TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

POLYNOMIAL MULTIPLICATION
BRUTE FORCE
VS
DIVIDE AND CONQUER

Disusun oleh:

Jones Napoleon
13518086

Mata Kuliah Strategi Algoritma
Semester 2 Tahun Ajaran 2019/2020

Prodi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung 2020

DAFTAR ISI

DAFTAR ISI

BAB I

POLYNOMIAL MULTIPLICATION, ALGORITMA BRUTE FORCE DAN DIVIDE AND CONQUER

1.1. *OVERVIEW*

Polinomial (*polynomial*) merupakan pernyataan matematika yang melibatkan jumlahan perkalian pangkat dalam satu atau lebih variabel dengan koefisien. Pangkat tertinggi suatu polinomial disebut dengan orde atau derajat. Pada program ini, polynomial yang digunakan diasumsikan hanya bervariabel satu (x).

Secara dasar, program menerima *input* berupa jumlah derajat (anggap *N* buah), kemudian program akan menggenerasikan dua polinomial dengan derajat yang sama sebesar N. Setelah itu, program akan memproses perkalian kedua polinomial tersebut dengan cara *bruteforce* dan *divide and conquer*. Program ini juga akan membandingkan nilai efisiensi waktu kedua algoritma dengan fungsi waktu dari C dan perbandingan jumlah perkalian dan penjumlahan kedua metode perkalian polinomial.

1.2. STRUKTUR DATA

Program dibuat dengan model berupa *array* yang dimulai dari indeks 0 sampai N (dimana N merupakan derajat polinomial tersebut). Masing-masing elemen berisi koefisien dari elemen ke indeks array tersebut.

```
#define IdxMin 0
#define IdxUndef -999

typedef struct {
  float *TI;
  int Degree;
  int MaxEl;
} TabFLOAT;
```

Gambar 1. Struktur Data polinomial

```
#define Degree(T) (T).Degree
#define TI(T) (T).TI
#define Elmt(T, i) (T).TI[(i)]
#define MaxEl(T) (T).MaxEl
```

Gambar 2. Selektor polinomial

1.3. ALGORITMA BRUTE FORCE

Program setelah menyimpan dua buah polinomial (sebut polinomial 1 dan 2) akan selanjutnya membuat suatu polinomial kosong (polinomial 3) dengan derajat bernilai penjumlahan derajat polinomial 1 dan polinomial 2. Polinomial 3 ini akan menjadi polinomial hasil perkalian polinomial 1 dan polinomial 2.

Selanjutnya, setiap indeks di polinomial 1 akan diiterasikan masing-masing dengan setiap indeks di polinomial 2, dan penjumlahan kedua elemen indeks polinomial 1 dan 2 tesebut akan ditambahkan ke elemen indeks polinomial 3.

Misalkan ada *N* derajat. Program akan mulai dari elemen indeks ke-0 di polinomial 1 dan elemen indeks ke-0 di polinomial 2, penjumlahan kedua elemen tersebut kemudian ditambahkan pada elemen indeks ke-(0+0) di polinomial 3. Selanjutnya, elemen indeks ke-0 di polinomial 1 dan elemen indeks ke-1 di polinomial 2, penjumlahan kedua elemen tersebut kemudian ditambahkan pada elemen indeks ke-(0+1) di polinomial 3. Iterasi tersebut berulang sampai indeks ke-N di polinomial 1 dan polinomial 2.

Karena dilakukan iterasi secara kombinatorial dari kedua polinomial, kompleksitas *big-O*-nya adalah $O(n^2)$ karena untuk setiap indeks di polinomial 1 dilakukan pemasangan dengan semua indeks di polinomial 2.

1.4. ALGORITMA DIVIDE AND CONQUER

Program setelah menyimpan dua buah polinomial (sebut polinomial 1 dan 2, masing-masing berderajat A dan B - A dan B akan selalu sama) akan selanjutnya dilakukan pengecekan apakah kedua polinomial tersebut sudah berderajat lebih kecil dari 2 (basis). Jika sudah, hasil akan dikembalikan secara *bruteforce* yakni elemen ketiga (indeks 2) polinomial 3 adalah hasil perkalian elemen kedua (indeks 1) polinomial 1 dan polinomial 2, elemen kedua polinomial 3 adalah hasil penjumlahan perkalian elemen pertama polinomial 1 dan elemen kedua polinomial 2 dengan perkalian elemen kedua polinomial 1 dan elemen pertama polinomial 2, dan elemen pertama polinomial 3 adalah hasil perkalian elemen pertama polinomial 1 dan elemen pertama polinomial 2.

