

Nama : Muhammad Hikmal Al-Ghifary


Kelas : TI – 1B

Matkul : Praktikum Algoritma dan Struktur Data

BRUTE FORCE DAN DIVIDE CONQUER

Percobaan 1: Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer

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

 Jobsheet 5

Create ASD Pertemuan 5

18 minutes ago

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:

```
1 public class Faktorial15 {
2     int faktorialBF (int n){
3         int fakto = 1;
4         for (int i = 1; i <= n; i++) {
5             fakto = fakto * i;
6         }
7
8         return fakto;
9     }
10
11     int faktorialDC (int n) {
12         if (n==1) {
13             return 1;
14         } else {
15             int fakto = n * faktorialDC(n-1);
16             return fakto;
17         }
18     }
19 }
```

4. Jalankan (Run) class Faktorial dengan membuat class baru MainFaktorial. Tambahkan kode untuk menerima input, membuat objek, dan memanggil method.

```
1 import java.util.Scanner;
2
3 public class MainFaktorial15 {
4     Run | Debug
5     public static void main(String[] args) {
6         Scanner input = new Scanner (System.in);
7
8         System.out.print(s:"Masukkan nilai: ");
9         int nilai = input.nextInt();
10
11         Faktorial15 fk = new Faktorial15();
12         System.out.println("Nilai faktorial " + nilai + " menggunakan Brute Force: " + fk.faktorialBF(nilai));
13         System.out.println("Nilai faktorial " + nilai + " menggunakan Divide Conquer: " + fk.faktorialDC(nilai));
14     }
15 }
```

5. Pastikan program sudah berjalan dengan baik

Perbaiki kode program pada bagian operasi method `faktorialBF()`. Jika dikali 1, berapapun nilai yang diinput, maka akan tetap menghasilkan output 1. Hal yang perlu dilakukan adalah mengubahnya menjadi `fakto = fakto * i;`

6. Cek hasil

```
Masukkan nilai: 5
Nilai faktorial 5 menggunakan Brute Force: 120
Nilai faktorial 5 menggunakan Divide Conquer: 120
```

PERTANYAAN

1. Pada base line Algoritma Divide Conquer untuk melakukan pencarian nilai faktorial, jelaskan perbedaan bagian kode pada penggunaan if dan else!

Penggunaan if else pada method DC tersebut menggunakan kasus rekursif. Dimana fungsi tersebut akan selalu memanggil dirinya sendiri sebanyak n kali, hingga base line (batas akhir) tercapai.

Contoh: input 5, maka nilai n adalah 5. Karena n bukan sama dengan 1, maka perintah else akan dijalankan. Setelah operasi di else berjalan, maka nilai n sekarang adalah 4. Hal yang sama akan terus terjadi sampai dengan nilai `n == 1`. Saat nilai n adalah 1, maka perintah if akan dijalankan dan mengembalikan angka 1 sebagai nilai akhir faktorial.

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

Ya, bisa menggunakan while

```
2  int faktorialBF (int n){
3      int fakto = 1;
4      int i = 1;
5      while (i <= n) {
6          fakto *= i;
7          i++;
8      }
9
10     return fakto;
11 }
```

3. Jelaskan perbedaan antara `fakto *= i;` dan `int fakto = n * faktorialDC(n-1);`!

- Penggunaan `fakto *= i;` dikenal dengan perulangan iteratif. Nilai fakto akan selalu diperbarui dengan mengalikan nilai n, selama loop yang dijalankan masih memenuhi. Perulangan ini biasanya menggunakan perulangan / loop while maupun for.
- Penggunaan `fakto = n * faktorialDC(n-1);` dikenal dengan perulangan rekursif. Perulangan ini akan selalu memanggil dirinya sendiri dan mengalikan nilainya selama belum mencapai batas (*base case*).

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

Kesimpulan dari kedua method

- Penggunaan BF lebih mudah digunakan karena kode yang lebih mudah dipahami dan efisien dalam penggunaan memori. Jika menggunakan DC, rawan terjadi kesalahan jika kurang teliti dalam menetapkan base case dan operasi di else.
- Penggunaan DC lebih efektif untuk menyelesaikan masalah yang cukup rumit dan efisiensi algoritma yang tinggi karena penulisan kode lebih sederhana.

