Brute Force
&
Divide Conquer (DC)

Teaching Team

Algorithm and Data Structure

2023/2024

Learning Outcome

- Students have to understand the basic concept of *brute force* and *divide conquer (DC)*
- Students must be able to create a flowchart based on the *brute force* and *divide conquer* algorithm

Outlines

- *Brute Force*
- *Divide Conquer*
- Big O Notation

Pengantar

- Deciding the data structure algorithm to solve the problem, is related to the number of data or *instance*
- The correct data structure decided, the lower time of complexity and the lower cost will be
- 2 main approach of problem solving:
  - *Brute Force*
  - *Divide Conquer*

Brute Force

Just do it! (one by one?)

Definisi Brute Force #1

- ***Brute force*** adalah pendekatan yang lempang (*straightforward*)
- Dasar pemecahan dengan algoritma brute force didapatkan dari pernyataan pada persoalan (*problem statement*) dan definisi konsep yang dilibatkan.
- Algoritma ***brute force*** lebih cocok untuk persoalan yang berukuran kecil karena mudah diimplementasikan dan tata cara yang sederhana.

Definisi Brute Force #2

- Biasanya didasarkan pada:
  - pernyataan pada persoalan (*problem statement*)
  - definisi konsep yang dilibatkan.
- Algoritma *brute force* memecahkan persoalan dengan
  - sangat sederhana,
  - langsung,
  - jelas (*obvious way*).
- *Just do it!* atau *Just Solve it!*

Kareteristik Algoritma Brute Force

- Kata “force” mengindikasikan “tenaga” ketimbang “otak”
- Kadang-kadang algoritma *brute force* disebut juga **algoritma naif** (*naïve algorithm*).

Algoritma *brute force* lebih cocok untuk persoalan yang berukuran kecil.

- Sederhana,
- Implementasinya mudah

Algoritma *brute force* sering digunakan sebagai basis pembanding dengan algoritma yang lebih mangkus/baik.

Meskipun bukan metode yang mangkus, hampir semua persoalan dapat diselesaikan dengan algoritma *brute force.*

- Sukar menunjukkan persoalan yang tidak dapat diselesaikan dengan metode *brute force*.
- Bahkan, ada persoalan yang hanya dapat diselesaikan dengan metode *brute force*.

Contoh #1 - Mencari elemen terbesar (terkecil)

Persoalan :

Diberikan sebuah senarai yang beranggotakan *n* buah bilangan bulat (a_1, a_2, …, a_n). Carilah elemen terbesar di dalam senarai tersebut.

Algoritma *brute force*:

Bandingkan setiap elemen senarai untuk menemukan elemen terbesar

```
procedure CariElemenTerbesar(input a1, a2, ..., an : integer,
                             output maks : integer)
{ Mencari elemen terbesar di antara elemen a1, a2, ..., an. Elemen
terbesar akan disimpan di dalam maks.
Masukan: a1, a2, ..., an
Keluaran: maks
}
Deklarasi
  k : integer

Algoritma:
  maks←a1
  for k←2 to n do
    if ak > maks then
      maks←ak
    endif
  endfor
```

Contoh #2 - Pencocokan String *(String Matching)*

Persoalan:

Diberikan

a)teks (*text*), yaitu (*long*) *string* dengan panjang *n* karakter

b)pattern, yaitu *string* dengan panjang *m* karakter (asumsi: $m < n$)

Carilah lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.

Penyelesaian dengan Algoritma *brute force*:

1) Mula-mula *pattern* dicocokkan pada awal teks.
2) Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *pattern* dengan karakter yang bersesuaian di dalam teks sampai:
   - semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau
   - dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3) Bila *pattern* belum ditemukan kecocokannya dan teks belum habis, geser *pattern* satu karakter ke kanan dan ulangi langkah 2.

Ilustrasi Contoh #2

Pattern: NOT
Teks: NOBODY NOTICED HIM

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Pattern: 001011
Teks: 10010101**001011**110101010001

```
  10010101001011110101010001
1 001011
2  001011
3   001011
4    001011
5     001011
6      001011
7       001011
8        001011
9         001011
```

Case Brute Force

Worst Case

- Pada setiap pergeseran pattern, semua karakter di *pattern* dibandingkan.
- Contoh:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaaab"
  - P: "aaab“

Best Case

- Terjadi bila karakter pertama *pattern P* sama dengan karakter teks *T*
- Atau terjadi bila karakter pertama *pattern P* tidak pernah sama dengan karakter teks *T* yang dicocokkan
- Jumlah perbandingan maksimal *n* kali:
- Contoh:

  T: String ini berakhir dengan zzz

  P: Str

Kelebihan dan Kelemahan Brute Force

Kelebihan

1. Metode *brute force* dapat digunakan untuk memecahkan hampir sebagian besar masalah (*wide applicability*).
2. Metode *brute force* sederhana dan mudah dimengerti.
3. Metode *brute force* menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan *string*, perkalian matriks.
4. Metode *brute force* menghasilkan algoritma baku (standard) untuk tugas-tugas komputasi seperti penjumlahan/perkalian *n* buah bilangan, menentukan elemen minimum atau maksimum di dalam tabel (*list*).

Kelemahan

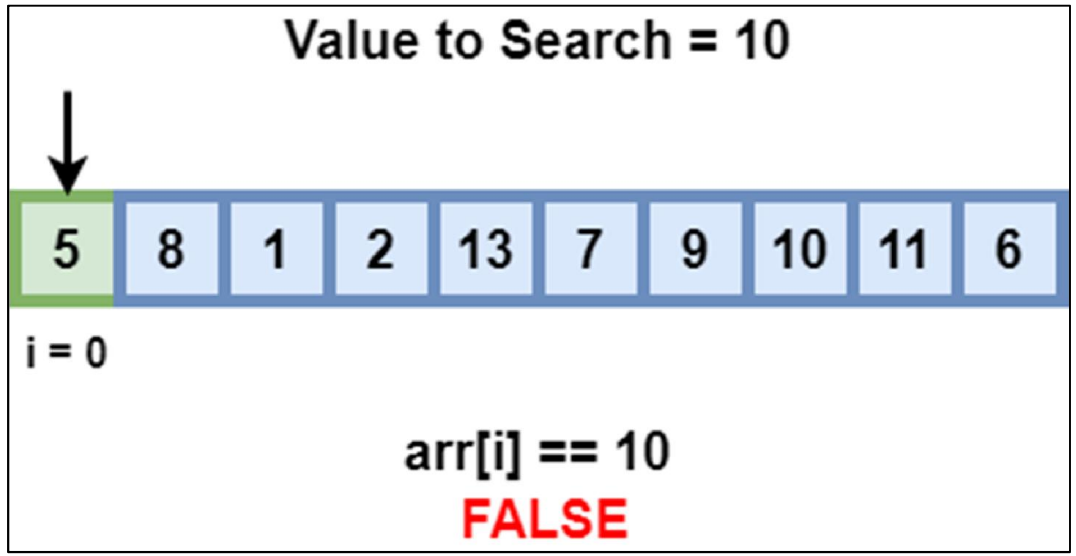
1. Metode *brute force* jarang menghasilkan algoritma yang mangkus.
2. Beberapa algoritma *brute force* lambat sehingga tidak dapat diterima.
3. Tidak sekontruktif/sekreatif teknik pemecahan masalah lainnya.

