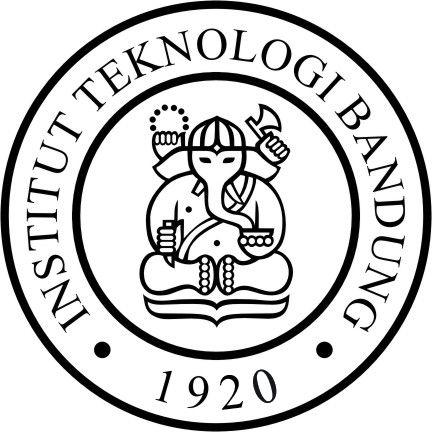
**LAPORAN TUGAS KECIL II IF-2211**

**STRATEGI ALGORITME**

**PUSTAKA PERKALIAN POLINOM DENGAN ALGORTIMA KARATSUBA - DIVIDE AND CONQUER**



|  |  |
| --- | --- |
| Disusun oleh : |  |
| Naufal Prima Yoriko | 13518146 |

**DAFTAR ISI**

|  |  |
| --- | --- |
| [**DAFTAR ISI**](#page2) | [**1**](#page2) |
| [**BAB I**](#page3) | [**3**](#page3) |
| Algoritme Divide and Conquer | [3](#page3) |
| Polinom | [3](#page3) |
| [Penjelasan](#page4) Langkah Algoritme | [4](#page4) |
| [Analisis](#page4) Kompleksitas Algoritme | [5](#page4) |
| [**BAB II**](#page6) | [**6**](#page6) |
| [Spesifikasi](#page6) Komputer | [6](#page6) |
| [Kode](#page6) Program | [6](#page6) |
| Uji Kasus | 19 |
| Ringkasan Hasil Uji Kasus | 21 |

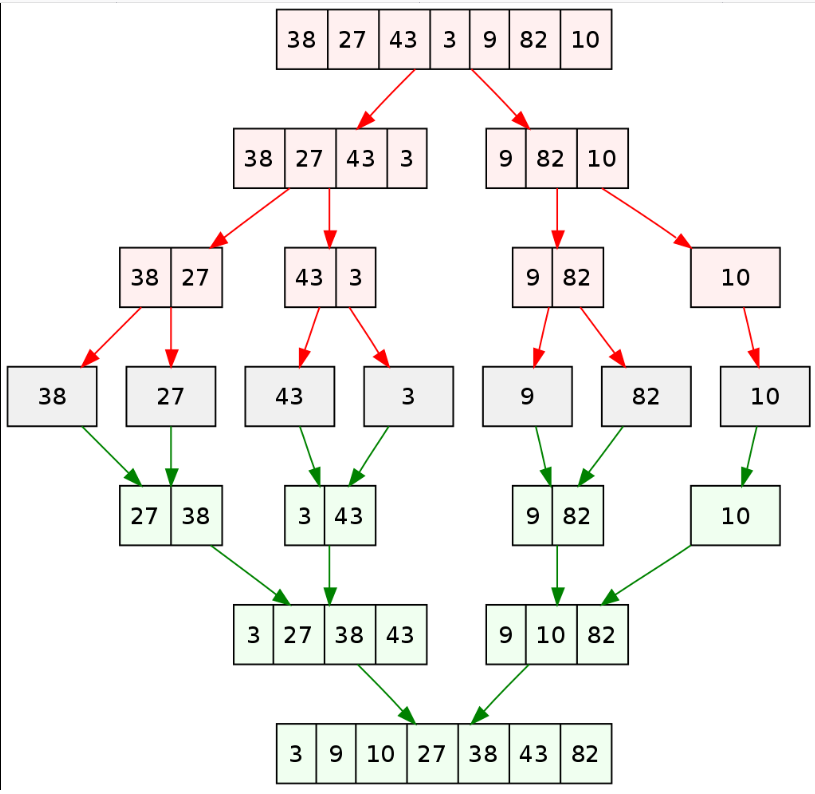
|  |  |
| --- | --- |
| [**BAB III**](#page17) | [**22**](#page17) |
| [Kesimpulan](#page17) | [22](#page17) |
| [Catatan](#page19) Program | [22](#page19) |
| [**BAB IV**](#page23) | 23 |

**BAB I**

**TEORI DASAR**

1. **Algoritme Divide And Conquer**

Divide and Conquer awalnya berasal dari salah satu strategi perang yang terkenal, yakni memecah belah pasukan dan kekuatan lawan, untuk kemudian ditaklukan. Kinerja algoritme ini pun demikian. Namun secara teknis, algoritme ini akan memecah suatu persoalan menjadi beberapa upa-persoalan dengan cara rekursi bercabang(yang secara implisit berarti solusi ini memiliki kondisi basis/ dasar, dan memiliki kondisi transisi tertentu), untuk kemudian dipecahkan tiap upa-masalah. Saat upa-masalah dipecahkan, mereka akan disatukan kembali untuk menjadi bagian dari solusi masalah yang lebih besar. Algoritme ini menadi basis dari banyak persoalan yang ada sehingga memiliki banyak implementasi, seperti pengurutan (merge sort, quick sort), multiplikasi angka besar, mencari pasangan titik terdekat, Fast Fourier Transform / FFT, dan masih banyak lagi.

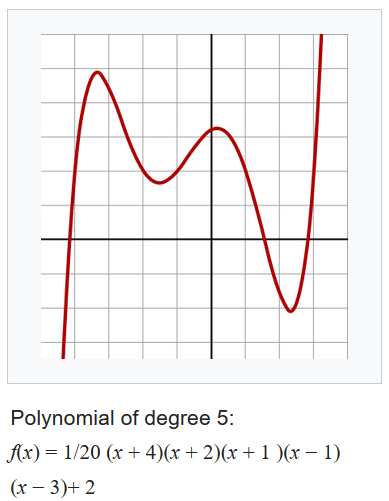


Gambar 1.1. *Contoh skema algoritme divide and conquer, yakni merge sort.*

Sumber : http://en.wikipedia.org/

**II. Polinom**

Pada matematika, polinomial berarti sebuah ekspresi yang terdiri dari variabel/indeterminan dan juga koefisien. Antar polinom dapat melakukan berbagai macam operasi matematis, termasuk operasi dasar seperti perkalian dan penjumlahan. Implementasi dari penggunaan polinom ini sangat luas tak terbatas pada ilmu matematika murni, walaupun polinom menjadi karakteristik dari ilmu aljabar di matematika. Pada kalkulus, polinom ini bahkan bisa menjadi aproksimasi berbagai fungsi kompleks yang ada di cabang keilmuan lain, seperti trigonometri, *power series,* dan lainnya.



