LAPORAN HASIL PRAKTIKUM

ALGORITMA DAN STRUKTUR DATA

JOBSHEET 5

ATHAULLA HAFIZH

244107020030

TI 1 E

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

Percobaan 1

Kode Program Class

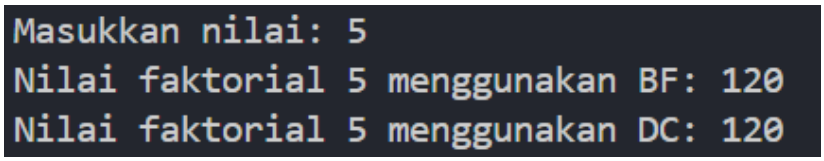
```
package Jobsheet5;
public class Faktorial {
    int faktorialBF (int n){
        int fakto = 1;
        int i = 1;
        while (i <= n){
            fakto = fakto * i;
            i++;
        }
        return fakto;
    }

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

Kode program Main

```
package Jobsheet5;
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();
```

```
        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));

    }

}
```

Output



Pertanyaan

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

   Perbedaannya adalah bagian if (n == 1) adalah base case yang menghentikan rekursif, mengembalikan 1 karena faktorial dari 1 adalah 1. Sementara itu, bagian else menangani kasus rekursif ketika nilai n lebih besar dari 1, di mana fungsi akan memanggil dirinya sendiri untuk menghitung faktorial dari n-1, kemudian mengalikan hasil rekursif tersebut dengan n. Proses ini berlanjut hingga mencapai base case, sehingga akhirnya menghasilkan nilai faktorial yang diinginkan. Singkatnya, bagian if berfungsi untuk menghentikan rekursif, sementara else digunakan untuk melanjutkan rekursif dengan memecah masalah menjadi sub masalah yang lebih kecil.

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

   Bisa, dengan merubah perulangan for menggunakan perulangan while. Seperti kode program dibawah ini :

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

Output

```
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); !

   Fakto *= i; adalah operasi iteratif yang digunakan dalam perulangan untuk menghitung nilai faktorial secara bertahap. Setiap iterasi perulangan mengalikan nilai fakto dengan i, dimulai dari 1 hingga mencapai nilai n, sedangkan int fakto = n * faktorialDC(n-1); digunakan dalam rekursi untuk menghitung faktorial, di mana fungsi faktorialDC() memanggil dirinya sendiri dengan argumen yang lebih kecil (n-1), hingga mencapai kondisi dasar (base case).

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

   Keduanya menghitung faktorial, namun dengan pendekatan yang berbeda. faktorialBF() menggunakan metode iteratif, yaitu dengan menggunakan perulangan (seperti for atau while) untuk menghitung faktorial secara bertahap, mulai dari 1 hingga n. Setiap iterasi mengalikan nilai sebelumnya dengan angka berikutnya hingga mencapai faktorial yang diinginkan. Di sisi lain, faktorialDC() menggunakan pendekatan rekursif dengan memecah masalah faktorial menjadi sub-masalah yang lebih kecil. Fungsi ini memanggil dirinya sendiri dengan nilai yang lebih kecil (n-1) hingga mencapai kondisi dasar (n == 1), dan hasilnya dikembalikan melalui proses rekursif. Dengan demikian, faktorialBF() lebih langsung dalam prosesnya menggunakan perulangan, sedangkan faktorialDC() mengandalkan rekursi untuk menyelesaikan masalah dengan membagi tugas menjadi langkah-langkah yang lebih kecil.

Percobaan 2

Kode Program Class

```
    }

    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));
            }
        }
    }
}
```

Kode program Main

```
package Jobsheet5;
import java.util.Scanner;
public class MainPangkat {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("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("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));

        }

    }

}
```

