JOBSHEET III
BRUTE FORCE and DIVIDE CONQUER

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:

Advantages :

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.

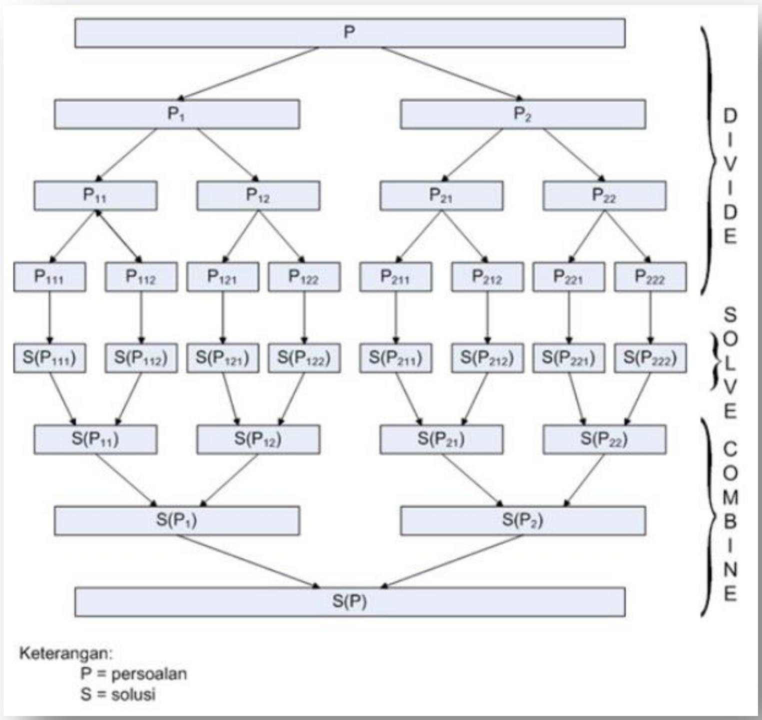
As for sorting, bruteforce is used as the basic principle of the stages of the bubble sort and selection sort methods. The Bubble Sort algorithm is a sorting process that gradually moves to the right position (Bubble). This algorithm will sort the data from the largest to the smallest (ascending) or 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 other words the data in sub sections becomes single. After the data cannot be divided again, the merging 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.

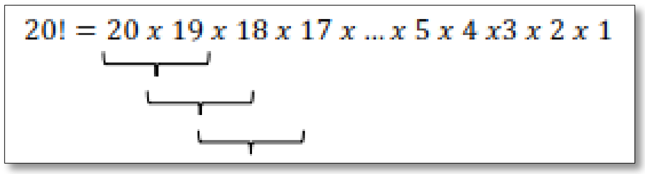
2.3 Calculating Factorial Values with Brute Force and Divide and Conquer Algorithms

Consider the following Class Diagram:

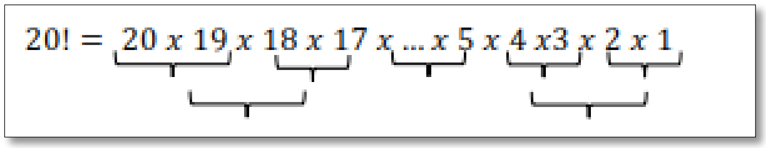
Faktorial
value: int
faktorialBF(): int faktorialDC(): int

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:

