Jobsheet 5

Brute Force and Divide Conquer

Ka Abi Muhammad R.F./12/TI-1B

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

b) Kemudian buat objek dari class Faktorial dan tampilkan hasil pemanggilan method faktorialDC() dan faktorialBF()

d) Pastikan program sudah berjalan dengan baik!

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

    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));
}
```

5.2.2. Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
\bin' 'Faktorial'
Masukkan nilai: 5
Nilai faktorial 5 Menggunakan BF: 120
Nilai faktorial 5 Menggunakan DC: 120
PS C:\Users\kaabi\OneDrive\Desktop\Praktikum ASD\Jobsheet 5>
```

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!

Jawab: Dalam algoritma **Divide and Conquer (DC)** untuk mencari nilai faktorial, penggunaan **if dan else** berfungsi untuk menentukan **base case** dan **recurrence relation**

Tanpa **base case (if)**, rekursi akan berjalan tanpa henti dan menyebabkan error. Tanpa **recursive case (else)**, algoritma tidak bisa memecah masalah dan hanya akan mengembalikan nilai tetap.

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

Jawab: Dapat menggunakan while loop

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

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

Jawab:

fakto *= i; === fakto = fakto * i;
Setiap iterasi, nilai fakto dikalikan dengan i, lalu i bertambah hingga mencapai n

fakto = 1

1 * 1 = 1

1 * 2 = 2

2 * 3 = 6

6 * 4 = 24

24 * 5 = 120

int fakto = n * faktorialDC(n-1); memanggil dirinya sendiri secara rekursif.
Jika n belum mencapai 1, fungsi akan terus memanggil faktorialDC(n-1)

faktorialDC(5) = 5 * faktorialDC(4)

faktorialDC(4) = 4 * faktorialDC(3)

faktorialDC(3) = 3 * faktorialDC(2)

faktorialDC(2) = 2 * faktorialDC(1)

faktorialDC(1) = 1 (Base case, rekursi berhenti di sini)

Setelah mencapai base case, hasil dihitung mundur hingga menghasilkan 120

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

Kesimpulan Perbedaan Cara Kerja faktorialBF() dan faktorialDC()

1. Pendekatan Algoritma
    - faktorialBF() (Brute Force) menggunakan perulangan (looping) untuk menghitung faktorial secara langsung.
    - faktorialDC() (Divide and Conquer) menggunakan rekursi, yaitu memanggil dirinya sendiri hingga mencapai kondisi dasar (base case).
2. Cara Perhitungan
    - faktorialBF() menghitung faktorial dengan melakukan perkalian dalam perulangan dari 1 ke n.
    - faktorialDC() menghitung faktorial dengan cara memecah masalah menjadi lebih kecil (n * faktorialDC(n-1)) hingga mencapai 1.
3. Efisiensi dan Penggunaan Memori
    - faktorialBF() lebih efisien, karena hanya menggunakan satu variabel dan tidak menyimpan pemanggilan fungsi dalam memori.
    - faktorialDC() kurang efisien, karena setiap pemanggilan fungsi disimpan dalam stack memori, yang bisa menyebabkan Stack Overflow untuk nilai n yang besar.
4. Kecepatan Eksekusi
    - faktorialBF() lebih cepat, karena langsung melakukan perhitungan dalam satu loop.
    - faktorialDC() lebih lambat, karena harus memanggil fungsi berkali-kali sebelum mendapatkan hasil akhir.
5. Kelebihan dan Kekurangan
    - faktorialBF(): Lebih sederhana, mudah dipahami, lebih cepat.
    - faktorialDC(): Lebih elegan, cocok untuk memahami konsep rekursi, tetapi kurang efisien untuk input besar.

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

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);
        }
        return (pangkatDC(a, n/2)* pangkatDC(a, n/2));
    }
}
Run | Debug
```

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.

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

    Pangkat[] png = new Pangkat[elemen];
    for (int i = 0; i < elemen; i++) {
        System.out.println("Masukkan nilai basis elemen ke-" + (i+1)+": ");
        int basis = input.nextInt();
        System.out.println("Masukkan nilai pangkat elemen ke-"+(i+1)+": ");
        int pangkat = input.nextInt();
        png[i] = new Pangkat(basis, pangkat);
    }
        System.out.println(x:"HASIL PANGKAT BRUTE FORCE:");
        for (Pangkat p : png){
            System.out.println(p.nilai+ "^"+ p.pangkat+": "+p.pangkatBF(p.nilai, p.pangkat));
        }
        System.out.println(x:"HASIL PANGKAT DEVIDE AND CONQUER:");
        for(Pangkat p : png){
            System.out.println(p.nilai + "^" + p.pangkat+ ": " + p.pangkatDC(p.nilai, p.pangkat));
        }
```

