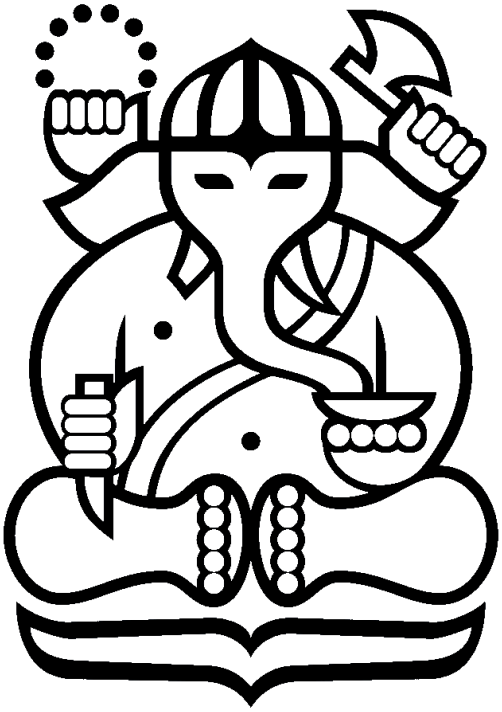
Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Semester II Tahun 2023/2024



Disusun oleh:

Albert Ghazaly- 13522150

INSTITUT TEKNOLOGI BANDUNG

2023

BAB I

Deskripsi Masalah

Kurva Bézier adalah kurva yang digunakan luas dalam desain grafis, animasi, dan manufaktur karena kemampuannya menciptakan kurva halus. Kurva ini dibentuk oleh titik-titik kontrol yang menentukan bentuk dan arahnya. Pembuatan kurva Bézier relatif mudah dengan menghubungkan titik-titik kontrolnya. Kurva ini memiliki aplikasi praktis dalam berbagai bidang, seperti alat pena dalam perangkat lunak desain grafis, animasi realistis, desain produk yang kompleks, dan pembuatan font yang artistik. Keunggulan kurva Bézier adalah kemampuannya untuk diubah dan dimanipulasi dengan mudah, sehingga memungkinkan pembuatan desain yang presisi dan sesuai dengan kebutuhan.

BAB II

Landasan Teori

2.1 Kurva *Bézier*

Sebuah Kurva *Bézier* didefinisikan oleh serangkaian titik kontrol, mulai dari P0 hingga Pn, dengan n merujuk pada order kurva (n = 1 untuk kurva linier, n = 2 untuk kurva kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) biasanya tidak berada pada kurva itu sendiri. Dalam gambar di atas, P0 merupakan titik kontrol pertama, sedangkan P3 merupakan titik kontrol terakhir. Titik kontrol P1 dan P2, yang terletak di antara, disebut titik kontrol antara yang tidak selalu berada pada kurva yang terbentuk.
Jika ada tiga titik, P0, P1, dan P2, maka akan dibentuk titik-titik diantara ketiga titik tersebut, yakni Q0 dan Q1 dengan persamaan:

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

Selanjutnya, lakukan substitusi untuk Q0 dan Q1:

$$R_0 = B(t) = (1-t)^2P_0 + (1-t)tP_1 + t^2P_2, \qquad t \in [0,1]$$

Persamaan di atas merupakan persamaan akhir untuk menghitung titik-titik kurva *Bézier*. Perlu diketahui bahwa kurva *Bézier* tidak sebatas tiga titik melainkan bisa lebih.

2.2 Algoritma *Brute Force*

Metode brute force merupakan pendekatan yang secara langsung digunakan untuk menyelesaikan suatu permasalahan dengan mengacu pada pernyataan permasalahan dan konsep yang terlibat. Algoritma ini menyelesaikan permasalahan secara sederhana, langsung, dan dengan cara yang jelas.

2.3 Algoritma *Divide and Conquer*

Algoritma Divide and Conquer merupakan pendekatan dalam pemrograman komputer yang digunakan untuk membagi permasalahan yang kompleks menjadi beberapa bagian yang lebih kecil dan lebih mudah untuk dipecahkan secara terpisah. Prinsipnya adalah memecah permasalahan utama menjadi submasalah yang lebih kecil, menyelesaikan masing-masing submasalah secara independen, dan kemudian menggabungkan solusi dari submasalah tersebut menjadi solusi akhir untuk permasalahan yang lebih besar.

BAB III

Analisis Algoritma

3.1 Kurva *Bézier* dengan Algoritma *Brute Force*

Metode *Brute Force* pada pembentukan kurva Bézier terletak pada hasil dari fungsi yang kurva Bézier pada 2.1. Metode ini, memasukkan sebanyak n buah input, tergantung iterasi yang diinginkan, berupa nilai t pada fungsi tersebut. Lalu, disimpan hasilnya yang nantinya akan digunakan untuk menggambar kurva tersebut. Untuk lebih jelasnya, berikut langkah-langkah

1. Cari banyaknya pasangan titik pada grafik dengan mencari jumlah t, t0,t1,t2, dan seterusnya ,yang akan digunakan dengan mengimplementasikan jumlah iterasi. Caranya adalah mencari jumlah titik hasil dengan menerapkan rumus sesuai pola, 3, 5, 9, 17, dan seterusnya, sesuai nilai jumlah iterasi (dimulai dari 1). Pola tersebut dapat disederhanakan menjadi fungsi rekursi
   $f(n) = 2, untuk\ n = 0$
   $f(n) = 2 * f(n-1) - 1, untuk\ n > 1$
   n: $jumlah\ iterasi,\ n > 0$
   $f(n)$: $jumlah\ titik\ yang\ dihasilkan\ pada\ iterasi\ ke - n,\ f(n) > 2$
2. Substitusi kan nilai t, dengan *range* 0 sampai 1 yang dipecah dengan selisih 1 ms atau 0.001 s, menggunakan persamaan yang telah diturunkan pada 2.1 dan simpan dalam sebuah larik, misal dinamakan T.