Gambar 1.2. *Contoh grafik dan fungsi dari polinom*

Sumber : http://en.wikipedia.org/

1. **Penjelasan Langkah Algoritme**

Algortime dari perkalian polinom ini dapat bermacam-macam, namun yang utama didekati dengan setidaknya dua cara berikuti ini, diantaranya

1. Brute Force

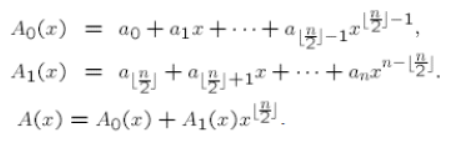
Pendekatan brute force cukup sederhana, selayaknya pengerjaan secara manual. Semisal ada dua polinom dimana P1 berderajat n dan P2 berderajat m, maka

* 1. Iterasi seluruh elemen polinom P1.
  2. Tiap iterasi, elemen P1 tersebut diiterasi kembali untuk dikalikan terhadap seluruh elemen polinom P2.
  3. Hasil perkalian disimpan dan dijumlahkan dengan hasil perkalian lainnya dengan derajat variabel (misal x) yang sama.

1. Divide and Conquer

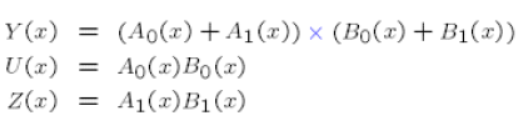
Dengan referensi dari slide kuliah, pendekatan divide and conquer ini menggunakan metode dari Karatsuba, yang cukup sederhana dan mudah, selain itu efisien dibanding brute force pada derajat polinom yang besar, walaupun sulit agak dilakukan secara manual. Semisal ada dua polinom A dan B dimana keduanya berderajat n (tidak mesti sama hanya simplifikasi), maka

1. Polinom akan dibagi(*dividing*) menjadi dua bagian yang sama besar, dimana satu bagian berorde 0 hingga floor(n/2), dan satu lagi dari floor(n/2) + 1 hingga n. Sehingga



Dan hal yang sama berlaku pada polinom B.

1. Selama elemen belum ada yang satu maka terus ulangi langkah dua hingga salah satu (atau keduanya) A dan/atau B menjadi satu elemen.
2. Jika salah satu polinom berelemen satu (misal A, dimana elemen Ai tersebut adalah k) maka hasil perkaliannya adalah k \* B (*conquer*).
3. Ulangi hingga seluruh bagian mencapai kondisi di langkah ketiga.
4. Kemudian dari elemen yang sudah dibagi-bagi(*divide*) tersebut, hitung tiga polinom baru sebagai berikut



Karena jika persamaan pada persamaan di atas berlaku, maka hasil dari perkalian A dan B adalah penggabungan(*merge/combine*) ketiga polinom tersebut, yang persamaannya



1. Terus ulangi langkah kelima hingga seluruh bagian tergabung, atau dengan kata lain menjadi polinom bersuku 2n.
2. **Analisis Kompleksitas Algoritme**

Algoritme perkalian polinom di atas dapat dijabarkan kompleksitas masing-masing metodenya sebagai berikut

1. Brute Force

Pendekatan brute force cukup sederhana, seperti pengerjaan manual yang biasa kita lakukan. Semisal ada dua polinom dimana P1 berderajat n dan P2 berderajat m, maka

1. Iterasi seluruh elemen polinom P1 (total n iterasi).
2. Tiap iterasi, elemen P1 tersebut diiterasi kembali untuk dikalikan terhadap seluruh elemen polinom P2 (dilakukan m iterasi).
3. Hasil perkalian disimpan dan dijumlahkan dengan hasil perkalian lainnya dengan derajat variabel (misal x) yang sama (dilakukan pada faktor komplesitas konstan O(k)).

Maka berdasar pehitungan di atas komplesitasnya adalah

T(n) = O(n) \* O(m) \* O(k) = k \* O(mn)

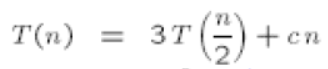
Jika m = n dan k = 1, maka

T(n) = O(n2)

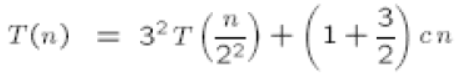
1. Divide and Conquer

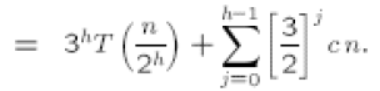
Dengan referensi dari slide kuliah, kompleksitas dari metode karatsuba dapat dijabarkan dengan teorema Master sebagai berikut, jika diasumsikan n = 2h, maka

1. Berdasar teroema Master, kompleksitas program divide and conquer dapat dimodelkan

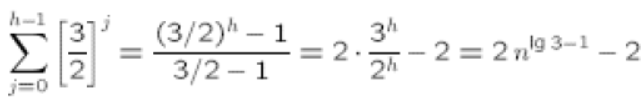


1. Dengan substitusi diperoleh









1. Maka diperoleh kompleksitas akhir algoritme adalah



**BAB II**

**IMPLEMENTASI**

1. **Spesifikasi Komputer**

Pada pengerjaan tugas kecil ini saya menggunakan sebuah laptop dengan spesifikasi sebagai berikut

|  |  |
| --- | --- |
| Merk dan Model Laptop | HP Pavilion Laptop 14-bf1xx |
| BIOS | F.34 |
| Processor | Inter Core i7-8550U CPU @1.80 GHz (8 CPUs) |
| Memory RAM | 16384 MB |
| DirectX Version | DirectX 12 |
| Display GPU | Intel UHD Graphics 620 (128 MB) |
| Render GPU | NVIDIA GeForce 940MX (4055 MB) |
| OS | Windows 10 Home |

Tabel 2.1. Tabel Spesifikasi Laptop

1. **Kode Program**

Berikut ini adalah program yang saya tulis dalam bahasa C++ dan akan saya pisah penjabarannya per file .cpp /.hpp karena fungsi yang cukup banyak sehingga tidak memungkinkan, selain itu penjabaran per file masih cukup mudah dipahami.

