

LAPORAN TUGAS KECIL 02

IF2211 STRATEGI ALGORITMA

“Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis *Divide and Conquer*”



Disusun oleh:

13522145 - Farrel Natha Saskoro

13522163 - Atqiya Haydar Luqman

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

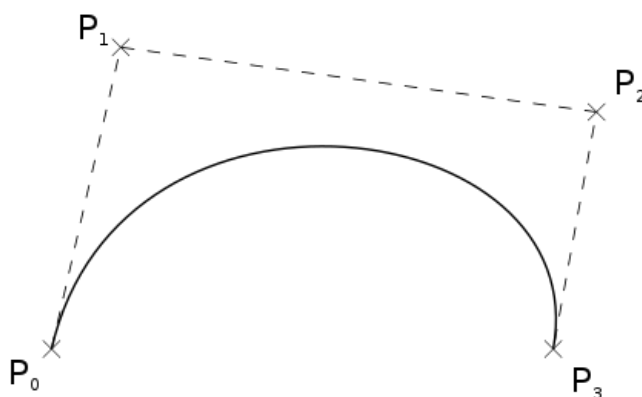
INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

DAFTAR ISI	2
BAB 1	3
BAB 2	6
BAB 3	8
BAB 4	15
BAB 5	17
BAB 6	23

BAB 1

DESKRIPSI MASALAH



Gambar 1.1 Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. 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.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , dengan n disebut order ($n = 1$ untuk linier, $n = 2$ untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P_0 , sedangkan titik kontrol terakhir adalah P_3 . Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan **kurva Bézier linier**. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1 , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1 . Misalnya ketika $t = 0.25$, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1 . sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1 .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2 , dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2 , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadrat** terhadap titik P_0 dan P_2 . Berikut adalah uraian persamaannya.

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

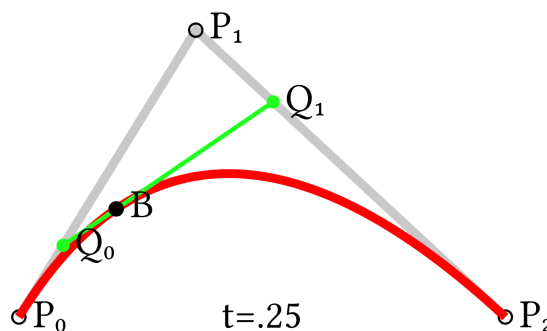
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

dengan melakukan substitusi nilai Q_0 dan Q_1 , maka diperoleh persamaan sebagai berikut.

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

Berikut adalah ilustrasi dari kasus diatas.



Gambar 1.2 Pembentukan Kurva Bézier Kuadrat.

(Sumber: <https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/>)

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan **kurva Bézier kubik**, lima titik akan menghasilkan **kurva Bézier**

kuartik, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3, \quad t \in [0, 1]$$

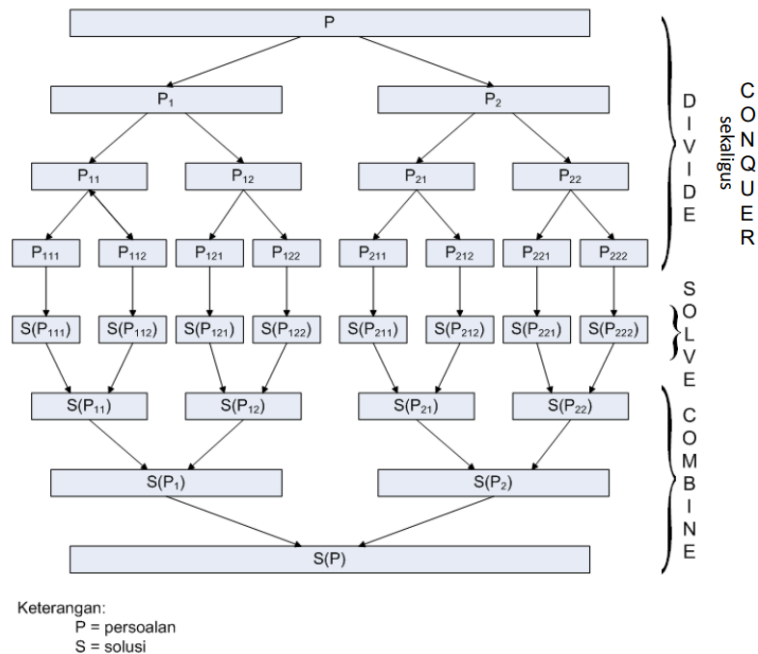
$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \quad t \in [0, 1]$$

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis *divide and conquer*.

BAB 2

TEORI SINGKAT

2.1 Algoritma Divide and Conquer



Gambar 2.1.1 Visualisasi Algoritma Divide and Conquer

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/>)

Algoritma Divide and Conquer adalah strategi dalam pemecahan masalah yang terdiri dari tiga langkah utama. Pertama, langkah Divide membagi masalah utama menjadi beberapa sub-masalah yang lebih kecil namun serupa dengan masalah aslinya. Kedua, langkah Conquer menyelesaikan setiap sub-masalah, baik secara langsung jika sudah cukup kecil atau secara rekursif jika masih berukuran besar. Terakhir, langkah Combine menggabungkan solusi dari setiap sub-masalah sehingga membentuk solusi untuk masalah utama.

Strategi Divide and Conquer sangat berguna dalam menangani masalah kompleks dengan memecahnya menjadi bagian-bagian yang lebih kecil dan lebih mudah dipecahkan. Ini memungkinkan penyelesaian masalah secara efisien dengan memanfaatkan rekursi untuk menyelesaikan sub-masalah.

Keuntungan utama dari algoritma ini adalah kemampuannya untuk menangani masalah besar dan kompleks dengan cara yang efisien, serta kemudahan dalam memanfaatkan

paralelisme untuk meningkatkan kinerja. Namun, strategi ini memerlukan analisis yang cermat dalam pemilihan ukuran sub-masalah agar tidak terlalu kecil atau terlalu besar, yang dapat mempengaruhi kinerja secara keseluruhan.

