**LAPORAN HASIL PRAKTIKUM ALGORITMA DAN STRUKTUR DATA**

**JOBSHEET 5 BRUTE FORCE DAN DIVIDE CONQUER**



Faiva Puspa Sahara

244107020036

TI – 1E

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2025

**HASIL PERCOBAAN**

* 1. **Percobaan 1**

1. Buat class **Faktorial09** dan lengkapi dengan atribut dan method yang telah digambarkan di dalam diagram class

public class Faktorial09 {

public int nilai;

1. Tambahkan method faktorialBF()

public int faktorialBF(int n) {

int fakto = 1;

for(int i=1; i<=n; i++) {

fakto \*= i;

}

return fakto;

}

1. Tambahkan method faktorialDC()

public int faktorialDC(int n) {

if(n==1) {

return 1;

} else {

int fakto = n \* faktorialDC(n-1);

return fakto;

}

}

1. Run class Faktorial dengan membuat class baru MainFaktorial

import java.util.Scanner;

public class MainFaktorial09 {

public static void main(String[] args) {

1. Dalam fungsi main sediakan komunikasi dengan user untuk memasukkan nilai yang akan dicari faktorialnya

Scanner input = new Scanner(System.in);

System.out.print("Masukkan nilai: ");

int nilai = input.nextInt();

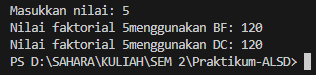
1. Tampilkan hasil pemanggilan method faktorialDC() dan faktorialBF()

Faktorial09 fk = new Faktorial09();

System.out.println("Nilai faktorial "+nilai+"menggunakan BF: " +fk.faktorialBF(nilai));

System.out.println("Nilai faktorial "+nilai+"menggunakan DC: " +fk.faktorialDC(nilai));

1. Run program



* **Pertanyaan 1**

1. Sebagai pemisahan alur eksekusi kode berdasarkan nilai faktorial sudah langsung diketahui (kondisi dasar) atau perlu dihitung dengan memecahnya menjadi nilai yang lebih kecil (kondisi rekursif).

* If

if (n == 0) {

return 1;

}

* Else

else {

return n \* factorial(n - 1);

}

1. Bisa, menggunakan perulangan While.

while (i > 0) {

hasil \*= i;

i--;

}

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

* **Fakto \*= i** adalah operasi dalam loop untuk mengakumulasi hasil perkalian dalam variabel fakto selama iterasi loop.
* **int fakto = n \* faktorialDC(n-1)** adalah deklarasi dan inisialisasi variabel fakto dengan hasil dari operasi rekursif, menggabungkan hasil sebelumnya dengan nilai saat ini.

1. Perbedaan cara kerja method faktorialBF() dan faktorialDC()!

* **faktorialBF =** Menggunakan perulangan untuk menghitung nilai faktorial dari sebuah bilangan.
* **faktorialDC =** Menggunakan rekursi untuk menghitung nilai faktorial dari sebuah bilangan.
  1. **Percobaan 2**

1. Buatlah class baru **Pangkat09.java** dan buatlah atribut angka yang akan dipangkatkan sekaligus dengan angka pemangkatnya.

public class Pangkat09 {

int nilai, pangkat;

1. Tambahkan konstruktor berparameter

public Pangkat09(int nilai, int pangkat) {

this.nilai = nilai;

this.pangkat = pangkat;

}

1. Tambahkan method **PangkatBF( )**

public int pangkatBF(int a, int n) {

int hasil = 1;

for(int i=0; i<n; i++) {

hasil = hasil\*a;

}

return hasil;

}

1. 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));

}

}

}

1. Buatlah class baru **MainPangkat09.java.** Tambahkan kode untuk menginput jumlah elemen yang akan dihitung pangkatnya

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();

1. Tambahkan proses pengisian beberapa nilai yang akan dipangkatkan sekaligus dengan pemangkatnya

Pangkat09[] png = new Pangkat09[elemen];

for (int i=0; i < elemen; i++) {

System.out.print("Masukkan nilai basis element ke-"+(i + 1)+": ");

int basis = input.nextInt();

System.out.print("Masukkan nilai pangkat element ke-"+(i + 1)+": ");

int pangkat = input.nextInt();

png[i] = new Pangkat09(basis, pangkat);

}

1. panggil hasil nya dengan mengeluarkan return value dari method **PangkatBF( )** dan **PangkatDC( )**

System.out.println("HASIL PANGKAT BRUTEFORCE:");

for (Pangkat09 p : png) {

System.out.println(p.nilai+"^"+p.pangkat+": "+p.pangkatBF(p.nilai, p.pangkat));

}

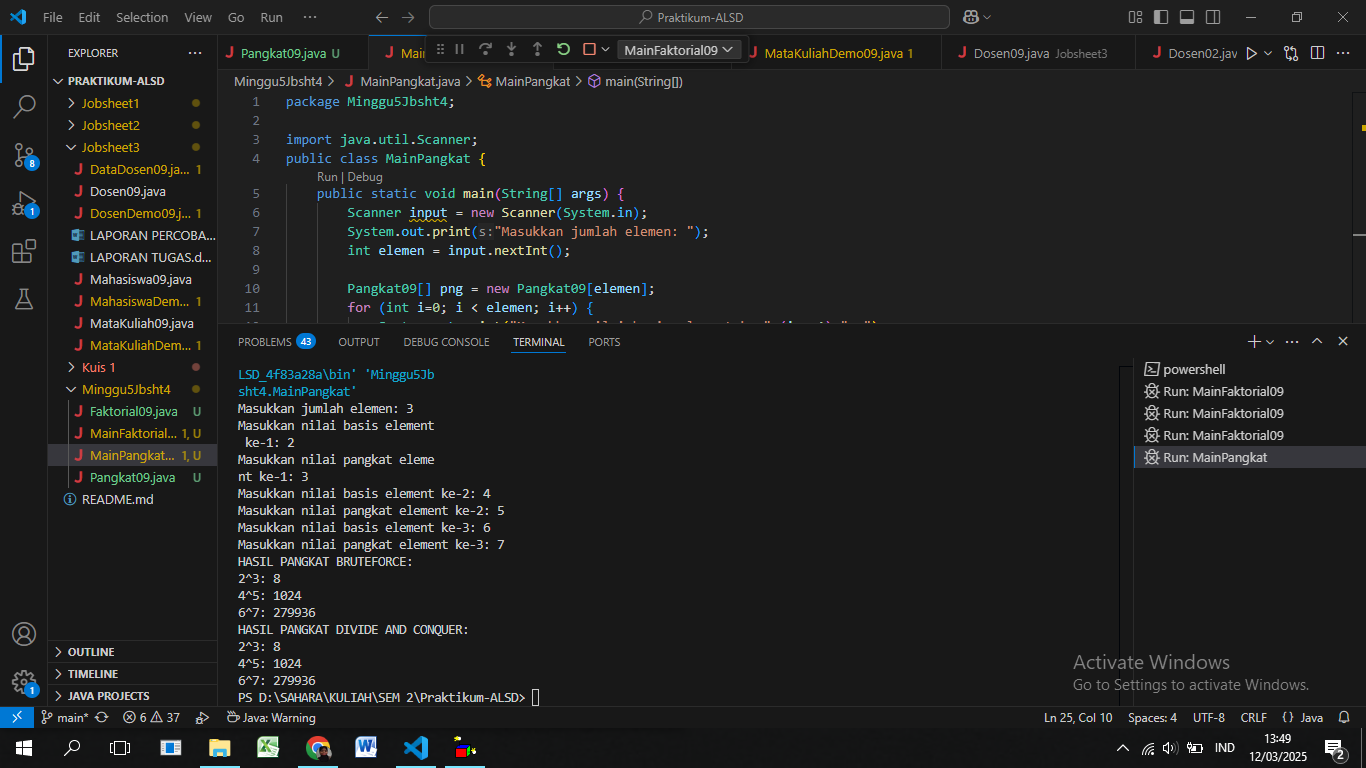
System.out.println("HASIL PANGKAT DIVIDE AND CONQUER:");

for (Pangkat09 p : png) {

System.out.println(p.nilai+"^"+p.pangkat+": "+p.pangkatDC(p.nilai, p.pangkat));

}

1. Run program



* **Pertanyaan Percobaan 2**

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

* ***Metode rekursif*** **pangkatDC( )** dapat lebih efisien dalam beberapa situasi karena mengurangi jumlah perkalian yang diperlukan, namun ***metode iteratif* pangkatBF()** cenderung lebih sederhana dan stabil.
* **pangkatBF()** menggunakan pendekatan iteratif dengan loop, sedangkan **pangkatDC()** menggunakan pendekatan rekursif dengan membagi masalah menjadi sub-masalah yang lebih kecil.

1. Iya, tahap combine dalam metode **pangkatDC()** terjadi ketika hasil dari pemanggilan rekursif digabungkan untuk menghasilkan nilai pangkat akhir.

else {

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

}

1. **Pangkat09** sudah memiliki atribut nilai dan pangkat, maka metode **pangkatBF( )** dapat dibuat tanpa parameter dan menggunakan atribut. Ketika tidak memiliki parameter dan menggunakan atribut nilai dan pangkat yang ada di dalam objek kelas. Hal ini memungkinkan kode menjadi lebih ringkas dan mudah dibaca.
2. Tarik tentang cara kerja method pangkatBF() dan pangkatDC()!

* **pangkatBF() =** cara kerjanya menggunakan pendekatan iteratif (Brute Force) untuk menghitung nilai pangkat, dan memanfaatkan perulangan untuk mengalikan nilai berulang kali hingga mencapai pangkat yang diinginkan.
* **PangkatDC() =** cara kerjanya menggunakan pendekatan rekursif Divide and Conquer untuk menghitung nilai pangkat, dan memanggil dirinya sendiri dengan nilai eksponen yang lebih kecil dan menggabungkan hasil rekursif tersebut.
  1. **Percobaan 3**

1. Buatlah class baru **Sum09.java** dan tambahkan konstruktor

package Minggu5Jbsht4;

public class Sum09 {

double keuntungan[];

Sum09(int el){

keuntungan = new double[el];

}

1. Tambahkan method **TotalBF()** untuk menghitung total nilai array dengan iterative.

double totalBF(){

double total = 0;

for (int i = 0; i < keuntungan.length; i++) {

total = total+keuntungan[i];

}

return total;

}

1. 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;

}

1. Buat class MainSum09.java dalam method ini user dapat menuliskan berapa bulan keuntungan yang akan dihitung. Dalam kelas ini sekaligus dibuat instansiasi objek untuk memanggil atribut ataupun fungsi

package Minggu5Jbsht4;

import java.util.Scanner;

public class MainSum09 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Masukkan jumlah elemen: ");

int elemen = input.nextInt();

1. Menghitung total nilai keuntungan, ditambahkan pada method Main array yang akan dihitung.

Sum09 sum = new Sum09(elemen);

for (int i = 0; i < elemen; i++) {

System.out.print("Masukkan keuntungan ke-"+(i+1)+": ");

sum.keuntungan[i] = input.nextDouble();

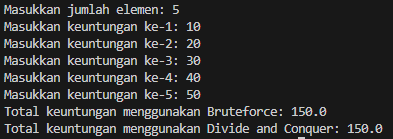
}

1. 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: "+sum.totalBF());

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

1. Run program



* **Pertanyaan Percobaan 3**

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

* Karena untuk membagi array menjadi dua bagian dalam pendekatan Divide and Conquer.

1. Dilakukan untuk memecah masalah menjadi dua sub-masalah yang lebih kecil dan kemudian menggabungkan hasil dari kedua sub-masalah.
2. Diperlukan untuk menggabungkan hasil dari sub-masalah sehingga kita mendapatkan total keuntungan keseluruhan dari array asli.
3. Base case penting untuk menghentikan rekursi dan memberikan hasil langsung ketika kondisi tertentu tercapai, yaitu ketika bagian array yang sedang diproses hanya terdiri dari satu elemen.
4. Tarik Kesimpulan tentang cara kerja totalDC()!

* Metode **totalDC()** menggunakan pendekatan rekursif Divide and Conquer untuk menghitung total keuntungan dari array. Masalah besar dipecah menjadi sub-masalah yang lebih kecil, kemudian hasil dari sub-masalah digabungkan untuk mendapatkan hasil akhir.