1. **Polinom.hpp**

|  |
| --- |
| #ifndef Polinom\_HPP  #define Polinom\_HPP  #include <bits/stdc++.h>  class Polinom  {  public:  // Konstruktor Destruktor      Polinom();      Polinom(int);      Polinom(double \*, int);      Polinom(Polinom const &);      ~Polinom();  // Operator Overload      Polinom operator=(Polinom const &);      Polinom operator+(Polinom const &);      Polinom operator-(Polinom const &);      virtual Polinom operator\*(Polinom const &);  // Getter      int getDegree();      int getSize();      double getElmt(int);      int getNumPlusMinus();      int getNumProduct();      double getOpDuration();  // Setter      void resizeElmt(int);      void setElmt(int, double);      void setElmt(int, double \*);  // Fungsi Misc      void printAll();      void printAll(const char \*);  protected:  // Data      int size;      double \*elmt;      int numplusminus, numproduct;      clock\_t begins, ends;  };  #endif |

Tabel 2.2.1 Kode Polinom.hpp

Pada tabel 2.2.1 berisi kode dari Polinom.hpp yang merupakan spesifikasi dari kelas PolinomNaive, sekaligus base class dari keseluruhan library ini. Disini berikan spesifikasi dasar dari suatu class sekaligus yang spesifik dibutuhkan program ini.

1. **Polinom.cpp**

|  |
| --- |
| #include <bits/stdc++.h>  #include "Polinom.hpp"  using namespace std;  Polinom::Polinom()  {      numplusminus = 0;      numproduct = 0;      size = 0;  }  Polinom::Polinom(int siz)  {      numplusminus = numproduct = 0;      size = siz;      elmt = new double[size];  }  Polinom::Polinom(double \*d, int siz)  {      numplusminus = 0;      numproduct = 0;      while (siz >= 1 && d[siz - 1] == 0)      {          siz--;      }      size = siz;      elmt = new double[size];      //cout << sizeof(elmt) << endl;      for (int i = 0; i < size; i++)      {          elmt[i] = d[i];      }  }  Polinom::Polinom(Polinom const &p)  {      size = p.size;      elmt = new double[size];      for (int i = 0; i < size; i++)      {          elmt[i] = p.elmt[i];      }      numplusminus = p.numplusminus;      numproduct = p.numproduct;      begins = p.begins;      ends = p.ends;  }  Polinom::~Polinom()  {      if (size != 0)          delete[] elmt;  }  Polinom Polinom::operator=(Polinom const &p)  {      if (size != 0)          delete[] elmt;      Polinom x(p);      return x;  }  Polinom Polinom::operator+(Polinom const &p)  {      Polinom x;      x.size = max(size, p.size);      x.elmt = new double[x.size];      x.begins = clock();      for (int i = 0; i < x.size; i++)      {          if (i > size)              x.elmt[i] = p.elmt[i];          else if (i > p.size)              x.elmt[i] = elmt[i];          else              x.elmt[i] = elmt[i] + p.elmt[i];          x.numplusminus++;      }      x.ends = clock();      int k = x.size;      while (k >= 1 && x.elmt[k - 1] == 0)      {          k--;      }      Polinom y(x.elmt, k);      y.begins = x.begins;      y.ends = x.ends;      y.numplusminus = x.numplusminus;      return y;  }  Polinom Polinom::operator-(Polinom const &p)  {      Polinom x;      x.size = max(size, p.size);      x.elmt = new double[x.size];      x.begins = clock();      for (int i = 0; i < x.size; i++)      {          if (i > size)              x.elmt[i] = -1 \* p.elmt[i];          else if (i > p.size)              x.elmt[i] = elmt[i];          else              x.elmt[i] = elmt[i] - p.elmt[i];          x.numplusminus++;      }      x.ends = clock();      int k = x.size;      while (k >= 1 && x.elmt[k - 1] == 0)      {          k--;      }      Polinom y(x.elmt, k);      y.begins = x.begins;      y.ends = x.ends;      y.numplusminus = x.numplusminus;      return y;  }  Polinom Polinom::operator\*(Polinom const &p)  {      return \*this;  }  int Polinom::getDegree()  {      return size - 1;  }  int Polinom::getSize()  {      return size;  }  double Polinom::getElmt(int i)  {      return elmt[i];  }  int Polinom::getNumPlusMinus()  {      return numplusminus;  }  int Polinom::getNumProduct()  {      return numproduct;  }  double Polinom::getOpDuration()  {      return (double)(ends - begins) / (double)CLOCKS\_PER\_SEC \* (double)1e6;  }  void Polinom::resizeElmt(int siz)  {      // if (size != 0)      //     delete[] elmt;      elmt = new double[siz];      size = siz;  }  void Polinom::setElmt(int i, double x)  {      elmt[i] = x;  }  void Polinom::setElmt(int siz, double \*arr)  {      size = siz;      if (size != 0)          delete[] elmt;      elmt = new double[siz];      for (int i = 0; i < siz; i++)          elmt[i] = arr[i];  }  void Polinom::printAll()  {      for (int i = size - 1; i > -1; i--)      {          if (elmt[i] == 0)              continue;          else if (elmt[i] < 0)          {              if (i == size - 1)                  cout << "-";              else                  cout << " - ";          }          else          {              if (i != size - 1)                  cout << " + ";          }          if (abs(elmt[i]) != 1 || (i == 0 && elmt[i] != 0))              cout << abs(elmt[i]);          if (i != 0)              cout << "x^" << i;      }      cout << endl;  }  void Polinom::printAll(const char \*filename)  {      freopen(filename, "w", stdout);      printAll();      fclose(stdout);  } |

Tabel 2.2.2 Kode Polinom.cpp

Pada tabel 2.2.2 berisi kode dari Polinom.cpp yang berisi realisasi dari spesifikasi kelas Polinom di Polinom.hpp. Hampir seluruh pengaturan memori dari kelas (baik base maupun inherit) juga dilakukan melalui realisasi dari kelas ini.

1. **PolinomDnC.hpp**

|  |
| --- |
| #ifndef POLINOMNAIVE\_HPP  #define POLINOMNAIVE\_HPP  #include "Polinom.hpp"  class PolinomNaive : public Polinom  {  public:      PolinomNaive();      PolinomNaive(int);      PolinomNaive(double \*, int);      PolinomNaive(PolinomNaive const &);      PolinomNaive operator\*(PolinomNaive const &);  };  #endif |

Tabel 2.2.3 Kode PolinomNaive.hpp

Pada tabel 2.2.3 berisi kode dari PolinomNaive.hpp yang merupakan spesifikasi dari kelas PolinomNaive, sekaligus inherit/kelas turunan dari kelas Polinom.