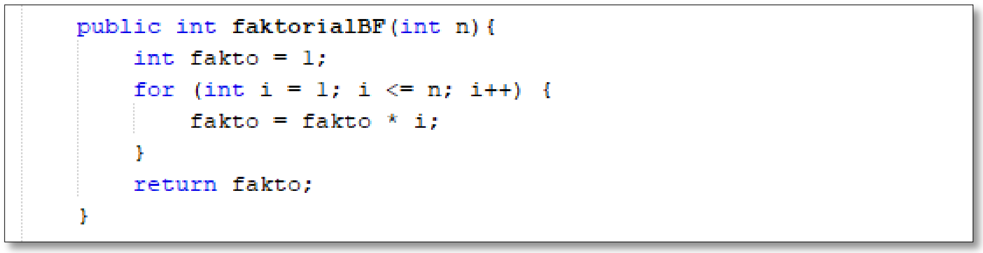


2.3.1 Practicum

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

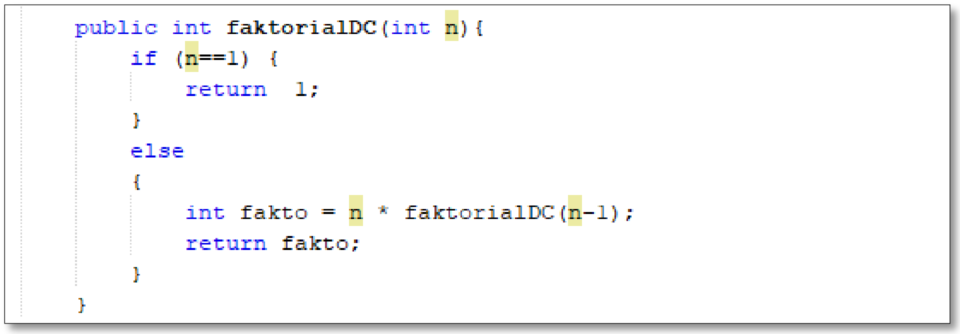
   ```
   public int num;
   ```

   b) Add method `faktorialBF()`



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

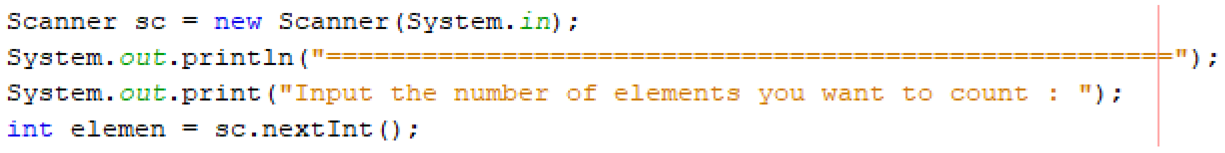
c) Add method `faktorialDC()`



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

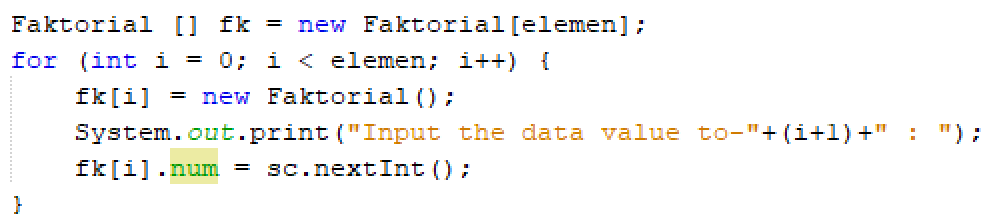
3. Run the `Faktorial class` y creating a new `MainFaktorial` class.

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



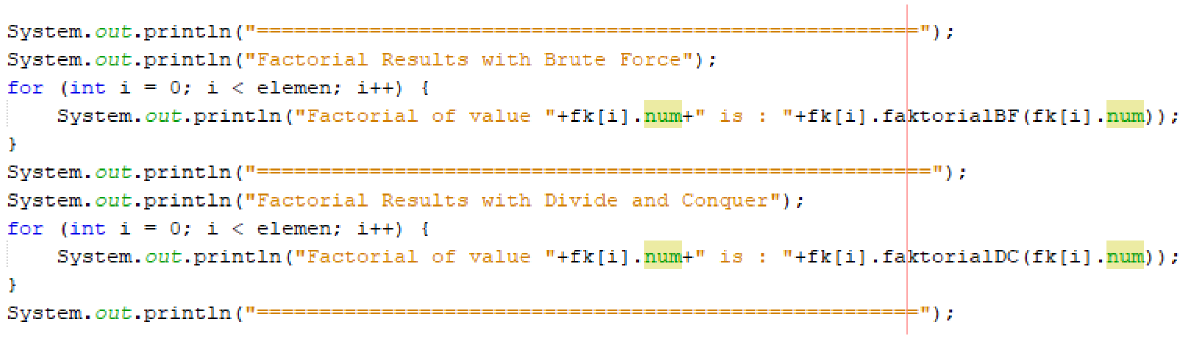
```java
Scanner sc = new Scanner(System.in);
System.out.println("=====================================================");
System.out.print("Input the number of elements you want to count : ");
int elemen = sc.nextInt();
```

