

LAPORAN TUGAS KECIL
MEMBANGUN KURVA BEZIER DENGAN ALGORITMA TITIK
TENGAH BERBASIS DIVIDE AND CONQUER
IF2211 STRATEGI ALGORITMA



Disusun oleh :
Maulvi Ziadinda Maulana (13522122)
Ikhwan Al Hakim (13522147)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023/2024

DAFTAR ISI

DAFTAR ISI.....	1
DAFTAR GAMBAR.....	3
BAB I DESKRIPSI MASALAH.....	5
1.1. Tujuan.....	5
1.2. Spesifikasi.....	5
BAB II DASAR TEORI.....	7
2.1. Algoritma Divide and Conquer.....	7
2.2. Kurva Bezier.....	7
BAB III ALGORITMA PEMBENTUKAN BEZIER CURVE.....	9
3.1 Algoritma Brute Force.....	9
3.2 Algoritma Divide and Conquer.....	9
3.3 Algoritma Bonus.....	16
BAB IV ANALISIS DAN IMPLEMENTASI.....	17
4.1. Implementasi Algoritma.....	17
4.2.1. Algoritma Divide and Conquer.....	17
4.2.2. Algoritma Brute Force.....	18
4.2. Analisis Kompleksitas Algoritma.....	19
4.2.1 Algoritma Brute Force.....	20
4.2.2 Algoritma Divide and Conquer.....	21
4.3. Pengujian.....	23
4.3.1. Three Points.....	23
4.3.2. N Points.....	29
4.4. Analisis.....	30
BAB V KESIMPULAN DAN SARAN.....	31
5.1. Kesimpulan.....	31

5.2. Saran.....	31
LAMPIRAN.....	32
DAFTAR PUSTAKA.....	33

DAFTAR GAMBAR

Gambar 3.2.1 Contoh Permasalahan Kurva Bezier.....	10
Gambar 3.2.2 Hasil Iterasi Pertama.....	11
Gambar 3.2.3 Hasil Iterasi Kedua.....	11
Gambar 3.2.4 Ilustrasi Fungsi newPoint.....	12
Gambar 3.2.5 Hasil Pemanggilan Fungsi newPoint.....	13
Gambar 3.2.6 Ilustrasi Pemanggilan Fungsi Divide and Conquer.....	14
Gambar 3.2.7 Ilustrasi Pemanggilan Fungsi Divide and Conquer dengan Iterasi 2.....	14
Gambar 3.2.8 Ilustrasi Pemanggilan Fungsi Divide and Conquer dengan Iterasi 1.....	15
Gambar 3.2.9 Hasil Divide and Conquer.....	15
Gambar 3.3.1 Ilustrasi pembentukan array visualisasi.....	16
Gambar 4.3.1.1.1 Titik Pertama ((2, 5), (6, 9), (3, 7)) dengan brute-force.....	23
Gambar 4.3.1.1.2 Titik Pertama ((2, 5), (6, 9), (3, 7)) dengan divide and conquer.....	23
Gambar 4.3.1.2.1 Titik Kedua ((4, 8), (1, 3), (5, 2)) dengan algoritma brute force.....	24
Gambar 4.3.1.2.2 Titik Kedua ((4, 8), (1, 3), (5, 2)) dengan algoritma divide and conquer.....	24
Gambar 4.3.1.2.2 Titik Ketiga ((7, 4), (2, 6), (9, 1)) dengan algoritma brute force.....	25
Gambar 4.3.1.2.2 Titik Ketiga ((7, 4), (2, 6), (9, 1)) dengan algoritma divide and conquer.....	25
Gambar 4.3.1.4.1 Titik Keempat ((5, 7), (1, 4), (9, 3)) dengan algoritma brute force.....	26
Gambar 4.3.1.4.2 Titik Keempat ((5, 7), (1, 4), (9, 3)) dengan algoritma divide and conquer.....	26
Gambar 4.3.1.5.1 Titik Kelima ((2, 8), (7, 5), (4, 1)) dengan algoritma brute force.....	27
Gambar 4.3.1.5.2 Titik Kelima ((2, 8), (7, 5), (4, 1)) dengan algoritma divide and conquer.....	27
Gambar 4.3.1.6.1 Titik Keenam ((6, 3), (3, 6), (8, 9)) dengan algoritma brute force.....	28
Gambar 4.3.1.6.2 Titik Keenam ((6, 3), (3, 6), (8, 9)) dengan algoritma divide and conquer.....	28
Gambar 4.3.2.1.1 Hasil kurva Bezier lima titik pertama dengan algoritma Divide and Conquer.	29
Gambar 4.3.2.1.2 Hasil kurva Bezier lima titik kedua dengan algoritma Divide and Conquer....	29

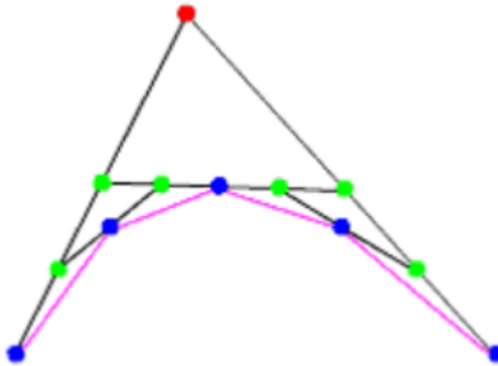
Gambar 4.4.1. Grafik hubungan waktu dengan jumlah iterasi dari kedua algoritma..... 30

BAB I

DESKRIPSI MASALAH

1.1. Tujuan

Tujuan dari pembuatan tugas kecil ini adalah mengimplementasikan algoritma titik tengah berbasis *divide and conquer* untuk membuat kurva Bezier. Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur.



Gambar 1. Kurva Bezier kuadratik dengan dua iterasi

Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

1.2. Spesifikasi

Terdapat beberapa spesifikasi yang harus diikuti pada pembuatan tugas kecil ini, diantaranya adalah:

1. Bahasa pemrograman yang digunakan harus C++/Python/JavaScript/Go.

2. Selain implementasi dalam algoritma *divide and conquer*, program dengan algoritma *brute force* juga harus dibuat sebagai pembandingan.
3. Tugas dapat dikerjakan secara individu atau berkelompok dengan anggota maksimal 2 orang.
4. Format masukan dibebaskan, dengan catatan dijelaskan pada README dan laporan. Komponen yang perlu menjadi masukan yaitu:
 - a. Tiga buah pasangan titik kontrol. Sebagai catatan, titik yang paling awal dimasukkan akan menjadi titik awal kurva dan titik yang terakhir dimasukkan akan menjadi titik akhir kurva.
 - b. Jumlah iterasi yang ingin dilakukan.
5. Hasil keluaran dari program adalah hasil kurva Bezier yang terbentuk dan juga waktu eksekusi pembentukan kurva.

Selain spesifikasi wajib, juga terdapat beberapa spesifikasi bonus, yaitu:

1. Melakukan generalisasi algoritma, ide, serta melakukan implementasinya sehingga program dapat membentuk kurva Bézier kubik, kuadratik, dan selanjutnya dengan 4, 5, 6, hingga n titik kontrol.
2. Memberikan visualisasi proses pembentukan kurva, sehingga tidak hanya hasil akhirnya saja. Tentu saja proses visualisasinya perlu melibatkan Graphical User Interface (GUI) yang kakasnya dibebaskan.

BAB II

DASAR TEORI

2.1. Algoritma Divide and Conquer

Algoritma divide and conquer adalah pendekatan dalam pemrograman dan matematika yang menguraikan masalah besar menjadi masalah yang lebih kecil, kemudian menyelesaikan setiap masalah kecil secara terpisah. Proses ini terus berlanjut hingga masalah yang tersisa cukup sederhana untuk dipecahkan dengan mudah.

Algoritma ini terdiri dari tiga langkah utama: pertama, "*divide*", di mana masalah dibagi menjadi dua atau lebih submasalah atau upa persoalan yang lebih kecil. Tiap-tiap upa-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakteristik persoalan semula namun berukuran lebih kecil sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif. Kedua, "*conquer*", di mana setiap submasalah diselesaikan secara rekursif. Dan ketiga, "*combine*", di mana solusi dari setiap submasalah digabungkan kembali untuk memberikan solusi akhir dari masalah asli.

Salah satu contoh penerapan algoritma divide and conquer adalah algoritma pencarian biner, yang mencari nilai dalam array terurut dengan membagi array menjadi setengah bagian pada setiap langkah dan mencari nilai target hanya di salah satu setengahnya. Jika nilai target lebih kecil dari elemen tengah array, pencarian dilakukan di setengah kiri; jika lebih besar, pencarian dilakukan di setengah kanan. Proses ini terus berlanjut secara rekursif hingga nilai target ditemukan atau seluruh array diperiksa. Algoritma ini memanfaatkan prinsip divide and conquer untuk mempercepat pencarian dalam data terurut.