Jika kedua polinomial derajatnya diatas 1, maka dilakukan proses rekurensi dengan membagi polinomial 1 menjadi polinomial 11 dan polinomial 12 dan polinomial 2 menjadi polinomial 21 dan polomial 22. Polinomial 11 adalah polinomial 1 yang dipotong hanya berderajat (A *div* 2) – 1, begitu pula untuk polinomial 21 terhadap polinomial 2. Sedangkan polinomial 12 adalah sisa hasil potongan polinomial 1 terhadap polinomial 11, hanya saja indeks polinomial 12 bergeser sehingga dimulai dari 0, dan begitu pula berlaku untuk polinomial 22 dan polinomial 2.

Selanjutnya, anggap X adalah perkalian polinomial 11 dan 21, Z perkalian polinomial 12 dan 22, dan Y adalah perkalian dari penjumlahan polinomial 11 dan 12 dan penjumlahan polinomial 21 dan 22. Hasil dari perkalian polinomial tersebut adalah X ditambah dengan Z yang digeser sejauh (A *div* 2) dikali 2 dan (Y dikurangi X dan Z) dan kemudian digeser sejauh (A *div* 2). Karena prosedur perkalian hanya dipanggil tiga kali, dan bukan empat, maka algoritma ini kompleksitas *big-O*-nya adalah $O(n^{\log 3})$.

BAB II

IMPLEMENTASI KODE PROGRAM DALAM BAHASA C

2.1. HEADER *ARRAY.H*

Di bawah ditampilkan keempat prosedur yang ada pada header *array.h*

```
void MakeEmpty(TabFLOAT *T, int maxel);
void BacaIsi(TabFLOAT *T, int degree);
void TulisIsi(TabFLOAT T);
void Clear(TabFLOAT *C);
```

Gambar 3. Prosedur pada header *point.h*

Prosedur **MakeEmpty** merupakan prosedur *generic* yang hanya melakukan metode seperti bagaimana semestinya (tidak ditampilkan lagi karena merupakan bentuk abstraksi biasa dari program pada mata kuliah Algoritma dan Struktur Data).

Prosedur **BacaIsi** akan meng-**MakeEmpty**-kan suatu polinomial berderajat *degree* dan mengacakkan tanda dan nilai suatu *float* pada setiap iterasi elemen polinomial dan meng-*assign*-kannya pada polinomial tersebut.

```
void BacaIsi(TabFLOAT *T, int N){
  MakeEmpty(T, N + 3);
  Degree(*T) = N;
  for(int i = IdxMin; i <= Degree(*T); i++){
    float value = rand() % 100;
    float sign = rand() % 100;
    if(sign == 1){
      Elmt(*T, i) = value * -1;
    }
    else {
      Elmt(*T, i) = value;
    }
  }
}
```

Gambar 4. Prosedur **BacaIsi** pada header *array.h*

Prosedur **TulisIsi** berfungsi untuk menampilkan *array* seperti pola polinomial di matematika pada umumnya, sedangkan prosedur **Clear** berfungsi untuk menge-*set* elemen di tiap elemen *array* dengan 0. Menjadikan polinomial bernilai 0. Karena kedua prosedur hanya berupa pemograman logika biasa, maka implementasi program tidak lagi ditampilkan.

2.2. HEADER *BRUTEFORCE.H*

Sesuai metode yang ditulis diatas, metode *bruteforce* adalah sebagai berikut.

```
void BruteForce(TabFLOAT *C, TabFLOAT *A, TabFLOAT *B, int *countAddBruteforce, int *countMultipleBruteforce){
  *countAddBruteforce = 0;
  *countMultipleBruteforce = 0;
  for(int a = IdxMin; a <= Degree(*A); a ++){
    for(int b = IdxMin; b <= Degree(*B); b ++){
      Elmt(*C, a + b) += Elmt(*A, a) * Elmt(*B, b);
      *countAddBruteforce = *countAddBruteforce + 1;
      *countMultipleBruteforce = *countMultipleBruteforce + 1;
    }
  }
}
```

Gambar 5. Implementasi prosedur *bruteforce* pada *bruteforce.h*

Polinomial **A** dan **B** masing-masing adalah polinomial *input* dan polinomial **C** adalah hasil perkalian polinomial **A** dan **B** (polinomial hasil). ***countAddBruteforce*** berisi jumlah perkalian yang dilakukan algoritma *bruteforce* sedangkan ***countMultipleBruteforce*** menyimpan jumlah perkaliannya.

Sesuai dengan algoritma bruteforce, nilai ***countAddBruteforce*** dan ***countMultipleBruteforce*** akan selalu bernilai sama karena untuk setiap *assignment* di polinomial hasil, *assignment*-nya menyangkut penjumlahan dari perkalian kedua elemen di polinomial **A** dan **B**.

