**TUGAS KECIL II**

**IF2211 STRATEGI ALGORITMA**

**MEMBUAT PROGRAM PERKALIAN POLINOM DENGAN ALGORITMA *DIVIDE AND CONQUER***

**LAPORAN TUGAS KECIL**

Diajukan untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma

Oleh

**MUHAMMAD FAUZAN AL-GHIFARI**

**(13518112 - K01)**

****

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2020**

1. **Algoritma *Brute Force* dan Algoritma *Divide and Conquer***

Pada tugas kecil 1 ini saya menyelesaikan permasalahan perkalian polinom menggunakan algoritma *brufe force* dan *divide and conquer* dalam bahasa C++. Berikut adalah algoritma *brute force* dan *divide and conquer* berserta kompleksitasnya.

**Algoritma Brute Force**

|  |
| --- |
| #include "BF.hpp"  #include <iostream>  #include <vector>  using namespace std;  vector<int> BF(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah){  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Brute Force*      int i,j;      \*cKali = 0;      \*cTambah = 0;      vector<int> pol3(pol1.size()+pol2.size());      for (i = 0; i < pol1.size(); i++){          for (j = 0; j< pol2.size(); j++){              pol3[i+j] += pol1[i] \* pol2[j];              \*cKali += 1;*// Menghitung jumlah operasi kali*              \*cTambah += 1;*// Menghitung jumlah operasi tambah*          }      }      return pol3;  } |

Algoritma Brute Force merupakan jenis algoritma yang bersifat straight forward dan memiliki kompleksitas yang tinggi. Pemecahan masalah dengan brute forcedilakukan berdasarkan pada pernyataan soaldan defenisi konsep. Algoritma di atas menggunakan dua buah *for loop* sehingga algoritma di atas memiliki kompleksitas O(n2)

**Algoritma Divide and Conquer**

|  |
| --- |
| #include "DNC.hpp"  #include <iostream>  #include <vector>  using namespace std;  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Divide and Conquer*  vector<int> DNC(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah){  *// Basis jika polinom berderajat 0*      if(pol1.size() == 1 or pol2.size() == 1){          vector<int> pol3(1);          pol3[0] = pol1[0] \* pol2[0];          \*cKali += 1;*// Menghitung jumlah operasi kali*          return pol3;      }  *// Rekurens*      else{          int i;          int n = pol1.size()/2;          vector<int> A0, B0, A1, B1;          Divide(pol1, pol2, &A0, &B0, &A1, &B1, n);*// Membagi polinom menjadi dua bagian*            vector<int> Y = DNC(Sum(A0, A1), Sum(B0, B1), &\*cKali, &\*cTambah);          vector<int> U = DNC(A0, B0, &\*cKali, &\*cTambah);          vector<int> Z = DNC(A1, B1, &\*cKali, &\*cTambah);          \*cTambah += 6\*A1.size();*// Mengitung jumlah operasi tambah*          return Sum(Sum(Multi(Min(Min(Y, U), Z), n), U), Multi(Z, n\*2));      }  } |

Dalam menyelesaikan persoalan perkalian polinom, kita dapat melakukan optimisasi menggunakan algoritma *Divide and Conquer* seperti yang telah penulis lakukan di atas. Ide dasarnya adalah dengan berusaha mengurangi jumlah operasi perkalian yang dilakukan oleh program. Konsekuensi dari mengurangi jumlah operasi perkalian adalah jumlah operasi penjumlahan yang akan bertambah. Hal ini memang merupakan tujuan yang diharapkan karena operasi perkalian jauh lebih berat membebani performa program digandingkan dengan operasi penjumlahan.