2.2. Kurva Bezier

Kurva Bezier adalah salah satu jenis kurva matematis yang digunakan dalam bidang desain komputer grafis dan animasi. Kurva ini ditemukan oleh seorang insinyur dari perusahaan mobil Perancis, Pierre Bezier, pada tahun 1962. Kurva Bezier sering digunakan untuk membuat bentuk yang halus dan kompleks dalam grafika komputer. Kurva ini didefinisikan dengan sejumlah titik kontrol yang menentukan jalannya kurva,

yang disebut sebagai titik awal, titik akhir, dan titik-titik kontrol. Titik-titik kontrol ini adalah titik yang mengatur bentuk kurva diantara titik awal dan titik akhir. Kurva Bezier memiliki sifat yang fleksibel dan dapat digunakan untuk membuat berbagai jenis kurva, mulai dari kurva sederhana hingga kurva kompleks dengan menggunakan tingkat kehalusan dan kompleksitas yang berbeda.

Penerapan umum dari Kurva Bezier adalah dalam perangkat lunak grafis, di mana kurva ini digunakan untuk membuat objek seperti huruf, angka, logo, dan bentuk geometris lainnya. Dengan menggunakan titik kontrol, desainer dapat dengan mudah mengubah bentuk kurva dan membuatnya sesuai dengan kebutuhan desain. Selain itu, Kurva Bezier juga digunakan dalam animasi komputer untuk mengatur gerakan objek dan karakter dengan lebih halus dan alami. Keunggulan utama dari Kurva Bezier adalah kemampuannya untuk menghasilkan kurva yang halus dan tetap mempertahankan kontrol yang baik terhadap bentuk keseluruhan, membuatnya menjadi alat yang sangat berguna dalam desain grafis dan animasi.

BAB III

ALGORITMA PEMBENTUKAN BEZIER CURVE

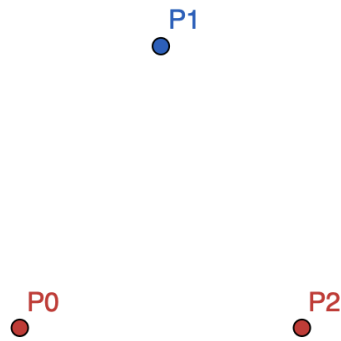
3.1 Algoritma Brute Force

Algoritma ini akan menggunakan pendekatan iteratif dalam membuat kurva bezier. Algoritma akan digunakan untuk membandingkan performa dari algoritma Divide and Conquer. Pada awalnya, algoritma menginisialisasi *array result* dengan titik pertama dan ketiga dari *array* titik kontrol. Apabila 3 titik pertama adalah P0, P1, dan P2, maka *result* akan berisi P0 dan P2. Selanjutnya, algoritma menghitung titik-titik tengah di antara setiap pasang titik kontrol awal menggunakan loop dan titik-titik tengah tersebut disimpan dalam *array middle*.

Langkah algoritma selanjutnya adalah mengulangi proses pembentukan titik tengah sesuai dengan iterasi yang diberikan. Pada setiap iterasi, algoritma menghitung titik-titik baru pada kurva Bezier dengan memanfaatkan titik-titik tengah yang telah dihitung sebelumnya. Proses ini dilakukan dengan menghitung titik tengah antara setiap pasang titik-titik tengah yang dihasilkan sebelumnya. Dengan demikian, setiap iterasi baru menghasilkan titik-titik yang membentuk kurva Bezier yang diinginkan. Selanjutnya, *array result* diperbarui dengan titik-titik baru yang dihasilkan setelah setiap iterasi. Begitu pula dengan *array middle*, yang diperbarui untuk menyimpan titik-titik tengah baru yang akan digunakan dalam iterasi berikutnya. Namun algoritma ini masih memiliki kekurangan yaitu belum bisa membuat kurva bezier untuk n titik.

3.2 Algoritma Divide and Conquer

Sebelumnya diperlukan gambaran masalah yang lebih detail teruntuk bisa membuat solusi permasalahan. Akan digunakan sebuah contoh masalah pembentukan kurva Bezier menggunakan 3 titik awal. Berikut adalah gambaran dari contoh permasalahan.



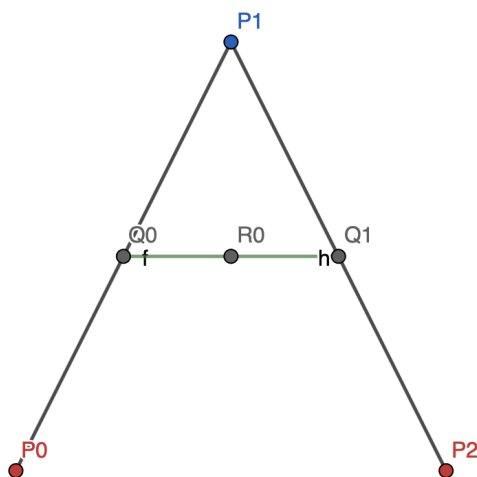
Gambar 3.2.1 Contoh Permasalahan Kurva Bezier

Akan digunakan algoritma yang sudah dijelaskan pada ilustrasi kasus Tugas Kecil

2. Algoritma akan bekerja dengan P1 adalah titik kontrol antara titik P0 dan P2. Untuk setiap iterasi akan dilakukan:

- a. Buatlah sebuah titik baru Q0 yang berada di tengah garis yang menghubungkan P0 dan P1, serta titik Q1 yang berada di tengah garis yang menghubungkan P1 dan P2.
- b. Hubungkan Q0 dan Q1 sehingga terbentuk sebuah garis baru.
- c. Buatlah sebuah titik baru R0 yang berada di tengah Q0 dan Q1.
- d. Buatlah sebuah garis yang menghubungkan P0 - R0 - P2.

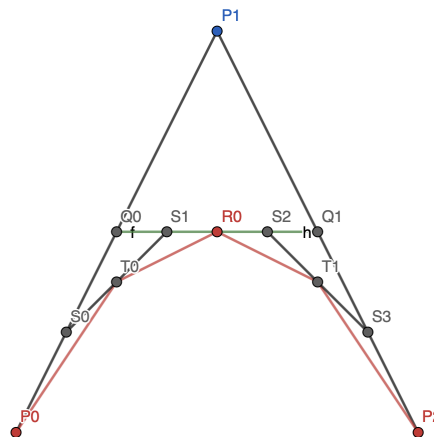
Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” dengan hasil sebagai berikut.



Gambar 3.2.2 Hasil Iterasi Pertama

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

- f. Buatlah beberapa titik baru, yaitu S_0 yang berada di tengah P_0 dan Q_0 , S_1 yang berada di tengah Q_0 dan R_0 , S_2 yang berada di tengah R_0 dan Q_1 , dan S_3 yang berada di tengah Q_1 dan P_2 .
- g. Hubungkan S_0 dengan S_1 dan S_2 dengan S_3 sehingga terbentuk garis baru.
- h. Buatlah dua buah titik baru, yaitu T_0 yang berada di tengah S_0 dan S_1 , serta T_1 yang berada di tengah S_2 dan S_3 .
- i. Buatlah sebuah garis yang menghubungkan $P_0 - T_0 - R_0 - T_1 - P_2$.



Gambar 3.2.3 Hasil Iterasi Kedua

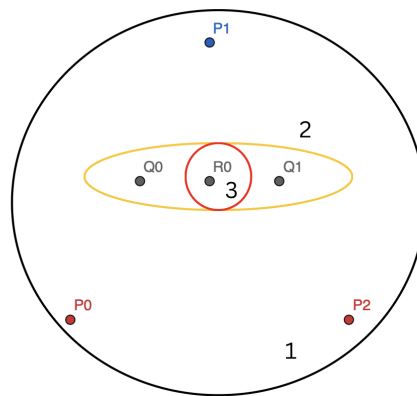
Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik.

Berdasarkan ilustrasi yang dilakukan dapat disimpulkan bahwa fungsi pembuatan titik baru hanya mengulangi langkah - langkah yang sama. Pada iterasi kedua, proses yang dilakukan sebenarnya sama dengan proses yang dilakukan pada iterasi pertama dengan titik P_0 , Q_0 , dan R_0 , dan R_0 sebagai titik kontrol. Maka dapat dibuat sebuah fungsi untuk melakukan proses tersebut, misalkan dengan nama **newControlPoint**.

