Brute Force
&
Divide Conquer (DC)

Teaching Team

Algorithm and Data Structure

2022/2023

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?)

The Definition #1

- Brute force is a **straightforward approach**
- The basis for solving the brute force algorithm is obtained from the statement of the problem and the definition of the concepts involved.
- The brute force algorithm solves problems in a simple, direct and clear way.
- Brute force algorithm is more suitable for small problems because it has an easy implementation and a simple procedure.

The Definition #2

- Based on the :
  - problem statement
  - Definition of concept involved
- *brute force* solve the problem with
  - Very simple,
  - Direct,
  - *Obvious*
- *Just do it!* Or *Just Solve it!*

Kareteristik Algoritma Brute Force

Brute force algorithm is generally not "smart" and not efficient, because it requires a large number of computation and a long time to complete.

- The word "force" indicates "energy" than "brain"
- Brute force algorithm is also called naive algorithm

The brute force algorithm is more suitable for small problems

- Simple,
- Easy to implement

Brute force algorithms are often used as for a comparison with more “smart” algorithms.

Although it is not a mature method, almost all problems can be solved using the brute force algorithm.

- Difficult to show problems that can not be solved by the brute force method.
- In fact, there are problems that can only be solved by the brute force method.

Example #1 – Find a Max or Min value

Problem :

Given a list of n integers (a1, a2, ..., an). Find the Maximum element in the list

Algorithm:

Compare each element of the listing to find the biggest element

```
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
```

Example #2 - *String Matching*

Problem:

You are given:

a)text (text), i.e. (long) string with length **n** characters

b)b. pattern, i.e. string with length **m** characters (assuming: **m <n**)

Look for the first location in the text that **matches** the **pattern**

Algorithm:

1) The pattern is firstly matched (checked) from the **beginning** of the text.
2) By moving from left to right, compare each character in the pattern with the corresponding characters in the text until:
    - all characters compared match (a successful search), or
    - found a character mismatch (search not successful)
1) If the pattern has not found a match and the text has not been used, move the pattern one character to the right and repeat step 2.

Example #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

- The pattern found at the end of the text.
- Example:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaaab"
  - P: "aaab“

Best Case

- The pattern found at the beginning of the text:
- Example:

```
T:  String ini berakhir dengan zzz
P:  Str
```

Pros and Cons Brute Force

Pros

1. The brute force method can be used to solve almost all problems (wide applicability).
2. The brute force method is simple and easy to understand.
3. The brute force method produces a suitable algorithm for several important problems such as searching, sorting, string matching, matrix multiplication.
4. The brute force method produces a standardized algorithm for computational tasks such as the addition / multiplication of n numbers, determining the minimum or maximum elements in a table (list).

Cons

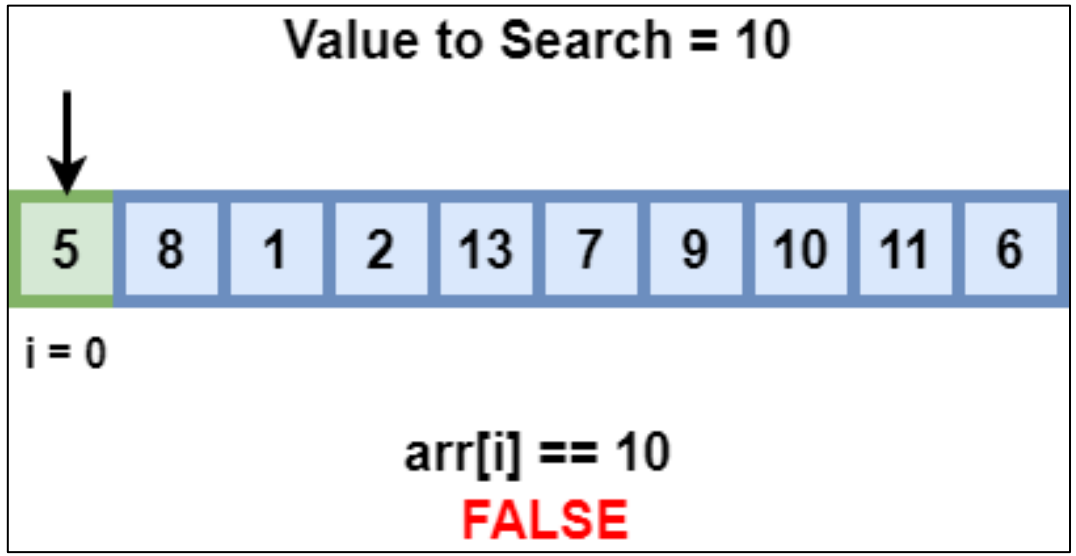
1. The brute force method rarely produces efficient algorithms.
2. Some brute force algorithms are very slow, so that they cannot be accepted.
3. Not as constructive / creative as other problem solving techniques.

Brute Force Implementation

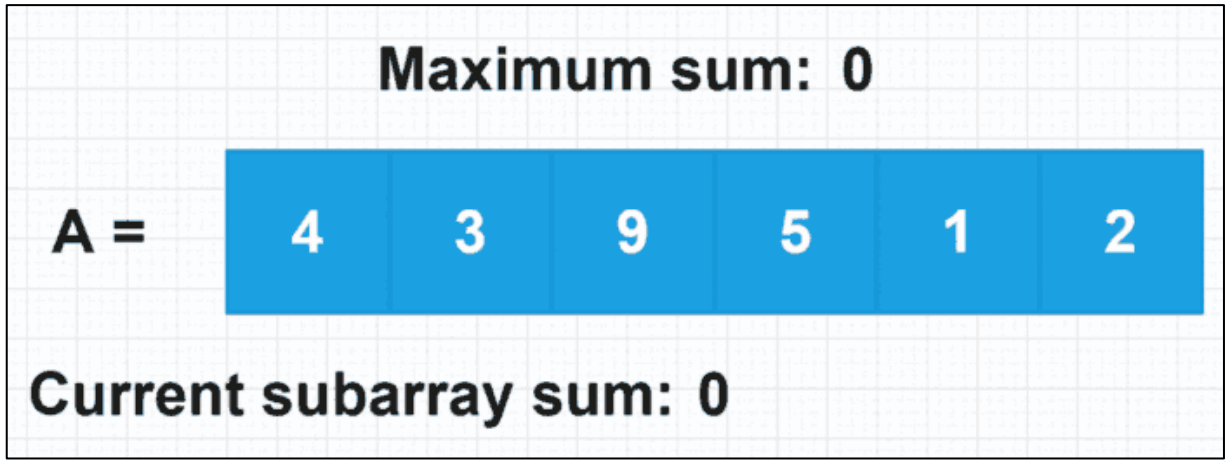
- *Sequential Search / Linear Search*
- *Bubble Sort*
- *Selection Sort*