Penerapan Algoritma Brute Force

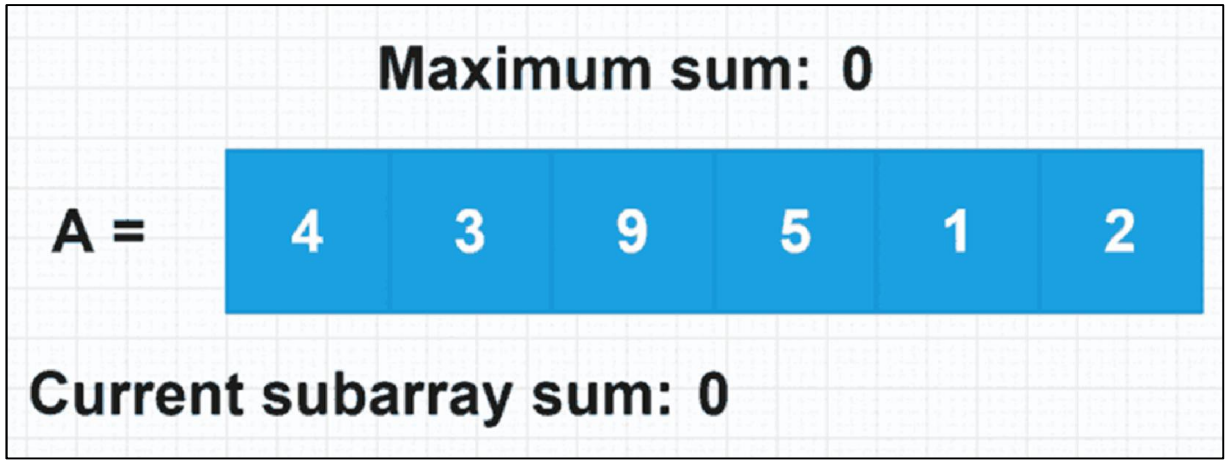
- *Sequential Search / Linear Search*
- *Bubble Sort*
- *Selection Sort*

Ilustrasi Penerapan Pendekatan Brute Force

Searching

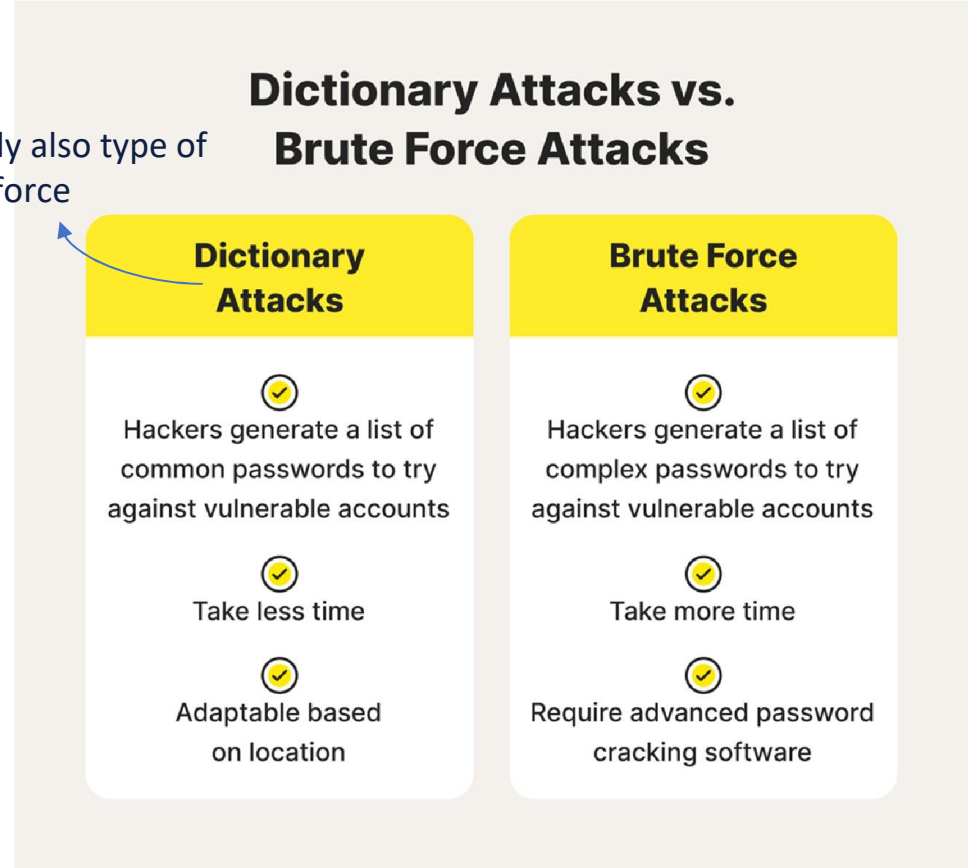


Highest Sum

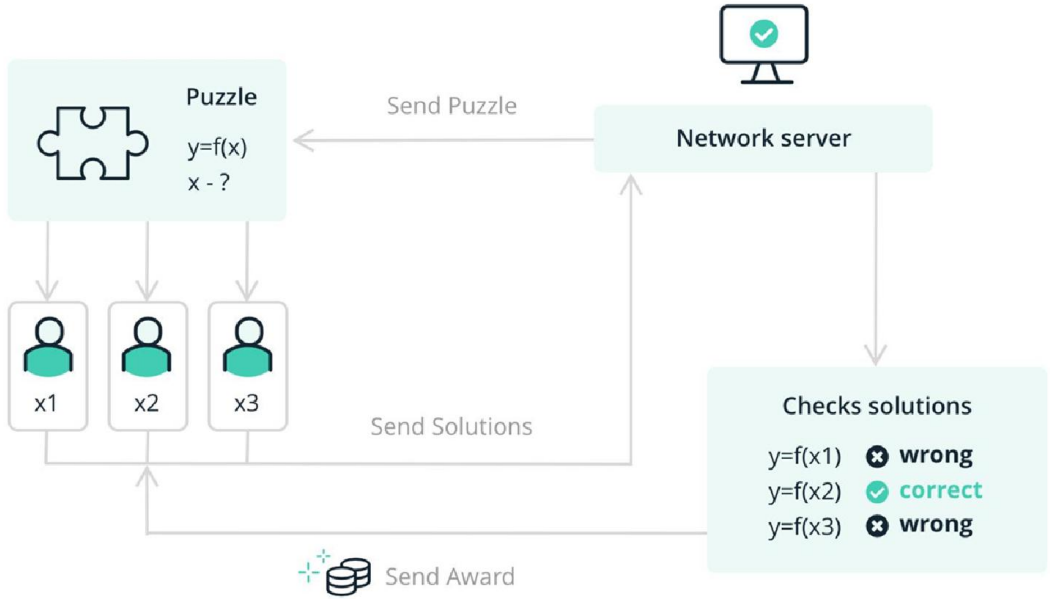


Trivia: *Brute Force* Dunia Nyata

Actually also type of brute force



Cybersecurity



Konsep Proof-of-Work

Blockchain

Divide Conquer

Pengenalan *Divide and 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*.