2.3. HEADER *DIVIDEANDCONQUER.H*

Berikut ditampilkan beberapa prosedur dan fungsi di *header divideandconquer.h*

```
void DivideConquer(TabFLOAT *D, TabFLOAT *A, TabFLOAT *B, int *countAddDivideConquer, int *countMultipleDivideConquer);
TabFLOAT Divide(TabFLOAT *A, TabFLOAT *B, int *countAddDivideConquer, int *countMultipleDivideConquer);

void DivideTwo(TabFLOAT *A, TabFLOAT *Z1, TabFLOAT *Z2);
TabFLOAT Shift(TabFLOAT F, int shift);

TabFLOAT MinusThree(TabFLOAT A, TabFLOAT B, TabFLOAT C, int *countAddDivideConquer, int *countMultipleDivideConquer);
TabFLOAT AddTwo(TabFLOAT A, TabFLOAT B, int *countAddDivideConquer, int *countMultipleDivideConquer);
TabFLOAT AddThree(TabFLOAT A, TabFLOAT B, TabFLOAT C, int *countAddDivideConquer, int *countMultipleDivideConquer);
```

Gambar 6. Fungsi dan prosedur pada *header divideandconquer.h*

Prosedur **DivideConquer** akan menjadi prosedur berabstraksi tertinggi (yang akan dipanggil dari *main.c* secara langsung). Parameter **D** berupa polinomial hasil, **A** berupa polinomial *input* 1, **B** berupa polinomial *input 2*, sedangkan ***countAddDivideConquer*** dan ***countMultipleDivideConquer*** masing-masing berupa nilai penjumlahan dan perkalian pada prosedur ***DivideConquer.***

```
void DivideConquer(TabFLOAT *D, TabFLOAT *A, TabFLOAT *B ,int *countAddDivideConquer, int *countMultipleDivideConquer){
  *countMultipleDivideConquer = 0;
  *countAddDivideConquer = 0;
  *D = Divide(A, B, countAddDivideConquer, countMultipleDivideConquer);
}
```

Gambar 7. Implementasi **DivideConquer** yang menjadi *interface* terluar bagi *main.c* untuk melakukan perkalian polinom secara *divide and conquer*.

Selanjutnya pada **DivideConquer**, dipanggil fungsi **Divide** yang akan menjadi implementasi utama perkalian polinomial secara *divide and conquer*. Berikut adalah implementasi program sesuai dengan yang telah diutarakan di bagian **ALGORITMA DIVIDE AND CONQUER**.

```
TabFLOAT Divide(TabFLOAT *A, TabFLOAT *B, int *countAddDivideConquer, int *countMultipleDivideConquer){
  TabFLOAT D;
  MakeEmpty(&D, Degree(*A) + Degree(*B) + 5);
  Degree(D) = Degree(*A) + Degree(*B);
  Clear(&D);

  if(Degree(*A) <= 1 && Degree(*B) <= 1){
    Elmt(D, 0) = Elmt(*A, 0) * Elmt(*B, 0);
    Elmt(D, 1) = Elmt(*A, 0) * Elmt(*B, 1) + Elmt(*A, 1) * Elmt(*B, 0);
    Elmt(D, 2) = Elmt(*A, 1) * Elmt(*B, 1);
    *countAddDivideConquer = *countAddDivideConquer + 1;
    *countMultipleDivideConquer = *countMultipleDivideConquer + 4;
    return D;
  }

  else {
    TabFLOAT Z1, Z2, Z3, Z4;
    TabFLOAT DC1, DC2, DC21, DC22, DC3;

    DivideTwo(A, &Z1, &Z2);
    DivideTwo(B, &Z3, &Z4);

    int fullDegree = (Degree(*A) / 2 ) * 2;
    int halfDegree = Degree(*A) / 2;

    DC1 = Divide(&Z1, &Z3, countAddDivideConquer, countMultipleDivideConquer);
    DC3 = Divide(&Z2, &Z4, countAddDivideConquer, countMultipleDivideConquer);
    DC21 = AddTwo(Z1, Z2, countAddDivideConquer, countMultipleDivideConquer);
    DC22 = AddTwo(Z3, Z4, countAddDivideConquer, countMultipleDivideConquer);
    DC2 = MinusThree(Divide(&DC21, &DC22, countAddDivideConquer, countMultipleDivideConquer), DC1, DC3, countAddDivideConquer, countMultipleDivideConquer);

    return AddThree( DC1, Shift(DC2, halfDegree), Shift(DC3, fullDegree), countAddDivideConquer, countMultipleDivideConquer);
  }

}
```

Gambar 8. Implementasi **Divide** yang melakukan implementasi utama perkalian polinom secara *divide and conquer.*