Fungsi **newControlPoint** akan menjadi sebuah fungsi rekursif yang bertujuan untuk menghasilkan serangkaian titik kontrol baru berdasarkan titik kontrol awal yang

diberikan. Fungsi akan menerima parameter berupa kumpulan dari titik-titik kontrol awal yang akan digunakan sebagai input untuk menghasilkan titik kontrol baru. Berikut adalah algoritma dari fungsi tersebut.

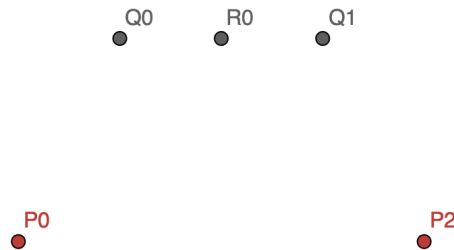
- a. Pertama, fungsi memeriksa apakah jumlah titik kontrol yang diberikan kurang dari atau sama dengan 1. Jika iya, maka fungsi akan langsung memberikan *return* titik tersebut. Kasus ini akan menjadi *base case* dari fungsi.
- b. Jika ada lebih dari satu titik kontrol, fungsi akan membuat array kosong *midPoints* yang akan digunakan untuk menyimpan titik-titik tengah antara setiap pasang titik kontrol.
- c. Selanjutnya akan dilakukan iterasi untuk mencari titik tengah dari setiap pasang titik kontrol yang berturut-turut dan menambahkannya ke dalam array *midPoints*.
- d. Setelah itu, fungsi memanggil dirinya sendiri secara rekursif dengan parameter baru yaitu *midPoints* tadi sebagai input untuk mencari titik-titik yang baru.
- e. Hasil dari rekursi ini dihubungkan (*concatenated*) dengan titik kontrol awal pertama dan terakhir.



Gambar 3.2.4 Ilustrasi Fungsi newPoint

Proses yang dilakukan fungsi **newControlPoint** dapat dijelaskan sebagai berikut. Akan digunakan titik *P0*, *P1*, dan *P2* yang sama dengan titik pada gambar 3.2.1. Pada awalnya fungsi **newControlPoint** akan dipanggil dengan parameter 3 titik tersebut. Fungsi akan menghasilkan *midPoints* yang berisi *Q0* dan *Q1*. Fungsi lalu akan melakukan rekursi dengan parameter *Q0* dan *Q1* yang akan menghasilkan *R0*. Fungsi akan kembali melakukan rekursi dengan parameter *R0*, namun karena parameternya

hanya 1 titik maka fungsi langsung melakukan *return R0*. Pada akhirnya akan dilakukan konkatenasi sehingga hasil akhirnya adalah *P0*, *Q0*, *R0*, *Q1*, dan *P1*.



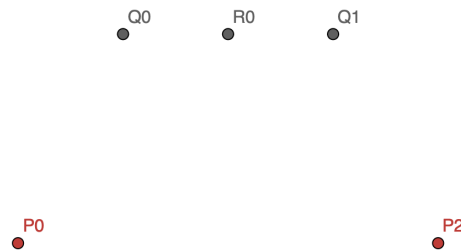
Gambar 3.2.5 Hasil Pemanggilan Fungsi `newPoint`

Fungsi **`newControlPoint`** tersebut akan menjadi fungsi *SOLVE*. Setelah didapatkan fungsi *SOLVE*, dapat dibuat algoritma Divide and Conquer untuk menyelesaikan permasalahan. Algoritma ini akan diterapkan menjadi sebuah fungsi dengan 2 parameter. Parameter pertama adalah kumpulan titik awal, dan parameter kedua adalah banyak iterasi. Algoritma ini akan bekerja sebagai berikut.

- a. Pertama, fungsi memanggil fungsi **`newControlPoint`** untuk mendapatkan titik-titik yang baru dari titik awal.
- b. Selanjutnya, fungsi memeriksa apakah jumlah iterasi yang tersisa kurang dari atau sama dengan 1. Jika iya, itu berarti algoritma telah mencapai tingkat rekursi yang diinginkan, sehingga fungsi langsung mengembalikan serangkaian titik kontrol yang diperbarui dari pemanggilan **`newControlPoint`**.
- c. Jika masih ada iterasi yang tersisa, fungsi membagi titik hasil fungsi **`newControlPoint`** yang menjadi dua bagian: *leftSlice* dan *rightSlice*. Bagian kiri (*leftSlice*) akan terdiri dari titik kontrol dari indeks 0 hingga indeks tengah, sedangkan bagian kanan (*rightSlice*) akan terdiri dari titik kontrol dari indeks tengah hingga indeks terakhir. Selanjutnya, fungsi secara rekursif memanggil dirinya sendiri untuk memproses bagian kiri dan kanan dari titik-titik tersebut dengan parameter iterasi yang dikurangi satu.

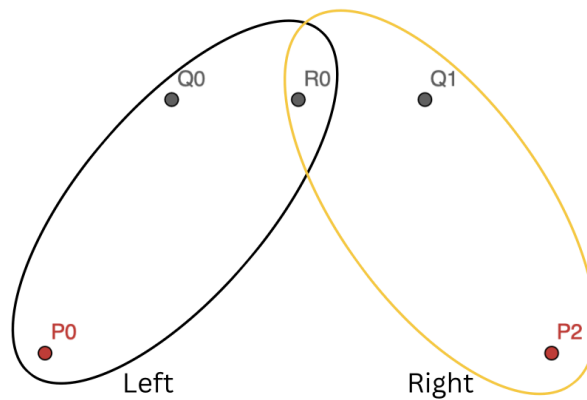
Hasil dari kedua pemanggilan rekursif di atas digabungkan dengan menggunakan metode *concat*, dengan mengambil semua elemen dari bagian kiri (*left*) dan semua

elemen dari bagian kanan kecuali elemen pertama (*right*). Berikut adalah contoh ilustrasi dari algoritma Divide and Conquer dengan point P0, P1, dan P2, dengan iterasi 2.



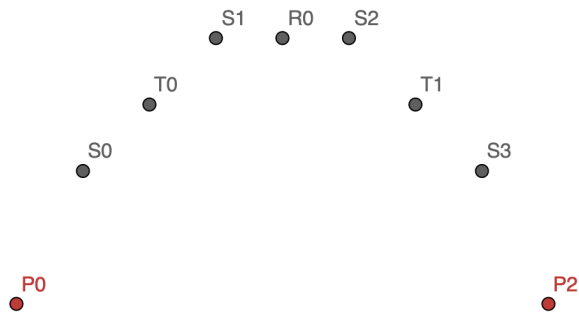
Gambar 3.2.6 Ilustrasi Pemanggilan Fungsi Divide and Conquer

Pada pemanggilan pertama dengan iterasi 2, akan dihasilkan titik-titik seperti diatas yang merupakan hasil dari fungsi **newControlPoint**. Karena iterasi saat ini masih lebih dari 1, maka akan dilakukan rekursi pada bagian *left* dan *right*.



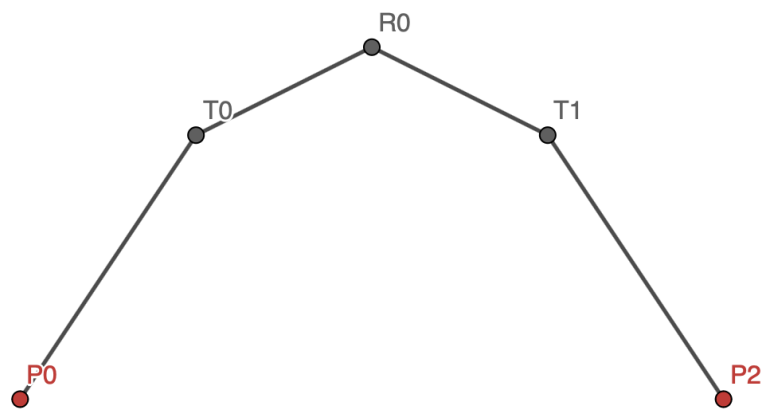
Gambar 3.2.7 Ilustrasi Pemanggilan Fungsi Divide and Conquer dengan Iterasi 2

Akan dilakukan proses yang sama dengan iterasi saat ini adalah 1 sehingga akan didapatkan hasil sebagai berikut.