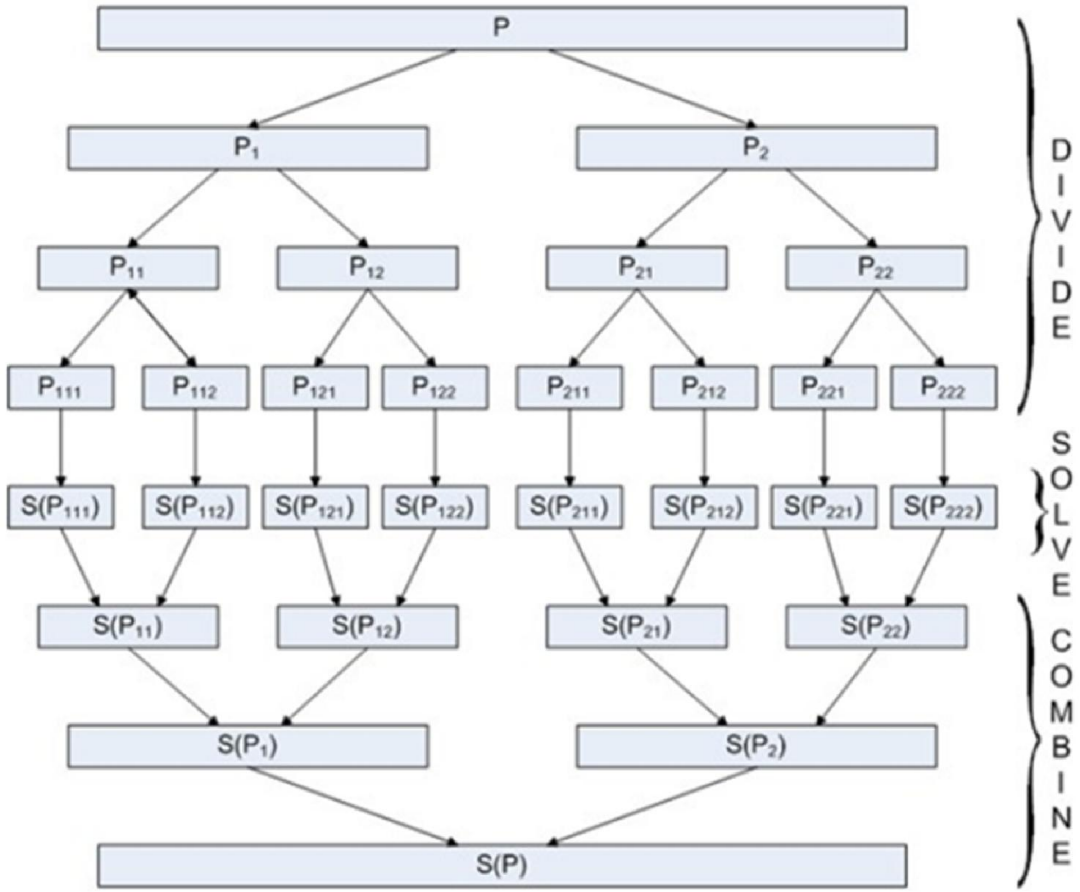
Definisi Divide Conquer #1

- ***Divide***: membagi masalah menjadi beberapa bagian masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),
- ***Conquer***: memecahkan (menyelesaikan) masing-masing bagian masalah (secara rekursif), dan
- ***Combine****:* mengabungkan solusi masing-masing bagian masalah sehingga membentuk solusi masalah semula.

Definisi Divide Conquer #2

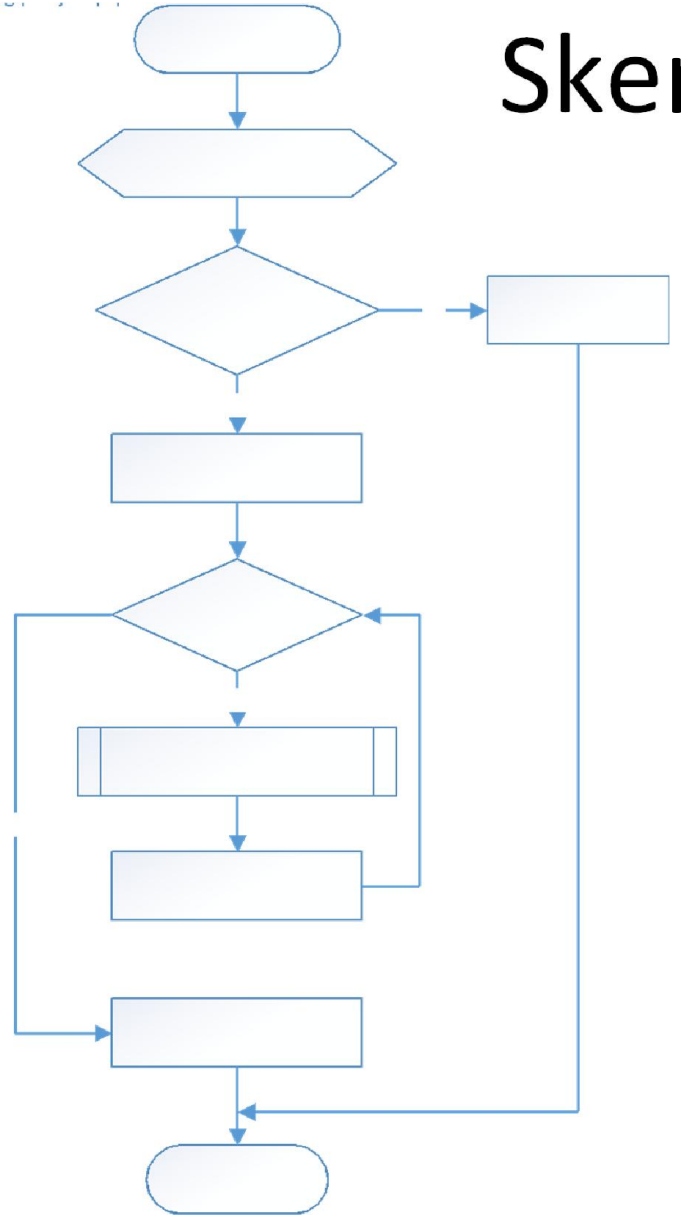
- Obyek persoalan yang dibagi : masukan (input) atau instances persoalan yang berukuran n seperti:
  - Tabel (larik),
  - Matriks,
  - Eksponen,
  - Dll, bergantung persoalannya.
- Tiap-tiap bagian masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal
- Sehingga metode *Divide and Conquer* lebih natural diungkapkan dengan skema rekursif.

