

## JOB SHEET V

### BRUTE FORCE DAN DIVIDE CONQUER

#### 5.1 Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Mahasiswa mampu membuat algoritma brute force dan divide-conquer
2. Mahasiswa mampu menerapkan penggunaan algoritma brute force dan divide-conquer

#### 5.2 Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer

Perhatikan Diagram Class berikut ini:

Faktorial
faktorialBF(): int
faktorialDC(): int

Berdasarkan diagram class di atas, akan dibuat program class dalam Java. Untuk menghitung nilai faktorial suatu angka menggunakan 2 jenis algoritma, Brute Force dan Divide and Conquer. Jika digambarkan terdapat perbedaan proses perhitungan 2 jenis algoritma tersebut sebagai berikut : Tahapan pencarian nilai faktorial dengan algoritma

$$20! = 20 \times 19 \times 18 \times 17 \times \dots \times 5 \times 4 \times 3 \times 2 \times 1$$

Brute Force:

Tahapan pencarian nilai faktorial dengan algoritma Divide and Conquer:

$$20! = (20 \times 19) \times (18 \times 17) \times \dots \times (5 \times 4) \times (3 \times 2) \times 1$$

##### 5.2.1. Langkah-langkah Percobaan

1. Buat folder baru bernama **Jobsheet5** di dalam repository **Praktikum ASD**

**Praktikum-ASD / Jobsheet5 /**

2. Buatlah class baru dengan nama `Faktorial`



3. Lengkapi class `Faktorial` dengan atribut dan method yang telah digambarkan di dalam diagram class di atas, sebagai berikut

- a) Tambahkan method `faktorialBF()`:

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

- b) Tambahkan method `faktorialDC()`:

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

4. Coba jalankan (Run) class `Faktorial` dengan membuat class baru `MainFaktorial`.

- a) Di dalam fungsi `main` sediakan komunikasi dengan user untuk memasukkan nilai yang akan dicari faktorialnya

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

- b) Kemudian buat objek dari class `Faktorial` dan tampilkan hasil pemanggilan method

```
faktorialDC() dan faktorialBF()  
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));
```

d) Pastikan program sudah berjalan dengan baik!

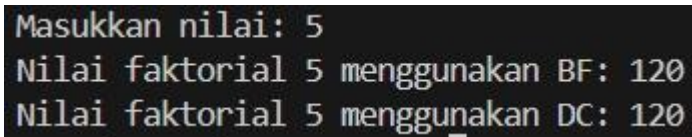
### 5.2.2. Verifikasi Hasil Percobaan

Masukkan nilai: 5

Nilai faktorial 5 menggunakan BF: 120

Nilai faktorial 5 menggunakan DC: 120

Cocokkan hasil compile kode program anda dengan gambar berikut ini.



```
Masukkan nilai: 5
Nilai faktorial 5 menggunakan BF: 120
Nilai faktorial 5 menggunakan DC: 120
```

### 5.2.3. 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()` !

### Jawaban Pertanyaan

1. Perbedaan if dan else di metode `faktorialDC()`

Dalam metode rekursif, ada dua bagian penting:

if (`n == 1`) → Ini adalah kondisi berhenti (base case). Kalau `n` sudah 1, langsung balikin nilai 1 karena faktorial 1 itu ya 1.

else → Kalau `n` lebih dari 1, hitung faktorial dengan cara `n * faktorialDC(n-1)`, yaitu kita panggil lagi metode ini untuk nilai `n-1`, terus begitu sampai akhirnya ketemu base case.

Jadi, if itu buat berhentiin rekursi biar nggak jalan terus tanpa ujung, dan else itu bagian yang bikin rekursi terus berjalan sampai selesai.

2. nggak harus pakai for, bisa juga dengan while atau bisa tanpa loop sama sekali kalau pakai rekursi.

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

3. `fakto *= i;` → Ini adalah perhitungan menggunakan perulangan. Nilai faktorial dihitung secara langsung dalam setiap iterasi, sehingga prosesnya berjalan secara bertahap.

`int fakto = n * faktorialDC(n-1);` → Ini adalah perhitungan menggunakan rekursi. Sebelum mendapatkan hasil akhir, metode ini terlebih dahulu memanggil dirinya sendiri hingga mencapai base case, baru kemudian mulai menghitung hasilnya dari yang paling kecil ke yang terbesar.