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.println("Masukkan nilai basis elemen ke-" + (i+1)+": ");
    int basis = input.nextInt();
    System.out.println("Masukkan 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 BRUTE FORCE:");
    for (Pangkat p : png){
        System.out.println(p.nilai+ "^"+ p.pangkat+": "+p.pangkatBF(p.nilai, p.pangkat));
    }
    System.out.println(x:"HASIL PANGKAT DEVIDE 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

```
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 BRUTE FORCE:
2^3: 8
4^5: 1024
6^7: 279936
HASIL PANGKAT DEVIDE AND CONQUER:
2^3: 8
4^5: 1024
6^7: 279936
6^7: 279936
HASIL PANGKAT DEVIDE AND CONQUER:
2^3: 8
4^5: 1024
6^7: 279936
PS C:\Users\kaabi\OneDrive\Desktop\Praktikum ASD\Jobsheet 5
>
```

5.3.3. Pertanyaan

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

PangkatBF()

- Menggunakan perulangan for untuk mengalikan a dengan dirinya sendiri sebanyak n kali.
- Dimulai dari hasil = 1, lalu dikalikan a dalam setiap iterasi.
- Sifatnya iteratif, artinya setiap perhitungan dilakukan secara langsung dalam loop tanpa memecah masalah lebih kecil.

Iterasi 1: hasil = 1 * 2 = 2

Iterasi 2: hasil = 2 * 2 = 4

Iterasi 3: hasil = 4 * 2 = 8

Iterasi 4: hasil = 8 * 2 = 16

Output: 2^4 = 16

PangkatDC()

- Menggunakan rekursi (pemanggilan fungsi dirinya sendiri) untuk menghitung pangkat dengan strategi Divide and Conquer.
- Memecah masalah menjadi dua bagian yang lebih kecil hingga mencapai base case (n == 1).
- Jika n ganjil, hasilnya dikalikan dengan a tambahan.
- Jika n genap, cukup mengalikan dua hasil rekursi.

Misal, a=2, n=4

pangkatDC(2,4) → (pangkatDC(2,2) * pangkatDC(2,2))

pangkatDC(2,2) → (pangkatDC(2,1) * pangkatDC(2,1))

pangkatDC(2,1) → 2 (Base Case)

Menghitung mundur:

pangkatDC(2,2) = 2 * 2 = 4

pangkatDC(2,4) = 4 * 4 = 16

Output: 2^4 = 16

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

return (pangkatDC(a, n / 2) * pangkatDC(a, n / 2) * a);

atau

return (pangkatDC(a, n / 2) * pangkatDC(a, n / 2));

Jadi, tahap Combine terjadi saat hasil dari rekursi digabungkan kembali dengan perkalian, yang ditunjukkan dalam kode return (pangkatDC(a, n / 2) * pangkatDC(a, n / 2) * a);

3. Pada method pangkatBF()terdapat parameter untuk melewatkan 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?

Metode pangkatBF() saat ini memang menerima parameter, meskipun atribut nilai dan pangkat sudah tersedia dalam kelas Pangkat. Dalam hal ini, metode tersebut tetap relevan jika ingin digunakan secara fleksibel untuk berbagai nilai tanpa harus bergantung pada atribut instance.

Namun, jika ingin mengubahnya agar tidak menggunakan parameter dan langsung menggunakan atribut instance nilai dan pangkat

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

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

PangkatBF()

- Menggunakan perulangan for untuk mengalikan a dengan dirinya sendiri sebanyak n kali.
- Dimulai dari hasil = 1, lalu dikalikan a dalam setiap iterasi.
- Sifatnya iteratif, artinya setiap perhitungan dilakukan secara langsung dalam loop tanpa memecah masalah lebih kecil.

