Data Structure and Algorithm Practicum
Brute Force and Divide Conquer

Name
Muhammad Baihaqi Aulia Asy'ari

NIM
2241720145

Class
1I

Department
Information Technology

Study Program
D4 Informatics Engineering

2.1 Purpose

1. Students know about the use of the brute force algorithm and the devide-conquer algorithm
2. Students are able to apply the use of the brute force and devide-conquer algorithms

2.2 Theory

Computational problem solving can be solved in various ways. Brute force and divide conquer algorithms are two different types of approaches that are usually used in problem solving.

2.2.1 Brute Force

Brute force is a straightforward approach to solving a problem. The basis for solving the brute force algorithm is obtained from the statement of the problem (problem statement) and the definition of the concepts involved. The brute force algorithm solves problems very simply, directly, clearly. Brute force algorithms are generally not "smart" and not efficient, because they require a large amount of computation and a long time to complete. Sometimes the brute force algorithm is also called a naive algorithm. Brute force algorithm is more suitable for small problems because it is easy to implement and simple procedures. Brute force algorithms are often used as a basis for comparison with more sophisticated algorithms. Although it is not a good method, almost all problems can be solved using the brute force algorithm. There are several weaknesses and strengths of the brute force algorithm: Advatages:

1. The brute force method can be used to solve most problems (wide applicability).
2. The brute force method is simple and easy to understand.
3. The brute force method produces a decent 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).

Weakness:

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

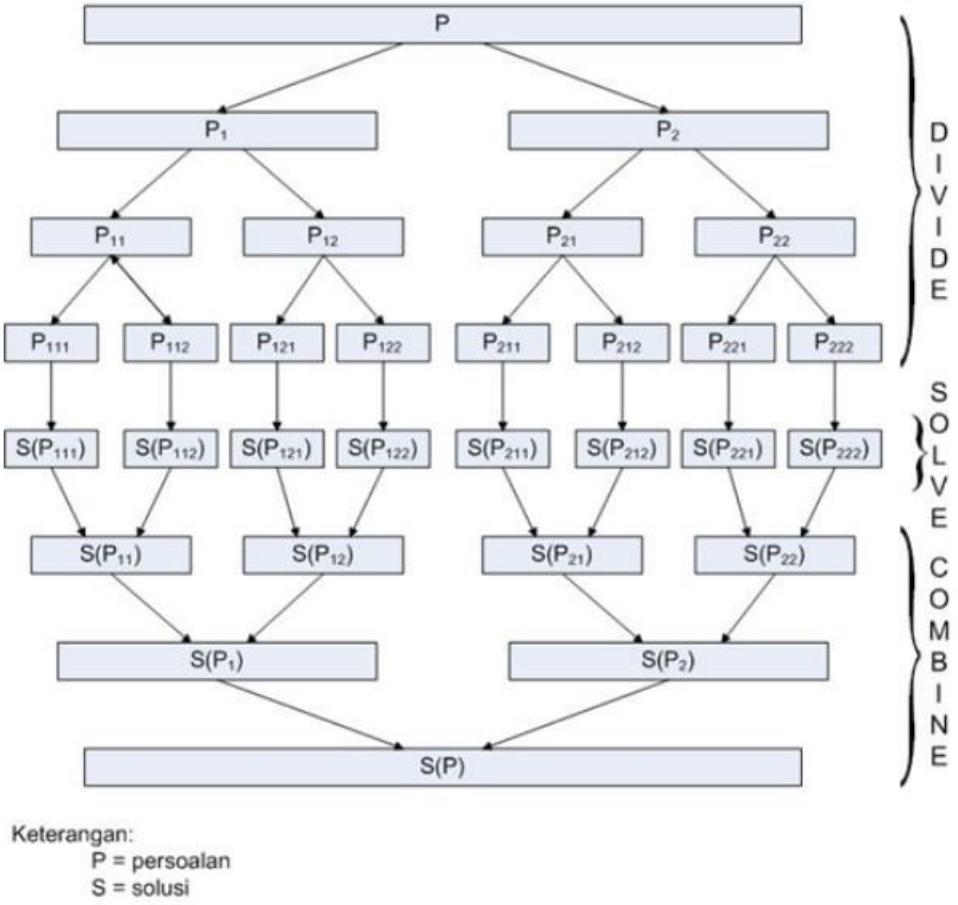
Bruteforce can also be implemented in searching and sorting methods which will be explained in more detail in the following weeks. In the searching process, the principle of completion with the bruteforce method can be seen in the sequential search method. Sequential Search is also called Linear Search. The search process compares key values with all value elements. Compare key values with the first element to the last element, or, The process stops if the key value matches the element value without having to compare all elements. As for sorting, bruteforce is used as the basic principle of the stages of the bubble sort and selection sort methods. The Bubble Sort algorithm is a sorting process that gradually moves to the right position (Bubble). This algorithm will sort the data from the largest to the smallest (ascending) or vice versa (descending). Simply stated, the Bubble Sort algorithm can be defined as sorting by exchanging data with the data next to it continuously until there is no change in one iteration. The selection sort method is an improvement of the bubble sort method by reducing the number of comparisons. Selection sort is a sorting method by finding the smallest data values starting from data in position 0 to position N-1. If there are N data and data collected from sequence 0 to N-1 then the sorting algorithm with the selection sort method is as follows: 1. Find the smallest data in the interval j = 0 to j = N-1 2. If at the post position is found the smallest data, exchange the data in the pos position with the data in position i if k. 3. Repeat steps 1 and 2 with j = j + i until j = N-1, and so on until j = N - 1.