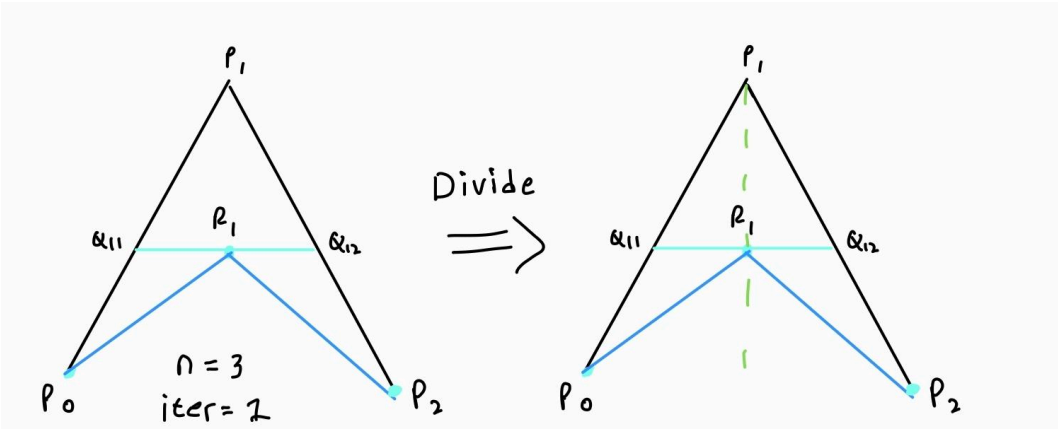
   $$R_0 = B(t) = (1-t)^2 P_0 + (1-t)tP_1 + t^2 P_2, \qquad t \in [0,1]$$

3. Pecah nilai t, dengan *range* 0 sampai 1, menjadi sebanyak jumlah titik yang telah dihitung pada langkah 1 dan simpan titik-titik tersebut pada larik, misal dinamakan R.
4. Ambil, pasangan titik-titik pada larik, yakni larik T, yang disimpan pada langkah 2 dengan kesamaan nilai t pada larik yang disimpan pada langkah 3, yakni larik R.
5. Petakan pasangan titik-titik yang didapatkan dari langkah 4 ke dalam grafik.

3.2 Kurva *Bézier* dengan Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* dapat diterapkan dalam implementasi nilai titik tengah untuk menyelesaikan dan menentukan titik kurva *Bézier*. Penerapannya terletak pada, pembagian titik-titik kurva menjadi bagian lebih kecilnya untuk dicari titik tengah dan diselesaikan.Untuk mempermudah penjelasan, perhatikan ilustrasi berikut. Terdapat tiga

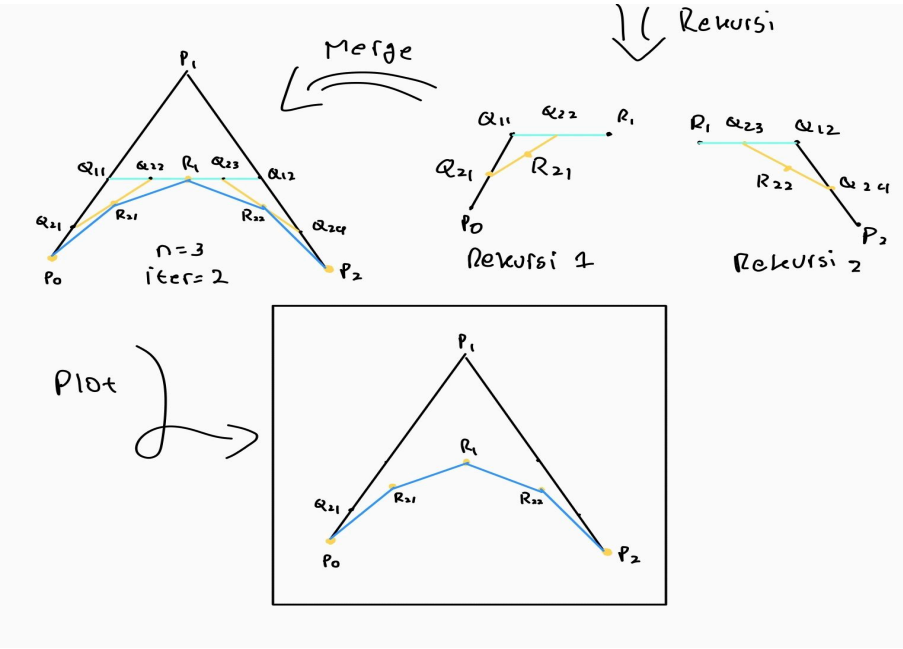
titik, yakni P0, P1, dan P2, berikut adalah langkah-langkah untuk menggambar kurva *Bézier* pada iterasi kedua.



Gambar 3.2.1 Ilustrasi Kurva Bézier dengan Algoritma Divide and Conquer

Pertama, dibuat titik baru yakni titik-titik tengah antara P0, P1, dan P2 yang disimpan pada Q11 dan Q12. Selanjutnya, buatlah garis baru yang menghubungkan titik-titik yang baru dibentuk (dalam kasus ini q11 dan q22), garis tersebut ditandai dengan warna hijau-biru terang. Karena tidak ada titik-titik tersisa yang dapat dihubungi, titik yang berada di tengah-tengah garis tersebut, adalah R1, yakni titik yang dihasilkan pada iterasi 1. Jadi, pada iterasi 1 titik-titik yang dihasilkan adalah P0,R1, dan P2, seperti ilustrasi di atas.

Untuk menentukan iterasi kedua, perlu diperhatikan bahwa titik-titik yang di dapat dan dimiliki dari iterasi pertama dapat menjadi landasan untuk menentukan titik-titik pada iterasi kedua. Caranya adalah dengan membagi bentuk menjadi dua bagian, yakni bagian 1 (P0,Q11, R1) dan bagian 2 (R1,Q12,P2) yang akan dikalkulasi menggunakan cara yang sama dengan pendekatan rekursi.



Gambar 3.2.2 Ilustrasi Kurva Bézier dengan Algoritma Divide and Conquer

Karena memiliki pola yang sama kedua bagian hasil pemecahan akan diolah menggunakan cara yang sama. Dengan begitu, kita menghitung titik yang dihasilkan dari dua hasil akhir pembagian bidang, yakni bagian 1 menghasilkan titik R21 dan bagian 2 menghasilkan titik R22. Setelah itu, kita menggabungkan hasil dari rekursi bagian 1 dan 2 bersama dengan titik-titik lain yang dihasilkan, seperti Q21 dan Q22. Akan tetapi, pada tahap ilustrasi akhir, titik yang akan dihasilkan dan digambar adalah P0,R21, R1,R22, dan P1.

