**PERTEMUAN 5**

**JOBSHEET 4 – BRUTE FORCE and DEVIDE CONQUER**

Untuk Memenuhi Salah Satu Tugas

Mata Kuliah Praktikum Algoritma Struktur Dasar

Dosen: Ibu Rokhimatul Wakhidah, S.Pd., M.T.

****

Disusun oleh: Revani Nanda Putri (NIM: 2341760056)

**PROGRAM STUDI SISTEM INFORMASI BISNIS**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI MALANG**

**2024**

JOBSHEET IV  
BRUTE FORCE DAN DIVIDE CONQUER

1. Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

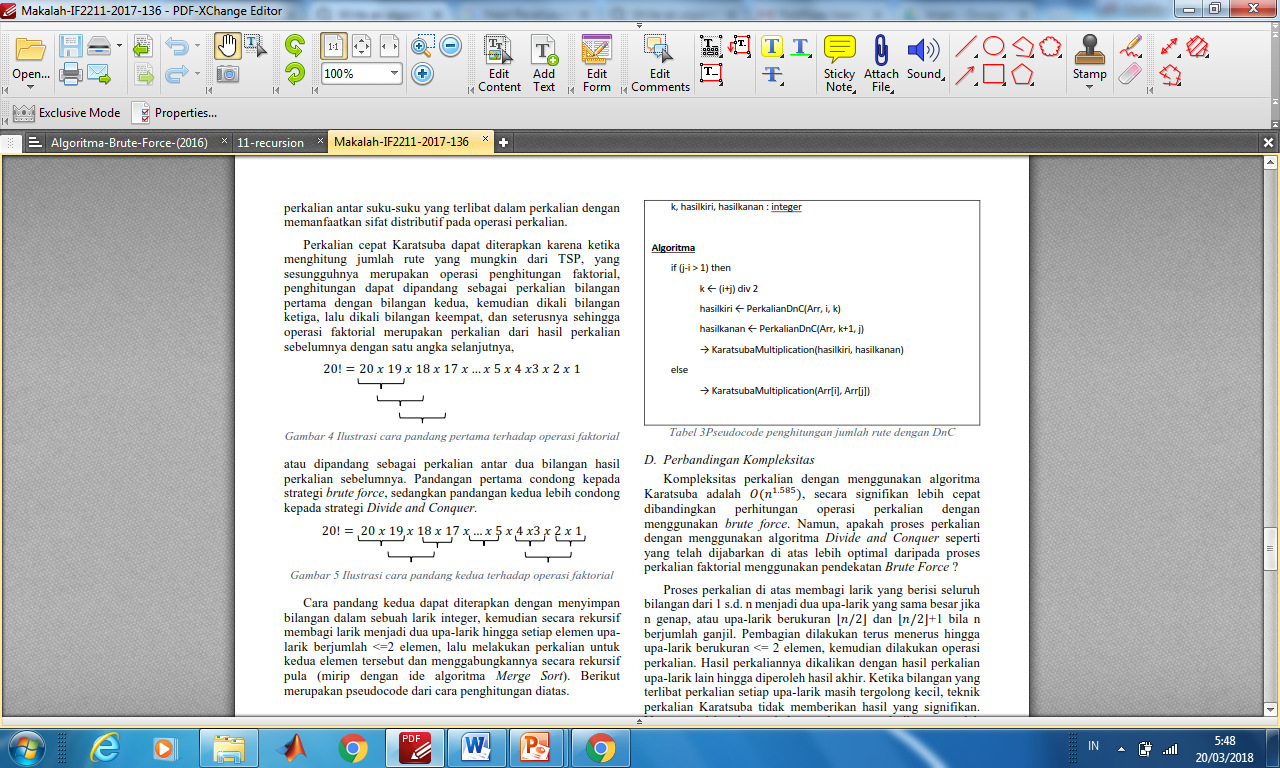
1. Mahasiswa mampu membuat algoritma bruteforce dan divide-conquer
2. Mahasiswa mampu menerapkan penggunaan algorima bruteforce dan divide-conquer
3. Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer

Perhatikan Diagram Class berikut ini :

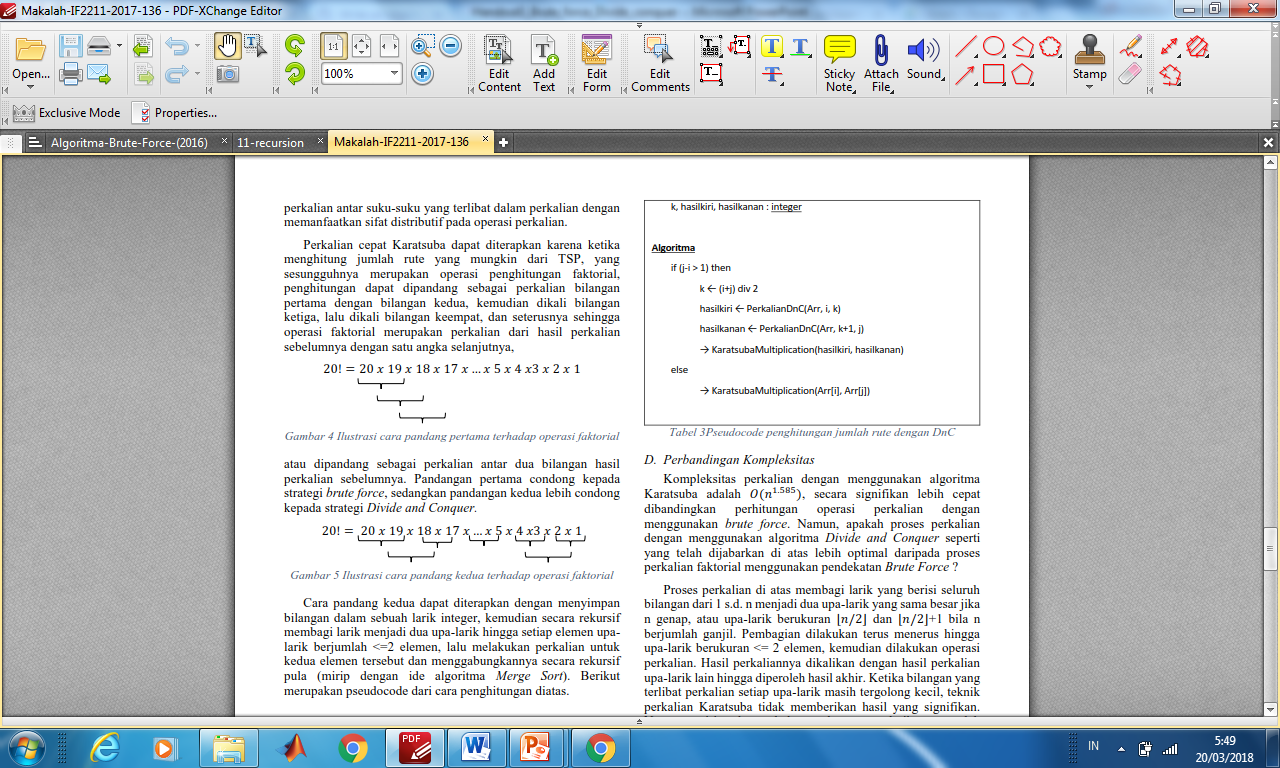
|  |
| --- |
| Faktorial |
| nilai: int |
| faktorialBF(): int  faktorialDC(): int |

Berdasarkan diagram class di atas, akan dibuat program class dalam Java. Untuk menghitung nilai faktorial suatu angka menggunakan 2 jenis algoritma, Brute Force dan Divide and Conquer. Jika digambarkan terdapat perbedaan proses perhitungan 2 jenis algoritma tersebut sebagai berikut :

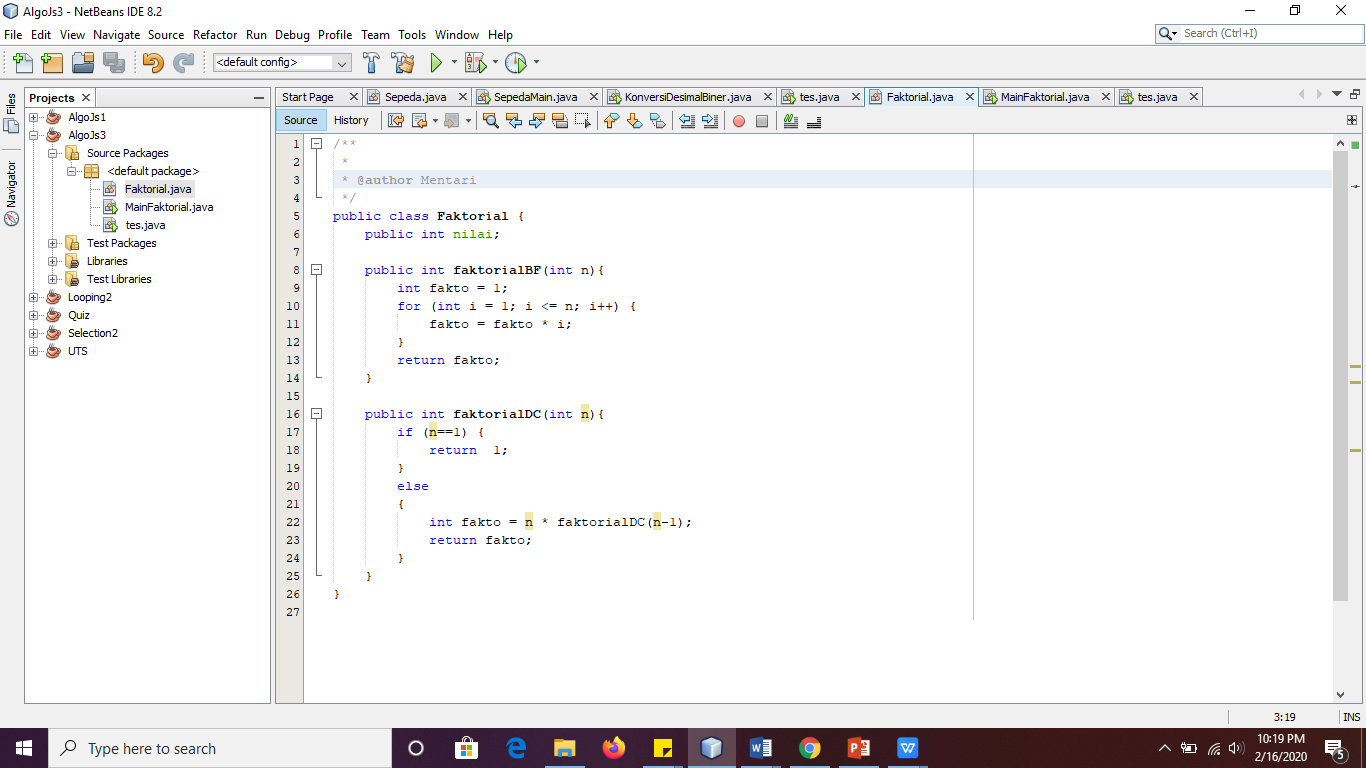
Tahapan pencarian nilai faktorial dengan algoritma Brute Force :



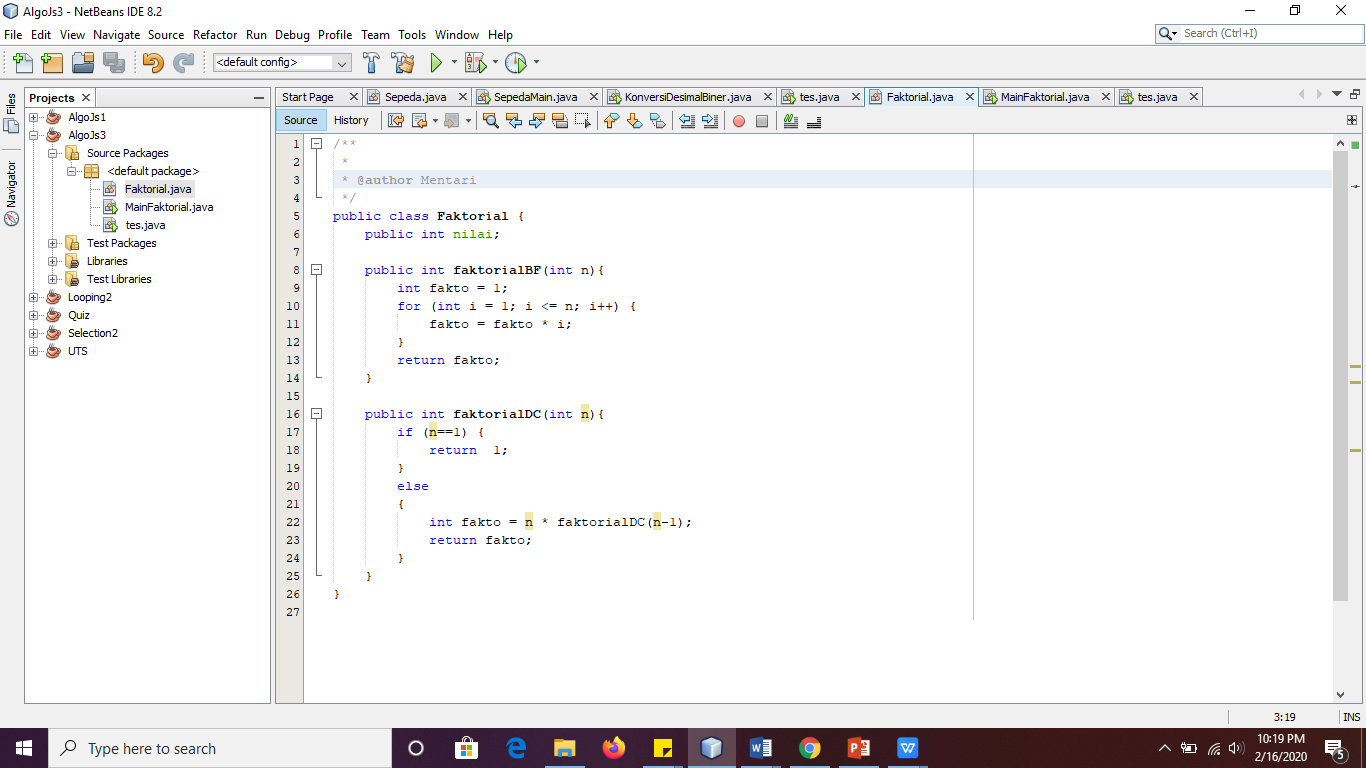
Tahapan pencarian nilai faktorial dengan algoritma Divide and Conquer :