Brute Force based Sequential Search

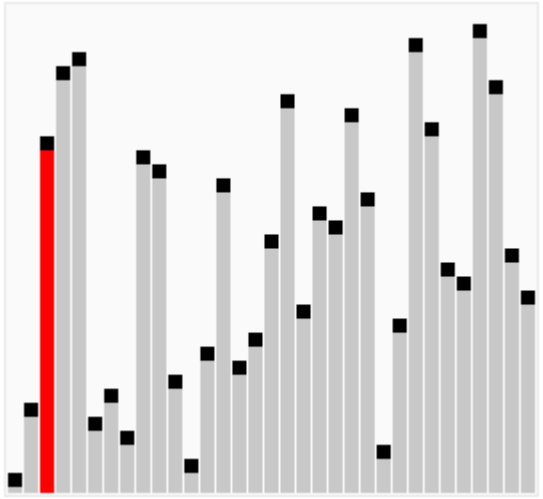
Searching



Highest Sum

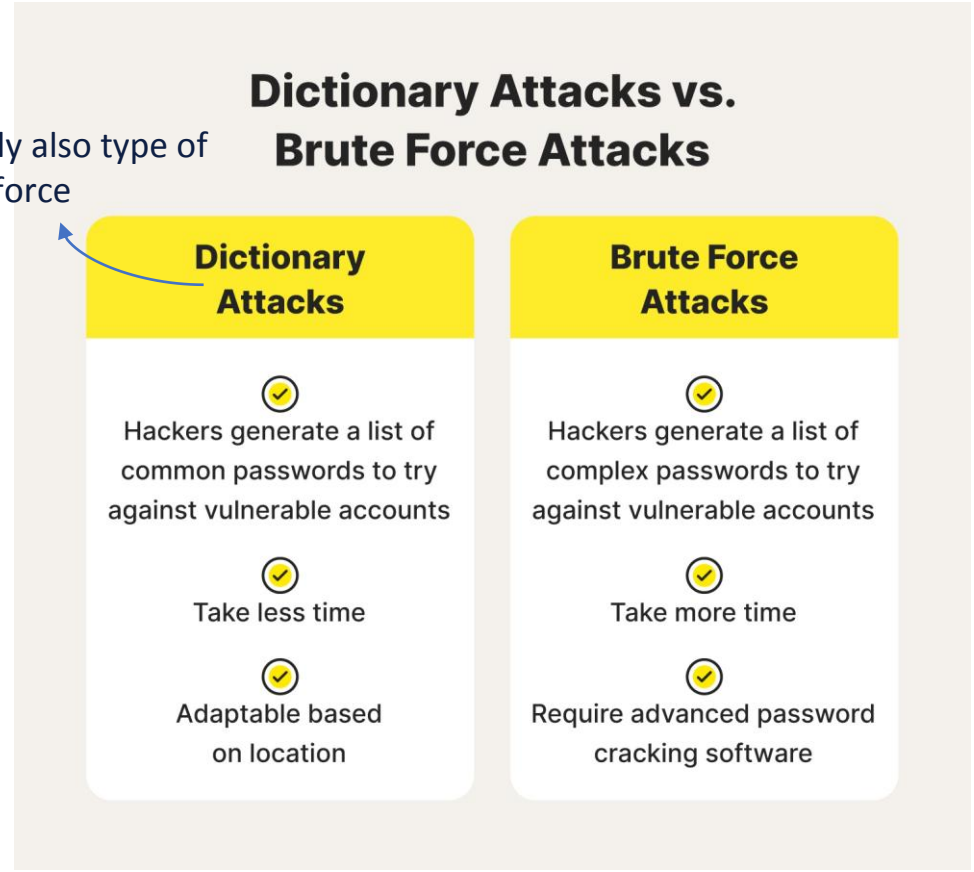


Brute Force based Bubble Sort

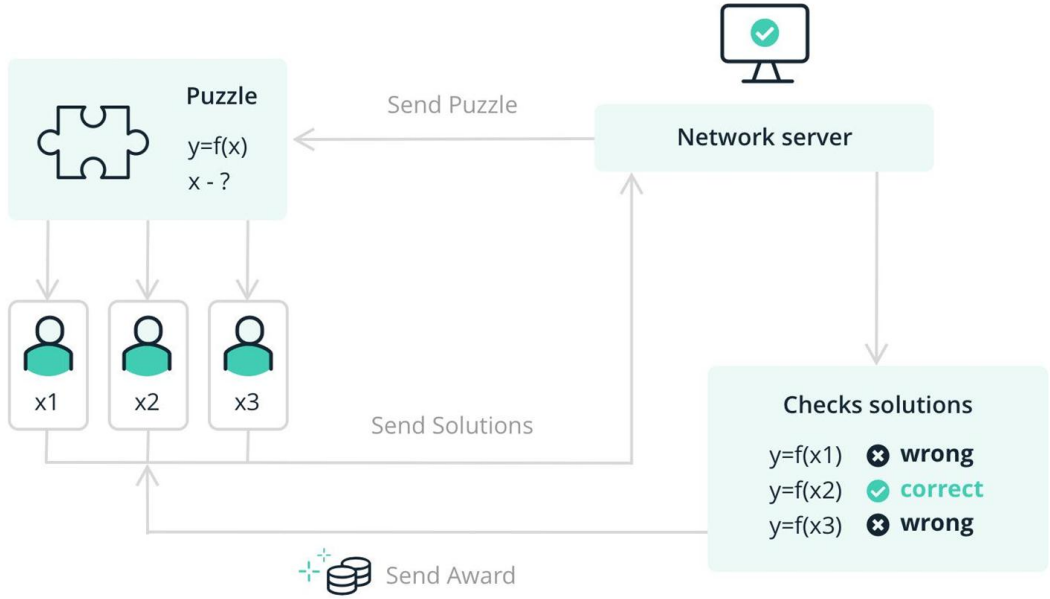


Trivia: *Brute Force* in a Real World

Actually also type of brute force



Cybersecurity



Proof-of-Work

Blockchain

Divide Conquer (DC)

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*.

