**LAPORAN TUGAS KECIL 2**

**MEMBUAT PUSTAKA UNTUK PERKALIAN POLINOM DENGAN ALGORITMA DIVIDE AND CONQUER**

**IF2211 Strategi Algoritma**

**­­**

Oleh:

**Daffa Pratama Putra 13518033**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2020**

# **BAB I STRATEGI ALGORITMA**

1. **Algoritma *Brute Force***

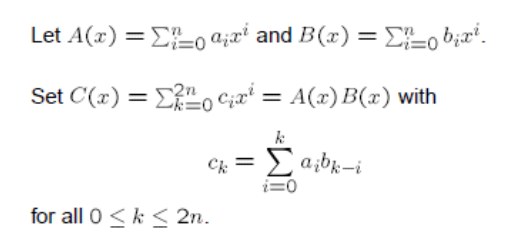
Algoritma *brute force* adalah salah satu algoritma yang digunakan untuk menyelesaikan persoalan komputasi. Pendekatan yang dilakukan oleh algoritma *brute force* adalah secara *straight forward* atau secara langsung. Biasanya algoritma ini digunakan untuk menyelesaikan persoalan yang sederhana, langsung, dan jelas persoalannya. Contoh penggunaan algoritma *brute force* adalah untuk mencari elemen terbesar/terkecil pada senarai, mencari elemen yang ada pada senarai, menghitung faktorial, dan lain sebagainya, termasuk untuk menyelesaikan persoalan perkalian polinomial.

Perkalian polinomial dapat diselesaikan dengan algoritma *brute force*. Algoritma *brute force* ini sama seperti mengalikan polinomial di kehidupan nyata, yaitu mengalikan semua koefisien polinom pertama dengan koefisien polinom kedua. Misalkan dua polinom berderajat n sebagai berikut :

A(x) = a0 + a1x + a2x2 + … + an-1xn-1 + anxn

B(x) = b0 + b1x + b2x2 + … + bn-1xn-1 + bnxn

maka secara umum perkalian kedua polinom tersebut adalah sebagai berikut:



sebagai contoh seperti di bawah ini :



Dengan menggunakan algoritma *brute force*, persoalan mengalikan dua polinom memiliki kompleksitas waktu sebesar O(n2). Kompleksitas waktu tersebut didapatkan karena pada algoritma *brute force* akan mengalikan sebanyak n derajat polinom terhadap n derajat polinom. Sehingga akan ada perkalian sebanyak n2, maka didapatkan kompleksitas waktu sebesar O(n2).

1. **Algoritma *Divide and Conquer***

Algoritma *divide and conquer* adalah salah satu strategi algoritma yang menyelesaikan persoalan dengan cara membagi persoalan yang besar menjadi persoalan yang lebih kecil, sehingga lebih mudah diselesaikan. Langkah yang dilakukan dalam algoritma *divide and conquer* ada 3 yaitu :

* *Divide* : membagi persoalan besar menjadi sub-persoalan yang lebih kecil
* *Conquer* : menyelesaikan sub-persoalan dengan konsep rekursif
* *Combine* : menggabungkan solusi masing-masing sub-persoalan, sehingga membentuk kembali solusi semula

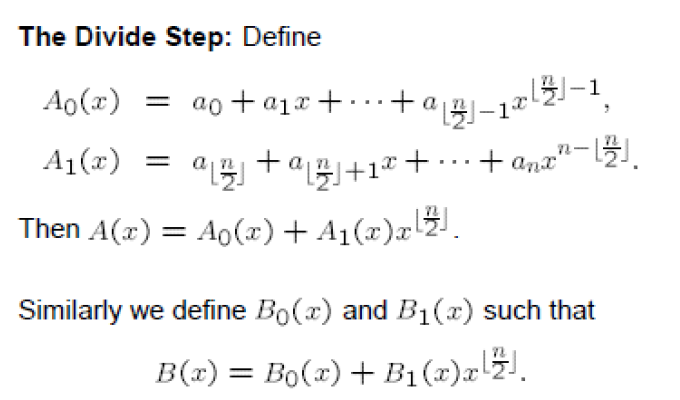
Biasanya objek yang dibagi menjadi sub-persoalan adalah berupa larik (*array*) berukuran n. Tiap sub-persoalan memiliki karakteristik yang sama dengan persoalan awal, sehingga sub-solusi dapat digunakan untuk memecahkan persoalan awal. Maka dari itu solusi dari algoritma *divide and conquer* dapat dicari dengan cara rekursif. Persoalan yang dapat diselesaikan oleh algoritma *divide and conquer* antara lain perpangkatan, mengurutkan larik (*quick sort, merge sort, insertion sort,* dan *selection sort*), dan perkalian polinom.