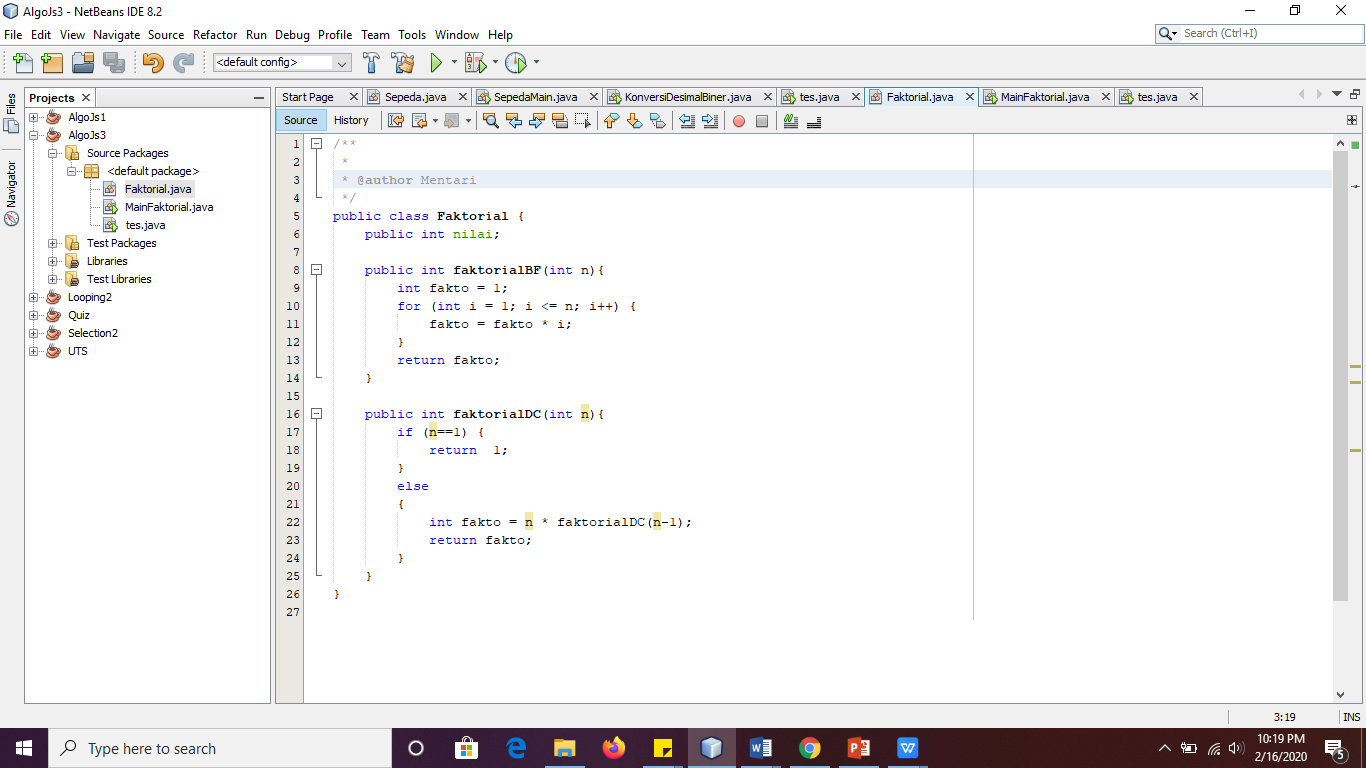
1. Langkah-langkah Percobaan
2. Buat Project baru, dengannama “**BruteForceDivideConquer**”. Buat package dengan nama minggu5.
3. Buatlah class baru dengan nama **Faktorial**
4. Lengkapi class **Faktorial** dengan atribut dan method yang telah digambarkan di dalam diagram class di atas, sebagai berikut:
5. Tambahkan atribut nilai



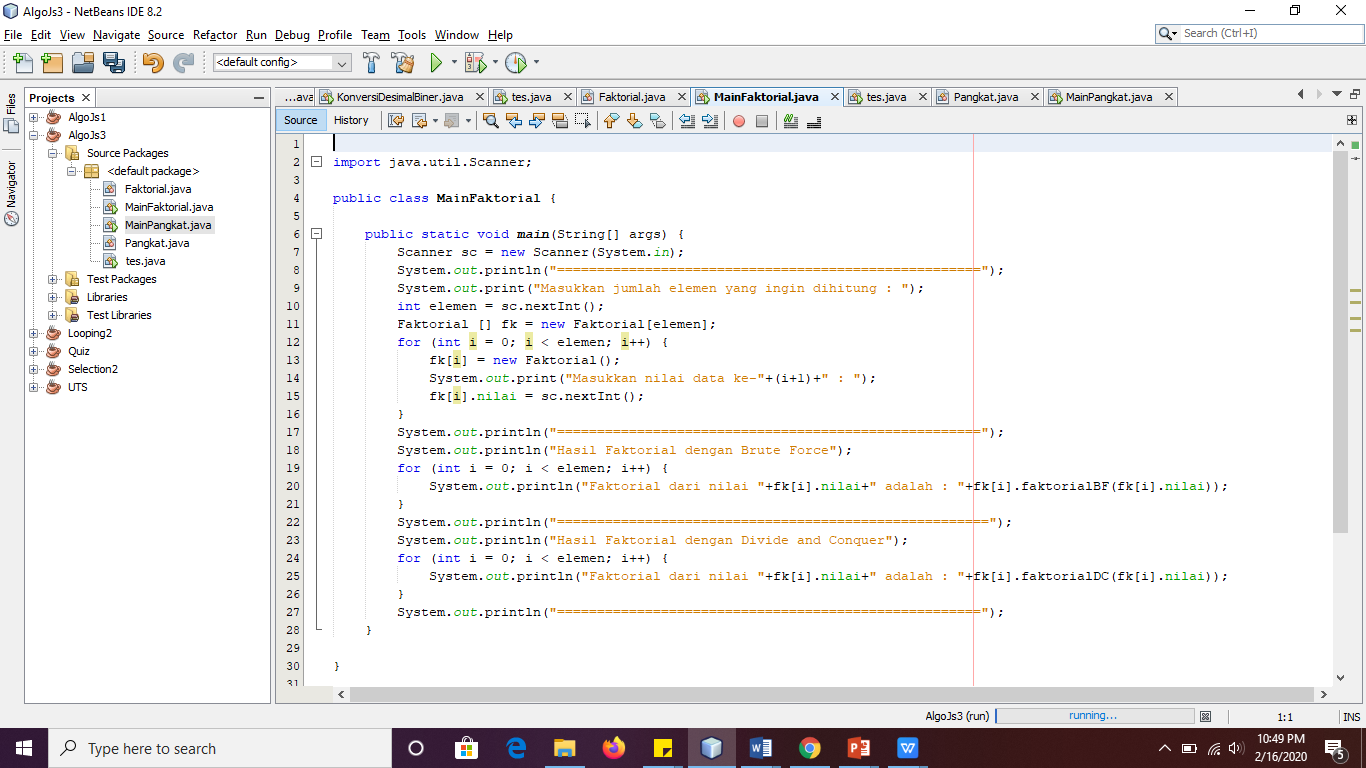
1. Tambahkan method faktorialBF() nilai



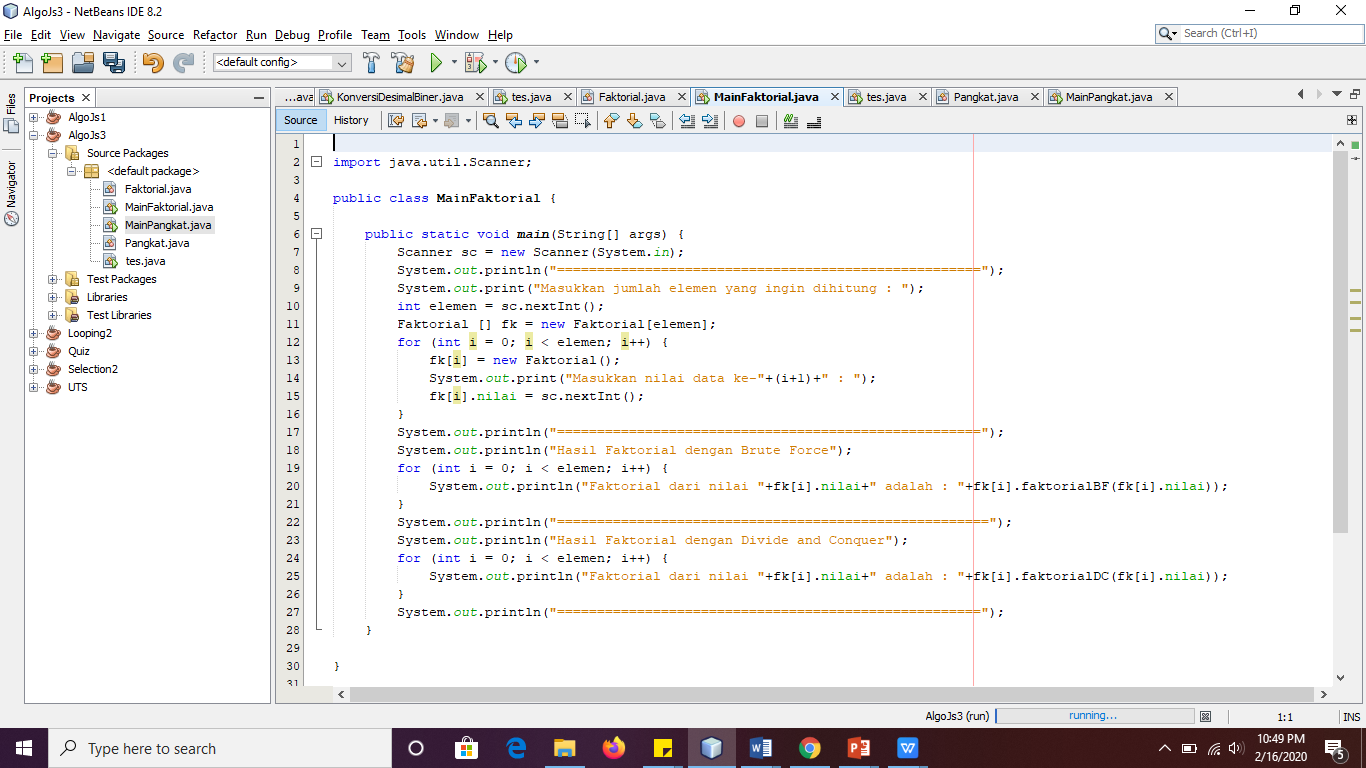
1. Tambahkan method faktorialDC() nilai



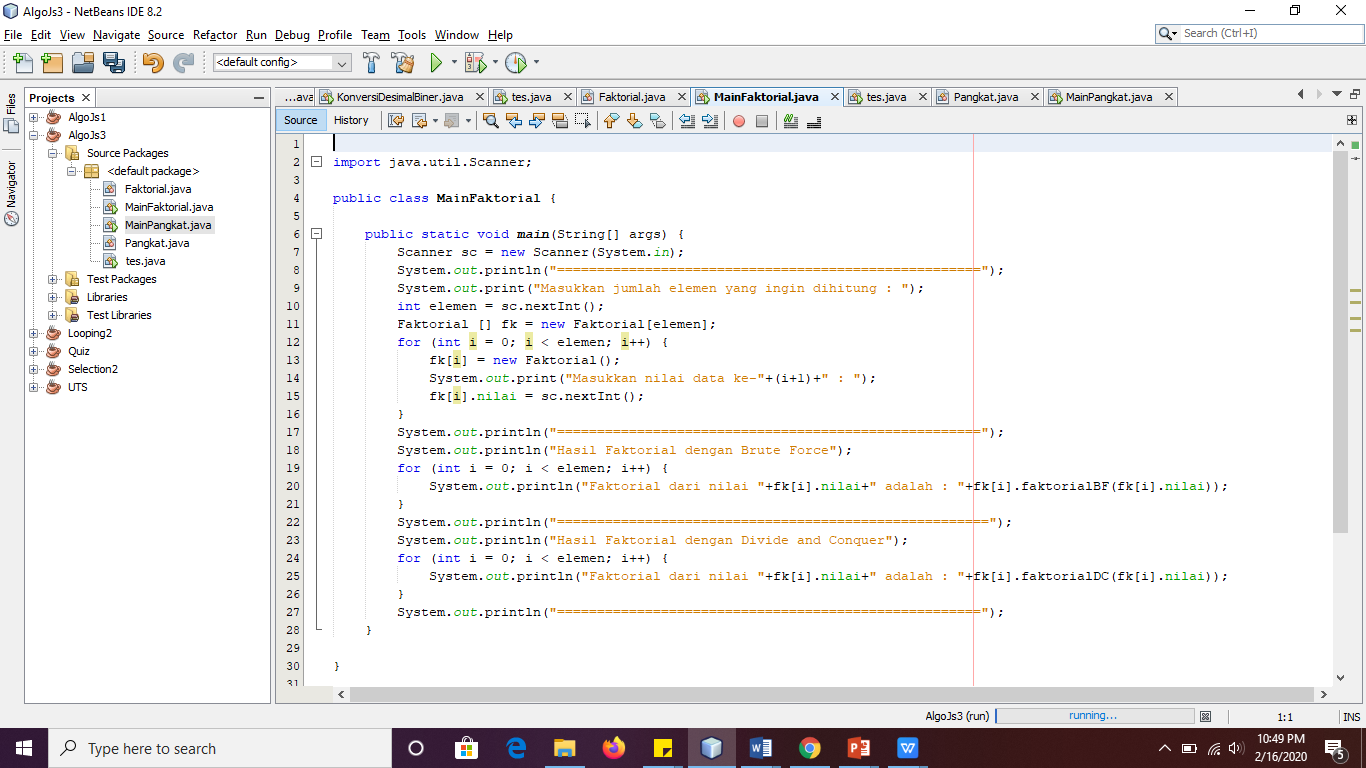
1. Coba jalankan (Run) class Faktorial dengan membuat class baru MainFaktorial.
2. Di dalam fungsi main sediakan komunikasi dengan user untuk menginputkan jumlah angka yang akan dicari nilai faktorialnya



* 1. Buat Array of Objek pada fungsi main, kemudian inputkan beberapa nilai yang akan dihitung faktorialnya



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



* 1. Pastikan program sudah berjalan dengan baik!