Gambar 3.2.8 Ilustrasi Pemanggilan Fungsi Divide and Conquer dengan Iterasi 1

Dan di atas adalah hasil akhir dari fungsi Divide and Conquer. Namun hasil tersebut masih memiliki beberapa control point yang tidak perlu ditampilkan, dalam hal ini adalah S0, S1, S2, dan S3. Oleh karena itu akan dilakukan filter titik-titik hasil. Titik-titik yang akan ditampilkan pada Bezier Curve hanyalah titik-titik yang memiliki indeks yang habis dibagi dengan hasil dari banyak point di awal dikurangi 1. Contohnya adalah ketika banyak titiknya 3, maka titik yang diambil adalah titik yang memiliki indeks yang habis dibagi 2, sehingga hasilnya adalah sebagai berikut.



Gambar 3.2.9 Hasil Divide and Conquer

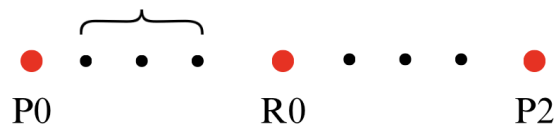
Proses pembuatan garis tidak dimasukkan ke dalam algoritma ini karena fungsi ini kan menggunakan *library* yang ada di *Javascript*. Implementasi yang lebih jelas dari algoritma ini terdapat pada bab 4.

3.3 Algoritma Bonus

Algoritma bonus adalah algoritma yang bisa meng-*handle* titik awal yang lebih dari 3. Namun sebenarnya algoritma yang diterapkan pada bagian 3.2 sudah bisa melakukan hal tersebut. Hal ini karena fungsi **newControlPoint** yang dibuat sudah melakukan iterasi untuk mencari titik tengah dari semua titik yang ada. Namun perlu diketahui bahwa generalisasi ini bisa berdampak pada performa algoritma menurun untuk kasus 3 titik. Hal ini akan dibahas lebih lanjut pada bab 4.

Selain itu ada juga algoritma yang digunakan untuk membuat visualisasi kurva pada setiap iterasi. Ketahui bahwa berapapun titik awal yang diberikan, pada iterasi n kurva pasti akan memiliki titik sebanyak $2^n - 1$. Sehingga untuk mencari 3 titik pada iterasi pertama, kita hanya perlu mengambil titik awal, titik tengah, dan titik akhir, atau dengan kata lain kita hanya perlu mengambil 3 titik yang bisa membagi *array* hasil menjadi 3 bagian sama besar. Untuk iterasi kedua yang memiliki 5 titik, kita hanya perlu mencari 5 titik yang bisa membagi *array* menjadi 4 bagian sama besar. Hal ini bisa terjadi karena titik-titik yang baru pasti akan berada diantara titik - titik yang lama. Misalkan ada 3 titik P0, R0, dan P2 pada hasil iterasi pertama, maka titik-titik baru yang akan dibentuk pada iterasi berikutnya pasti akan berada di antara P0 dan R0, dan diantara R0 dan P2. Berikut adalah ilustrasi dari analisis tersebut.

Titik-titik hasil iterasi selanjutnya



Gambar 3.3.1 Ilustrasi pembentukan array visualisasi

BAB IV

ANALISIS DAN IMPLEMENTASI

4.1. Implementasi Algoritma

Berikut adalah implementasi program dalam bahasa TypeScript.

4.2.1. Algoritma Divide and Conquer

```
function findMidPoint(p1: Point, p2: Point): Point {
    return {
        x: (p1.x + p2.x) / 2,
        y: (p1.y + p2.y) / 2,
    };
}

function newControlPoint(points: Point[]): Point[] {
    if (points.length <= 1) {
        return points;
    }

    const midPoints: Point[] = [];

    for (let i = 0; i < points.length - 1; i++) {
        midPoints.push(findMidPoint(points[i], points[i + 1]));
    }

    return [points[0], ...newControlPoint(midPoints), points[points.length - 1]];
}

function DivideAndConquer(points: Point[], iteration: number): Point[] {
    let temp = newControlPoint(points);

    if (iteration <= 1) {
        return temp;
    }
}
```

```

const leftSlice = temp.slice(0, points.length);
const rightSlice = temp.slice(points.length - 1, temp.length);

const left = DivideAndConquer(leftSlice, iteration - 1);
const right = DivideAndConquer(rightSlice, iteration - 1);

return left.concat(right.slice(1));
}

export function bezierCurves(points: Point[], iterations: number): Point[] {
  let result: Point[] = DivideAndConquer(points, iterations);

  return result.filter((_, index) => index % (points.length - 1) === 0);
}

```

4.2.2. Algoritma Brute Force

```

export function bezierCurves(points: Point[], iteration: number): Point[] {
  let result: Point[] = [points[0], points[2]];
  let middle: Point[] = [];

  for (let i = 0; i < points.length - 1; i++) {
    let temp: Point = { x: 0, y: 0 };
    temp.x = 0.5 * points[i].x + 0.5 * points[i + 1].x;
    temp.y = 0.5 * points[i].y + 0.5 * points[i + 1].y;
    middle.push(temp);
  }

  for (let i = 1; i < iteration; i++) {
    let resultTemp: Point[] = [];
    resultTemp.push(result[0]);
    for (let j = 0; j < middle.length - 1; j++) {
      let temp: Point = { x: 0, y: 0 };
      temp.x = 0.5 * middle[j].x + 0.5 * middle[j + 1].x;
      temp.y = 0.5 * middle[j].y + 0.5 * middle[j + 1].y;
    }
  }
}

```

```

        resultTemp.push(temp);
    }
    resultTemp.push(result[result.length - 1]);
    result = resultTemp.slice();

    let middleTemp: Point[] = [];
    while (middle.length !== 0) {
        let temp: Point = { x: 0, y: 0 };
        temp.x = 0.5 * resultTemp[0].x + 0.5 * middle[0].x;
        temp.y = 0.5 * resultTemp[0].y + 0.5 * middle[0].y;
        middleTemp.push(temp);
        if (middleTemp.length % 2 === 1) {
            resultTemp.shift();
        } else {
            middle.shift();
        }
    }
    middle = middleTemp.slice();
}

let resultTemp: Point[] = [];
resultTemp.push(result[0]);
for (let j = 0; j < middle.length - 1; j++) {
    let temp: Point = { x: 0, y: 0 };
    temp.x = 0.5 * middle[j].x + 0.5 * middle[j + 1].x;
    temp.y = 0.5 * middle[j].y + 0.5 * middle[j + 1].y;
    resultTemp.push(temp);
}
resultTemp.push(result[result.length - 1]);
result = resultTemp.slice();
return result; }

```

4.2. Analisis Kompleksitas Algoritma

Berikut adalah analisis kompleksitas dari tiap algoritma.

4.2.1 Algoritma Brute Force

Karena algoritma ini hanya terdiri dari satu fungsi saja, maka kita akan melakukan analisis per bagian kodenya.

```
for (let i = 0; i < points.length - 1; i++) {  
  let temp: Point = { x: 0, y: 0 };  
  temp.x = 0.5 * points[i].x + 0.5 * points[i + 1].x;  
  temp.y = 0.5 * points[i].y + 0.5 * points[i + 1].y;  
  middle.push(temp);  
}
```

Anggap panjang array *points* sebagai n , maka kompleksitas di bagian ini adalah:

$$T(n) = O(n - 1) \approx O(n)$$

Setelah itu bagian selanjutnya adalah:

```
for (let i = 1; i < iteration; i++) {  
  let resultTemp: Point[] = [];  
  resultTemp.push(result[0]);  
  for (let j = 0; j < middle.length - 1; j++) {  
    let temp: Point = { x: 0, y: 0 };  
    temp.x = 0.5 * middle[j].x + 0.5 * middle[j + 1].x;  
    temp.y = 0.5 * middle[j].y + 0.5 * middle[j + 1].y;  
    resultTemp.push(temp);  
  }  
}
```

Disini dilakukan pengulangan sebanyak jumlah iterasi dikalikan dengan panjang array *middle*. Anggap jumlah iterasi sebagai m dan dari bagian sebelumnya sudah diketahui bahwa panjang array *middle* adalah $n-1$. Maka kompleksitas pada bagian ini adalah:

$$T(n) = O(m) \times O(n - 1) \approx O(mn)$$

```
resultTemp.push(result[result.length - 1]);  
result = resultTemp.slice();
```

Setelah itu dilakukan *slicing* array *resultTemp* yang memiliki panjang n sehingga kompleksitasnya adalah:

$$T(n) = O(n)$$

```
let middleTemp: Point[] = [];  
while (middle.length !== 0) {  
  let temp: Point = { x: 0, y: 0 };  
  temp.x = 0.5 * resultTemp[0].x + 0.5 * middle[0].x;  
  temp.y = 0.5 * resultTemp[0].y + 0.5 * middle[0].y;  
  middleTemp.push(temp);  
  if (middleTemp.length % 2 === 1) {  
    resultTemp.shift();  
  } else {  
    middle.shift();  
  }  
}  
middle = middleTemp.slice();  
}
```

Kemudian dilanjutkan dengan pengulangan sebanyak panjang array *middle* dan slicing array *middleTemp*. Dari bagian sebelumnya, sudah diketahui panjang array *middle* yaitu $n-1$ dan karena array *middleTemp* dipush sebanyak jumlah pengulangan yang sama, maka panjangnya juga $n-1$. Maka kompleksitas di bagian ini adalah:

$$T(n) = O(n - 1) \times O(n - 1) \approx O(n^2)$$

Karena itu semua masih berada dalam pengulangan yang sama dengan pengulangan iterasi diawal, maka dapat kita hitung kompleksitas totalnya yaitu:

$$T(n) = O(mn) \times O(n) \times O(n^2) = O(n^4 m)$$

Dimana n adalah jumlah titik dan m adalah jumlah iterasi.