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



```java
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.nextInt();
}
```

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

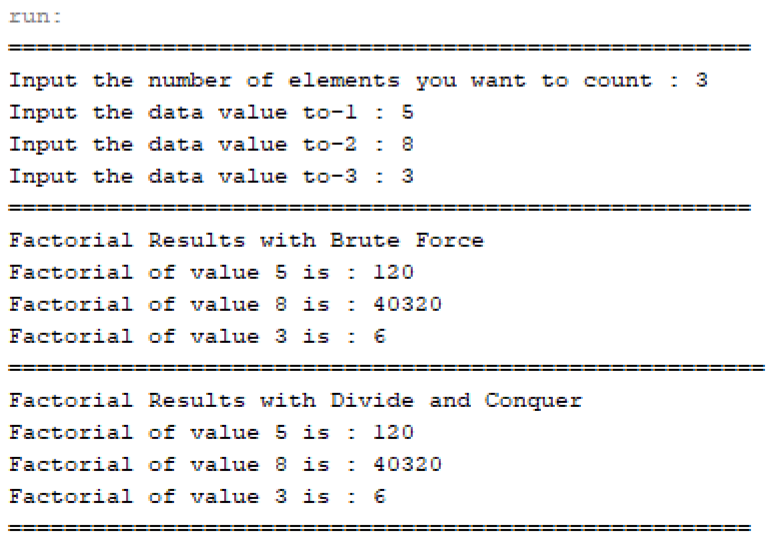


```java
System.out.println("=======================================================");
System.out.println("Factorial Results with Brute Force");
for (int i = 0; i < elemen; i++) {
    System.out.println("Factorial of value "+fk[i].num+" is : "+fk[i].faktorialBF(fk[i].num));
}
System.out.println("========================================================");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

2.3.2 Verification of Practicum Results

Match the compile results of your program code with the following picture.