Algoritma Divide and Conquer digunakan secara luas dalam berbagai bidang, termasuk pengurutan data (seperti algoritma merge sort, quicksort), pencarian data (seperti binary search), penghitungan (seperti perkalian matriks), dan pemodelan algoritma paralel. Pemahaman yang kuat tentang strategi ini penting bagi pengembang perangkat lunak untuk menangani masalah yang kompleks dengan cara yang efisien dan efektif.

2.2 Algoritma Brute Force

Algoritma brute force adalah pendekatan sederhana dalam pemecahan masalah komputasi yang mencoba semua kemungkinan solusi secara berurutan untuk menemukan yang optimal. Pendekatan ini tidak memanfaatkan struktur atau pengetahuan khusus tentang masalah yang sedang dipecahkan.

Prinsip dasar dari algoritma brute force adalah mencoba semua kemungkinan solusi dan memilih yang paling sesuai. Meskipun pendekatan ini seringkali dianggap tidak efisien karena memerlukan waktu yang sangat banyak, terutama untuk masalah dengan ruang pencarian besar, namun algoritma brute force masih sering digunakan dalam beberapa kasus di mana jumlah kemungkinan solusi terbatas atau tidak ada metode yang lebih efisien yang tersedia.

Kelebihan utama dari algoritma brute force adalah kesederhanaannya dan kemampuannya untuk menemukan solusi optimal dalam konteks masalah tertentu. Namun, kelemahannya adalah kinerja yang buruk saat digunakan untuk masalah dengan ruang pencarian yang besar, karena memerlukan waktu komputasi yang sangat lama.

Meskipun demikian, algoritma brute force tetap penting karena dapat menjadi langkah awal yang berguna dalam merancang algoritma yang lebih canggih. Selain itu, dalam beberapa kasus, pendekatan brute force adalah satu-satunya cara yang dapat diandalkan untuk menemukan solusi optimal. Oleh karena itu, pemahaman tentang algoritma brute force dan kapan harus menggunakannya menjadi penting dalam bidang ilmu komputer dan pengembangan perangkat lunak.

BAB 3

ANALISIS & IMPLEMENTASI PROGRAM

3.1 Algoritma Divide and Conquer

Implementasi algoritma Divide and Conquer pada tucil berfokus pada membagi masalah awal menjadi sub masalah yang lebih kecil, menyelesaikannya secara rekursif, dan menggabungkan solusi dari setiap sub-masalah.

```
export const divideAndConquer = (points, left, right, iteration, iterations) => {
  let leftPoints = [points[0]];
  let rightPoints = [points[points.length - 1]];

  while (points.length > 1) {
    let newPoints = [];
    for (let i = 0; i < points.length - 1; i++) {
      newPoints.push(midPoint(points[i], points[i + 1]));
    }

    leftPoints = leftPoints.concat([newPoints[0]]);
    rightPoints = [newPoints[newPoints.length - 1]].concat(rightPoints);
    points = newPoints;
  }

  if (iteration === iterations - 1) {
    return points;
  } else {
    iteration += 1;
    return divideAndConquer(
      leftPoints.concat(points), left, points, iteration, iterations
    ).concat(points).concat(divideAndConquer(points.concat(rightPoints), [], right, iteration, iterations));
  }
};
```

Gambar 3.1.1 Fungsi Divide and Conquer

Fungsi `divideAndConquer` ini mengimplementasikan algoritma divide and conquer untuk menemukan titik tengah dari serangkaian titik dalam ruang dua dimensi. Pada awalnya, fungsi menerima parameter `points` yang merupakan array titik-titik, `left` dan `right` yang menunjukkan batas kiri dan kanan dari rentang titik, `iteration` yang merepresentasikan iterasi saat ini, dan `iterations` yang merupakan total iterasi yang diinginkan. Algoritma ini secara rekursif membagi rentang titik menjadi dua bagian, yaitu `leftPoints` dan `rightPoints`, dan menghitung titik tengahnya dengan menggunakan fungsi `midPoint`. Iterasi dilakukan dengan mengulangi proses ini hingga hanya tersisa satu titik dalam array `points`.

Setelah iterasi selesai, jika iterasi saat ini sudah mencapai iterasi terakhir (`iteration === iterations - 1`), maka fungsi mengembalikan titik-titik yang tersisa. Namun, jika belum mencapai iterasi terakhir, maka iterasi ditingkatkan, dan fungsi `divideAndConquer` dipanggil lagi untuk membagi dan menaklukkan bagian kiri dan kanan, kemudian hasilnya digabungkan dengan titik-titik tengah yang telah dihitung sebelumnya, dan proses ini terus berlanjut hingga mencapai

iterasi terakhir. Akhirnya, hasil dari kedua bagian ini digabungkan dan dikembalikan sebagai array titik-titik tengah dari seluruh rentang.

Pada bagian front end, fungsi `handleSubmitDnC()` menginisiasi proses pencarian dengan algoritma Divide and Conquer ketika tombol “Submit with DnC” ditekan. Hasil dari algoritma tersebut digunakan untuk menentukan koordinat yang akan ditampilkan sebagai hasil, sementara waktu eksekusi juga diukur untuk analisis kinerja.

```
const handleSubmitDnC = () => {
  setMethod('DnC')
  const start = performance.now()

  const totalIteration = parseInt(iter)
  let coordinatePoints = []
  coordinatePoints.push(useCoordinates[0])
  let NewPointPoints = (divideAndConquer(useCoordinates, [], [], 0, totalIteration))
  coordinatePoints = [...coordinatePoints, ...NewPointPoints]
  coordinatePoints.push(useCoordinates[useCoordinates.length - 1])
  const end = performance.now()

  setResultCoordinates(coordinatePoints)
  setTimeExecution((end - start).toFixed(3))
}
```

Gambar 3.1.2 Fungsi Handle Submit Algoritma Divide and Conquer

Dengan pendekatan ini, algoritma Divide and Conquer dapat menghasilkan solusi yang optimal dalam menyelesaikan masalah dengan membaginya menjadi sub-masalah yang lebih kecil dan menyelesaikannya secara efisien.