4.2.2 Algoritma Divide and Conquer

Untuk menghitung kompleksitas algoritma dengan n titik dan m iterasi, setiap fungsi dapat dianalisis secara terpisah:

a. **findMidPoint**(p1, p2):

Fungsi ini melakukan operasi sederhana untuk menemukan titik tengah antara dua titik sehingga kompleksitasnya adalah:

$$T(n) = 2$$

b. **newControlPoint**(points):

```
for (let i = 0; i < points.length - 1; i++) {
  midPoints.push(findMidPoint(points[i], points[i + 1]));
}
```

Anggaplah *array* yang dimasukkan sebagai parameter memiliki panjang n sehingga iterasi melalui titik-titik untuk menghitung titik tengah adalah $T(n) = 2 * (n - 1)$.

```
return [points[0], ...newControlPoint(midPoints), points[points.length - 1]];
```

Selanjutnya Rekursi terjadi pada setiap iterasi untuk memanggil dirinya sendiri dengan parameter hasil titik tengah, ketahuilah bahwa panjang array *midpoints* adalah $n - 1$. Fungsi ini akan terus menerus dipanggil hingga array parameter hanya memiliki panjang 1. Sehingga kompleksitas totalnya adalah

$$T(n) = T(n - 1) + T(n - 2) + \dots + 1$$

$$T(n) = \sum_{i=1}^{n-1} T(n-i)$$

$$T(n) = \sum_{i=1}^{n-1} 2 \times (n-i-1)$$

$$T(n) = 2 \times \frac{(n-1)n}{2} = n^2 - n.$$

c. **DivideAndConquer**(points, iteration):

Ketika dilakukan pemanggilan fungsi newControlPoint, kompleksitasnya adalah

$$T(n) = n^2 - n.$$

Pemanggilan rekursif terjadi dalam dua bagian setelah membagi array menjadi 2. Namun ketahuilah bahwa pemanggilan newControlPoint menghasilkan array yang memiliki panjang $2 \times n - 1$ sehingga pemanggilan rekursi 2 fungsi selanjutnya pasti akan selalu sama yaitu points dengan banyak n . Rekursi yang terjadi akan membentuk deret geometri yaitu $1, 2, 3, \dots, 2^{m-1}$ sehingga kompleksitas totalnya adalah

$$T(n) = \sum_{i=0}^{m-1} 2^i \times (n^2 - n)$$

gunakan rumus deret geometri sehingga didapat

$$T(n) = \frac{1(2^m-1)}{2} \times (n^2 - n)$$

$$T(n) = \frac{1}{2} \times (2^m n^2 - 2^m n - n^2 + n)$$

$$T(n) \approx O(n^2 2^m).$$

d. **bezierCurves**(points, iterations):

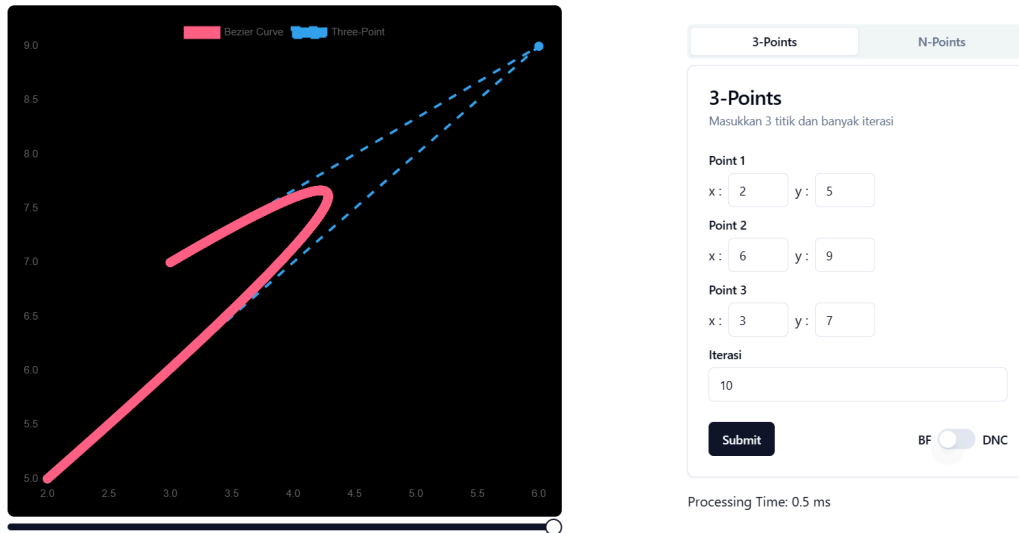
Pemanggilan fungsi DivideAndConquer(): $O(n^2 2^m)$

4.3. Pengujian

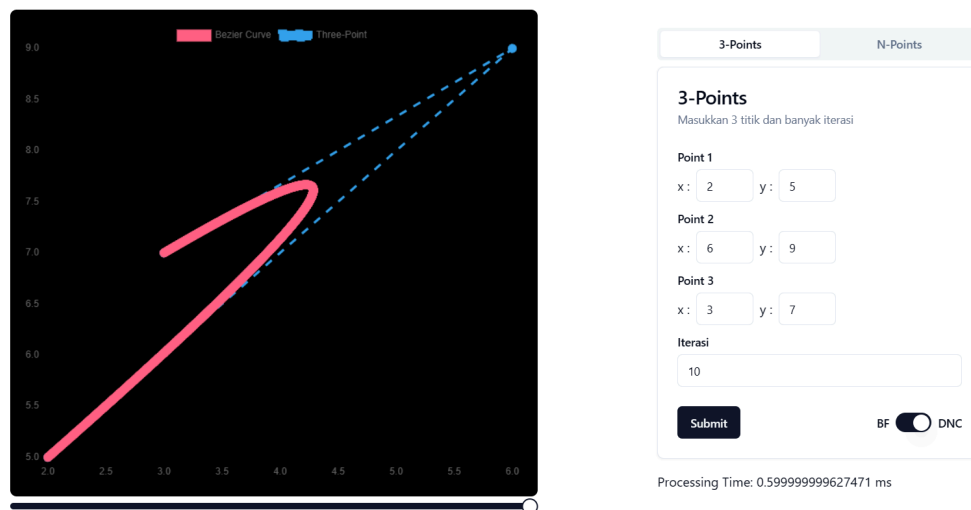
Kami melakukan pengujian program kami dengan beberapa titik. Berikut adalah hasil kurva Bezier yang dikeluarkan oleh program kami:

4.3.1. Three Points

4.3.1.1. Titik Pertama ((2, 5), (6, 9), (3, 7))

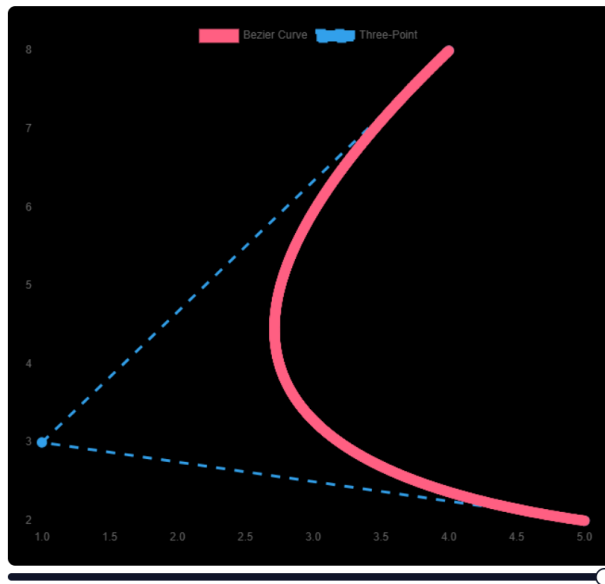


Gambar 4.3.1.1.1 Titik Pertama ((2, 5), (6, 9), (3, 7)) dengan brute-force



Gambar 4.3.1.1.2 Titik Pertama ((2, 5), (6, 9), (3, 7)) dengan divide and conquer

4.3.1.2. Titik Kedua ((4, 8), (1, 3), (5, 2))



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x : y :

Point 2
x : y :

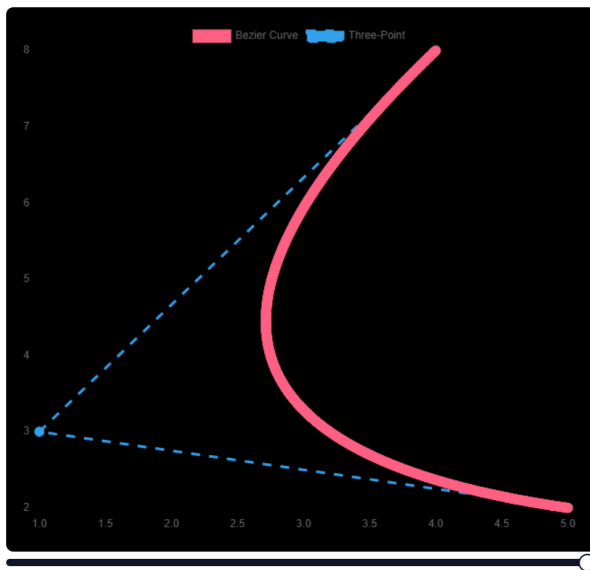
Point 3
x : y :

Iterasi

☒ BF ☐ DNC

Processing Time: 0.40000000037252903 ms

Gambar 4.3.1.2.1 Titik Kedua ((4, 8), (1, 3), (5, 2)) dengan algoritma brute force



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x : y :

Point 2
x : y :

Point 3
x : y :

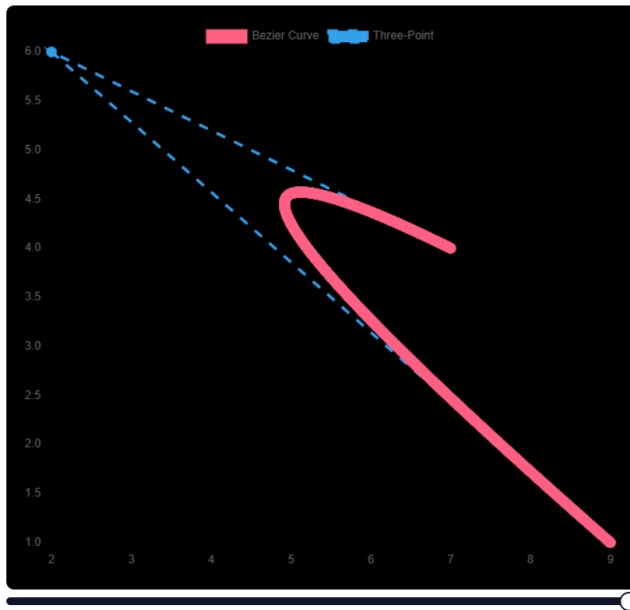
Iterasi

☐ BF ☒ DNC

Processing Time: 0.2999999988824129 ms

Gambar 4.3.1.2.2 Titik Kedua ((4, 8), (1, 3), (5, 2)) dengan algoritma divide and conquer

4.3.1.3. Titik Ketiga ((7, 4), (2, 6), (9, 1))



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x: y:

Point 2
x: y:

Point 3
x: y:

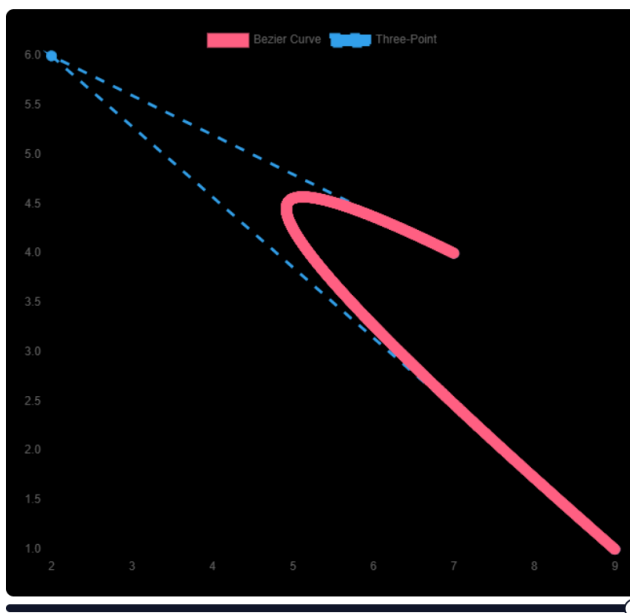
Iterasi

Submit

BF ☐ DNC ☒

Processing Time: 0.3999999985098839 ms

Gambar 4.3.1.2.2 Titik Ketiga ((7, 4), (2, 6), (9, 1)) dengan algoritma brute force



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x: y:

Point 2
x: y:

Point 3
x: y:

Iterasi

Submit

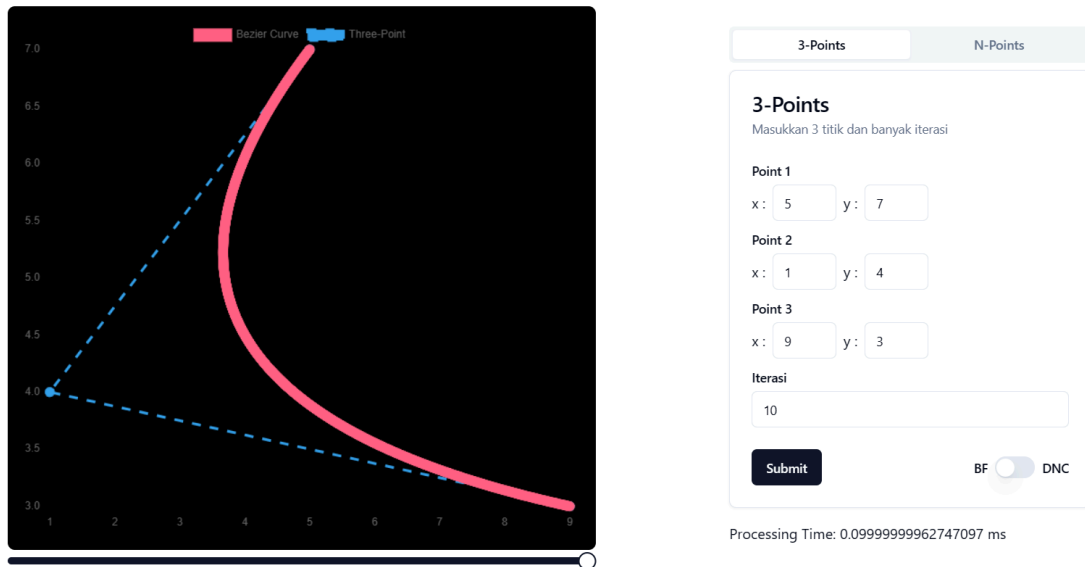
BF ☒ DNC ☐

Processing Time: 0.30000000074505806 ms

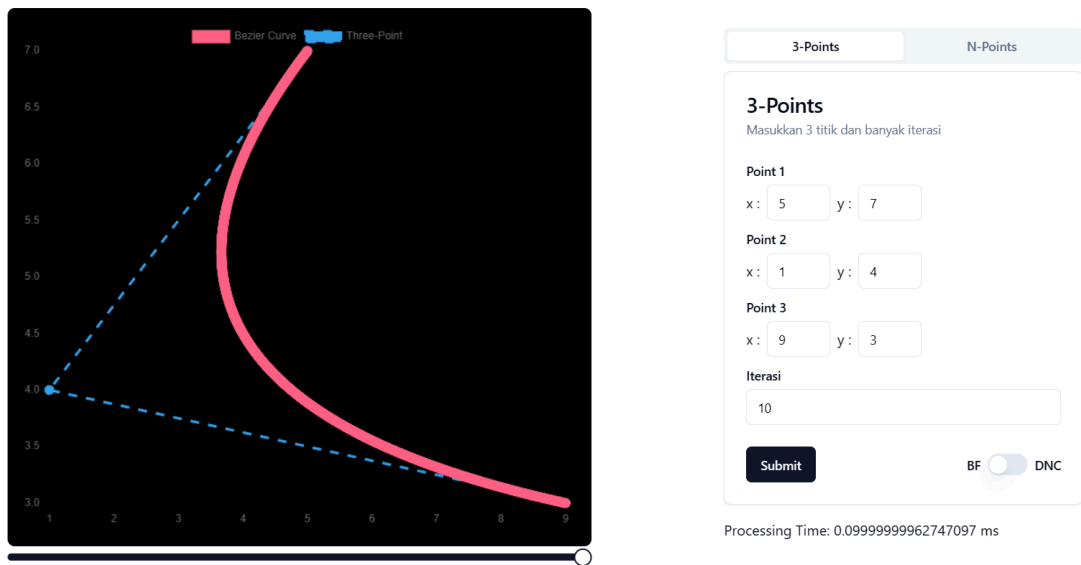
Gambar 4.3.1.2.2 Titik Ketiga ((7, 4), (2, 6), (9, 1)) dengan algoritma divide and conquer

IF2211 Strategi Algoritma - Tugas Kecil 2 | 25

4.3.1.4. Titik Keempat ((5, 7), (1, 4), (9, 3))

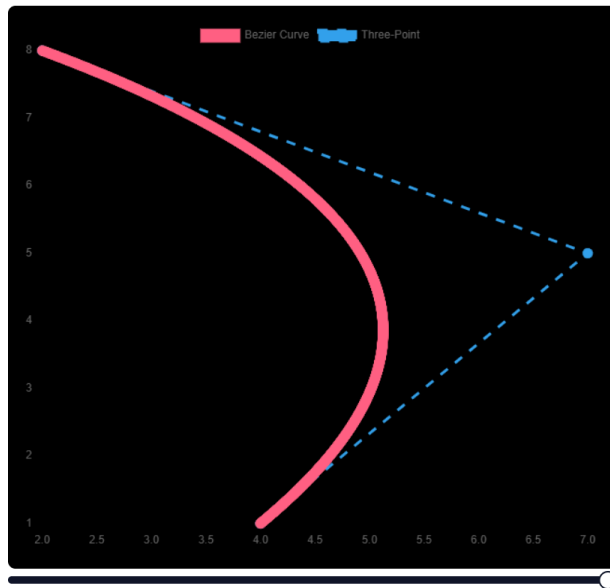


Gambar 4.3.1.4.1 Titik Keempat ((5, 7), (1, 4), (9, 3)) dengan algoritma brute force



Gambar 4.3.1.4.2 Titik Keempat ((5, 7), (1, 4), (9, 3)) dengan algoritma divide and conquer

4.3.1.5. Titik Kelima ((2, 8), (7, 5), (4, 1))



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x: y:

Point 2
x: y:

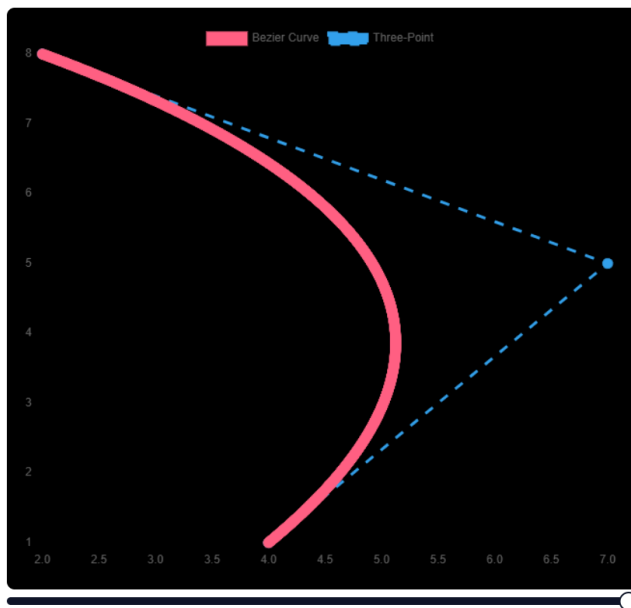
Point 3
x: y:

Iterasi

☒ BF ☐ DNC

Processing Time: 0.900000000372529 ms

Gambar 4.3.1.5.1 Titik Kelima ((2, 8), (7, 5), (4, 1)) dengan algoritma brute force



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x: y:

Point 2
x: y:

Point 3
x: y:

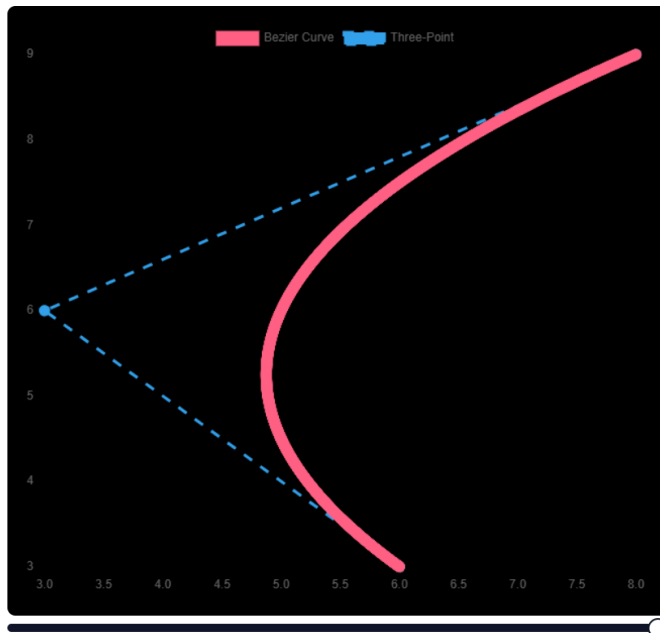
Iterasi

☒ BF ☐ DNC

Processing Time: 0.09999999962747097 ms

Gambar 4.3.1.5.2 Titik Kelima ((2, 8), (7, 5), (4, 1)) dengan algoritma divide and conquer

4.3.1.6. Titik Keenam ((6, 3), (3, 6), (8, 9))



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x : y :

Point 2
x : y :

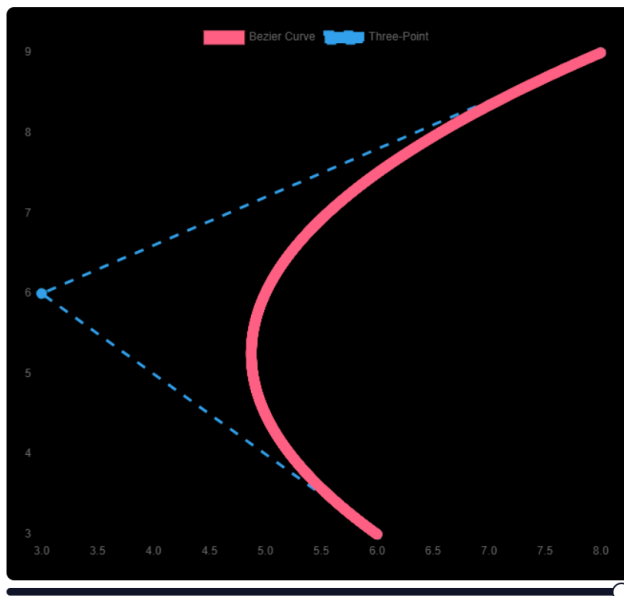
Point 3
x : y :

Iterasi

☐ BF ☒ DNC

Processing Time: 0.19999999925494194 ms

Gambar 4.3.1.6.1 Titik Keenam ((6, 3), (3, 6), (8, 9)) dengan algoritma brute force



3-Points

N-Points

3-Points

Masukkan 3 titik dan banyak iterasi

Point 1
x : y :

Point 2
x : y :

Point 3
x : y :

Iterasi

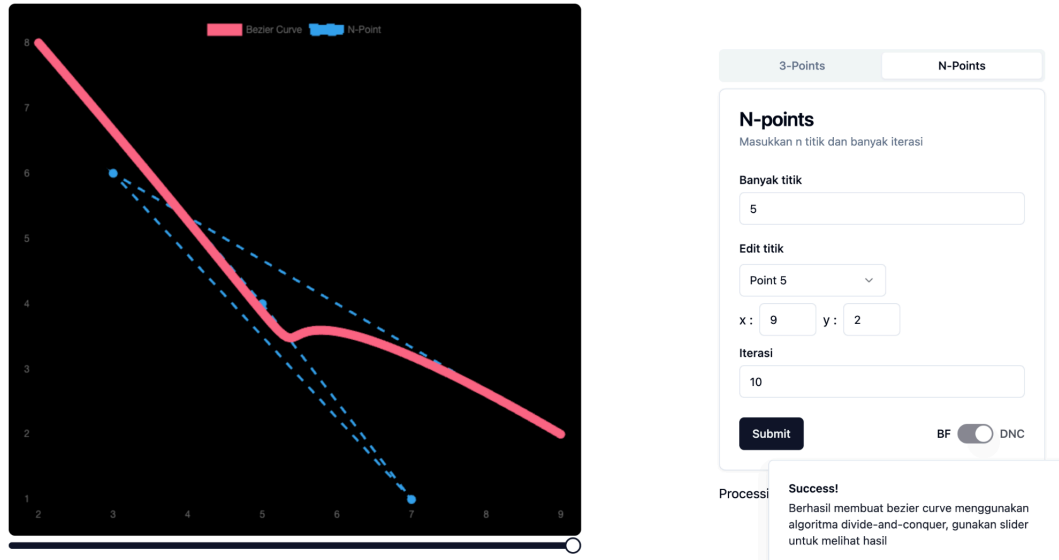
☐ BF ☒ DNC

Processing Time: 0.09999999962747097 ms

Gambar 4.3.1.6.2 Titik Keenam ((6, 3), (3, 6), (8, 9)) dengan algoritma divide and conquer

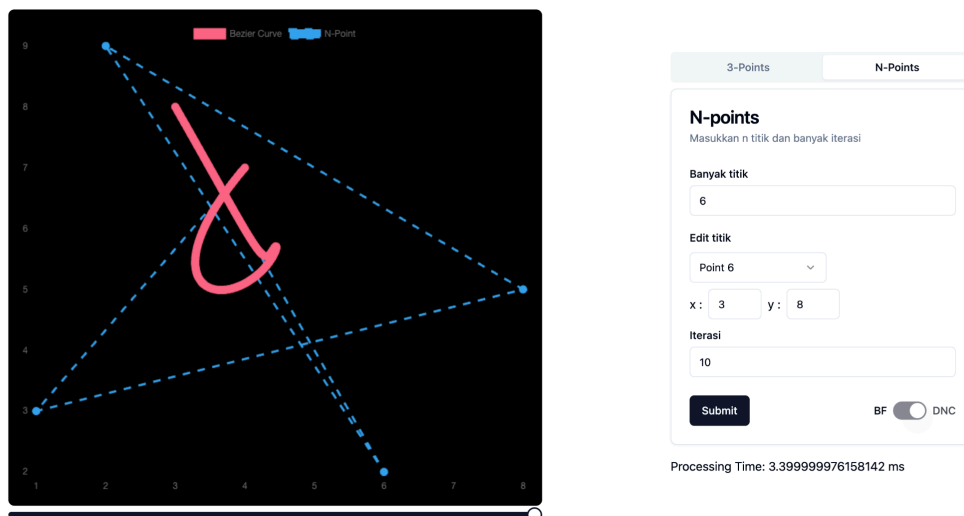
4.3.2. N Points

4.3.2.1. Titik pertama ((2, 8), (5, 4), (7, 1), (3, 6), (9, 2))



Gambar 4.3.2.1.1 Hasil kurva Bezier lima titik pertama dengan algoritma Divide and Conquer

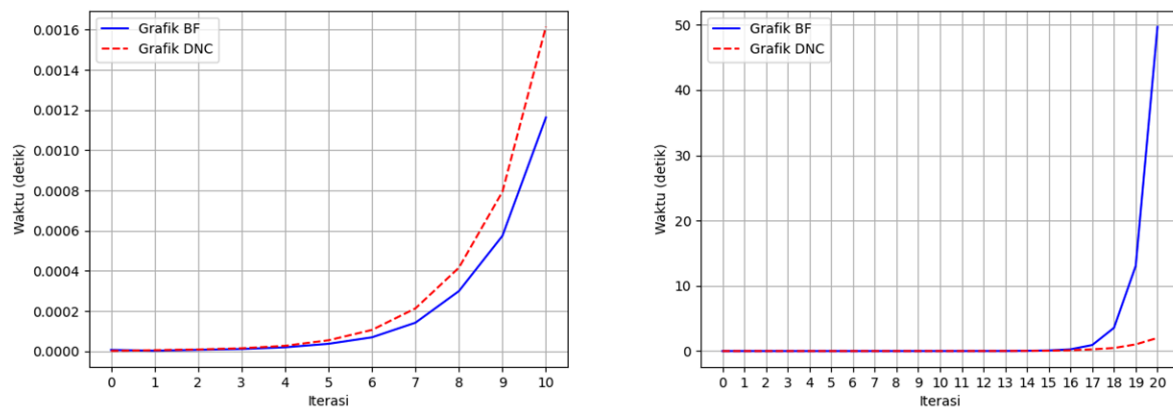
4.3.2.2. Titik kedua ((4, 7), (1, 3), (8, 5), (2, 9), (6, 2), (3, 8))



Gambar 4.3.2.1.2 Hasil kurva Bezier lima titik kedua dengan algoritma Divide and Conquer

4.4. Analisis

Untuk memudahkan analisis, kami membuat plotting grafik waktu yang dibutuhkan antara algoritma Divide and Conquer dengan algoritma Brute Force.



Gambar 4.4.1. Grafik hubungan waktu dengan jumlah iterasi dari kedua algoritma

Dapat dilihat untuk jumlah iterasi kecil, waktu yang dibutuhkan Divide and Conquer cenderung lebih banyak dibandingkan dengan Brute Force dan untuk jumlah iterasi besar, waktu yang dibutuhkan Brute Force jauh melampaui waktu yang dibutuhkan oleh Divide and Conquer.

Hal ini disebabkan karena Divide and Conquer biasanya memerlukan waktu tambahan untuk membagi masalah menjadi submasalah yang lebih kecil, memprosesnya secara terpisah, dan kemudian menggabungkan solusi-solusi tersebut. Untuk masalah dengan ukuran kecil, waktu tambahan ini dapat menjadi signifikan dibandingkan dengan penyelesaian langsung menggunakan pendekatan Brute Force.

Tetapi untuk masalah dengan jumlah iterasi yang besar, Divide and Conquer lebih mangkus dibandingkan dengan Brute Force karena Divide and Conquer membagi masalah tersebut menjadi submasalah yang lebih kecil sehingga kompleksitas waktu dari penyelesaian setiap submasalah dapat menjadi lebih baik daripada menyelesaikan masalah utuh secara langsung.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Melalui penyelesaian tugas ini kami berhasil mengimplementasikan program untuk membuat kurva Bezier dengan algoritma Divide and Conquer dan algoritma Brute Force. Kemudian berdasarkan analisis yang kami lakukan juga, algoritma Divide and Conquer terbukti membutuhkan waktu yang lebih sedikit dibandingkan dengan algoritma Brute Force untuk nilai iterasi yang besar.

Tetapi harus diingat bahwa untuk nilai iterasi yang kecil, algoritma Brute Force masih lebih unggul dibandingkan dengan algoritma Divide and Conquer. Hal ini karena waktu yang dibutuhkan Divide and Conquer untuk membagi masalah menjadi submasalah membutuhkan waktu yang lebih banyak dibandingkan dengan metode penyelesaian secara langsung dengan algoritma Brute Force di nilai iterasi yang kecil.

5.2. Saran

Dalam menyelesaikan tugas ini, terdapat beberapa saran dan peningkatan yang bisa kami terapkan untuk tugas-tugas selanjutnya. Diantaranya adalah:

1. Algoritma brute-force seharusnya bisa dikembangkan lebih lanjut untuk n titik sehingga bisa dilakukan perbandingan kompleksitas dan performa algoritma lebih jauh.
2. Mendalami teori perhitungan terlebih dahulu di awal agar tidak salah mengimplementasi program.
3. Lebih sering untuk buka QnA agar tidak tertinggal informasi.
4. Tidak menunda-nunda pengerjaan Tugas Kecil hanya karena judulnya “Tugas Kecil”.

LAMPIRAN

Tautan repository: https://github.com/Nerggg/Tucil2_13522147_13522122/.

Hasil deployment website: https://maulvi-zm.github.io/Tucil2_13522147_13522122/

DAFTAR PUSTAKA

- Munir, Rinaldi. 2024. Algoritma Divide and Conquer (Bagian 1). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf). Diakses pada 14 Maret 2024.
- Munir, Rinaldi. 2024. Algoritma Divide and Conquer (Bagian 2). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf). Diakses pada 16 Maret 2024.
- Munir, Rinaldi. 2024. Algoritma Divide and Conquer (Bagian 3). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf). Diakses pada 17 Maret 2024.
- Munir, Rinaldi. 2024. Algoritma Divide and Conquer (Bagian 4). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf). Diakses pada 17 Maret 2024.