

**LAPORAN HASIL PRAKTIKUM  
ALGORITMA DAN STRUKTUR DATA  
JOBSHEET 5**



Disusun Oleh :

Nama : Nawaf Azril Annaufal

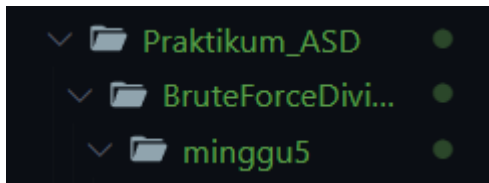
Nim : 244107020047

Kelas : TI 1E

**PROGRAM STUDI TEKNIK INFORMATIKA  
JURUSAN TEKNOLOGI INFORMASI  
POLINEMA  
2025**

## Percobaan 1

### 1. Membuat Project baru



### 2. Hasil penulisan kode program

#### Class Faktorial21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;
public class faktorial21 {

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

    int faktorialDC(int n) {
        if (n == 0) {
            return 1;
        } else {
            int fakto = n * faktorialDC(n - 1);
            return fakto;
        }
    }
}
```

### Class MainFakorial21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;
import java.util.Scanner;

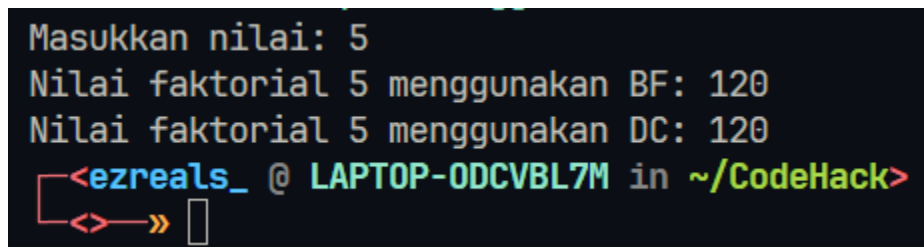
public class MainFakorial21 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan nilai: ");
        int nilai = sc.nextInt();

        faktorial21 fk = new faktorial21();
        System.out.println("Nilai faktorial " + nilai + " menggunakan BF: " +
fk.faktorialBF(nilai));
        System.out.println("Nilai faktorial " + nilai + " menggunakan DC: " +
fk.faktorialDC(nilai));

        sc.close();
    }
}
```

### 3. Hasil run kode program



```
Masukkan nilai: 5
Nilai faktorial 5 menggunakan BF: 120
Nilai faktorial 5 menggunakan DC: 120
<ezreals_ @ LAPTOP-ODCVBL7M in ~/CodeHack>
```

### Pertanyaan:

1. Pada base line Algoritma Divide Conquer untuk melakukan pencarian nilai faktorial, jelaskan perbedaan bagian kode pada penggunaan if dan else!
2. Apakah memungkinkan perulangan pada method faktorialBF() diubah selain menggunakan for? Buktikan!
3. Jelaskan perbedaan antara fakto \*= i; dan int fakto = n \* faktorialDC(n-1); !
4. Buat Kesimpulan tentang perbedaan cara kerja method faktorialBF() dan faktorialDC()!

Jawab:

1. Pada bagian if berfungsi sebagai kondisi berhenti agar rekursif tidak berjalan tanpa batas. Untuk bagian else digunakan untuk menangani perhitungan faktorial.
2. Ya, perulangan pada method faktorialBF() bisa diubah menggunakan while atau do-while loop.

Pembuktian menggunakan while dan do-while

```
i = 1, fakto = 1 * 1 = 1
i = 2, fakto = 1 * 2 = 2
i = 3, fakto = 2 * 3 = 6
i = 4, fakto = 6 * 4 = 24
i = 5, fakto = 24 * 5 = 120
Perulangan berhenti karena i > 5
```

3. fakto \*= i; adalah teknik iteratif, menggunakan loop untuk menghitung faktorial, sedangkan  
int fakto = n \* faktorialDC(n - 1); adalah teknik rekursif, menggunakan pemanggilan fungsi berulang hingga mencapai base case.

4. Metode Iteratif pada faktorialBF()

Metode iteratif menggunakan perulangan untuk menghitung faktorial secara langsung. Dalam metode ini, program akan menginisialisasi nilai awal sebagai 1, lalu melakukan perkalian berulang dengan angka-angka dari 1 hingga n menggunakan perulangan for.