Percobaan 2: Menghitung Hasil Pangkat dengan Algoritma Brute Force dan Divide and Conquer

1. Buatlah class baru dengan nama Pangkat, dan di dalam class Pangkat tersebut, buat atribut angka yang akan dipangkatkan sekaligus dengan angka pemangkatnya
2. Tambahkan konstruktor berparameter
3. Pada class Pangkat tersebut, tambahkan method PangkatBF()
4. Pada class Pangkat juga tambahkan method PangkatDC()

```
1 public class Pangkat15 {
2     int nilai, pangkat;
3
4     Pangkat15(int n, int p){
5         nilai = n;
6         pangkat = p;
7     }
8
9     int pangkatBF(int a, int n){
10        int hasil = 1;
11        for (int i = 0; i < n; i++) {
12            hasil = hasil * nilai;
13        }
14
15        return hasil;
16    }
17
18    int pangkatDC(int a, int n){
19        if (n==0) {
20            return 1;
21        } else {
22            if (n%2==1) {
23                return (pangkatDC(a, n/2)*pangkatDC(a, n/2)*a);
24            } else {
25                return (pangkatDC(a, n/2)*pangkatDC(a, n/2));
26            }
27        }
28    }
29 }
```

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

```

1  import java.util.Scanner;
2
3  public class MainPangkat15 {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          System.out.print(s:"Masukkan jumlah elemen: ");
8          int elemen = input.nextInt();
9
10         Pangkat15[] png = new Pangkat15[elemen];
11         for (int i = 0; i < elemen; i++) {
12             System.out.print("Masukkan nilai basis elemen ke-" + (i+1) + ": ");
13             int basis = input.nextInt();
14             System.out.print("Masukkannilai pangkat elemen ke-" + (i+1) + ": ");
15             int pangkat = input.nextInt();
16             png[i] = new Pangkat15(basis, pangkat);
17         }
18
19         System.out.println(x:"HASIL PANGKAT BRUTE FORCE");
20         for (Pangkat15 p : png) {
21             System.out.println(p.nilai + "^" + p.pangkat + ": " + p.pangkatBF(p.nilai, p.pangkat));
22         }
23
24         System.out.println(x:"HASIL PANGKAT DIVIDE AND CONQUER");
25         for (Pangkat15 p : png) {
26             System.out.println(p.nilai + "^" + p.pangkat + ": " + p.pangkatDC(p.nilai, p.pangkat));
27         }
28     }
29 }

```

9. Cek hasil

```

Masukkan jumlah elemen: 3
Masukkan nilai basis elemen ke-1: 2
Masukkannilai pangkat elemen ke-1: 3
Masukkan nilai basis elemen ke-2: 4
Masukkannilai pangkat elemen ke-2: 5
Masukkan nilai basis elemen ke-3: 6
Masukkannilai pangkat elemen ke-3: 7
HASIL PANGKAT BRUTE FORCE
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\Muhammad Hikmal AG\OneDri

```

PERTANYAAN

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu `pangkatBF()` dan `pangkatDC()` !
 - Pada BF, metode yang digunakan adalah perulangan/*loop* berupa `for` untuk melakukan perkalian basis secara berulang. Nama lainnya adalah iteratif. Dari sisi efisiensi, penggunaannya kurang efisien jika digunakan untuk menghitung pangkat besar. Total pemanggilan fungsi yang dilakukan sebanyak 1 kali.
 - Pada DC, metode yang digunakan adalah rekursi untuk membagi perhitungan menjadi submasalah lebih kecil. Nama lainnya adalah rekursif. Penggunaannya lebih efisien karena jumlah perhitungan lebih sedikit. Fungsi dipanggil berulang kali dalam rekursi hingga mencapai base case `n == 0`.
2. Apakah tahap `combine` sudah termasuk dalam kode tersebut? Tunjukkan!

Tahap combine sudah ada dalam metode pangkatDC().

```
if (n==0) {
    return 1;
} 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));
    }
}
```

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?

Penggunaan parameter pada method tersebut tidak terlalu diperlukan, karena dalam class `Pangkat15` sudah ada atribut nilai dan pangkat.