Iterasi 1: hasil = 1 * 2 = 2

Iterasi 2: hasil = 2 * 2 = 4

Iterasi 3: hasil = 4 * 2 = 8

Iterasi 4: hasil = 8 * 2 = 16

Output: 2^4 = 16

PangkatDC()

- Menggunakan rekursi (pemanggilan fungsi dirinya sendiri) untuk menghitung pangkat dengan strategi Divide and Conquer.
- Memecah masalah menjadi dua bagian yang lebih kecil hingga mencapai base case (n == 1).
- Jika n ganjil, hasilnya dikalikan dengan a tambahan.
- Jika n genap, cukup mengalikan dua hasil rekursi.

Misal, a=2, n=4

pangkatDC(2,4) → (pangkatDC(2,2) * pangkatDC(2,2))

pangkatDC(2,2) → (pangkatDC(2,1) * pangkatDC(2,1))

pangkatDC(2,1) → 2 (Base Case)

Menghitung mundur:

pangkatDC(2,2) = 2 * 2 = 4

pangkatDC(2,4) = 4 * 4 = 16

Output: 2^4 = 16

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.

```
Sum(int el){
    keuntungan = new double[el];
}
double totalBF(){
```

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

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

Sum sm = new Sum(elemen);
for (int i = 0; i < elemen; i++) {
    System.out.println("Masukkan kentungan ke- "+ (i+1)+": ");
    sm.keuntungan[i] = input.nextDouble();
}
System.out.println("Total keuntungan menggunakan BF: "+ sm.totalBF());
System.out.println("Total keuntungan menggunakan D&C: "+sm.totalDC(sm.keuntungan, l:0, elemen-1));
}
```

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

```
for (int i = 0; i < elemen; i++) {
    System.out.println("Masukkan kentungan 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 BF: "+ sm.totalBF());
System.out.println("Total keuntungan menggunakan D&C: "+sm.totalDC(sm.keuntungan, l:0, elemen-1));
}
```

5.4.2. Verifikasi Hasil Percobaan Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
Jobsheet 5_88ccf3f4\bin' 'Sum'
Masukkan jumlah elemen: 5
Masukkan kentungan ke- 1:
10
Masukkan kentungan ke- 2:
20
Masukkan kentungan ke- 3:
30
Masukkan kentungan ke- 4:
40
Masukkan kentungan ke- 5:
50
Total keuntungan menggunakan BF: 150.0
Total keuntungan menggunakan D&C: 150.0
PS C:\Users\kaabi\OneDrive\Desktop\Praktikum ASD\Jobsheet 5>
```

5.4.3. Pertanyaan

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

Variabel mid berfungsi sebagai titik tengah array, yang digunakan untuk membagi array menjadi dua bagian:

- Bagian kiri: dari indeks l sampai mid
- Bagian kanan: dari indeks mid + 1 sampai r

Setelah array terbagi, metode totalDC() akan memanggil dirinya sendiri (rekursi) untuk menghitung jumlah total dari kedua bagian, lalu menjumlahkannya:

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

Fungsi Statement

double lsum = totalDC(arr, l, mid);

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

1. Memecah Masalah (Divide)
    - totalDC(arr, l, mid) menghitung total keuntungan dari bagian kiri array (dari indeks l sampai mid).
    - totalDC(arr, mid + 1, r) menghitung total keuntungan dari bagian kanan array (dari indeks mid + 1 sampai r).
2. Menghitung Hasil dari Submasalah (Conquer & Combine)
    - Setelah nilai lsum dan rsum diperoleh dari pemanggilan rekursi, kedua hasil tersebut digabungkan dengan:

      return lsum + rsum;

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

Untuk menggabungkan nilai yang sudah dipisah dari kiri dan kanan

4. Apakah base case dari totalDC()?

```
if (l == r) {
   return arr[l];
}
```

5. Tarik Kesimpulan tentang cara kerja totalDC()

Kesimpulan Cara Kerja totalDC()

1. Divide (Pecah Masalah)
    - Array dibagi menjadi dua bagian yang lebih kecil menggunakan variabel mid.
    - Pembagian ini terus dilakukan secara rekursif sampai setiap bagian hanya memiliki satu elemen (base case).
2. Conquer (Selesaikan Per Bagian)
    - Setelah mencapai base case (satu elemen), fungsi mulai mengembalikan nilai elemen tersebut.
    - Setiap pemanggilan rekursi menghitung jumlah total dari bagian kiri dan kanan.
3. Combine (Gabungkan Hasil)
    - Nilai dari kedua bagian (kiri dan kanan) dijumlahkan secara bertahap dari rekursi terakhir hingga kembali ke pemanggilan pertama.
    - Hasil akhirnya adalah total seluruh elemen dalam array.