1. **PolinomNaive.hpp**

|  |
| --- |
| #include "PolinomNaive.hpp"  using namespace std;  PolinomNaive::PolinomNaive() : Polinom() {}  PolinomNaive::PolinomNaive(int n) : Polinom(n) {}  PolinomNaive::PolinomNaive(double \*q, int r) : Polinom(q, r) {}  PolinomNaive::PolinomNaive(PolinomNaive const &q) : Polinom(q) {}  PolinomNaive PolinomNaive::operator\*(PolinomNaive const &p)  {      PolinomNaive x;      double temp[size + p.size - 1];      for (int i = 0; i < sizeof(temp) / sizeof(double); i++)          temp[i] = 0;      x.begins = clock();      for (int i = 0; i < size; i++)      {          for (int j = 0; j < p.size; j++)          {              if (elmt[i] != 0 && p.elmt[j] != 0)              {                  x.numplusminus++;                  x.numproduct++;                  temp[i + j] += elmt[i] \* p.elmt[j];              }          }      }      x.ends = clock();      int k = size + p.size - 1;      while (k >= 1 && temp[k - 1] == 0)      {          k--;      }      x.size = k;      x.elmt = new double[k];      for (int i = 0; i < k; i++)      {          x.elmt[i] = temp[i];      }      return x;  } |

Tabel 2.2.4 Kode PolinomNaive.cpp

Pada tabel 2.2.4 adalah kode dari PolinomNaive.cpp yang berisi realisasi dari spesifikasi kelas PolinomNaive di PolinomNaive.hpp, sekaligus overloading dari operator kali Polinom.cpp, dimana PolinomNaive mengimplementasikan algoritme brute force pada kalkulasinya.

Sebagai catatan, disini penghitungan operasi kali saat perkalian dua suku dari polinom pertama dan kedua, sedangkan perhitungan penjumlahan hanya saat menginput hasil perkalian ke suatu indeks array dari polinom hasil(sehingga nilai operasi kali dan jumlah hampir selalu sama), dan kedua perhitungan ini hanya dilakukan saat operan nilainya tak nol.

Selain itu, perhitungan waktu dilakukan saat tepat sebelum dan setelah iterasi seluruh elemen polinom pertama terhadap seluruh elemen polinom kedua, sehingga durasi operasi adalah selisih waktu keduanya.

1. **PolinomDnC.hpp**

|  |
| --- |
| #ifndef POLINOMDNC\_HPP  #define POLINOMDNC\_HPP  #include <bits/stdc++.h>  #include "Polinom.hpp"  using namespace std;  class PolinomDnC : public Polinom  {  public:      PolinomDnC();      PolinomDnC(int);      PolinomDnC(double \*, int);      PolinomDnC(PolinomDnC const &);      PolinomDnC operator\*(PolinomDnC const &);  protected:      vector<double> countProduct(vector<double> \*, vector<double> \*, int, int, int \*, int \*);  };  #endif |

Tabel 2.2.5 Kode PolinomDnC.hpp

Pada tabel 2.2.5 berisi kode dari PolinomDnC.hpp yang merupakan spesifikasi dari kelas PolinomDnC, sekaligus inherit/kelas turunan dari kelas Polinom.

1. **PolinomDnC.cpp**

|  |
| --- |
| #include "PolinomDnC.hpp"  #include <bits/stdc++.h>  using namespace std;  PolinomDnC::PolinomDnC() : Polinom() {}  PolinomDnC::PolinomDnC(int n) : Polinom(n) {}  PolinomDnC::PolinomDnC(double \*q, int r) : Polinom(q, r) {}  PolinomDnC::PolinomDnC(PolinomDnC const &q) : Polinom(q) {}  PolinomDnC PolinomDnC::operator\*(PolinomDnC const &p)  {      PolinomDnC x;      int N = max(pow(2, ceil(log2(size))), pow(2, ceil(log2(p.size))));      vector<double> temp1, temp2, ans;      temp1.resize(N);      temp2.resize(N);      ans.resize(2 \* N);      for (int i = 0; i < size; i++)          temp1[i] = elmt[i];      for (int i = 0; i < p.size; i++)          temp2[i] = p.elmt[i];      x.begins = clock();      ans = countProduct(&temp1, &temp2, 0, N, &(x.numplusminus), &(x.numproduct));      x.ends = clock();      x.size = size + p.size - 1;      x.elmt = new double[x.size];      for (int i = 0; i < x.size; i++)          x.elmt[i] = ans[i];      return x;  }  vector<double> PolinomDnC::countProduct(vector<double> \*a, vector<double> \*b, int cnt, int N, int \*addPlus, int \*addProd)  {      int i, j;      // Base Case (Conquer Part)      if (N == 1)      {          (\*addProd)++;          vector<double> res = {a->at(cnt) \* b->at(cnt)};          return res;      }      // Divide Part      int mid = N / 2;      vector<double> c1, c2;      c1.resize(mid);      c2.resize(mid);      for (i = 0; i < mid; i++)      {          c1[i] = a->at(i + cnt + mid) + a->at(i + cnt);          c2[i] = b->at(i + cnt + mid) + b->at(i + cnt);      }      vector<double> p1 = countProduct(a, b, cnt, mid, addPlus, addProd);      vector<double> p2 = countProduct(&c1, &c2, 0, mid, addPlus, addProd);      vector<double> p3 = countProduct(a, b, cnt + mid, mid, addPlus, addProd);      vector<double> res;      res.resize(2 \* N);      for (i = 0; i < 2 \* N; i++)          res[i] = 0;      for (i = 0; i < N; i++)      {          if (abs(p1[i]) > 1e-8)          {              res[i] += p1[i];              (\*addPlus)++;          }          if (abs(p2[i] - p1[i] - p3[i]) > 1e-8)          {              res[i + mid] += (p2[i] - p1[i] - p3[i]);              (\*addPlus)++;          }          if (abs(p3[i]) > 1e-8)          {              res[i + N] += p3[i];              (\*addPlus)++;          }      }      return res;  } |

Tabel 2.2.6 Kode PolinomDnC.cpp

Pada tabel 2.2.6 adalah kode dari PolinomDnC.cpp yang berisi realisasi dari spesifikasi kelas PolinomDnC pada PolinomDnC.hpp, sekaligus overloading dari operator kali Polinom.cpp, dimana PolinomDnC mengimplementasikan algoritme Karatsuba pada kalkulasinya (dengan fungsi bantuan countProduct).

