**LAPORAN PRAKTIKUM**

**ALGORITMA DAN STRUKTUR DATA**

**PERTEMUAN KE 5**

**BRUTE FORCE DAN DIVIDE CONQUER**

Dosen Pengampuh : Bu. Triana Fatmawati, S.T., M.T.



Muhammad Afiq Firdaus

2341760189 / SIB-1E

**PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI MALANG**

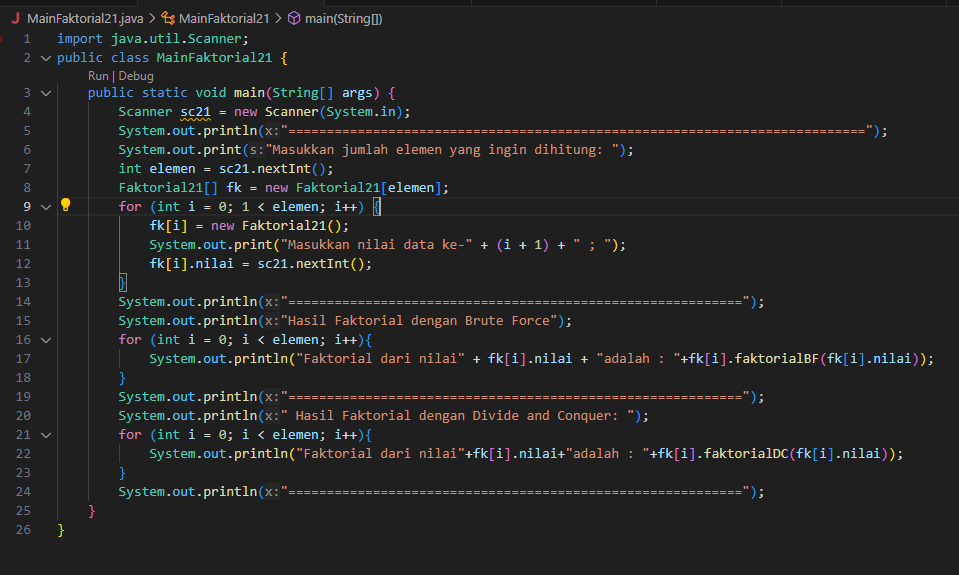
**2024**

**4.2 Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer**

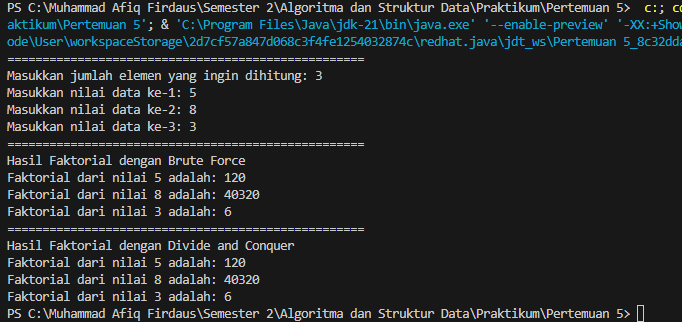
Berikut adalah hasil coding Faktorial berdasarkan jobsheet.



Berikut adalah code MainFaktorial berdasarkan pada jobsheet



Berikut adalah hasil run dari code diatas :



**Pertanyaan**

1. Jelaskan mengenai base line Algoritma Divide Conquer untuk melakukan pencarian nilai faktorial!

2. Pada implementasi Algoritma Divide and Conquer Faktorial apakah lengkap terdiri dari 3 tahapan

divide, conquer, combine? Jelaskan masing-masing bagiannya pada kode program!

3. Apakah memungkinkan perulangan pada method faktorialBF() dirubah selain menggunakan

for?Buktikan!