Berikut ini adalah langkah-langkah yang dilakukan dalam menyelesaikan operasi perkalian polinom dengan algoritma *Divide and Conquer.*

1. Algoritma ini menggunakan rekursif dimana terdapat basis dan rekurens
2. Basis pada algoritma ini adalah polinom berderajat 0
3. Rekurens terjadi saat polinom berderajat lebih tinggi dari 0
4. Pada rekurens terjadi pemisahan polinom menjadi dua bagian dengan panjang polinom yang sama
5. Jika panjang polinom ganjil, maka polinom yang kedua akan memiliki satu derajat lebih tinggi. Misalnya 2 + 3x + 5x^2 + 6x^3 + 7x^4 akan di divide menjadi
6. Polinom 1 : 2 + 3x
7. Polinom 2 : 5 + 6x + 7x^2
8. Proses divide ini akan terus dilakukan sampai polinom mecapai basis
9. Saat sudah mencapai basis maka polinom berderajat 0 itu akan dikalikan dan di combine (pada bagian return dari fungsi Divide and Conquer)
10. Pada saat combine ini lah jumlah operasi perkalian lebih sedikit dibanding dengan algoritma *Brute Force* sehingga memiliki kompleksitas yang lebih kecil
11. Intinya dari Algoritma Divide and Conquernya adalah
12. Divide : Terjadi saat pembagian polinom menjadi 2 buah bagian
13. Conquer : Terjadi saat polinom mencapai basis
14. Combiner : Terjadi saat return dari fungsi DNC

**Menghitung kompleksitas Algoritma Divide and Conquer**

Asumsikan n = 2k. lgX menandakan log2 x.

Dengan metode substitusi

....

Jadi kita memiliki

Dan

Sehingga Algoritma Divide and Conquer memiliki kompleksitas sbesar

1. **Kode Program**

**Brute Force**

Berisi Algoritma Brute force untuk menyelesaikan perkalian polinom

BF.hpp

|  |
| --- |
| #ifndef \_\_\_BF\_\_\_HPP  #define \_\_\_BF\_\_\_HPP  #include <vector>  using namespace std;  vector<int> BF(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah);  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Brute Force*  #endif |

BF.cpp

|  |
| --- |
| #include "BF.hpp"  #include <iostream>  #include <vector>  using namespace std;  vector<int> BF(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah){  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Brute Force*      int i,j;      \*cKali = 0;      \*cTambah = 0;      vector<int> pol3(pol1.size()+pol2.size());      for (i = 0; i < pol1.size(); i++){          for (j = 0; j< pol2.size(); j++){              pol3[i+j] += pol1[i] \* pol2[j];              \*cKali += 1;*// Menghitung jumlah operasi kali*              \*cTambah += 1;*// Menghitung jumlah operasi tambah*          }      }      return pol3;  } |

**Divide and Concquer**

Berisi Algoritma Divide and Conquer untuk menyelesaikan perkalian polinom

DNC.hpp

|  |
| --- |
| #ifndef \_\_\_DNC\_\_\_HPP  #define \_\_\_DNC\_\_\_HPP  #include <vector>  #include <iostream>  #include "other.hpp"  using namespace std;  vector<int> DNC(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah);  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Divide and Conquer*  #endif |

DNC.cpp

|  |
| --- |
| #include "DNC.hpp"  #include <iostream>  #include <vector>  using namespace std;  *// Output berupa array polinom hasil perkalian pol1 dan pol2 dengan metode Divide and Conquer*  vector<int> DNC(vector<int> pol1, vector<int> pol2, int \*cKali, int \*cTambah){  *// Basis jika polinom berderajat 0*      if(pol1.size() == 1 or pol2.size() == 1){          vector<int> pol3(1);          pol3[0] = pol1[0] \* pol2[0];          \*cKali += 1;*// Menghitung jumlah operasi kali*          return pol3;      }  *// Rekurens*      else{          int i;          int n = pol1.size()/2;          vector<int> A0, B0, A1, B1;          Divide(pol1, pol2, &A0, &B0, &A1, &B1, n);*// Membagi polinom menjadi dua bagian*            vector<int> Y = DNC(Sum(A0, A1), Sum(B0, B1), &\*cKali, &\*cTambah);          vector<int> U = DNC(A0, B0, &\*cKali, &\*cTambah);          vector<int> Z = DNC(A1, B1, &\*cKali, &\*cTambah);          \*cTambah += 6\*A1.size();*// Mengitung jumlah operasi tambah*          return Sum(Sum(Multi(Min(Min(Y, U), Z), n), U), Multi(Z, n\*2));      }  } |

