JOBSHEET 5

LAPORAN HASIL PRAKTIKUM

ALGORITMA DAN STRUKTUR

DATA

MUHAMMAD DAFFI FIROS ZAIDAN

244107020182

TI 1E

PROGRAM STUDI D-IV TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2025

PERCOBAAN 1

Kode Program

Class

```
public class Faktorial17 {
    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 {
            return n * faktorialDC(n - 1);
        }
    }
}
```

Main

```
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 Faktorial17();

        System.out.println("Nilai faktorial " + nilai + " menggunakan BF: "
                + fk.faktorialBF(nilai));
        System.out.println("Nilai faktorial " + nilai + " menggunakan DC: "
                + fk.faktorialDC(nilai));
    }
}
```

Hasil Kode Program

```
Masukkan nilai faktorial: 6
Nilai faktorial 6 menggunakan BF: 720
Nilai faktorial 6 menggunakan DC: 720
PS D:\Tugas Kuliah\SEMESTER 2\ASLD> ^C
```

PERTANYAAN

1. Dalam algoritma Divide and Conquer untuk menghitung faktorial, terdapat dua bagian utama:

- **Base Case (if statement):** Berfungsi sebagai kondisi penghentian rekursi, yaitu saat n = 0 atau n = 1, nilai faktorial langsung dikembalikan tanpa melakukan pemanggilan rekursif lebih lanjut.
- **Recursive Case (else statement):** Bagian ini akan memecah permasalahan dengan memanggil kembali fungsi yang sama secara rekursif hingga mencapai base case.

2. Perulangan dalam metode faktorialBF() yang menggunakan for loop dapat diubah menjadi while loop atau do-while loop sebagai alternatif. Berikut adalah implementasi kedua metode tersebut sebagai bukti:

- Menggunakan while loop: Perulangan akan terus berjalan selama kondisi i <= n terpenuhi, dengan nilai faktorial dihitung dalam setiap iterasi.

- Menggunakan do-while loop: Perulangan tetap dijalankan setidaknya satu kali sebelum kondisi diperiksa, memastikan bahwa proses perkalian faktor tetap berlangsung meskipun n = 1.

- 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 antara Pendekatan Iteratif dan Rekursif dalam Menghitung Faktorial

   Perbedaan utama antara fakto *= i; dan int fakto = n * faktorialDC(n-1); terletak pada pendekatan yang digunakan untuk menghitung faktorial. Pendekatan pertama menggunakan iterasi, yang lebih efisien dalam hal kecepatan dan penggunaan memori, sedangkan pendekatan kedua menggunakan rekursi, yang lebih elegan dan sesuai dengan prinsip Divide & Conquer. Jika tujuan utama adalah efisiensi, maka metode iteratif lebih disarankan. Namun, jika ingin memahami konsep rekursi dengan lebih mendalam, maka pendekatan rekursif adalah pilihan yang tepat.

4. Metode faktorialBF() dan faktorialDC()

Metode faktorialBF() dan faktorialDC() memiliki cara kerja yang berbeda dalam menghitung faktorial. faktorialBF() menggunakan pendekatan iteratif, di mana proses perhitungan dilakukan secara bertahap hingga mencapai hasil akhir. Metode ini lebih cepat dan lebih hemat memori, sehingga lebih cocok untuk menghitung faktorial dari angka besar. Di sisi lain, faktorialDC() menggunakan pendekatan rekursif, yang melibatkan pemanggilan fungsi itu sendiri hingga mencapai kondisi dasar (base case). Meskipun metode ini lebih elegan dan sesuai dengan konsep Divide & Conquer, ia cenderung lebih lambat dan menggunakan lebih banyak memori, terutama untuk angka besar, karena menyimpan banyak informasi dalam tumpukan memori. Oleh karena itu, jika kecepatan dan efisiensi adalah prioritas, metode iteratif (faktorialBF()) lebih disarankan. Namun, jika tujuan Anda adalah untuk memahami konsep rekursi dengan lebih baik, maka metode rekursif (faktorialDC()) bisa menjadi pilihan yang baik.

PERCOBAAN 2

Class

```
public class Pangkat17 {
    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 *= a;
        }
        return hasil;
    }

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

Main

```java
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();

        Pangkat17[] png = new Pangkat17[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 Pangkat17(basis, pangkat);
        }

        System.out.println("\nHASIL PANGKAT BRUTE FORCE:");
        for (Pangkat17 p : png) {
            System.out.println(p.nilai + "^" + p.pangkat + " = " + p.pangkatBF(p.nilai,
p.pangkat));
        }

        System.out.println("\nHASIL PANGKAT DIVIDE AND CONQUER:");
        for (Pangkat17 p : png) {
            System.out.println(p.nilai + "^" + p.pangkat + " = " + p.pangkatDC(p.nilai,
p.pangkat));
        }