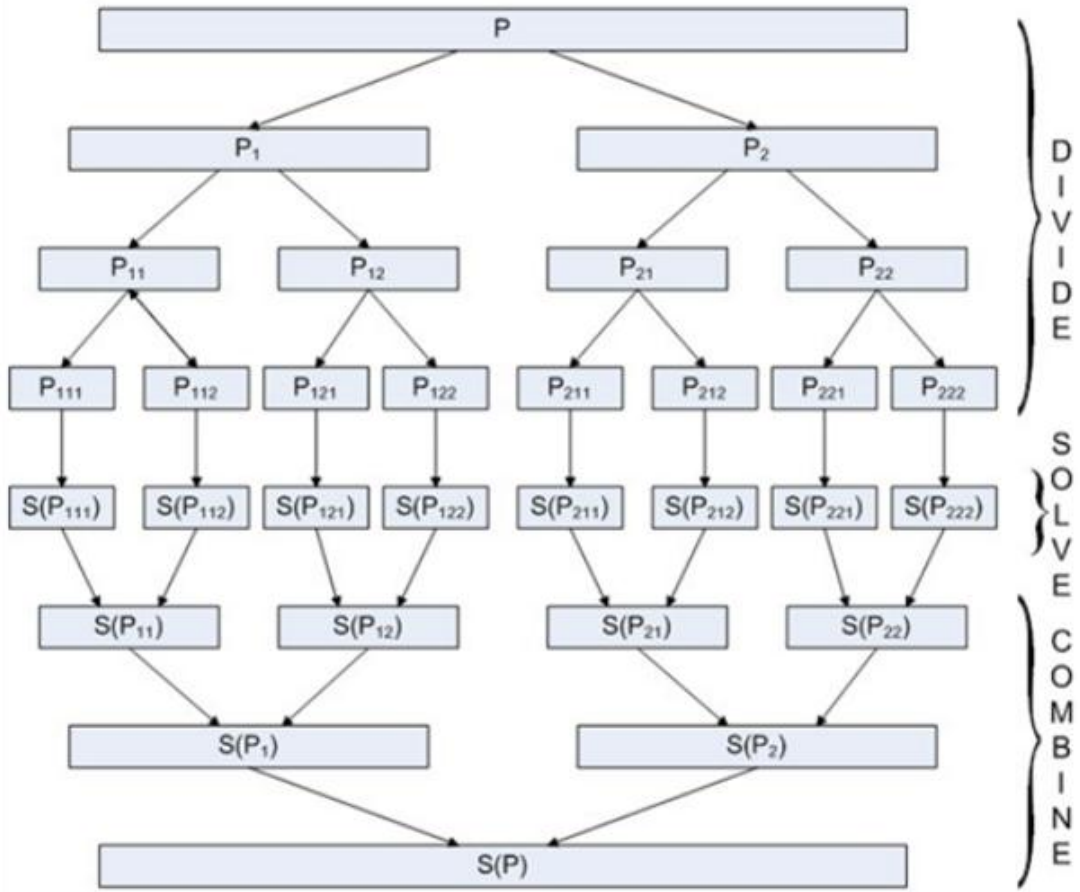
The Definition #1

- ***Divide***: dividing the original problem into several problems that are similar to the original problem but are smaller in size,
- ***Conquer***: solving each of the problems (recursively), and
- ***Combine****:* combining the solutions of each sub-problem to form the original problem solution

The Definition #2

- Problem objects are divided into a smaller size problem: input or instances of n-sized problems such as:
  - Table (array),
  - Matrix,
  - Exponent,
  - Etc., depending on the problem..
- Each of the problems has the same characteristics (the same type) with the original problem characteristics
- So the DC method usually uses recursive.