1. Verifikasi Hasil Percobaan

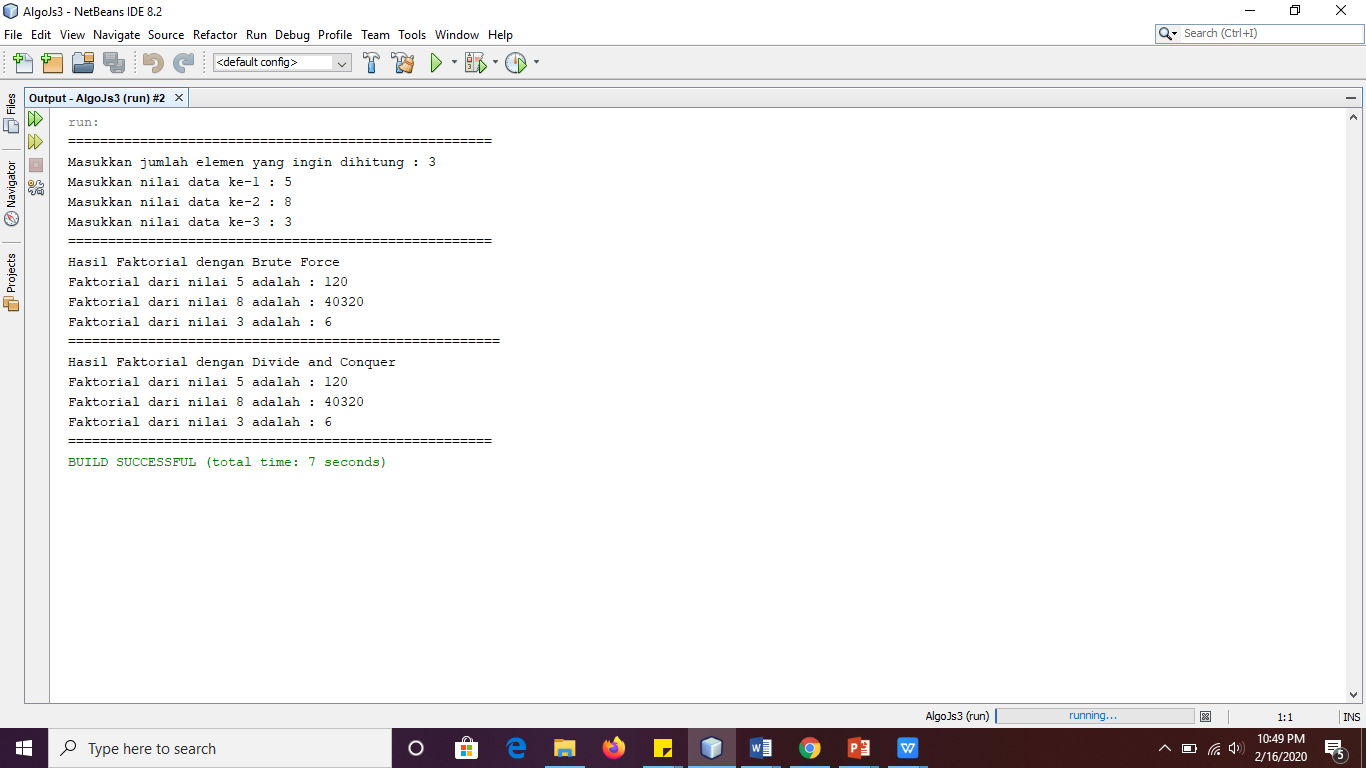
File Class Faktorial.java

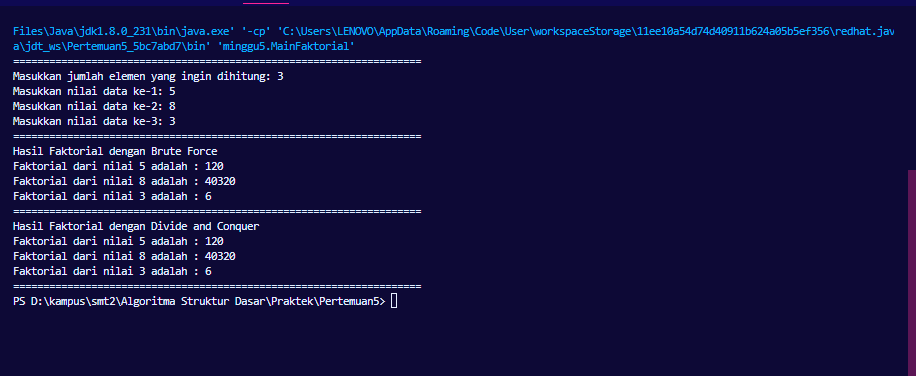


File MainFaktorial.java



Cocokkan hasil compile kode program anda dengan gambar berikut ini.

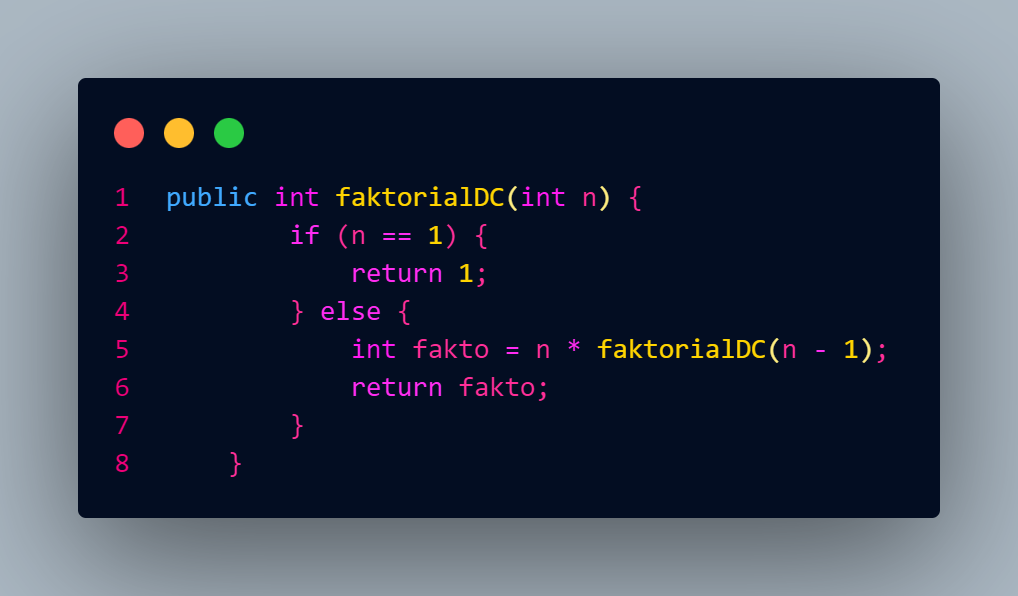




1. Pertanyaan
2. Jelaskan mengenai base line Algoritma Divide Conquer untuk melakukan pencarian nilai faktorial!

* Algoritma divide conquer

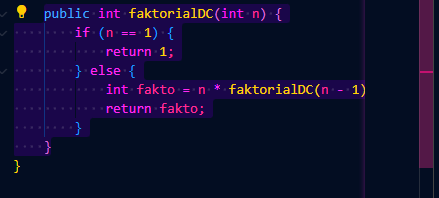
ketika nilai n sama dengan 1. Faktorial 1 (1!) didefinisikan sebagai 1, sama seperti faktorial 0.



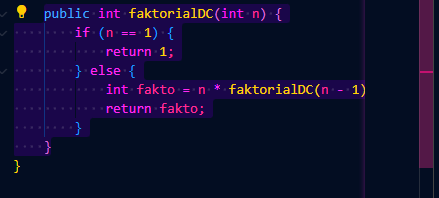
1. Pada implementasi Algoritma Divide and Conquer Faktorial apakah lengkap terdiri dari 3 tahapan divide, conquer, combine? Jelaskan masing-masing bagiannya pada kode program!

Jawab:

Tahapan devide and conquer tersebut terletak pada blok kode program berikut :



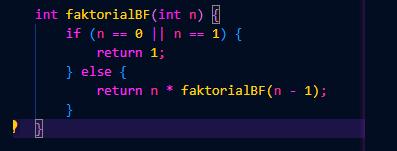
* Divide : Terjadi dalam faktorialDC(int n). Pada bagian ini, masalah besar (mencari faktorial dari n) dibagi menjadi submasalah yang lebih kecil dengan mengurangi nilai n satu per satu hingga mencapai kasus dasar ketika n sama dengan 1. Setiap panggilan rekursif pada faktorialDC(n - 1) mewakili proses pembagian ini.
* Conquer : Tahap ini ada pada blok else statement, karena ketika nilai n tidak sama dengan 1, hal tersebut merupakan tahap di mana submasalah diselesaikan secara rekursif.
* Combine : Dalam kode program tersebut, combine atau penggabungan solusi dari banyak sub-masalah terletak pada baris berikut:



Hasil dari pemanggilan rekursif faktorialDC(n - 1) (solusi submasalah yang lebih kecil) digunakan untuk mengalikan nilai n, sehingga menghasilkan solusi untuk masalah aslinya. Kemudian solusi submasalah digabungkan untuk membentuk solusi untuk masalah aslinya

1. Apakah memungkinkan perulangan pada method faktorialBF() dirubah selain menggunakan for?Buktikan!

Memungkinkan

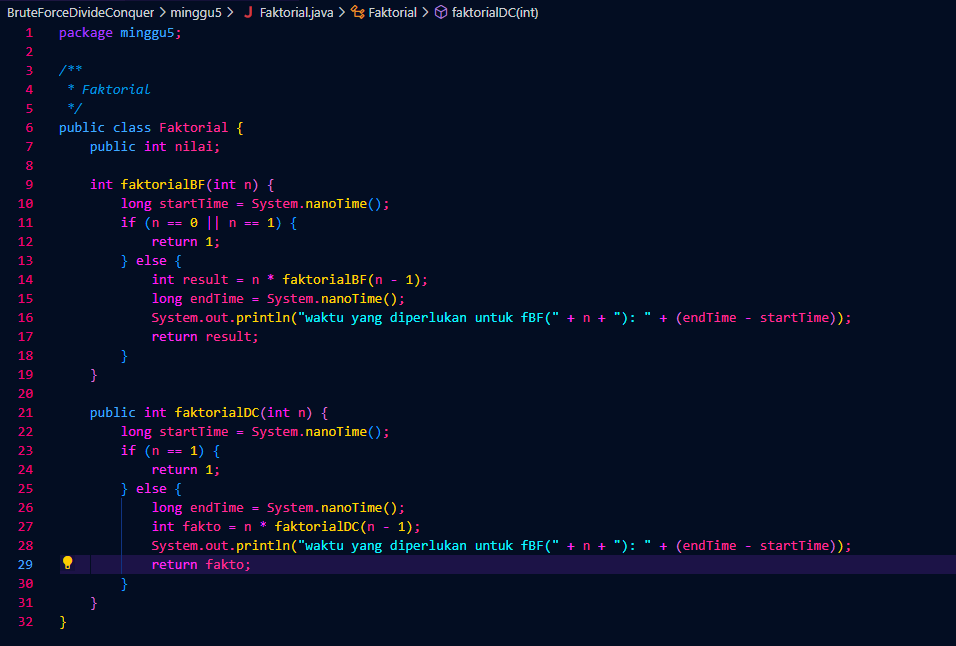


faktorialBF() diimplementasikan dengan menggunakan rekursi. Jika nilai n adalah 0 atau 1, maka fungsi akan mengembalikan 1 sebagai nilai faktorialnya. Jika n lebih besar dari 1, maka fungsi akan memanggil dirinya sendiri dengan parameter n - 1 dan mengalikan hasilnya dengan n. Implementasi ini menghasilkan hasil yang sama dengan implementasi menggunakan perulangan for, namun dengan pendekatan yang berbeda

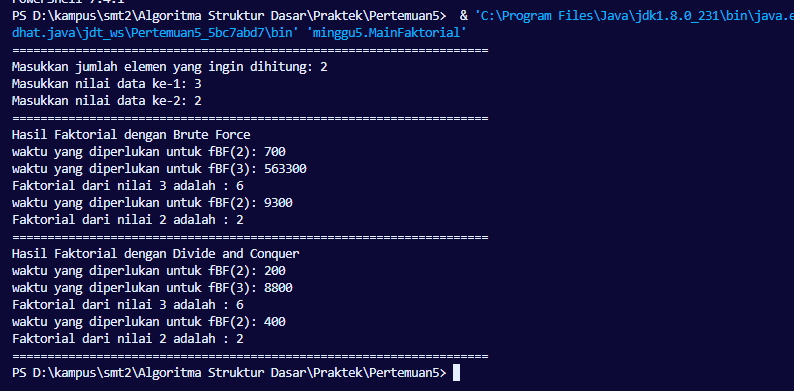
1. Tambahkan pegecekan waktu eksekusi kedua jenis method tersebut!

Untuk menambahkan pengecekan waktu eksekusi pada kedua jenis method, kita dapat

menggunakan System.nanoTime(), sebelum dan sesudah pemanggilan method faktorisasi. Program tersebut berfungsi untuk mencatat waktu awal sebelum pemanggilan rekursif dan waktu akhir setelah rekursi selesai. Selisih kedua waktu tersebut memberikan waktu yang diperlukan untuk menjalankan method faktorisasi dalam satuan nanodetik.



