JOBSHEET V
BRUTE FORCE AND DIVIDE CONQUER

5.1 Learning Outcome

After completing this practical session, students will be able to develop algorithms using Brute Force and Divide and Conquer approaches. They will also be able to implement these algorithms effectively in problem-solving scenarios.

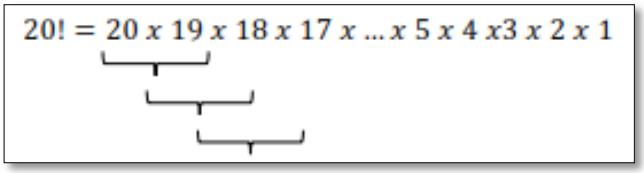
5.2 Calculating Factorial Using Brute Force and Divide and Conquer Algorithms

There is a class diagram as follows:

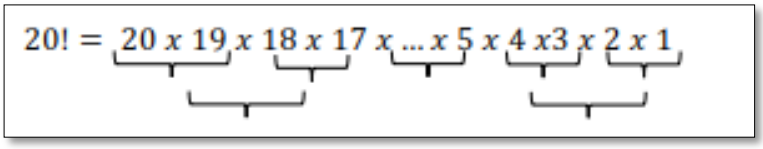
Factorial
factorialBF(): int factorialDC(): int

Based on the class diagram above, a Java program will be created to calculate the factorial of a number using two different algorithms: Brute Force and Divide and Conquer. The calculation process differs between these two approaches, as illustrated below:

- Steps for computing factorial using the **Brute Force** algorithm:



- Steps for computing factorial using the Divide and Conquer method:



5.2.1. Experiment Steps

1. Create a new project named **BruteForceDivideConquer** (or continue the previous project) and a package called **week5**.
2. Create a new class, `Factorial`
3. Complete the `Factorial` class with the attributes and methods as illustrated in the class diagram above.

a) Create `faktorialBF()` method:

```
int factorialBF(int n){
    int facto = 1;
    for(int i=1; i<=n; i++){
        facto = facto * i;
    }
    return facto;
}
```

b) Create `faktorialDC()` method:

```
int factorialDC(int n){
    if(n==1){
        return 1;
    }else{
        int facto = n * factorialDC(n-1);
        return facto;
    }
}
```

c) Run the `Faktorial` class by creating a new class named `MainFaktorial` and executing it.

d) Add `main()` function within `MainFactorial` class, and get the user input:

```
Scanner input = new Scanner(System.in);
System.out.print(s:"Input a number: ");
int num = input.nextInt();
```

e) Then, create an object of `Factorial` and call the `factorialDC()` method and `factorialBF()` method:

```
Factorial fk = new Factorial();
System.out.println("The factorial of "+num+" using BF: "+fk.factorialBF(num));
System.out.println("The factorial of "+num+" using DC: "+fk.factorialDC(num));
```

d) Make sure that no error appears!

5.2.2. Verification of Experiment Results

Compare the output of your compiled program with the following image.

```
Input a number: 5
The factorial of 5 using BF: 120
The factorial of 5 using DC: 120
```

5.2.3. Questions

1. In the base case of the Divide and Conquer algorithm for calculating factorial, explain the differences in the code structure between the `if` and `else` conditions!
2. Is it possible to modify the loop in the `factorialBF()` method to use an alternative to the `for` loop? Please explain and give example if needed!

3. Please explain the differences between `facto = facto*i;` and `int facto = n * factorialDC(n-1);` !
4. Make a conclusion about the differences in how each method works. `factorialBF()` and `factorialDC()`!

5.3 Calculating Exponentiation Using Brute Force and Divide and Conquer Algorithms

In this practical session, we will create a Java program to calculate the exponentiation of a number using two different algorithms: Brute Force and Divide and Conquer. This session will utilize an Array of Objects to manage multiple objects, unlike the previous session, which focused only on a single factorial object.

5.3.1. Experiment Steps

1. In the **week5** package, create a new class named **Power**. Inside this class, define attributes for the base number and its exponent.

```
int baseNumber, exponent;
```

2. Add parameterized constructor

```
Power(int n, int e){
    baseNumber = n;
    exponent = e;
}
```

3. Add `powerBF()` method

```
int powerBF(int n, int e){
    int result = 1;
    for(int i=0; i<e; i++){
        result = result*n;
    }
    return result;
}
```

4. Add `powerDC()` method

```
int powerDC(int n, int e){
    if(e==1){
        return n;
    }else{
        if(e%2==1){
            return (powerDC(n, e/2)*powerDC(n, e/2)*n);
        }else{
            return (powerDC(n, e/2)*powerDC(n, e/2));
        }
    }
}
```

5. Next, create a new class that contains the `main` method. This class can be named `PowerMain`. Add code in the main class to input the number of elements for which the exponentiation will be calculated.

```
Scanner input = new Scanner(System.in);
System.out.print(s:"Input element number: ");
int elemen = input.nextInt();
```

6. The value obtained in step 5 will be used to instantiate an array of objects. In the following code, add a process to fill the array with multiple base numbers along with their respective exponents.

```
Power[] png = new Power[elemen];
for(int i=0;i<elemen;i++){
    System.out.print("Input base number for "+(i+1)+"th element: ");
    int basis = input.nextInt();
    System.out.print("Input exponent for "+(i+1)+"th element: ");
    int exp = input.nextInt();
    png[i] = new Power(basis, exp);
}
```

