LAPORAN HASIL PRAKTIKUM

ALGORITMA SISTEM DATA

JOBSHEET 5

IRFAN PANDU PRATAMA

244107020193

TI 1E

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2025

Percobaan 1

1. Kode program faktorial13

```
package Jobsheet5;

public class faktorial13 {
    int faktorialBF(int n) {
        int faktor = 1;
        for (int i = 1; i <=n; i++) {
            faktor = faktor * i;
        }
        return faktor;
        }

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

kode program mainfaktorial13

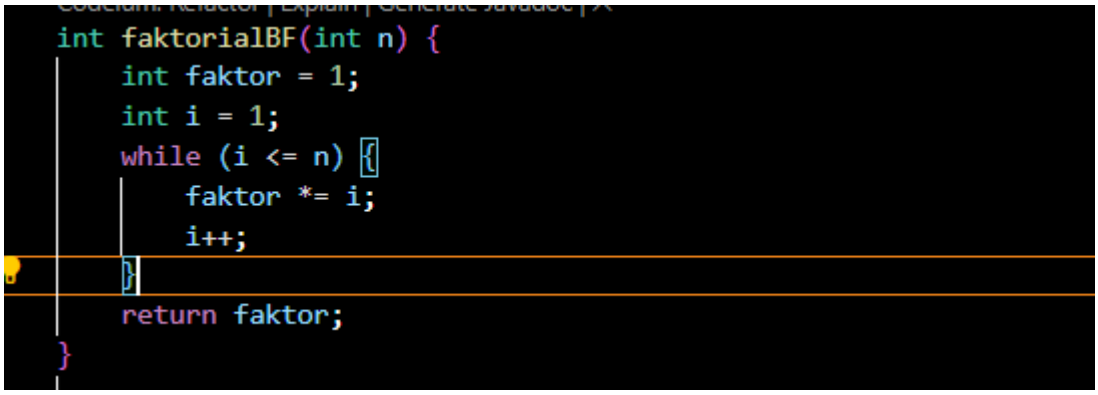
```
package Jobsheet5;
import java.util.Scanner;
public class MainFaktorial13 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan nilai: ");
        int nilai = sc.nextInt();

        faktorial13 fk = new faktorial13();
        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 :

```
orial13'
Masukkan nilai: 5
nilai faktorial 5 menggunakan BF: 120
nilai faktorial 5 menggunakan DC: 120
PS C:\Users\LENOVO\Praktikum-ASD>
```

Jawaban pertanyaan percobaan 1 :

1. Bagian `if` pada algoritma Divide and Conquer digunakan sebagai **base case** untuk menghentikan rekursi dengan mengembalikan nilai 1 saat `n == 0` atau `n == 1`, sedangkan bagian `else` adalah **recursive case** yang memanggil fungsi kembali dengan `n-1` hingga mencapai base case.
2. Perulangan pada method dapat diganti menggunakan selain for, berikut pembuktiannya



```
int faktorialBF(int n) {
    int faktor = 1;
    int i = 1;
    while (i <= n) {
        faktor *= i;
        i++;
    }
    return faktor;
}
```

3. `faktor *= i;` digunakan dalam perulangan untuk mengalikan nilai sebelumnya dengan `i` secara iteratif, sedangkan `int fakto = n * faktorialDC(n-1);` digunakan dalam rekursi untuk mengalikan `n` dengan hasil pemanggilan fungsi faktorial sebelumnya hingga mencapai base case.
4. Metode `faktorialBF()` menggunakan perulangan untuk menghitung faktorial secara iteratif, lebih efisien dalam penggunaan memori, sedangkan `faktorialDC()` menggunakan rekursi dengan pemanggilan fungsi berulang, lebih elegan namun memiliki overhead memori yang lebih besar.