3.3 Kurva *Bézier* dengan Titik Kontrol yang Banyak

3.3.1 Metode *Brute Force*

Untuk kasus kurva dengan titik kontrol yang banyak atau titik kontrol lebih dari 3. Maka, persamaan yang digunakan tentu berbeda karena bertambahnya titik kontrol. Oleh karena itu, kita akan menggunakan *Bernstein Basis Polynomial* untuk menentukan titik-titiknya. Berikut adalah langkah-langkah.

1. Cari banyaknya pasangan titik pada grafik dengan mencari jumlah t, t0,t1,t2, dan seterusnya ,yang akan digunakan dengan mengimplementasikan jumlah iterasi. Caranya adalah mencari jumlah titik hasil dengan menerapkan rumus sesuai pola, 3, 5, 9, 17, dan seterusnya, sesuai nilai jumlah iterasi (dimulai dari 1). Pola tersebut dapat disederhanakan menjadi fungsi rekursi

f(n) = 2 , untuk n =0
f(n) = 2*f(n-1) - 1 , untuk n>1
n: jumlah iterasi, n > 0
f(n): jumlah titik yang dihasilkan pada iterasi ke-n, f(n) > 2

2. Substitusi kan nilai t, dengan range 0 sampai 1 yang dipecah dengan selisih 1 ms atau 0.001 s, menggunakan persamaan *Bernstein form* dan simpan dalam sebuah larik, misal dinamakan T.

$$B(t) = \sum_{i=0}^{n} \beta_i b_{i,n}(t),$$

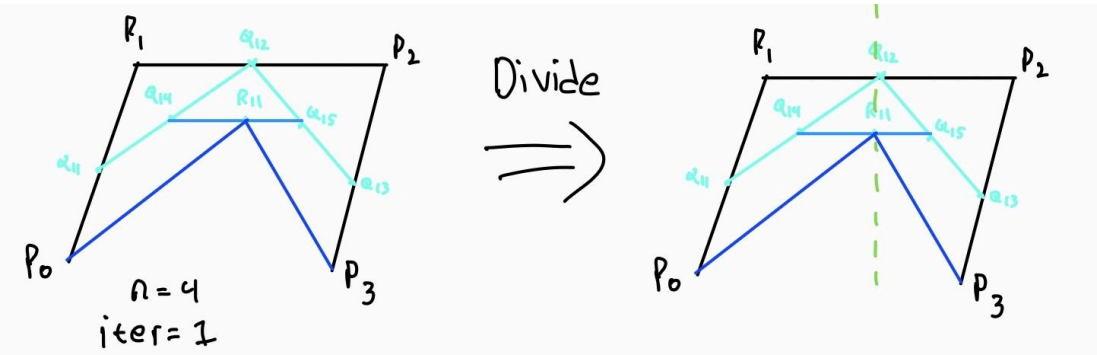
Dengan

$\beta_0, \ldots, \beta_n$ adalah titik kontrol dan

$$b_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i.$$

3. Pecah nilai t, dengan range 0 sampai 1, menjadi sebanyak jumlah titik yang telah dihitung pada langkah 1 dan simpan titik-titik tersebut pada larik, misal dinamakan R.
4. Ambil, pasangan titik-titik pada larik, yakni larik T, yang disimpan pada langkah 2 dengan kesamaan nilai t pada larik yang disimpan pada langkah 3, yakni larik R.
5. Petakan pasangan titik-titik yang didapatkan dari langkah 4 ke dalam grafik.

3.3.2 Metode Divide and Conquer

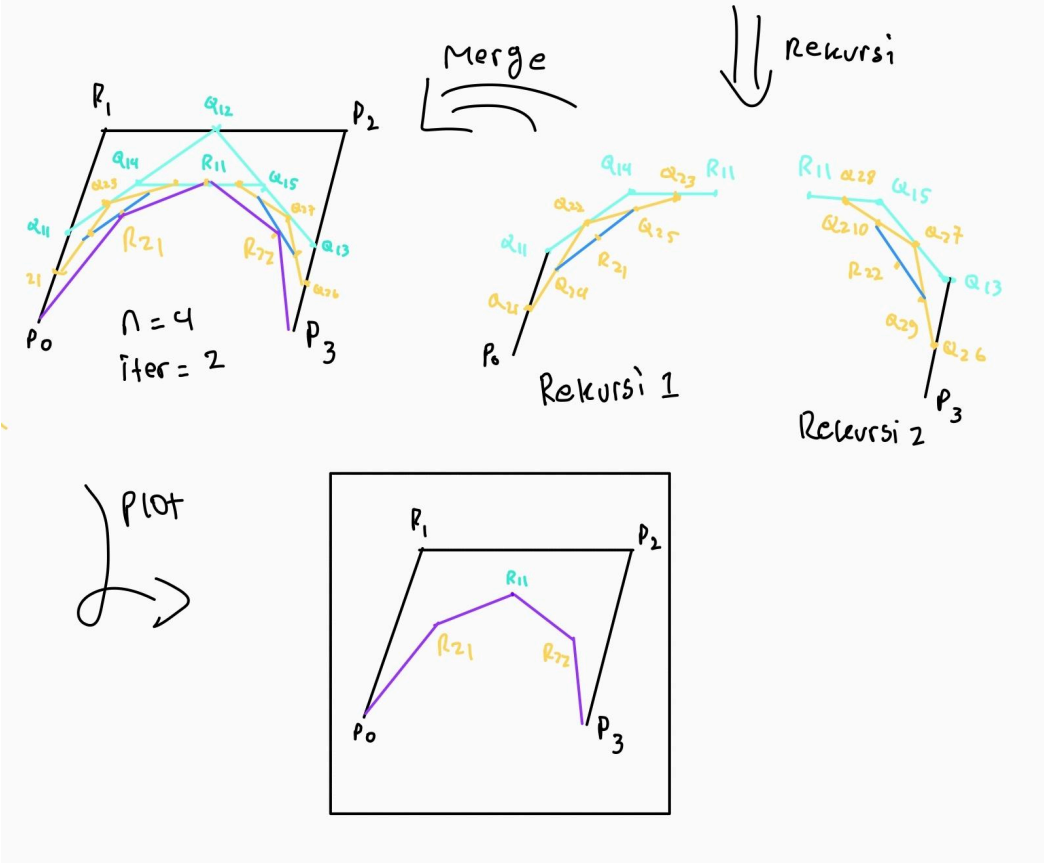


Gambar 3.3.2.1 Ilustrasi Kurva Bézier (n>3) dengan Algoritma Divide and Conquer

Pada titik kontrol lebih dari 3, titik baru juga dibuat sebagai titik tengah dari titik-titik yang sudah ada, tetapi perbedaan perhitungan terletak pada penyelesaian titik tengah. Titik tengah yang dihasilkan melibatkan

titik-titik yang baru dibuat yakni Q14 dan Q15 sehingga menghasilkan R11 dan tidak ada titik lain yang dapat dihubungkan lagi. Dengan begitu, pada iterasi 1 dihasilkan titik P0, R11, dan P3.

