Niky Tania Sari (18/SIB-1B)

JOOBSHET 5 ALSD

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

```java
package minggu5;

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

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

```java
package minggu5;
import java.util.Scanner;

public class MainFaktorial {
    Run main | Debug main | 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));
    }
}
```

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

Pertanyaan 1

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

- **if (n == 1)**: Ini adalah *base case* atau kondisi penghentian rekursi. Ketika n bernilai 1, fungsi akan mengembalikan nilai 1 karena faktorial dari 1 adalah 1. Ini menghentikan rekursi dan mencegah infinite loop.
- **else**: Ini adalah bagian *recursive case*, di mana nilai n dikalikan dengan hasil rekursi faktorialDC(n-1). Artinya, fungsi akan terus memanggil dirinya sendiri dengan n-1 hingga mencapai *base case*.

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

   Jawaban: Bisa, saya menggunakan perulangan do-while

```java
package minggu5;

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

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

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

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

   Jawaban:

   - fakto *= i;
     - Operasi perulangan yang mengalikan nilai fakto dengan i secara iteratif
     - Digunakan dalam metode *Brute Force* (faktorialBF) untuk menghitung faktorial secara berurutan.
       Contoh: Jika n = 4, maka fakto akan dihitung sebagai 1 * 1 * 2 * 3 * 4.
   - int fakto = n * faktorialDC(n-1);
     - Operasi rekursif yang memanggil fungsi faktorialDC dengan parameter n-1
     - Digunakan dalam metode Divide and *Conquer* (faktorialDC) untuk membagi masalah menjadi sub-masalah yang lebih kecil
       Contoh: Jika n = 4, maka
       4 * faktorialDC(3)

3 * faktorialDC(2)
2 * faktorialDC(1), dan seterusnya hingga mencapai base case.

4. Buat Kesimpulan tentang perbedaan cara kerja method faktorialBF() dan faktorialDC()!
   Jawaban:
   - faktorialBF()
     - Menggunakan pendekatan *Brute Force* dengan perulangan iteratif (for, while, atau do-while).
     - Menghitung faktorial secara langsung dengan mengalikan angka dari 1 hingga n.
     - Tidak melibatkan pemanggilan fungsi rekursif.
     - Lebih sederhana dan mudah dipahami, tetapi kurang efisien untuk masalah yang lebih kompleks.
   - faktorialDc()
     - Menggunakan pendekatan Divide and Conquer dengan rekrusi
     - Membagi masalah menjadi sub-masalah yang lebih kecil (n-1) dan menyelesaikannya secara rekrusif
     - Memerlukan base cae untuk mengentikan rekrusi

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

```
package minggu5;

public class Pangkat {
  int nilai, pangkat;

  Pangkat(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;
  }

  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));
      }
    }
  }
}
```

```java
package minggu5;
import java.util.Scanner;

public class MainPangkat {
    Run | Debug | Run main | Debug main
    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.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 Pangkat(basis, pangkat);
        }
        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));
        }
    }
}
```

```
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
```

Pertanyaan 2

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu *pangkatBF()* dan *pangkatDC()*!
   Jawaban:
   - **pangkatBF() (Brute Force)**
     - Menggunakan *loop* untuk mengalikan basis sebanyak n kali
       Contoh: Jika a = 2 dan n = 3, maka 2^3 = 2 * 2 * 2 = 8
     - Sederhana dan mudah dipahami, tetapi tidak optimal untuk nilai n yang besar.

- Kompleksitas waktu: **O(n)** karena ada iterasi sebanyak n kali.

- **pangkatDC() (Divide and Conquer)**
  - Menggunakan rekursi untuk membagi pangkat menjadi dua bagian yang lebih kecil (n/2)
  - Jika pangkat genap, cukup mengalikan hasil dari dua sub-masalah yang sama (a^(n/2) * a^(n/2))
  - Jika pangkat ganjil, perlu dikalikan sekali lagi dengan basis a (a^(n/2) * a^(n/2) * a)
  - Kompleksitas waktu: **O(log n)** karena setiap langkah memangkas jumlah operasi menjadi setengahnya.

2. Apakah tahap *combine* sudah termasuk dalam kode tersebut?Tunjukkan!
   Jawaban: Ya, tahap **combine** sudah termasuk dalam kode pangkatDC(). Tahap combine terjadi ketika hasil dari sub-masalah digabungkan untuk menghasilkan solusi akhir

```
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 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?
   Jawaban:
   - Tidak sepenuhnya relevan, karena dalam class Pangkat sudah ada atribut nilai dan pangkat. Jika method ini dipanggil dalam konteks objek Pangkat, seharusnya cukup menggunakan atribut yang sudah ada, tanpa perlu menerima parameter lagi
   - Ya, method ini bisa dibuat tanpa parameter dengan langsung menggunakan atribut nilai dan pangkat dari objek tersebut.

   Method tanpa parameter:

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

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

- **pangkatBF()**:
  - Menggunakan iterasi dengan *loop* untuk mengalikan nilai dasar secara langsung.
  - Efisien untuk nilai pangkat kecil, tetapi kurang optimal untuk pangkat besar karena memiliki **O(n)** kompleksitas waktu.
- **pangkatDC()**:
  - Menggunakan rekursi dengan strategi *Divide and Conquer*, yang membagi masalah menjadi dua bagian yang lebih kecil.
  - Lebih cepat untuk pangkat besar karena hanya membutuhkan **O(log n)** langkah.
  - Menggunakan lebih banyak memori dibanding metode iteratif karena memanggil fungsi secara rekursif.

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