    }
}
```

Hasil Kode Program

```
Masukkan Jumlah Elemen: 4
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
Masukkan Nilai Basis Elemen Ke-4: 8
Masukkan Pangkat Basis Elemen Ke-4: 9

HASIL PANGKAT BRUTE FORCE:
2^3 = 8
4^5 = 1024
6^7 = 279936
8^9 = 134217728

HASIL PANGKAT DIVIDE AND CONQUER:
2^3 = 8
4^5 = 1024
6^7 = 279936
8^9 = 134217728
PS D:\Tugas Kuliah\SEMESTER 2\ASLD>
```

Pertanyaan

1. Perbandingan Metode pangkatBF() dan pangkatDC()

   pangkatBF() menerapkan metode brute force, di mana bilangan dikalikan berulang kali sebanyak nilai eksponen. Hal ini menyebabkan kompleksitas O(n), yang berarti waktu eksekusi meningkat seiring bertambahnya eksponen.
   Sebaliknya, pangkatDC() menggunakan divide and conquer, di mana perhitungan dibagi menjadi bagian yang lebih kecil. Dengan pendekatan ini, kompleksitasnya berkurang menjadi O(log n), sehingga jauh lebih efisien dibandingkan metode brute force, terutama untuk eksponen yang besar.

2. Tahapan Combine dalam pangkatDC()
   Proses combine dalam algoritma divide and conquer bertujuan untuk menyatukan hasil dari submasalah yang telah dipecah sebelumnya.
   Dalam metode pangkatDC(), langkah combine sudah ada dalam bagian yang mengalikan hasil dari pemanggilan rekursif. Proses ini memastikan bahwa nilai akhir dihitung dengan cara yang lebih optimal dibandingkan pendekatan iteratif.
3. Pada metode pangkatBF(), terdapat parameter a (sebagai basis) dan n (sebagai pangkat), padahal dalam kelas Pangkat17 sudah tersedia atribut yang menyimpan nilai basis dan pangkat tersebut. Penggunaan parameter dalam metode ini sebenarnya tidak diperlukan karena atribut instance yang ada di kelas sudah dapat diakses secara langsung.
   Metode pangkatBF() dapat dibuat tanpa parameter dengan langsung menggunakan atribut nilai dan pangkat dari kelas Pangkat17.
4.

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

Method pangkatBF() (Brute Force)
- Metode ini menggunakan pendekatan iteratif untuk menghitung perpangkatan

```
public class Sum17 {
    double keuntungan[];

    Sum17(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)) × (x^(n/2)).
- Jika pangkat ganjil, misalnya x^n = x × (x^((n-1)/2)) × (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

Class

```
public class Sum17 {
    double keuntungan[];

    Sum17(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

```
import java.util.Scanner;

public class MainSum17 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Masukkan Jumlah Elemen: ");
        int elemen = input.nextInt();

        Sum17 sm = new Sum17(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));


    }
}
```

Hasil 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 D:\Tugas Kuliah\SEMESTER 2\ASLD>
```

PERTANYAAN

1. Variabel mid diperlukan untuk membagi array menjadi dua bagian dalam metode Divide and Conquer. Dengan cara ini, rekursi dapat berjalan lebih optimal dan mempercepat proses perhitungan dibandingkan dengan pendekatan brute force.

2. Pernyataan ini berfungsi untuk menjumlahkan total keuntungan dari dua bagian array:lsum menghitung keuntungan pada bagian kiri array.rsum menghitung keuntungan pada bagian kanan array.Kedua hasil tersebut dijumlahkan agar mendapatkan total keuntungan keseluruhan.

3. Operasi lsum + rsum dibutuhkan karena metode Divide and Conquer membagi array menjadi dua bagian, menghitung keuntungan pada masing-masing bagian, lalu menggabungkannya. Jika tidak dilakukan penjumlahan ini, maka hanya sebagian nilai yang dihitung, sehingga total keuntungan tidak akan lengkap. Dengan menjumlahkan lsum dan rsum, semua elemen dalam array diperhitungkan dengan benar.

4. Base case dalam totalDC() adalah kondisi yang menghentikan rekursi, yaitu ketika hanya tersisa satu elemen dalam array. Base case ini penting agar rekursi tidak berjalan terus-menerus tanpa henti (infinite recursion) dan memungkinkan metode menyelesaikan perhitungan dengan benar.

5. totalDC() bekerja dengan membagi array menjadi dua bagian, menghitung total keuntungan dari masing-masing bagian secara rekursif, lalu menggabungkan hasilnya. Jika hanya ada satu elemen, nilai tersebut langsung dikembalikan. Pendekatan ini lebih efisien karena menyelesaikan masalah secara bertahap, bukan dengan menghitung semua elemen sekaligus seperti metode brute force.