```
run:
=====================================================
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 is : 120
Factorial of value 8 is : 40320
Factorial of value 3 is : 6
======================================================
Factorial Results with Divide and Conquer
Factorial of value 5 is : 120
Factorial of value 8 is : 40320
Factorial of value 3 is : 6
=====================================================
```

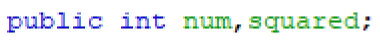
2.3.3 Questions

1. Explain the Divide Conquer Algorithm for calculating factorial values!
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!
3. Is it possible to repeat the factorial BF () method instead of using for? Prove it!
4. Add a check to the execution time of the two types of methods!
5. Prove by inputting elements that are above 20 digits, is there a difference in execution time?

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

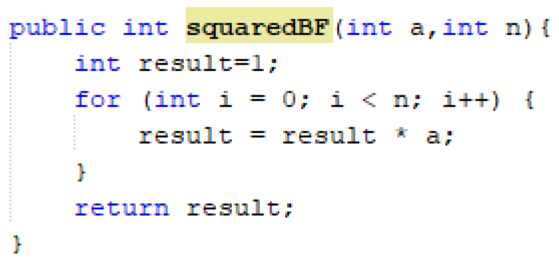
2.4.1 Practicum

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



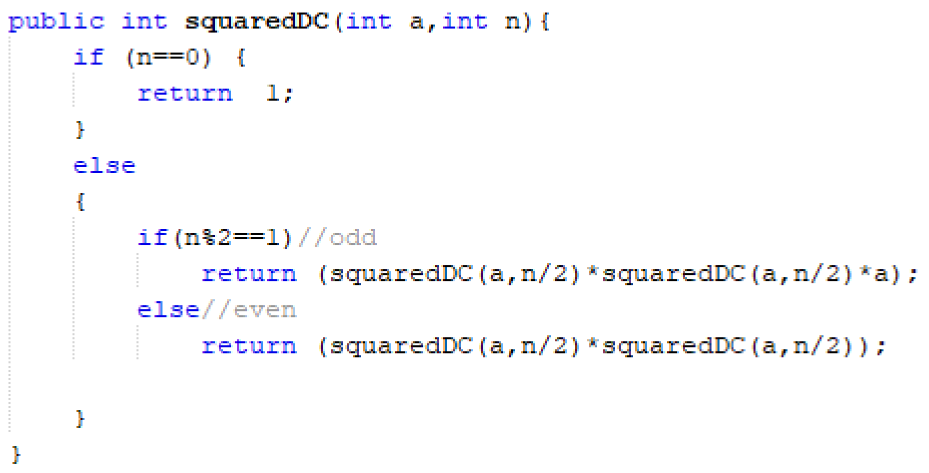
```
public int num,squared;
```

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



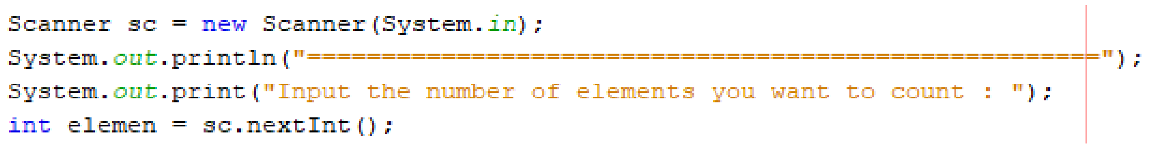
```
public int squaredBF(int a,int n){
    int result=1;
    for (int i = 0; i < n; i++) {
        result = result * a;
    }
    return result;
}
```

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



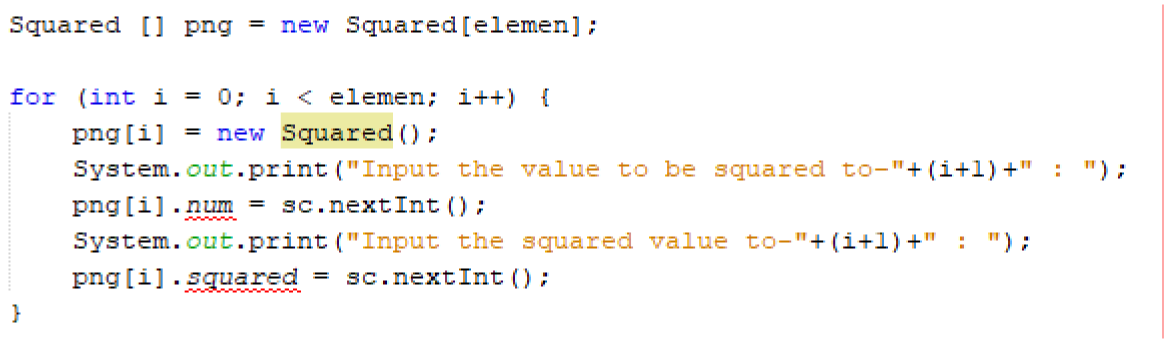
```
public int squaredDC(int a,int n){
    if (n==0) {
        return  1;
    }
    else
    {
        if(n%2==1)//odd
            return (squaredDC(a,n/2)*squaredDC(a,n/2)*a);
        else//even
            return (squaredDC(a,n/2)*squaredDC(a,n/2));

    }
}
```

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.



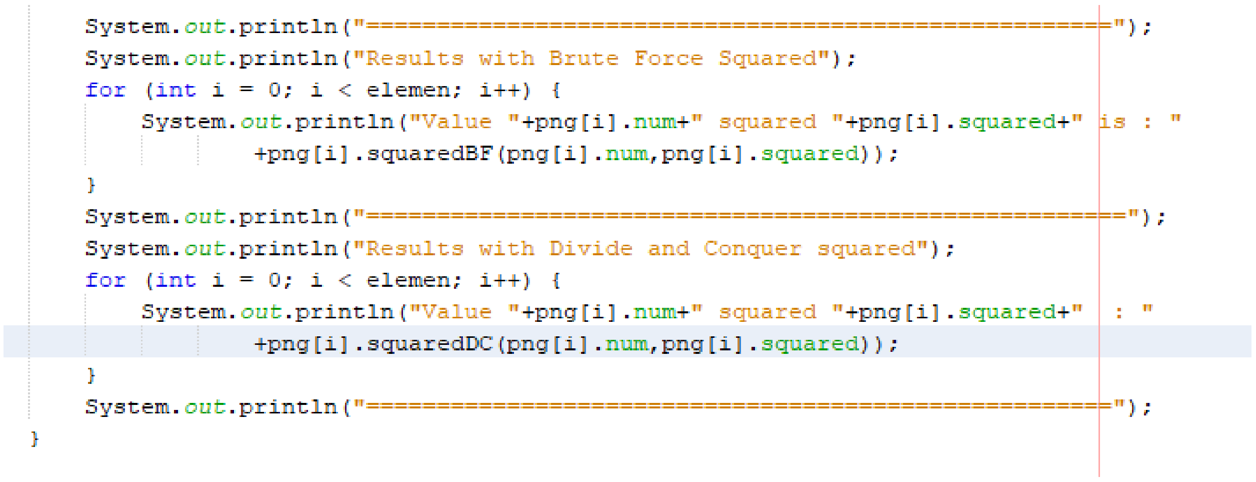
```
Scanner sc = new Scanner(System.in);
System.out.println("=====================================================");
System.out.print("Input the number of elements you want to count : ");
int elemen = sc.nextInt();
```

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();
}
```