4. Tambahkan pegecekan waktu eksekusi kedua jenis method tersebut!

5. Buktikan dengan inputan elemen yang di atas 20 angka, apakah ada perbedaan waktu eksekusi?

**Jawaban :**

1. Algoritma Divide-and-Conquer untuk mencari nilai faktor membagi masalah

menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah secara

terpisah, dan menggabungkan solusi untuk mendapatkan solusi akhir.

Sebagai bagian dari pencarian faktorial, algoritma pertama-tama memeriksa

apakah nilai yang diinginkan adalah 1. Jika demikian, ia segera mengembalikan nilai

1. Jika tidak, algoritma membagi permasalahan menjadi permasalahan yang lebih

kecil dengan mengalikan nilai yangg ditemukan dengan faktorial dari nilai

sebelumnya. Proses ini diulangi hingga kasus dasar tercapai, dimana nilai yang

diinginkan adalah 1.

Hasil dari setiap langkah rekursi kemudian digabungkan untuk mendapatkan

nilai faktor akhir. Oleh karena itu, algoritma ini menggunakan pendekatan rekursif

untuk menyelesaikan masalah, menggabungkan solusi hingga sampai pada solusi

akhir.

2. Pada implementasi Algoritma Divide and Conquer Faktorial apakah lengkap

terdiri dari 3 tahapan divide, conquer, combine? Jelaskan masing-masing

bagiannya pada kode program!

Pada percobaan 1 hanya terdapat 2 tahapan yaitu Divide dan Conquer karena tahapan

Combine tidak selalu eksplisit dalam implementasi faktorial menggunakan algoritma

Divide and Conquer.

* Devide

Terjadi ketika pada fungsi faktorialDC(int n), kita membagi masalah menjadi

masalah yang lebih kecil dengan mengurangi nilai faktorial yang dicari (n) satu per

satu hingga mencapai kasus dasar (n==1).

* Conquer

Terjadi di dalam blok else pada fungsi faktorialDC(int n). Pada tahap ini, kita

menyelesaikan setiap submasalah secara rekursif dengan mengalikan nilai faktorial

yang sedang dicari (n) dengan nilai faktorial dari masalah yang lebih kecil

(faktorialDC(n-1)). Setiap langkah rekursif ini akan terus berulang sampai mencapai

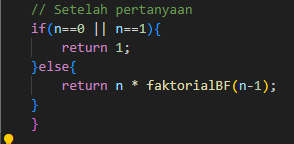
kasus dasar.

3. Apakah memungkinkan perulangan pada method faktorialBF() dirubah selain

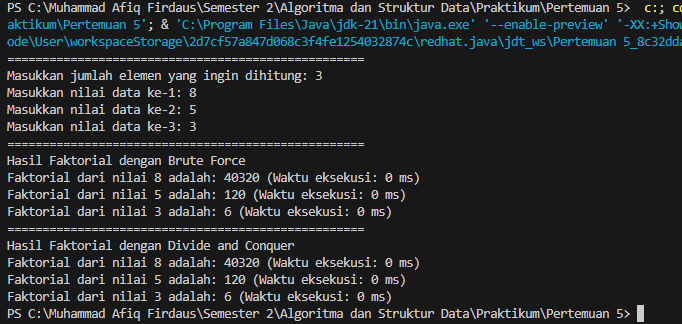
menggunakan for?Buktikan!

Perulangan pada method faktorialBF() dapat diubah menjadi rekursif dengan

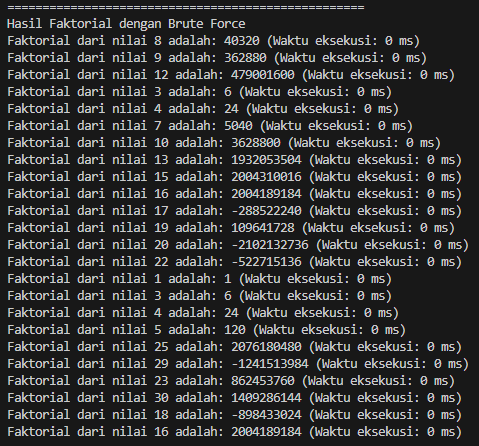
menggunakan if else seperti berikut :