3.2 Algoritma Brute Force

Algoritma brute force digunakan di dalam tucil kali ini sebagai algoritma pembanding dengan algoritma divide and conquer.

```
// Functions to calculate Bezier curve using brute force with n-points
function BezierFormula(points, n, t) {
  const pascal = pascalTriangle(n);
  let x = 0;
  let y = 0;
  for (let i = 0; i < n; i++) {
    x += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].x;
    y += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].y;
  }
  return { x, y };
}
```

Gambar 3.2.1 Fungsi BezierFormula

Dalam fungsi BezierFormula terdapat tiga argumen yang terdiri dari points adalah array of point control, n adalah banyaknya titik kontrol, dan t adalah banyaknya iterasi. pertama akan digenerate segitiga pascal sesuai dengan input jumlah titik kontrol dan akan diambil layer yang paling bawahnya (contoh, jika $n = 3$, maka pascal = [1,2,1], setelah itu akan dicari titik yang membentuk kurva Bézier dengan memasukkan pascal, points, n, dan t kedalam rumus tersebut, terakhir fungsi akan mengembalikan titik pembuat kurva Bézier.

Proses tersebut akan terus diiterasi sebanyak $2^{\text{iterasi}} - 1$ di dalam fungsi BezierGeneratorBruteForce. Tiap melakukan iterasi, titik pembuat kurva Béziernya akan disimpan kedalam array dan setelah iterasi selesai dilakukan akan di kembalikan oleh fungsi.

```
export function BezierGeneratorBruteForce(points, n, iterations) {
  let result = [];

  let total = Math.pow(2, iterations) - 1;

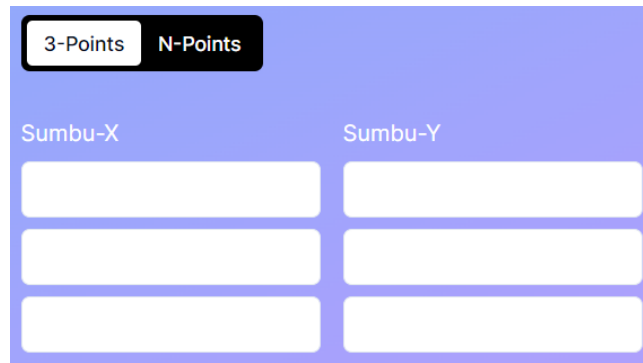
  for (let i = 0; i <= total; i++) {
    const t = i / total;
    const point = BezierFormula(points, n, t);
    result.push(point);
  }

  return result;
}
```

Gambar 3.2.2 Fungsi Bezier Generator Brute Force

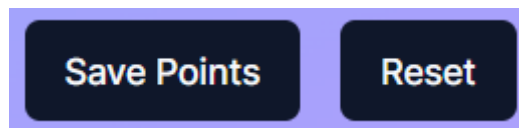
3.3 Implementasi Visualisasi Hasil

Pengguna diminta untuk memasukkan koordinat titik-titik yang akan digunakan sebagai data input untuk menghasilkan kurva Bezier. Terdapat dua mode input yang tersedia, yaitu 3-Points dan N-Points, dimana N-Points memungkinkan pengguna untuk memasukkan jumlah titik secara dinamis. Setiap titik yang dimasukkan memiliki koordinat sumbu-x dan sumbu-y yang dapat dimasukkan oleh pengguna melalui antarmuka pengguna.



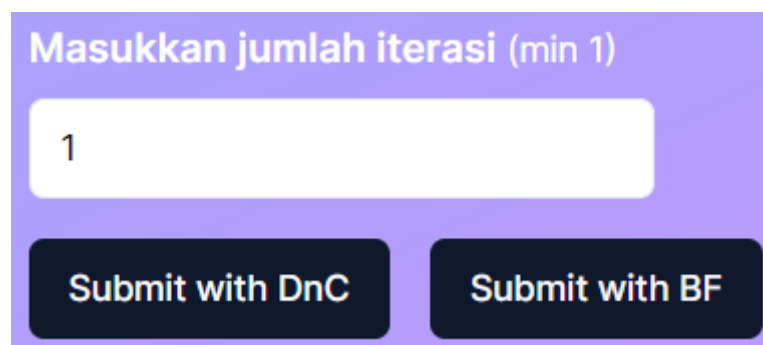
Gambar 3.3.1 Input Titik

Setelah titik-titik dimasukkan oleh pengguna, mereka memiliki opsi untuk menyimpan titik-titik tersebut. Titik-titik yang disimpan akan digunakan sebagai input untuk algoritma pencarian yang dipilih.



Gambar 3.3.2 Save Input Button

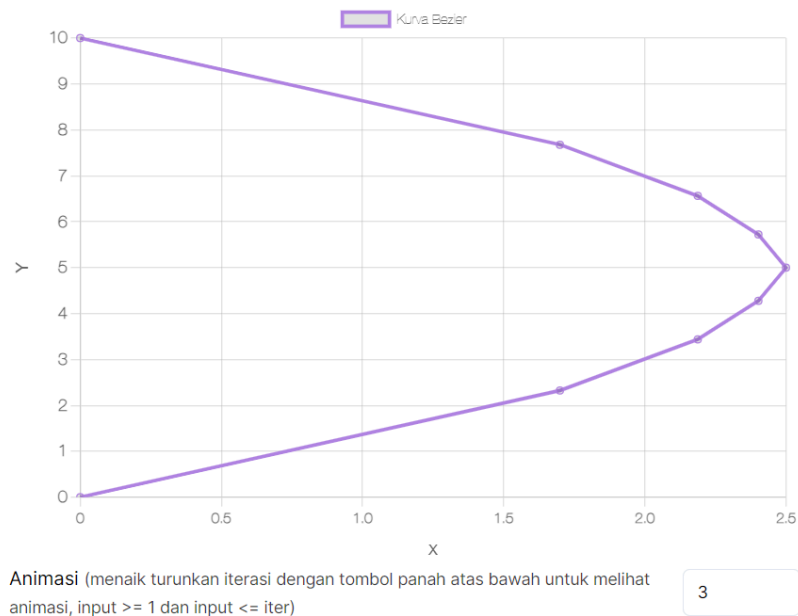
Pengguna dapat memilih jumlah iterasi yang diinginkan dan melakukan submit dengan menggunakan algoritma Divide and Conquer (DnC) atau Brute Force (BF). Saat pengguna melakukan submit, algoritma yang dipilih akan dijalankan untuk menghasilkan titik-titik yang akan digunakan dalam visualisasi grafis.



