

ALGORITMA DAN STRUKTUR DATA

“Laporan hasil Praktikum pada Jobsheet 5 “Brute Force dan Divide Conquer” ”

Oleh:

Hafiz Rizqi Hernanda

NIM (244107020154)



Jurusan Teknologi informasi

Teknik Informatika

Politeknik Negeri Malang

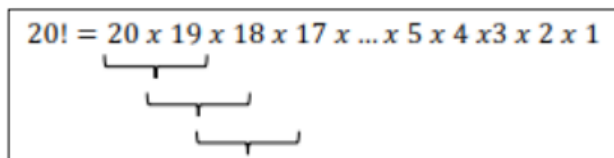
5.2 Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer

Perhatikan Diagram Class berikut ini:

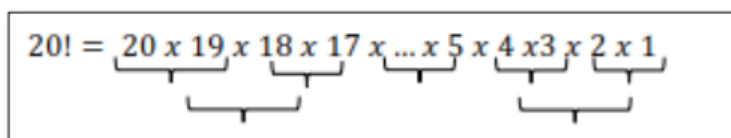
Faktorial
faktorialBF(): int faktorialDC(): int

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 Brute Force:

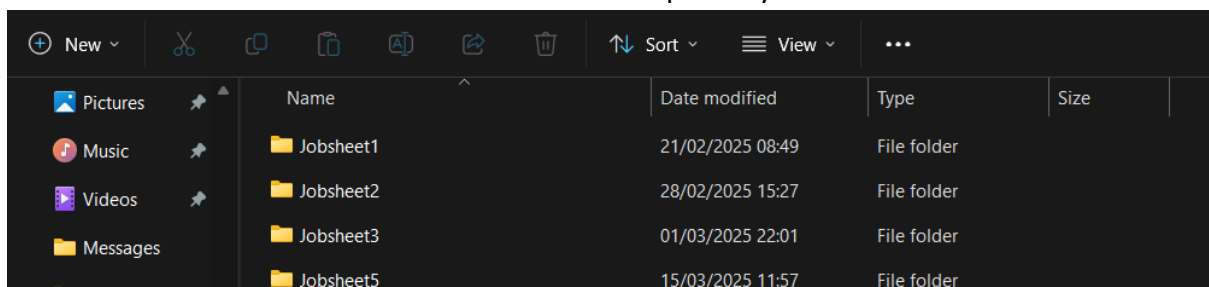
$$20! = 20 \times 19 \times 18 \times 17 \times \dots \times 5 \times 4 \times 3 \times 2 \times 1$$


Tahapan pencarian nilai faktorial dengan algoritma Divide and Conquer:

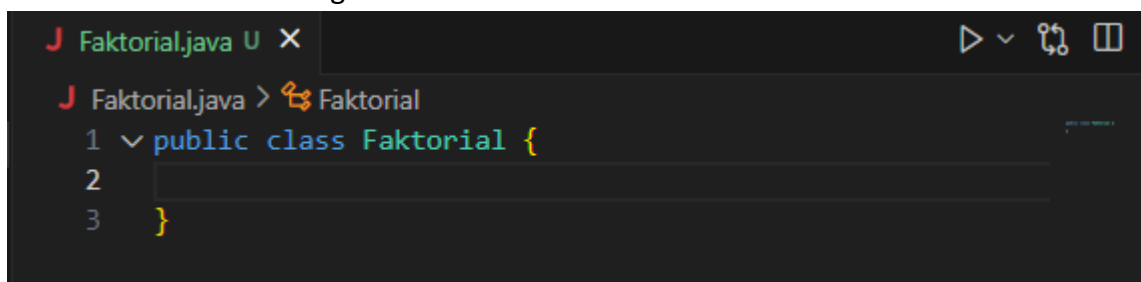
$$20! = (20 \times 19 \times 18 \times 17 \times \dots \times 5 \times 4 \times 3 \times 2 \times 1)$$


5.2.1. Langkah-langkah Percobaan

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



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 * faktorialDC(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

```
MainFaktorial.java > MainFaktorial > main(String[])  
1 import java.util.Scanner;  
2 public class MainFaktorial {  
3     public static void main(String[] args) {  
4         Scanner input = new Scanner(System.in);  
5         System.out.print("Masukkan nilai: ");  
6         int nilai = input.nextInt();  
7     }  
8 }  
9  
10
```