Penyelesaian persoalan perkalian polinom dengan algoritma *divide and conquer* mengadopsi algoritma Karatsuba. Algoritma Karatsuba dinilai lebih cepat dan efektif dalam menyelesaikan persoalan mengalikan dua polinom, yakni dengan kompleksitas waktu sebesar O() yang lebih cepat dibanding algoritma *brute force*. Pada algoritma *divide and conquer*, diasumsikan kedua polinom memiliki derajat polinom yang sama. Misalkan kedua polinom A dan polinom B sebagai berikut :

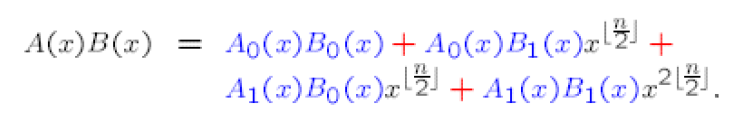
A(x) = a0 + a1x + a2x2 + … + an-1xn-1 + anxn

B(x) = b0 + b1x + b2x2 + … + bn-1xn-1 + bnxn

Maka bagi kedua polinom tersebut menjadi 4 bagian, misalkan A0­, A1, B0, dan B1. Pembagian polinom dilakukan tepat di tengah suku ke-n/2. Lebih jelasnya sebagai berikut :



Dengan begitu, kita dapat menulis perkalian polinom A terhadap polinom B sebagai



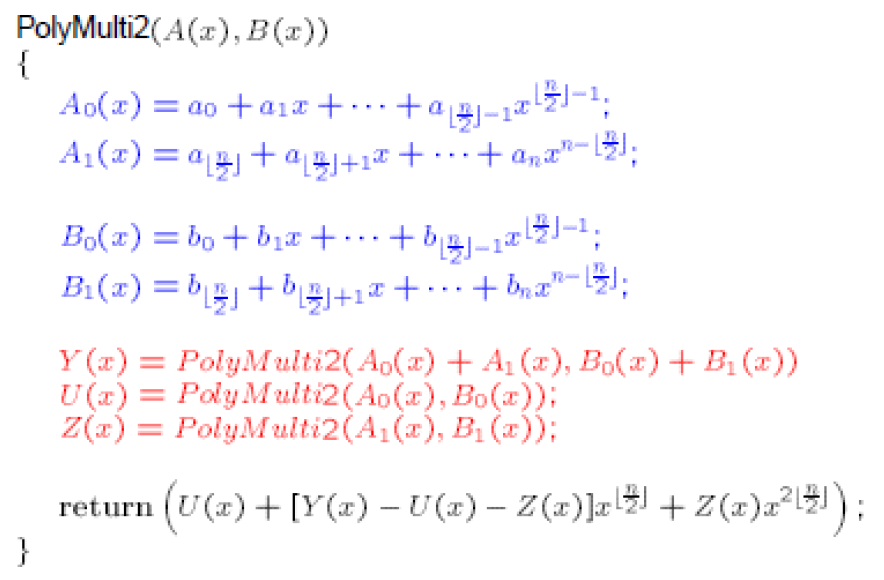
Dapat dilihat dari suku ke-2 dan ke-3 dapat digabung menjadi (A0B1 + A1B0), sehingga dengan melakukan manipulasi didapatkan

(A0B1 + A1B0) = (A0+A1)(B0+B1) – A0B0 – A1B1

sehingga dapat disusun keempat sub-polinom tersebut menjadi polinom U = A0B0, polinom Y = (A0+A1)(B0+B1), dan polinom Z = A1B1. Maka solusi perkalian polinom tersebut menjadi