**Other**

Berisi fungsi-fungsi tambahan yang digunakan dalam algoritma *Divide and Conquer*. Fungsi pada other ini dibutuhkan karena implementasi polinom pada pogram kali ini berupa array, sehingga untuk operasi seperti penambahan atau perkalian dibutuhkan fungsi khusus untuk menanganinya

Other.hpp

|  |
| --- |
| #ifndef \_\_\_OTHER\_\_\_HPP  #define \_\_\_OTHER\_\_\_HPP  #include <vector>  using namespace std;  void CetakPol (vector<int> pol);  *//Prosedur untuk mencetak polinom*  int max(int x, int y);  *//Fungsi untuk menghasilkan integer maksimum*  vector<int> Sum (vector<int> pol1, vector<int> pol2);  *//Fungsi untuk menjumlahkan polinom*  vector<int> Min (vector<int> pol1, vector<int> pol2);  *//Fungsi untuk mengurangkan polinom*  vector<int> Multi (vector<int> pol1, int n);  *//Fungsi untuk menaikkan derajat polinom sebanyak n*  void Divide (vector<int> pol1, vector<int> pol2, vector<int> \*A0, vector<int> \*B0, vector<int> \*A1, vector<int> \*B1, int n);  *//Membagi polinom menjadi dua bagian yang sama besar, jika ganjil dilebihkan pada polinom yang kedua*  #endif |

other.cpp

|  |
| --- |
| #include "other.hpp"  #include <iostream>  #include <vector>  using namespace std;  void CetakPol (vector<int> pol){  *//Prosedur untuk mencetak polinom*      int i;      for (i = 0; i < pol.size(); i++){          if (pol[i] != 0){              if (i == 0){                  cout << pol[i];              }              else if (i == 1){                  if (pol[i] < 0) cout << " - " << (-1\*pol[i]) << "x";                  else if (pol[i] > 0) cout << " + " << pol[i] << "x";              }              else{                  if (pol[i] < 0) cout << " - " << (-1\*pol[i]) << "x^" << i;                  else if (pol[i] > 0) cout << " + " << pol[i] << "x^" << i;              }          }      }      cout << endl;  }  int max(int x, int y){  *//Fungsi untuk menghasilkan integer maksimum*      return (x>y)? x: y;  }  vector<int> Sum (vector<int> pol1, vector<int> pol2){  *//Fungsi untuk menjumlahkan polinom*      int size = max(pol1.size(), pol2.size());      int i;      vector<int> sum(size);      for (int i = 0; i < pol1.size(); i++){          sum[i] = pol1[i];      }      for (int i = 0; i < pol2.size(); i++){          sum[i] += pol2[i];      }      return sum;  }  vector<int> Min (vector<int> pol1, vector<int> pol2){  *//Fungsi untuk mengurangkan polinom*      int size = max(pol1.size(), pol2.size());      int i;      vector<int> sum(size);      for (int i = 0; i < pol1.size(); i++){          sum[i] = pol1[i];      }      for (int i = 0; i < pol2.size(); i++){          sum[i] -= pol2[i];      }      return sum;  }  vector<int> Multi (vector<int> pol1, int n){  *//Fungsi untuk menaikkan derajat polinom sebanyak n*      int i;      vector<int> multi(pol1.size()+n);      for (i = 0; i < pol1.size(); i ++){          multi[i+n] = pol1[i];      }      return multi;  }  void Divide (vector<int> pol1, vector<int> pol2, vector<int> \*A0, vector<int> \*B0, vector<int> \*A1, vector<int> \*B1, int n){  *//Membagi polinom menjadi dua bagian yang sama besar, jika ganjil dilebihkan pada polinom yang kedua*      int i;      for(i = 0; i <= n-1; i++){          (\*A0).push\_back(pol1[i]);          (\*B0).push\_back(pol2[i]);      }      for(i = n; i < pol1.size(); i++){          (\*A1).push\_back(pol1[i]);          (\*B1).push\_back(pol2[i]);      }  } |