Metode Rekursif pada faktorialDC()

Metode rekursif bekerja dengan cara memanggil dirinya sendiri hingga mencapai kondisi dasar atau base case, yaitu ketika nilai n mencapai 0. Dalam pendekatan ini, masalah utama (n!) dipecah menjadi sub-masalah yang lebih kecil hingga mencapai kondisi yang paling sederhana, yaitu  $0! = 1$ .

## Percobaan 2

### 1. Hasil penulisan kode program

#### Class pangkat21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;
public class pangkat21 {

    int nilai, pangkat;

    pangkat21(int nilai, int pangkat) {
        this.nilai = nilai;
        this.pangkat = pangkat;
    }

    int pangkatBF(int a, int n) {
        int hasil = 1;
        for(int i = 0; i < n; i++) {
            hasil = hasil * a;
        }
        return hasil;
    }

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

## Class MainPangkat21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;

import java.util.Scanner;

public class MainPangkat21 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan jumlah elemen: ");
        int elemen = sc.nextInt();

        pangkat21[] png = new pangkat21[elemen];
        for (int i = 0; i < elemen; i++) {
            System.out.print("Masukkan nilai base elemen ke-" + (i+1) + ": ");
            int basis = sc.nextInt();
            System.out.print("Masukkan nilai pangkat elemen ke-" + (i+1) + ": ");
            int pangkat = sc.nextInt();
            png[i] = new pangkat21(basis, pangkat);
        }

        System.out.println("HASIL PANGKAT DENGAN BRUTE FORCE");
        for (pangkat21 p : png) {
            System.out.println(p.nilai + "^" + p.pangkat + ": " +
p.pangkatBF(p.nilai, p.pangkat));
        }

        System.out.println("HASIL PANGKAT DENGAN DIVIDE CONQUER");
        for (pangkat21 p : png) {
            System.out.println(p.nilai + "^" + p.pangkat + ": " +
p.pangkatDC(p.nilai, p.pangkat));
        }

        sc.close();

    }
}
```

Pertanyaan:

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu pangkatBF() dan pangkatDC()!
2. Apakah tahap combine sudah termasuk dalam kode tersebut? Tunjukkan!
3. Pada method pangkatBF() terdapat parameter untuk melewati nilai yang akan dipangkatkan dan pangkat berapa, padahal di sisi lain di class Pangkat telah ada atribut nilai dan pangkat, apakah menurut Anda method tersebut tetap relevan untuk memiliki parameter? Apakah bisa jika method tersebut dibuat dengan tanpa parameter? Jika bisa, seperti apa method pangkatBF() yang tanpa parameter?
4. Tarik tentang cara kerja method pangkatBF() dan pangkatDC()!

Jawab:

1. pangkatBF() (Metode Brute Force – Iteratif) Metode ini menggunakan perulangan for untuk mengalikan nilai dasar (a) sebanyak n kali, sedangkan pangkatDC() (Metode Divide and Conquer - Rekursif) Metode ini menggunakan konsep Divide and Conquer, yaitu membagi masalah menjadi bagian lebih kecil, menyelesaikannya secara rekursif, dan menggabungkan hasilnya.
2. Combine terjadi di bagian return dalam kondisi else, di mana hasil dari dua pemanggilan pangkatDC() dikalikan bersama.

```
return (pangkatDC(a, n/2) * pangkatDC(a, n/2));  
return (pangkatDC(a, n/2) * pangkatDC(a, n/2) * a);
```

3. Pada class pangkat21, sudah terdapat atribut nilai dan pangkat, sehingga dalam metode pangkatBF() sebenarnya bisa menggunakan atribut tersebut secara langsung tanpa harus menerima parameter tambahan.  
Ya, bisa. Dengan mengubah pangkatBF() agar menggunakan atribut nilai dan pangkat langsung dari objek tanpa parameter.

pangkatBF tanpa parameter

```
int pangkatBF() {  
    int hasil = 1;  
    for (int i = 0; i < this.pangkat; i++) {  
        hasil *= this.nilai;  
    }  
    return hasil;  
}
```

#### 4. Cara Kerja BF

- Menginisialisasi variabel hasil dengan nilai 1.
- Menggunakan perulangan for sebanyak  $n$  kali untuk mengalikan hasil dengan  $a$ .
- Setelah perulangan selesai, hasil mengandung hasil akhir perpangkatan  $a^n$  dan dikembalikan sebagai output.

#### Cara Kerja DC

- Base case: Jika  $n == 1$ , langsung kembalikan  $a$  sebagai hasil.
- Divide: Jika  $n$  lebih dari 1, pecah masalah menjadi lebih kecil dengan menghitung  $\text{pangkatDC}(a, n/2)$ .
- Combine:
  - Jika  $n$  genap, maka hasil akhir adalah hasil rekursi dikalikan dengan dirinya sendiri.
  - Jika  $n$  ganjil, maka hasil akhir adalah hasil rekursi dikalikan dengan dirinya sendiri dan dikali  $a$  sekali lagi untuk menyesuaikan hasil.



## Percobaan 3

### 1. Hasil penulisan kode program

#### Class sum21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;

public class sum21 {

    double keuntungan[];
    sum21(int e1) {
        keuntungan = new double[e1];
    }

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

    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;
    }
}
```