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

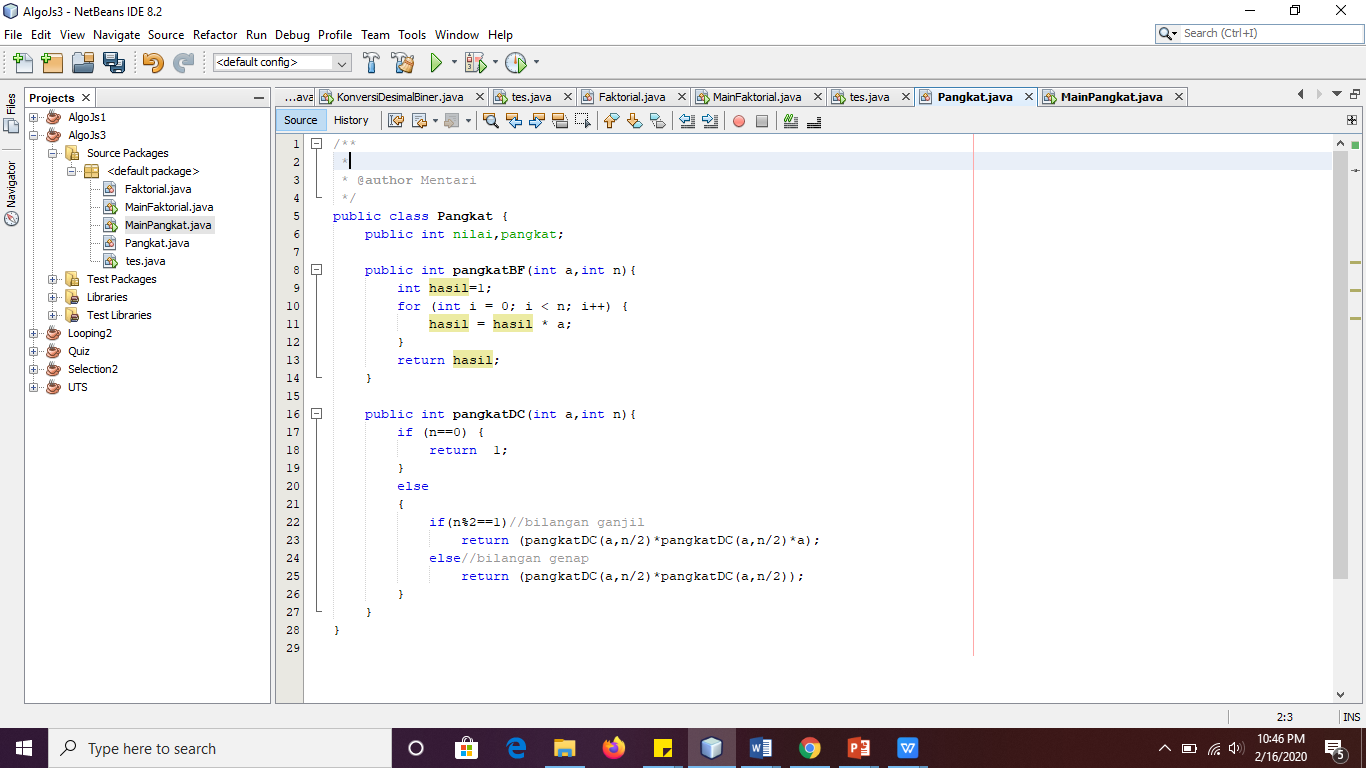


1. Menghitung Hasil Pangkat dengan Algoritma Brute Force dan Divide and Conquer

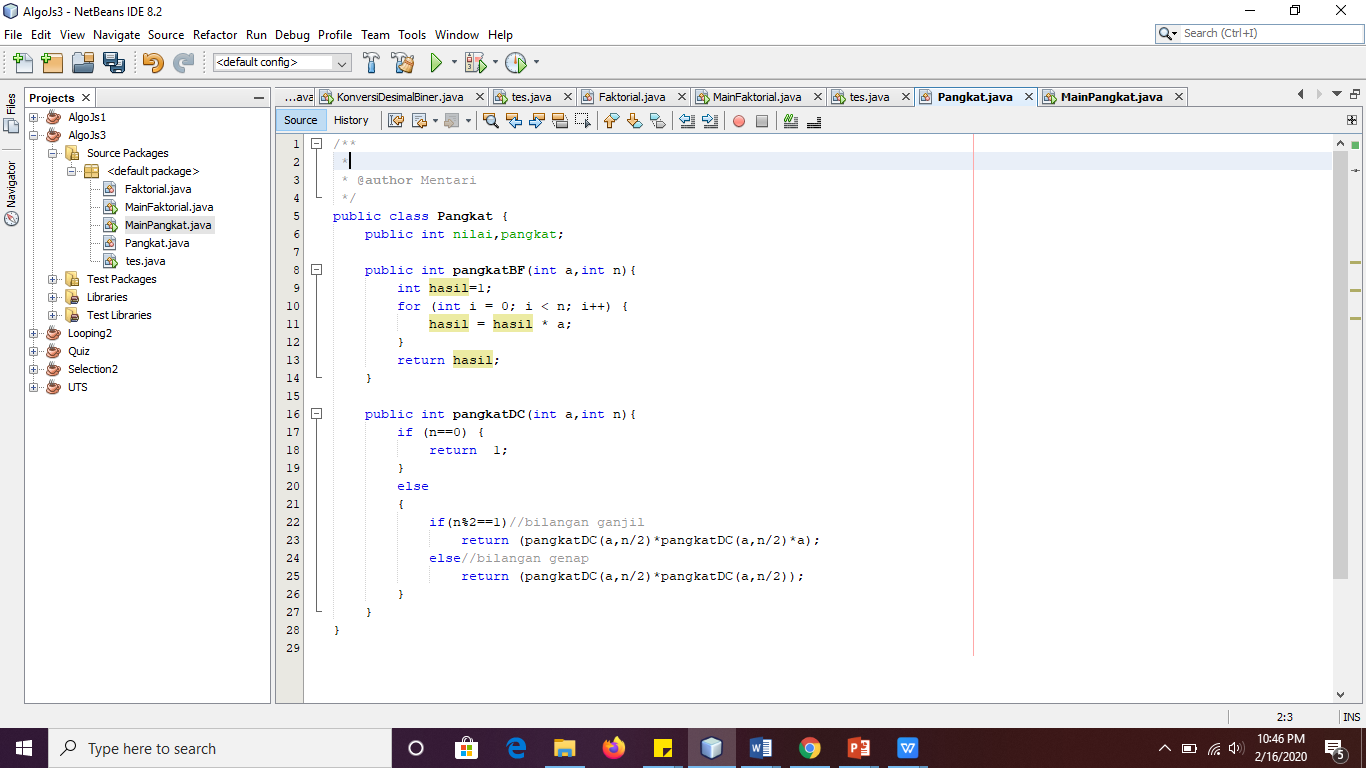
Pada praktikum ini kita akan membuat program class dalam Java. Untuk menghitung nilai pangkat suatu angka menggunakan 2 jenis algoritma, Brute Force dan Divide and Conquer.

* + 1. Langkah-langkah Percobaan

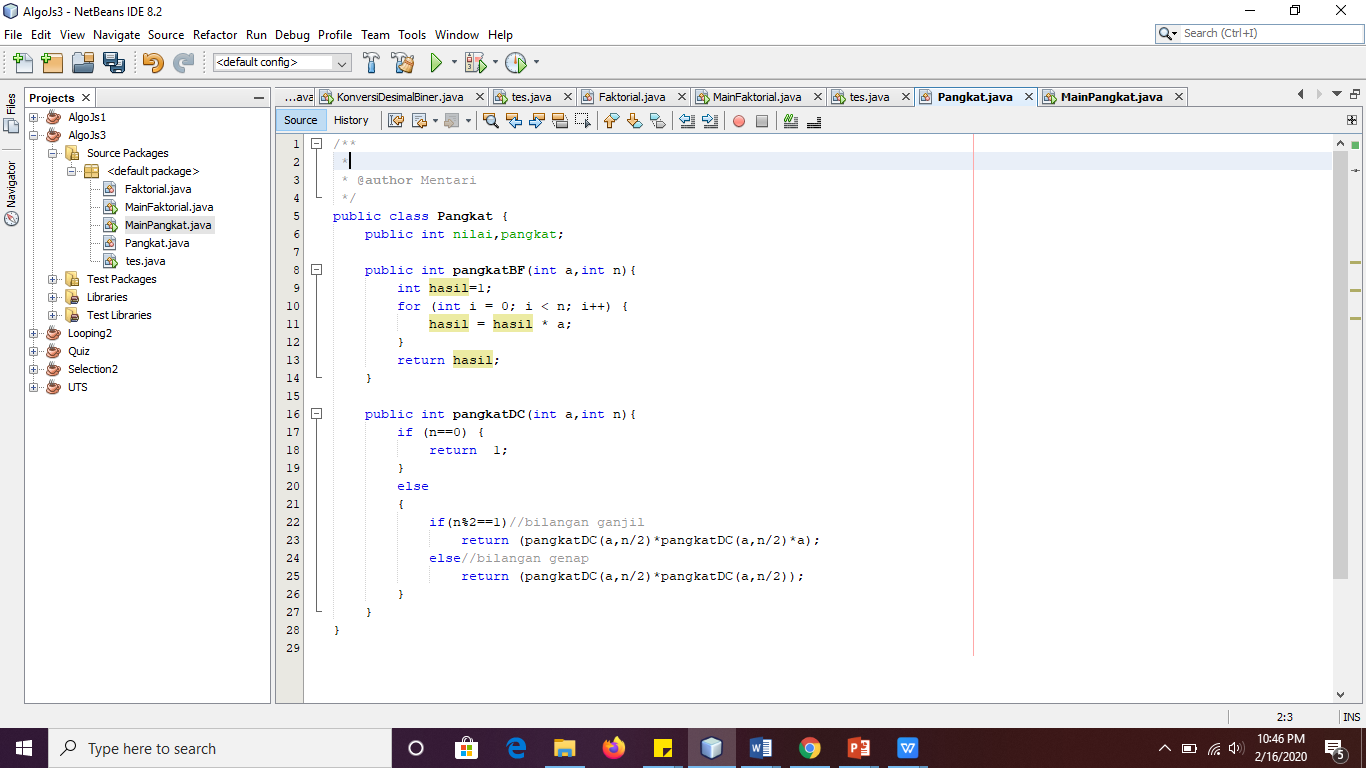
1. Di dalam paket **minggu5**, buatlah class baru dengan nama **Pangkat**. Dan di dalam class **Pangkat** tersebut, buat atribut angka yang akan dipangkatkan sekaligus dengan angka pemangkatnya



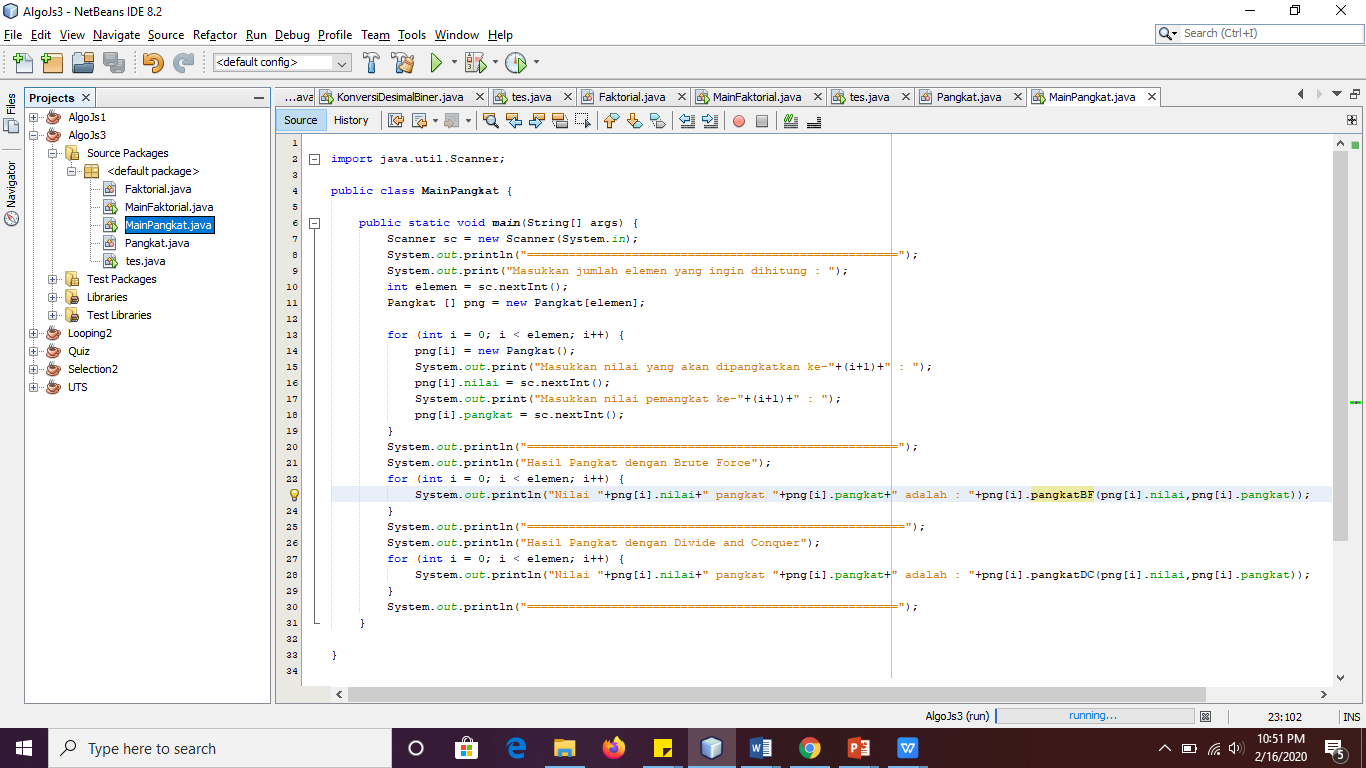
1. Pada class Pangkat tersebut, tambahkan method PangkatBF()



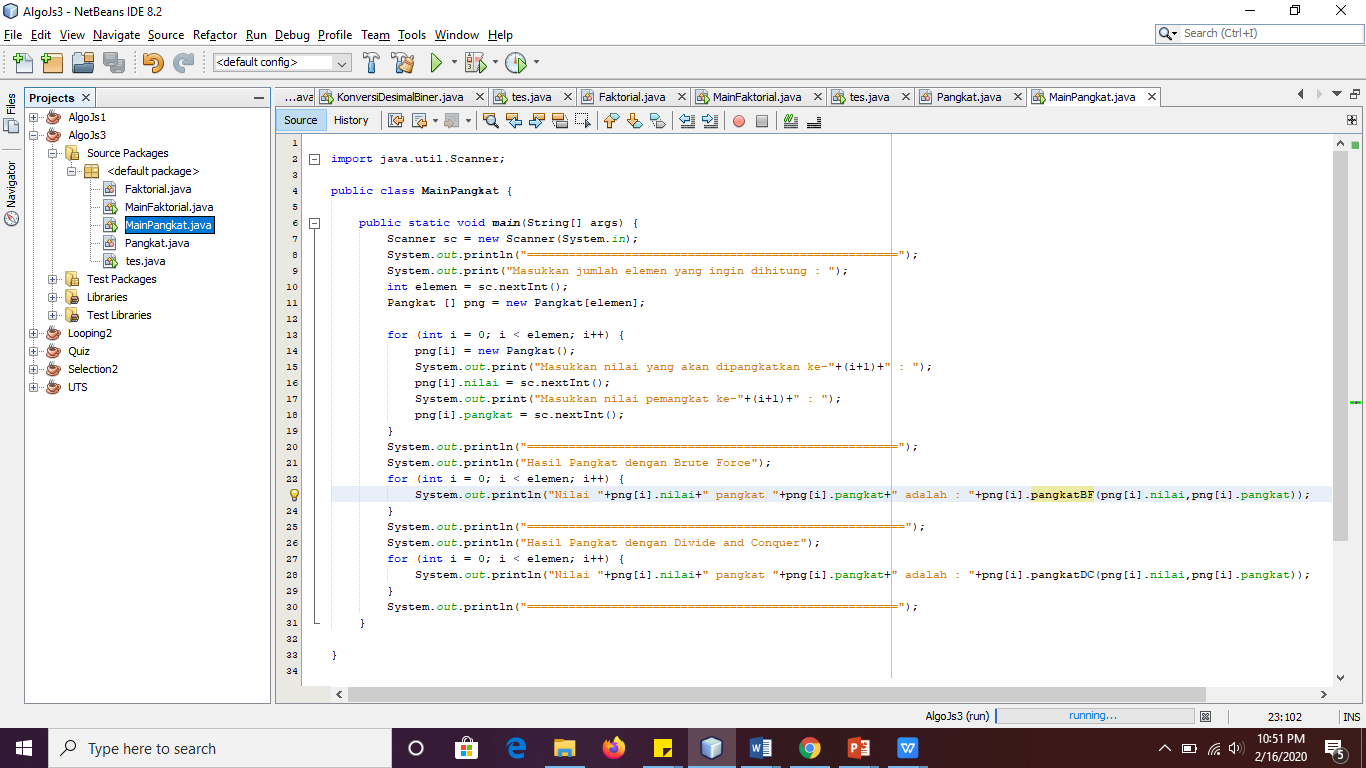
1. Pada class Pangkat juga tambahkan method PangkatDC()



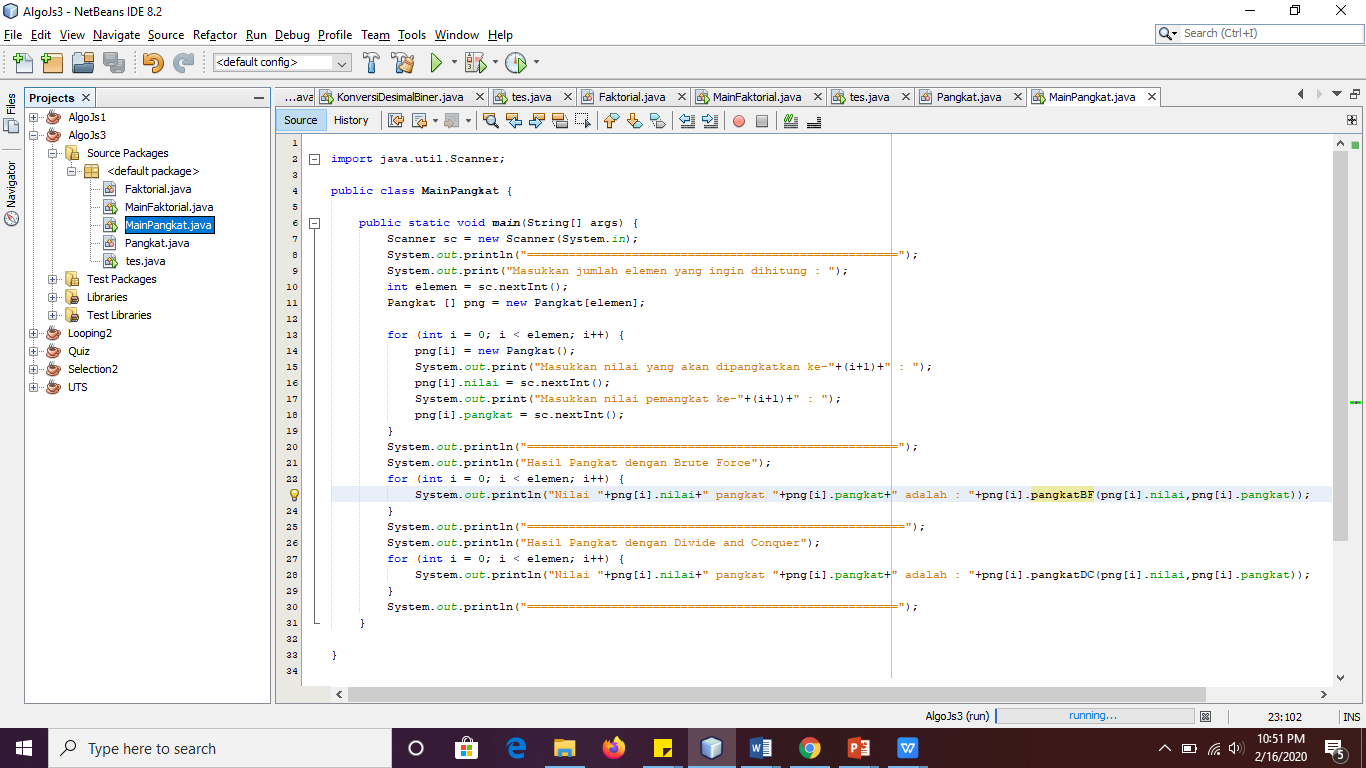
1. Perhatikan apakah sudah tidak ada kesalahan yang muncul dalam pembuatan class Pangkat
2. Selanjutnya buat class baru yang di dalamnya terdapat method main. Class tersebut dapat dinamakan MainPangkat. Tambahkan kode pada class main untuk menginputkan jumlah nilai yang akan dihitung pangkatnya.

