JOBSHEET III
BRUTE FORCE dan DIVIDE CONQUER

2.1 Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Mahasiswa mengetahui tentang penggunaan algoritma brute force dan algoritma devide-conquer
2. Mahasiswa mampu menerapkan penggunaan algorima brute force dan devide-conquer

2.2 Ulasan Teori

Pemecahan masalah secara komputasional dapat diselesaikan dengan berbagai cara. Algoritma brute force dan divide conquer adalah dua jenis pendekatan berbeda yang biasanya digunakan dalam penyelesaian masalah.

2.2.1 Brute Force

Brute force adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Dasar pemecahan dengan algoritma brute force didapatkan dari pernyataan pada persoalan (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan persoalan dengan sangat sederhana, langsung, jelas (*obvious way*).

Algoritma brute force umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan jumlah komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Terkadang algoritma brute force disebut juga algoritma naif (*naïve algorithm*). Algoritma brute force lebih cocok untuk persoalan yang berukuran kecil karena mudah diimplementasikan dan tata cara yang sederhana. Algoritma brute force sering digunakan sebagai basis pembanding dengan algoritma yang lebih mangkus. Meskipun bukan metode yang mangkus, hampir semua persoalan dapat diselesaikan dengan algoritma brute force. Terdapat beberapa kelemahan dan kelebihan algoritma brute force, sebagai berikut :

Kelebihan :

1. Metode brute force dapat digunakan untuk memecahkan hampir sebagian besar masalah (*wide applicability*).
2. Metode brute force sederhana dan mudah dimengerti.
3. Metode brute force menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
4. Metode brute force menghasilkan algoritma baku (*standard*) untuk tugas-tugas komputasi seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam tabel (*list*).

Kelemahan :

1. Metode brute force jarang menghasilkan algoritma yang mangkus.
2. Beberapa algoritma brute force lambat sehingga tidak dapat diterima.
3. Tidak sekontruktif/sekreatif teknik pemecahan masalah lainnya.

Bruteforce dapat juga diimplementasikan dalam metode searching dan sorting yang akan lebih detil dijelaskan pada minggu-minggu berikutnya. Pada proses searching, prinsip penyelesaian dengan metode bruteforce dapat dilihat pada metode sequential search. Sequential Search disebut juga Linear Search. Proses pencarian membandingkan nilai kunci dengan semua elemen nilai. Membandingkan nilai kunci dengan elemen pertama sampai elemen terakhir, atau, Proses terhenti jika nilai kunci cocok dengan nilai elemen tanpa harus membandingkan semua elemen.

Sedangkan untuk sorting, bruteforce digunakan sebagai prinsip dasar tahapan metode bubble sort dan Selection Sort. Algoritma Bubble Sort merupakan proses pengurutan yang secara berangsur-angsur berpindah ke posisi yang tepat (Bubble). Algoritma ini akan mengurutkan data dari yang terbesar ke yang terkecil (ascending) atau sebaliknya (descending). Secara sederhana, bisa

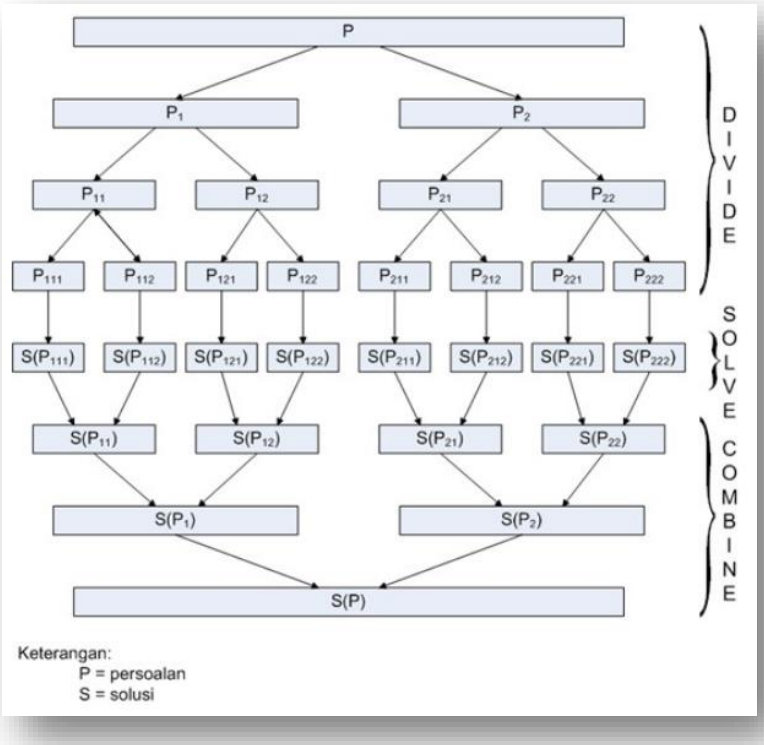
didefenisikan algoritma Bubble Sort adalah pengurutan dengan cara pertukaran data dengan data disebelahnya secara terus menerus sampai dalam satu iterasi tertentu tidak ada lagi perubahan. Metode selection sort merupakan perbaikan dari metode bubble sort dengan mengurangi jumlah perbandingan. Selection sort merupakan metode pengurutan dengan mencari nilai data terkecil dimulai dari data diposisi 0 hingga diposisi N-1. Jika terdapat N data dan data terkoleksi dari urutan 0 sampai dengan N-1 maka algoritma pengurutan dengan metode selection sort adalah sebagai berikut: 1. Cari data terkecil dalam interval j = 0 sampai dengan j = N-1 2. Jika pada posisi pos ditemukan data yang terkecil, tukarkan data diposisi pos dengan data di posisi i jika k. 3. Ulangi langkah 1 dan 2 dengan j = j+i sampai dengan j = N-1, dan seterusnya sampai j = N – 1.