7. Call `powerBF()` and `powerDC()` method to perform power calculation using both brute force and DC approach!

```
System.out.println(x:"POWER RESULT USING BRUTEFORCE:");
for (Power p : png) {
    System.out.println(p.baseNumber+"^"+p.exponent+": "+p.powerBF(p.baseNumber, p.exponent));
}
System.out.println(x:"POWER RESULT USING DIVIDE AND CONQUER:");
for (Power p : png) {
    System.out.println(p.baseNumber+"^"+p.exponent+": "+p.powerDC(p.baseNumber, p.exponent));
}
```

5.3.2. Verification of Experiment Results

The result must be like this:

```
Masukkan jumlah elemen: 3
Masukan nilai basis elemen ke-1: 2
Masukan nilai pangkat elemen ke-1: 3
Masukan nilai basis elemen ke-2: 4
Masukan nilai pangkat elemen ke-2: 5
Masukan nilai basis elemen ke-3: 6
Masukan nilai pangkat elemen ke-3: 7
HASIL PANGKAT BRUTEFORCE:
2^3: 8
4^5: 1024
6^7: 279936
HASIL PANGKAT DIVIDE AND CONQUER:
2^3: 8
4^5: 1024
6^7: 279936
```

5.3.3. Questions

1. Explain the differences between the two methods created `powerBF()` dan `powerDC()`!
2. Does the **combine** stage exist in the provided code? Show the relevant part!

3. In the `powerBF()` method, parameters are used to pass the base number and its exponent, even though the `Power` class already contains attributes for these values (`baseNumber` and `exponent` atribute). Do you think it is still relevant for the method to have parameters? Could the method be implemented without parameters instead? If so, how would the `powerBF()` method be structured without parameters?
4. Summarize how the `powerBF()` and `powerDC()` methods work!

5.4 Calculating Array Sum Using Brute Force and Divide and Conquer Algorithms

In this experiment, we will practice how the divide, conquer, and combine processes are applied in a case study of calculating a company's profit over several months.

5.4.1. Experiment Steps

1. In the **week5** package, create a new class named **Sum**. Add the parametrized constructor as well .

```
double profits[];

Sum(int el){
    profits = new double[el];
}
```

2. Add `totalBF()` method which will calculate the total value of the array using an iterative approach.

```
double totalBF(){
    double total=0;
    for(int i=0;i<profits.length;i++){
        total = total+profits[i];
    }
    return total;
}
```

3. Add `totalDC()` method that will calculate the total value of the array using Divide and Conquer

```
double totalDC(double arr[], int l, int r){
    if(l==r){
        return arr[l];
    }

    int mid = (l+r)/2;
    double lsum = totalDC(arr, l, mid);
    double rsum = totalDC(arr, mid+1, r);
    return lsum+rsum;
}
```

4. Create a new class named `SumMain`. This class should contain the `main` method, where users can specify the number of months for which the profit will be calculated. Additionally, instantiate an object within this class to access attributes and methods from the `Sum` class.

```
Scanner input = new Scanner(System.in);
System.out.print(s:"Input element number: ");
int element = input.nextInt();
```

5. Create `Sum` object and assign the profit value in each array element

```
Sum sm = new Sum(element);
for(int i=0;i<element;i++){
    System.out.print("Masukkan keuntungan ke-"+(i+1)+": ");
    sm.profits[i] = input.nextDouble();
}
```

6. Call both oof `totalBF()` and `totalDC()` methods!

```
System.out.println("Profit total using BF: "+sm.totalBF());
System.out.println("Profit total using DC: "+sm.totalDC(sm.profits,l:0,element-1));
```

5.4.2. Verification of Experiment Results

Verify the results and it must be matched with the following!

```
Input element number: 5
Input profit #1: 10
Input profit #2: 20
Input profit #3: 30
Input profit #4: 40
Input profit #5: 50
Profit total using BF: 150.0
Profit total using DC: 150.0
```

5.4.3. Questions

1. Why is `mid` variable needed in `totalDC()` method?
2. Explain the following statements in `totalDC()` method?

```
double lsum = totalDC(arr, l, mid);
double rsum = totalDC(arr, mid+1, r);
```

3. Why is it necessary to sum the results of `lsum` and `rsum` as shown below??

```
return lsum+rsum;
```

4. What is the base case of `totalDC()` method?
5. Draw a conclusion about how `totalDC()` works!

5.5. Assignments

A university has a list of student grades with data as shown in the table below.

Name	Student ID (NIM)	Year of Admission	Midterm Score (UTS)	Final Score (UAS)
Ahmad	220101001	2022	78	82
Budi	220101002	2022	85	88
Cindy	220101003	2021	90	87
Dian	220101004	2021	76	79
Eko	220101005	2023	92	95
Fajar	220101006	2020	88	85
Gina	220101007	2023	80	83
Hadi	220101008	2020	82	84

a) Find the highest Midterm Score (UTS) using the Divide and Conquer approach.

b) Find the lowest Midterm Score (UTS) using the Divide and Conquer approach.

c) Calculate the average Final Score (UAS) of all students using the Brute Force approach.