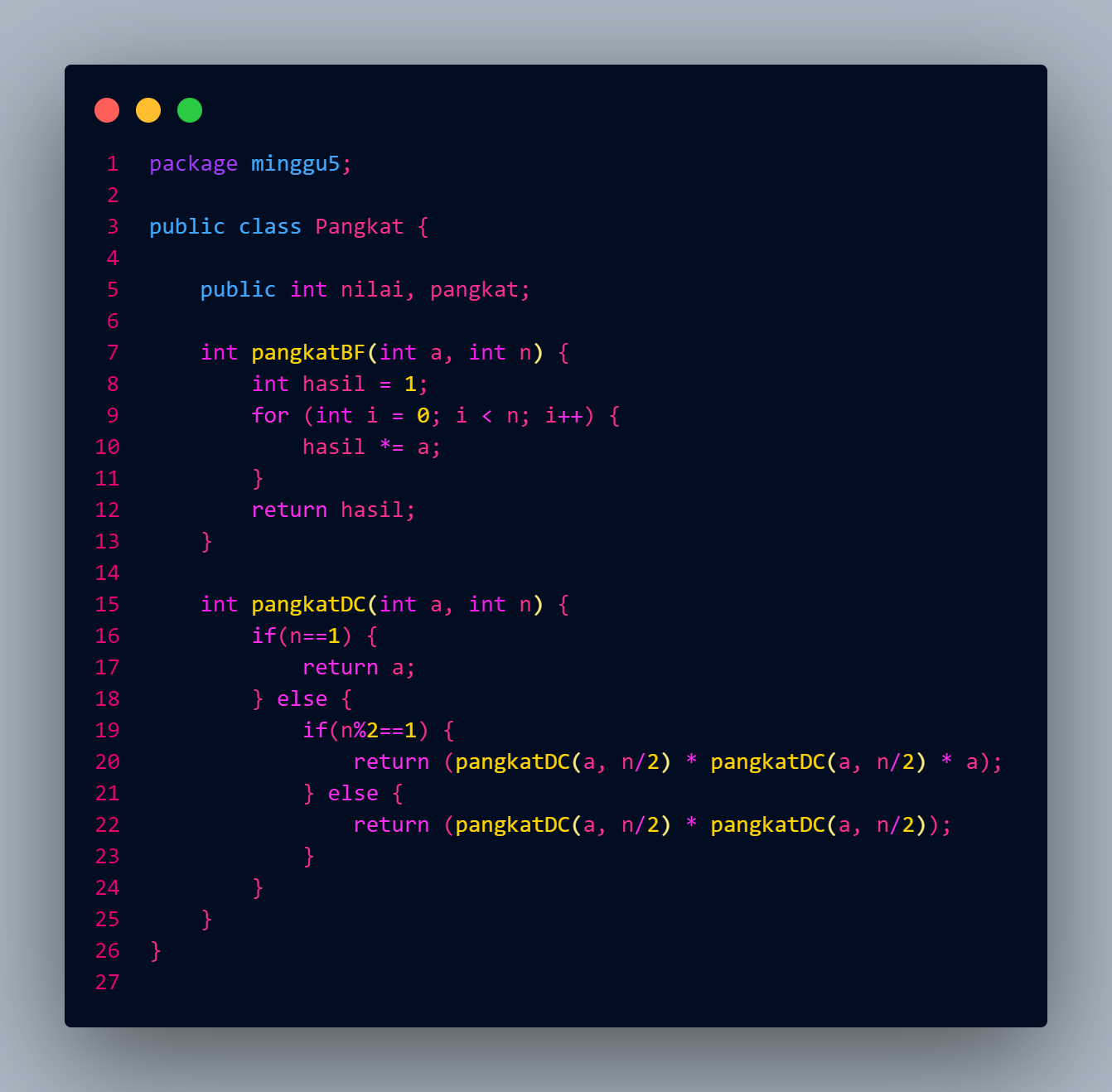


1. Nilai pada tahap 5 selanjutnya digunakan untuk instansiasi array of objek. Di dalam Kode berikut ditambahkan proses pengisian beberapa nilai yang akan dipangkatkan sekaligus dengan pemangkatnya.



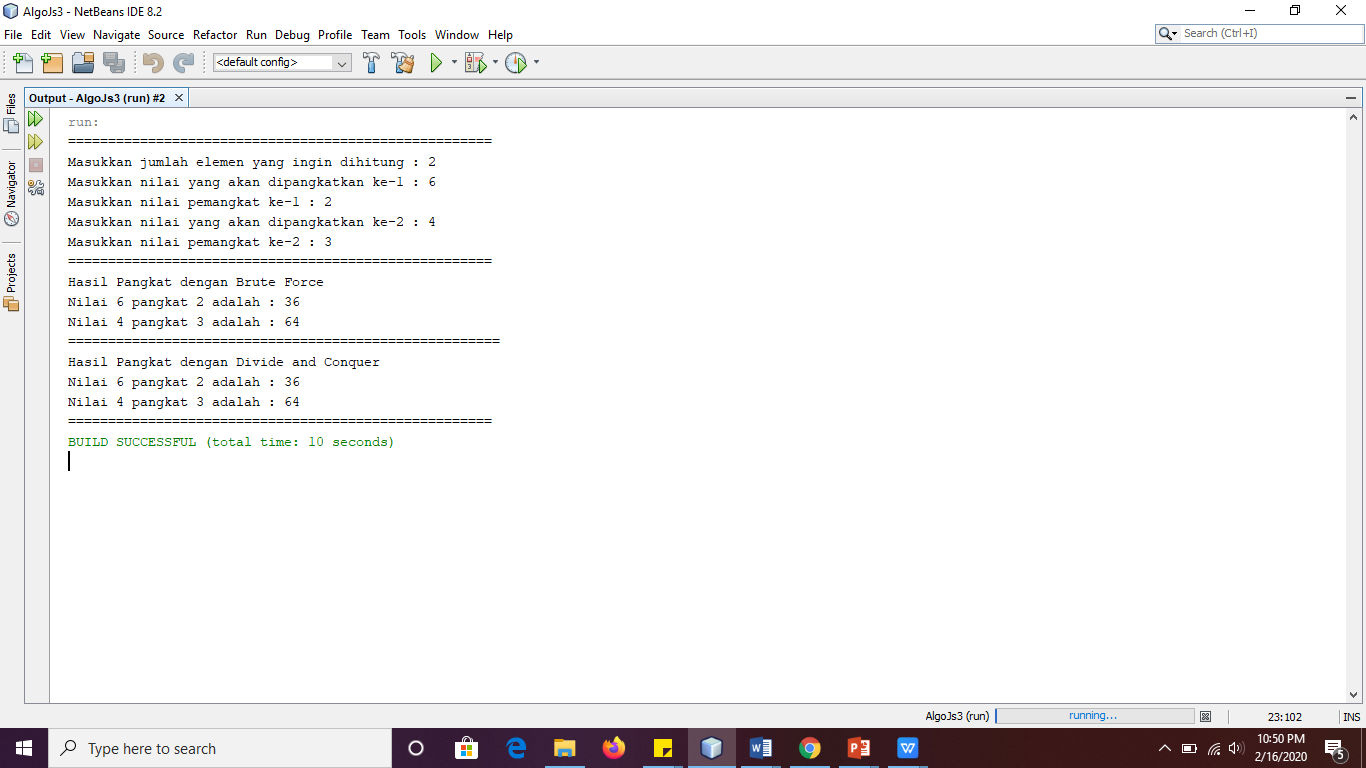
1. Kemudian, panggil hasil nya dengan mengeluarkan return value dari method PangkatBF() dan PangkatDC().





* + 1. Verifikasi Hasil Percobaan

Pastikan output yang ditampilkan sudah benar seperti di bawah ini.



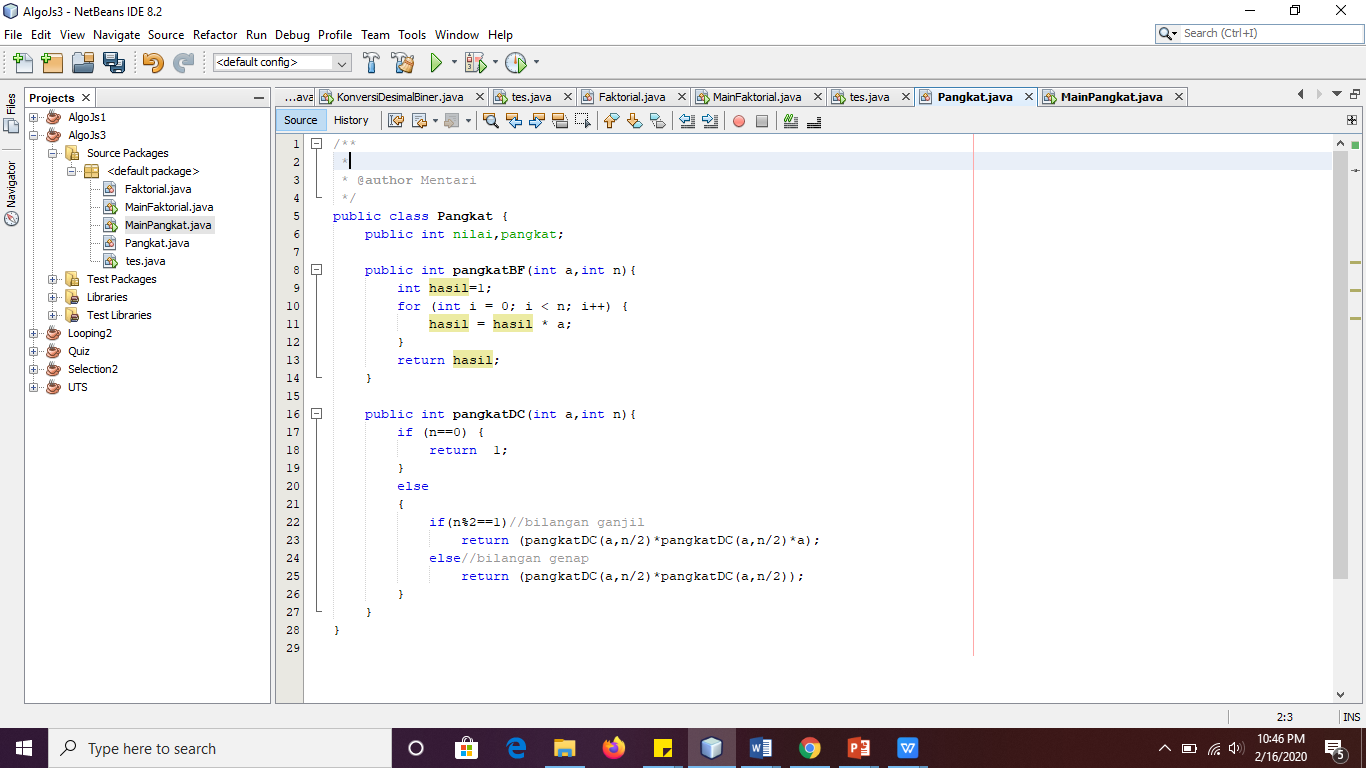


* + 1. Pertanyaan

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

Jawaban: PangkatBF() melakukan perulangan iteratif, sedangkan PangkatDC() menggunakan rekursif

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



Jelaskan arti potongan kode tersebut

Jawaban:

Kode tersebut merupakan bagian dari method pangkatDC jika bernilai, yang berarti jika n

modulo (hasil sisa pembagian) 2 == 1, bilangan tersebut merupakan bilangan ganjil dan akan

mengembalikan nilai dari pemanggilan rekursif ke method pangkatDC dengan membagi n menjadi

dua dan mengalikan hasilnya dengan dirinya sendiri serta dengan bilangan a. Lalu kondisi jika n

adalah bilangan genap, maka algoritma akan melakukan pemanggilan rekursif ke method

pangkatDC dengan membagi n menjadi dua dan mengalikan hasilnya dengan dirinya sendiri.

Karena pada case ini, pangkat n bisa dianggap sebagai hasil perkalian dua pangkat n/2.