Percobaan 2

2. Kode program pangkat13

```
package Jobsheet5;

public class pangkat13 {
    int  nilai, pangkat;

    public pangkat13(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;
    }
```

kode program pangkatmain13

```
package Jobsheet5;
import java.util.Scanner;
public class pangkatmain13 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan elemen: ");
        int elemen = sc.nextInt();

        pangkat13[] png = new pangkat13[elemen];
        for (int i = 0; i < elemen; i++) {
            System.out.print("Masukkan nilai basis elemen ke-"+(i+1)+":
");
            int basis = sc.nextInt();
            System.out.print("Masukkan pangkat elemen ke-"+(i+1)+": ");
            int pangkat = sc.nextInt();
            png[i] = new pangkat13(basis, pangkat);
        }
```

hasil kode program :

```
Masukkan elemen: 3
Masukkan nilai basis elemen ke-1: 2
Masukkan pangkat elemen ke-1: 3
Masukkan nilai basis elemen ke-2: 4
Masukkan pangkat elemen ke-2: 5
Masukkan nilai basis elemen ke-3: 6
Masukkan 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
PS C:\Users\LENOVO\Praktikum-ASD>
```

Jawaban pertanyaan percobaan 2 :

1. Perbedaan pangkatBF() dan pangkatDC(): pangkatBF() menggunakan perulangan untuk mengalikan nilai berulang kali sehingga lebih sederhana namun kurang efisien untuk pangkat besar, sedangkan pangkatDC() menggunakan rekursi dengan metode Divide and Conquer yang membagi pangkat menjadi dua untuk mengurangi jumlah perkalian, sehingga lebih cepat tetapi memerlukan pemanggilan fungsi berulang.
2. Tahap combine sudah ada dalam kode program diatas

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

3. Relevansi Parameter dalam pangkatBF(): Parameter dalam pangkatBF() tetap relevan karena memungkinkan fleksibilitas dalam menentukan nilai basis dan pangkat tanpa bergantung pada atribut kelas, tetapi metode ini juga bisa dibuat tanpa parameter dengan langsung menggunakan atribut kelas,contoh perubahan

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

4. Cara Kerja pangkatBF() dan pangkatDC(): pangkatBF() bekerja dengan iterasi, mengalikan basis sebanyak nilai pangkat sehingga lebih sederhana tetapi kurang efisien untuk angka besar, sedangkan pangkatDC() menggunakan rekursi untuk membagi perhitungan menjadi submasalah lebih kecil sehingga lebih cepat dengan kompleksitas logaritmik tetapi menggunakan lebih banyak memori untuk pemanggilan fungsi berulang.

Percobaan 3

1. Kode program sum13

```
package Jobsheet5;

public class sum13 {
    double keuntungan[];

    public sum13(int el){
        keuntungan = new double[el];
    }

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

kode program summain13

```
package Jobsheet5;
import java.util.Scanner;
public class summain13 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("masukkan jumlah elemen: ");
        int elemen = sc.nextInt();

        sum13 sm = new sum13(elemen);
        for (int i = 0; i < elemen; i++) {
            System.out.print("masukkan keuntungan ke-"+(i+1)+": ");
            sm.keuntungan[i] = sc.nextDouble();
        }

        System.out.println("Hasil total bruteforce: "+sm.totalBF());
        System.out.println("Hasil total 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
Hasil total bruteforce: 150.0
Hasil total divide and conquer: 150.0
```

Jawaban pertanyaan percobaan 3 :

1. Variabel mid pada metode TotalDC() dibutuhkan untuk membagi array atau rentang data menjadi dua bagian dalam proses rekursi, sehingga metode dapat menerapkan strategi Divide and Conquer dengan membagi masalah besar menjadi dua submasalah yang lebih kecil sebelum menggabungkan hasilnya.
2. Statement ini dilakukan dalam metode TotalDC() untuk membagi masalah menjadi dua submasalah dengan menggunakan Divide and Conquer, di mana lsum menghitung total dari bagian kiri array (dari indeks l hingga mid), sedangkan rsum menghitung total dari bagian kanan array (dari mid+1 hingga r), kemudian hasil kedua bagian ini akan digabungkan untuk mendapatkan total keseluruhan.
3. Penjumlahan lsum + rsum diperlukan untuk menggabungkan hasil perhitungan dari dua submasalah dalam metode Divide and Conquer, sehingga total nilai dari seluruh elemen dalam array dapat diperoleh dengan menjumlahkan hasil dari bagian kiri (lsum) dan bagian kanan (rsum).
4. Base case dari totalDC() adalah ketika array hanya memiliki satu elemen, sehingga fungsi langsung mengembalikan nilai elemen tersebut tanpa perlu membagi lebih lanjut.
5. Kesimpulan cara kerja totalDC(): Metode ini menggunakan pendekatan Divide and Conquer dengan membagi array menjadi dua bagian, menghitung jumlah masing-masing secara rekursif, lalu menggabungkan hasilnya untuk mendapatkan total keseluruhan.