4. Metode `faktorialBF()` menggunakan pendekatan iteratif dengan perulangan (for atau while). Cara ini lebih efisien dalam penggunaan memori karena tidak menyimpan banyak pemanggilan fungsi. Proses perhitungannya berjalan secara langsung dari angka 1 hingga `n`.

Sementara itu, metode `faktorialDC()` menggunakan pendekatan rekursif, di mana fungsi memanggil dirinya sendiri sampai mencapai base case (`n == 1`). Cara ini lebih elegan dan sesuai dengan konsep matematis faktorial, tetapi jika `n` terlalu besar, metode ini bisa menyebabkan stack overflow karena terlalu banyak pemanggilan rekursi yang harus disimpan dalam memori.

Kesimpulannya, jika ingin cara yang lebih efisien dan aman untuk nilai `n` yang besar, metode iteratif (`faktorialBF()`) lebih baik digunakan. Namun, jika ingin pendekatan yang lebih matematis dan elegan, metode rekursif (`faktorialDC()`) dapat menjadi pilihan, dengan catatan harus berhati-hati terhadap batasan

memori.

### 5.3 Menghitung Hasil Pangkat dengan Algoritma Brute Force dan Divide and Conquer

Pada praktikum ini kita akan membuat program class dalam Java, untuk menghitung nilai pangkat suatu angka menggunakan 2 jenis algoritma, Brute Force dan Divide and Conquer. Pada praktikum ini akan digunakan **Array of Object** untuk mengelola beberapa objek yang akan dibuat, berbeda dengan praktikum tahap sebelumnya yang hanya berfokus pada 1 objek factorial saja.

#### 5.3.1. Langkah-langkah Percobaan

1. Buatlah class baru dengan nama **Pangkat**, dan di dalam class **Pangkat** tersebut, buat atribut angka yang akan dipangkatkan sekaligus dengan angka pemangkatnya

```
public class Pangkat {  
  
    int nilai, pangkat;  
  
}
```

2. Tambahkan konstruktor berparameter

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

3. Pada class **Pangkat** tersebut, tambahkan method **PangkatBF()**

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

4. Pada class **Pangkat** juga tambahkan method **PangkatDC()**

```

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));
        }
    }
}
}

```

5. Perhatikan apakah sudah tidak ada kesalahan yang muncul dalam pembuatan class Pangkat
6. Selanjutnya buat class baru yang di dalamnya terdapat method main. Class tersebut dapat dinamakan MainPangkat. Tambahkan kode pada class main untuk menginputkan jumlah elemen yang akan dihitung pangkatnya.

```

import java.util.Scanner;

public class MainPangkat {

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

```

7. Nilai pada tahap 5 selanjutnya digunakan untuk instansiasi array of objek. Di dalam Kode berikut ditambahkan proses pengisian beberapa nilai yang akan dipangkatkan sekaligus dengan pemangkatnya.

```

public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Masukka jumlah elemen: ");
    int elemen = input.nextInt();
    Pangkat[] png = new Pangkat[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 nilai pangkat elemen ke-"+(i+1)+" : ");
        int pangkat = input.nextInt();
    }
}

```

```

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

```

8. Kemudian, panggil hasil nya dengan mengeluarkan return value dari method `PangkatBF()` dan `PangkatDC()`.

```

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

```

### 5.3.2. Verifikasi Hasil Percobaan

Pastikan output yang ditampilkan sudah benar seperti di bawah ini.



```

Masukkan jumlah elemen: 3
Masukkan nilai basis elemen ke-1: 2
Masukkan nilai pangkat elemen ke-1: 3
Masukkan nilai basis elemen ke-2: 4
Masukkan nilai pangkat elemen ke-2: 5
Masukkan nilai basis elemen ke-3: 6
Masukkan 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

```

```

Masukka jumlah elemen: 3
Masukkan nilai basis elemen ke-1: 2
Masukkan nilai pangkat elemen ke-1: 3
Masukkan nilai basis elemen ke-2: 4
Masukkan nilai pangkat elemen ke-2: 5
Masukkan nilai basis elemen ke-3: 6
Masukkan 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
PS C:\Users\WINDOWS 11>

```

### 5.3.3. 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()` !

Jawaban Pertanyaan

1. Perbedaan `pangkatBF()` dan `pangkatDC()`

`pangkatBF()`: Menggunakan Brute Force, melakukan perkalian berulang sebanyak pangkatnya (iteratif).