Ilustrasi *Divide Conquer*



Keterangan:
P = persoalan
S = solusi

Skema Umum Divide Conquer



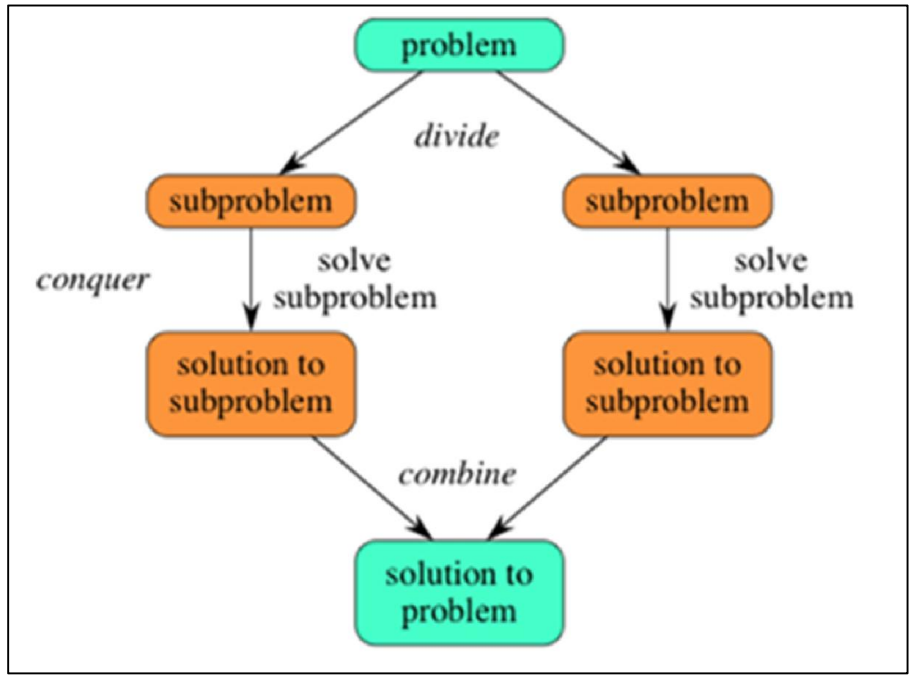
```
procedure DIVIDE_and_CONQUER(input n : integer)
{ Menyelesaikan masalah dengan algoritma D-and-C.
  Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}
Deklarasi
     r, k : integer
Algoritma
  if n ≤ n0 then  {ukuran masalah sudah cukup kecil }
     SOLVE upa-masalah yang berukuran n ini
  else
     Bagi menjadi r upa-masalah, masing-masing berukuran n/k
     for masing-masing dari r upa-masalah do
        DIVIDE_and_CONQUER(n/k)
     endfor
     COMBINE solusi dari r upa-masalah menjadi solusi masalah semula }
  endif
```

Skema Jika Pembagian Menghasilkan Dua Bagian Masalah Berukuran Sama

```
procedure DIVIDE_and_CONQUER(input n : integer)
{ Menyelesaikan masalah dengan algoritma D-and-C.
  Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}
Deklarasi
     r, k : integer

Algoritma
  if n ≤ n0 then {ukuran masalah sudah cukup kecil }
     SOLVE upa-masalah yang berukuran n ini
  else
     Bagi menjadi 2 upa-masalah, masing-masing berukuran n/2
     DIVIDE_and_CONQUER(upa-masalah pertama yang berukuran n/2)
     DIVIDE_and_CONQUER(upa-masalah kedua yang berukuran n/2)
     COMBINE solusi dari 2 upa-masalah
  endif
```

Ilustrasi Divide Conquer

1 Tahap Rekursif

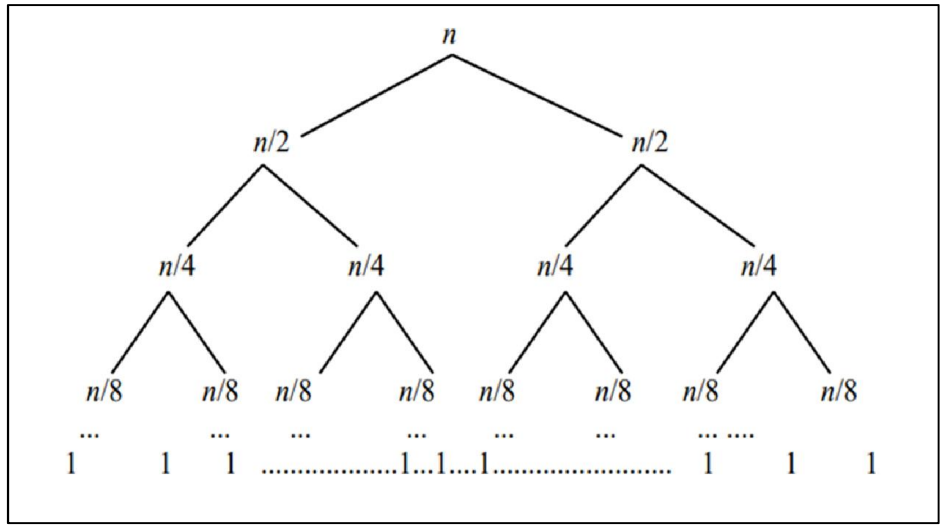


2 Tahap Rekursif

Case Divide Conquer

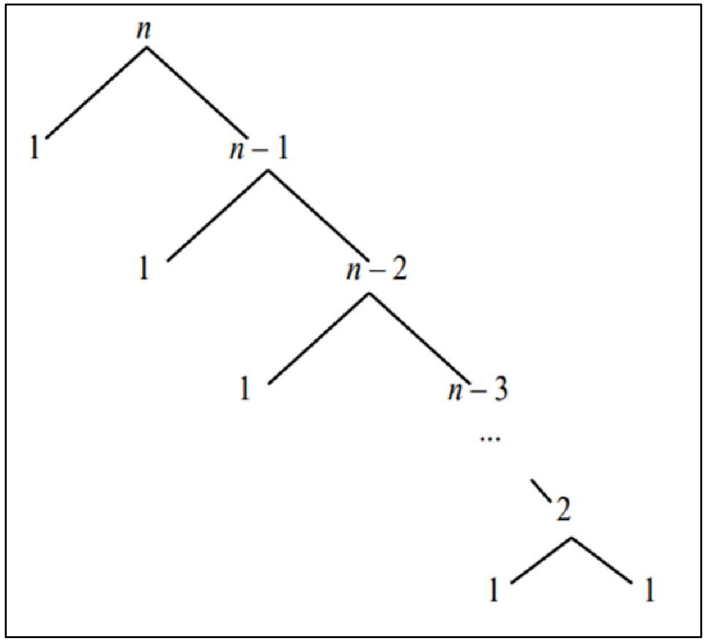
- Best Case

  Best case ditemukan saat elemen median bagian-tabel berukuran relatif sama setiap partisi