Jika ditulis secara keseluruhan menjadi



# **BAB II KODE PROGRAM**

Kode program yang dibuat menggunakan bahasa C++ dengan memanfaatkan fitur kelas. Berikut adalah kode program untuk pustaka algoritma *Bruteforce* :

Kode program dari file header “Bruteforce.hpp”

|  |
| --- |
| #ifndef \_\_BRUTE\_\_FORCE\_\_HPP\_\_  #define \_\_BRUTE\_\_FORCE\_\_HPP\_\_  #include <iostream>  #include <time.h>  using namespace std;  class Polinom {      private :          int \*coef;          int degree;          static int numOp;      public  :          // Constructor          Polinom();          Polinom(int degree);          // Copy constructor          Polinom(const Polinom& p);          // Destructor          ~Polinom();          // Operator assignment          Polinom& operator=(const Polinom& p);          // Getter setter          int getCoef(int idx) const;          int getDegree() const;          static int getNumOp();          void setCoef(int idx, int val);          void setDegree(int degree);          // Other operator          Polinom operator+(const Polinom& A);          Polinom operator-(const Polinom& A);          Polinom operator\*(const Polinom& A);          // Other method          void print();          void inputCoef();          void inputRandom();  };  #endif |

Kode program dari file “Bruteforce.cpp”

|  |
| --- |
| #include "Bruteforce.hpp"  using namespace std;  // Static init  int Polinom::numOp = 0;  // Constructor  Polinom::Polinom() {      this->degree = 0;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = 0;      }  }  Polinom::Polinom(int degree) {      if (degree < 0) {          this->degree = 0;      }      else {          this->degree = degree;      }      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = 0;      }  }  // Copy constructor  Polinom::Polinom(const Polinom& p) {      this->degree = p.degree;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = p.coef[i];      }  }  // Destructor  Polinom::~Polinom() {      delete[]this->coef;  }  // Operator assignment  Polinom& Polinom::operator=(const Polinom& p) {      this->degree = p.degree;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = p.coef[i];      }      return \*this;  }  // Getter setter  int Polinom::getCoef(int idx) const{      return this->coef[idx];  }  int Polinom::getDegree() const{      return this->degree;  }  int Polinom::getNumOp() {      return numOp;  }  void Polinom::setCoef(int idx, int val) {      this->coef[idx] = val;  }  void Polinom::setDegree(int degree) {      this->degree = degree;  }  // Other operator  Polinom Polinom::operator+(const Polinom& A) {      if (A.getDegree() == this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= res.getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          return res;      }      else if (A.getDegree() > this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= this->getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          for (int i = this->getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = A.getCoef(i);              numOp++;          }          return res;      }      else {          Polinom res(this->getDegree());          for (int i = 0; i <= A.getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          for (int i = A.getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = this->getCoef(i);              numOp++;          }          return res;      }  }  Polinom Polinom::operator-(const Polinom& A) {      if (A.getDegree() == this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= res.getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);          }          return res;      }      else if (A.getDegree() > this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= this->getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);          }          for (int i = this->getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = 0 - A.getCoef(i);          }          return res;      }      else {          Polinom res(this->degree);          for (int i = 0; i <= A.getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);          }          for (int i = A.getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = this->getCoef(i);          }          return res;      }  }  Polinom Polinom::operator\*(const Polinom& A) {      Polinom res(this->getDegree()+A.getDegree());      int idx;      cout << this->getDegree() << endl;      res.print();      for (int i = 0; i <= this->getDegree(); i++) {          // cout << "AAA" << endl;          idx = i;          for (int j = 0; j <= A.getDegree(); j++) {              // cout << "BBB" << endl;              res.coef[idx] += (this->getCoef(i) \* A.getCoef(j));              numOp++;              numOp++;              idx++;          }      }      return res;  }  // Other method  void Polinom::print() {      cout << this->coef[0];      for (int i = 1; i <= this->degree; i++) {          if (this->coef[i] > 0) {              cout << " + ";              if (this->coef[i] != 1) {                  cout << this->coef[i];              }              cout << "x";              if (i > 1) {                  cout << "^" << i;              }          }          else if (this->coef[i] < 0) {              int temp = this->coef[i];              temp \*= -1;              cout << " - ";              if (temp != 1) {                  cout << temp;              }              cout << "x";              if (i > 1) {                  cout << "^" << i;              }          }      }      cout << endl;  }  void Polinom::inputCoef() {      for (int i = 0; i <= this->degree; i++) {          cin >> this->coef[i];      }  }  void Polinom::inputRandom() {      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = rand() % 11 - rand() % 11;          while (this->coef[i] == 0) {              this->coef[i] = rand() % 11 - rand() % 11;          }      }  } |