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));
```

c) Pastikan program sudah berjalan dengan baik!

5.2.2. Verifikasi Hasil Percobaan

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
```

```
PS C:\Regulus\Praktikum-ASD\Jobsheet5> & 'C:\Program Files\Java\jdk-9.0.4\bin\java.exe' -Xmx5m -jar %AppData%\Roaming\Code\User\workspaceStorage\9309bdfe274c83df4c\workspace\praktikum-asd-jobsheet5\src\main\Main.java
Masukkan nilai: 5
Nilai faktorial 5menggunakan BF: 120
Nilai faktorial 5menggunakan 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!

Method faktorialDC(int n) menggunakan konsep **Divide and Conquer**, yang berarti masalah besar dipecah menjadi submasalah yang lebih kecil.

- **Bagian if ($n == 1$)** adalah **base case**, yang menghentikan rekursi saat n mencapai 1, mengembalikan nilai 1.
- **Bagian else** adalah bagian rekursif yang memanggil dirinya sendiri dengan nilai $n-1$, sehingga terjadi pemecahan masalah menjadi submasalah yang lebih kecil sampai mencapai base case.

2. Apakah memungkinkan perulangan pada method faktorialBF() diubah selain menggunakan for? Buktikan!

Metode faktorialBF() saat ini menggunakan **loop for** untuk menghitung faktorial.

Perulangan bisa diganti menggunakan **while** atau **do-while**. Berikut adalah buktinya:

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

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

Hasil run:

```
PS C:\Regulus\Praktikum-ASD\Jobsheet5> c::; cd 'c:\Regulus\Praktikum-ASD\Jobsheet5'
Masukkan nilai: 5
Nilai faktorial 5 menggunakan BF: 120
Nilai faktorial 5 menggunakan DC: 120
PS C:\Regulus\Praktikum-ASD\Jobsheet5>
```

3. Jelaskan perbedaan antara fakto *= i; dan int fakto = n * faktorialDC(n-1); !
 fakto *= i; digunakan dalam **iterasi**, yang berarti nilai fakto diperbarui setiap kali loop berjalan. Sedangkan
 int fakto = n * faktorialDC(n-1); digunakan dalam **rekursi**, di mana pemanggilan rekursif terus berjalan hingga mencapai base case (n == 1), lalu hasilnya dikalikan secara bertahap saat fungsi kembali dari setiap pemanggilan rekursif.

4. Buat Kesimpulan tentang perbedaan cara kerja method faktorialBF() dan faktorialDC()!

a. **faktorialBF() (Brute Force)**

- Menggunakan **iterasi** dengan perulangan (for, while, atau do-while).
- Memiliki **Notasi Big O(n)** karena berjalan dalam loop dari 1 hingga n.
- Lebih efisien dalam penggunaan memori karena tidak menggunakan pemanggilan fungsi berulang.

b. **faktorialDC() (Divide and Conquer)**

- Menggunakan **rekursi**, membagi masalah besar menjadi masalah lebih kecil (n-1).
- Memiliki kompleksitas **O(n)** tetapi menggunakan **stack rekursi**, yang dapat menyebabkan **stack overflow** jika n terlalu besar.
- Lebih efisien dan sesuai untuk pemecahan masalah berbasis rekursi.

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

```
J Pangkat.java > Pangkat > pangkatDC(int, int)
1 public class Pangkat {
2     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.

```
MainPangkat.java > ...  
1 import java.util.Scanner;  
2 public class MainPangkat {  
    Run | Debug  
3     public static void main(String[] args) {  
4         Scanner input = new Scanner(System.in);  
5         System.out.print(s:"Masukkan jumlah elemen: ");  
6         int elemen = input.nextInt();  
7     }
```

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.

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

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

```
System.out.println(x:"HASIL PANGKAT BRUTEFORCE: ");
for (Pangkat p : png) {
    System.out.println(p.nilai+"^"+p.pangkat+": "+p.pangkatBF(p.nilai, p.pangkat));
}
System.out.println(x:"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
Masukan nilai basis elemen ke-1: 2
Masukan nilai pangkat elemen ke-1: 3
Masukan nilai basis elemen ke-2: 4
Masukan nilai pangkat elemen ke-2: 5
Masukan nilai basis elemen ke-3: 6
Masukan 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:\Regulus\Praktikum-ASD\Jobsheet5> & 'C:\Program Files\Java\jdk-23\bin\
AppData\Roaming\Code\User\workspaceStorage\9309bdfc274c83df4cafa3fa707ca1df
Masukkan jumlah elemen: 3
Masukan nilai basis elemen ke-1: 2
Masukan nilai pangkat elemen ke-1: 3
Masukan nilai basis elemen ke-2: 4
Masukan nilai pangkat elemen ke-2: 5
Masukan nilai basis elemen ke-3: 6
Masukan 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:\Regulus\Praktikum-ASD\Jobsheet5> 

```

5.3.3. Pertanyaan

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu pangkatBF() dan pangkatDC()!

pangkatBF() (Brute Force)

- Menggunakan **iterasi** (for-loop) untuk mengalikan angka secara berulang sebanyak n kali.
- Memiliki **Notasi Big O(n)** karena jumlah perulangan sebanding dengan nilai pangkat (n).

pangkatDC() (Divide and Conquer)

- Menggunakan **rekursi** untuk membagi perhitungan pangkat menjadi submasalah yang lebih kecil.
- Jika n genap: hasilnya dihitung dengan $(\text{pangkatDC}(a, n/2) * \text{pangkatDC}(a, n/2))$.
- Jika n ganjil: ditambahkan perkalian dengan a , yaitu $(\text{pangkatDC}(a, n/2) * \text{pangkatDC}(a, n/2) * a)$.
- Memiliki **Notasi Big O(log n)** karena jumlah pemanggilan rekursif berkurang secara eksponensial.