Sebagai catatan, disini penghitungan operasi kali saat di base case(*conquer*), yakni mengalikan satu bilangan dengan bilangan lain, sedangkan operasi tambah terjadi saat proses penyatuan(*merging*). Perhitungan kali dan tambah hanya dilakukan saat operan nilainya tak nol.

Selain itu, perhitungan waktu dilakukan saat tepat sebelum eksekusi dan tepat setelah eksekusi fungsi bantuan countProduct yang pertama kali (rekursi pertama), sehingga durasi operasi adalah selisih waktu keduanya.

1. **main.cpp**

|  |
| --- |
| #include <bits/stdc++.h>  #include "PolinomDnC.hpp"  #include "PolinomNaive.hpp"  using namespace std;  void mainMenu();  void menuGenerator()  {      printf("Pilih orde/suku/derajat polinom yang akan di generate: ");      int n, range = 100;      cin >> n;      n++;      if (!(n > 0))      {          printf("Input salah!!\n\n");          mainMenu();      }      Polinom a(n), b(n);      for (int i = 0; i < n; i++)      {          srand((time(0) \* i) + 7);          int num = rand() % (2 \* range + 1);          num -= range;          a.setElmt(i, num);      }      for (int i = 0; i < n; i++)      {          srand((time(0) \* i) + 13);          int num = rand() % (2 \* range + 1);          num -= range;          b.setElmt(i, num);      }      printf("\nPolinom hasil generate 1:\n");      a.printAll();      printf("\nPolinom hasil generate 2:\n");      b.printAll();      printf("\nData Polinom berhasil disimpan di Pol.txt!\n");      freopen("Pol.txt", "w", stdout);      cout << n << endl;      for (int i = 0; i < n; i++)          cout << a.getElmt(i) << ' ';      cout << endl;      for (int i = 0; i < n; i++)          cout << b.getElmt(i) << ' ';      cout << endl;      fclose(stdout);  }  void menuOperasi()  {      printf("\nPilih Operasi:\n 1. Perkalian\n 2. Penjumlahan\n 3. Pengurangan\n\nMasukan pilihan: ");      int opsi, opsi2, n;      cin >> opsi;      printf("\nPilih metode input:\n 1. Dari textfile(Pol.txt)\n 2. Manual\n\nMasukkan opsi: ");      cin >> opsi2;      if (opsi2 == 1)          freopen("Pol.txt", "r", stdin);      else if (opsi2 == 2)          printf("\nMasukkan orde/derajat: ");      cin >> n;      n++;      if (!(n > 0))      {          printf("Input salah!!\n\n");          mainMenu();      }      if (opsi == 1)      {          double temp;          PolinomDnC D1(n), D2(n);          PolinomNaive N1(n), N2(n);          for (int i = 0; i < n; i++)          {              cin >> temp;              D1.setElmt(i, temp);              N1.setElmt(i, temp);          }          for (int i = 0; i < n; i++)          {              cin >> temp;              D2.setElmt(i, temp);              N2.setElmt(i, temp);          }          if (opsi2 == 1)              fclose(stdin);          PolinomDnC Dres = D1 \* D2;          PolinomNaive Nres = N1 \* N2;          printf("\nPolinom hasil DnC vers. Karatsuba:\n");          Dres.printAll();          printf("Jumlah perkalian: %d   |   Jumlah penjumlahan/pengurangan: %d\n", Dres.getNumProduct(), Dres.getNumPlusMinus());          printf("Waktu eksekusi : %.5f microseconds\n\n", Dres.getOpDuration());          printf("Polinom hasil Brute Force:\n");          Dres.printAll();          printf("Jumlah perkalian: %d   |   Jumlah penjumlahan/pengurangan: %d\n", Nres.getNumProduct(), Nres.getNumPlusMinus());          printf("Waktu eksekusi : %.5f microseconds\n\n", Nres.getOpDuration());          Dres.printAll("resDnC.txt");          Nres.printAll("resNaive.txt");      }      else if (opsi == 2 || opsi == 3)      {          double temp;          Polinom D1(n), D2(n);          for (int i = 0; i < n; i++)          {              cin >> temp;              D1.setElmt(i, temp);          }          for (int i = 0; i < n; i++)          {              cin >> temp;              D2.setElmt(i, temp);          }          if (opsi2 == 1)              fclose(stdin);          Polinom Dres = (opsi == 2) ? (D1 + D2) : (D1 - D2);          printf("\nPolinom hasil %s:\n", (opsi == 2) ? "Penjumlahan" : "Pengurangan");          Dres.printAll();          printf("Jumlah perkalian: %d   |   Jumlah penjumlahan/pengurangan: %d\n", Dres.getNumProduct(), Dres.getNumPlusMinus());          printf("Waktu eksekusi : %.5f microseconds\n\n", Dres.getOpDuration());          Dres.printAll("resPlusMinus.txt");      }      else      {          printf("Input salah!!\n\n");          mainMenu();      }  }  void mainMenu()  {      printf("Polinom Apps:\n\n 1. Generator Polinom\n 2. Operasi Polinom\n 3. Exit\n\nMasukan pilihan: ");      int opsi;      cin >> opsi;      if (opsi == 1)          menuGenerator();      else if (opsi == 2)          menuOperasi();      else if (opsi == 3)          printf("Selamat tinggal!!\n");      else      {          printf("Input salah!!\n\n");          mainMenu();      }  }  int main()  {      mainMenu();      return 0;  } |

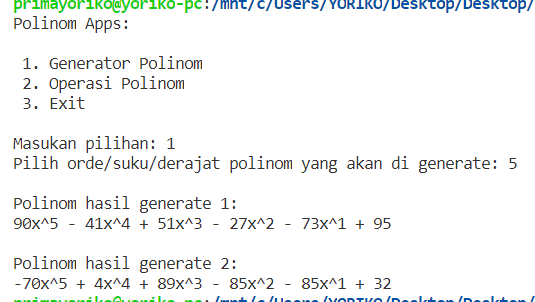
Tabel 2.2.7 Kode main.cpp

Pada tabel 2.2.7 berisi fungsi-fungsi penyusun tampilan yang digunakan dalam uji coba. Terdiri dari tiga fungsi utama:

