

**JOB SHEET 5**  
**PRAKTIKUM ALGORITMA**  
**STRUKTUR DATA**



**Burhnauddin ihsan**

**244107020189**

**TI 1E/06**

**PROGRAM STUDI D-IV TEKNIK INFORMATIKA**  
**JURUSAN TEKNOLOGI INFORMASI**  
**POLITEKNIK NEGERI MALANG**

**2025**

## PERCOBAAN 1

### 1. Kode program

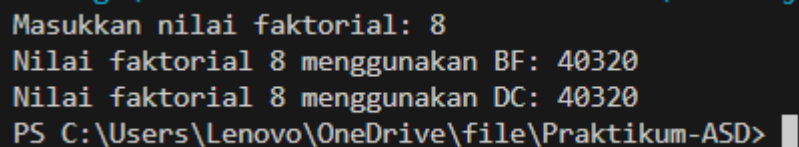
#### CLASS FAKTORIAL

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

#### MAIN FAKTORIAL

```
import java.util.Scanner;  
  
public class mainFaktorial {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Masukkan nilai faktorial: ");  
        int nilai = scanner.nextInt();  
  
        Faktorial fk = new Faktorial();  
  
        System.out.println("Nilai faktorial " + nilai + " menggunakan BF: " +  
            fk.faktorialBF(nilai));  
        System.out.println("Nilai faktorial " + nilai + " menggunakan DC: " +  
            fk.faktorialDC(nilai));  
    }  
}
```

### 2. Hasil dari kode program



```
Masukkan nilai faktorial: 8  
Nilai faktorial 8 menggunakan BF: 40320  
Nilai faktorial 8 menggunakan DC: 40320  
PS C:\Users\Lenovo\OneDrive\file\Praktikum-ASD>
```

## PERTANYAAN

1. Dalam algoritma divide and conquer untuk menghitung faktorial, terdapat perbedaan antara bagian if dan else, yang berfungsi sebagai base case dan rekursif case.
  - if (Base Case): Menentukan kapan rekursi berhenti ( $n = 0$  atau  $n = 1$ ).
  - else (Recursive Case): Memecah masalah dan memanggil dirinya sendiri hingga mencapai base case.
2. Ya, perulangan dalam metode faktorialBF() dapat diubah menggunakan while loop atau do-while loop sebagai alternatif dari for loop. Berikut adalah pembuktiannya dengan implementasi kedua metode tersebut yaitu
  - Menggunakan while loop

```
public class Faktorial {  
    public int faktorialBFWhile(int n) {  
        int hasil = 1;  
        int i = 1;  
        while (i <= n) {  
            hasil *= i;  
            i++;  
        }  
        return hasil;  
    }  
}
```

- Menggunakan do while loop

```
Public class Faktorial{  
    public int faktorialBFDoWhile(int n) {  
        int hasil = 1;  
        int i = 1;  
        do {  
            hasil *= i;  
            i++;  
        } while (i <= n);  
        return hasil;  
    }  
}
```

3. Perbedaan utama antara fakto \*= i; dan int fakto = n \* faktorialDC(n-1); terletak pada pendekatan iteratif dan rekursif dalam menghitung faktorial. Jika ingin kode lebih efisien, maka harus menggunakan perulangan dan Jika ingin kode lebih elegan dan sesuai konsep Divide & Conquer, gunakan rekursi.
4. Metode faktorialBF() dan faktorialDC() bekerja dengan cara yang berbeda untuk menghitung faktorial. faktorialBF() menggunakan perulangan, jadi prosesnya dilakukan langkah demi langkah hingga mendapatkan hasil akhir. Cara ini lebih cepat dan tidak terlalu banyak menggunakan memori, sehingga cocok untuk angka besar. Sementara itu, faktorialDC() menggunakan rekursi, yaitu memanggil dirinya sendiri sampai mencapai titik akhir (base case). Meskipun terlihat lebih elegan dan sesuai dengan konsep Divide & Conquer, cara ini bisa lebih lambat dan boros memori, terutama

untuk angka besar, karena menyimpan banyak proses di dalam memori. Jadi, kalau butuh kecepatan dan efisiensi, pakai metode iteratif (faktorialBF()), tapi kalau ingin memahami konsep rekursi dengan lebih baik, metode rekursif (faktorialDC()) bisa jadi pilihan.