Akan tetapi, untuk menghitung titik-titik iterasi 2, kita akan membagi bidang menjadi dua bagian. Bagian 1, yakni (P0,Q11,Q14,R11), dan bagian 2, yakni (R11,Q15,Q13,P3), akan dipisah dan diproses dengan cara yang sama melalui pendekatan rekursi.



Gambar 3.3.2.2 Ilustrasi Kurva Bézier (n>3) dengan Algoritma Divide and Conquer

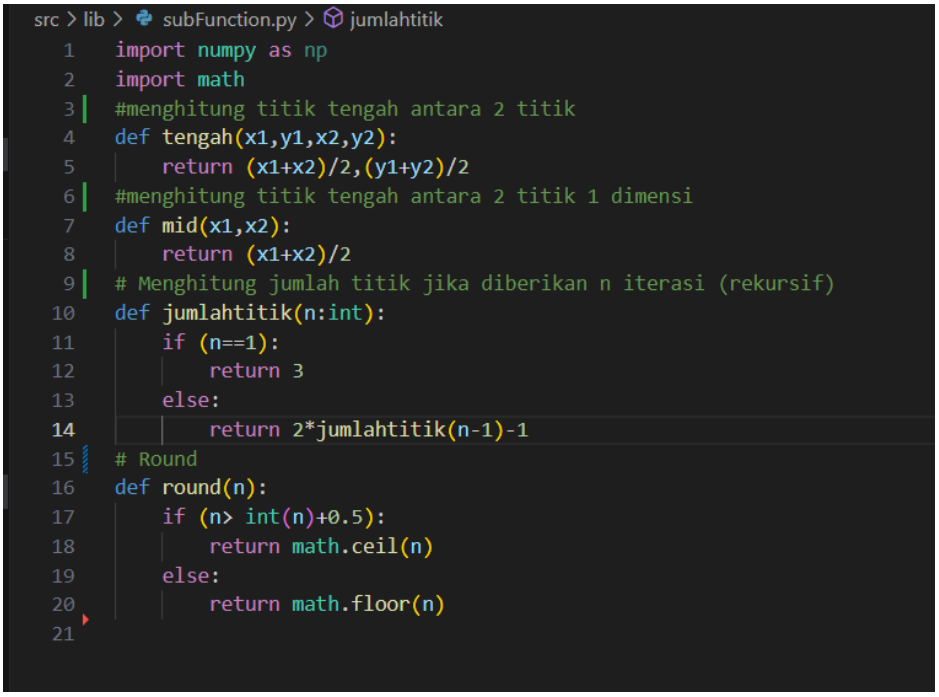
Karena memiliki pola yang sama kedua bagian hasil pemecahan akan diolah menggunakan cara yang sama. Dengan begitu, kita menghitung titik yang dihasilkan dari dua hasil akhir pembagian bidang, yakni bagian 1 menghasilkan titik R21 dan bagian 2 menghasilkan titik R22. Setelah itu, kita menggabungkan hasil dari rekursi bagian 1 dan 2 bersama dengan titik-titik lain yang dihasilkan, seperti Q21 dan Q22. Akan tetapi, pada tahap ilustrasi akhir, titik yang akan dihasilkan dan digambar adalah P0,R21, R11,R22, dan P1.

BAB IV
Implementasi dan Pengujian Program

4.1 Implementasi Program

4.1.1 subFunction.py

Fungsionalitas: Menyediakan fungsi-fungsi tambahan

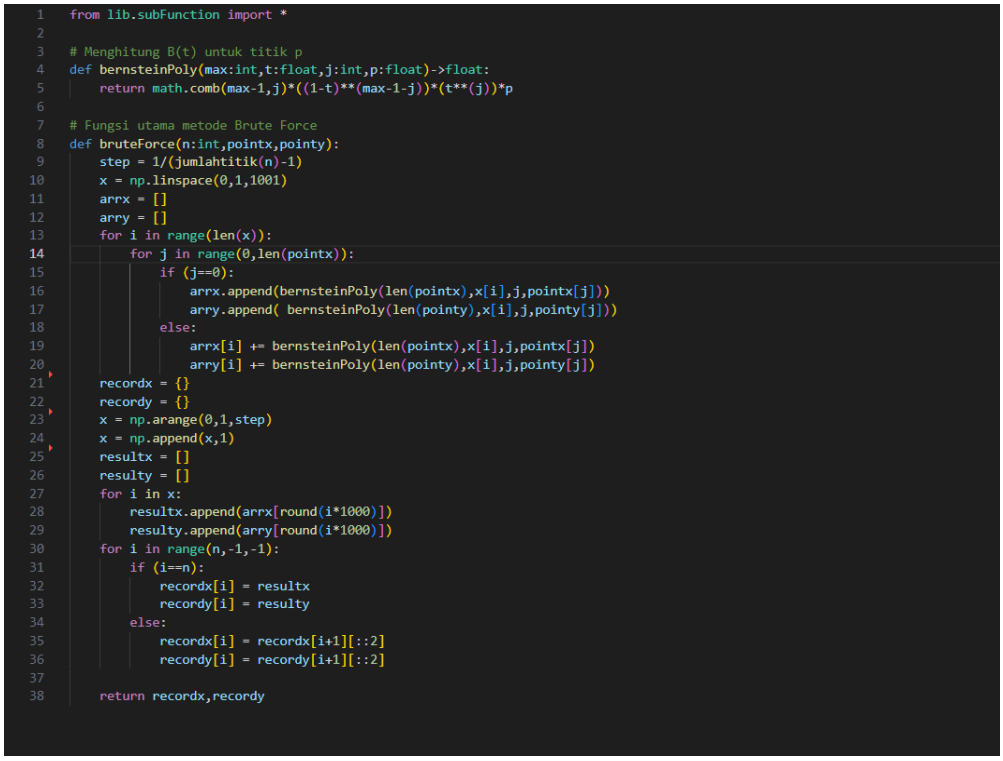


Gambar 4.1.1.1 Program subFunction.py

4.1.2 bruteForce.py

Fungsionalitas: Implementasi metode *Brute Force*
Deskripsi:

- n (iterasi): int
- pointx (titik x) : array of int
- pointy (titik y): array of int
- recordx (data titik x setiap iterasi) : dictionary
- recordy (data titik y setiap iterasi): dictionary



```python
from lib.subFunction import *

# Menghitung B(t) untuk titik p
def bernsteinPoly(max:int,t:float,j:int,p:float)->float:
    return math.comb(max-1,j)*((1-t)**(max-1-j))*(t**(j))*p

# Fungsi utama metode Brute Force
def bruteForce(n:int,pointx,pointy):
    step = 1/(jumlahtitik(n)-1)
    x = np.linspace(0,1,1001)
    arrx = []
    arry = []
    for i in range(len(x)):
        for j in range(0,len(pointx)):
            if (j==0):
                arrx.append(bernsteinPoly(len(pointx),x[i],j,pointx[j]))
                arry.append( bernsteinPoly(len(pointy),x[i],j,pointy[j]))
            else:
                arrx[i] += bernsteinPoly(len(pointx),x[i],j,pointx[j])
                arry[i] += bernsteinPoly(len(pointy),x[i],j,pointy[j])
    recordx = {}
    recordy = {}
    x = np.arange(0,1,step)
    x = np.append(x,1)
    resultx = []
    resulty = []
    for i in x:
        resultx.append(arrx[round(i*1000)])
        resulty.append(arry[round(i*1000)])
    for i in range(n,-1,-1):
        if (i==n):
            recordx[i] = resultx
            recordy[i] = resulty
        else:
            recordx[i] = recordx[i+1][::2]
            recordy[i] = recordy[i+1][::2]

    return recordx,recordy
```

Gambar 4.1.2.1 Program bruteForce.py

4.1.3 divAndConq.py

Fungsionalitas: Implementasi metode *Divide and Conquer*
Deskripsi:

- n (iterasi): int
- pointx (titik x) : array of int
- pointy (titik y): array of int
- recordx (data titik x setiap iterasi) : dictionary
- recordy (data titik y setiap iterasi): dictionary

```python
#Mencai titik-titik kontrol dan menyimpannya sekaligus hasil
def findControlPoint(point, result, cpoint, iter):
    length = len(point)
    if length == 2:
        midPoint = mid(point[0], point[1])
        result[:] = [point[0], midPoint, point[1]]
        cpoint[iter].append(result)

    else:
        midarr = []
        temp = []
        for i in range(length - 1):
            midPoint = mid(point[i], point[i + 1])
            midarr.append(midPoint)
            temp.extend([point[i], midPoint])
        temp.append(point[length - 1])


        resultN = []
        findControlPoint(midarr, resultN, cpoint, iter)

        cpoint[iter].append(temp)


        result[:] = [point[0], *resultN, point[length - 1]]

    return result

# Fungsi implementasi Divide and Conquer dalam mendekomposisi masalah menggunakan pendekatan rekursi
def dncRekursi(n:int, point, result, cpoint, max_val):
    length = len(point)
    if n == 0:
        result = []
    else:
        newPoint = []
        findControlPoint(point, newPoint, cpoint, max_val - n + 1)
        newLength = len(newPoint)
        resultL = []

        dncRekursi(n - 1, newPoint[0:newLength // 2 + 1], resultL, cpoint, max_val)
        resultR = []
        dncRekursi(n - 1, newPoint[newLength // 2:newLength], resultR, cpoint, max_val)
        result[:] = resultL + [newPoint[newLength // 2]] + resultR
```

Gambar 4.1.3.1 Potongan program divAndConq.py

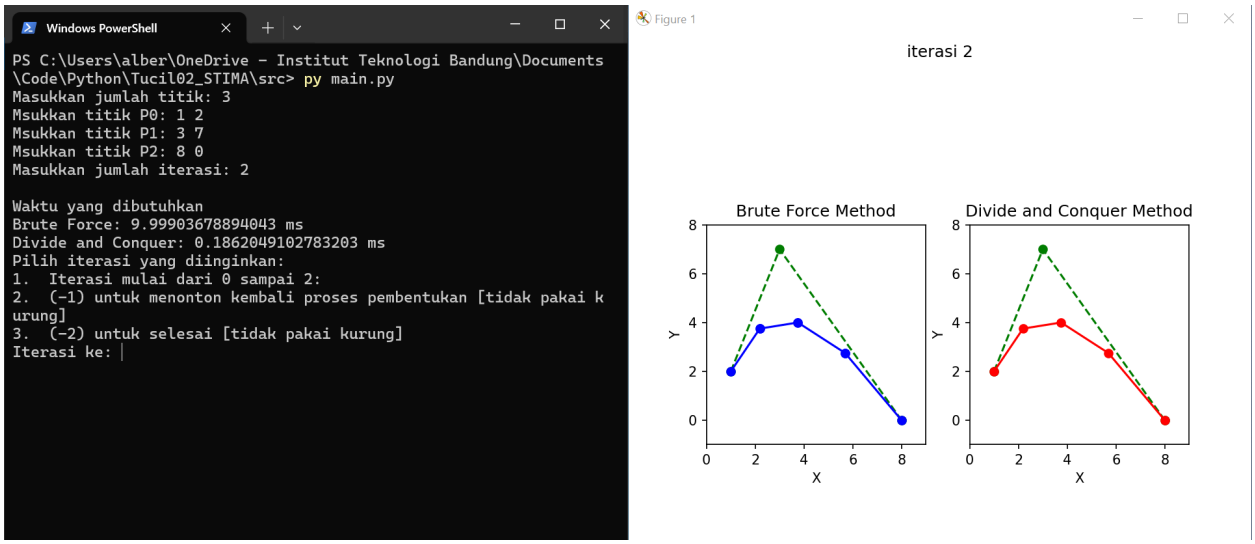
```python

# Fungsi utama metode Divide and Conquer
def dvnconq(n:int, pointx, pointy):
    recordx = {}
    recordy = {}
    cpointx = {}
    cpointy = {}
    for i in range(n):
        cpointx[i+1] = []
        cpointy[i+1] = []
    arrx = []
    arry = []
    dncRekursi(n, pointx, arrx, cpointx, n)
    dncRekursi(n, pointy, arry, cpointy, n)
    arrx = [pointx[0]] + arrx + [pointx[-1]]
    arry = [pointy[0]] + arry + [pointy[-1]]
    for i in range(n, -1, -1):
        if i == n:
            recordx[i] = arrx
            recordy[i] = arry
        else:
            recordx[i] = recordx[i + 1][::2]
            recordy[i] = recordy[i + 1][::2]

    return recordx, recordy, cpointx, cpointy

```

Gambar 4.1.3.2 Potongan program divAndConq.py

4.2 Pengujian Program

4.2.1 Test Case 1

X = (1,3,8)
Y = (2,7,0)
iter = 2

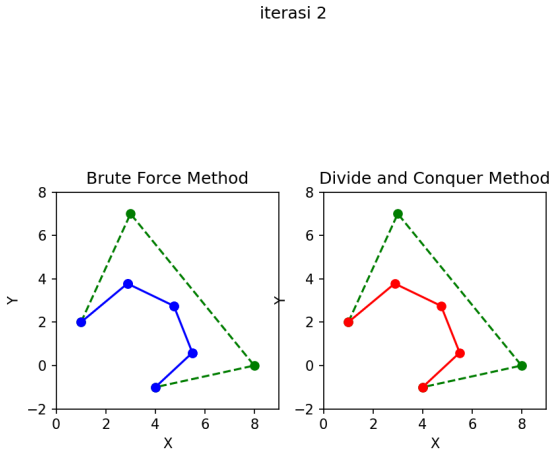


Gambar 4.2.1.1 Hasil test case 1

4.2.2 Test Case 2

X = (1,3,8,4)
Y = (2,7,0,-1)
iter = 2

```
PS C:\Users\alber\OneDrive - Institut Teknologi Bandung\Documents
\Code\Python\Tucil02_STIMA\src> py main.py
Masukkan jumlah titik: 4
Msukkan titik P0: 1 2
Msukkan titik P1: 3 7
Msukkan titik P2: 8 0
Msukkan titik P3: 4 -1
Masukkan jumlah iterasi: 2

Waktu yang dibutuhkan
Brute Force: 14.041900634765625 ms
Divide and Conquer: 0.0 ms
Pilih iterasi yang diinginkan:
1.  Iterasi mulai dari 0 sampai 2:
2.  (-1) untuk menonton kembali proses pembentukan [tidak pakai k
urung]
3.  (-2) untuk selesai [tidak pakai kurung]
Iterasi ke:
```

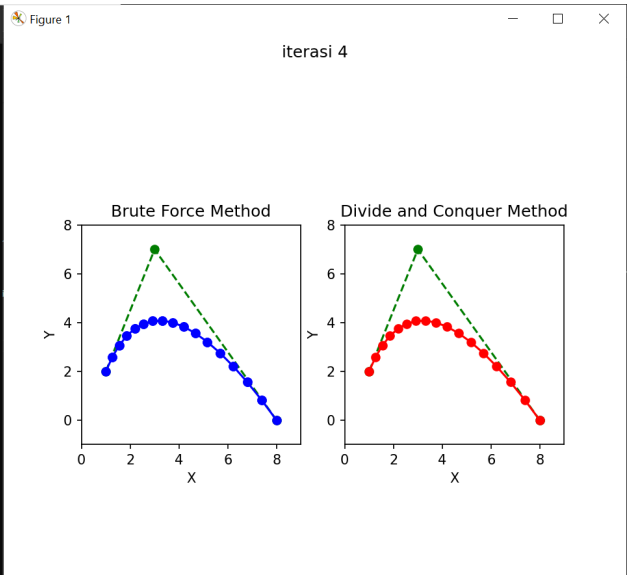


Gambar 4.2.2.1 Hasil test case 2

4.2.3 Test Case 3

X = (1,3,8)
Y = (2,7,0)
iter = 4

```
Windows PowerShell
PS C:\Users\alber\OneDrive - Institut Teknologi Bandung\Documents
\Code\Python\Tucil02_STIMA\src> py main.py
Masukkan jumlah titik: 3
Msukkan titik P0: 1 2
Msukkan titik P1: 3 7
Msukkan titik P2: 8 0
Masukkan jumlah iterasi: 4

Waktu yang dibutuhkan
Brute Force: 6.953716278076172 ms
Divide and Conquer: 13.000726699829102 ms
Pilih iterasi yang diinginkan:
1.  Iterasi mulai dari 0 sampai 4:
2.  (-1) untuk menonton kembali proses pembentukan [tidak pakai k
urung]
3.  (-2) untuk selesai [tidak pakai kurung]
Iterasi ke:
```

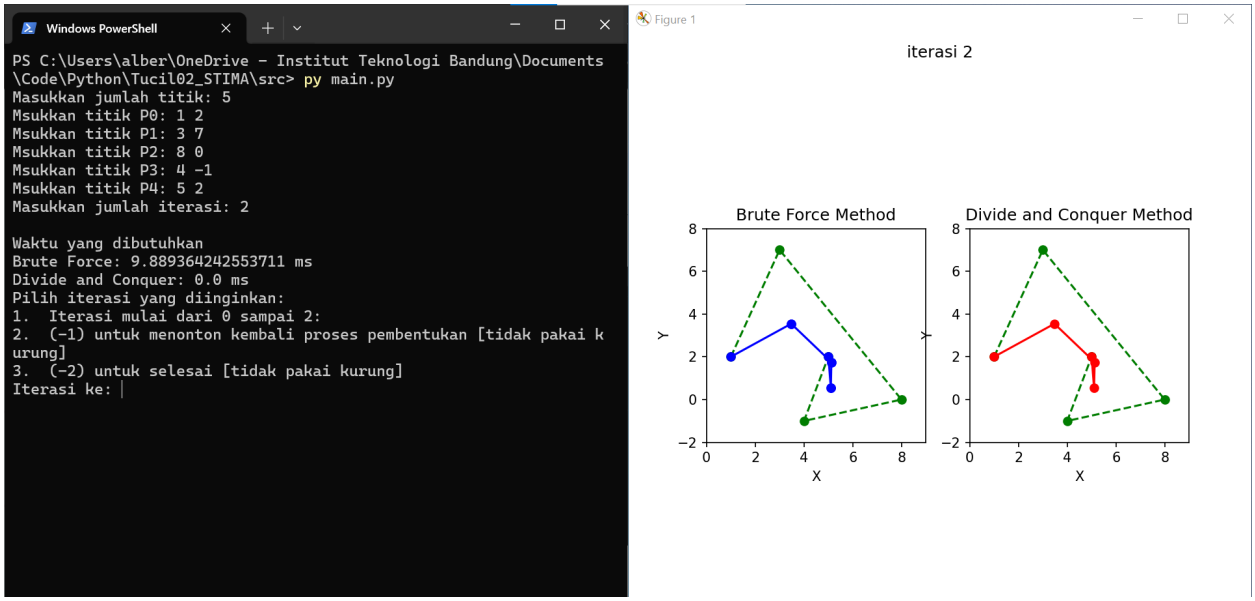


Gambar 4.2.3.1 Hasil test case 3

4.2.4 Test Case 4

X = (1,3,8,4,5)