1. mainMenu, menampilkan menu utama pemlihan fungsionalitas utama, yakni generate polinom atau operasi polinom.
2. menuGenerator, menampilkan menu dari fungsionalitas generator polinom, meminta input order, dan mencetak hasil generator ke layar dan data angkanya ke file ‘Pol.txt’.
3. menuOperasi, me menu dari fungsionalitas operasi polinom, memiliki tiga operasi, yakni perkalian (Karatsuba dan Brute Force), penjumlahan, dan pengurangan, hasilnya (nilai polinom, jumlah operasi kali dan tambah/kurang, serta waktu operasi) dicetak ke layar, sekaligus disimpan ke file .txt.
4. **Uji Kasus**

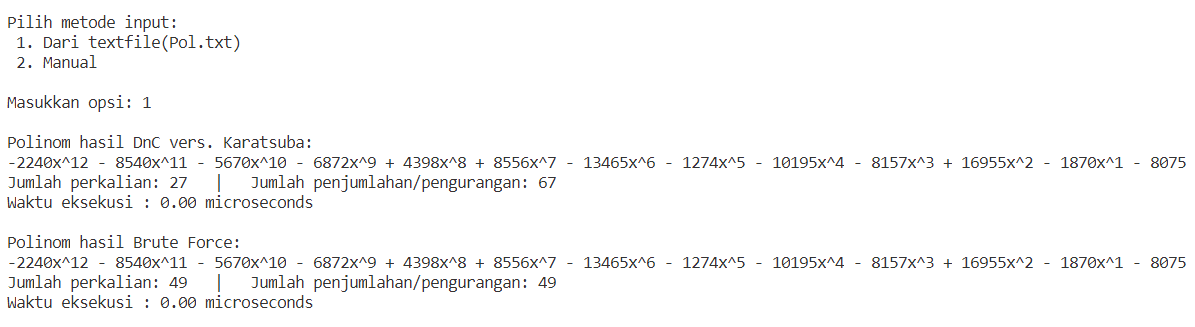
Berikut akan saya berikan contoh input dan output program ini pada beberapa kasus uji