2.2.2 Divide Conquer

Divide and Conquer pada awalnya merupakan strategi militer yang dikenal dengan nama *divide ut imperes*. Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama Divide and Conquer. Divide artinya membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama), Conquer adalah cara memecahkan (menyelesaikan) masing-masing secara rekursif, dan Combine yang tujuanya adalah mengabungkan solusi masing-masing masalah sehingga membentuk solusi masalah semula. Obyek permasalahan tersebut, dibagi menjadi masukan (*input*) atau instances yang berukuran n seperti:

- tabel (larik),
- matriks,
- eksponen,
- dll, bergantung pada masalahnya.

Tahapan divide, conquer dan combine dapat dilihat pada gambar di bawah ini.



Tiap-tiap masalah mempunyai karakteristik yang sama dengan karakteristik masalah asal, sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif. Pseudocode secara umum penyelesaian masalah dengan algortima divide conquer dapat dilihat sebagai berikut :

```
procedure DIVIDE_and_CONQUER(input n : integer)
{ Menyelesaikan masalah dengan algoritma D-and-C.
  Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula}
Deklarasi
    r, k : integer
Algoritma
  if n ≤ n0 then  {ukuran masalah sudah cukup kecil }
     SOLVE upa-masalah yang berukuran n ini
  else
     Bagi menjadi r upa-masalah, masing-masing berukuran n/k
     for masing-masing dari rupa-masalah do
        DIVIDE_and_CONQUER(n/k)
     endfor
     COMBINE solusi dari r upa-masalah menjadi solusi masalah semula
  endif
```

Kelemahan Divide dan Conquer

- Lambatnya proses perulangan
    - Lambatnya proses perulangan Proses pemanggilan sub-rutin (dapat memperlambat proses perulangan) yang berlebih akan menyebabkan call stack penuh. Hal ini dapat menjadi beban yang cukup signifikan pada prosesor. Lebih rumit untuk masalah yang sederhana
- Lebih rumit untuk masalah yang sederhana
    - Untuk pemecahan masalah yang relatif sederhana, algoritma sekuensial terbukti lebih mudah dibuat daripada algoritma divide and conquer.

Kelebihan Divide dan Conquer

- Dapat memecahkan masalah yang sulit. Memecahkan masalah Divide and conquer merupakan cara yang sangat efektif jika masalah yang akan diselesaikan cukup rumit.
- Memiliki efisiensi algoritma yang tinggi. Pendekatan divide and conquer ini lebih efisien dalam menyelesaikan algoritma sorting.
- Dapat bekerja secara paralel. Divide and Conquer telah didisain untuk dapat bekerja dalam mesin-mesin yang memiliki banyak prosesor. Terutama mesin yang memiliki sistem pembagian memori, dimana komunikasi data antar prosesor tidak perlu direncanakan terlebih dahulu, hal ini karena pemecahan sub-rutin dapat dilakukan di prosesor lainnya.
- Akses memori yang cukup kecil. Untuk akses memori, Divide and Conquer dapat meningkatkan efisiensi memori yang ada cukup baik. Hal ini karena, sub-rutin memerlukan memori lebih kecil daripada masalah utamanya.