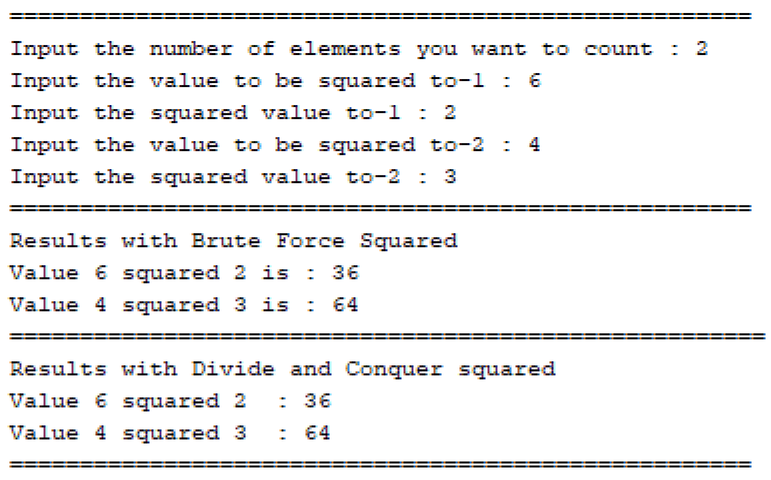
7. Call the results by returning the return value of the method `SquaredBF()` dan `SquaredDC()`.



```java
    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+"  : "
                +png[i].squaredDC(png[i].num,png[i].squared));
    }
    System.out.println("=====================================================");
}
```

2.4.2 Verification of Practicum Results

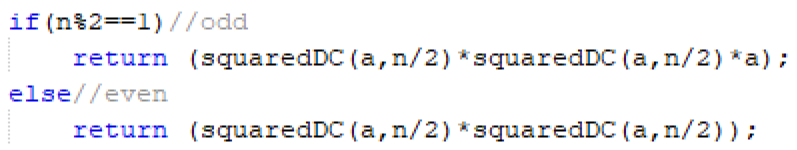
Make sure the output is correct



```
=====================================================
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  : 36
Value 4 squared 3  : 64
=====================================================
```

2.4.3 Questions

1. Explain the differences between the 2 methods made are SquaredBF () and SquaredDC ()!
2. In the SuaredDC () method there is a program as follows:



```java
if(n%2==1)//odd
    return (squaredDC(a,n/2)*squaredDC(a,n/2)*a);
else//even
    return (squaredDC(a,n/2)*squaredDC(a,n/2));
```

   Explain the meaning of the code!

3. Explain whether the combine stage is included in the code!
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.

2.5.1 Practicum

1. On week package3. Create a new class, the Sum class. IN the class Sum there are several attributes of the number of array elements, arrays, and total.

```
public int elemen;
public double profit[];
public double total;
```

2. Add the constructor Sum () with the parameter int elemen

```
public Sum(int element) {
    elemen = element;
    profit = new double[elemen];
    total = 0;
}
```

3. Add the TotalBF () method which will calculate the total value of the array with iterative.

```
double totalBF(double arr[]){
    for (int i = 0; i < elemen; i++) {
        total = total + arr[i];
    }
    return total;
}
```