## PERCOBAAN 2

### 1. Kode program

#### CLASS PANGKAT

```
public class Pangkat6 {
    int nilai, pangkat;

    public Pangkat6(int n, int p) {
        nilai = n ;
        pangkat = p;
    }

    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));
            }
        }
    }
}
```

## MAIN PANGKAT

```
import java.util.Scanner;
public class mainPangkat6 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Masukkan Jumlah Elemen : ");
        int elemen = input.nextInt();

        Pangkat6[] png = new Pangkat6[elemen];
        for (int i = 0; i < elemen; i++) {
            System.out.print("Masukkan Nilai Basis Elemen Ke- " + (i+1) + ": ");
            int basis = input.nextInt();
            System.out.print("Masukkan Pangkat Basis Elemen Ke- " + (i+1) + ": ");
            int pangkat = input.nextInt();
            png[i] = new Pangkat6(basis, pangkat);
        }

        System.out.println("HASIL PANGKAT BRUTEFORCE : ");
        for (Pangkat6 p : png) {
            System.out.println(p.nilai+"^"+p.pangkat+": "+p.pangkatBF(p.nilai,
p.pangkat));
        }
        System.out.println("HASIL PANGKAT DIVIDE AND CONQUER");
        for (Pangkat6 p : png) {
            System.out.println(p.nilai+"^"+p.pangkat+": "+p.pangkatDC(p.nilai,
p.pangkat));
        }
    }
}
```

## 2. Hasil dari kode program

```
Masukkan Jumlah Elemen : 3
Masukkan Nilai Basis Elemen Ke- 1: 2
Masukkan Pangkat Basis Elemen Ke- 1: 3
Masukkan Nilai Basis Elemen Ke- 2: 4
Masukkan Pangkat Basis Elemen Ke- 2: 5
Masukkan Nilai Basis Elemen Ke- 3: 6
Masukkan Pangkat Basis 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
PS C:\Users\Lenovo\OneDrive\file\Praktikum-ASD> █
```

## PERTANYAAN

1. pangkatBF menggunakan metode bruteforce dengan cara mengalikan bilangan secara berulang sebanyak eksponen yang diberikan, sehingga lebih lambat dengan kompleksitas  $O(n)$ . sementara itu, pangkatDC() menggunakan metode divide conquer, yang membagi perhitungan menjadi lebih kecil, sehingga lebih cepat dengan kompleksitas  $O(\log n)$ . Singkatnya, pangkatDC() lebih efisien dibandingkan pangkatBF(), terutama eksponen yang besar.
2. Tahap combine sudah terdapat dalam metode pangkatDC(), Combine merupakan Menggabungkan hasil dari submasalah untuk mendapatkan solusi akhir/ penggabungan hasil dari submasalah. bagian yang terdapat combine :

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

3. - Pada method pangkatBF(), terdapat parameter a (basis) dan n (pangkat), padahal atribut nilai dan pangkat sudah ada di dalam class Pangkat07. menggunakan parameter dalam method ini sebenarnya tidak terlalu diperlukan, karena nilai basis dan pangkat sudah dapat diakses langsung dari atribut kelas.  
  
- pangkatBF() bisa dibuat tanpa parameter dengan langsung menggunakan atribut instance nilai dan pangkat dari class Pangkat6.