1. Uji kasus **N = 5**



Gambar 2.3.1.1 Input dan Hasil Generate Polinom N = 5

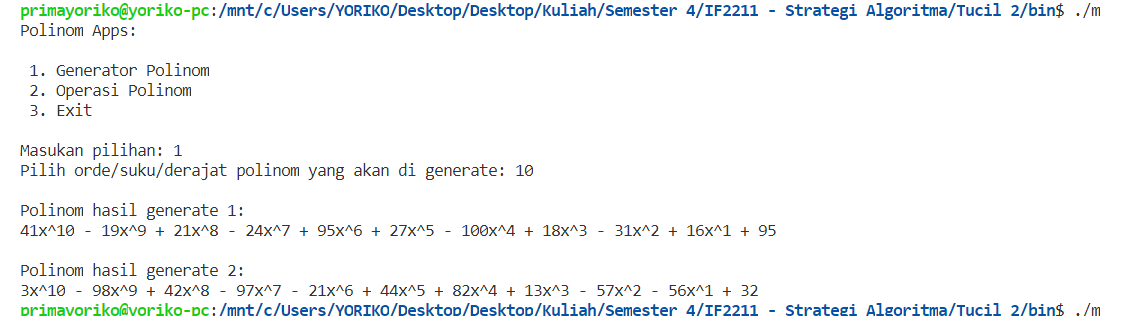
*Sumber : Dokumen Pribadi*



Gambar 2.3.1.2 Polinom Hasil Perkalian Polinom N = 5

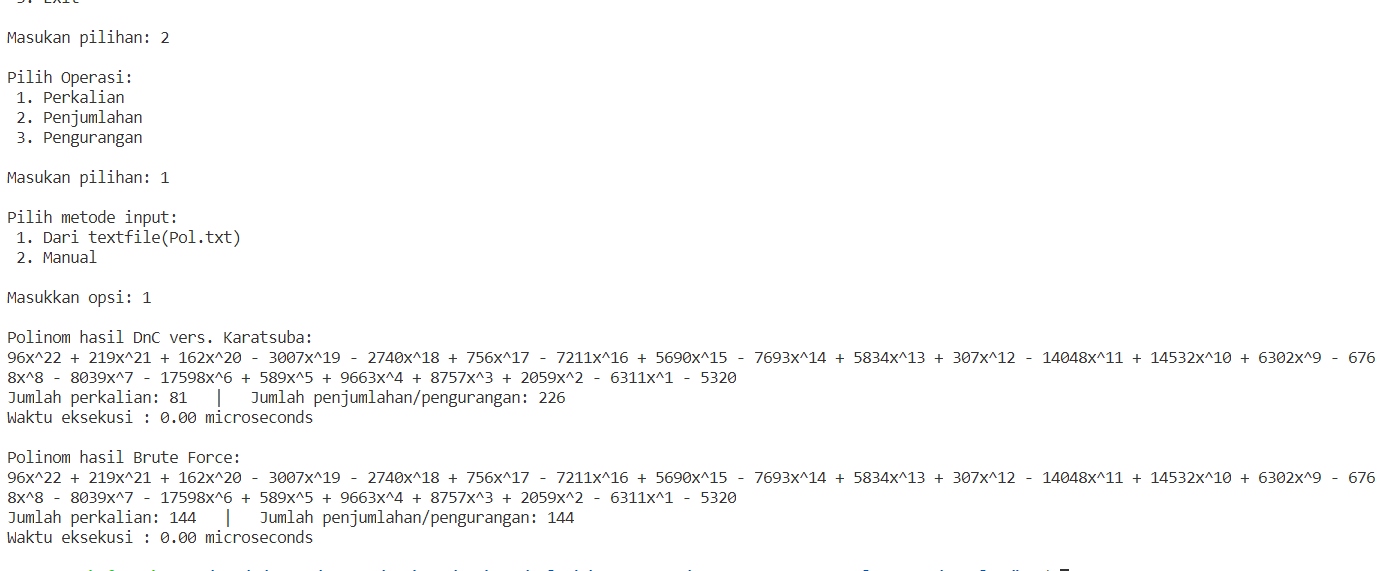
*Sumber : Dokumen Pribadi*

1. Uji kasus **N = 10**



Gambar 2.3.2.1 Input dan Hasil Generate Polinom N = 10

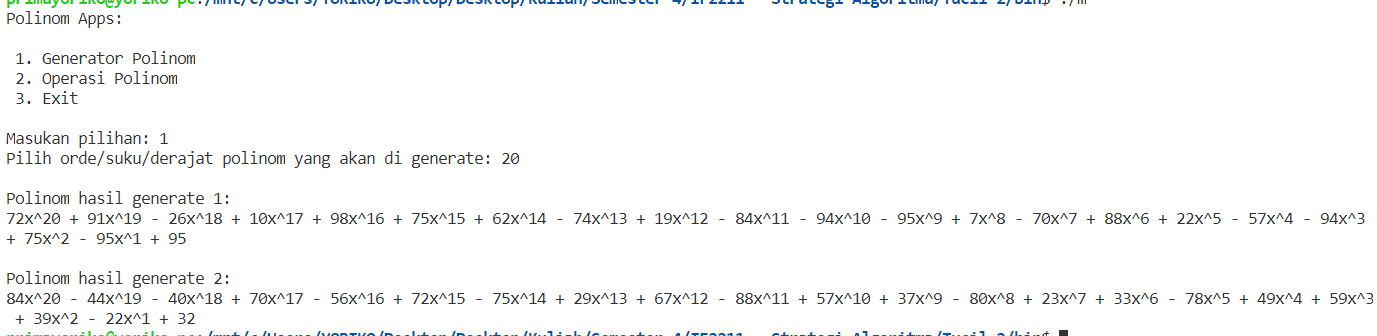
*Sumber : Dokumen Pribadi*



Gambar 2.3.2.2 Polinom Hasil Perkalian Polinom N = 10

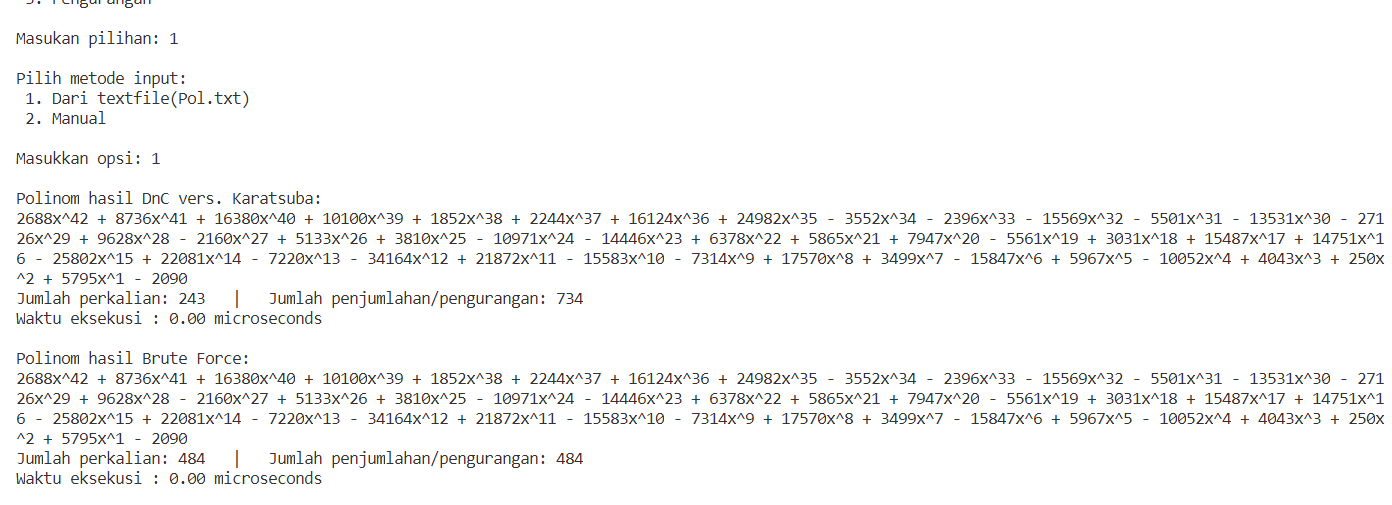
*Sumber : Dokumen Pribadi*

1. Uji kasus **N = 20**



Gambar 2.3.3.1 Input dan Hasil Generate Polinom N = 20

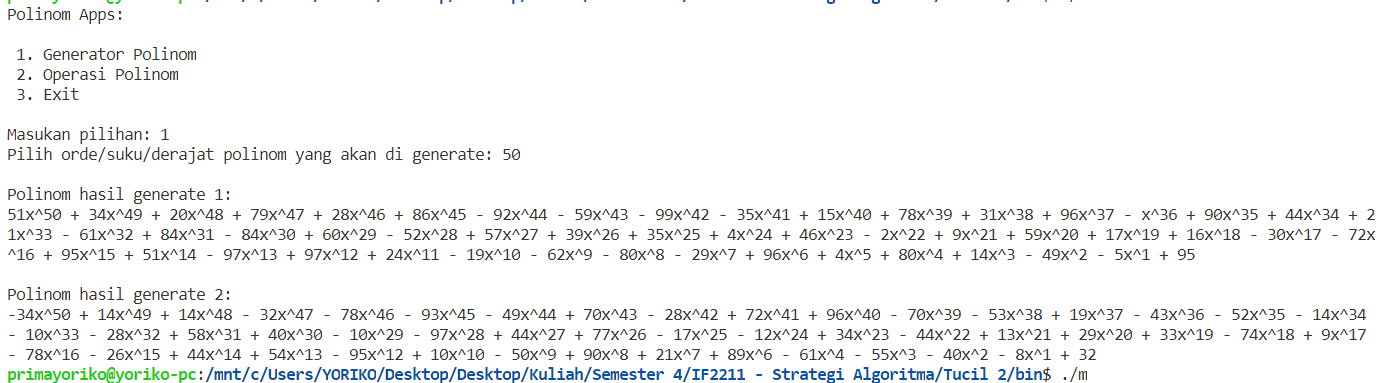
*Sumber : Dokumen Pribadi*



Gambar 2.3.3.2 Polinom Hasil Perkalian Polinom N = 20

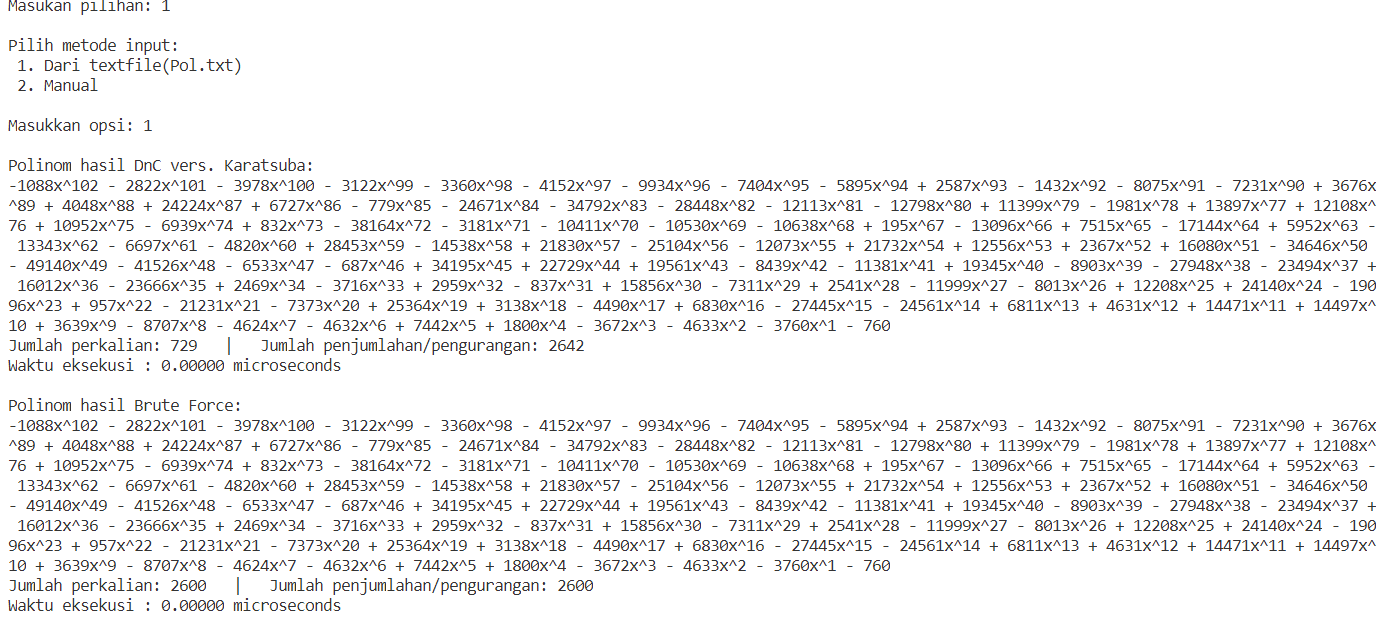
*Sumber : Dokumen Pribadi*

1. Uji kasus **N = 50**



Gambar 2.3.4.1 Input dan Hasil Generate Polinom N = 50

*Sumber : Dokumen Pribadi*



Gambar 2.3.4.2 Polinom Hasil Perkalian Polinom N = 50

*Sumber : Dokumen Pribadi*

**IV. Ringkasan Hasil Uji Kasus**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **Brute Force** | | | **DnC Karatsuba** | | | **Kebenaran** |
| **Tambah/ Kurang** | **Kali** | **Waktu Operasi (ms)** | **Tambah/ Kurang** | **Kali** | **Waktu Operasi (ms)** |
| 5 | 49 | 49 | 0 | 67 | 27 | 0 | YA |
| 10 | 144 | 144 | 0 | 226 | 81 | 0 | YA |
| 20 | 484 | 484 | 0 | 734 | 243 | 0 | YA |
| 50 | 2600 | 2600 | 0 | 2642 | 729 | 0 | YA |

Tabel 2.4. Tabel Ringkasan Hasil Uji Kasus

**BAB III**

**PENUTUP**

1. **Kesimpulan**

Berdasarkan hasil implementasi di bab dua, dapat disimpulkan hasil dari program sebagai berikut

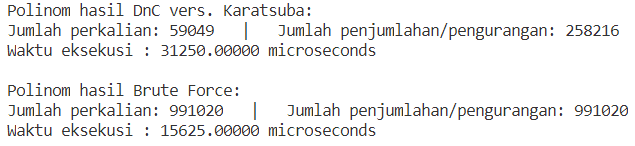
|  |  |  |  |
| --- | --- | --- | --- |
| No | Poin | Ya | Tidak |
| 1 | Program berhasil dikompilasi | v |  |
| 2 | Program berhasil *running* | v |  |