Berikut adalah kode program untuk pustaka algoritma *Divide and Conquer* :

Kode program dari file header “DividenConquer.hpp”

|  |
| --- |
| #ifndef \_\_DIVIDE\_\_AND\_\_CONQUER\_\_HPP\_\_  #define \_\_DIVIDE\_\_AND\_\_CONQUER\_\_HPP\_\_  #include <iostream>  #include <time.h>  using namespace std;  class Polinom {      private :          int \*coef;          int degree;          static int numOp;      public  :          // Constructor          Polinom();          Polinom(int degree);          // Copy constructor          Polinom(const Polinom& p);          // Destructor          ~Polinom();          // Operator assignment          Polinom& operator=(const Polinom& p);          // Getter setter          int getCoef(int idx) const;          int getDegree() const;          static int getNumOp();          void setCoef(int idx, int val);          void setDegree(int degree);          // Other operator          Polinom operator+(const Polinom& A);          Polinom operator-(const Polinom& A);          Polinom operator\*(const Polinom& A);          // Other method          void print();          void inputCoef();          void inputRandom();          Polinom multiplyX(int d);  };  #endif |

Kode program dari file “DividenConquer.cpp”

|  |
| --- |
| #include "DividenConquer.hpp"  using namespace std;  // Static Init  int Polinom::numOp = 0;  // Constructor  Polinom::Polinom() {      this->degree = 0;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = 0;      }  }  Polinom::Polinom(int degree) {      if (degree < 0) {          this->degree = 0;      }      else {          this->degree = degree;      }      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = 0;      }  }  // Copy constructor  Polinom::Polinom(const Polinom& p) {      this->degree = p.degree;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = p.coef[i];      }  }  // Destructor  Polinom::~Polinom() {      delete[]this->coef;  }  // Operator assignment  Polinom& Polinom::operator=(const Polinom& p) {      this->degree = p.degree;      this->coef = new int[this->degree+1];      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = p.coef[i];      }      return \*this;  }  // Getter setter  int Polinom::getCoef(int idx) const{      return this->coef[idx];  }  int Polinom::getDegree() const{      return this->degree;  }  int Polinom::getNumOp() {      return numOp;  }  void Polinom::setCoef(int idx, int val) {      this->coef[idx] = val;  }  void Polinom::setDegree(int degree) {      this->degree = degree;  }  // Other operator  Polinom Polinom::operator+(const Polinom& A) {      if (A.getDegree() == this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= res.getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          return res;      }      else if (A.getDegree() > this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= this->getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          for (int i = this->getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = A.getCoef(i);              numOp++;          }          return res;      }      else {          Polinom res(this->getDegree());          for (int i = 0; i <= A.getDegree(); i++) {              res.coef[i] = A.getCoef(i) + this->coef[i];              numOp++;          }          for (int i = A.getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = this->getCoef(i);              numOp++;          }          return res;      }  }  Polinom Polinom::operator-(const Polinom& A) {      if (A.getDegree() == this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= res.getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);              numOp++;          }          return res;      }      else if (A.getDegree() > this->degree) {          Polinom res(A.getDegree());          for (int i = 0; i <= this->getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);              numOp++;          }          for (int i = this->getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = 0 - A.getCoef(i);              numOp++;          }          