Pada implementasi **Divide** masa rekursif, program awalnya memanggil **DivideTwo**. Sesuai dengan namanya, program ini akan membagi polinomial awal menjadi dua upa-polinomial dimana polinomial pertama adalah polinomial awal yang dipotong sampai berderajat awal div 2 dikurang 1 dan polinomial kedua adalah polinomial sisa yang digeser sehingga indeks dimulai dari nol.

```
void DivideTwo(TabFLOAT *F, TabFLOAT *Z1, TabFLOAT *Z2){

  MakeEmpty(Z1, (Degree(*F) - 1) / 2 + 5);
  Degree(*Z1) = Degree(*F) / 2 - 1;
  Clear(Z1);

  MakeEmpty(Z2, Degree(*F) - Degree(*Z1) + 5);
  Degree(*Z2) = Degree(*F) - Degree(*F) / 2;
  Clear(Z2);

  for(int i = IdxMin; i <= Degree(*Z1); i ++){
    Elmt(*Z1, i) = Elmt(*F, i);
  }

  int countZ2 = 0;

  for(int i = Degree(*Z1) + 1; i <= Degree(*F); i ++){
    Elmt(*Z2, countZ2) = Elmt(*F, i);
    countZ2 ++;
  }

}
```

Gambar 9. Implementasi **DivideTwo** yang membagi polinom **F** menjadi polinom **Z1** dan polinom **Z2**.

fullDegree dan ***halfDegree*** masing-masing berupa pergeseran indeks dari polinomial **DC3** dan pergeseran indeks dari polinomial **DC2**

DC1 merupakan perkalian (**Divide**) **Z1** dan **Z3**, **DC3** merupakan perkalain **Z2** dan **Z4**, sedangkan **DC2** merupakan pengurangan dari perkalian **DC21 dan DC22** terhadap **DC1** dan **DC3**. Sedangkan, **DC21** dan **DC22** masing-masing merupakan penjumlahan dari **Z1** dan **Z2** dan penjumlahan dari **Z3** dan **Z4**

Proses pengurangan polinom pada konteks algoritma ini melibatkan tiga polinom. Argumen polinom pertama (**A**) pada **MinusThree** merupakan polinom dengan derajat tertinggi, argumen kedua (**B**) merupakan polinom dengan derajat terendah, sedangkan argument ketiga (**C**) merupakan polinom dengan derajat di tengah. Implementasi algoritma **MinusThree** adalah sebagai berikut.

```
TabFLOAT MinusThree(TabFLOAT A, TabFLOAT B, TabFLOAT C, int *countAddDivideConquer, int *countMultipleDivideConquer){
  TabFLOAT X;
  MakeEmpty(&X, Degree(A) + 5);
  Degree(X) =  Degree(A);
  Clear(&X);
  for(int i = IdxMin; i <= Degree(B); i ++){
    Elmt(X, i) = Elmt(A, i) - Elmt(B, i) - Elmt(C, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  for(int i = Degree(B) + 1; i <= Degree(C); i ++){
    Elmt(X, i) = Elmt(A, i) - Elmt(C, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  for(int i = Degree(C) + 1; i <= Degree(A); i ++){
    Elmt(X, i) = Elmt(A, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  return X;
}
```

Gambar 10. Implementasi **MinusThree**

Program rekursif pada **Divide** juga mengembalikan polinomial yang telah ditambahkan dengan tiga argumen (yang membuat kompleksitas *big-O*-nya hanya $O(n^{\log 3})$). Penjumlahan polinomial ini diimplementasikan melalui fungsi **AddThree** dimana derajat **A** < derajat **B** < derajat **C**.

```
TabFLOAT AddThree(TabFLOAT A, TabFLOAT B, TabFLOAT C, int *countAddDivideConquer, int *countMultipleDivideConquer){
  TabFLOAT X;
  MakeEmpty(&X, Degree(C) + 5);
  Degree(X) =  Degree(C);
  Clear(&X);
  for(int i = IdxMin; i <= Degree(A); i ++){
    Elmt(X, i) = Elmt(A, i) + Elmt(B, i) + Elmt(C, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  for(int i = Degree(A) + 1; i <= Degree(B); i ++){
    Elmt(X, i) = Elmt(B, i) + Elmt(C, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  for(int i = Degree(B) + 1; i <= Degree(C); i ++){
    Elmt(X, i) = Elmt(C, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  return X;
}
```