| 3 | Program dapat menerima input dan menuliskan output | v |  |
| 4 | Hasil sudah benar untuk semua n | v |  |

1. **Catatan Program**

Program memang berjalan sangat baik dan sesuai dengan yang diinginkan, namun tetap memerlukan untuk memerhatikan beberapa hal diantaranya

1. Waktu operasi tertulis hanya 0 ms untuk semua kasus uji, hal ini dikarenakan pada nilai n yang demikian terlalu kecil sehingga perbedaan waktu awal dan akhir tidak dapat terbaca, dan menyebabkan hasil dari waktu operasi adalah 0 ms.

Pada nilai n yang cukup besar tentu, hasil dari waktu uji tidak hanya 0 ms, seperti gambar berikut



Gambar 4.2 Contoh waktu operasi tak nol pada N = 1000

*Sumber : Dokumen Pribadi*

1. Sampai rentang tertentu, waktu operasi pada metode Karatsuba secara signifikan lebih besar, seperti di gambar 4.2 di atas. Hal ini disebabkan ada overhead proses dari algoritma seperti pembuatan array/vektor, yang membutuhkan proses *deep copy* dan memakan waktu lama, sehingga untuk persiapan akan memakan waktu yang jauh lebih besar dibandingkan proses perhitungan itu sendiri. Maka, pada n yang sangat besar sehingga waktu operasi yang dimiliki lebih signifikan.

**BAB IV**

**REFERENSI**

Munir, Rinaldi. *Algoritma Divide and Conquer, Bahan Kuliah IF2211 Strategi Algoritma*. Diakses pada 20-24 Februari 2020, dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/

Anonim. *Tugas Kecil 2 IF2211 Strategi Algoritma*. Diakses pada 17-24 Februari 2020, dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/

Singh, Harsh. *Karatsuba algorithm for fast multiplication using Divide and Conquer algorithm*. Diakses pada 22-24 Februari 2020. https://www.geeksforgeeks.org/karatsuba-algorithm-for-fast-multiplication-using-divide-and-conquer-algorithm/.

Harsh. *Multiply two polynomials*. Diakses pada 18-21 Februari 2020. https://www.geeksforgeeks.org/multiply-two-polynomials-2/.