Divide conquer menjadi dasar pula dalam searching dan sorting. Metode sorting yang menggunakan dasar divide conquer adalah merge sort. Metode pengurutan merge sort adalah metode pengurutan lanjut, sama dengan metode Quick Sort. Metode ini juga menggunakan konsep

devide and conquer yang membagi data S dalam dua kelompok yaitu S1 dan S2 yang tidak beririsan (*disjoint*). Proses pembagian data dilakukan secara rekursif sampai data tidak dapat dibagi lagi atau dengan kata lain data dalam sub bagian menjadi tunggal. Setelah data tidak dapat dibagi lagi, proses penggabungan (*merging*) dilakukan antara sub-sub bagian dengan memperhatikan urutan data yang diinginkan (*ascending*/kecil ke besar atau descending/besar ke kecil). Proses penggabungan ini dilakukan sampai semua data tergabung dan terurut sesuai urutan yang diiginkan.
Sedangkan metode searching yang menggunakan cara pembagian solusi seperti divide conquere adalah binary search. Binary search merupakan salah satu algoritma untuk melalukan pencarian pada array yang sudah terurut. Jika kita tidak mengetahui informasi bagaimana integer dalam array, maka penggunaan binary search akan menjadi tidak efisien, kita harus melakukan sorting terlebih dahulu atau menggunakan metode lain yaitu linear search. Namun jika kita telah mengetahui integer dalam array terorganisasi baik secara menaik atau menurun, maka bisa dengan cepat menggunakan algoritma binary search.

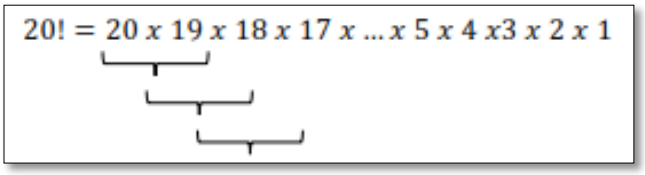
2.3 Menghitung Nilai Faktorial dengan Algoritma Brute Force dan Divide and Conquer

Perhatikan Diagram Class berikut ini :

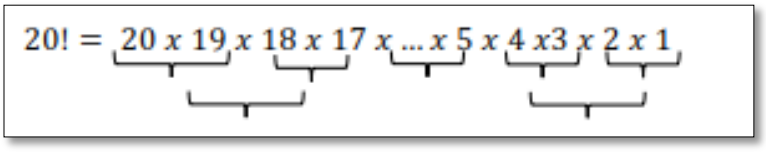
Faktorial
nilai: int
faktorialBF(): int faktorialDC(): int

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 :



2.3.1 Langkah-langkah Percobaan

1. Buat Project baru, dengan nama **AlgoStruDat**/Nama Project Disamakan dengan minggu lalu. Buat paket dengan nama minggu3**,** buatlah class baru dengan nama **`Faktorial`**.
2. Lengkapi class **`Faktorial`** dengan atribut dan method yang telah digambarkan di dalam diagram class di atas, sebagai berikut:

a) Tambahkan atribut nilai

```
public int nilai;
```

b) Tambahkan method `faktorialBF()` nilai

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

c) Tambahkan method `faktorialDC()` nilai

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

3. Coba jalankan (Run) class `Faktorial` dengan membuat class baru `MainFaktorial`.

a) Di dalam fungsi main sediakan komunikasi dengan user untuk menginputkan jumlah angka yang akan dicari nilai faktorialnya

```
Scanner sc = new Scanner(System.in);
System.out.println("=====================================================");
System.out.print("Masukkan jumlah elemen yang ingin dihitung : ");
int elemen = sc.nextInt();
```

b) Buat Array of Objek pada fungsi main, kemudian inputkan beberapa nilai yang akan dihitung faktorialnya

```
Faktorial [] fk = new Faktorial[elemen];
for (int i = 0; i < elemen; i++) {
    fk[i] = new Faktorial();
    System.out.print("Masukkan nilai data ke-"+(i+1)+" : ");
    fk[i].nilai = sc.nextInt();
}
```

c) Tampilkan hasil pemanggilan method `faktorialDC()` dan `faktorialBF()`

```
System.out.println("=====================================================");
System.out.println("Hasil Faktorial dengan Brute Force");
for (int i = 0; i < elemen; i++) {
    System.out.println("Faktorial dari nilai "+fk[i].nilai+" adalah : "+fk[i].faktorialBF(fk[i].nilai));
}
System.out.println("=====================================================");
System.out.println("Hasil Faktorial dengan Divide and Conquer");
for (int i = 0; i < elemen; i++) {
    System.out.println("Faktorial dari nilai "+fk[i].nilai+" adalah : "+fk[i].faktorialDC(fk[i].nilai));
}
System.out.println("=====================================================");
```

d) Pastikan program sudah berjalan dengan baik!