2. Apakah tahap combine sudah termasuk dalam kode tersebut? Tunjukkan!

Ya, tahap **Combine** sudah termasuk dalam kode pangkatDC(), pada bagian:

```

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. 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?

Apakah tetap relevan?

Saat ini, method `pangkatBF(int a, int n)` menerima parameter meskipun atribut `nilai` dan `pangkat` sudah ada di dalam kelas `Pangkat`. Ini membuat metode kurang efisien karena harus melewati nilai yang sebenarnya sudah tersimpan dalam objek.

Bisakah dibuat tanpa parameter?

Ya, metode bisa dibuat tanpa parameter dengan langsung menggunakan atribut `nilai` dan `pangkat` dalam kelas `Pangkat`. Berikut method `pangkatBF()` tanpa berparameter:

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

4. Tarik tentang cara kerja method `pangkatBF()` dan `pangkatDC()`!

- **pangkatBF()**
 - Menggunakan **iterasi**.
 - Notasi Big **$O(n)$** .
 - Mudah dipahami, tetapi kurang efisien untuk nilai pangkat yang besar.
- **pangkatDC()**
 - Menggunakan **rekursi** dengan metode **Divide and Conquer**.
 - Notasi Big **$O(\log n)$** , jauh lebih cepat dibandingkan metode iteratif.
 - Lebih efisien, tetapi menggunakan lebih banyak memori karena membutuhkan stack rekursi.

Kesimpulannya, **pangkatDC()** lebih efisien daripada **pangkatBF()**, terutama saat n besar. Namun, `pangkatBF()` lebih sederhana dan lebih mudah dipahami.

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.

```
J Sum.java > Sum > totalDC(double[], int, int)
1 public class Sum {
2
3     double keuntungan[];
4
5     Sum (int el) {
6         keuntungan = new double[el];
7     }
```

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

```
J MainSum.java > MainSum > main(String[])
1 import java.util.Scanner;
2 public class MainSum {
    Run | Debug
3     public static void main(String[] args) {
4         Scanner input = new Scanner(System.in);
5         System.out.print(s:"Masukkan jumlah elemen: ");
6         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!

```
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, 1:0 , 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
```

```
PS C:\Regulus\Praktikum-ASD\Jobsheet5> & 'C:\Program Files\Java\jdk-23\bin\java.exe' '--enable-preview' -cp 'AppData\Roaming\Code\User\workspaceStorage\9309bdfc274c83df4cafa3fa707ca1df\redhat.java\jdt_w
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:\Regulus\Praktikum-ASD\Jobsheet5> 
```

5.4.3. Pertanyaan

1. Kenapa dibutuhkan variable mid pada method TotalDC()?

Variabel mid digunakan untuk membagi array menjadi dua bagian dalam pendekatan **Divide and Conquer**. Dengan membagi array menjadi dua bagian yang lebih kecil, jadi kita bisa menghitung jumlah elemen di masing-masing bagian secara rekursif sebelum menggabungkan hasilnya.

2. Untuk apakah statement di bawah ini dilakukan dalam TotalDC()?

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

Kode program tersebut digunakan untuk melakukan pemanggilan rekursif pada dua bagian array yang telah dibagi:

- lsum menghitung jumlah elemen dari indeks l hingga mid.
- rsum menghitung jumlah elemen dari indeks mid+1 hingga r.

Dengan ini, setiap pemanggilan rekursif akan terus membagi array hingga mencapai kasus dasar, lalu menjumlahkan hasil dari kedua bagian.

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

```
return lsum+rsum;
```

Karena kita telah membagi array menjadi dua bagian, kita perlu menggabungkan kembali hasilnya. Dalam pendekatan **Divide and Conquer**, tahap **Combine** dilakukan dengan menjumlahkan hasil dari dua sub-array yang dihitung sebelumnya.

4. Apakah base case dari totalDC()?

Base case dalam totalDC() adalah ketika hanya ada satu elemen yang tersisa dalam array, yaitu ketika $l == r$. Pada kondisi ini, fungsi akan mengembalikan nilai dari elemen itu sendiri:

5. Tarik Kesimpulan tentang cara kerja totalDC()

- **Divide:** Array dibagi menjadi dua bagian berdasarkan nilai mid.
- **Conquer:** Setiap bagian dihitung totalnya secara rekursif.
- **Combine:** Hasil dari dua bagian dijumlahkan.

Pendekatan **Divide and Conquer** ini lebih efisien dibandingkan metode **Brute Force**, terutama untuk dataset besar, karena mengurangi jumlah operasi penjumlahan dibandingkan iterasi langsung pada seluruh array.

TANPA TUGAS....