2.2.2 Divide Conquer

Divide and Conquer was originally a military strategy known as divide ut imperes. Now the strategy is a fundamental strategy in computer science called Divide and Conquer. Divide means dividing the problem into several problems that have similarities to the original problem but are smaller in size (ideally about the same size), Conquer is a way to solve (solve) each one recursively, and Combine whose goal is to combine the solutions of each problem thus forming the original problem solution. The object of the problem, divided into input (input) or instances of size n such as:

- table (array),
- matrix,
- exponent,
- etc., depending on the problem.

The stages of divide, conquer and combine can be seen in the image below.



Each problem has the same characteristics as the original problem, so the Divide and Conquer method is more naturally expressed in a recursive scheme. The general pseudocode for solving problems with the divide conquer algorithm is:

```
procedure DIVIDE_and_CONQUER(input n : integer)
{ Resolve problems with the D-and-C algorithm.
    Input: input size n
    Output: the solution of the original problem }
Declaration
        r, k : integer
Algorithm
    if n <= n0 then { the size of the problem is small enough }
        SOLVE this n-sized sub-problem
    else
        Divide r into sub-problems, each of which is sized n / k
        for each of the likeness-problems do
            DIVIDE_and_CONQUER(n/k)
```

```
            endfor
            COMBINE the solution of the problem solution becomes the original
problem solution
        endif
```

Weakness Divide and Conquer

- Slow iteration process
  - 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 simple problem solving, a sequential algorithm is proven to be easier to create than the divide and conquer algorithm.

Advantages of Divide and Conquer

- Can solve difficult problems. Solving Divide and Conquer problems is a very effective way if the problem to be solved is complicated enough.
- Has 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.

Divide conquer is also the basis for searching and sorting. The sorting method that uses basic divide conquer is merge sort. The sorting method for merge sort is the advanced sorting method, the same as the Quick Sort method. This method also uses the concept of devide and conquer which divides S data into two groups, namely S1 and S2 which are not intersected (disjoint). The process of data sharing is done recursively until the data cannot be divided again or in ot the data in sub sections becomes single. After the data cannot be divided again, the me process is carried out between sub-sections by paying attention to the desired data sequence (ascending /

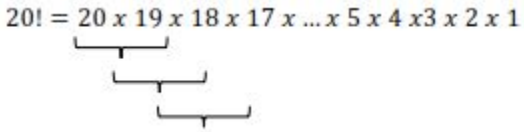
small to large or descending / large to small). This merging process is carried out until all data is combined and ordered in the desired order. While searching methods that use a way of sharing solutions such as divide conquere is binary search. Binary search is an algorithm for passing searches in an ordered array. If we do not know how integer information is in an array, then the use of binary search will be inefficient, we must sort first or use another method, namely linear search. But if we already know that integers in organized arrays are either ascending or descending, then we can quickly use the binary search algorithm.

2.3 Calculating Factorial Values with Brute Force and Divide and Conquer Algorithms

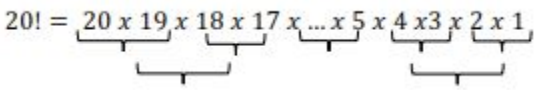
Consider the following Class Diagram:

Faktorial
value: int
faktorialBF(): int faktorialDC(): int

Based on the class diagram above, a class program will be created in Java. To calculate the factorial value of a number using 2 types of algorithms, Brute Force and Divide and Conquer. If described there are differences in the calculation process of the 2 types of algorithm as follows:
The stages of factorial value searching with Brute Force algorithm:



The stages of factorial value searching with the Divide and Conquer algorithm:



2.3.1 Practicum

1. Create a new Project, with the name AlgoStruDat / Project Name Equated to last week. Make a package with the name **Week3**, make a new class with the name **Faktorial**.