return res;      }      else {          Polinom res(this->degree);          for (int i = 0; i <= A.getDegree(); i++) {              res.coef[i] = this->coef[i] - A.getCoef(i);          }          for (int i = A.getDegree() + 1; i <= res.getDegree(); i++) {              res.coef[i] = this->getCoef(i);          }          return res;      }  }  Polinom Polinom::operator\*(const Polinom& A) {      if (A.getDegree() == 0 || this->getDegree() == 0) {          Polinom C;          C.coef[0] = A.getCoef(0)\*this->getCoef(0);          numOp++;          return C;      }      else {          int lowerSize = 0;          if (A.getDegree() % 2 == 0) {             lowerSize = (A.getDegree()/2) - 1;          }          else {              lowerSize = A.getDegree()/2;          }          // int lowerSize = A.getDegree()/2 - 1;          int higherSize = (A.getDegree() - lowerSize - 1);          Polinom ALower(lowerSize);          Polinom AHigher(higherSize);          Polinom BLower(lowerSize);          Polinom BHigher(higherSize);          for (int i = 0; i <= lowerSize; i++) {              ALower.setCoef(i, A.getCoef(i));              BLower.setCoef(i, this->getCoef(i));          }          for (int i = 0; i <= higherSize; i++) {              AHigher.setCoef(i, A.getCoef(i + 1 + lowerSize));              BHigher.setCoef(i , this->getCoef(i + 1 + lowerSize));          }          Polinom Y = (AHigher + ALower) \* (BHigher + BLower);          numOp++;          Polinom U = ALower \* BLower;          numOp++;          Polinom Z = AHigher \* BHigher;          numOp++;          Polinom res = U + (Y - U - Z).multiplyX(lowerSize+1) + Z.multiplyX((lowerSize+1)\*2);          return res;      }  }  // Other method  void Polinom::print() {      cout << this->coef[0];      for (int i = 1; i <= this->degree; i++) {          if (this->coef[i] > 0) {              cout << " + ";              if (this->coef[i] != 1) {                  cout << this->coef[i];              }              cout << "x";              if (i > 1) {                  cout << "^" << i;              }          }          else if (this->coef[i] < 0) {              int temp = this->coef[i];              temp \*= -1;              cout << " - ";              if (temp != 1) {                  cout << temp;              }              cout << "x";              if (i > 1) {                  cout << "^" << i;              }          }      }      cout << endl;  }  void Polinom::inputCoef() {      for (int i = 0; i <= this->degree; i++) {          cin >> this->coef[i];      }  }  void Polinom::inputRandom() {      for (int i = 0; i <= this->degree; i++) {          this->coef[i] = rand() % 11 - rand() % 11;          while (this->coef[i] == 0) {              this->coef[i] = rand() % 11 - rand() % 11;          }      }  }  Polinom Polinom::multiplyX(int d) {      Polinom res(this->getDegree() + d);      for (int i = d; i <= res.getDegree(); i++) {          res.coef[i] = this->coef[i-d];      }      return res;  } |

# **BAB III UJI COBA**

1. **Spesifikasi Komputer**

Berikut adalah spesifikasi komputer yang digunakan untuk melakukan uji coba.

* Nama : ASUS X550IK
* OS : Windows 10 64 bit
* CPU : QuadCore AMD Radeon FX-9830P 3.8GHz
* GPU : AMD Radeon R7 & AMD Radeon RX560
* RAM : 8GB

1. **Uji Coba dengan Algoritma *Brute Force***

Berikut adalah hasil uji coba dengan menggunakan algoritma *brute force* dengan membangkitkan koefisien polinom secara acak :

* + - 1. Untuk n = 5
      2. Untuk n = 10
      3. Untuk n = 20
      4. Untuk n = 50

1. **Uji Coba dengan Algoritma *Divide and Conquer***

Berikut adalah hasil uji coba dengan menggunakan algoritma *divide and conquer* dengan membangkitkan koefisien polinom secara acak :

* + - 1. Untuk n = 5
      2. Untuk n = 10
      3. Untuk n = 20
      4. Untuk n = 50

Penilaian Asisten

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | V |  |
| 1. Program berhasil *running* | V |  |
| 1. Program dapat menerima input dan menuliskan output | V |  |
| 1. Luaran sudah benar untuk semua *n* | V |  |