Gambar 11. Implementasi **AddThree**

Untuk mencari **DC21** dan **DC22**, program menambahkan dua polinom, secara berturut-turut **Z1** dengan **Z2** dan **Z3** dengan **Z4**. Penambahan kedua polinom ini diimplementasikan melalui fungsi **AddThree**

```
TabFLOAT AddTwo(TabFLOAT A, TabFLOAT B, int *countAddDivideConquer, int *countMultipleDivideConquer){
  TabFLOAT X;
  MakeEmpty(&X, Degree(B) + 5);
  Degree(X) =  Degree(B);
  Clear(&X);
  for(int i = IdxMin; i <= Degree(A); i ++){
    Elmt(X, i) = Elmt(A, i) + Elmt(B, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  for(int i = Degree(A) + 1; i <= Degree(B); i ++){
    Elmt(X, i) = Elmt(B, i);
    *countAddDivideConquer = *countAddDivideConquer + 1;
  }
  return X;
}
```

Gambar 12. Implementasi **AddTwo**

2.4. *MAIN PROGRAM*

Program utama berjalan sebagai berikut. Dimulai dengan membaca sebuah *integer input* (sebut N) dari pengguna. Kemudian program akan menggenerasikan dua polinom berbeda dengan derajat yang sama sebesar N, dan menampilkan keduanya di *console*.

```
int main(){
  time_t t;
  clock_t startBruteforce, endBruteforce, startDivideConquer, endDivideConquer;
  srand((unsigned)time(&t));

  TabFLOAT A, B, C, D; int N;
  printf("Input polinom degree: ");
  scanf("%d", &N);

  printf("Polinom pertama\n"); Spread;
  BacaIsi(&A, N);
  TulisIsi(A); Spread; Nl;

  printf("Polinom kedua\n"); Spread;
  BacaIsi(&B, N);
  TulisIsi(B); Spread; Nl;
```

Gambar 13. Deklarasi dan generasi polinom kedua polinom.

Setelah itu, program akan memulai waktu perhitungan perkalian dengan metode *bruteforce* dari memulai mengosongkan *array* polinom hasil, menge-*set* derajat polinom hasil sebesar N + N, hingga memanggil fungsi **BruteForce** dari *library bruteforce.h*.

Selain metode *bruteforce*¸ program selanjutnya menggunakan metode **DivideConquer** dari *divideandconquer.h* dengan proses pemulaian waktu perhitungan, deklarasi derajat, dan pengosongan *array* yang sama dengan metode pra-**BruteForce**.

```
// START BRUTEFORCE PROCESS
startBruteforce = clock();
int countAddBruteforce, countMultipleBruteforce;
MakeEmpty(&C, Degree(A) + Degree(B) + 5);
Degree(C) = Degree(A) + Degree(B);
Clear(&C);
BruteForce(&C, &A, &B, &countAddBruteforce, &countMultipleBruteforce);
endBruteforce = clock();

// START DIVIDE AND CONQUER PROCESS
startDivideConquer = clock();
int countAddDivideConquer, countMultipleDivideConquer;
MakeEmpty(&D, Degree(A) + Degree(B) + 5);
Degree(D) = Degree(A) + Degree(B);
Clear(&D);
DivideConquer(&D, &A, &B, &countAddDivideConquer, &countMultipleDivideConquer);
endDivideConquer = clock();
```

Gambar 14. Proses pengisian polinom hasil dengan metode **BruteForce** dan **DivideConquer**.

Setelah kedua *array* dikalkulasi, program akan menampilkan banyak penambahan, perkalian, dan hasil perkalian polinom akhir di *console*. Juga tentunya menampikan lama waktu yang dimakan kedua proses perkalian polinom. Hasil analisis sementara dengan input N kecil dari program menyatakan bahwa waktu yang dimakan oleh proses **DivideConquer** lebih lama dibanding **BruteForce**.

```
  printf("Hasil kali kedua polinom secara bruteforce\n"); Spread;
  printf("Jumlah pertambahan elemen di perkalian kedua polinom secara Bruteforce: %d\n", countAddBruteforce);
  printf("Jumlah perkalian elemen di perkalian kedua polinom secara Bruteforce: %d\n", countMultipleBruteforce); Spread;
  TulisIsi(C); Spread; Nl;

  printf("Hasil kali kedua polinom secara divide and conquer\n"); Spread;
  printf("Jumlah pertambahan elemen di perkalian kedua polinom secara DivideConquer: %d\n", countAddDivideConquer);
  printf("Jumlah perkalian elemen di perkalian kedua polinom secara DivideConquer: %d\n", countMultipleDivideConquer); Spread;
  TulisIsi(D); Spread; Nl;

  printf("Bruteforce program was running for %.3f miliseconds\n", ((float)endBruteforce - startBruteforce));
  printf("Divide and Conquer program was running for %.3f miliseconds\n", ((float)endDivideConquer - startDivideConquer));

  return 0;
}
```

Gambar 15. Proses pencetekan hasil proses perkalian polinomial ke *console*.