**Main**

Merupakan program utama yang menyatukan semua file di atas. Program akan meminta input berupa n yang merupakan panjang suku polinom

|  |
| --- |
| #include <iostream>  #include "DNC.hpp"  #include "BF.hpp"  #include <iostream>  #include <vector>  #include <stdio.h>  #include <chrono>  #include <cstdlib>  #include <ctime>  using namespace std;  using namespace std::chrono;  int main(){      int n, i,a ,b;      int cKali, cTambah;      vector<int> pol1;      vector<int> pol2;      vector<int> pol3;      vector<int> pol4;  *// Meminta input panjang suku polinom*      cout << "Masukkan panjang suku polinom : ";      cin >> n;      while (n < 0){          cout << "Minimal n bernilai 0" << endl;          cout << "Masukkan panjang suku polinom : ";          cin >> n;      }  *// Membangun angka random*      srand(time(0));      for (i = 0; i <= n; i++){          a = (rand() % 200) - 100;          b = (rand() % 200) - 100;          pol1.push\_back(a);          pol2.push\_back(b);      }      cout << "polinom 1  : "; CetakPol(pol1);      cout << "polinom 2  : "; CetakPol(pol2);      cout << endl;  *// Memulai algoritma brute force*      cout << "[ALGORITMA BRUTE FORCE]" << endl;  *// Menghitung durasi algoritma brute force*      auto startBF = high\_resolution\_clock::now();      pol3 = BF(pol1, pol2, &cKali, &cTambah);      auto stopBF = high\_resolution\_clock::now();      auto durationBF = duration\_cast<microseconds>(stopBF - startBF).count();  *// Mencetak hasil algoritma brute force*      cout << "hasil perklian polinom : "; CetakPol(pol3);      cout << "jumlah operasi kali    : " << cKali << endl;      cout << "jumlah operasi tambah  : " << cTambah << endl;      cout << "waktu yang dibutuhkan  : " << durationBF << " microseconds" << endl;      cout << endl;  *// Memulai algoritma divide and conquer*      cout << "[ALGORITMA DIVIDE AND CONQUER]" << endl;      cKali = 0;      cTambah = 0;  *// Menghitung durasi algoritma divide and conquer*      auto startDNC = high\_resolution\_clock::now();      pol4 = DNC(pol1, pol2, &cKali, &cTambah);      auto stopDNC = high\_resolution\_clock::now();      auto durationDNC = duration\_cast<microseconds>(stopDNC - startDNC).count();  *// Mencetak hasil algoritma divide and conquer*      cout << "hasil perklian polinom : "; CetakPol(pol4);      cout << "jumlah operasi kali    : " << cKali << endl;      cout << "jumlah operasi tambah  : " << cTambah << endl;      cout << "waktu yang dibutuhkan  : " << durationDNC << " microseconds" << endl;      cout << endl;      return 0;  } |

1. **Screenshot Input-Output Program**

Untuk n = 5

Untuk n = 10

Untuk n = 20

Untuk n = 50

**Analisis**

Dari hasil Screenshot di atas terlihat bahwa waktu yang dibutuhkan oleh algoritma *Divide and Conquer* lebih lama daripada algoritma *Brute Force.* Padahal secara teori seharusnya algoritma *Divide and Conquer* dengan kompleksitas O(nlog3) dan dengan umlah perkalian yang lebih sedikit akan menyelesaikan permasalah lebih cepat dibandung algoritma *Brute Force* dengan kompleksitas O(n2)

Hal ini terjadi karena implementasi program polinom menggunakan array, sehingga untuk menambahkan, mengurangi, membagi dan memangkatkan sebuah array program harus memangil fungsi lain lagi. Di mana fungsi-fungsi yang dipanggil ini juga memanfaatkan loop di dalamnya sehingga kompleksitas algoritmanya tidak murni O(nlog3) tetapi juga terbebani oleh kompleksitas dari fungsi-fungsi lain yang ikut dipanggil.

**Spesifikasi komputer yang digunakan**

Nama Laptop : Asus E202SA

OS : Lubuntu 19.04

RAM : 2048 MB / 2GB

Processor : Intel(R) Celeron(R) CPU N3060 @1.60Ghz (2 CPUs), ~1.6Ghz

|  |  |  |  |
| --- | --- | --- | --- |
| No | Point | 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 | ✓ |  |