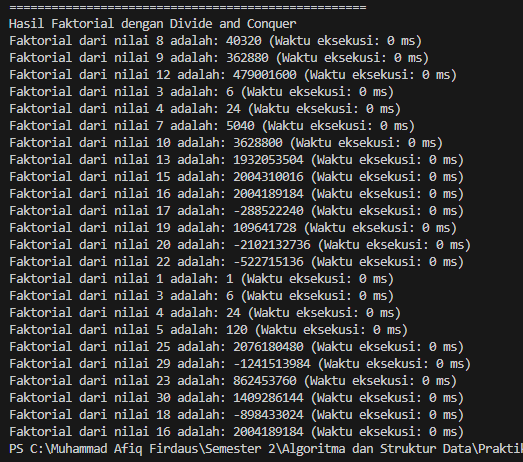


4. Tambahkan pegecekan waktu eksekusi kedua jenis method tersebut!



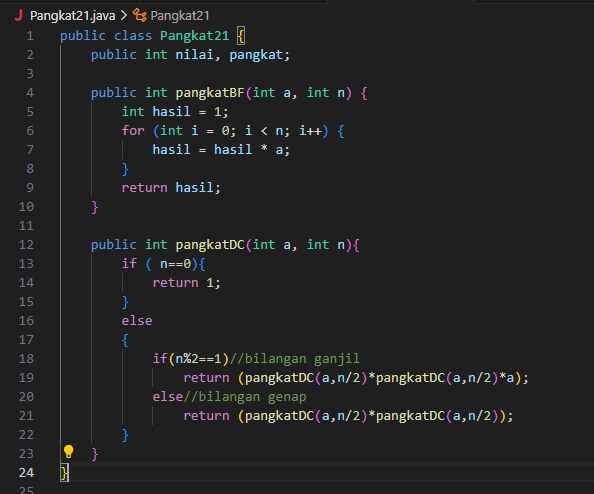
5. Buktikan dengan inputan elemen yang di atas 20 angka, apakah ada perbedaan waktu eksekusi?





**Percobaan 2**

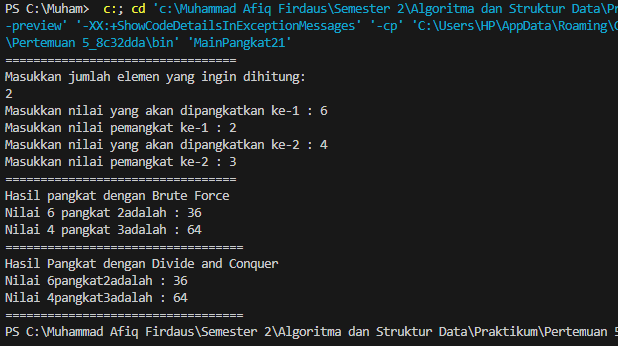
Berikut adalah code Pangkat21 berdasarkan pada jobsheet



Berikut adalah MainPangkat berdasarkan pada Jobsheet



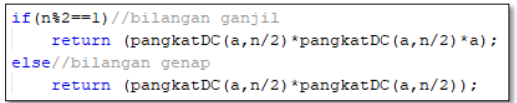
Verifikasi hasil percobaan



**Pertanyaan**

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu PangkatBF() dan PangkatDC()!

2. Pada method PangkatDC() terdapat potongan program sebagai berikut:



Jelaskan arti potongan kode tersebut

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

4. Modifikasi kode program tersebut, anggap proses pengisian atribut dilakukan dengan

konstruktor.

5. Tambahkan menu agar salah satu method yang terpilih saja yang akan dijalankan!

**Jawaban :**

1.