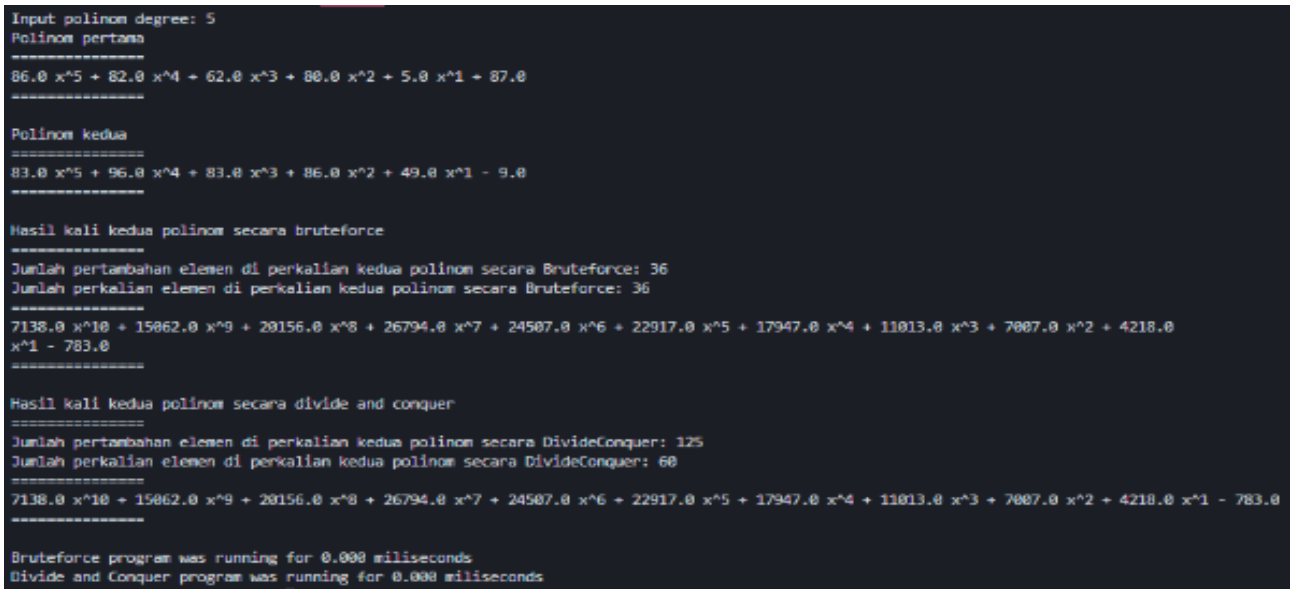
NB:

- **Spread** merupakan deklarasi makro buatan untuk mencetak "==========" di *console*.
- **Nl** merupakan deklarasi makro buatan untuk pindah ke *new line* di *console*.
- Program dengan masukan derajat (N) kecil akan memakan waktu < 0.5 ms.
- Penjumlahan lebih banyak terjadi pada metode **DivideConquer**.