4. Add the TotalDC () method to implement the calculation of the total array value using the Divide and Conquer algorithm

```
double totalDC(double arr[], int l, int r){
    if(l==r)
        return arr[l];
    else if(l<r){
     int mid=(l+r)/2;
     double lsum=totalDC(arr,l,mid-1);
     double rsum=totalDC(arr,mid+1,r);
     return lsum+rsum+arr[mid];
    }

    return 0;
}
```

5. In this class there is a main method. In this method the user can write how many months the profit will be calculated. In this class an object instance is also created to call the attributes or functions of the Sum class

```
Scanner sc = new Scanner(System.in);
System.out.println("===============================================================");
System.out.println("Program for Calculating Total Profits");
System.out.println("Input the Number of Months : ");
int totElemen= sc.nextInt();
Sum sm = new Sum(totElemen);
sm.elemen = totElemen;
```

6. Make a loop to fill the profit amount according to the number of months entered (fill in the profit value to each array index)

```
System.out.println("===============================================================");
for (int i = 0; i < sm.elemen; i++) {
    System.out.print("Input the profit of the month to - "+(i+1)+" = ");
    sm.profit[i] = sc.nextDouble();
}
```

7. Show the results of calculations through objects that have been made for both algorithm (Brute Force and Divide Conquer) by calling the totalBF () and totalDF () methods of the Sum class.

```
System.out.println("===============================================================");
System.out.println("Algoritma Brute Force");
System.out.println("Total profits of the company for " + sm.elemen + " month is = "
        +sm.totalBF(sm.profit));
System.out.println("===============================================================");
System.out.println("Algoritma Divide Conquer");
System.out.println("Total profits of the company for  " + sm.elemen + " month is = "
        +sm.totalDC(sm.profit, 0,sm.elemen-1));
```

2.5.2 Verification of Practicum Results

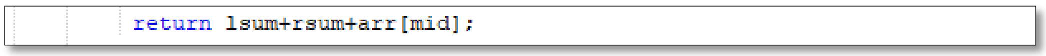
Match the compile results of your program code with the following picture.

```
===============================================================
Program for Calculating Total Profits
Input the Number of Months :
4
===============================================================
Input the profit of the month to - 1 = 1000
Input the profit of the month to - 2 = 2000
Input the profit of the month to - 3 = 3000
Input the profit of the month to - 4 = 1000
===============================================================
Algoritma Brute Force
Total profits of the company for 4 month is = 7000.0
===============================================================
Algoritma Divide Conquer
Total profits of the company for  4 month is = 7000.0
```

2.5.2 Questions

1. Give an illustration of the difference in profit calculation with the TotalBF () or TotalDC () method.

2. Why is there the following return value? Explain!

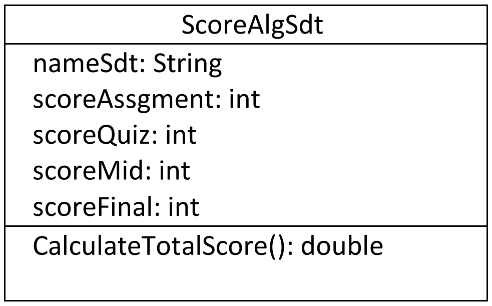


```
return lsum+rsum+arr[mid];
```

3. Why is the **mid variable** required for the **TotalDC () method**?
4. The profit calculation program for a company is only for one company. How do you calculate several months of profit for several companies at once (each company can have a different number of months)? Prove it with the program!

2.6 Assignments

1. Create a program based on the following class diagram!



Score are calculated based on total Assignment of 30%, Quiz 20%, Mid 20%, Final 30%. Adjust the method if it must have parameters.

2. (continued about question no. 1). Create an array of objects in the main class to find out the value of several students in one calculation at a time!
3. (Continued question no. 2) with the modified Brute Force algorithm program in order to know the average value of all students who have been inputted to the Algorithm and data structure course.