* PangkatBF()

a. Menggunakan iterasi sederhana dengan looping for untuk mengalikan

bilangan a sebanyak n kali

b. Algoritma ini efektif untuk nilai n yang relatif kecil, tetapi memiliki

kompleksitas waktu O(n) karena membutuhkan waktu linear sesuai dengan

nilai eksponen n

* PangkatDC()

a. Membagi masalah menjadi submasalah yang lebih kecil dan

menyelesaikannya secara rekursif.

b. Algoritma ini memanfaatkan sifat matematis bahwa a^n dapat dipecah

menjadi a^(n/2) \* a^(n/2) untuk eksponen n genap dan (a^(n/2) \* a^(n/2) \* a)

untuk eksponen n ganjil.

c. Algoritma ini memiliki kompleksitas waktu yang lebih baik daripada Brute

Force, yaitu O(log n), karena melakukan pembagian masalah secara rekursif

menjadi submasalah yang lebih kecil

2.

* if (n % 2 == 1) { // Bilangan Ganjil

Pernyataan ini memeriksa apakah eksponen n adalah bilangan ganjil. Jika sisa

pembagian n dengan 2 sama dengan 1, itu berarti n adalah bilangan ganjil.

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

Jika eksponen n adalah bilangan ganjil, maka masalah dibagi menjadi dua

bagian: a^(n/2) dan a^(n/2). Kedua submasalah ini diselesaikan secara rekursif

menggunakan metode pangkatDC(). Kemudian, hasilnya dikalikan dengan a, karena

eksponen n ganjil menambahkan satu faktor a tambahan.

* else { // Bilangan Genap

Ini adalah bagian dari struktur if-else yang menangani kasus ketika eksponen n

adalah bilangan genap

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

Jika eksponen n adalah bilangan genap, maka masalah dibagi menjadi dua

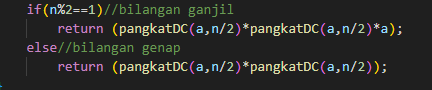
bagian yang sama: a^(n/2) dan a^(n/2). Kedua submasalah ini diselesaikan secara

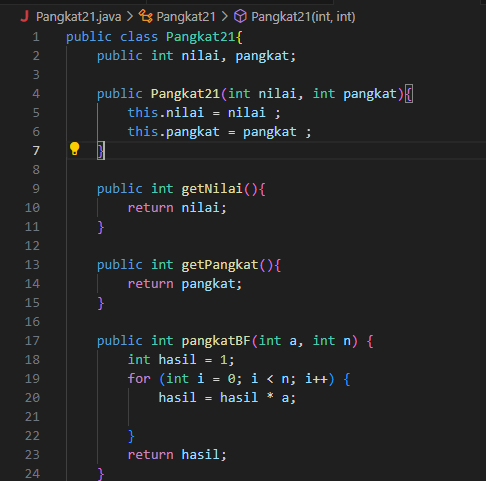
rekursif menggunakan metode pangkatDC(), dan hasilnya dikalikan bersama.

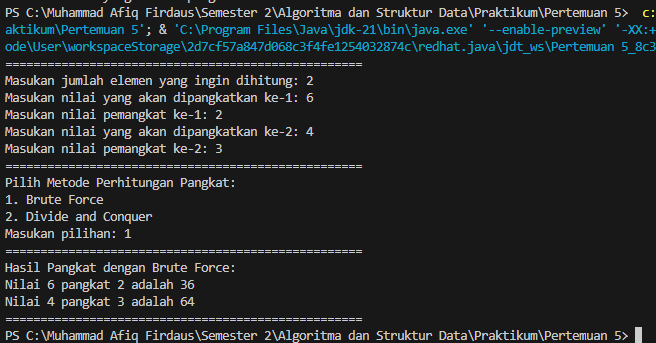
Dengan cara ini, pendekatan Divide and Conquer secara rekursif memecah

masalah menjadi submasalah yang lebih kecil dan menyelesaikannya secara efisien.

3.