BAB III

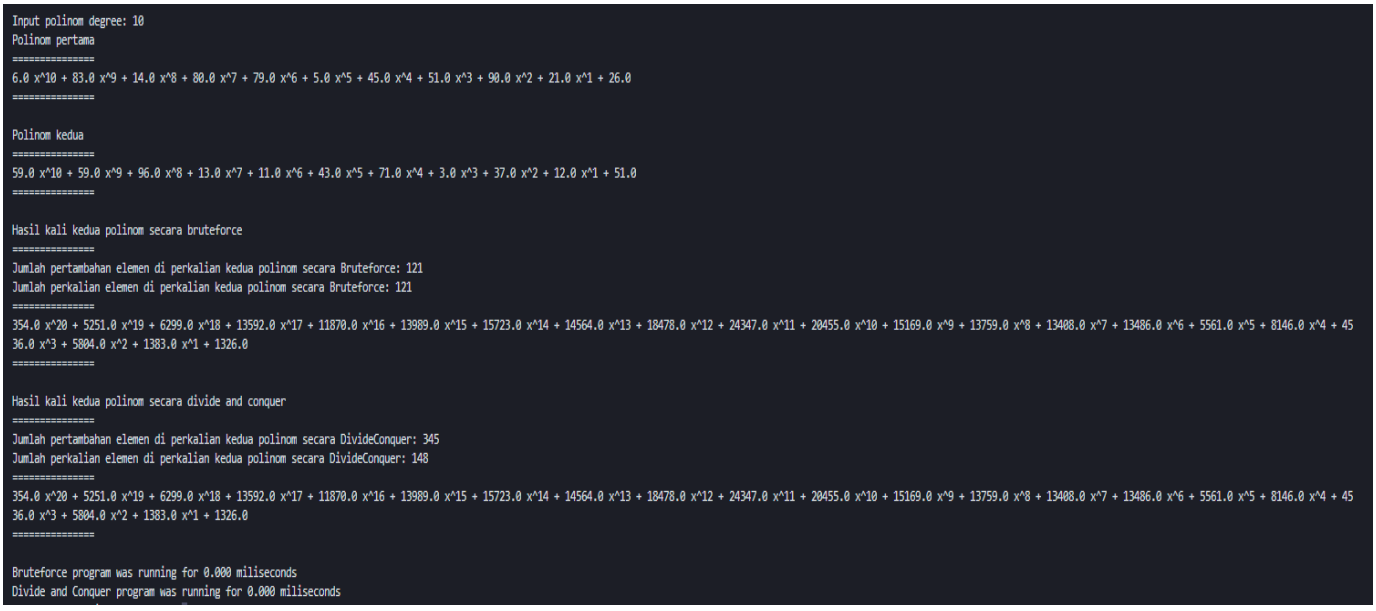
SCREENSHOT HASIL PROGRAM

3.1. INPUT BERJUMLAH 5



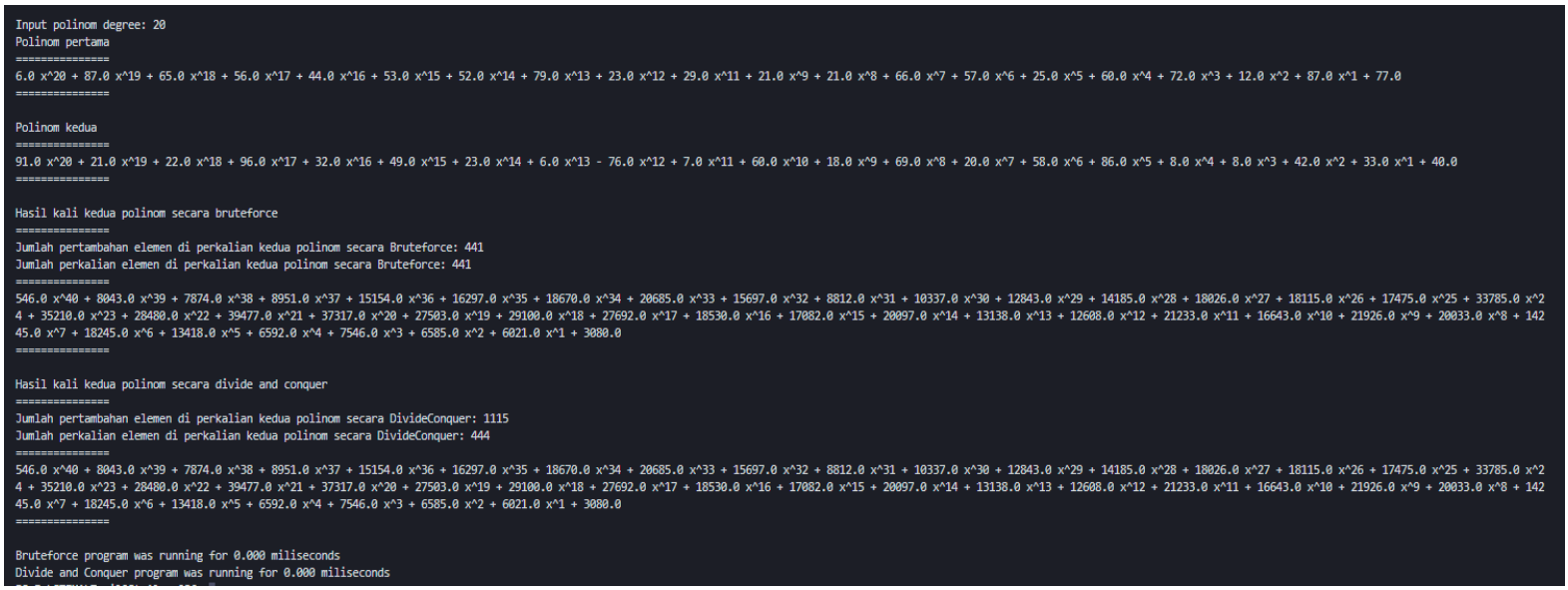
Gambar 16. Hasil eksekusi program dengan masukan 5 derajat.

