Algoritma :** adalah kumpulan metode atau teknik untuk memecahkan masalah guna mencapai tujuan yang ditentukan, yang dalam hal ini deskripsi metode atau teknik tersebut dinyatakan dalam suatu urutan langkah-langkah penyelesaian.

Kelompok Strategi Algoritma ?

- **Strategi solusi langsung** (direct solution strategies)
  - **Algoritma Brute Force**
  - **Algoritma Greedy**

- **Strategi berbasis pencarian pada ruang status** (state-space base strategies)
  - **Algoritma Backtracking**
  - **Algoritma Branch and Bound**

- **Strategi solusi atas-bawah** (top-down solution strategies)
  - **Algoritma Divide and Conquer**

- **Strategi solusi bawah-atas** (bottom-up solution strategies)
  - **Dynamic Programming**



Video Selanjutnya

#5a Direct Solution Strategies
BRUTE FORCE

Thank you

#KEEPLEARNING
#KEEPSPIRITS



Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 5A

Direct Solution Strategies

BRUTE FORCE

Brute Force ?

- adalah sebuah pendekatan yang lempang (straightforward) untuk memecahkan suatu masalah

- biasanya didasarkan pada pernyataan masalah (problem statement) dan definisi konsep yang dilibatkan.

- algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (obvious way).

Karakter Brute Force ?

- Algoritma brute force umumnya tidak “cerdas” dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang-kadang algoritma brute force disebut juga algoritma naif (naïve algorithm).

- Algoritma brute force seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan , atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.

- Untuk masalah yang ukurannya kecil, kesederhanaan brute force biasanya lebih diperhitungkan daripada ketidakmangkusannya. Algoritma brute force sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus.

- Meskipun brute force bukan merupakan teknik pemecahan masalah yang mangkus, namun teknik brute force dapat diterapkan pada sebagian besar masalah. Agak sukar menunjukkan masalah yang tidak dapat dipecahkan dengan teknik brute force. Bahkan ada masalah yang hanya dapat dipecahkan secara brute force.

- Selain itu, algoritma brute force seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya, kadang-kadang algoritma brute force dapat lebih mangkus (ditinjau dari segi implementasi).

Contoh Penggunaan Brute Force ?

- Perhitungan Faktorial

- Pencarian Bilangan Terbesar atau Terkecil

- Sequential Search

- Bubble Sort

- dll..

Pseudocode Sequential Search ?

```
procedure sequentialSearch (var ar as array, var key)
 status = false

 for  i from 0 to length(arr)-1
   if (ar[i] == key )
     print( key + " ada di index ke" + i);
     status=true;
     break;
   endif
 endfor

 if(status==false)
   print( key + " tidak ada");
 endif
endprocedure
```

Video Selanjutnya

TUNGGU AJA ☺

Thank you

#KEEPLEARNING
#KEEPSPIRITS



Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 6A

State-space Base Strategies

BACKTRACKING Algorithm

Backtracking ?

- Runut-balik (backtracking) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus.

- Runut-balik, yang merupakan perbaikan dari algoritma brute-force, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada.

- Dengan metode runut-balik, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat.

- Runut-balik merupakan bentuk tipikal dari algoritma rekursif.

- Saat ini algoritma runut-balik banyak diterapkan untuk program games (seperti permainan tic-tac-toe, menemukan jalan keluar dalam sebuah labirin, catur, dll) dan masalah-masalah pada bidang kecerdasan buatan (artificial intelligence).

*Note

- Walaupun Backtracking perbaikan dari Brute Force, Backtracking tidak bisa menyelesaikan semua masalah yang dapat diselesaikan dengan Brute Force.

Video Selanjutnya

Top-Down Solution Strategies
DIVIDE & CONQUER

Thank you

#KEEPLEARNING
#KEEPSPIRITS

Analisis & Strategi Algoritma

Saniati,S.ST.,M.T. | Ir. Rinaldi Munir,M.T.

EPISODE 7A

Top-Down Solution Strategies

DIVIDE & CONQUER

Divide & Conquer ?

- Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama divide ut imperes.

- Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama Divide and Conquer.

Definisi Divide & Conquer ?

- **Devide**
  membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),

- **Conquer**
  memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan

- **Combine**
  mengabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

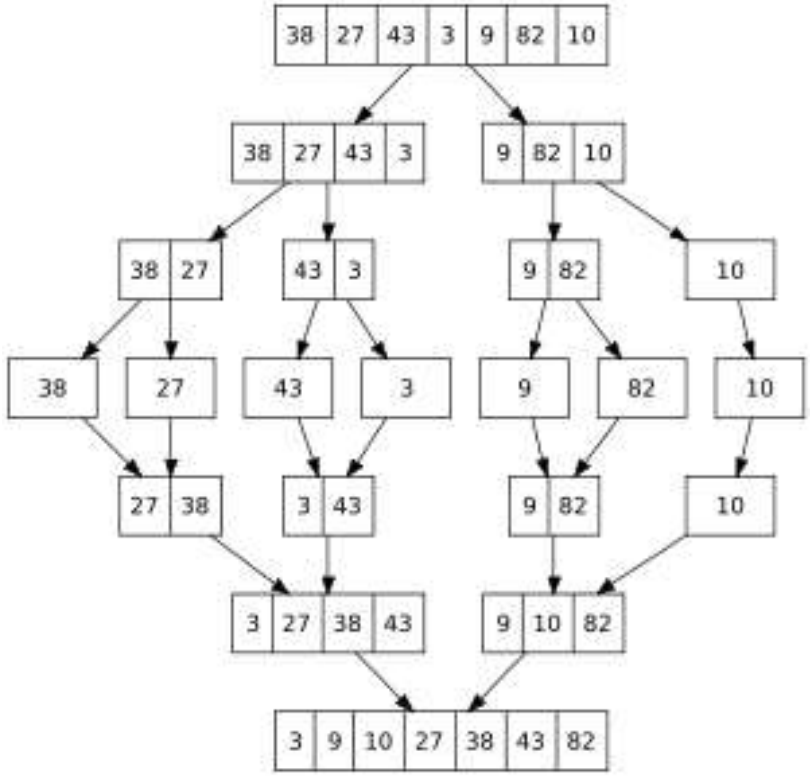
Hal-hal Divide & Conquer ?

- Obyek permasalahan yang dibagi adalah masukan (input) atau instances yang berukuran n: tabel (larik), matriks, eksponen, dan sebagainya, bergantung pada masalahnya.

- Tiap-tiap upaya masalah mempunyai karakteristik yang sama (the same type) dengan karakteristik masalah asal, sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif.

Contoh Divide & Conquer ?

- Merge Sort



Video Selanjutnya

ENDING...

Thank you

#KEEPLEARNING
#KEEPSPIRITS