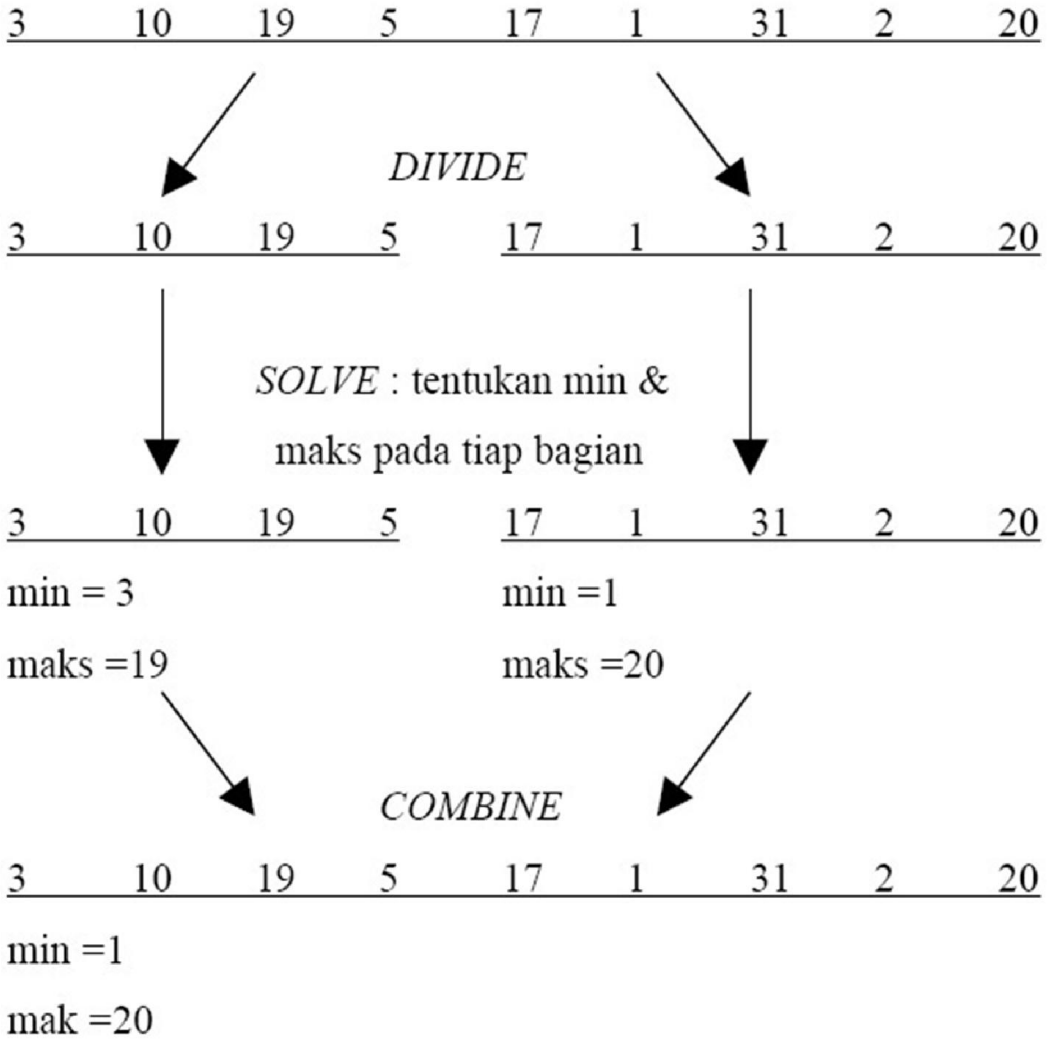


- Worst Case

  Worst Case ditemukan saat upa tabel selalu minimum atau maksimum (tidak berukuran sama) setiap partisi.



Contoh – Mencari Nilai Min dan Max



Kelebihan dan Kelemahan Divide Conquer

Kelebihan

- Dapat memecahkan masalah yang sulit (Efektif untuk masalah yang cukup rumit)
- Memiliki efisiensi algoritma yang tinggi. (Efisien menyelesaikan algoritma sorting)
- Bekerja secara paralel. Divide and Conquer didesain bekerja dalam mesin-mesin yang memiliki banyak prosesor (memiliki sistem pembagian memori)
- Akses memori yang cukup kecil, sehingga meningkatkan efisiensi memori