Gambar 3.3.3 Input Iterasi dan Submit Button

Hasil dari algoritma yang dipilih akan divisualisasikan menggunakan grafik garis. Kurva Bezier akan ditampilkan berdasarkan titik-titik yang dihasilkan oleh algoritma yang dipilih. Pengguna dapat melihat animasi perubahan kurva Bezier dengan mengatur jumlah iterasi melalui

antarmuka pengguna. Pengguna dapat mengganti nomor iterasi yang berada pada input dibawah kurva untuk melihat bagaimana proses pembentukan kurva dari iterasi pertama.



Gambar 3.3.4 Contoh Hasil Grafik

Setelah proses submit selesai dilakukan, waktu eksekusi dari algoritma yang dipilih akan ditampilkan kepada pengguna. Ini memberikan informasi kepada pengguna tentang kinerja relatif dari algoritma yang digunakan.

Time Execution : 0.100 ms

Gambar 3.3.5 Waktu Eksekusi

3.4 Generalisasi Algoritma

Pada program kami sudah dilakukan generalisasi algoritma sehingga bisa membuat kurva Bézier kubik, kuartik, dan selanjutnya dengan 4, 5, 6, hingga n titik kontrol. Pada Algoritma Brute Force, generalisasi dilakukan dengan menggeneralisasi rumus yang digunakan karena rumus memiliki pola yang sama dengan konstanta pada rumus berpola segitiga pascal.

```
function BezierFormula(points, n, t) {
  const pascal = pascalTriangle(n);
  let x = 0;
  let y = 0;
  for (let i = 0; i < n; i++) {
    x += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].x;
    y += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].y;
  }
  return { x, y };
}
```

Gambar 3.4.1 Generalisasi Algoritma Brute Force

Pada Algoritma Divide and Conquer, generalisasi terdapat pada penggunaan fungsi `midPoint` yang menghitung titik tengah antara dua titik kontrol, rekursi dalam fungsi `divideAndConquer` yang memungkinkan pembagian titik kontrol dalam jumlah apa pun, dan parameter `iterations` yang mengontrol seberapa banyak iterasi rekursi yang dilakukan untuk pembagian titik kontrol. Dengan demikian, program ini fleksibel dalam menciptakan kurva bezier dengan jumlah titik kontrol dari kubik hingga n .

```
export const divideAndConquer = (points, left, right, iteration, iterations) => {
  let leftPoints = [points[0]];
  let rightPoints = [points[points.length - 1]];

  while (points.length > 1) {
    let newPoints = [];
    for (let i = 0; i < points.length - 1; i++) {
      newPoints.push(midPoint(points[i], points[i + 1]));
    }

    leftPoints = leftPoints.concat([newPoints[0]]);
    rightPoints = [newPoints[newPoints.length - 1]].concat(rightPoints);
    points = newPoints;
  }

  if (iteration === iterations - 1) {
    return points;
  } else {
    iteration += 1;
    return divideAndConquer(
      leftPoints.concat(points), left, points, iteration, iterations
    ).concat(points).concat(divideAndConquer(points.concat(rightPoints), [], right, iteration, iterations));
  }
};
```

Gambar 3.4.2 Generalisasi Algoritma Divide and Conquer

3.5 Analisis Perbandingan Algoritma

Solusi brute force dalam kode yang diberikan melakukan perhitungan kurva Bezier dengan memeriksa setiap titik di dalam ruang parameter. Dengan melakukan iterasi sebanyak yang ditentukan oleh pengguna, algoritma brute force menghitung posisi titik kurva pada setiap langkahnya. Kompleksitas waktu dari algoritma ini bergantung pada jumlah iterasi yang ditentukan, yang secara eksponensial meningkat sesuai dengan jumlah iterasi. Dalam kode yang disediakan, kompleksitas waktu dari algoritma brute force untuk menghitung kurva bezier dengan tiga titik atau jumlah titik variabel adalah

$$T(n) = O(n * 2^n)$$

Sementara itu, solusi divide and conquer dalam kode mengadopsi strategi membagi masalah menjadi sub-masalah yang lebih kecil, menyelesaikannya secara rekursif, dan menggabungkan solusi dari setiap sub-masalah. Kompleksitas waktu algoritma divide and conquer tergantung pada jumlah titik dan iterasi yang ditentukan oleh pengguna. Namun, kompleksitas waktu pada umumnya lebih baik daripada brute force karena hanya membagi masalah menjadi dua bagian setiap iterasinya. Dalam kode yang diberikan, kompleksitas waktu algoritma divide and conquer diperkirakan sebagai

$$T(n) = O(n \log n)$$

Dari analisis di atas, dapat disimpulkan bahwa solusi divide and conquer cenderung memiliki kompleksitas waktu yang lebih baik daripada solusi brute force untuk perhitungan kurva Bezier, terutama ketika jumlah iterasi relatif kecil dan jumlah titik tidak terlalu besar. Namun, untuk kasus di mana jumlah titik dan iterasi cukup besar, solusi divide and conquer masih mungkin mengalami kompleksitas waktu yang tinggi, meskipun secara umum masih lebih efisien daripada brute force.