2.3.2 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
=====================================================
Masukkan jumlah elemen yang ingin dihitung : 3
Masukkan nilai data ke-1 : 5
Masukkan nilai data ke-2 : 8
Masukkan nilai data ke-3 : 3
=====================================================
Hasil Faktorial dengan Brute Force
Faktorial dari nilai 5 adalah : 120
Faktorial dari nilai 8 adalah : 40320
Faktorial dari nilai 3 adalah : 6
=====================================================
Hasil Faktorial dengan Divide and Conquer
Faktorial dari nilai 5 adalah : 120
Faktorial dari nilai 8 adalah : 40320
Faktorial dari nilai 3 adalah : 6
=====================================================
BUILD SUCCESSFUL (total time: 7 seconds)
```

2.3.3 Pertanyaan

1. Jelaskan mengenai base lain 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?

2.4 Menghitung Hasil Pangkat dengan Algoritma Brute Force dan Divide and Conquer

2.4.1 Langkah-langkah Percobaan

1. Di dalam paket **`minggu3`**, buatlah class baru dengan nama **`Pangkat`**. Dan di dalam class **`Pangkat`** tersebut, buat atribut angka yang akan dipangkatkan sekaligus dengan angka pemangkatnya

```
    public int nilai,pangkat;
```

2. Pada class Pangkat tersebut, 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;
}
```

3. Pada class Pangkat juga tambahkan method `PangkatDC()`

```
public int pangkatDC(int a,int n){
    if (n==0) {
        return  1;
    }
    else
    {
        if(n%2==1)//bilangan ganjil
            return (pangkatDC(a,n/2)*pangkatDC(a,n/2)*a);
        else//bilangan genap
            return (pangkatDC(a,n/2)*pangkatDC(a,n/2));
    }
}
```

4. Perhatikan apakah sudah tidak ada kesalahan yang muncul dalam pembuatan class Pangkat
5. 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.

```
Scanner sc = new Scanner(System.in);
System.out.println("=======================================================");
System.out.print("Masukkan jumlah elemen yang ingin dihitung : ");
int elemen = sc.nextInt();
```

6. 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.

```
Pangkat [] png = new Pangkat[elemen];

for (int i = 0; i < elemen; i++) {
    png[i] = new Pangkat();
    System.out.print("Masukkan nilai yang akan dipangkatkan ke-"+(i+1)+" : ");
    png[i].nilai = sc.nextInt();
    System.out.print("Masukkan nilai pemangkat ke-"+(i+1)+" : ");
    png[i].pangkat = sc.nextInt();
}
```

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

```
System.out.println("==================================================");
System.out.println("Hasil Pangkat dengan Brute Force");
for (int i = 0; i < elemen; i++) {
    System.out.println("Nilai "+png[i].nilai+" pangkat "+png[i].pangkat+" adalah : "+png[i].pangkatBF(png[i].nilai,png[i].pangkat));
}
System.out.println("==================================================");
System.out.println("Hasil Pangkat dengan Divide and Conquer");
for (int i = 0; i < elemen; i++) {
    System.out.println("Nilai "+png[i].nilai+" pangkat "+png[i].pangkat+" adalah : "+png[i].pangkatDC(png[i].nilai,png[i].pangkat));
}
System.out.println("==================================================");
```

2.4.2 Verifikasi Hasil Percobaan

Pastikan output yang ditampilkan sudah benar

```
run:
==================================================
Masukkan jumlah elemen yang ingin dihitung : 2
Masukkan nilai yang akan dipangkatkan ke-1 : 6
Masukkan nilai pemangkat ke-1 : 2
Masukkan nilai yang akan dipangkatkan ke-2 : 4
Masukkan nilai pemangkat ke-2 : 3
==================================================
Hasil Pangkat dengan Brute Force
Nilai 6 pangkat 2 adalah : 36
Nilai 4 pangkat 3 adalah : 64
==================================================
Hasil Pangkat dengan Divide and Conquer
Nilai 6 pangkat 2 adalah : 36
Nilai 4 pangkat 3 adalah : 64
==================================================
BUILD SUCCESSFUL (total time: 10 seconds)
```

2.4.3 Pertanyaan

1. Jelaskan mengenai perbedaan 2 method yang dibuat yaitu `PangkatBF()` dan `PangkatDC()`!
2. Pada method `PangkatDC()` terdapat potongan program sebagai berikut:

```
if(n%2==1)//bilangan ganjil
    return (pangkatDC(a,n/2)*pangkatDC(a,n/2)*a);
else//bilangan genap
    return (pangkatDC(a,n/2)*pangkatDC(a,n/2));
```

   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!