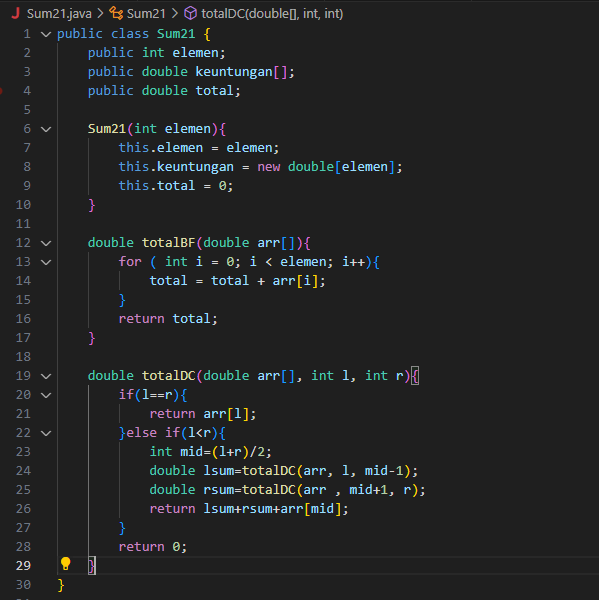


4. 

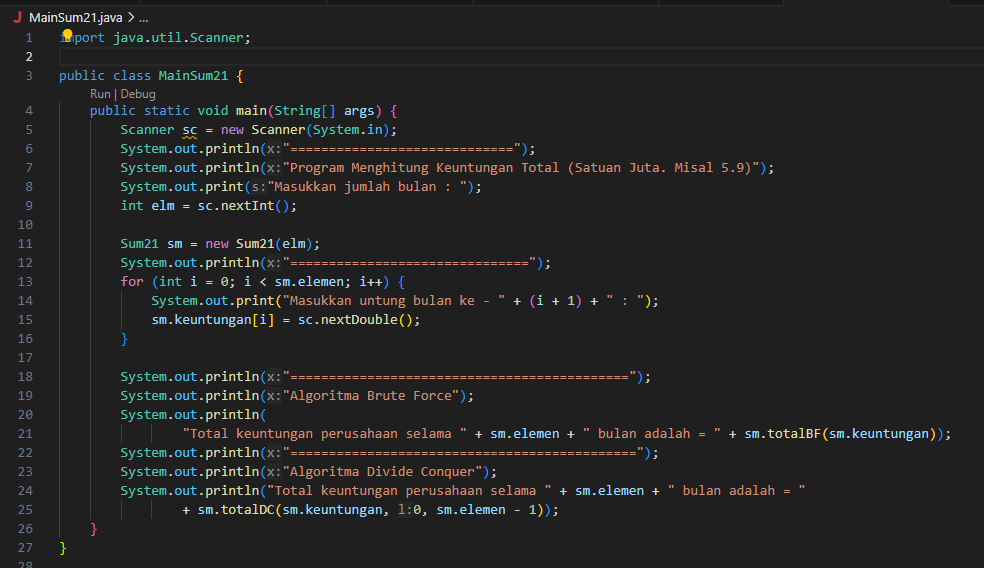
5. 

**Percobaan 3**

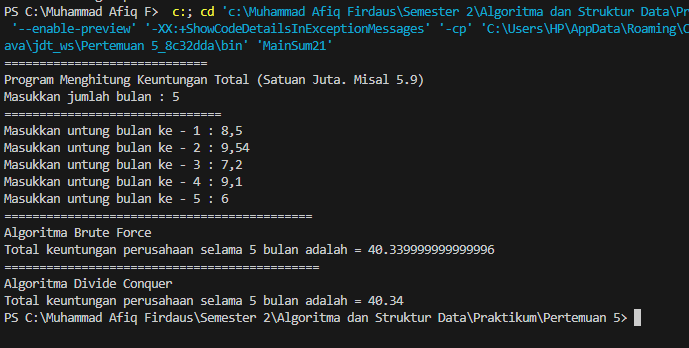
Berikut adalah code Sum berdasarkan pada Jobsheet