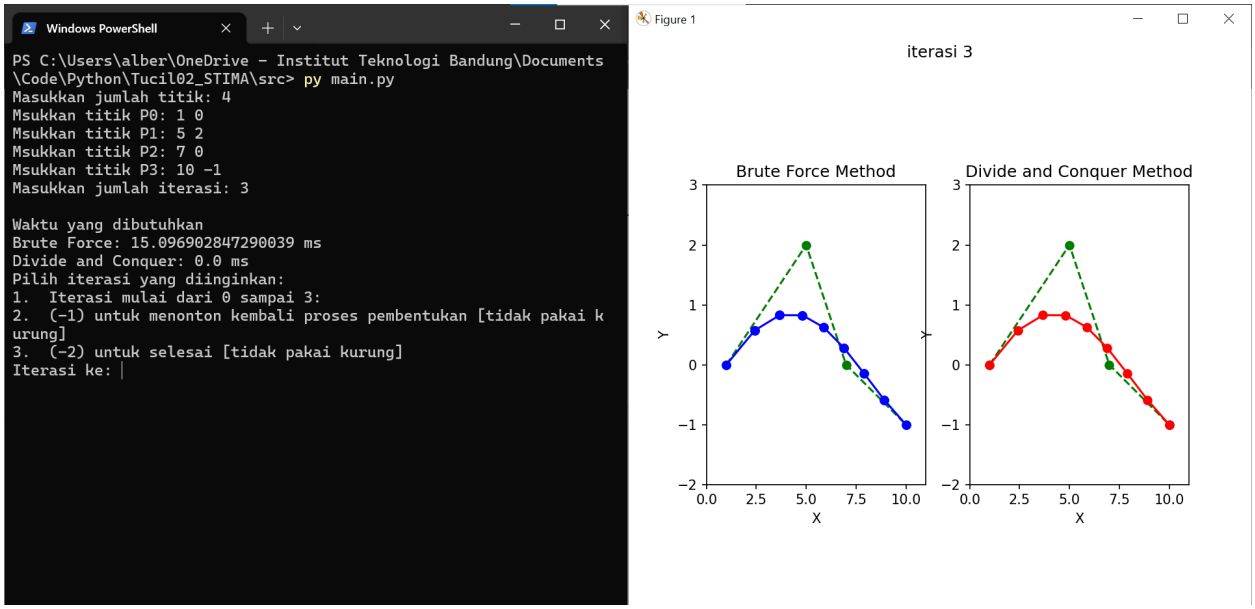
Y = (2,7,0,-1,2)
iter = 2



Gambar 4.2.4.1 Hasil test case 4

4.2.5 Test Case 5

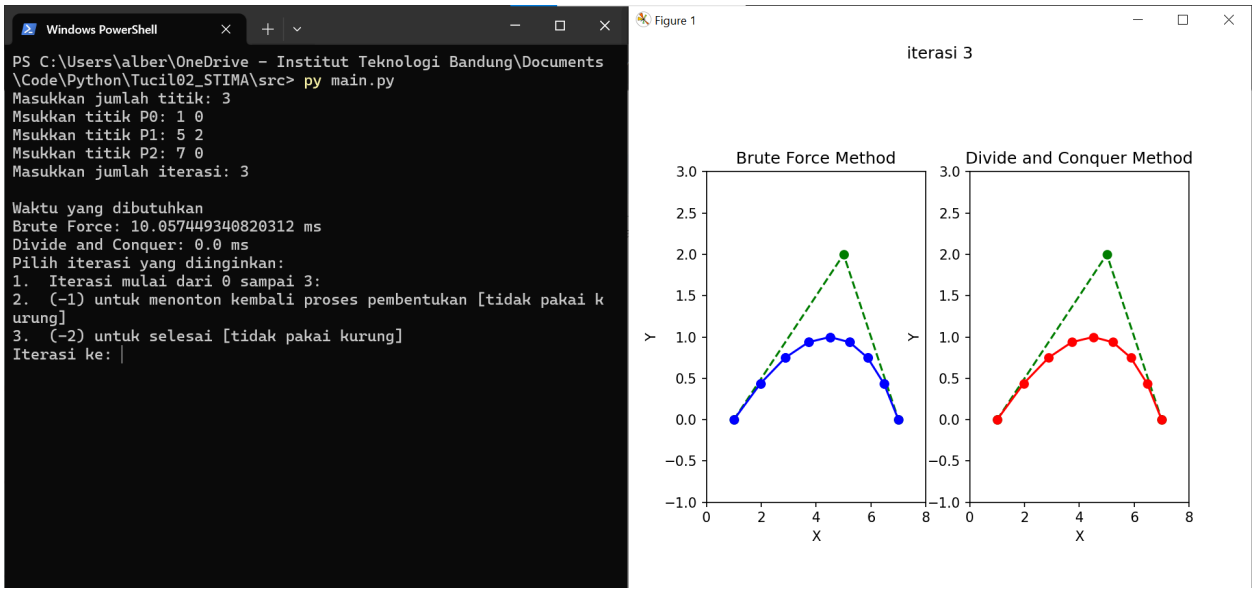
X = (1,5,7,10)
Y = (0,2,0,-1)
iter = 3



Gambar 4.2.5.1 Hasil test case 5

4.2.6 Test Case 6

X = (1,5,7)
Y = (0,2,0)
iter = 3



Gambar 4.2.6.1 Hasil test case 6

4.3 Analisis Hasil Pengujian

4.3.1 Kompleksitas *Brute Force*

Kompleksitas waktu dengan metode *Brute Force* tergantung dengan jumlah iterasi dan jumlah titik kontrol. Kita tetapkan, jumlah iterasi sebanyak m dan titik kontrol sebanyak n. Perlu kita ketahui bahwa jumlah titik yang digunakan dalam grafik selaras dengan 2^(m) + 1. menggunakan sebuah konstanta C untuk membagi nilai t [0,1] menjadi pecahan sebanyak C, dalam program ini C bernilai 1000 dengan asumsi jumlah titik hasil iterasi tidak akan melebihi 1000 titik, maka kompleksitas waktu maka kompleksitas waktu, dengan parameter perbandingannya adalah jumlah iterasi yang diperlukan untuk menghitung kurva adalah