2. Complete the Faktorial class with the attributes and methods described in the class diagram above:

   (a) Add value attributes

   ```
   public int num;
   ```

(b) Add method `faktorialBF()`

```
public int faktorialBF(int n) {
    int fakto = 1;
    for (int i = 1; i <= n; i++) {
        fakto = fakto * i;
    }
    return fakto;
}
```

(c) Add method `faktorialDC()`

```
public int faktorialDC(int n) {
    if (n==1) {
        return 1;
    }
    else
    {
        int fakto = n * faktorialDC(n-1);
        return fakto;
    }
}
```

3. Run the `Faktorial class` y creating a new `MainFaktorial` class.

(a) In the main function, provide input to input the number of numbers to find the factorial value

```
Scanner sc = new Scanner(System.in);
System.out.println("================================
↪  ================================");
System.out.print("Input the number of elements you want to
↪  count : ");
int elemen = sc.nextInt();
```

(b) Create an Array of Objects on the main function, then input some values that will be factorially calculated

```
Faktorial [] fk = new Faktorial[elemen];
for (int i = 0; i < elemen; i++) {
    fk[i] = new Factorial();
    System.out.print("Input the data value to-"+(i+1)+" : ");
    fk[i].num = sc.nextInt();
}
```

(c) Display the results of calling method `faktorialDC()` dan `faktorialBF()`

```
        System.out.println("================================
        ↪  ================================");
        System.out.println("Factorial Result with Brute Force");
        for (int i = 0; i < elemen; i++) {
            System.out.println("Factorial of value"+fk[i].num+" is :
            ↪  "+fk[i].faktorialBF(fk[i].num));
        }

        System.out.println("================================
        ↪  ================================");
        for (int i = 0; i < elemen; i++) {
            System.out.println("Factorial of value"+fk[i].num+" is :
            ↪  "+fk[i].faktorialDC(fk[i].num));
        }
        System.out.println("================================
        ↪  ================================");
```

(d) Make sure the program is running well!

1. Faktorial.java

```
package Faktorial;

public class Faktorial {
    public int num;
    public int faktorialBF(int n) {
        int fakto = 1;
        for (int i = 1; i <= n; i++) {
            fakto = fakto * i;
        }
        return fakto;
    }
    public int faktorialDC(int n) {
        if (n==1) {
            return 1;
        }
        else
        {
            int fakto = n * faktorialDC(n-1);
            return fakto;
        }
    }
}
```

2. MainFaktorial.java

```
package Faktorial;

public class MainFaktorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
            System.out.println("================================
            ↪  ================================");
            System.out.print("Input the number of elements you
            ↪  want to count : ");
            int elemen = sc.nextInt();
            Faktorial [] fk = new Faktorial[elemen];
            for (int i = 0; i < elemen; i++) {
                fk[i] = new Factorial();
                System.out.print("Input the data value
                ↪  to-"+(i+1)+" : ");
                fk[i].num = sc.nextInt();
            }
            System.out.println("================================
            ↪  ================================");
            System.out.println("Factorial Result with Brute
            ↪  Force");
            for (int i = 0; i < elemen; i++) {
                System.out.println("Factorial of value
                ↪  "+fk[i].num+" is :
                ↪  "+fk[i].faktorialBF(fk[i].num));
            }

            System.out.println("================================
            ↪  ================================");
            for (int i = 0; i < elemen; i++) {
                System.out.println("Factorial of value
                ↪  "+fk[i].num+" is :
                ↪  "+fk[i].faktorialDC(fk[i].num));
            }
            System.out.println("================================
            ↪  ================================");
    }
}
```

2.3.2 Verification of Practicum Results

```
============================================================
Input the number of elements you want to count : 3
Input the data value to-1 : 5
Input the data value to-2 : 8
Input the data value to-3 : 3
============================================================
Factorial Results with Brute Force
Factorial of value 5 : 120
Factorial of value 8 : 40320
Factorial of value 3 : 6
============================================================
Factorial Results with Divide and Conquer
Factorial of value 5 : 120
Factorial of value 8 : 40320
Factorial of value 3 : 6
============================================================
```