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

Ya, dalam algoritma Divide and Conquer, tahap combine terjadi pada saat hasil dari

submasalah yang lebih kecil digabungkan untuk menghasilkan solusi dari masalah asli. Tahap

combine terjadi pada kondisi jika n bilangan ganjil :

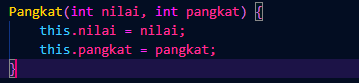
return (pangkatDC(a, n/2) \* pangkatDC(a, n/2) \* a);

Pada kode tersebut, hasil dari kedua pemanggilan rekursif pangkatDC(a, n/2) digabungkan dengan mengalikan kembali hasilnya dengan a. Selanjutnya pada kondisi jika n adalah bilangan genap, berikut adalah kodenya :

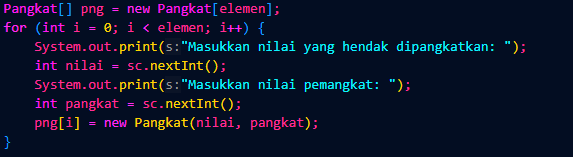
return (pangkatDC(a, n/2) \* pangkatDC(a, n/2));

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

Pangkat.java

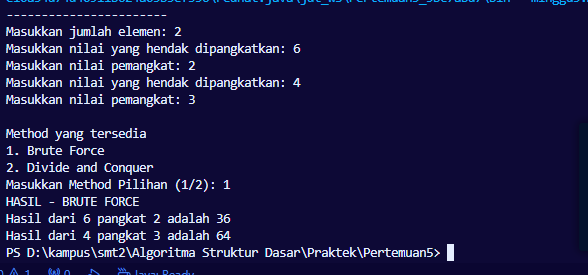


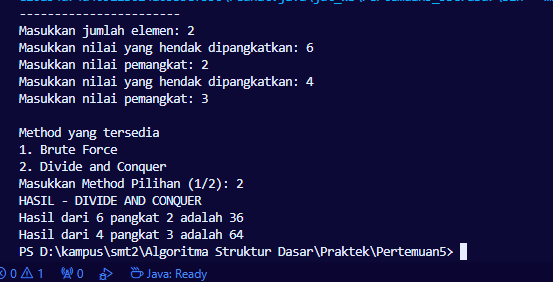
MainPangkat.java



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





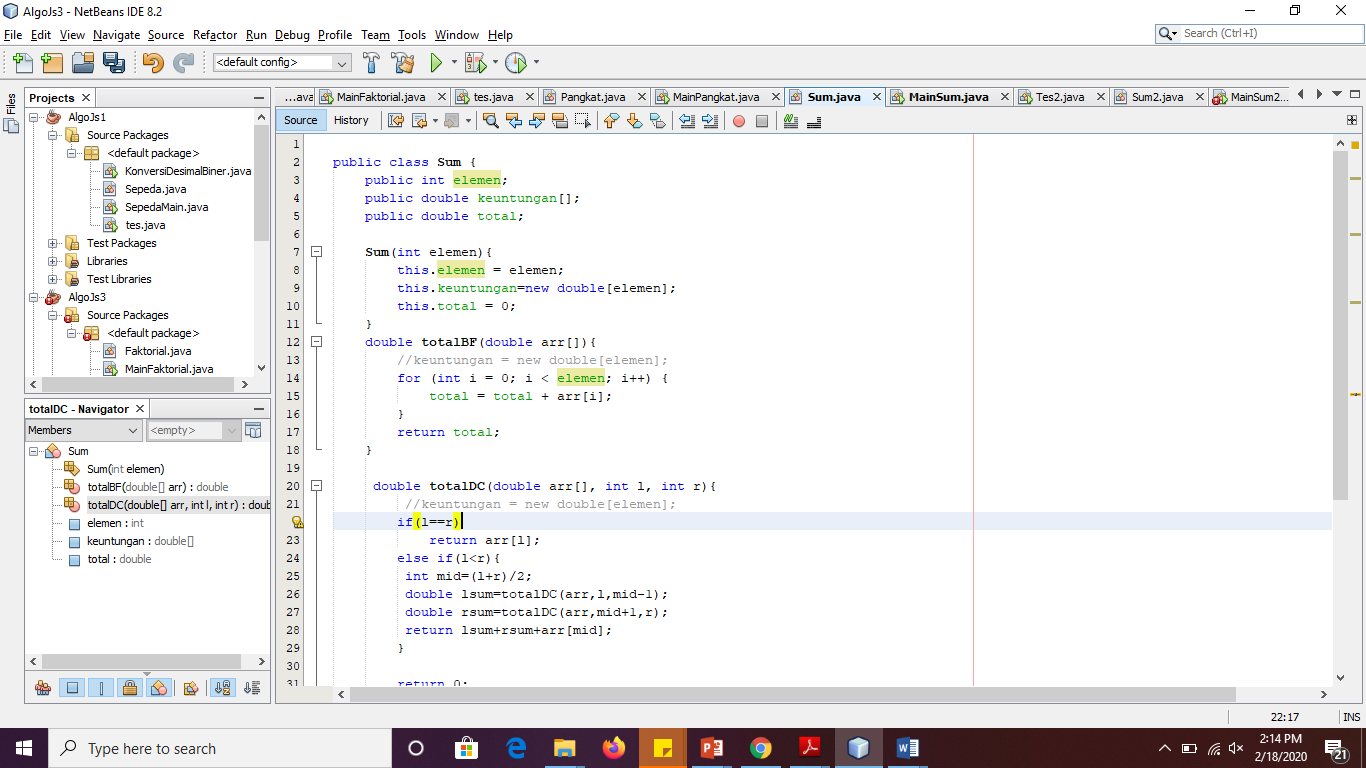


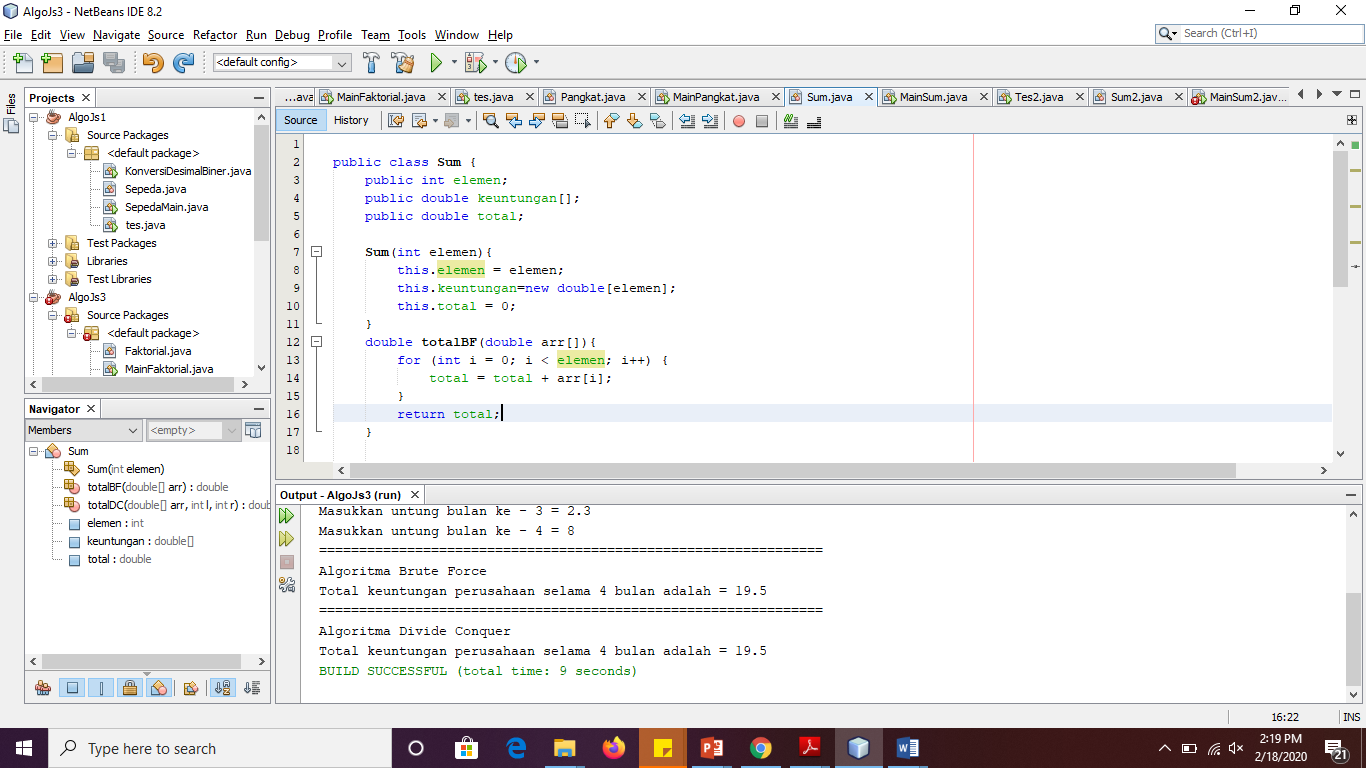
1. Menghitung Sum Array dengan Algoritma Brute Force dan Divide and Conquer

Di dalam percobaan ini, kita akan mempraktekkan bagaimana proses *divide, conquer*, dan *combine* diterapkan pada studi kasus penjumlahan keuntungan suatu perusahaan dalam beberapa bulan.

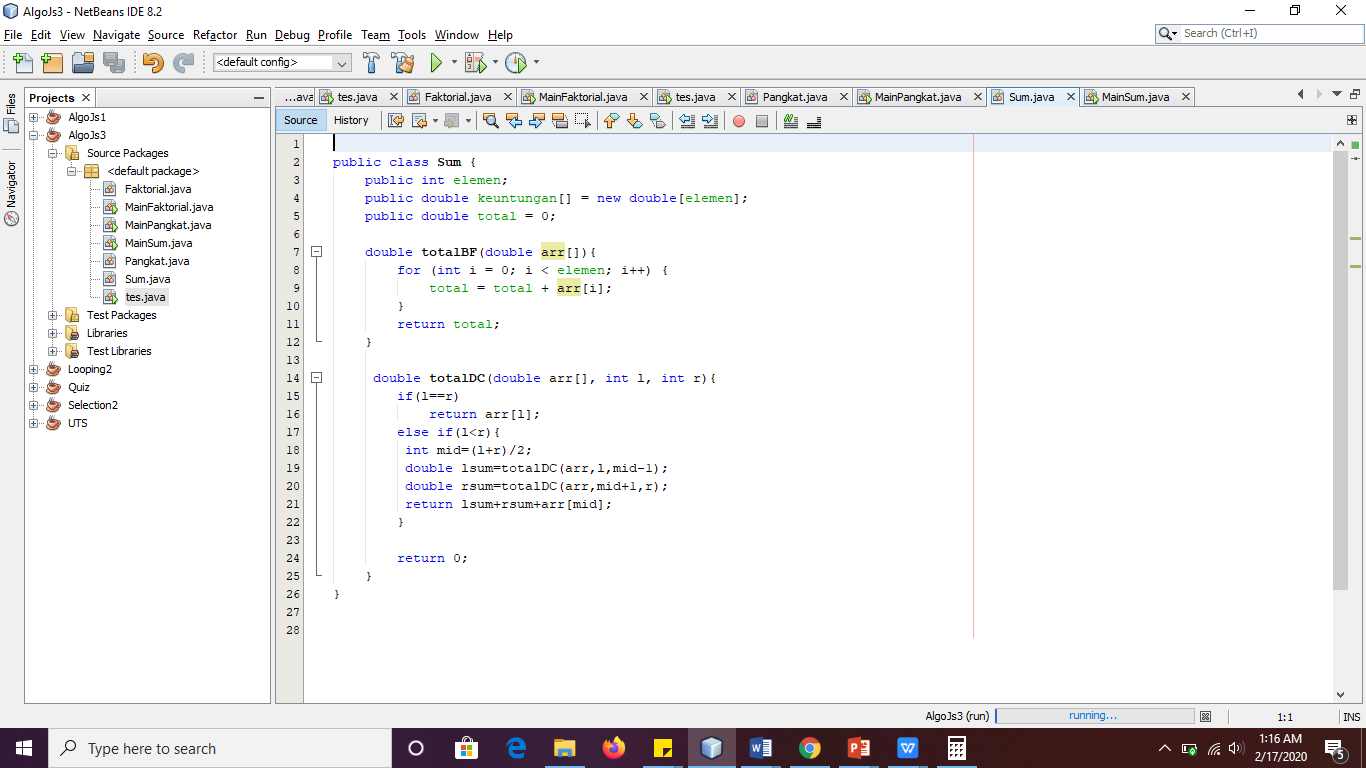
* + 1. Langkah-langkah Percobaan

1. Pada paket minggu5. Buat class baru yaitu class Sum. DI salam class tersebut terdapat beberapa atribut jumlah elemen array, array, dan juga total. Tambahkan pula konstruktor pada class Sum.