Kelemahan

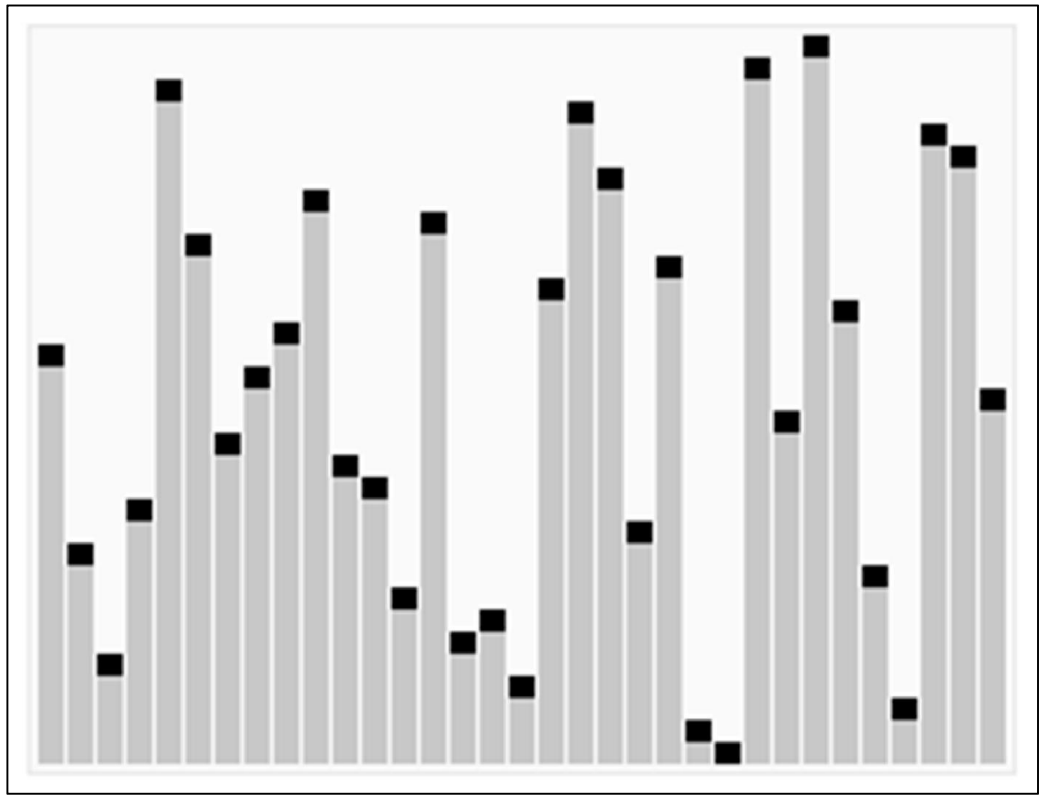
- Lambatnya proses perulangan (Beban yang cukup signifikan pada prosesor, jadi lebih lambat prosesnya untuk masalah yang sederhana)
- Lebih rumit untuk masalah yang sederhana (Algoritma sekuensial terbukti lebih mudah dibuat daripada algoritma divide and conquer untuk masalah sederhana)

Penerapan Algoritma Divide Conquer

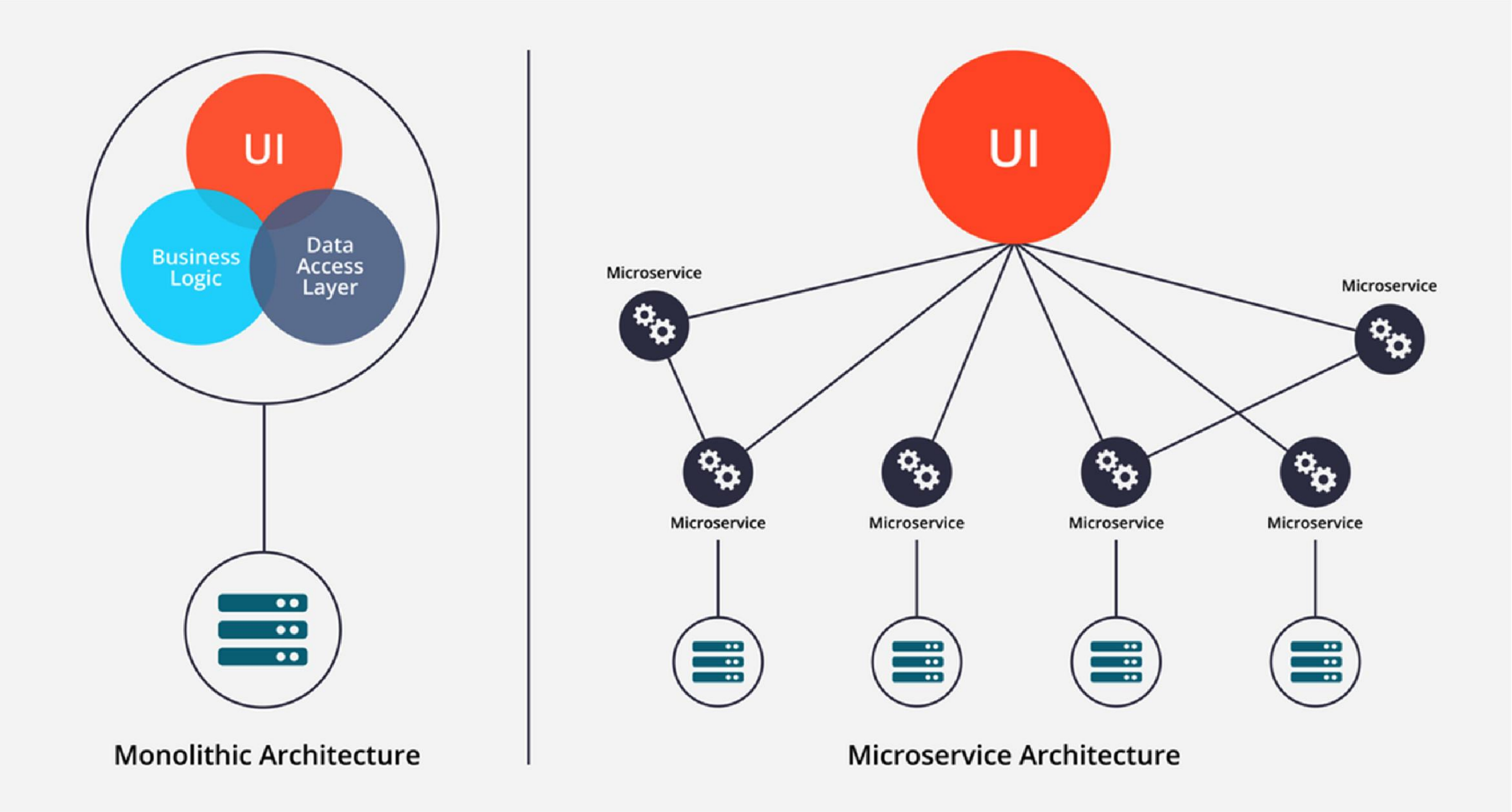
- *Merge Sort*
- *Quick Sort*
- *Binary Search*