`pangkatDC()`: Menggunakan Divide and Conquer, membagi masalah menjadi sub-masalah lebih kecil dan menggabungkan hasilnya (rekursif).

2. Dalam metode `pangkatDC()`, tahap *combine* pasti ada karena setelah sub-masalah selesai, hasilnya dikalikan kembali untuk mendapatkan nilai akhir. contoh jika kita menghitung  $2^8$  dibagi menjadi  $2^4$  dikali  $2^4$  setelah

mendapatkan hasil dari  $2^4$ , lalu mengalikannya kembali. jadi proses combine sudah termasuk dalam metode pangkatDC().

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

3.Bisa, akan seperti ini

Pada metdodnya:

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

```
int pangkatDC(int a, int n){
```

Cara pemanggilanya di main:

```
: "+p.pangkatBF());
```

4.Mengalikan hasil dengan basis sebanyak jumlah pangkat menggunakan perulangan, lalu mengembalikan hasil akhirnya.

Jika nilai pangkat besar, pecah masalah menjadi dua bagian dengan membagi n dengan 2. Lakukan rekursi dengan memanggil PangkatDC(a, n/2).

Jika pangkat bernilai ganjil, maka hasil akhirnya dikalikan dengan basis.

Terakhir, kembalikan hasil dari kombinasi rekursi tersebut.

## 5.4 Menghitung Sum Array dengan Algoritma Brute Force dan Divide and Conquer

Di dalam percobaan ini, kita akan mempraktekkan bagaimana proses *divide*, *conquer*, dan *combine* diterapkan pada studi kasus penjumlahan keuntungan suatu perusahaan dalam beberapa bulan.

### 5.4.1. Langkah-langkah Percobaan

1. Buat class baru yaitu class `Sum`. Tambahkan pula konstruktor pada class `Sum`.

```
public class Sum {  
    double keuntungan[];  
    Sum(int e1){  
        keuntungan = new double[e1];  
    }  
}
```

2. Tambahkan method `TotalBF()` yang akan menghitung total nilai array dengan cara *iterative*.

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

3. Tambahkan pula method `TotalDC()` untuk implementasi perhitungan nilai total array menggunakan algoritma 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. Buat class baru yaitu `MainSum`. Di dalam kelas ini terdapat method `main`. Pada method ini user dapat menuliskan berapa bulan keuntungan yang akan dihitung. Dalam kelas ini sekaligus dibuat instansiasi objek untuk memanggil atribut ataupun fungsi pada class `Sum`

```
import java.util.Scanner;

public class MainSum {

    Run | Debug
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print(s:"Masukkan jumlah elemen: ");
        int elemen = input.nextInt();
    }
}
```

5. Buat objek dari class `Sum`. Lakukan perulangan untuk mengambil input nilai keuntungan dan masukkan ke atribut `keuntungan` dari objek yang baru dibuat tersebut!

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

6. Tampilkan hasil perhitungan melalui objek yang telah dibuat untuk kedua cara yang ada (Brute Force dan Divide and Conquer)

```
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));
```

#### 5.4.2. Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
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
```

```
um
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\WINDOWS 11>
```

#### 5.4.3. Pertanyaan

1. Kenapa dibutuhkan variable `mid` pada method `TotalDC()`?
2. Untuk apakah statement di bawah ini dilakukan dalam `TotalDC()`?

---

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

3. Kenapa diperlukan penjumlahan hasil lsum dan rsum seperti di bawah ini?

```
return lsum+rsum;
```

4. Apakah base case dari `totalDC()` ?
5. Tarik Kesimpulan tentang cara kerja `totalDC()`

Jawaban Pertanyaan

1. Fungsi `mid` memiliki peran penting dalam metode Divide and Conquer karena menentukan titik pembagian array, memastikan proses rekursi berjalan dengan benar, serta membantu dalam menggabungkan hasil perhitungan dari dua bagian array.
2. Array dibagi menjadi dua bagian, lalu masing-masing bagian dihitung secara rekursif.
3. Setelah kedua bagian mencapai base case, nilainya akan dijumlahkan menggunakan sintaks itu.
4. `l == r`
5. Menggunakan pendekatan Divide and Conquer untuk menghitung jumlah elemen dalam array.