BAB 4

SOURCE CODE

4.1 Algoritma Divide and Conquer

```
1  const midPoint = (p0, p1) => ({
2    x: (parseFloat(p0.x) + parseFloat(p1.x)) / 2,
3    y: (parseFloat(p0.y) + parseFloat(p1.y)) / 2
4  });
5
6  export const divideAndConquer = (points, left, right, iteration, iterations) => {
7    let leftPoints = [points[0]];
8    let rightPoints = [points[points.length - 1]];
9
10   while (points.length > 1) {
11     let newPoints = [];
12     for (let i = 0; i < points.length - 1; i++) {
13       newPoints.push(midPoint(points[i], points[i + 1]));
14     }
15
16     leftPoints = leftPoints.concat([newPoints[0]]);
17     rightPoints = [newPoints[newPoints.length - 1]].concat(rightPoints);
18     points = newPoints;
19   }
20
21   if (iteration === iterations - 1) {
22     return points;
23   } else {
24     iteration += 1;
25     return divideAndConquer(
26       leftPoints.concat(points), left, points, iteration, iterations
27     ).concat(points).concat(divideAndConquer(points.concat(rightPoints), [], right, iteration, iterations));
28   }
29   };
```

Gambar 4.1. Source code algoritma divide and conquer

4.2 Algoritma Brute Force

```
1
2 // function to generate pascal triangle
3 function pascalTriangle(n) {
4   let result = [];
5   for (let i = 0; i < n; i++) {
6     result[i] = new Array(i + 1);
7     result[i][0] = 1;
8     result[i][i] = 1;
9     for (let j = 1; j < i; j++) {
10      result[i][j] = result[i - 1][j - 1] + result[i - 1][j];
11    }
12  }
13  return result[n - 1];
14 }
15
16 // Functions to calculate Bezier curve using brute force with n-points
17 function BezierFormula(points, n, t) {
18   const pascal = pascalTriangle(n);
19   let x = 0;
20   let y = 0;
21   for (let i = 0; i < n; i++) {
22     x += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].x;
23     y += Math.pow(1 - t, n - 1 - i) * Math.pow(t, i) * pascal[i] * points[i].y;
24   }
25   return { x, y };
26 }
27
28 export function BezierGeneratorBruteForce(points, n, iterations) {
29   let result = [];
30
31   let total = Math.pow(2, iterations) - 1;
32
33   for (let i = 0; i <= total; i++) {
34     const t = i / total;
35     const point = BezierFormula(points, n, t);
36     result.push(point);
37   }
38
39   return result;
40 }
41
42
43 // Functions to calculate Bezier curve using brute force with only 3-points
44 function quadraticBezierFormula(points, t) {
45   const x = Math.pow(1 - t, 2) * points[0].x + 2 * (1 - t) * t * points[1].x + Math.pow(t, 2) * points[2].x;
46   const y = Math.pow(1 - t, 2) * points[0].y + 2 * (1 - t) * t * points[1].y + Math.pow(t, 2) * points[2].y;
47   return { x, y };
48 }
49
50 export function quadraticBezierGeneratorBruteForce(points, iterations) {
51   let result = [];
52
53   let total = Math.pow(2, iterations) - 1;
54
55   for (let i = 0; i <= total; i++) {
56     const t = i / total;
57     const point = quadraticBezierFormula(points, t);
58     result.push(point);
59   }
60
61   return result;
62 }
63
64
```