Perubahan di class `Pangkat15`

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

Perubahan di class `MainPangkat15`

```
System.out.println(x:"HASIL PANGKAT BRUTE FORCE");
for (Pangkat15 p : png) {
    System.out.println(p.nilai + "^" + p.pangkat + ": " + p.pangkatBF()); // perubahan cara pemanggilan
}
```

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

- Penggunaan `PangkatBF()` dinilai lebih sederhana, namun kurang efisien jika nilai `n` besar. Sementara penggunaan `PangkatDC()` lebih efisien, namun rawan terjadi kesalahan jika kurang teliti dalam menentukan base case dan operasi pemanggilan fungsinya yang dapat menyebabkan *stack overflow*.
- Cara kerja `PangkatBF()` adalah melakukan perkalian nilai dengan dirinya sendiri sebanyak pangkat kali dalam perulangan `for`. Berbeda dengan `PangkatDC()` yang melakukan pemecahan masalah menjadi submasalah yang lebih kecil.

Percobaan 3: Menghitung Sum Array dengan Algoritma Brute Force dan Divide and Conquer

1. Buat class baru yaitu `class Sum`. Tambahkan pula konstruktor pada class `Sum`.
2. Tambahkan method `TotalBF()` yang akan menghitung total nilai array dengan cara iterative.
3. Tambahkan pula method `TotalDC()` untuk implementasi perhitungan nilai total array menggunakan algoritma Divide and Conquer

```
1 public class Sum15 {
2     double keuntungan[];
3
4     Sum15(int el){
5         keuntungan = new double[el];
6     }
7
8     double totalBF(){
9         double total = 0;
10        for (int i = 0; i < keuntungan.length; i++) {
11            total = total + keuntungan[i];
12        }
13
14        return total;
15    }
16
17    double totalDC(double arr[], int l, int r){
18        if (l==r) {
19            return arr[l];
20        }
21
22        int mid = (l+r)/2;
23        double lsum = totalDC(arr, l, mid);
24        double rsum = totalDC(arr, mid+1, r);
25        return lsum + rsum;
26    }
27 }
```

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`
5. Buat objek dari class `Sum`. Lakukan perulangan untuk mengambil input nilai keuntungan dan masukkan ke atribut `keuntungan` dari objek yang baru dibuat tersebut!
6. Tampilkan hasil perhitungan melalui objek yang telah dibuat untuk kedua cara yang ada (Brute Force dan Divide and Conquer)

```
1 import java.util.Scanner;
2
3 public class MainSum15 {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         System.out.print(s:"Masukkan jumlah elemen: ");
7         int elemen = input.nextInt();
8
9         Sum15 sm = new Sum15(elemen);
10        for (int i = 0; i < elemen; i++) {
11            System.out.print("Masukkan keuntungan ke-" + (i+1) + ": ");
12            sm.keuntungan[i] = input.nextDouble();
13        }
14
15        System.out.println("Total keuntungan menggunakan BruteForce: " + sm.totalBF());
16        System.out.println("Total keuntungan menggunakan Divide Conquer: " + sm.totalDC(sm.keuntungan, 1:0, elemen-1 ));
17    }
18 }
```

7. Cek hasil

```

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 Conquer: 150.0
PS C:\Users\Muhammad Hikmal AG\OneDrive\Documents\

```

PERTANYAAN

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

Variabel tersebut digunakan untuk membagi array menjadi 2 bagian (Divide Conquer). Pembagiannya adalah dengan menentukan titik tengah, kemudian dibagi menjadi subarray kanan serta subarray kiri.

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

```

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

```

Kode tersebut digunakan untuk memecah masalah menjadi dua bagian yang lebih kecil, yaitu:

- `lsum = totalDC(arr, l, mid);` => Menghitung jumlah keuntungan dari bagian kiri array.
- `rsum = totalDC(arr, mid + 1, r);` => Menghitung jumlah keuntungan dari bagian kanan array.

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

```

return lsum+rsum;

```

Penjumlahan tersebut harus ada dikarenakan array yang awalnya utuh, dibagi menjadi 2 subarray (left,right). Jika tidak ada operasi tersebut, maka nilai yang dikembalikan hanya setengah bagian dari total keseluruhan array.

4. Apakah base case dari totalDC () ?

Base case method tersebut adalah Jika `l == r`, maka hanya ada satu elemen dalam array, sehingga bisa langsung dikembalikan tanpa perlu melakukan rekursi lanjutan.

5. Tarik Kesimpulan tentang cara kerja totalDC () !

Bekerja secara DC (Divide Conquer)

- Divide : saat array dibagi menjadi subarray (left dan right)
- Conquer : saat menghitung jumlah keuntungan pada tiap bagian array
- Combine : menjumlahkan seluruh subarray menjadi hasil yang satu
- Base case : berhenti melakukan rekursi saat hanya tersisa 1 elemen (`l == r`)