Berikut adalah code MainSum berdasarkan pada Jobsheet



Berikut adalah verifikasi hasil percobaan



**Pertanyaan**

1. Berikan ilustrasi perbedaan perhitungan keuntungan dengan method TotalBF() ataupun

TotalDC()

2. Perhatikan output dari kedua jenis algoritma tersebut bisa jadi memiliki hasil berbeda di

belakang koma. Bagaimana membatasi output di belakang koma agar menjadi standar untuk

kedua jenis algoritma tersebut.

3. Mengapa terdapat formulasi *return value* berikut?Jelaskan!



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

5. Program perhitungan keuntungan suatu perusahaan ini hanya untuk satu perusahaan saja.

Bagaimana cara menghitung sekaligus keuntungan beberapa bulan untuk beberapa

perusahaan.(Setiap perusahaan bisa saja memiliki jumlah bulan berbeda-beda)? Buktikan

dengan program!

**Jawaban**

1.

* totalBF()

a. Pada metode ini akan menelusuri seluruh elemen dalam array keuntungan

b. Setiap elemen ditambahkan ke total secara berurutan dengan kompleksitas

waktu O(n), di mana n adalah jumlah elemen dalam array

* totalDC()

a. Metode ini membagi masalah menjadi submasalah yang lebih kecil,

menyelesaikan submasalah tersebut secara rekursif, dan menggabungkan

hasilnya

b. Ketika mencapai kasus dasar (basis), yaitu ketika kisaran (range) dari l dan r

hanya satu elemen, maka nilai dari elemen tersebut dikembalikan

c. Jika kisaran lebih dari satu elemen, kisaran dibagi menjadi dua bagian pada

tengahnya (mid), dan dua submasalah dipecahkan secara rekursif untuk

setiap bagian kiri dan kanan

d. Kemudian, total dari kiri, kanan, dan elemen tengah dihitung dan

digabungkan untuk memberikan total keseluruhan

e. Proses ini memiliki kompleksitas waktu O(n log n) karena membagi masalah

menjadi dua setengah pada setiap tingkat rekursi.

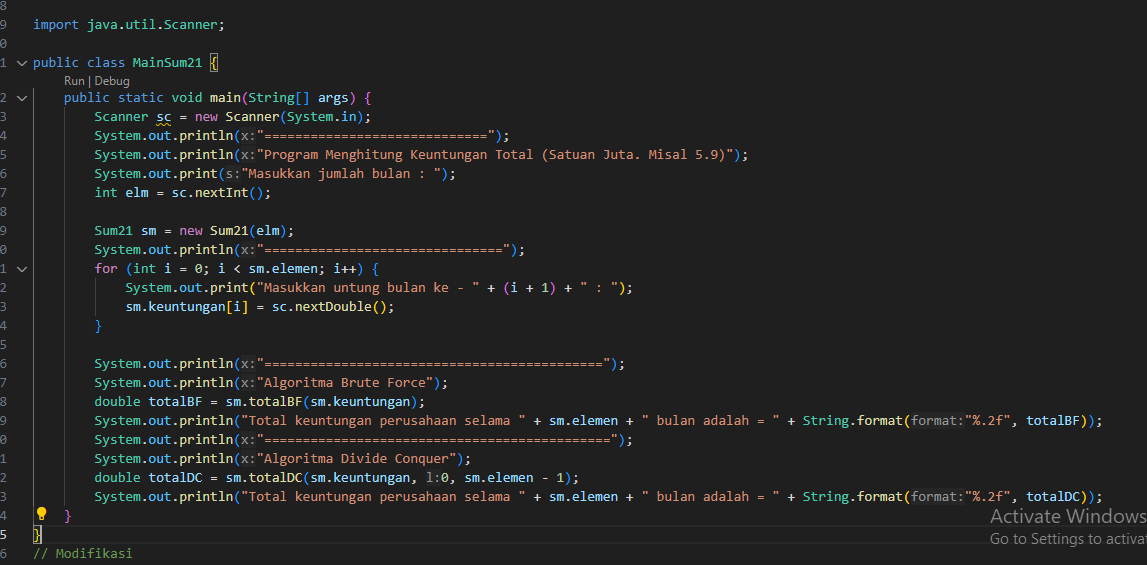
Jadi, TotalBF() sederhana dan langsung menambahkan keuntungan secara

berurutan, sementara TotalDC() membagi masalah menjadi submasalah lebih kecil,

menyelesaikannya secara rekursif, dan kemudian menggabungkan hasilnya untuk

mendapatkan total keseluruhan

2.



3.

* lsum

Ini adalah hasil dari pemanggilan rekursif totalDC() pada setengah bagian kiri

array (dari indeks l hingga mid - 1). Metode ini mengembalikan total keuntungan dari

setengah bagian kiri

* rsum

Ini adalah hasil dari pemanggilan rekursif totalDC() pada setengah bagian

kanan array (dari indeks mid + 1 hingga r). Metode ini mengembalikan total

keuntungan dari setengah bagian kanan

* arr[mid]

Ini adalah nilai keuntungan pada indeks mid, yang merupakan nilai

keuntungan tengah dari array.

Jadi, return lsum + rsum + arr[mid]; menggabungkan total keuntungan dari

setengah bagian kiri, setengah bagian kanan, dan nilai keuntungan tengah, yang

merupakan total keseluruhan keuntungan dari seluruh array. Ini adalah langkah

combine dalam algoritma Divide and Conquer, di mana hasil dari dua submasalah

yang lebih kecil digabungkan bersama-sama untuk membentuk solusi untuk masalah

yang lebih besar.

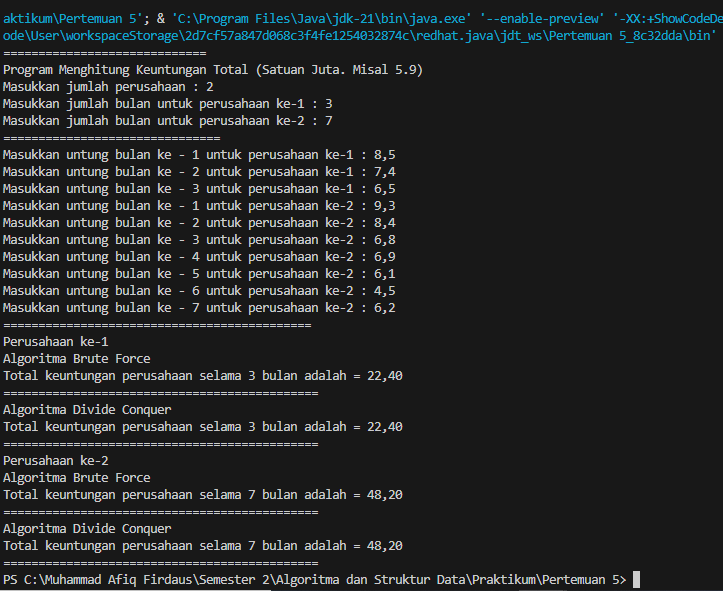
4. Variabel mid dibutuhkan dalam metode totalDC() karena itu merupakan

indeks yang menandai titik tengah dari array yang sedang diproses. Dalam pendekatan

Divide and Conquer, array dibagi menjadi dua bagian setiap kali metode rekursif

dipanggil. Variabel mid digunakan untuk menandai pembagian tersebut

5.



**4.5 Latihan Praktikum**

Buatlah kode program untuk menghitung nilai akar dari suatu bilangan dengan algoritma Brute

Force dan Divide Conquer! *Jika bilangan tersebut bukan merupakan kuadrat sempurna, bulatkan*

*angka ke bawah.*