Result:

```
PS D:\Kuliah>  d:; cd 'd:\Kuliah'; & 'C:\Program
↪  Files\Java\jdk-18.0.2.1\bin\java.exe'
↪  '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
↪  'C:\Users\ASUS\AppData\Roaming\Code\User\workspaceStorage\
↪  ce3fcb236261368a6cbd019dc8ddda8b\redhat.java\jdt_ws\
↪  Kuliah_28156aa7\bin' 'Faktorial.MainFaktorial'
============================================================
Input the number of elements you want to count : 3
Input the data value to-1 : 5
Input the data value to-2 : 8
Input the data value to-3 : 3
============================================================
Factorial Result with Brute Force
Factorial of value 5 is : 120
Factorial of value 8 is : 40320
Factorial of value 3 is : 6
============================================================
Factorial of value 5 is : 120
Factorial of value 8 is : 40320
Factorial of value 3 is : 6
============================================================
```

2.3.3 Questions

1. Explain the Divide Conquer Algorithm for calculating factorial values!
   `Answer:`
   The algorithm check if the parameter is 1. If it is, it will return 1. When the parameter anything else but 1, then it will calculate the parameter multiplied by the return of factorial divide and conquer function with the parameter of the original parameter minus one and then return the result of that calculation. Because of this any number else then one will recursively call it self with the parameter number minus one than before until the parameter becomes one.

2. In the implementation of Factorial Divide and Conquer Algorithm is it complete that consists of 3 stages of divide, conquer, combine? Explain each part of the program code!
   `Answer:`
   . The divide stage is implemented when the factorialDC function call it self. When a recursion happend the process of the previous function call is halted until the current function calculate has return a value. These function call happend until the function met its halting condition.
   . The conquer stage is process when the function calls has collaps on itself after meeting its halting condition. The function collaps by returning value from the function with halting condition as its parameter.
   . The combine stage happend when the function is processing its return value. The return value of the function is calculated before the value is sent to the previous function call. The previous function call will aslo do the same thing until there is no more function waiting for a value or to put it in another term the function will stop collaps on itself when it has reach the first call function on the stack.

3. Is it possible to repeat the factorial BF() method instead of using for? Prove it!
   `Answer:`
   Yes.

```
public int faktorialBF(int n) {
    int fakto = 1;
    int i = 0;
    While(i <= n) {
        fakto = fakto * i;
        i++
    }
    return fakto;
}
```

4. Add a check to the execution time of the two types of methods!

```
package Faktorial;

import java.util.Scanner;

public class MainFaktorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
            System.out.println("================================
            ↪  ================================");
            System.out.print("Input the number of elements you
            ↪  want to count : ");
            int elemen = sc.nextInt();
            Faktorial [] fk = new Faktorial[elemen];
            for (int i = 0; i < elemen; i++) {
                fk[i] = new Faktorial();
                System.out.print("Input the data value
                ↪  to-"+(i+1)+" : ");
                fk[i].num = sc.nextLong(10);
            }
            System.out.println("================================
            ↪  ================================");
            System.out.println("Factorial Result with Brute
            ↪  Force");
            for (int i = 0; i < elemen; i++) {
                long faktorialBFStart = System.nanoTime();
                System.out.println("Factorial of value
                ↪  "+fk[i].num+" is :
                ↪  "+fk[i].faktorialBF(fk[i].num));
                long faktorialBFEnd = System.nanoTime();
                System.out.printf("Time in nanoseconds:
                ↪  %,d\n",faktorialBFEnd - faktorialBFStart);
            }

            System.out.println("================================
            ↪  ================================");
            System.out.println("Factorial Result with Divide and
            ↪  Conquer");
            for (int i = 0; i < elemen; i++) {
                long faktorialDCStart = System.nanoTime();
```

```
                System.out.println("Factorial of value
                ↪  "+fk[i].num+" is :
                ↪  "+fk[i].faktorialDC(fk[i].num));
                long faktorialDCEnd = System.nanoTime();
                System.out.printf("Time in nanoseconds:
                ↪  %,d\n",faktorialDCEnd-faktorialDCStart);
            }
            System.out.println("==============================
            ↪  ==============================");

            sc.close();
        }
    }
```

5. Prove by inputting elements that are above 20 digits, is there a difference in execution time?

```
PS D:\Kuliah>  d:; cd 'd:\Kuliah'; & 'C:\Program
↪  Files\Java\jdk-18.0.2.1\bin\java.exe'
↪  '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
↪  'C:\Users\G4CE-PC\AppData\Roaming\Code\User\workspaceStorage\
↪  80d97a47d24665dc0bce7ab1e048ecbd\redhat.java\jdt_ws\
↪  Kuliah_28156aa7\bin' 'Faktorial.MainFaktorial'
================================================================
Input the number of elements you want to count : 1
Input the data value to-1 : 50
================================================================
Factorial Result with Brute Force
Factorial of value 50 is : -3258495067890909184
Time in nanoseconds: 335,000
================================================================
Factorial Result with Divide and Conquer
Factorial of value 50 is : -3258495067890909184
Time in nanoseconds: 172,100
================================================================
```

Yes, there is. With bigger base number, the difference in time rise.

2.4 Calculating Squared Results with Brute Force and Divide and Conquer Algorithms

2.4.1 Practicum

1. In the week 3 package, create a new class named Squared, then create the num attribute that will be raised with the squared number

```
public int num, squared;
```

2. In the Squared class, add the SquaredBF() method

```
public int squaredBF(int a, int n) {
    int result = 1;
    for (int i = 0; i < n; i++) {
        result = result * a;
    }
    return result;
}
```

3. In the Squared class also add a method SquaredDC()

```
public int squaredDC(int a, int n) {
    if (n == 0) {
        return 1;
    } else {
        if (n % 2 == 1) {
            return (squaredDC(a, n/2) * squaredDC(a, n/2) * a);
        } else {
            return (squaredDC(a, n/2) * squaredDC(a, n/2));
        }
    }
}
```

4. Observe whether there are no errors that appear in the making of the Squared class

5. Next create a new class in which there is a main method. This class can be called MainSquared. Add code to the main class to input the number of values to be counted.

```
Scanner sc = new Scanner(System.in);
System.out.println("================================================");
System.out.print("Input the number of elements you want to count
↪  : ");
int elemen = sc.nextInt();
```

6. The values in step 5 are then used for the instantiation of arrays of objects. The following code is added to the process of filling in some values which will be raised to the squared.

```
Squared[] png = new Squared[elemen];

for (int i = 0; i < elemen; i++) {
    png[i] = new Squared();
    System.out.print("Input the value to be squared to-"+(i+1)+"
    ↪  : ");
    png[i].num = sc.nextInt();
    System.out.print("Input the squared value to-"+(i+1)+" : ");
    png[i].squared = sc.nextInt();
}
```

7. Call the results by returning the return value of the method SquaredBF() dan SquaredDC().

```
System.out.println("================================================");
System.out.println("Results with Brute Force Squared");
for (int i = 0; i < elemen; i++) {
    System.out.println("Value "+png[i].num+" squared
    ↪  "+png[i].squared+" is : "+png[i].squaredBF(png[i].num,
    ↪  png[i].squared));
}
System.out.println("================================================");
System.out.println("Results with Divide and Conquer Squared");
for (int i = 0; i < elemen; i++) {
    System.out.println("Value "+png[i].num+" squared
    ↪  "+png[i].squared+" is : "+png[i].squaredDC(png[i].num,
    ↪  png[i].squared));
}
System.out.println("================================================");
```

2.4.2 Verification of Practicum Results

```
PS D:\Kuliah>  & 'C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe'
↪  '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
↪  'C:\Users\G4CE-PC\AppData\Roaming\Code\User\workspaceStorage\
↪  80d97a47d24665dc0bce7ab1e048ecbd\redhat.java\jdt_ws\
↪  Kuliah_28156aa7\bin' 'Faktorial.MainSquared'
================================================
Input the number of elements you want to count : 2
Input the value to be squared to-1 : 6
```

```
Input the squared value to-1 : 2
Input the value to be squared to-2 : 4
Input the squared value to-2 : 3
================================================
Results with Brute Force Squared
Value 6 squared 2 is : 36
Value 4 squared 3 is : 64
================================================
Results with Divide and Conquer Squared
Value 6 squared 2 is : 36
Value 4 squared 3 is : 64
================================================
```

2.4.3 Questions

1. Explain the differences between the 2 methods made are SquaredBF() and SquaredDC()!
   `Answer:`
   BF loops the base number trough the exponent. Meanwhile the DC divide the problem into smaller problem before combining the result.

2. In the SuaredDC() method there is a program as follows:

   ```
   if (n % 2 == 1) {
       return (squaredDC(a, n/2) * squaredDC(a, n/2) * a);
   } else {
       return (squaredDC(a, n/2) * squaredDC(a, n/2));
   }
   ```

   Explain the meaning of the code! `Answer:`
   If the exponent of the problem is odd, it will process the problem by dividing it into 2 function and a multiplication with the base number. If the exponent number is even, it will just divide the problem into 2 function.

3. Explain whether the combine stage is included in the code!
   `Answer:`
   It is included, as the division of the problem contain a process of combining the solution from the sub-problems.

4. Modification of the program code, assuming the attribute filling process is done by a constructor.

5. Add a menu so that only one of the selected methods will be run!

2.5 Calculating Sum Array with Brute Force and Divide and Conquer Algorithms

In this practicum, we will practice how the divide, conquer, and combine process is applied to a case study of the sum of profits of a company in a few months.