Example :

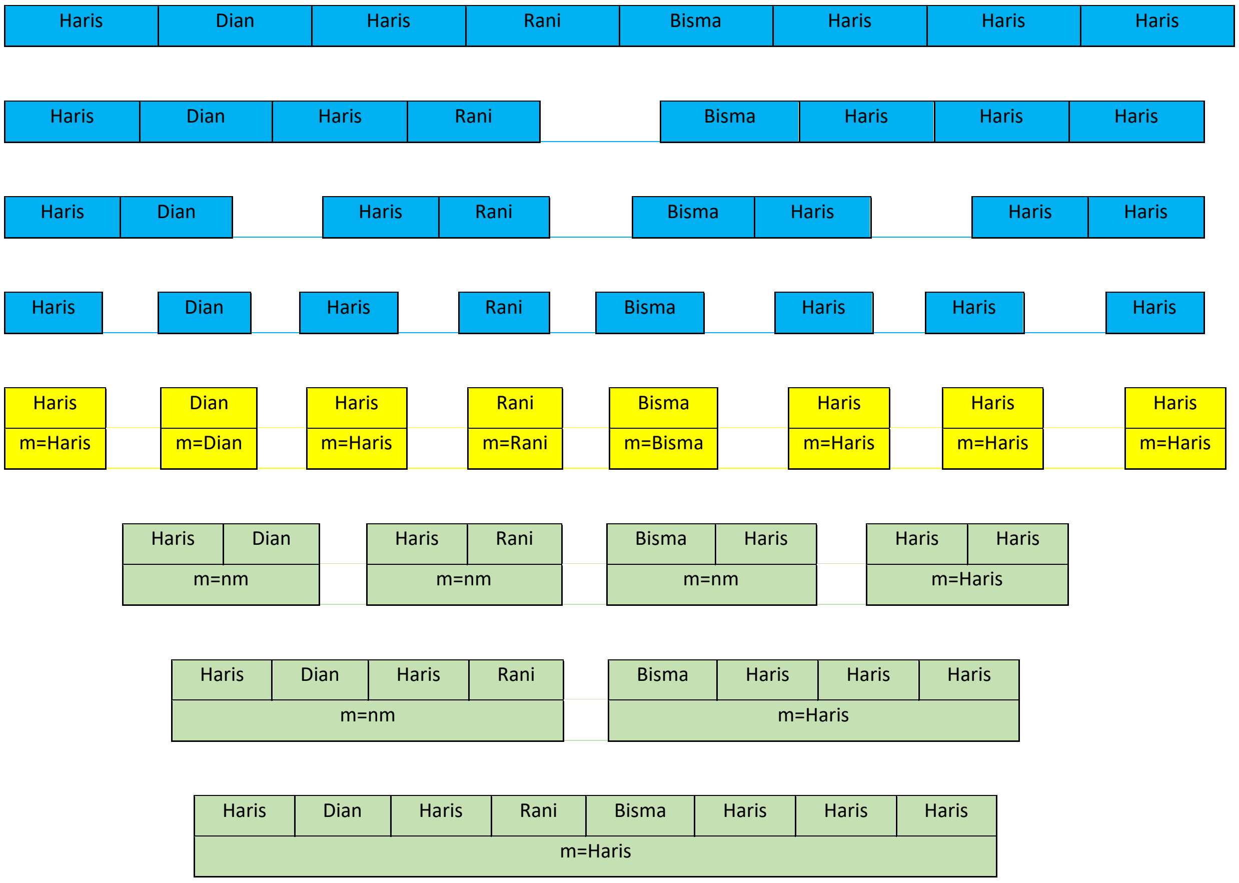
There are a total of the following student grades

Name	**Total Value of Algorithm Courses**
Rani	86,5
Dani	72,5
Saraswati	89
Average	82,67

Then the average total value of all students who have taken the algorithm course is 82.67

4. A university in Malang is holding a vote to elect the BEM chairman in 2020. If the number of votes collected is always even. Then by inputting the selected candidates, count the number of votes for each candidate. Make class diagrams and programs using the Divide and Conquer algorithm from the case study! (The number of array elements and the results of the vote are user input)

   Example: Voting results are as follows (m is majority, nm is not majority)



   Description: Blue is the divide process, yellow starts the conquer process, green starts the combining process

5. What if the number of votes is odd? should there be a program improvement? if yes, improve the program for case study no 4. If the number of votes collected is not always even!
6. Modify the program about the average value of algorithm courses using the Divide and Conquer Algorithm!