Ilustration of *Divide Conquer*

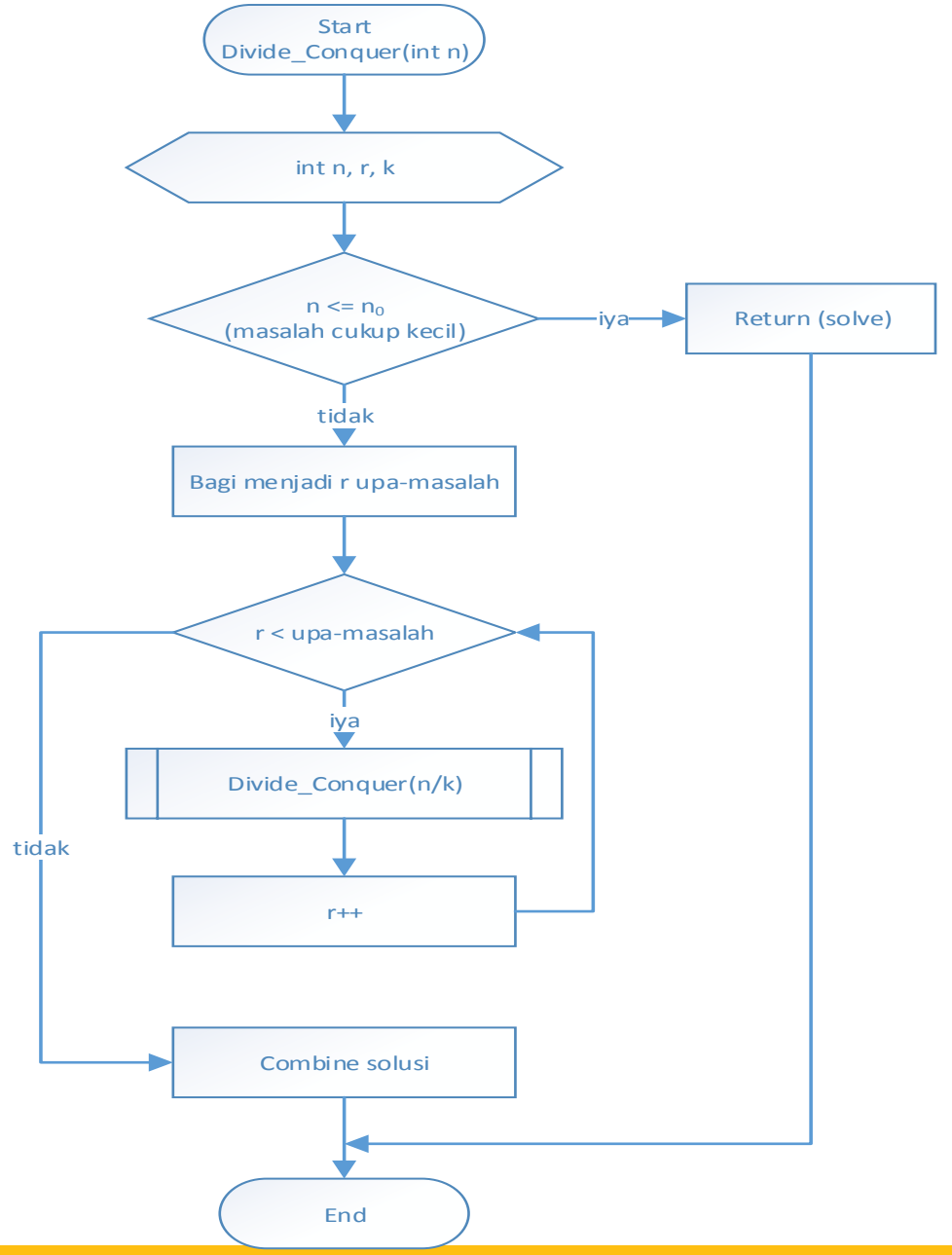


Keterangan:
P = persoalan
S = solusi

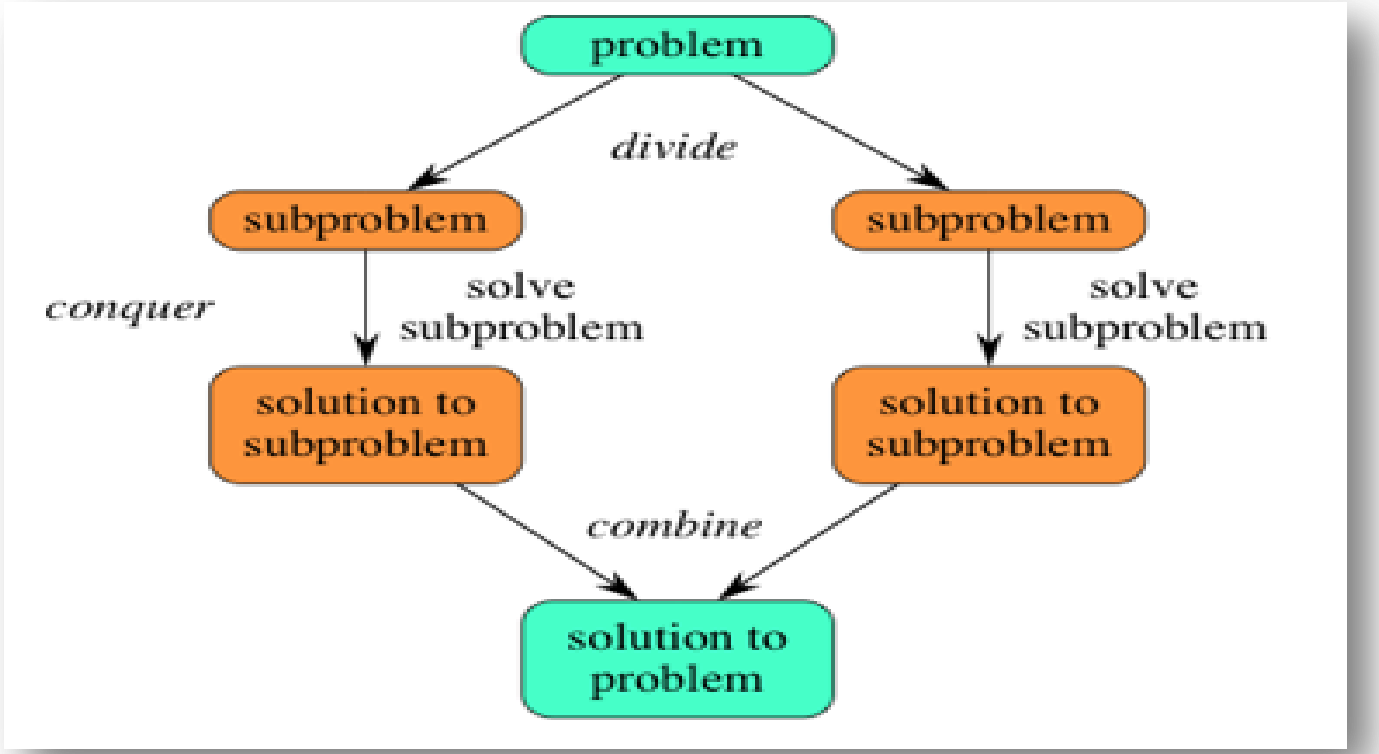
General Schema: Divide - Conquer

```
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
```

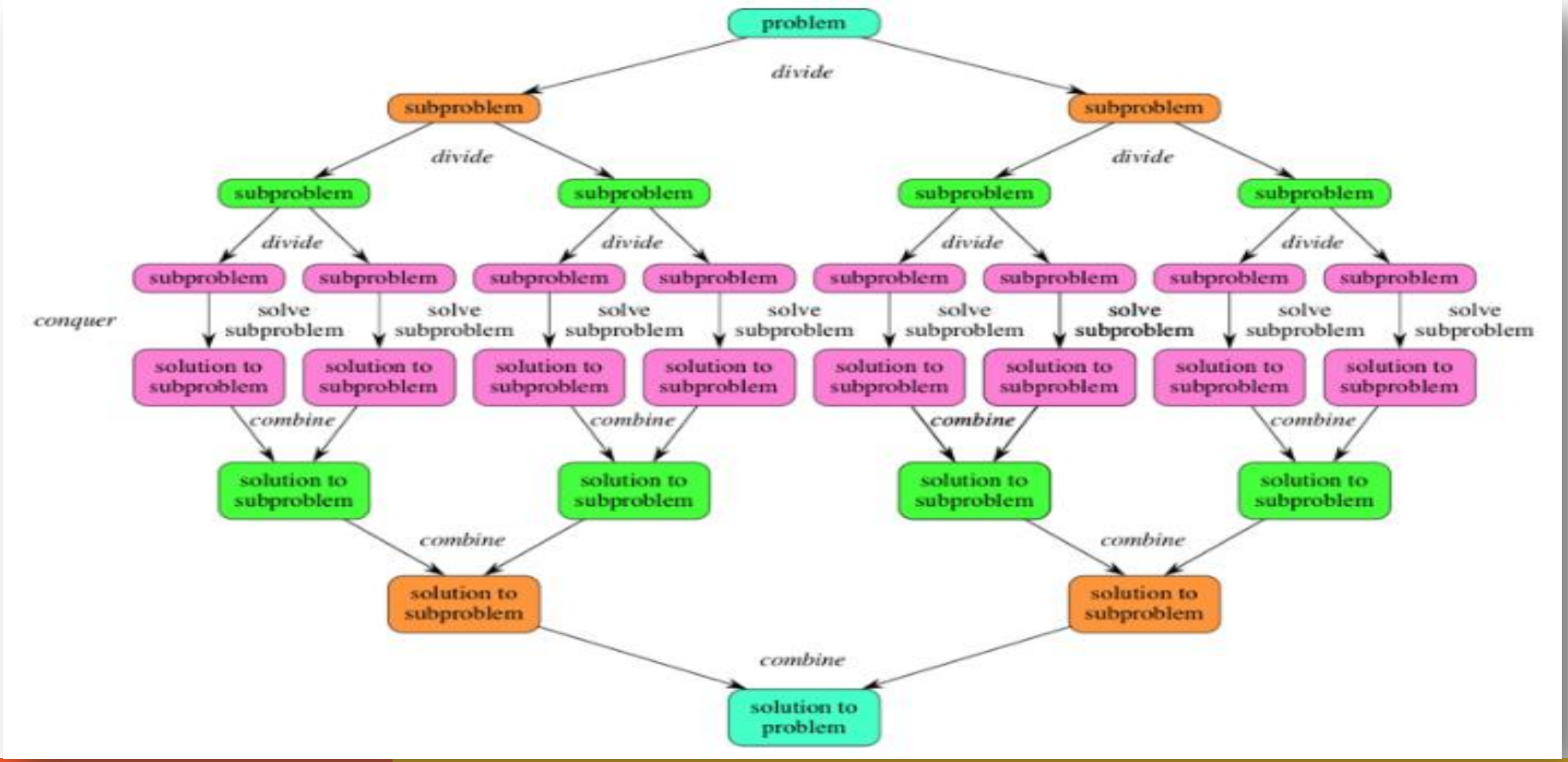
Flowchart General Form Divide and Conquer