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

4. Method pangkatBF() (Brute Force)

- Metode ini menggunakan pendekatan iteratif untuk menghitung perpangkatan.

```

public class Sum6 {
    double keuntungan[];

    Sum6(int el){
        keuntungan = new double[el];
    }
    double totalBF(){
        double total = 0;
        for (int i = 0; i < keuntungan.length; i++) {
            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;
    }
}

```

- Dimulai dengan inisialisasi variabel hasil = 1.
- Kemudian, melakukan perkalian berulang sebanyak nilai pangkat yang diberikan.
- Kompleksitas waktu metode ini adalah  $O(n)$ , karena jumlah operasi sebanding dengan pangkat.

#### Method pangkatDC() (Divide and Conquer)

- Metode ini menggunakan strategi divide and conquer, membagi masalah menjadi lebih kecil dan menyelesaikannya secara rekursif.
- Jika pangkat genap, misalnya  $x^n = (x^{(n/2)}) \times (x^{(n/2)})$ .
- Jika pangkat ganjil, misalnya  $x^n = x \times (x^{((n-1)/2)}) \times (x^{((n-1)/2)})$ .
- Dengan membagi pangkat menjadi lebih kecil, metode ini mengurangi jumlah perkalian yang diperlukan.
- Kompleksitas waktu metode ini adalah  $O(\log n)$ , karena setiap langkah membagi pangkat menjadi setengahnya.

## PERCOBAAN 3

### 1. Kode program

#### CLASS SUM

```
public class Sum6 {
    double keuntungan[];

    Sum6(int el){
        keuntungan = new double[el];
    }
    double totalBF(){
        double total = 0;
        for (int i = 0; i < keuntungan.length; i++) {
            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;
    }
}
```

#### MAIN SUM

```
import java.util.Scanner;
public class mainSum6 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Masukkan Jumlah Elemen : ");
        int elemen = input.nextInt();

        Sum6 sm = new Sum6(elemen);
        for (int i = 0; i < elemen; i++) {
            System.out.print("Masukkan Keuntungan Ke-"+(i+1)+" : ");
            sm.keuntungan[i] = input.nextDouble();
        }

        System.out.println("Total Keuntungan Menggunakan BruteForce : "+sm.totalBF());
        System.out.println("Total keuntungan Menggunakan Divide And Conquer : "+sm.totalDC(sm.keuntungan, 0, elemen-1));
    }
}
```



2. hasil dari 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 BruteForce : 150.0
Total keuntungan Menggunakan Divide And Conquer : 150.0
PS C:\Users\Lenovo\OneDrive\file\Praktikum-ASD> █
```

## PERTANYAAN

1. Karena variable mid dibutuhkan untuk membagi array menjadi dua bagian dalam metode divide and conquer, sehingga memungkinkan rekursi berjalan efisien dan mempercepat proses perhitungan dibandingkan metode brute force
2. Statement ini digunakan untuk menghitung total keuntungan dari dua bagian array.
  - lsum menghitung keuntungan di bagian kiri.
  - rsum menghitung keuntungan di bagian kanan.
  - Hasilnya dijumlahkan untuk mendapatkan total keseluruhan.
3. Penjumlahan lsum + rsum diperlukan karena metode Divide and Conquer membagi array menjadi dua bagian, menghitung total keuntungan masing-masing, lalu menggabungkan hasilnya. Tanpa penjumlahan ini, kita hanya mendapatkan sebagian hasil, bukan total keseluruhan keuntungan. Jadi, lsum + rsum memastikan semua elemen dalam array diperhitungkan
4. Base case dari TotalDC() adalah kondisi di mana rekursi berhenti, yaitu ketika hanya ada satu elemen yang perlu dijumlahkan. Base case ini penting agar metode tidak mengalami infinite recursion dan dapat menyelesaikan perhitungannya dengan benar.
5. totalDC() bekerja dengan cara membagi array menjadi dua bagian, menghitung total keuntungan masing masing secara rekursif, lalu menggabungkan hasilnya. Jika hanya satu elemen, langsung dikembalikan. pendekatan ini lebih cepat karena menyelesaikan masalah sedikit demi sedikit.