Output

```
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

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

   Perbedaannya adalah pangkatBF() menggunakan pendekatan iteratif, di mana perulangan dilakukan sebanyak n kali untuk mengalikan nilai a, sedangkan pangkatDC() menggunakan pendekatan rekursif dengan prinsip Divide and Conquer, membagi masalah menjadi sub-masalah yang lebih kecil dan menghitung pangkat dengan lebih sedikit perkalian, sehingga lebih efisien, terutama untuk nilai n yang besar. pangkatDC() lebih optimal dalam hal waktu eksekusi dibandingkan pangkatBF().

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

   Ya, tahap combine sudah termasuk dalam kode pada metode pangkatDC(), yang menggunakan pendekatan Divide and Conquer. Tahap combine terjadi ketika hasil rekursif dari sub-masalah digabungkan untuk menghasilkan solusi akhir. Dalam kode, saat n ganjil, hasil rekursif dari pangkatDC(a, n/2) dihitung dua kali dan digabungkan dengan mengalikan dengan a. Sementara untuk n genap, hasil rekursif dihitung dua kali tanpa perlu mengalikan dengan 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?

   Meskipun di class Pangkat sudah ada atribut nilai dan pangkat, metode pangkatBF() tetap relevan memiliki parameter karena memungkinkan fleksibilitas untuk menghitung pangkat dengan nilai yang berbeda tanpa mengubah nilai atribut class. Namun, jika metode pangkatBF() dibuat tanpa parameter, kita bisa memanfaatkan atribut nilai dan pangkat yang sudah ada di class untuk melakukan perhitungan tanpa perlu melewatkan parameter saat memanggil metode tersebut. Dengan demikian, metode pangkatBF() dapat diubah kode programnya menjadi :

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

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

   Method pangkatBF() menghitung pangkat dengan mengalikan nilai a sebanyak n kali dalam loop (iteratif). Sedangkan pangkatDC() menggunakan pendekatan rekursif dengan membagi nilai n menjadi dua bagian lebih kecil, mengalikan hasil rekursif, dan menyesuaikan untuk nilai ganjil dengan mengalikan hasil rekursif dengan a. Pendekatan pangkatDC() lebih efisien dibandingkan pangkatBF().

Percobaan 3

Kode Program Class

```
package Jobsheet5;
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;
    }
}
```

Kode program Main

```
package Jobsheet5;

import java.util.Scanner;

public class MainSum {

    public static void main(String[] args) {

        Scanner input = new Scanner (System.in);

        System.out.print("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, 0, elemen - 1));

    }

}
```

Output

```
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

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

Variabel mid dibutuhkan pada method totalDC() untuk membagi array menjadi dua bagian yang lebih kecil, yaitu bagian kiri (dari indeks l hingga mid) dan bagian kanan (dari indeks mid + 1 hingga r) / menggunakan pendekatan divide and conquer.

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

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

Statement double lsum = totalDC(arr, l, mid); dan double rsum = totalDC(arr, (mid + 1), r); digunakan untuk memanggil fungsi totalDC() secara rekursif untuk menghitung jumlah elemen pada dua bagian array yang dibagi. lsum menghitung jumlah elemen dari bagian kiri array (dari indeks l hingga mid), sedangkan rsum menghitung jumlah elemen dari bagian kanan array (dari indeks mid + 1 hingga r).

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

```
return lsum+rsum;
```

Penjumlahan hasil lsum dan rsum diperlukan untuk menggabungkan hasil penjumlahan elemen-elemen pada dua bagian array yang telah dihitung secara rekursif. Setelah array dibagi menjadi dua bagian (kiri dan kanan), lsum menyimpan jumlah elemen pada bagian kiri, sementara rsum menyimpan jumlah elemen pada bagian kanan. Dengan menjumlahkan keduanya (lsum + rsum), kita mendapatkan jumlah total seluruh elemen dalam rentang indeks yang diberikan (dari l hingga r).

4. Apakah base case dari totalDC()?

Base case dari totalDC() adalah kondisi ketika l == r. Pada kondisi ini, array hanya memiliki satu elemen (yaitu elemen pada indeks l), sehingga fungsi langsung mengembalikan nilai elemen tersebut (arr[l]).

5. Tarik Kesimpulan tentang cara kerja totalDC()

Fungsi totalDC() menggunakan pendekatan divide and conquer untuk menghitung jumlah elemen dalam array. Pertama, array dibagi menjadi dua bagian dengan mencari indeks tengah (mid). Fungsi ini kemudian dipanggil secara rekursif untuk menghitung jumlah elemen pada bagian kiri (dari indeks l hingga mid) dan bagian kanan (dari indeks mid + 1 hingga r). Proses ini berlanjut hingga mencapai base case, yaitu ketika hanya ada satu elemen dalam array (indeks l == r), yang langsung dikembalikan. Setelah itu, hasil penjumlahan dari bagian kiri dan kanan digabungkan dengan menambahkan lsum dan rsum untuk mendapatkan total jumlah elemen pada rentang indeks tersebut.