Ilustration of Divide Conquer 1: recursive stage



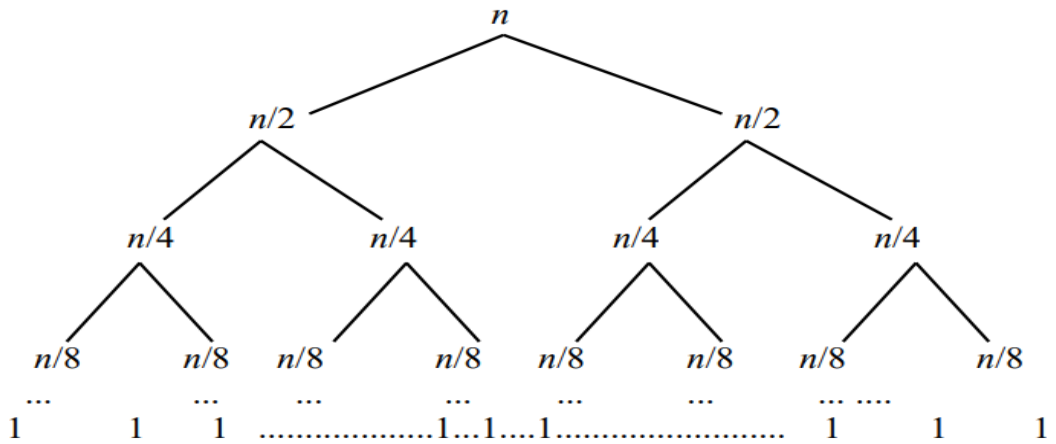
Ilustration of Divide Conquer: more than 2 recursive stages



Some Case Divide Conquer

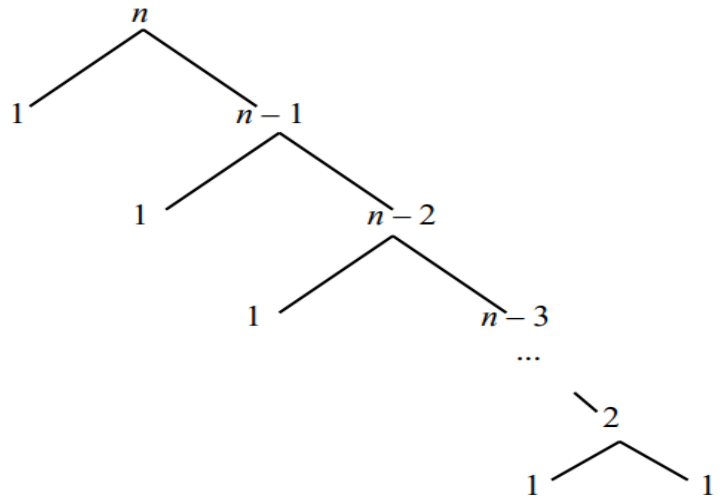
- Best Case

  The best case occurs when a pivot is a median element such that the two sub-tables are of the same size relative to each partition.

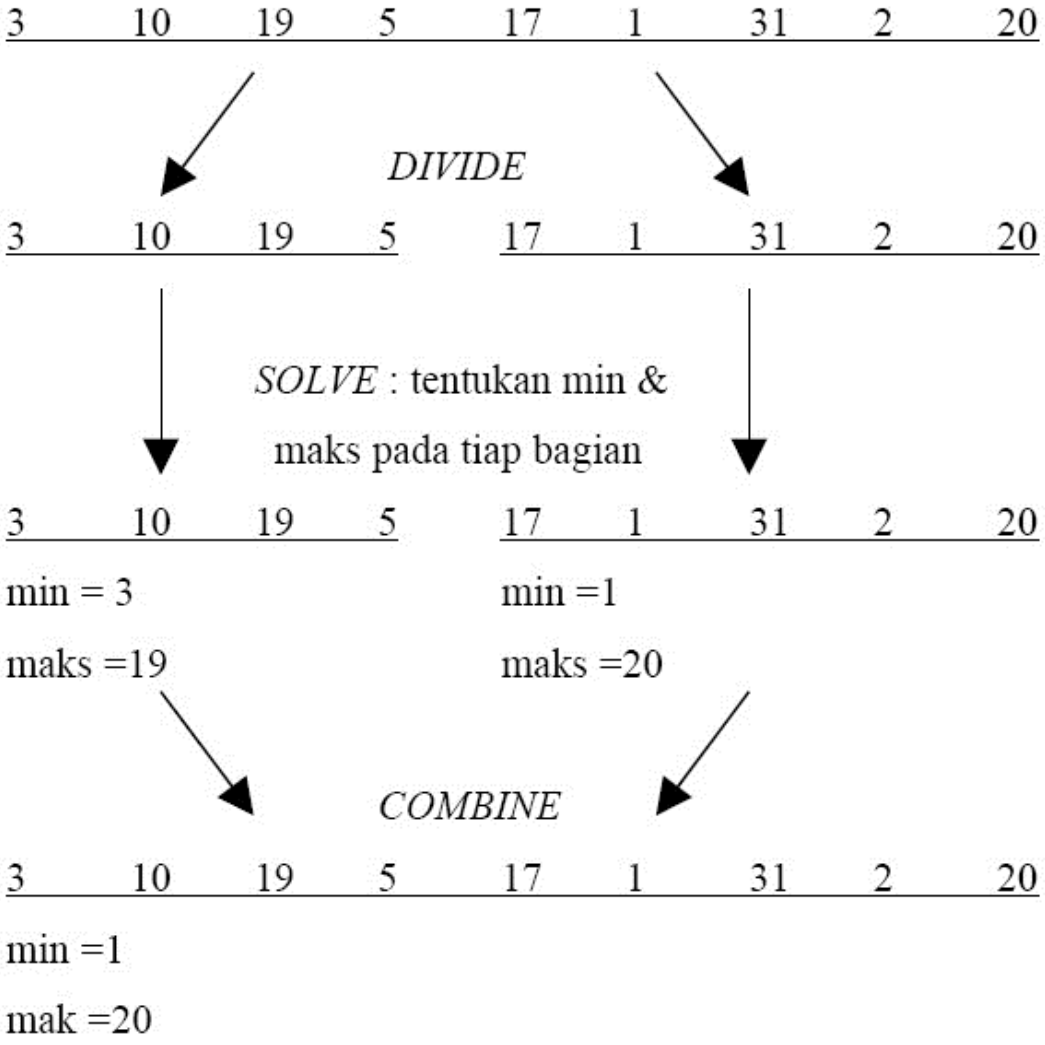


- Worst Case

  This case occurs when on every pivot partition there is always a maximum element (or minimum element) of the table. Case if the table is sorted ascending / descending and we want to order descending / ascending.