2.5 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.

2.5.1 Langkah-langkah Percobaan

1. Pada paket minggu3. 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`

```
public int elemen;
public double keuntungan[];
public double total;
```

```
Sum(int elemen){
    this.elemen = elemen;
    this.keuntungan=new double[elemen];
    this.total = 0;
}
```

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

```
double totalBF(double arr[]){
    for (int i = 0; i < elemen; i++) {
        total = total + arr[i];
    }
    return total;
}
```

3. 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];
    else if(l<r){
     int mid=(l+r)/2;
     double lsum=totalDC(arr,l,mid-1);
     double rsum=totalDC(arr,mid+1,r);
     return lsum+rsum+arr[mid];
    }

    return 0;
}
```

4. 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`

```
Scanner sc = new Scanner(System.in);
System.out.println("================================================================");
System.out.println("Program Menghitung Keuntungan Total (Satuan Juta. Misal 5.9)");
System.out.print("Masukkan jumlah bulan : ");
int elm = sc.nextInt();
```

5. 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.

```
Sum sm = new Sum(elm);
System.out.println("================================================================");
for (int i = 0; i < sm.elemen; i++) {
    System.out.print("Masukkan untung bulan ke - "+(i+1)+" = ");
    sm.keuntungan[i] = sc.nextDouble();
}
```

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

```
System.out.println("============================================================");
System.out.println("Algoritma Brute Force");
System.out.println("Total keuntungan perusahaan selama " + sm.elemen + " bulan adalah = "+sm.totalBF(sm.keuntungan));
System.out.println("============================================================");
System.out.println("Algoritma Divide Conquer");
System.out.println("Total keuntungan perusahaan selama " + sm.elemen + " bulan adalah = "+sm.totalDC(sm.keuntungan, 0,sm.elemen-1));
```

2.5.2 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
================================================================
Program Menghitung Keuntungan Total (Satuan Juta. Misal 5.9)
Masukkan jumlah bulan : 5
================================================================
Masukkan untung bulan ke - 1 = 8.5
Masukkan untung bulan ke - 2 = 9.54
Masukkan untung bulan ke - 3 = 7.2
Masukkan untung bulan ke - 4 = 9.1
Masukkan untung bulan ke - 5 = 6
============================================================
Algoritma Brute Force
Total keuntungan perusahaan selama 5 bulan adalah = 40.339999999999996
============================================================
Algoritma Divide Conquer
Total keuntungan perusahaan selama 5 bulan adalah = 40.34
BUILD SUCCESSFUL (total time: 11 seconds)
```

2.5.3 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!

```
return lsum+rsum+arr[mid];
```

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!

2.6 Tugas

1. Buat program berdasarkan diagram class berikut ini!

NilaiAlgoritma
namaMhs: String nilaiTugas: int nilaiKuis: int nilaiUTS: int nilaiUAS: int
hitungTotalNilai(): double

   Nilai dihitung berdasarkan total Tugas 30%, Kuis 20%, UTS 20%, UAS 30%. Sesuaikan method jika harus memiliki parameter.
2. Sebagai lanjutan soal no. 1. Buat instansiasi *array of objek* pada class main untuk mengetahui nilai beberapa mahasiswa dalam sekaligus sekali perhitungan!
3. Melanjutkan soal no. 2 dengan algoritma Brute Force modifikasi program agar dapat mengetahui rata-rata nilai seluruh mahasiswa yang telah diinputkan untuk mata kuliah Algoritma.