Gambar 4.2. Source code algoritma brute force

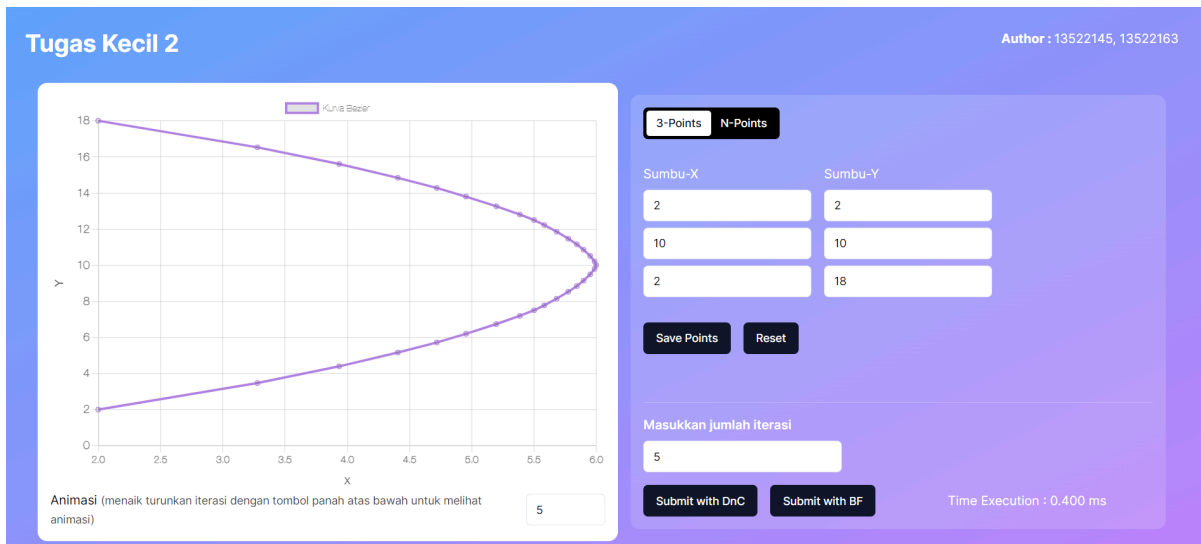
BAB 5

EKSPERIMEN

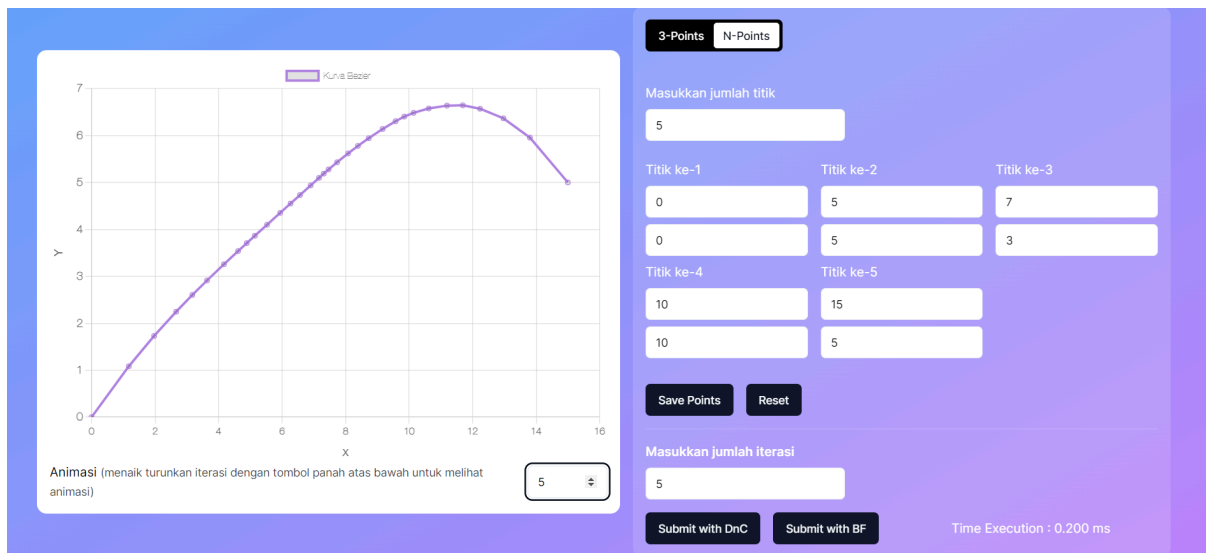
5.1 Algoritma Divide and Conquer



Gambar 5.1.1. Test case 1



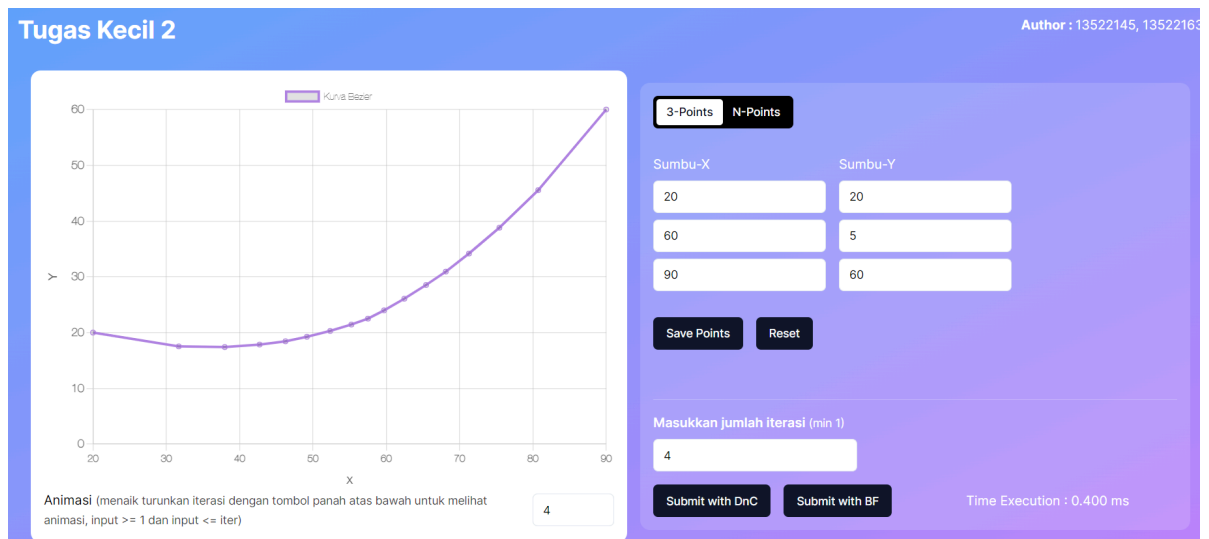
Gambar 5.1.2. Test case 2



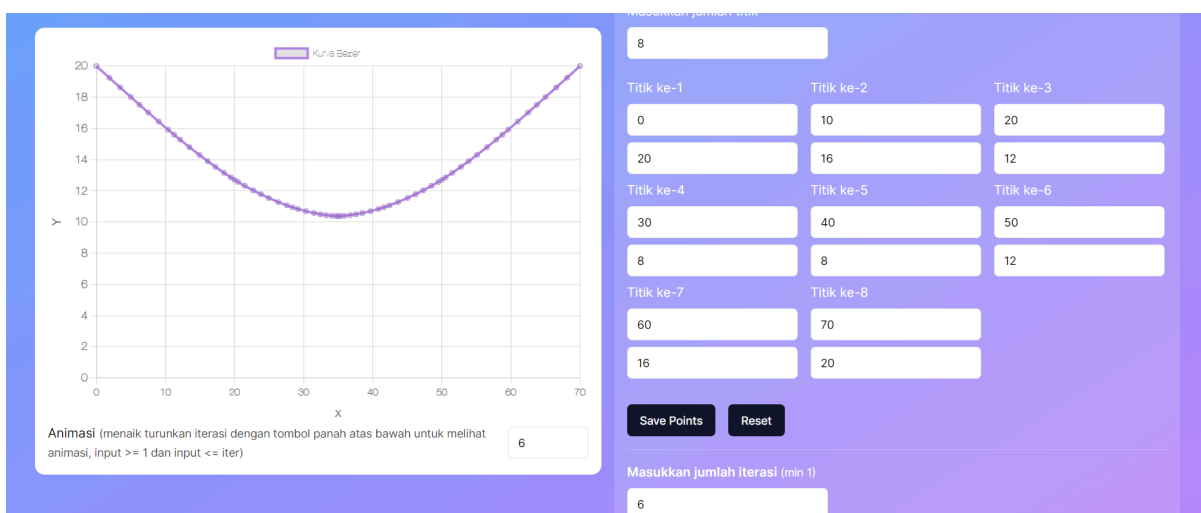
Gambar 5.1.3. Test case 3



Gambar 5.1.4. Test case 4

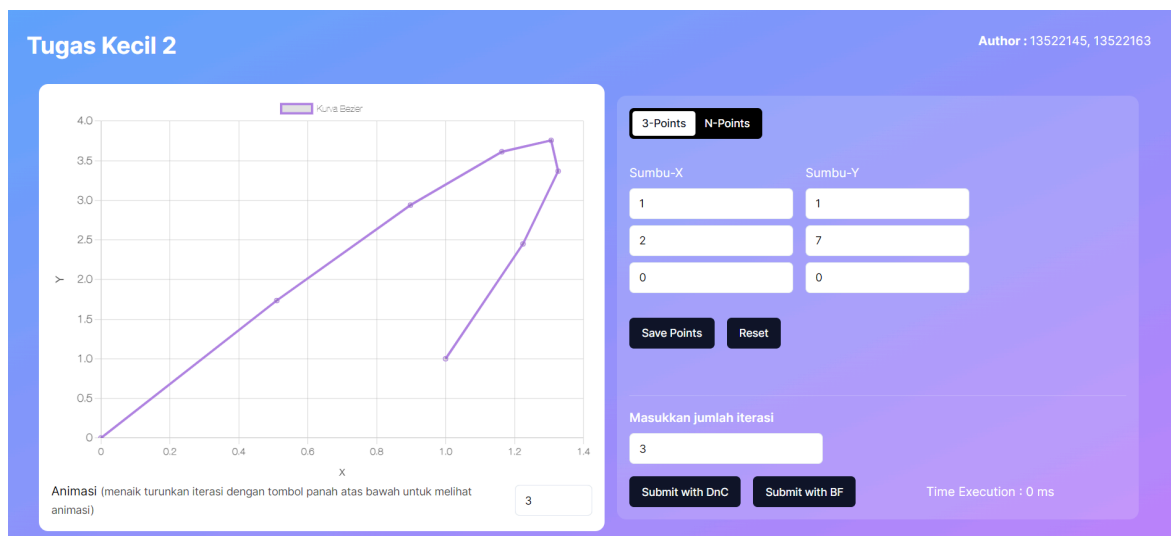


Gambar 5.1.5 Test case 5

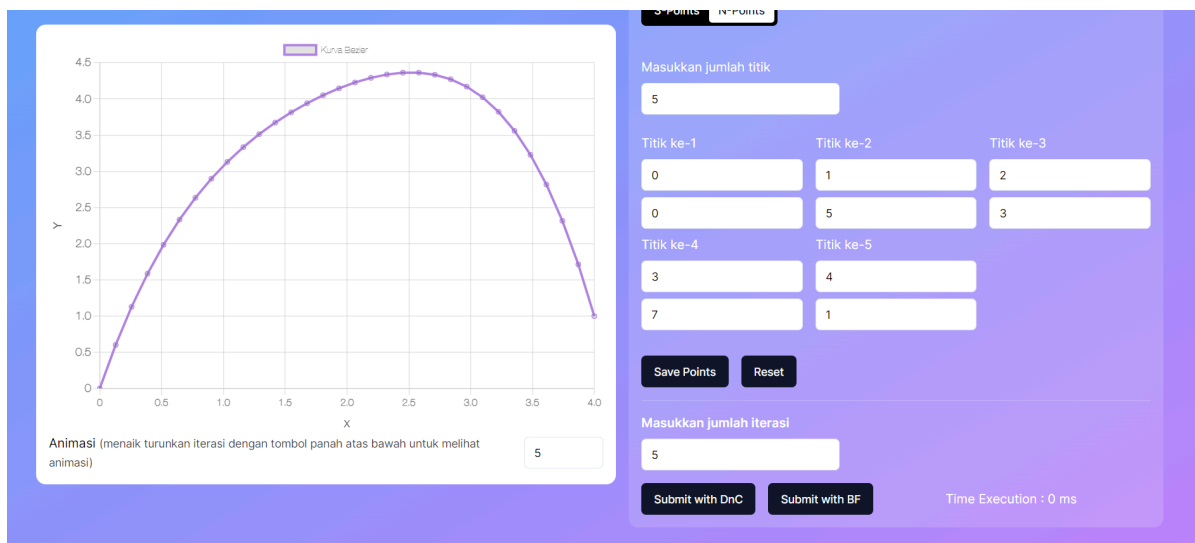


Gambar 5.1.6 Test case 6

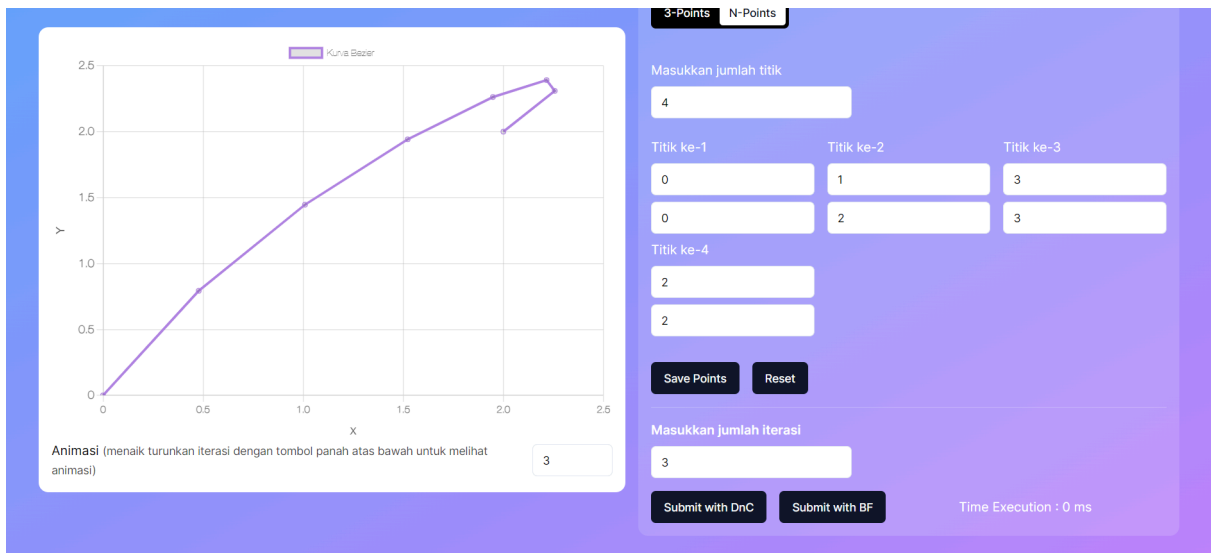