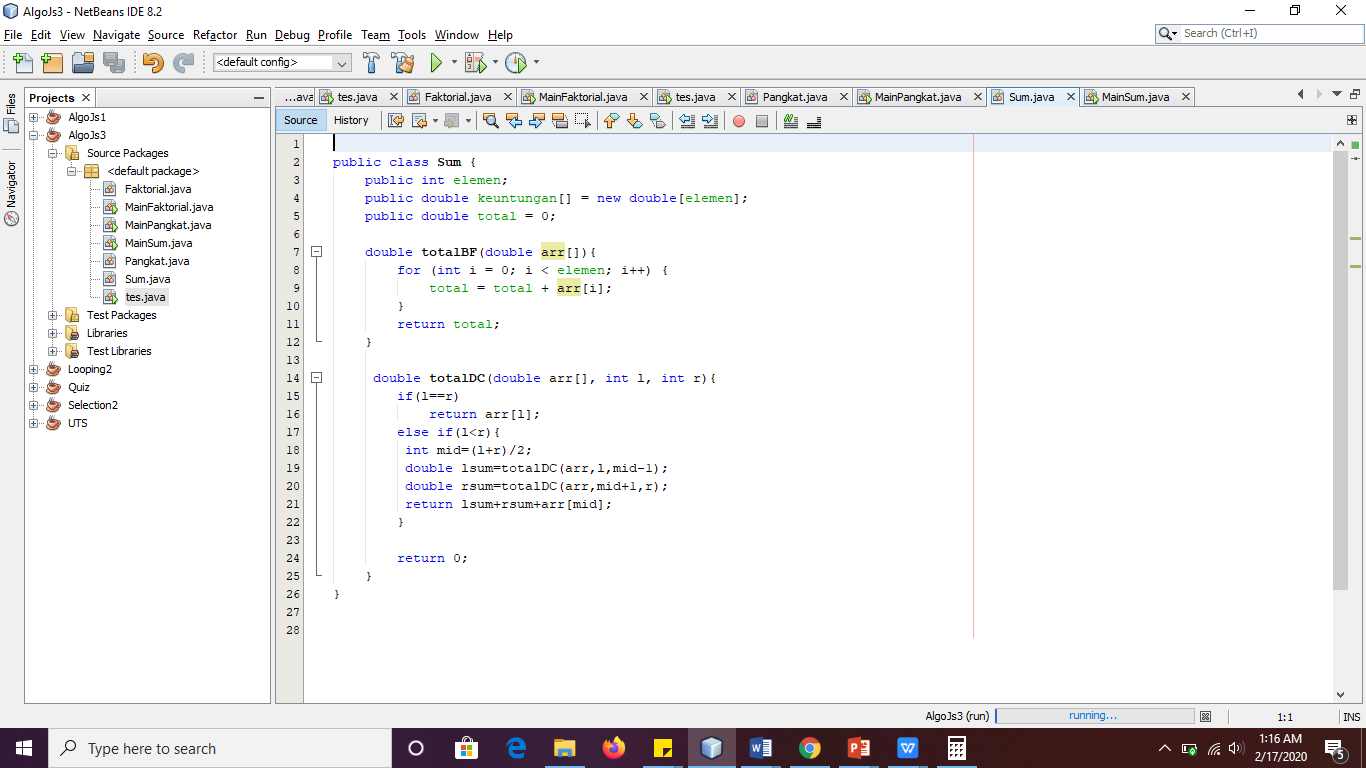




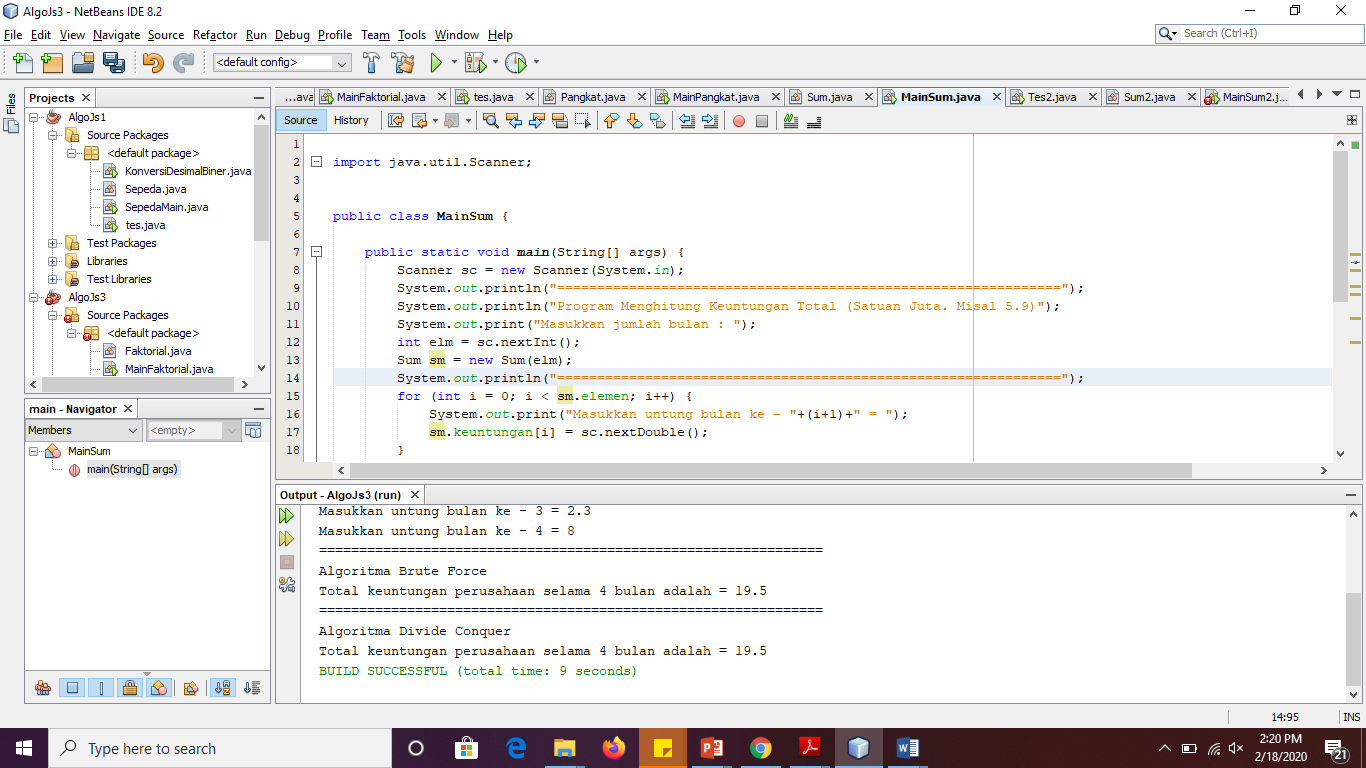
1. Tambahkan method TotalBF() yang akan menghitung total nilai array dengan cara *iterative*.



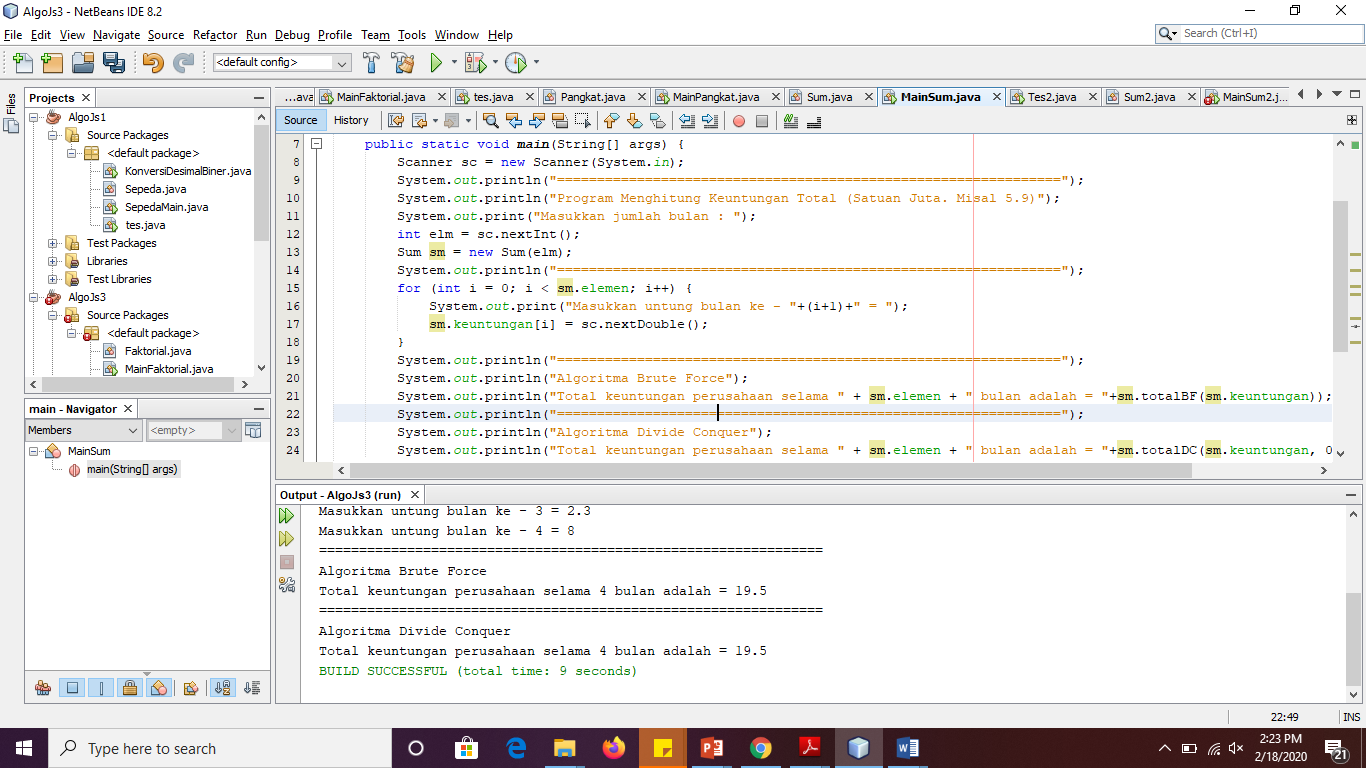
1. Tambahkan pula method TotalDC() untuk implementasi perhitungan nilai total array menggunakan algoritma Divide and Conquer



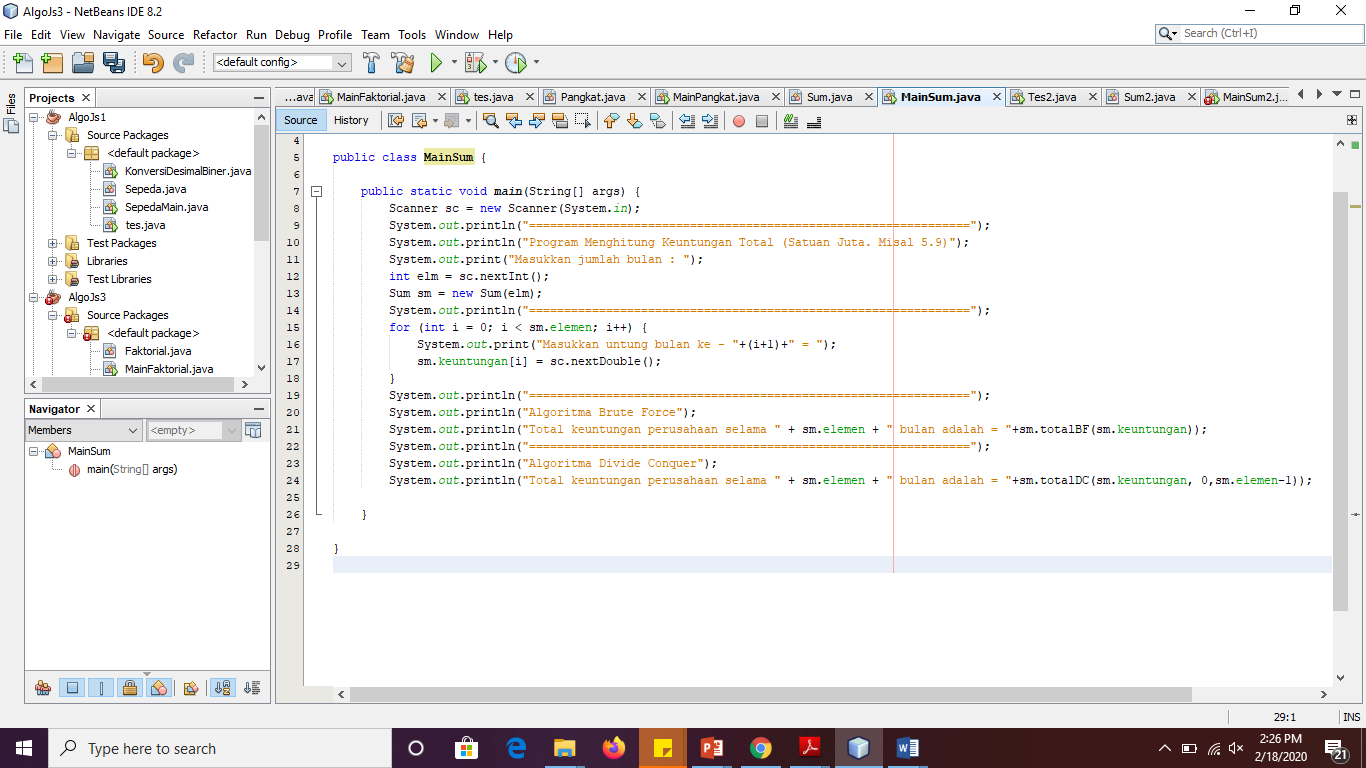
1. Buat class baru yaitu MainSum. Di dalam kelas ini terdapat method main. Pada method ini user dapat menuliskan berapa bulan keuntungan yang akan dihitung. Dalam kelas ini sekaligus dibuat instansiasi objek untuk memanggil atribut ataupun fungsi pada class Sum

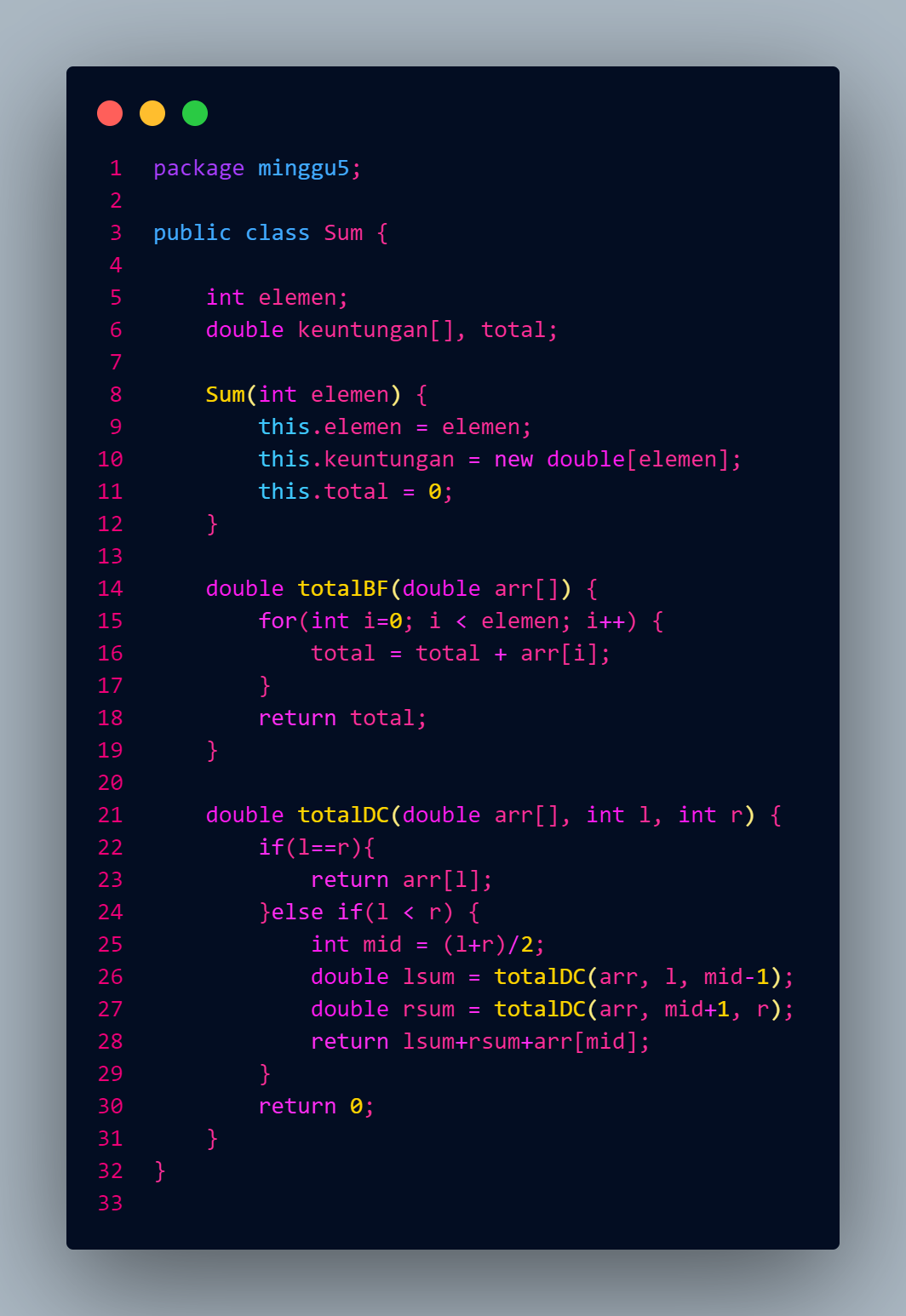


1. Karena yang akan dihitung adalah total nilai keuntungan, maka ditambahkan pula pada method main mana array yang akan dihitung. Array tersebut merupakan atribut yang terdapat di class Sum, maka dari itu dibutuhkan pembuatan objek Sum terlebih dahulu.