   Contohnya :

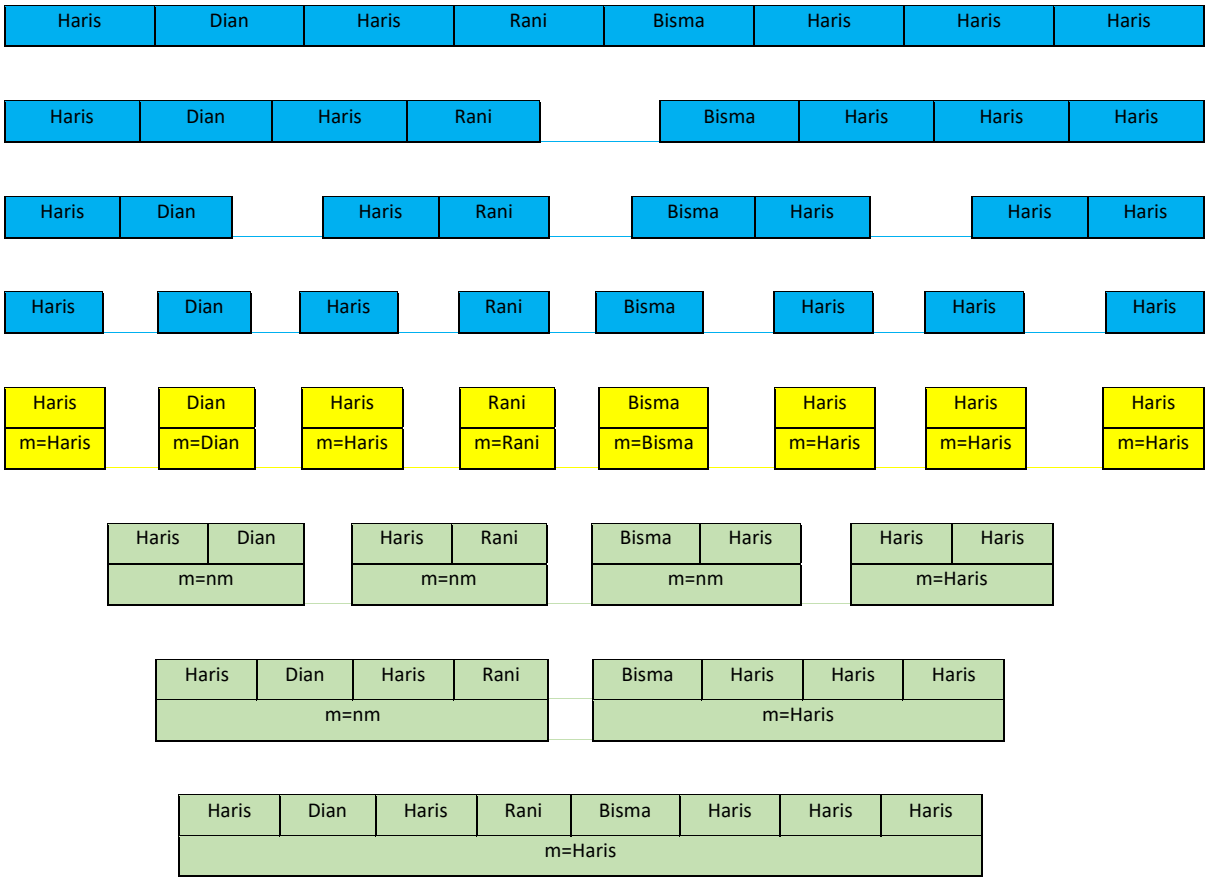
   Terdapat Total nilai mahasiswa berikut

Nama	Nilai Total Mata Kuliah Algoritma
Rani	86,5
Dani	72,5
Saraswati	89
Rata-rata	82,67

Maka rata-rata nilai total seluruh mahasiswa yang telah menempuh mata kuliah algoritma adalah 82,67

4. Suatu Perguruan Tinggi di kota Malang sedang mengadakan pemilihan suara untuk memilih ketua BEM tahun 2020. Jika jumlah suara yang terkumpul diumpamakan selalu genap. Maka dengan inputan kandidat terpilih, carilah mayoritas jumlah suara untuk masing-masing kandidat. Buatlah class diagram dan program menggunakan algoritma Divide and Conquer dari studi kasus tersebut! (Jumlah elemen array dan hasil pemilhan suara merupakan inputan user).

   Contoh : Hasil pemilihan suara sebagai berikut (m adalah mayoritas, nm adalah no mayoritas)



Keterangan : Warna Biru adalah proses divide, warna kuning dimulainya proses conquer, warna hijau dimulainya proses combine

5. Bagaimana jika jumlah suara ganjil? apakah harus ada perbaikan program?jika iya, perbaiki program untuk studi kasus no 4. Jika jumlah suara yang terkumpul tidak selalu genap!
6. Modifikasi program tentang rata-rata nilai mata kuliah algoritma dengan menggunakan algoritma Divide and Conquer!