5.2 Algoritma Brute Force



Gambar 5.2.1 Test case 1



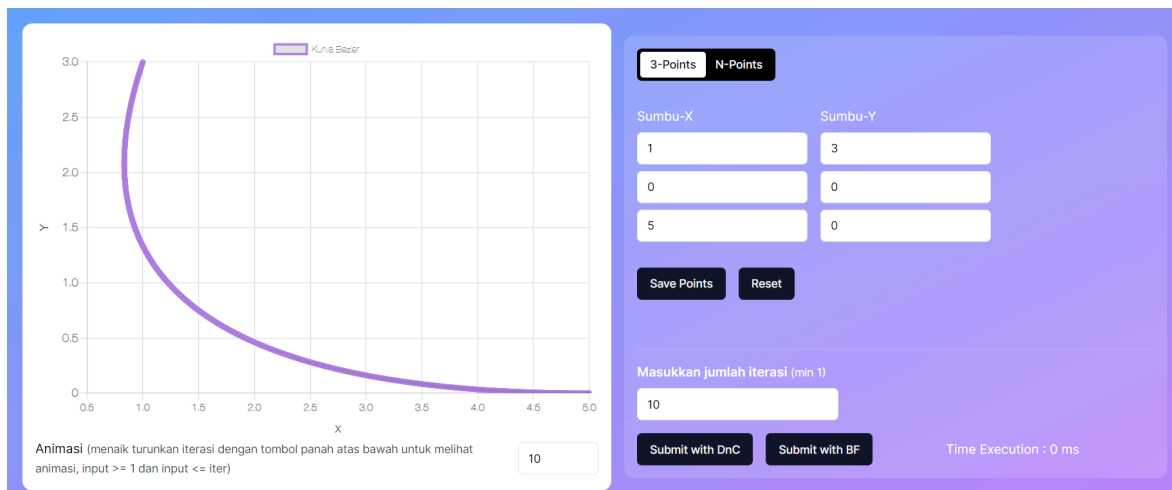
Gambar 5.2.2. Test case 2



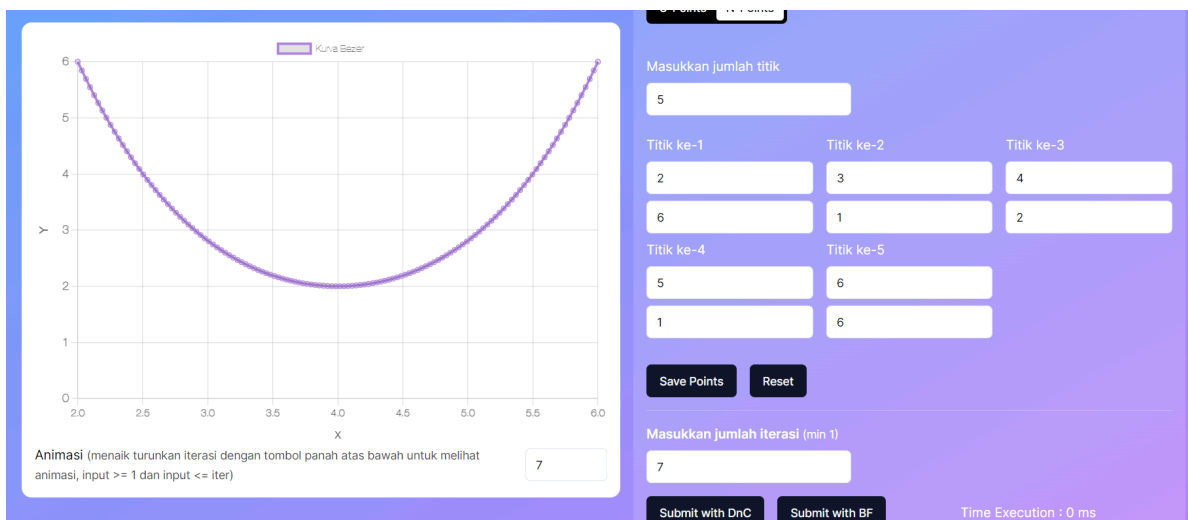
Gambar 5.2.3. Test case 3



Gambar 5.2.4 Test case 4



Gambar 5.2.5 Test case 5



Gambar 5.2.6 Test case 6

BAB 6

PENUTUP

6.1 Kesimpulan

Dari hasil analisis, Algoritma Divide and Conquer memberikan solusi yang lebih optimal dibandingkan dengan Algoritma Brute Force berdasarkan kompleksitas algoritmanya. Akan tetapi, dalam hasil eksperimen didapat Algoritma Brute Force lebih optimal saat jumlah iterasinya masih kecil, namun saat jumlah iterasinya mulai membesar, Algoritma Divide and Conquer didapat lebih Optimal.

6.2 Link Repository

Repository kami dapat diakses melalui pranala berikut:

https://github.com/AtqiyaHaydar/Tucil2_13522145_13522163

Link Website:

<https://tucil2-13522145-13522163-t4uo.vercel.app/>

6.3 Tabel Checkpoint Program

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. [Bonus] Program dapat membuat kurva untuk n titik kontrol.	✓	
5. [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	✓	