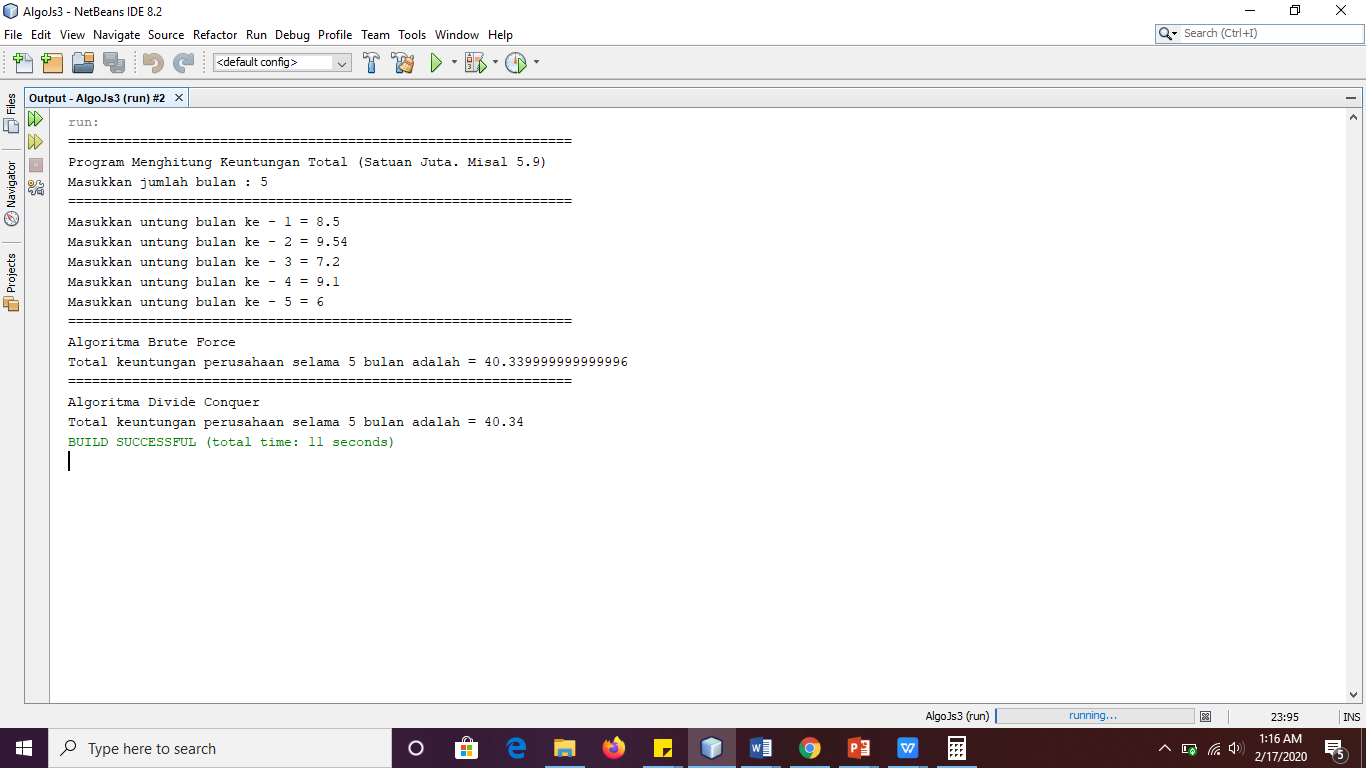
1. Tampilkan hasil perhitungan melalui objek yang telah dibuat untuk kedua cara yang ada (Brute Force dan Divide and Conquer)

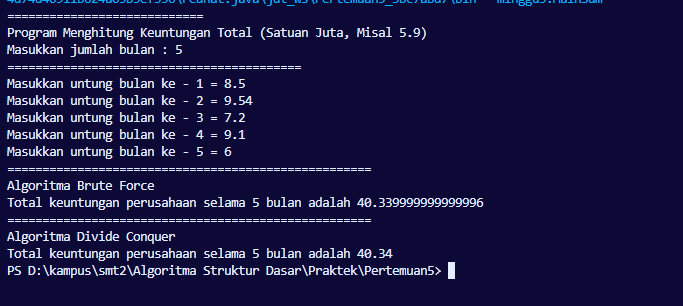




* + 1. Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.





* + 1. Pertanyaan

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

Jawaban: Ilustrasinya adalah sebagai berikut :

Misalkan kita memiliki data keuntungan harian selama seminggu (7 hari), dan kita ingin menghitung total keuntungan dari data tersebut. Contoh data keuntungan harian:

Hari 1: $100

Hari 2: $150

Hari 3: $200

Hari 4: $250

Hari 5: $300

Hari 6: $350

Hari 7: $400

Dengan menggunakan metode totalBF, kita akan menjumlahkan semua keuntungan harian secara berurutan, seperti berikut : Total keuntungan = 100 + 150 + 200 + 250 + 300 + 350 + 400 = $1750.

Sementara itu, dengan menggunakan metode totalDC, kita akan membagi data menjadi submasalah yang lebih kecil. Kita akan mencari total keuntungan untuk setengah pertama hari (hari 1 hingga hari 4) dan setengah kedua hari (hari 5 hingga hari 7) terlebih dahulu, lalu menambahkan keduanya bersamaan dengan keuntungan pada hari keempat. Berikut adalah perhitungannya :

Total keuntungan untuk setengah pertama hari: 100 + 150 + 200 + 250 = $700

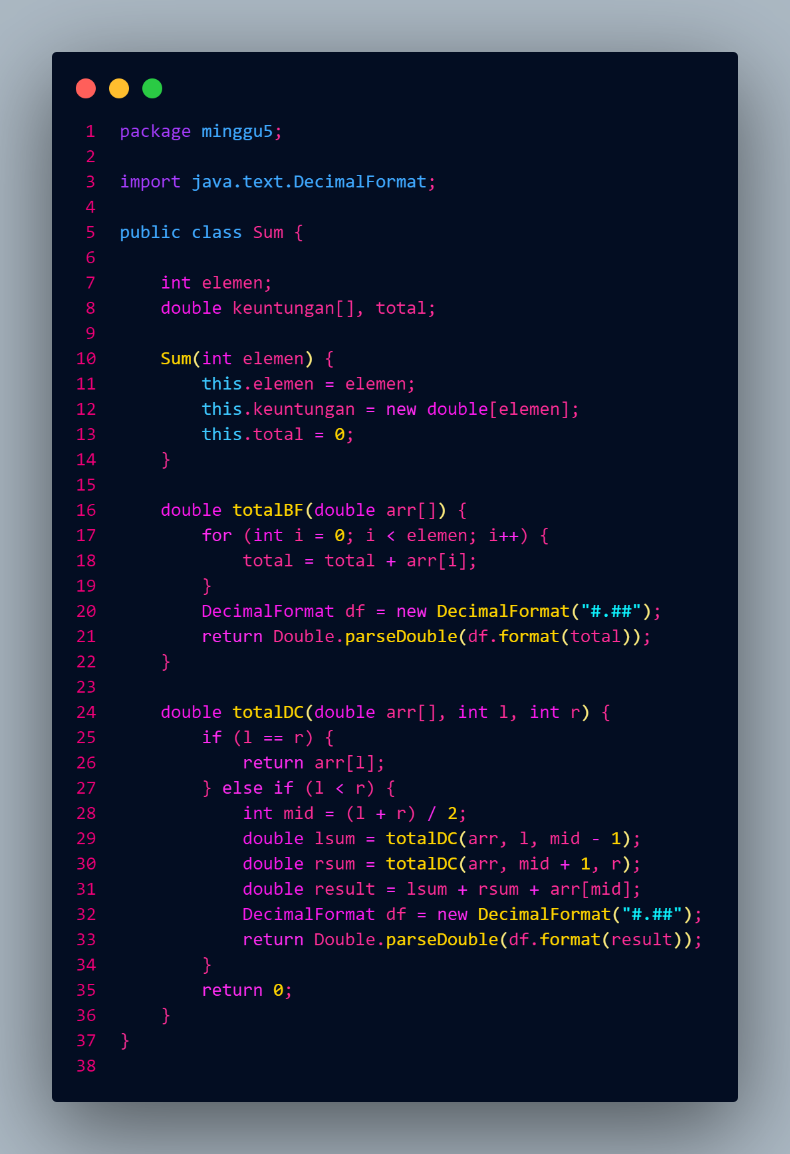
Total keuntungan untuk setengah kedua hari: 300 + 350 + 400 = $1050

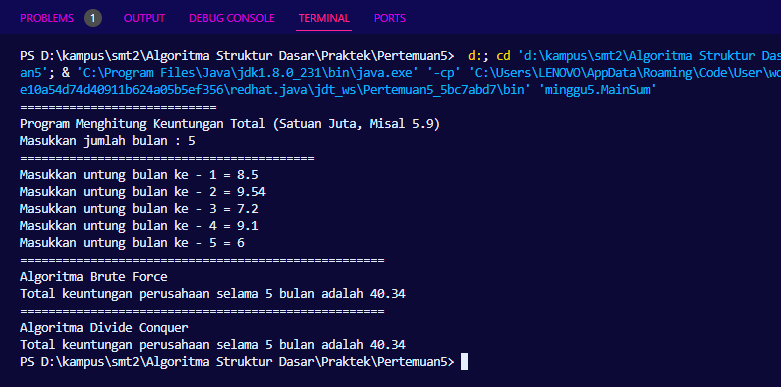
Total keuntungan = 700 + 1050 + 250 = $2000

Jadi, kita bisa melihat bahwa hasil perhitungan total keuntungan menggunakan metode totalDC akan berbeda dari hasil yang didapatkan menggunakan metode totalBF. Meskipun keduanya memberikan hasil yang benar, namun pendekatan yang digunakan untuk menghitungnya berbeda. Metode totalDC membagi masalah menjadi submasalah yang lebih kecil dan menggabungkan hasilnya, sementara metode totalBF langsung menjumlahkan semua data secara berurutan.

1. 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.

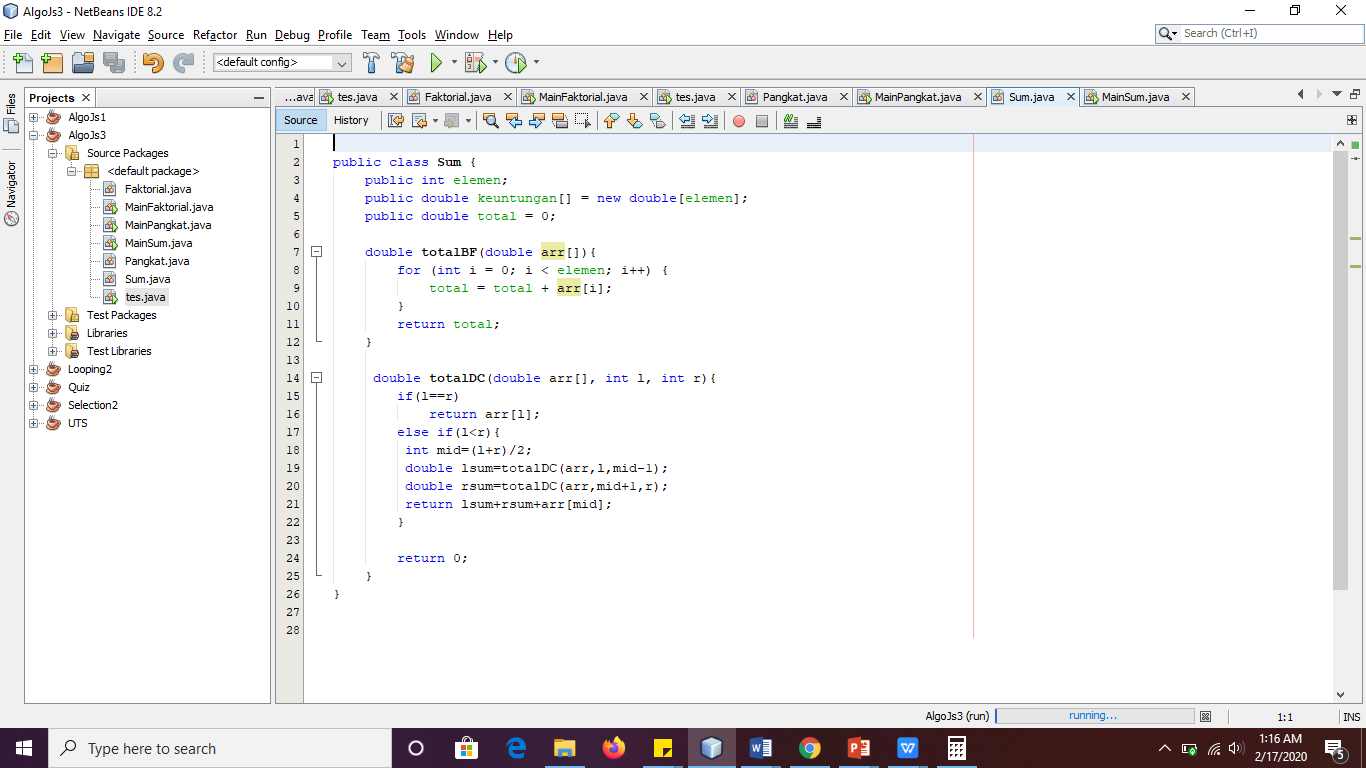
Untuk membatasi output di belakang koma agar menjadi standar untuk kedua jenis algoritma, kita dapat menggunakan DecimalFormat untuk memformat output sehingga memiliki jumlah angka di belakang koma yang sama. Berikut adalah kode mofdifikasinya :





Dengan menggunakan DecimalFormat, kita dapat memastikan bahwa output dari kedua metode memiliki jumlah angka di belakang koma yang sama

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



Jawaban:

Kode tersebut merupakan bagian dari algoritma Divide and Conquer untuk menghitung total keuntungan dari suatu array.

1. lsum adalah total keuntungan dari submasalah bagian kiri.
2. rsum adalah total keuntungan dari submasalah bagian kanan.
3. arr[mid] adalah keuntungan dari hari tengah (atau elemen tengah) di antara submasalah bagian kiri dan kanan.

Dengan demikian, formulasi return lsum + rsum + arr[mid]; tersebut memungkinkan algoritma Divide and Conquer untuk menggabungkan hasil dari submasalah yang lebih kecil menjadi solusi dari masalah asli, yaitu menghitung total keuntungan dari seluruh array.

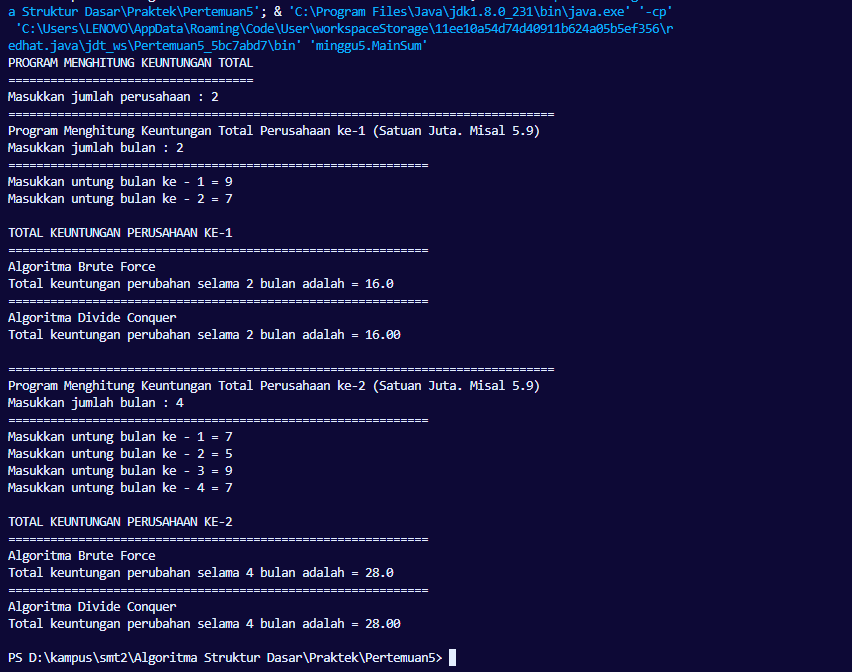
1. Kenapa dibutuhkan variable mid pada method TotalDC()?

Jawaban: Variabel mid dibutuhkan dalam metode totalDC() karena method tersebut menggunakan pendekatan Divide and Conquer. Ketika kita membagi array menjadi dua bagian, mid digunakan untuk menentukan indeks tengah dari array tersebut. Dengan mengetahui indeks tengah, kita dapat membagi array menjadi dua bagian yang sama besar (atau hampir sama besar jika jumlah elemen ganjil). ada tahap rekursif, submasalah tersebut kemudian dibagi kembali menjadi submasalah yang lebih kecil dengan cara yang sama, sampai mencapai submasalah yang dapat diselesaikan langsung (misalnya, submasalah dengan hanya satu elemen). Kemudian, hasil dari submasalah yang lebih kecil digabungkan kembali untuk mendapatkan solusi dari masalah asli.

1. 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. 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.*