## Class MainSum21

```
package Praktikum_ASD.BruteForceDivideConquer.minggu5;
import java.util.Scanner;
public class mainSum21 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan jumlah elemen: ");
        int elemen = sc.nextInt();

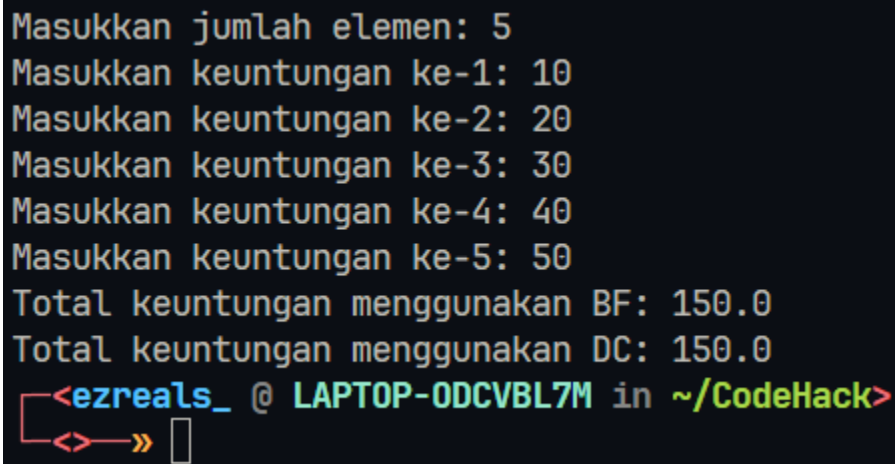
        sum21 sm = new sum21(elemen);

        for (int i = 0; i < elemen; i++) {
            System.out.print("Masukkan keuntungan ke-" + (i+1) + ": ");
            sm.keuntungan[i] = sc.nextDouble();
        }

        System.out.println("Total keuntungan menggunakan BF: " + sm.totalBF());
        System.out.println("Total keuntungan menggunakan DC: " +
sm.totalDC(sm.keuntungan, 0, elemen-1));

        sc.close();
    }
}
```

## 2. Hasil run kode program



```
Masukkan jumlah elemen: 5
Masukkan keuntungan ke-1: 10
Masukkan keuntungan ke-2: 20
Masukkan keuntungan ke-3: 30
Masukkan keuntungan ke-4: 40
Masukkan keuntungan ke-5: 50
Total keuntungan menggunakan BF: 150.0
Total keuntungan menggunakan DC: 150.0
<ezreals_ @ LAPTOP-ODCVBL7M in ~/CodeHack>
```

Pertanyaan:

1. Kenapa dibutuhkan variable mid pada method TotalDC()?
2. Untuk apakah statement di bawah ini dilakukan dalam TotalDC()?
3. Kenapa diperlukan penjumlahan hasil lsum dan rsum seperti di bawah ini?
4. Apakah base case dari totalDC()?
5. Tarik Kesimpulan tentang cara kerja totalDC()

Jawab:

1. Pada metode totalDC(), variabel mid digunakan untuk membagi array menjadi dua bagian dalam proses Divide and Conquer.
2. Statement tersebut digunakan untuk memecah array menjadi dua bagian, lalu menghitung total secara rekursif. Setelah kedua bagian dihitung, hasilnya dijumlahkan dalam tahap Combine, sehingga memperoleh hasil total keuntungan yang benar.
3. Statement `return lsum + rsum;` diperlukan agar hasil dari dua submasalah bisa dikombinasikan menjadi satu hasil akhir. Tanpa penjumlahan ini, fungsi hanya akan menghitung separuh data dan tidak memberikan total keseluruhan.
4. Base case totalDC  

```
if (l == r) {  
    return arr[l];  
}
```
5. Metode totalDC() bekerja dengan membagi array menjadi bagian-bagian kecil, menyelesaikan setiap bagian secara rekursif, lalu menggabungkan hasilnya untuk mendapatkan total keseluruhan.