Example #1: Min and Max Value



Pros and Cons Divide Conquer #1

Pros

- Can solve difficult problems. Solving Divide and Conquer problems is a very effective way if the problem to be solved is complicated enough.
- Has a high algorithmic efficiency. This divide and conquer approach is more efficient in completing the sorting algorithm.
- Can work in parallel. Divide and Conquer has been designed to work on machines that have multiple processors. Especially machines that have a memory sharing system, where data communication between processors does not need to be planned in advance, this is because solving sub-routines can be done on other processors.
- Memory access is quite small. For memory access, Divide and Conquer can improve the efficiency of existing memory quite well. This is because, sub-routine requires less memory than
- the main problem.

Pros and Cons Divide Conquer #2

Cons

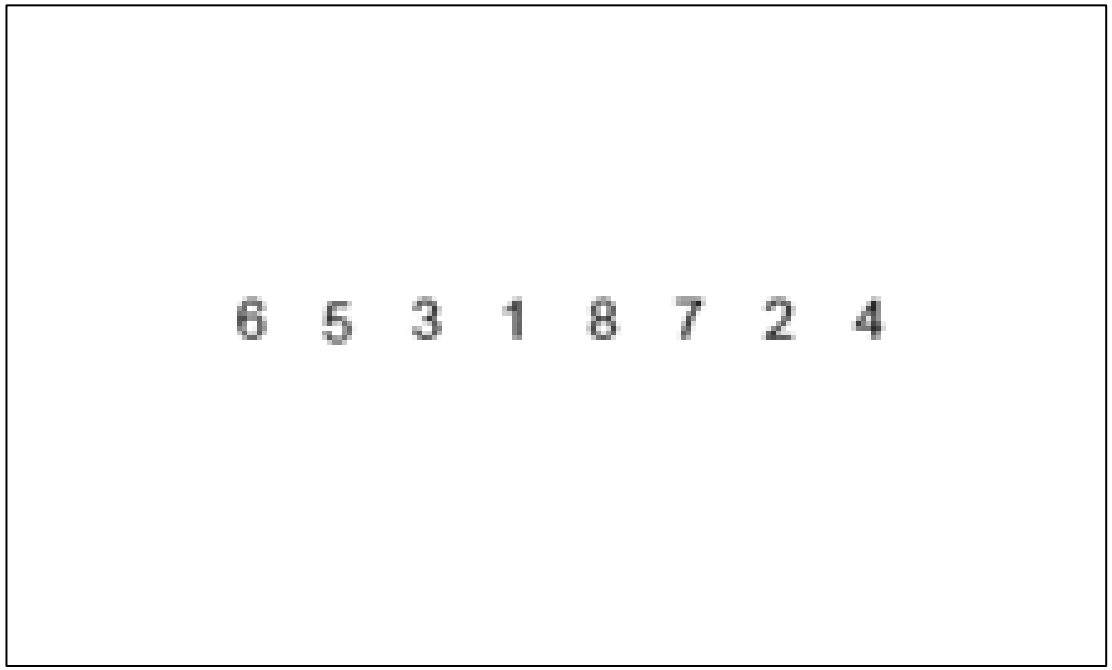
- The slow process of iteration
  - Slow looping The process of calling sub-routine (can slow down the looping process) will cause excessive call stack full. This can be a significant burden on the processor. More complicated for simple problems
- More complicated for simple problems
  - For relatively simple problem solving, a sequential algorithm is proven to be easier to create than the divide and conquer algorithm.