3.2. INPUT BERJUMLAH 10



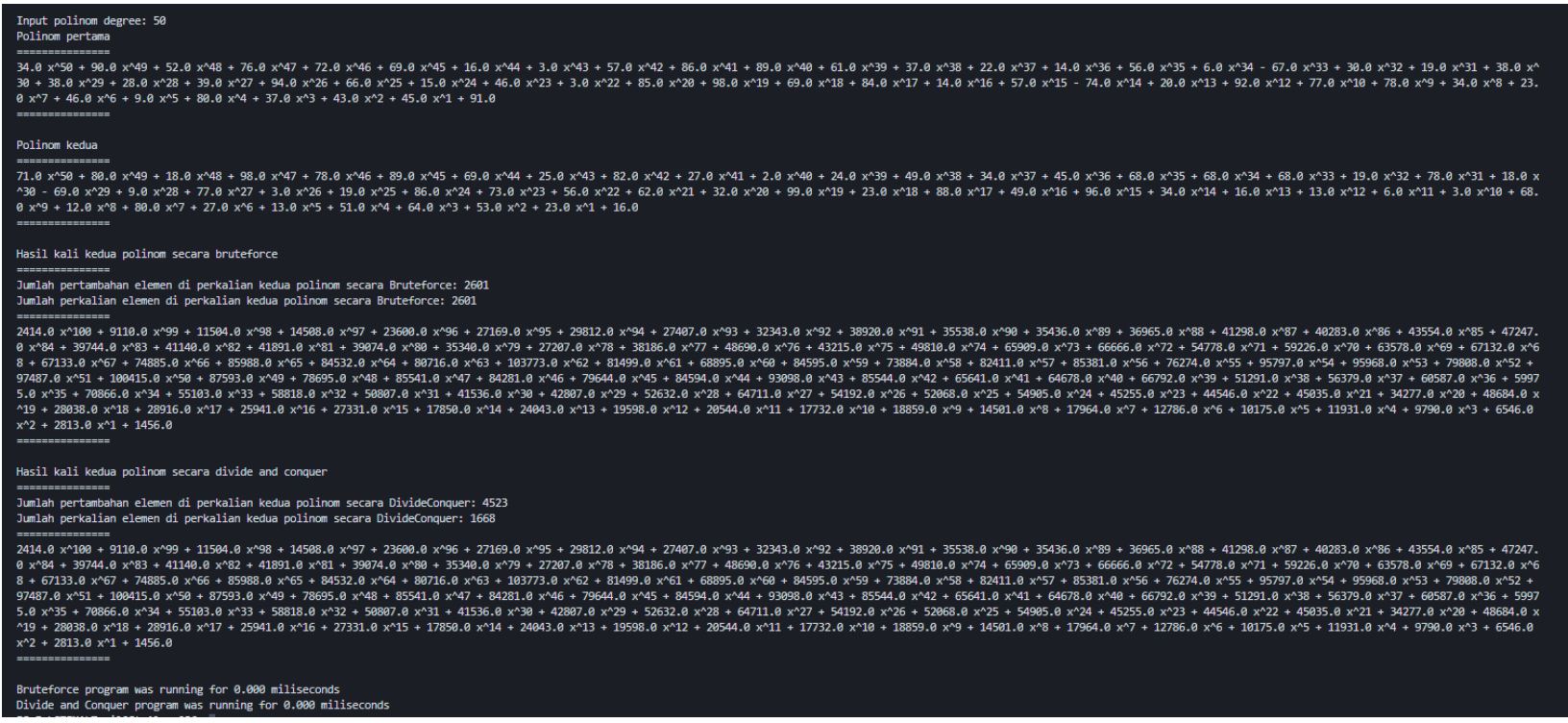
Gambar 17. Hasil eksekusi program dengan masukan 10 derajat.

3.3. INPUT BERJUMLAH 20



Gambar 18. Hasil eksekusi program dengan masukan 20 derajat.

3.4. INPUT BERJUMLAH 50



Gambar 19. Hasil eksekusi program dengan masukan 50 derajat.

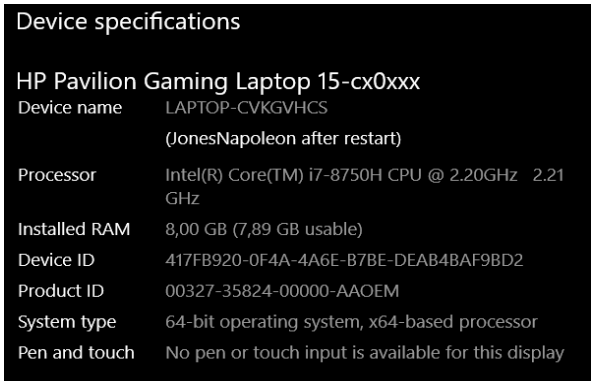
BAB IV

LAIN-LAIN

4.1. PENILAIAN INDIVIDU

No	Poin	Ya	Tidak
1.	Program berhasil dikompilasi	√	
2.	Program berhasil running	√	
3.	Program dapat menerima input dan menuliskan output.	√	
4.	Luaran sudah benar untuk semua n	√	

4.2. SPESIFIKASI PERANGKAT KERAS YANG DIGUNAKAN



Gambar 20. Spesifikasi Laptop *author* (Jones Napoleon)