T(n) = C*n
T(m,n) > n*(2^m + 1)

Maka

O(n) = C*n > n*(2^m + 1)
O(m,n) = n*(2^m)

Akan tetapi, dari segi penggunaan memori, algoritma *brute force* tidak efektif dan efisien karena metode ini menyimpan seluruh titik hasil berjumlah C buah yang di

dalamnya hanya beberapa titik yang akan ditampilkan dan diambil. Akan tetapi, metode *brute force* harus menyimpan semua karena harus dibandingkan dan disesuaikan dengan titik yang diminta, sesuai dengan penjelasan pada 3.1.

4.3.2 Kompleksitas *Divide and Conquer*

Menggunakan metode *Divide and Conquer* berbeda dengan metode sebelumnya. Metode ini berpusat pada visualisasi dan bentuk bidang yang membuatnya tidak harus mengandalkan persamaan dan menyimpan banyak titik seperti metode *brute force.* Kita tetapkan, jumlah iterasi sebanyak m dan titik kontrol sebanyak n. Perlu kita ketahui bahwa setiap mencari nilai tengah akhir atau R pada algoritma ini membutuhkan S(n-1) iterasi dan itu terus berlangsung selaras dengan jumlah iterasi yakni 2^m -1. Maka dari itu, kita dapat menghitung kompleksitas waktu dari algoritma tersebut

$T(m,n) = S(n-1)*(2^m-1)$

$T(m,n) = n*(n-1)/2* (2^m-1)$

Lalu

$O(m,n) = O(n^2 *2^m)$

Terlihat kompleks, akan tetapi penggunaan memori pada algoritma ini efektif dan efisien karena titik-titik yang disimpan merupakan titik-titik yang krusial.

4.3.2 Algoritma Paling Sesuai

Terdapat dua kriteria yang harus diperitmbangkan yakni *memory cost* dan *time complexity* karena kedua algoritma memiliki keunggulan dan kekurangan masing-masing.

Pertama, kita akan bahas mengenai *time complexity* dari kedua algoritma. Meskipun, Big O miliki *Brute Force* terlihat lebih simpel dari *Divide and Conquer*, hal tersebut tidak bisa membuktikan bahwa *Brute Force* lebih cepat kerna Big O *notation* merepresentasikan hubungan matematik dan algoritmik antara parameter n dan m. Oleh karena itu, kita dapat berpacu kepada T(n).

$C*n < n* (n-1)/2 * (2^m-1)$

$C < (n-1)*(2^m-1)/2$

Karakteristik yang dimiliki oleh algoritma *Divide and Conquer* adalah kepastian dalam menghitung kompleksitas karena jika C pada algoritma *Brute Force* lebih besar dari (n-1)*(2^m-1)/2 sesuai dengan pertidaksamaan di atas, maka algoritma *Divide and Conquer* lebih cepat dan sebaliknya. Dengan begitu, dapat disimpulkan bahwa untuk nilai iterasi rendah (m bernilai kecil), algoritma *Divide and Conquer* memiliki *time complexity* yang lebih rendah dari algoritma *Brute Force* dan sebaliknya.

Selanjutnya, mengenai *memory cost*, algoritma *Divide and Conquer* memakan *memory cost* lebih rendah relatif pada algoritma *Brute Force* seperti yang telah disebutkan pada 4.3.1 dan 4.3.2. Hal tersebut karena metode *Brute*

Force harus menyimpan semua titik, baik yang nantinya akan digunakan maupun yang tidak akan digunakan.

Oleh karena itu, algoritma yang paling sesuai dalam pertimbangan *time complexity* adalah *Brute Force* dengan syarat iterasi yang digunakan tidak sedikit, tetapi algoritma yang paling sesuai dalam pertimbangan *memory cost* adalah algoritma *Divide and Conquer* karena hanya menyimpan titik-titik yang akan digunakan.

Bab V
Kesimpulan

5.1 Kesimpulan

Dalam hal pertimbangan time complexity, Brute Force dapat lebih sesuai jika jumlah iterasi tidak terlalu banyak. Namun, dalam hal pertimbangan memory cost, algoritma Divide and Conquer lebih unggul karena membutuhkan memori yang lebih sedikit. Oleh karena itu, pemilihan algoritma tergantung pada kebutuhan spesifik masalah yang dihadapi, di mana trade-off antara time complexity dan memory cost harus diperhitungkan dengan cermat.

5.2 Saran

Untuk pembaca atau pengembang selanjutnya, dalam membuat algoritma, perlu dipertimbangkan desain awal agar algoritma tidak banyak masalah. Hal tersebut dapat mempercepat pembuatan algoritma. Selain itu, desain algoritma juga dapat menjadi acuan dalam berdiskusi dan memahami algoritma tersebut.

Lampiran

Link repository GitHub:
https://github.com/albert260302/Tucil2_13522150

Daftar Pustaka

Munir, Rinaldi. 2024. “Algoritma Brute Force (Bagian 1)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf (diakses 17 Maret 2024).

Munir, Rinaldi. 2024. “Algoritma Brute Force (Bagian 2)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf (diakses 17 Maret 2024).

Munir, Rinaldi. 2024. “Algoritma Divide and Conquer (Bagian 1)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf (diakses 17 Maret 2024).

Munir, Rinaldi. 2024. “Algoritma Divide and Conquer (Bagian 2)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf (diakses 17 Maret 2024).

Munir, Rinaldi. 2024. “Algoritma Divide and Conquer (Bagian 3)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf (diakses 17 Maret 2024).

Munir, Rinaldi. 2024. “Algoritma Divide and Conquer (Bagian 4)” di https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf (diakses 17 Maret 2024).