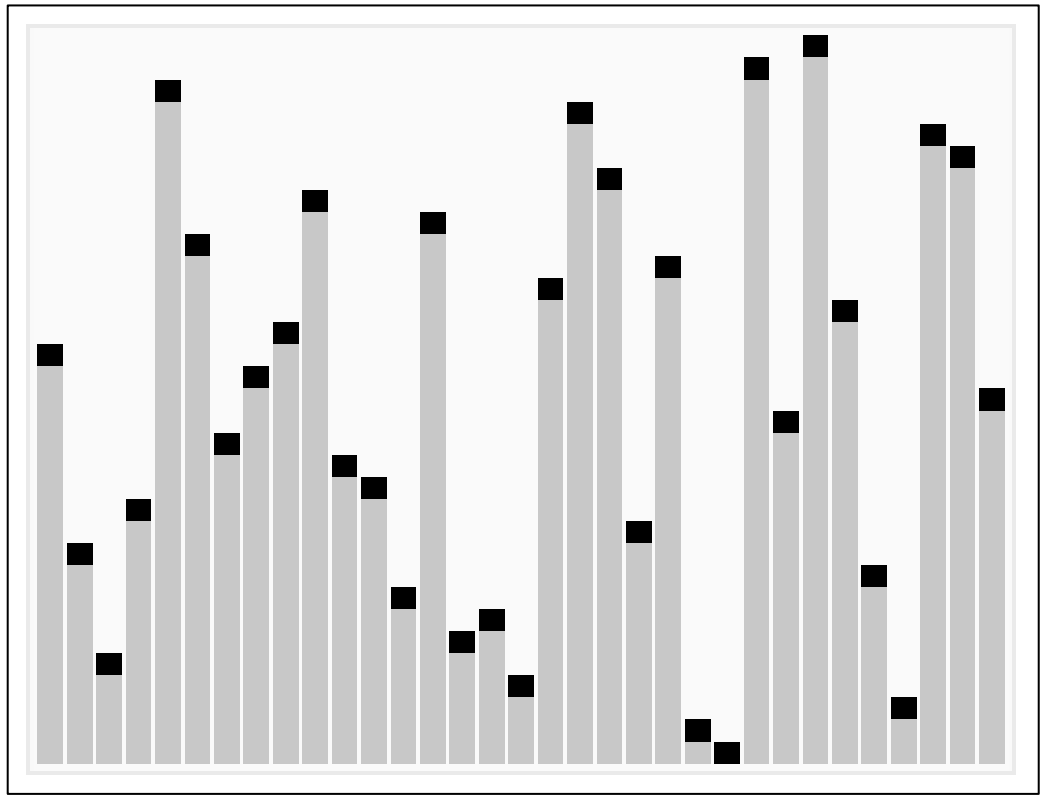
Divide Conquer Implementation

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

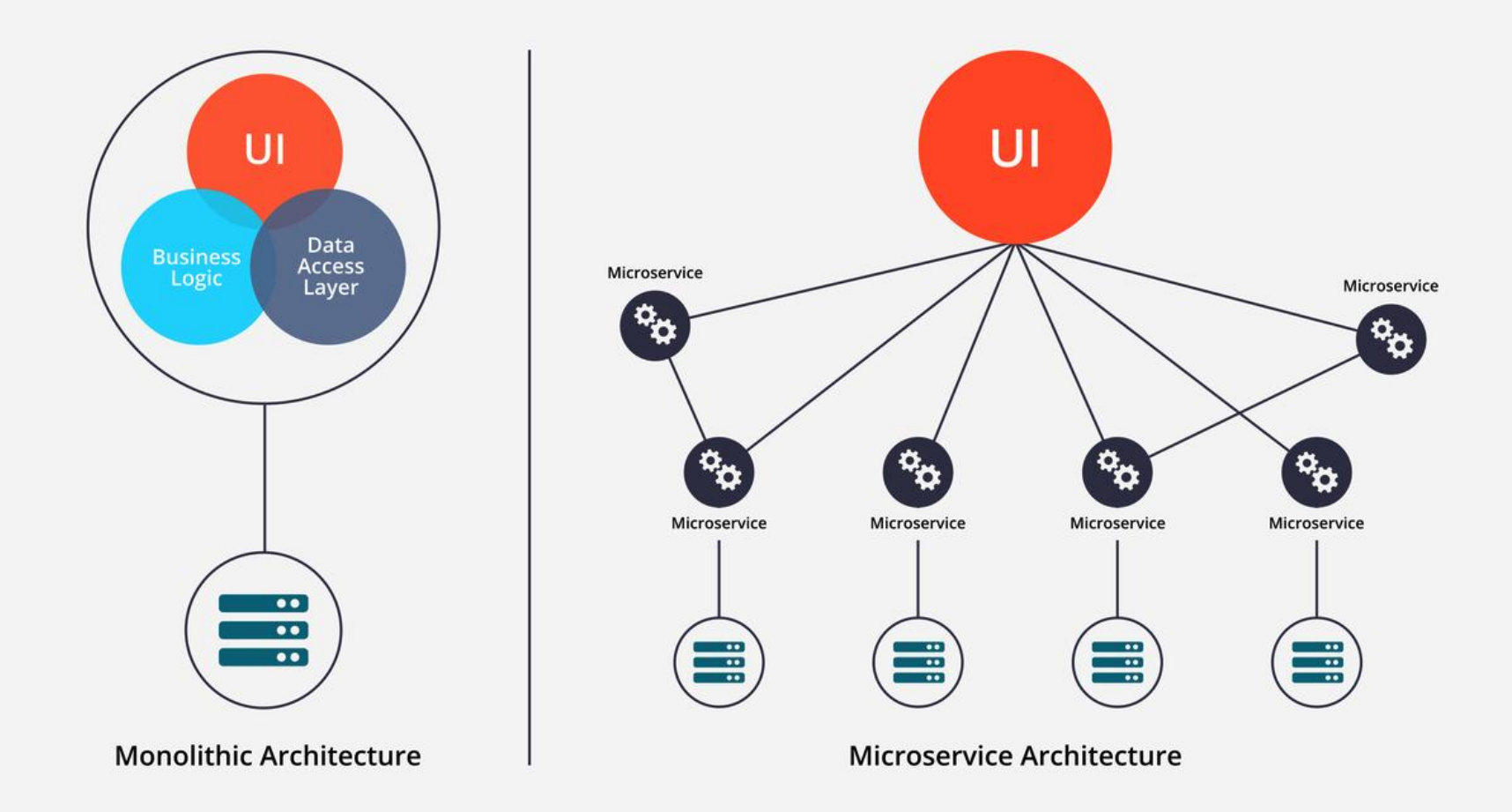
Ilustration of Merge Sort based on the *Divide Conquer*

MERGE SORT





Trivia: *Divide Conquer*



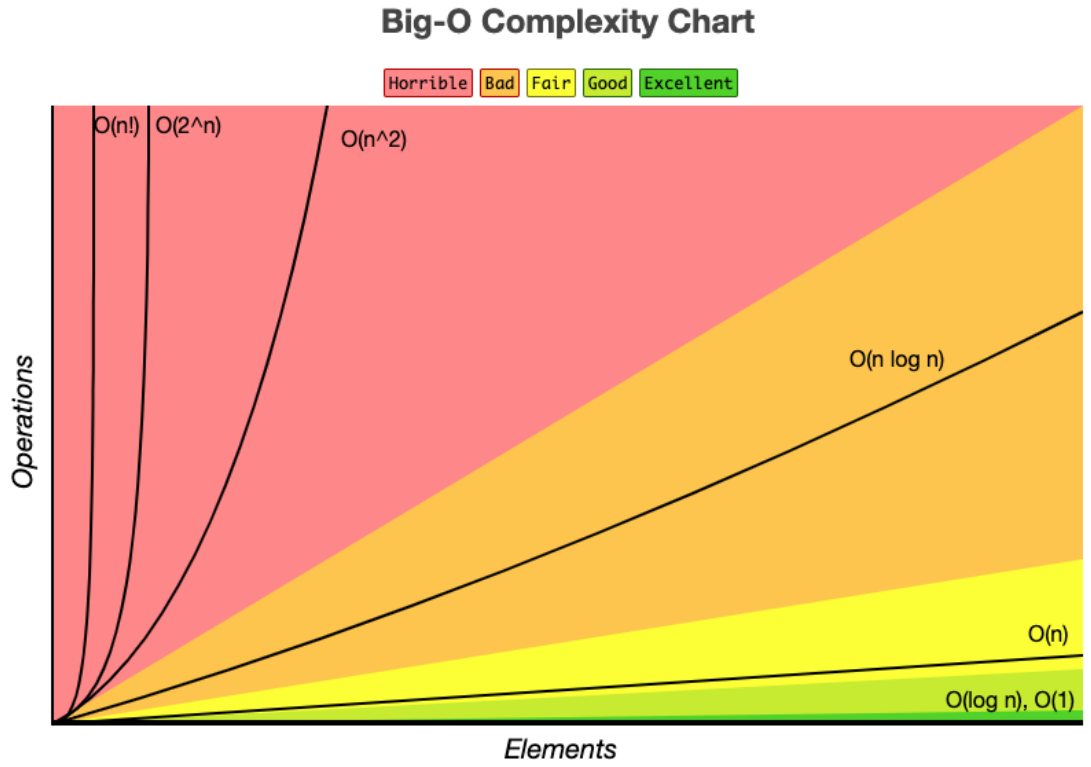
Arsitektur *Microservices*

Big O Notation

Is the algorithm efficient?

Big O Notation

- Analysis of algorithms for time or memory space complexity
- Can be measured or viewed based on the worst-case, best-case, average-case.
- The fastest to the slowest:
  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!)



O(1) Notation

Example

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

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

time complexity O(1) because it only runs one statement which is return instruction, no matter what input is entered into the function

O(log n) Notation

Example

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

If n = 8, then

Output
Hasil : 1
Hasil : 2
Hasil : 4

It will run log(8) = 3 times

O(n) Notation

Example

```
double average(double[] numbers) {
        double sum = 0;
        for(double number: numbers) {
                sum += number;
        }
        return sum / numbers.length;
}
```

- The above function has time complexity O (n) because it will run looping to sum the numbers from the array. The number of loops depends on the length of the array entered into the function.
- If the array has 3 components [2,3,4], then the function will sum in sequence 2, 3, and 4, then return the average. Thus, an array that has a length of 3, the function will loop 3 times as well, and so on.

O(n log n) Notation

Example

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

If n = 8, then it will run *8 * log(8) = 8 * 3 = 24* times.