Ilustrasi Penerapan *Divide Conquer* Pada Kasus *Sorting*

MERGE SORT



Trivia: *Divide Conquer*



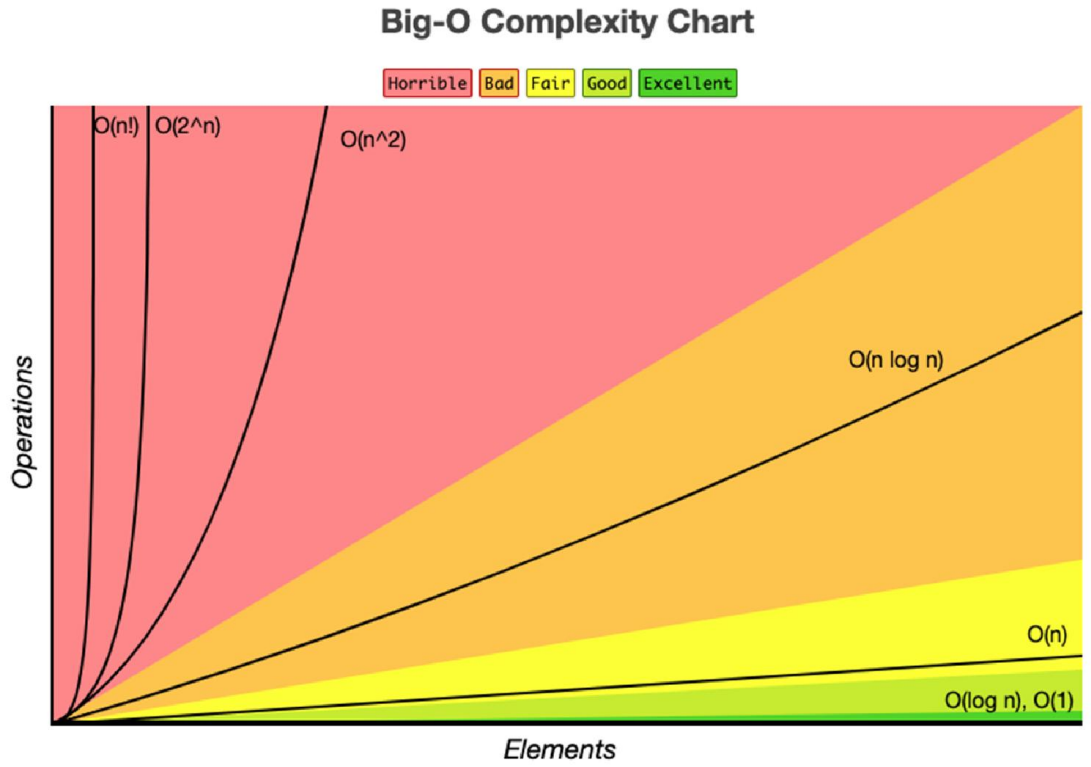
Arsitektur *Microservices*

Notasi Big O

Apakah algoritma kita sudah efisien?

Notasi Big O

- Analisis algoritma untuk kompleksitas waktu atau ruang memori
- Dapat diukur atau dipandang berdasarkan worst-case, best-case, average-case.
- Dari tercepat hingga terlambat :
  1. O(1)
  2. O(log n)
  3. O(n) .
  4. O(n log n)
  5. O(n^p)
  6. O(k^n)
  7. O(n!)



Notasi O(1)

Contoh Kode

```
int n = 1000;
System.out.println("Hey - your input is: " + n);
```

```
int add(int a, int b) {
    return a + b;
}
```

Time complexity **O(1)** dikarenakan hanya menjalankan sekali instruksi return, berapapun input yang dimasukkan ke dalam fungsi

Notasi O(log n)

```
for (int i = 1; i < n; i = i * 2)
{
        System.out.println("Hasil: " + i);
}
```

Jika n = 8, maka

Output
Hasil : 1
Hasil : 2
Hasil : 4

Algoritma ini berjalan log(8) = 3 kali

Notasi O(n)

```
double average(double[] numbers) {
    double sum = 0;

    for(double number: numbers) {
       sum += number;
    }

    return sum / numbers.length;
}
```

- Fungsi diatas memiliki time complexity **O(n)** dikarenakan ia akan menjalankan looping untuk menjumlahkan bilangan-bilangan yang ada didalam array. Jumlah loopingnya bergantung pada panjang array yang dimasukkan kedalam fungsi.
- Jika numbers memiliki panjang array 3 dengan isi [2,3,4] , maka fungsi akan menjumlahkan secara urut 2, 3, dan 4, kemudian mengembalikan rata-ratanya. Sehingga, array yang memiliki panjang 3, fungsi akan melakukan looping sebanyak 3 untuk menjumlahkan bilangan-bilangannya, dan seterusnya.

Notasi O(n log n)

```
for (int i = 1; i <= n; i++){
    for(int j = 1; j < 8; j = j * 2) {
        System.out.println(”Hasil: " + i + " dan " + j);
    }
}
```

Jika n = 8, maka algoritma akan berjalan
*8 * log(8) = 8 * 3 = 24* kali.

Notasi O(n^p)

```
for (int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        System.out.println(”Hasil: " + i + " and " + j);
    }
}
```

Jika n = 8, maka 8^2 = 64 kali

Notasi O(n^p) – Contoh Lain

```
int func(int n) {
    int count = 0;
    for (int i = 1 ; i <= n ; i++) {
        for (int j = 1 ; j <= i ; j++) {
            count++;
        }
    }
    return count;
}
```

Berapa kali count++ dijalankan dengan nilai n sembarang?

- Ketika i = 1 , maka akan dijalankan 1 kali.
- Ketika i = 2 , maka akan dijalankan 2 kali.
- Ketika i = 3 , maka akan dijalankan 3 kali.
- dan seterusnya…

$$1 + 2 + 3 + \cdots + n = \frac{n^2 + n}{2}$$

time complexity $\mathbf{O}(\mathbf{n^2})$

Notasi $O(k^n)$

```
for (int i = 1; i <= Math.pow(2, n); i++){
    System.out.println(”Hasil : " + i);
}
```

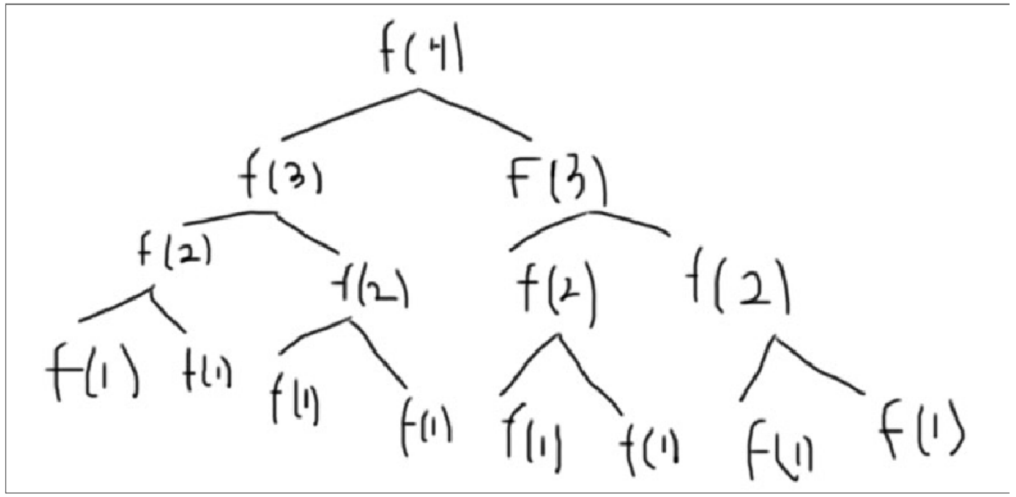
Muncul pada kasus atau factor yang terksponensial dengan ukuran input