```java
package minggu5;

public class Sum {
  double keuntungan[];

  Sum(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;
  }
  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;
  }
}
```

```java
package minggu5;
import java.util.Scanner;

public class MainSum {
  Run | Debug | Run main | Debug main
  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.print("Masukkan keuntungan ke-"+ (i+1) +": ");
      sm.keuntungan[i] = input.nextDouble();
    }

    System.out.println("Total keuntungan menggunakan Bruteforce: "+ sm.totalBF());
    System.out.println("Total keuntungan menggunakan Divide and Conquer: "+ sm.totalDC(sm.keuntungan,l:0,elemen-1));

  }
}
```

```
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
```

Pertanyaan 3

1. Kenapa dibutuhkan variable *mid* pada method *TotalDC()*?
   Jawaban: Variabel *mid* digunakan untuk membagi array menjadi dua bagian: bagian kiri (*l* hingga *mid*) dan bagian kanan (*mid + 1* hingga *r*). Ini adalah konsep dasar dari metode Divide and Conquer, di mana masalah besar dipecah menjadi dua sub-masalah yang lebih kecil.

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

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

   Jawaban: Kedua statement ini digunakan untuk **membagi dan menaklukkan** (Divide and Conquer)

   - totalDC(arr, l, mid) menghitung jumlah elemen di bagian kiri array.
   - totalDC(arr, mid+1, r) menghitung jumlah elemen di bagian kanan array.
   - Hasil dari kedua bagian ini kemudian dijumlahkan untuk mendapatkan hasil akhir.

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

   ```
   return lsum+rsum;
   ```

   Jawaban: Karena setelah membagi array menjadi dua bagian, kita perlu menggabungkan kembali hasil perhitungan dari dua bagian tersebut untuk mendapatkan total keseluruhan.

4. Apakah base case dari *totalDC()*?
   Jawaban: Base case terjadi ketika l == r, yaitu saat hanya ada satu elemen dalam array.

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

5. Tarik Kesimpulan tentang cara kerja *totalDC()*
   Jawaban:
   - totalDC() bekerja dengan cara membagi array menjadi dua bagian hingga mencapai elemen terkecil (basis rekursi).

- Setelah mencapai base case, nilai dari sub-masalah dijumlahkan kembali hingga mendapatkan total keseluruhan array.
- Metode ini lebih efisien dibandingkan Brute Force untuk dataset besar karena menggunakan pendekatan Divide and Conquer, yang memiliki kompleksitas O(n log n) dibandingkan O(n) untuk Brute Force.

Latihan Praktikum

```java
package minggu5;

public class Nilai {
  int uts[];
  int uas[];

  Nilai(int uts[], int uas[]) {
    this.uts = uts;
    this.uas = uas;
  }

  // Mencari nilai UTS tertinggi dengan Divide and Conquer
  int maxUTS(int arr[], int l, int r) {
    if (l == r) {
      return arr[l];
    }
    int mid = (l + r) / 2;
    int leftMax = maxUTS(arr, l, mid);
    int rightMax = maxUTS(arr, mid + 1, r);
    return Math.max(leftMax, rightMax);
  }

  // Mencari nilai UTS terendah dengan Divide and Conquer
  int minUTS(int arr[], int l, int r){
    if (l == r) {
      return arr[l];
    }
  int mid = (l + r) / 2;
  int leftMin = minUTS(arr, l, mid);
  int rightMin = minUTS(arr, mid + 1, r);
  return Math.min(leftMin, rightMin);
  }

  // Menghitung rata-rata nilai UAS menggunakan Brute Force
  double rataUAS(){
    int total = 0;
    for (int i = 0; i < uas.length; i++) {
      total += uas[i];
    }
      return (double) total / uas.length;
  }
}
```

```java
package minggu5;

public class MainNilai {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        int uts[] = {78, 85, 90, 76, 92, 88, 80, 82};
        int uas[] = {82, 88, 87, 79, 95, 85, 83, 84};

        Nilai nl = new Nilai(uts, uas);

        int maxUTS = nl.maxUTS(uts, l:0, uts.length - 1);
        int minUTS = nl.minUTS(uts, l:0, uts.length - 1);
        double rrUAS = nl.rataUAS();

        System.out.println("Nilai UTS tertinggi (Divide and Conquer): " + maxUTS);
        System.out.println("Nilai UTS terendah (Divide and Conquer): " + minUTS);
        System.out.printf(format:"Rata-rata nilai UAS (Brute Force): %.2f\n", rrUAS);
    }
}
```

```
Nilai UTS tertinggi (Divide and Conquer): 92
Nilai UTS terendah (Divide and Conquer): 76
Rata-rata nilai UAS (Brute Force): 85.38
PS C:\Matkul>
```