$O(n^p)$ Notation

Example

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

If n = 8, then it will run 8^2 = 64 times

O(n^p) Notation – Another Example

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

How many times does count++ run with any n value?

- When i = 1, it will run once.
- When i = 2, it will be run 2 times.
- When i = 3, it will be run 3 times.
- etc…

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

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

O(k^n) Notation

- Example

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

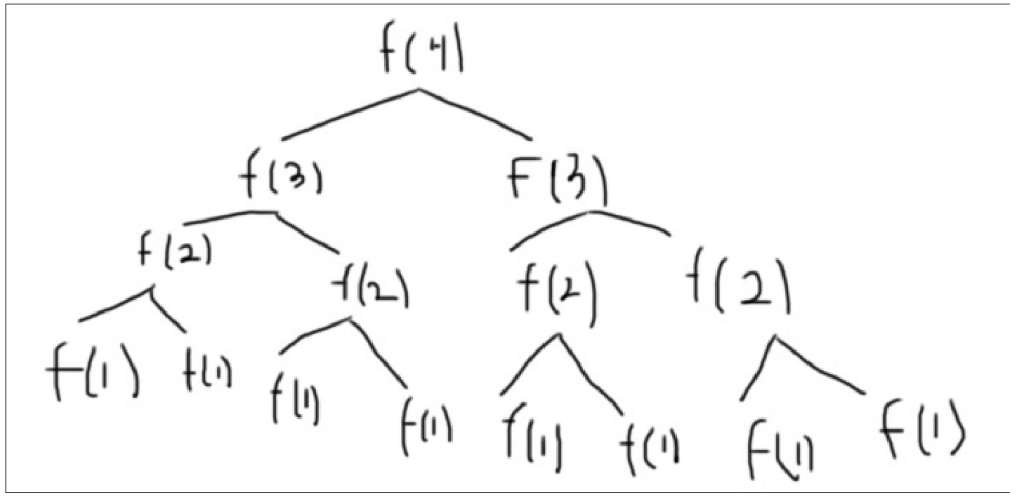
Can be seen in cases or factors that are respondent to the size of the input

If n = 8, then it will loop
2^8 = 256 times

O(k^n) Notation – Another Example

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

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

If we call **func(4)**



The time complexity is **O**(2^**n**)

O(n!) Notaion

Example

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

If n = 8 then it will be 8!
= 40320 times

Rule of Big O

- Include the most significant notation(usually “+” or O(1) notation is removed)
- “If” is usually “ + ”
- “for” is usually “ * ”
- Constant could be removed (example: 2 O(n) turns to O(n))

Big O Notation

- *Big-O notation is a way of converting the overall steps of an algorithm into algebraic terms, then excluding lower order constants and coefficients that don't have that big an impact on the overall complexity of the problem.*

```
Regular       Big-O

2             O(1)    --> It's just a constant number

2n + 10       O(n)    --> n has the largest effect

5n^2          O(n^2) --> n^2 has the largest effect
```

Example

```
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]);
            }
        }
    }
}
```

Example

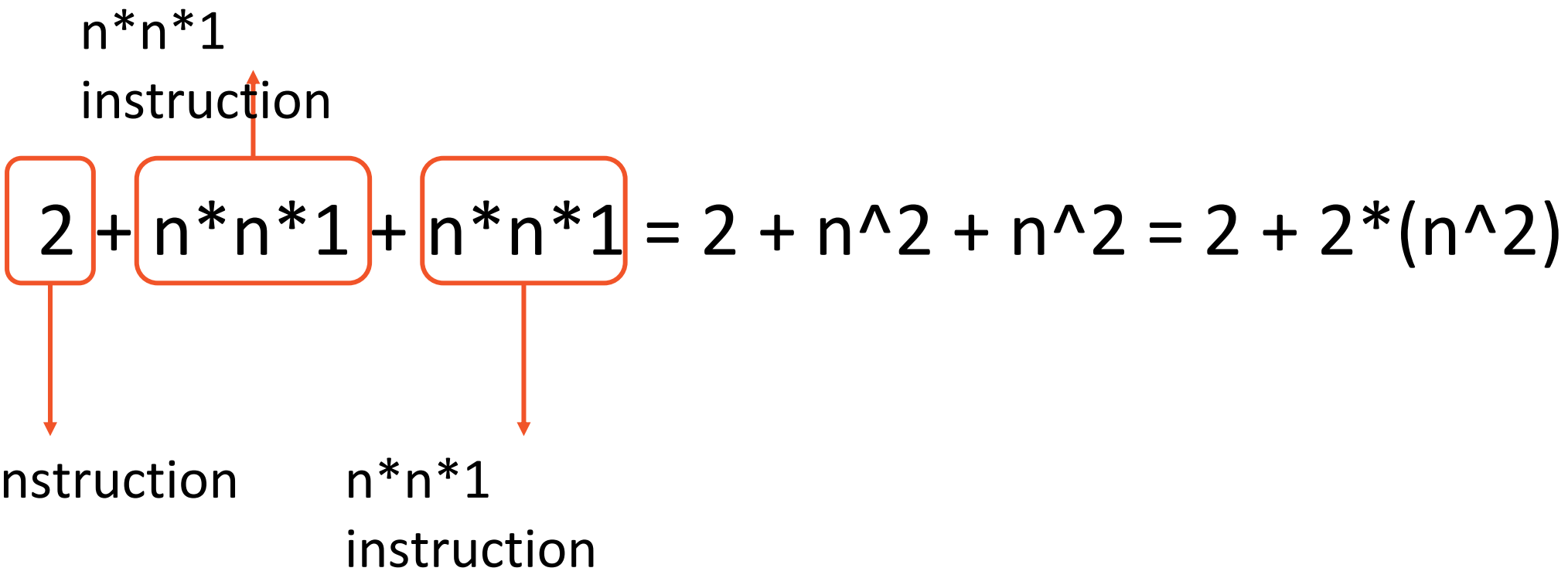
```java
public class ContohBigO{
    public static void contohBig0(int[] angka){
        System.out.println("Pairs: ");
        int n = angka.length;

        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]);
            }
        }
    }
}
```

2 instruction

n*n*1 instruction

n*n*1 instruction

Example



Example:

If n = 10 then the instruction will be run 2 + 2*(10^2) = 202 times

Exercises

1. Create the flowchart of n factorial using both of Brute Force and Divide Conquer!
2. Create the flowchart of the exponential value of n using both of Brute Force and Divide Conquer!
3. What is the Big O notation from the following code?

a.

```
function countVowels(word) {
    var vowels = ['a', 'i', 'e', 'o', 'u'];
    var count = 0;
                for (var i = 0; i < word.length; i++) {
        for (var j = 0; j < vowels.length; j++) {
            if (word[i] === vowels[j]) {
                count++;
            }
        }
    }
    return count;
}
```

b.

```
function itemInList(check, list){
        for (var i = 0; i < list.length; i++){
                if (list[i] === check) return true;
        }
        return false;
}
```


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