Jika n = 8, maka 2^8 = 256

Notasi $O(k^n)$ – Contoh Lain

```
int func(int n) {
    if (n <= 1) {
        return 1;
    }

    return func(n-1) + func(n-1);
}
```

Misalkan dipanggil dengan func(4)



time complexity sebesar **O(2^n)**

Notasi O(n!)

```
for (int i = 1; i <= factorial(n); i++){
    System.out.println(”Hasil: " + i);
}
```

Jika n = 8 maka 8! = 40320

Aturan Big O

- Cari notasi yang paling berkontribusi (biasanya yang bagian “+” atau notasi O(1) ditiadakan)
- “If” umumnya bersifat “ + ”
- “for” umumnya bersifat “ * ”
- Konstanta dari notasi Big O dapat ditiadakan (cth: 2 O(n) menjadi O(n))

Contoh Kasus

```java
public class ContohBigO{
    public static void contohBig0(int[] angka){
        System.out.println("Pairs: ");
        int n = angka.length;

        for(int i =0; i < n; i++){
            for (int j =0; j < n; j++){
                System.out.println(angka[i] + "-" + angka[j]);
            }
        }

        for(int i =0; i < n; i++){
            for (int j =0; j < n; j++){
                System.out.println(angka[i] + "-" + angka[j]);
            }
        }
    }
}
```

Contoh Kasus

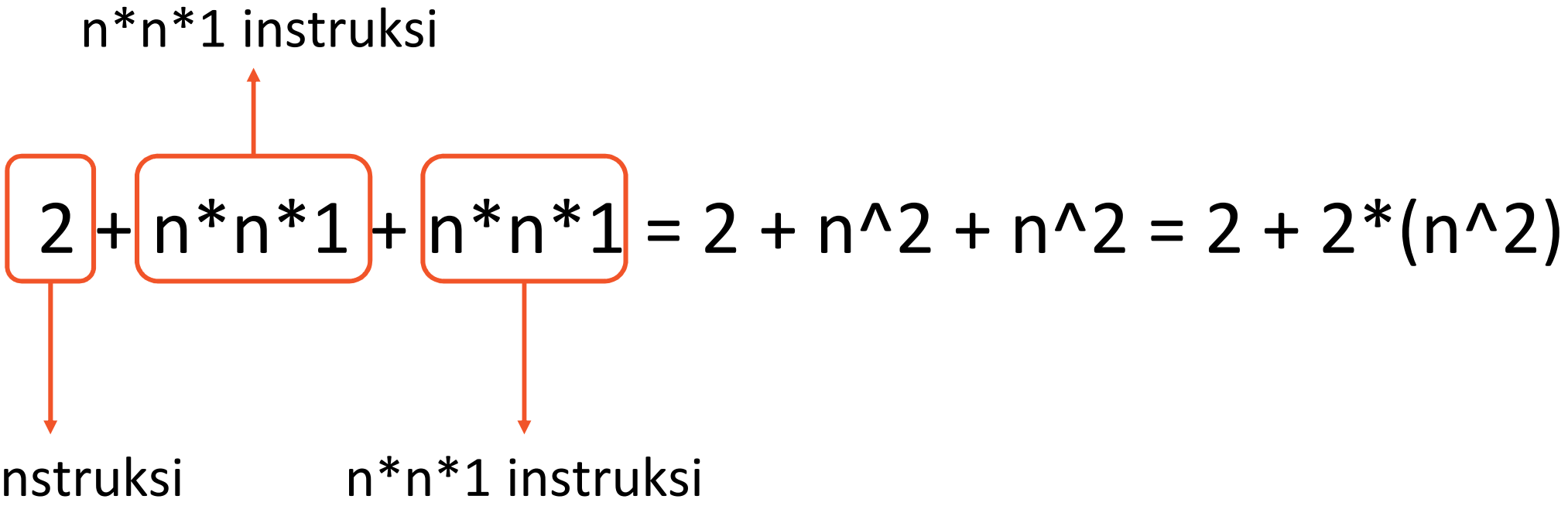
```
public class ContohBigO{
    public static void contohBig0(int[] angka){
        System.out.println("Pairs: ");
        int n = angka.length;

        for(int i =0; i < n; i++){
            for (int j =0; j < n; j++){
                System.out.println(angka[i] + "-" + angka[j]);
            }
        }

        for(int i =0; i < n; i++){
            for (int j =0; j < n; j++){
                System.out.println(angka[i] + "-" + angka[j]);
            }
        }
    }
}
```

2 instruksi

n*n*1 instruksi

n*n*1 instruksi

Contoh Kasus - Kesimpulan



Contoh:
Jika n = 10 elemen maka instruksi yang dijalankan adalah 2 + 2*(10^2) = 202

Latihan

1. Buatlah flowchart untuk menghitung nilai akar dari suatu bilangan dengan algoritma Brute Force dan Divide Conquer! *Jika bilangan tersebut bukan merupakan kuadrat sempurna, bulatkan angka ke bawah.*
2. Buatlah flowchart untuk menghitung hasil pangkat dari inputan suatu bilangan dengan algoritma Brute Force dan Divide Conquer!

Latihan

3. Tentukan notasi Big O yang sesuai dari kode program berikut!

a.

```
public int countVowels(char[] word){
    char[] vowels = {'a', 'i', 'u', 'e', 'o'};
    int count = 0;

    for (int i = 0; i < word.length; i++) {
        for (int j = 0; j < vowels.length; j++) {
            if (word[i] == vowels[j]) {
                count++;
            }
        }
    }

    return count;
}
```

Latihan

3. Tentukan notasi Big O yang sesuai dari kode program berikut!

b.

```
public boolean checkItemInList(String item, String[] list) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] == item) {
            return true;
        }
    }

    return false;
}
```


thank you
danke
謝謝
ngiyabonga
شكراً جزيلاً
рахмат
teşekkür ederim
спасибо
gracias
dank je
tapadh leat
Баярлалаа
faafetai lava
kiitos
dankie
dhanyavad
vinaka
спасибі
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
bedankt
hvala
mauruuru
köszönöm
dziękuję
enkosi
bayarlalaa
grâcie
nanni
nandri
akun
dankon
aciu
djiere dieuf
tau
mochchakkeram
murakoze
chokrane
дякую
mamnun
go raibh maith agat
chnorakaloutioun
gratias ago
grâcies
sulpáy
taiku
obrigado
sobodi
dèkuji
sukriya
kop khun krap
sagolun
mèsi
didi madloba
kam sah hamnida
najis tuke
terima kasih
ありがとう
tanemirt
rahmet
grazie
arigatô
takk
dakujem
trugarez
мерси
merce
shukriya
dhanyavadagalu
diolch
rahmat
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci