

LAPORAN TUGAS KECIL
STRATEGI ALGORITMA
IF 2211

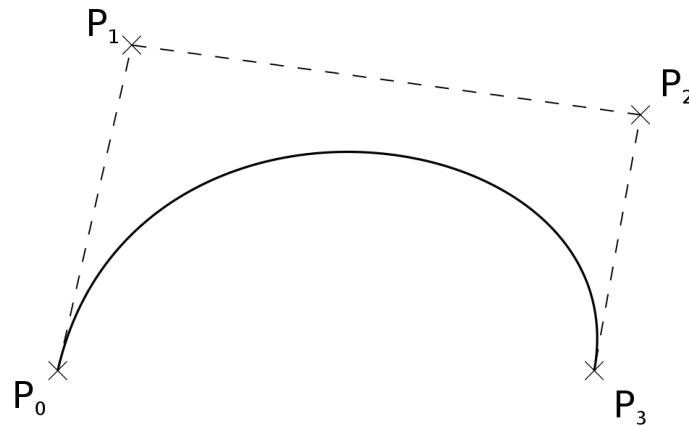


Disusun oleh:
Dhafin Fawwaz Ikramullah
13522084

Daftar Is

BAB I Deskripsi Masalah.....	3
BAB II Landasan Teori.....	6
BAB III Pembahasan.....	7
A. Analisis dan Implementasi Algoritma Brute Force.....	7
B. Analisis dan Implementasi Algoritma Brute Force dengan Turunan Rumus.....	10
C. Analisis dan Implementasi Algoritma Divide and Conquer Khusus Kuadratik.....	13
D. Analisis dan Implementasi Algoritma Divide and Conquer.....	15
E. Hasil Analisis Algoritma.....	19
BAB IV Hasil Pengujian.....	20
A. Pengujian Waktu Eksekusi.....	20
D. Pengujian Fitur Tambahan.....	30
BAB V Kesimpulan.....	33
BAB VI Lampiran.....	34

BAB I Deskripsi Masalah



Gambar 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.

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

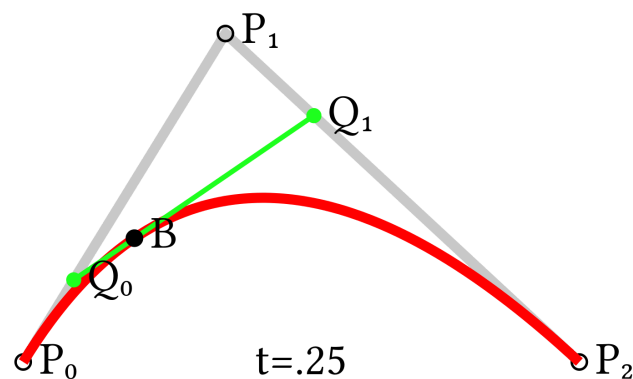
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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 2. Pembentukan Kurva Bézier Kuadratik.

(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.

$$S_0 = B(t) = (1 - t)^3 P_0 + 3(1 - t)^2 t P_1 + 3(1 - t) t^2 P_2 + t^3 P_3, \quad t \in [0, 1]$$

$$T_0 = B(t) = (1 - t)^4 P_0 + 4(1 - t)^3 t P_1 + 6(1 - t)^2 t^2 P_2 + 4(1 - t) t^3 P_3 + t^4 P_4, \quad t \in [0, 1]$$

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 II Landasan Teori

Algoritma Divide and Conquer merupakan sebuah algoritma dalam menyelesaikan sebuah persoalan dengan pemrograman dengan cara membagi persoalan menjadi beberapa upa persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. Lalu dicari solusi dari setiap upa persoalan tersebut secara rekursif hingga menjadi cukup kecil untuk diselesaikan. Lalu solusi dari masing masing upa persoalan tersebut digabung untuk membentuk solusi persoalan semula. Algoritma ini sering digunakan untuk pemecahan persoalan untuk masukan berupa tabel, matriks, eksponen, polinom dan lain-lain. Contoh persoalan yang dapat diselesaikan dengan algoritma ini misalnya persoalan mencari nilai minimum/maksimum, perpangkatan, pengurutan, pencarian pasangan titik terdekat, convex hull, perkalian matrix, dan sebagainya.

Algoritma Brute Force merupakan sebuah algoritma dalam menyelesaikan sebuah persoalan dengan pemrograman dengan cara menguji setiap kemungkinan yang ada secara sistematis untuk mencari solusi yang diinginkan. Umumnya pendekatan ini merupakan pendekatan yang tidak efektif karena membutuhkan waktu dan memory yang cukup besar dalam algoritmanya. Sesuai dengan cara kerja algoritmanya, algoritma Brute Force akan lebih sering lebih lambat dibanding algoritma Divide and Conquer.

BAB III Pembahasan

Terdapat 2 jenis algoritma yang akan dibahas yaitu algoritma Brute Force dan Divide and Conquer. Pada laporan ini sebenarnya hanya diperlukan satu algoritma brute force yaitu yang dengan dengan turunan rumus. Tambahan berikut merupakan inisiatif untuk membandingkan hasil algoritma dengan pendekatan yang sedikit berbeda. Untuk implementasi dari kurva bezier akan dilakukan dengan bahasa pemrograman Javascript, dengan framework dan library yaitu Vite, Pixi.js, Node.js, dan Tailwind. Perlu diingat bahwa akan implementasi code sebenarnya akan terlihat memiliki sedikit perbedaan untuk penyesuaian agar dapat menghasilkan performa seefisien mungkin. Hal ini terjadi karena behaviour dan abstraksi dari tiap bahasa pemrograman yang berbeda-beda. Ada pula tipe data tersendiri seperti CenterPoint, LerpPoint, dan FormulatedPoint yang masing-masing merupakan hasil inheritance dengan SyncablePoint. Hal ini dilakukan untuk kemudahan implementasi fitur Dynamic Update yang akan dibahas pada bab pengujian.

A. Analisis dan Implementasi Algoritma Brute Force

- a. Pertama rumus $B(t) = (1 - t)P_0 + tP_1$ disederhanakan menjadi $B(t) = P_0 + (P_1 - P_0)t$ sehingga menjadi rumus interpolasi linear pada umumnya dan memiliki jumlah operasi yang lebih sedikit. Maka algoritmanya akan sebagai berikut. Misal p adalah List of Point dengan panjang n, dan iterations adalah integer

Prosedur GenerateWithBruteForce(P, iterations)

1. Misal array hasil adalah R.
2. Lakukan loop sebanyak iterations, tiap loop adalah loop ke k
3. Misal $t = (k+1)/(iterations+1)$
4. Ambil titik dengan posisi B(t) untuk setiap elemen P_i dan P_{i+1} , masukkan ke sebuah array Q.
5. Misal a adalah banyak titik yang dimasukkan barisan (yaitu panjang P - 1), b adalah panjang Q saat ini (bernilai sama dengan a untuk saat ini).
6. Ambil titik dengan posisi B(t, Q_i , Q_{i+1}) untuk setiap elemen Q mulai dari elemen ke b-a sampai akhir (yaitu semua titik yang baru dimasukkan pada langkah sebelumnya), masukkan ke array Q. Dapat diartikan dengan mengambil tiap titik dengan interpolasi linear dari hasil setiap interpolasi linear sebelumnya.

7. a diubah nilainya jadi banyak titik yang dimasukkan barusan. b dijadikan panjang Q sekarang.
8. Ulangi langkah 6 dan 7 hingga a menjadi 1 (titik yang dimasukkan tinggal 1).
9. Maka hasil untuk loop ini adalah elemen terakhir Q. Masukkan elemen tersebut ke R, dan lanjut ke loop selanjutnya hingga selesai.
10. Setelah loop selesai, maka R adalah hasilnya.

b. Source Code

Class LerpPoint
<pre> export default class LerpPoint extends SyncablePoint { /** * @param {SyncablePoint} point1 * @param {SyncablePoint} point2 * @param {number} progress */ constructor(point1, point2, progress) { super(point1, point2); this.progress = progress; this.sync(); } /** * Update the position of this point to be in the center of both points. */ sync() { // override this.x = lerp(this.point1.x, this.point2.x, this.progress); this.y = lerp(this.point1.y, this.point2.y, this.progress); } } </pre>
Class Bezier Curve
<pre> ... /** * Generate a quadratic bezier curve based on the points and number of </pre>


```

iterations
  * @param {SyncablePoint[]} p List of input points.
  * @param {number} iterations number of iterations. The bigger, the
smoother.
  */
  generateByBruteForce(p, iterations) {
    const length = p.length;
    for(let k = 0; k < iterations; k++) {
      /** @type {LerpPoint[]} */
      const lerpPointsList = [];
      const progress = (k+1)/(iterations+1);

      // sub center points
      for(let i = 0; i < length-1; i++) {
        const centerPoint = new LerpPoint(p[i], p[i+1], progress);
        lerpPointsList.push(centerPoint);
      }

      // sub sub sub... center points
      let count = 0;
      for(let i = 2; i < length; i++) {
        const lengthMinI = length - i;
        for(let j = 0; j < lengthMinI; j++) {
          const countPlusJ = count + j;
          const centerPoint = new LerpPoint(lerpPointsList[countPlusJ],
lerpPointsList[countPlusJ+1], progress);
          lerpPointsList.push(centerPoint);
        }
        count += lengthMinI + 1;
      }

      this.syncablePointResult.push(lerpPointsList[lerpPointsList.length-1]);
    }
  }
  ...

```

c. Analisis Kompleksitas

Kompleksitas dihitung berdasarkan banyak masukan ke array yang terjadi.

Misal n = panjang array input, i = banyak iterasi.

Banyak iterasi (untuk menyesuaikan banyak titik hasil algoritma Divide and Conquer, maka jumlah iterasi disesuaikan): 2^i

Pembangkitan array $Q = (n - 1) + (n - 2) + (n - 3) + \dots + 1 = n(n - 1)/2$

Pemasukan array solusi = 1

Maka

$$T(n) = (2^i + 1)(n(n - 1)/2 + 1)$$

$$T(n) = (2^i + 1)(0.5n^2 - 0.5n + 1)$$

$$= O(2^i n^2)$$

B. Analisis dan Implementasi Algoritma Brute Force dengan Turunan Rumus

a. Algoritma

Pada algoritma ini, rumus $B(t)$ akan diturunkan terlebih dahulu sehingga menjadi

$$B(t) = \sum_{i=0}^n C_i^n t^i (1 - t)^{n-i} P_i$$

Sehingga algoritmanya menjadi sebagai berikut.

Misal p adalah List of Point dengan panjang n , dan iterations adalah integer

Prosedur GenerateWithBruteForceFormulated(P, iterations)

1. Misal array hasil adalah R .
2. Lakukan loop sebanyak iterations, tiap loop adalah loop ke k
3. Misal $t = (k+1)/(iterations+1)$
4. Ambil titik $q = B(t)$ dengan P_i pada rumus $B(t)$ adalah setiap titik P .
5. Masukkan q ke R .
6. R adalah hasilnya.

b. Source Code

```
class FormulatedPoint
```

```

export default class FormulatedPoint extends SyncablePoint {

  /**
   * @param {SyncablePoint[]} pointList
   * @param {number} progress
   */
  constructor(pointList, progress) {
    super(pointList[0], pointList[1]);
    /** @type {SyncablePoint[]} */
    this.pointList = pointList;
    this.progress = progress;
    this.sync();
  }

  /** Update the position of this point to be in the center of both points.
   */
  sync() { // override
    const n = this.pointList.length - 1;
    const oneMinProgress = 1 - this.progress
    let newX = 0;
    let newY = 0;
    for(let i = 0; i < this.pointList.length; i++) {
      const comb = combination(n, i);
      const oneMinProgressPow = Math.pow(oneMinProgress, n - i);
      const progressPow = Math.pow(this.progress, i);
      const temp = comb * oneMinProgressPow * progressPow;
      newX += temp * this.pointList[i].x;
      newY += temp * this.pointList[i].y;
    }
    this.x = newX;
    this.y = newY;
  }
}

```

Class Bezier Curve

...

```

/**
 * Generate a quadratic bezier curve based on the points and number of
 iterations
 * @param {CenterPoint[]} p List of input points.
 * @param {number} iterations number of iterations. The bigger, the
 smoother.
 * @param {CenterPoint[]} centerPointResult list of result center points.
 * @param {CenterPoint[]} centerPointWaste list of center points used in
 processing.
 */
generateByBruteForceFormulated(p, iterations) {
  for(let k = 0; k < iterations; k++) {
    const progress = (k+1)/(iterations+1);
    const point = new FormulatedPoint(p, progress);
    this.syncablePointResult.push(point);
  }
}
...

```

c. Analisis Kompleksitas

Kompleksitas dihitung dari banyak operasi aritmatika +, -, *, / (termasuk operasi increment untuk melanjutkan for loop dan pangkat dianggap sebagai for loop dengan perkalian sehingga pangkat n artinya 2^n operasi).

Misal n = panjang array input, i = banyak iterasi.

Banyak iterasi (untuk menyesuaikan banyak titik hasil algoritma Divide and Conquer, maka jumlah iterasi disesuaikan) = 2^i

Sum fungsi B(t) tanpa kombinasi (beserta increment for loop) = $n(1 + 0 + 2(n - k) + 2k + 2 + 2 + 2) = 2n^2 + 7n$

Sum fungsi kombinasi (beserta increment for loop) = $0 + 4 + 8 + \dots + 4(n - 1) = 4n(n - 1)/2 = 2n(n - 1)$

Operasi di luar sum (masih dalam iterasi) = 2

$T(n) = 2^i(2n^2 + 7n + 2n(n - 1) + 2)$

$T(n) = 2^i(4n^2 + 5n + 2)$

$= O(2^i n^2)$

C. Analisis dan Implementasi Algoritma Divide and Conquer Khusus Kuadratik

a. Algoritma

Misal p adalah List of Point dengan panjang 3, dan iterations adalah integer

Prosedur GenerateQuadratic(P, iterations)

1. Ambil titik tengah dari P_0 dan P_1 , misalkan Q_0

Ambil titik tengah dari P_1 dan P_2 , misalkan Q_1

Ambil titik tengah dari Q_0 dan Q_1 , misalkan R_0

2. Untuk kasis iterations > 1

DIVIDE:

Bagi menjadi dua array yaitu semua titik paling kiri dan semua titik paling kanan untuk setiap variabel, misal $Left = \{P_0, Q_0, R_0\}$ dan $Right = \{R_0, Q_1, P_2\}$. (R_0 merupakan titik paling kanan karena hanya ada 1 titik)

CONQUER:

GenerateQuadratic(Left, iterations-1)

COMBINE:

Masukkan hasil GenerateQuadratic (R_0) ke sebuah array.

GenerateQuadratic(Right, iterations-1)

b. Source Code

Class CenterPoint

```
export default class CenterPoint extends SyncablePoint {  
  
    /**  
     * @param {SyncablePoint} point1  
     * @param {SyncablePoint} point2  
     */  
    constructor(point1, point2) {  
        super(point1, point2);  
        this.sync();  
    }  
  
    // override  
    /**
```

```

    * Update the position of this point to be in the center of both points.
    */
    sync() {
        this.x = (this.point1.x + this.point2.x)/2;
        this.y = (this.point1.y + this.point2.y)/2;
    }
}

```

Class Bezier Curve

```

...
/**
    * Generate a quadratic bezier curve based on the points and number of
    iterations
    * @param {CenterPoint[]} p number of iterations. The bigger, the
    smoother.
    * @param {number} iterations number of iterations. The bigger, the
    smoother.
    */
    generateQuadratic(p, iterations) {
        if(iterations > 1) {
            const q0 = new CenterPoint(p[0], p[1]);
            const q1 = new CenterPoint(p[1], p[2]);
            const r0 = new CenterPoint(q0, q1);
            iterationMinusOne = iterations - 1;
            this.generateQuadratic([p[0], q0, r0], iterationMinusOne);
            this.syncablePointResult.push(r0);
            this.generateQuadratic([r0, q1, p[2]], iterationMinusOne);
        }
    }
}
...

```

c. Analisis Kompleksitas

Kompleksitas dihitung berdasarkan banyak masukan ke array yang terjadi.

Misal n = panjang array input, i = banyak iterasi.

Pembangkitan array Left dan Right = 6

Hasil fungsi awal (COMBINE) = 1

Rekursi fungsi (CONQUER) = 2^i

Maka

$$T(n) = (2^i + 1)(6)$$

$$= O(2^i)$$

D. Analisis dan Implementasi Algoritma Divide and Conquer

a. Algoritma

Misal p adalah List of Point (titik) dengan panjang n, dan iterations adalah integer

Prosedur GenerateWithDivideAndConquer(P, iterations)

1. Ambil titik tengah setiap elemen P, masukkan ke sebuah array Q.
2. Misal a adalah banyak titik tengah barusan (yaitu panjang P - 1), b adalah panjang Q saat ini (bernilai sama dengan a untuk saat ini).
3. Ambil titik tengah setiap elemen Q mulai dari elemen ke b-a sampai akhir (yaitu semua titik yang baru dimasukkan pada langkah sebelumnya), masukkan ke array Q.
4. a diubah nilainya jadi banyak titik tengah barusan. b dijadikan panjang Q sekarang.
5. Ulangi langkah 3 dan 4 hingga a menjadi 1 (banyak titik tengah hanya 1).
6. Untuk kasus iterations > 1

DIVIDE:

Bagi menjadi dua array dengan cara, elemen pertama P, dan setiap elemen pertama dari titik yang dimasukkan ke Q, masukkan ke array Left. Sedangkan elemen terakhir P, dan setiap elemen terakhir dari titik yang dimasukkan ke Q, masukkan ke array Right.

$$\text{Artinya Left} = \{P_0, Q_0, Q_{n-1}, Q_{(n-1)+(n-2)}, Q_{(n-1)+(n-2)+(n-3)}, \dots\}$$

$$\text{Right} = \{P_{n-1}, Q_{n-2}, Q_{(n-2)+(n-3)}, Q_{(n-2)+(n-3)+(n-4)}, \dots\}$$

Sebagai ilustrasi, dapat dilihat sebagai berikut:

Misal n = 5, maka

$$P = \{P_0, P_1, P_2, P_3, P_4\}, Q = \{Q_0, Q_1, Q_2, Q_3, | Q_4, Q_5, Q_6, | Q_7, Q_8, | Q_9\}$$

$$\text{Left} = \{P_0, Q_0, Q_4, Q_7, Q_9\}, \text{Right} = \{P_4, Q_3, Q_6, Q_8, Q_9\}$$

Dalam hal ini, Q_0 hingga Q_3 adalah titik tengah P_0 hingga P_4 , Q_4 hingga Q_6 adalah titik tengah Q_0 hingga Q_3 , dan seterusnya.

CONQUER:

GenerateWithDivideAndConquer(Left, iterations-1)

COMBINE:

Masukkan hasil GenerateQuadratic (elemen terakhir Q) ke sebuah array.

GenerateWithDivideAndConquer(Right, iterations-1)

b. Source Code

Class CenterPoint

```
export default class CenterPoint extends SyncablePoint {  
  
    /**  
     * @param {SyncablePoint} point1  
     * @param {SyncablePoint} point2  
     */  
    constructor(point1, point2) {  
        super(point1, point2);  
        this.sync();  
    }  
  
    // override  
    /**  
     * Update the position of this point to be in the center of both points.  
     */  
    sync() {  
        this.x = (this.point1.x + this.point2.x)/2;  
        this.y = (this.point1.y + this.point2.y)/2;  
    }  
}
```

Class BezierCurve

```
...  
  
/**  
 * get the next iteration from the current points  
 * @param {CenterPoint[]} p current points in this iteration  
 * @returns left points, right points, and all the middle points created.
```


All middle points will be used in dynamic update feature

```
*/
generateLeftRight(p) {
  const left = [];
  const right = [];
  const centerList = []; // same as center list but not a matrix
  const length = p.length;

  // sub center points
  for(let i = 0; i < length-1; i++) {
    const centerPoint = new CenterPoint(p[i], p[i+1]);
    centerList.push(centerPoint);
  }

  // sub sub sub... center points
  let count = 0;
  for(let i = 2; i < length; i++) {
    const lengthMinI = length - i;
    for(let j = 0; j < lengthMinI; j++) {
      const countPlusJ = count + j;
      const centerPoint = new CenterPoint(centerList[countPlusJ],
centerList[countPlusJ+1]);
      centerList.push(centerPoint);
    }
    count += lengthMinI + 1;
  }

  // 0 1 2 | 3 4 | 5
  // left: 0 3 5
  // right: 5 4 2

  let countLeft = 0;
  let countRight = centerList.length - 1;
  left.push(p[0]); // 0
  for(let i = 1; i < length; i++) {
    left.push(centerList[countLeft]);
    right.push(centerList[(countRight)]);
    countLeft += length - i;
```

```

        countRight -= i;
    }
    right.push(p[length-1]); // 3

    return [left, right, centerList];
}

/**
 * Generate a quadratic bezier curve based on the points and number of
iterations
 * @param {CenterPoint[]} p List of input points.
 * @param {number} iterations number of iterations. The bigger, the
smoother.
 */
generateByDivideAndConquer(p, iterations) {
    if(iterations > 0){
        /** @type {[CenterPoint[], CenterPoint[], CenterPoint[][]} */
        const [left, right] = this.generateLeftRight(p);
        const iterationMinusOne = iterations - 1;
        this.generateByDivideAndConquer(left, iterationMinusOne);
        this.syncablePointResult.push(left[p.length-1]);
        this.generateByDivideAndConquer(right, iterationMinusOne);
    }
}
...

```

c. Analisis Kompleksitas

Kompleksitas dihitung berdasarkan banyak masukan ke array yang terjadi.

Misal n = panjang array input, i = banyak iterasi.

Pembangkitan array $Q = (n - 1) + (n - 2) + (n - 3) + \dots + 1 = n(n - 1)/2$

Pembangkitan array Left dan Right = $2n$

Hasil fungsi awal (COMBINE) = 1

Rekursi fungsi (CONQUER) = 2^i

Maka

$T(n) = (2^i + 1)(n(n - 1)/2 + 2n)$

$$T(n) = (2^i)((n^2 + 3n)/2)$$

$$= O(2^i n^2)$$

E. Hasil Analisis Algoritma

Dari notasi Big O pada setiap algoritma yang digunakan, tentunya yang paling cepat adalah untuk Bezier Curve khusus Kuadratik. Ini wajar karena banyak titik masukannya konstan yaitu 3 titik. Maka untuk algoritma ini akan kita kecualikan pada pembahasan analisis ini.

Berdasarkan hasil analisis kompleksitas masing-masing algoritma, semuanya memiliki kompleksitas yang sama. Namun hal ini bukan berarti masing-masing algoritma akan memiliki waktu eksekusi yang sama. Terdapat kompleksitas waktu yang berbeda-beda pula. Walaupun begitu, kompleksitas waktu juga tidak dapat diukur dengan adil. Pada algoritma Brute Force dengan turunan rumus, terdapat banyak operasi aritmatika yang terjadi. Sedangkan pada algoritma Divide and Conquer, terdapat banyak manipulasi array yang terjadi. Kedua hal ini akan memberikan hasil yang berbeda pula untuk bahasa pemrograman yang berbeda. Misalnya untuk bahasa pemrograman seperti C yang memiliki array statik, masukan ke array akan lebih cepat dibanding array pada bahasa dengan array dinamis seperti javascript. Akan banyak pula faktor lain seperti caching, abstraksi implementasi array pada suatu bahasa pemrograman, tingkah laku memori stack saat pemanggilan fungsi yang banyak, ketersediaan sumber daya, dan sebagainya yang akan memengaruhi waktu pembangkitan kurva bezier. Dengan melihat notasi Big O yang sama untuk masing-masing algoritma, sebenarnya ketidakpastian hal ini adalah hal yang wajar.


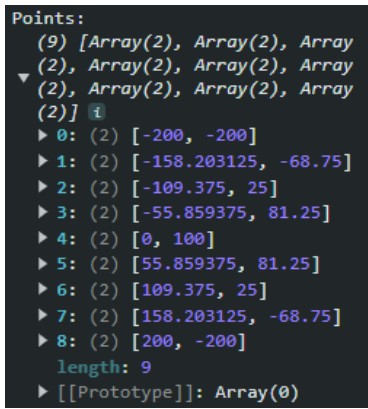
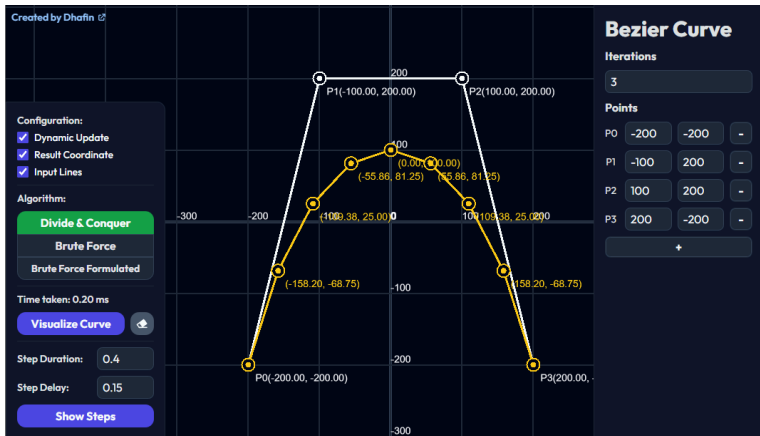
Oleh sebab itu, diperlukan pengujian implementasi secara langsung. Hal ini juga sebenarnya belum dapat menyimpulkan algoritma mana yang lebih cepat. Namun akan memberi kesimpulan algoritma mana, di lingkungan mana dan dengan sumber daya apa yang lebih cepat. Dengan melihat hasil pengujian pada bab selanjutnya, algoritma Brute Force dengan turunan rumus akan konsisten menjadi yang paling efisien. Tentunya akan ada pertimbangan dari penggunaan bahasa pemrograman yang dipilih. Pada bahasa Javascript, array yang digunakan adalah array dinamis, sehingga akan ada tambahan waktu jika kita menggunakan banyak manipulasi array seperti pada algoritma Divide and Conquer dibanding algoritma Brute Force.

BAB IV Hasil Pengujian

A. Pengujian Waktu Eksekusi

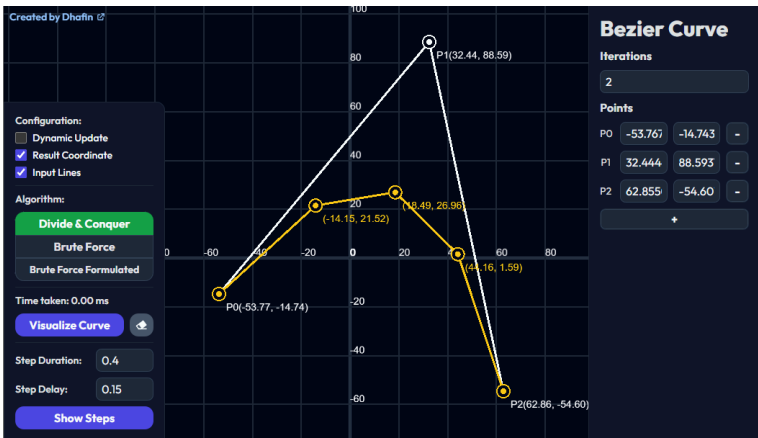
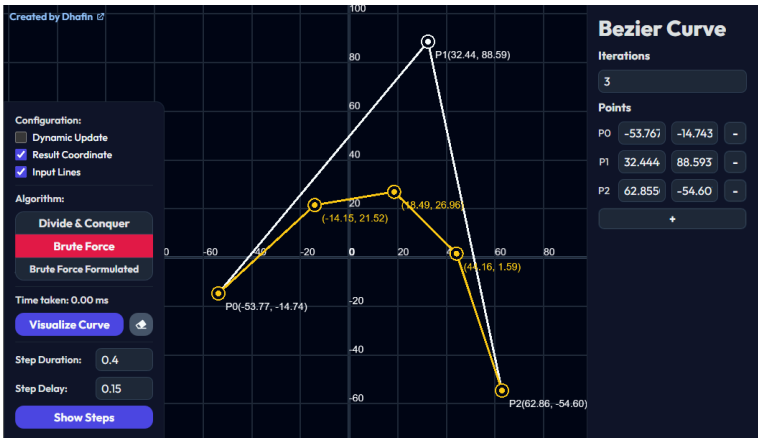
Pengujian akan dilakukan dengan menguji langsung website dan melihat detail kordinat melalui console dev tools browser agar terlihat lebih jelas. Pada console dev tools juga ada detail waktu eksekusi hingga beberapa angka di belakang koma yang lebih detail. Untuk mempermudah pengujian, ada pula point input dapat didrag dengan mouse, zoom dapat dilakukan dengan scroll mouse wheel, dan ada pula ceklis untuk dynamic update kurva. Lalu jika output terlalu banyak maka akan ditampilkan beberapa saja. Untuk titik-titik input dan output hasilnya selalu sama sehingga yang ditampilkan hanya yang untuk Divide and Conquer saja. Yang berbeda adalah jumlah iterasi. Untuk menyesuaikan, algoritma akan memiliki iterasi yang berbeda yaitu:

$$\text{Iterasi Brute Force} = 2^{\text{iterasi Divide and Conquer}} - 1$$

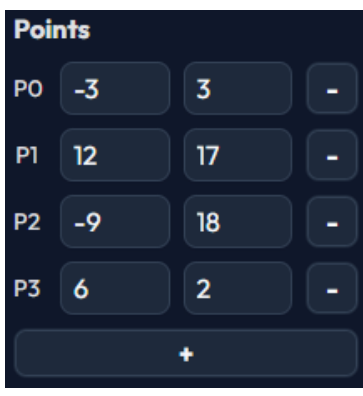
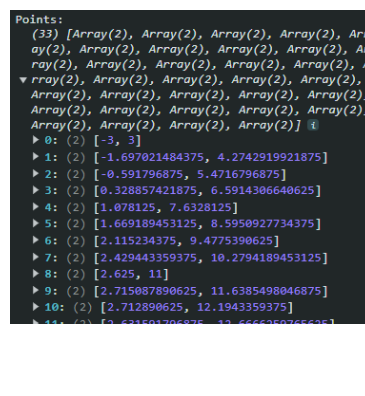
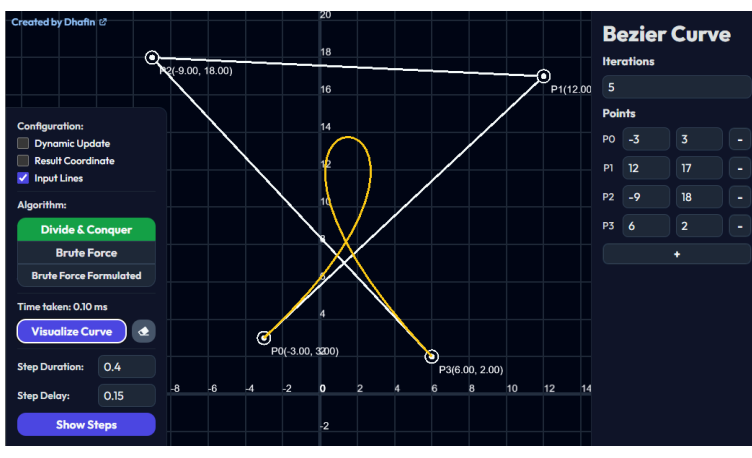
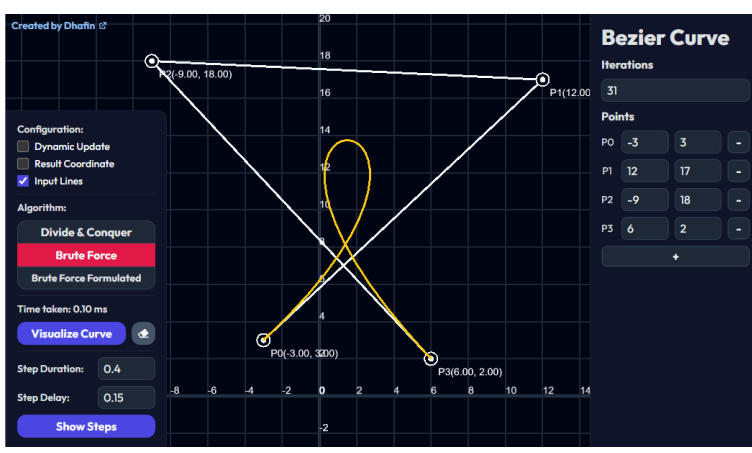
1	Iterasi		Input	Output
	Divide and Conquer: 3 Brute Force: 7 Brute Force dengan Turunan Rumus: 7			
	Algoritma	Waktu	Visualisasi	
	Divide and Conquer	0.2 ms		

	Brute Force	0.2 ms	
	Brute Force dengan Turunan Rumus	0.1 ms	

2	Iterasi Divide and Conquer		<div> <div>Points</div> <div> <div>P0</div> <div>-53.767</div> <div>-14.743</div> <div>-</div> </div> <div> <div>P1</div> <div>32.444</div> <div>88.593</div> <div>-</div> </div> <div> <div>P2</div> <div>62.855</div> <div>-54.60</div> <div>-</div> </div> <div>+</div> </div>	<div> <div>Points:</div> <div> <div>(5) [Array(2), Array(2), Array(2), Array(2), Arra</div> <div>y(2)]</div> <div>0: (2) [-53.76797610509202, -14.743161668420768]</div> <div>1: (2) [-14.149361072467556, 21.517005326971443]</div> <div>2: (2) [18.494015728982223, 26.96064362907867]</div> <div>3: (2) [44.16215429925732, 1.5877532379009196]</div> <div>4: (2) [62.85505463835773, -54.601665846561815]</div> <div>length: 5</div> <div>▶ [[Prototype]]: Array(0)</div> </div> </div>
	Algoritma	Waktu	Visualisasi	

	Divide and Conquer	0 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input checked="" type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div></div><div>Time taken: 0.00 ms</div><div><div>Visualize Curve</div></div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div><div></div></div>
	Brute Force	0 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input checked="" type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div></div><div>Time taken: 0.00 ms</div><div><div>Visualize Curve</div></div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div><div></div></div>
	Brute Force dengan Turunan Rumus	0 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input checked="" type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div></div><div>Time taken: 0.00 ms</div><div><div>Visualize Curve</div></div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div><div></div></div>

3	Iterasi Divide and Conquer	Input	Output
---	----------------------------	-------	--------

<p>Divide and Conquer: 5 Brute Force: 31 Brute Force dengan Turunan Rumus: 31</p>		
<p>Algoritma</p>	<p>Waktu</p>	<p>Visualisasi</p>
<p>Divide and Conquer</p>	<p>0.1 ms</p>	
<p>Brute Force</p>	<p>0.1 ms</p>	

Brute Force dengan Turunan Rumus

0 ms

Created by Dhaifin

Configuration:

☐ Dynamic Update

☐ Result Coordinate

☒ Input Lines

Algorithm:

Divide & Conquer

Brute Force

Brute Force Formulated

Time taken: 0.00 ms

Visualize Curve

Step Duration: 0.4

Step Delay: 0.15

Show Steps

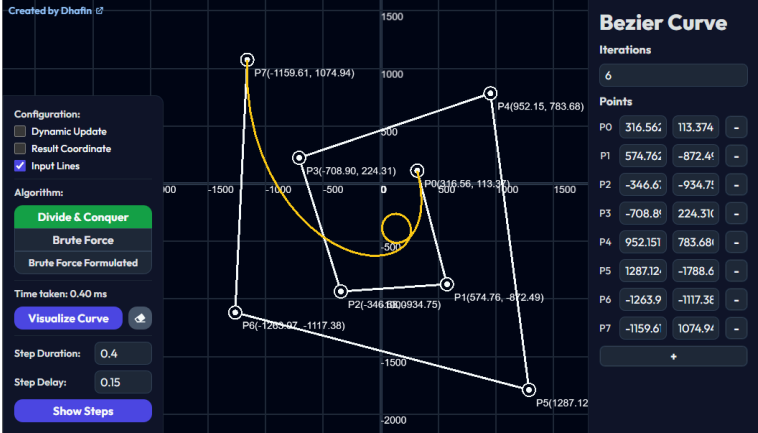
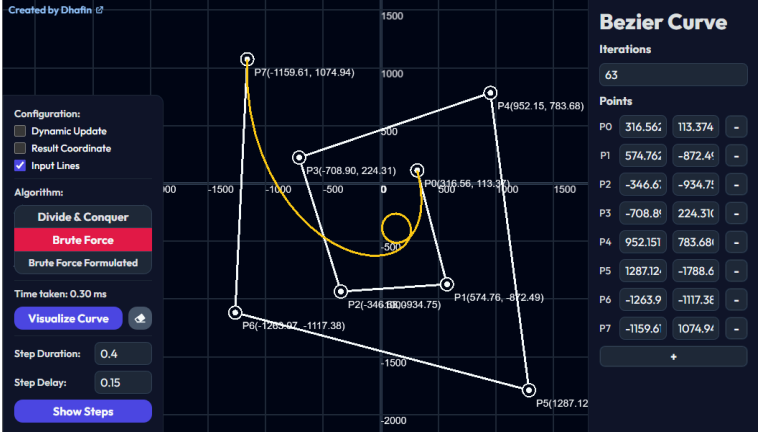
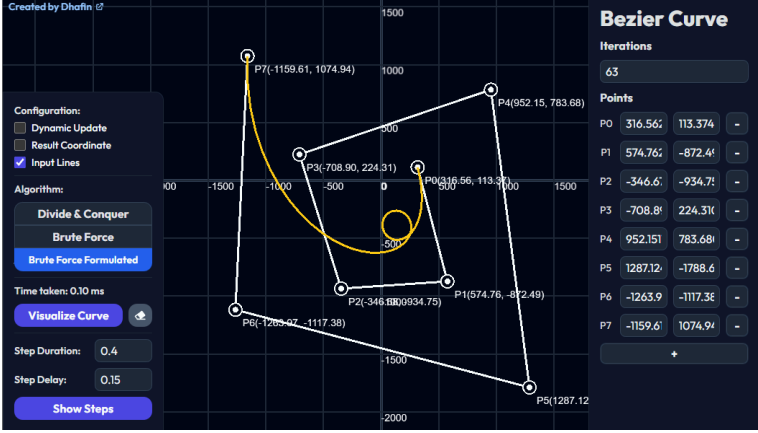
Bezier Curve

Iterations: 31

Points

P0	-3	3
P1	12	17
P2	-9	18
P3	6	2

+

	Divide and Conquer	0.4 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div><div>Time taken: 0.40 ms</div><div>Visualize Curve</div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div></div></div>
	Brute Force	0.3 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div><div>Time taken: 0.30 ms</div><div>Visualize Curve</div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div></div></div>
	Brute Force dengan Turunan Rumus	0.1 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div><div>Time taken: 0.10 ms</div><div>Visualize Curve</div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div></div></div>

5	Iterasi Divide and Conquer	Input	Output
---	----------------------------	-------	--------

Brute Force dengan Turunan Rumus

0 ms

Created by Dhaifin 62

Configuration:

☒ Dynamic Update

☐ Result Coordinate

☒ Input Lines

Algorithm:

Divide & Conquer

Brute Force

Brute Force Formulated

Time taken: 0.10 ms

Visualize Curve

🗑️

Step Duration: 0.4

Step Delay: 0.15

Show Steps

Bezier Curve

Iterations

127

Points

P0

10

10

-

P1

10

10

-

P2

10

10

-

P3

10

10

-

P4

10

10

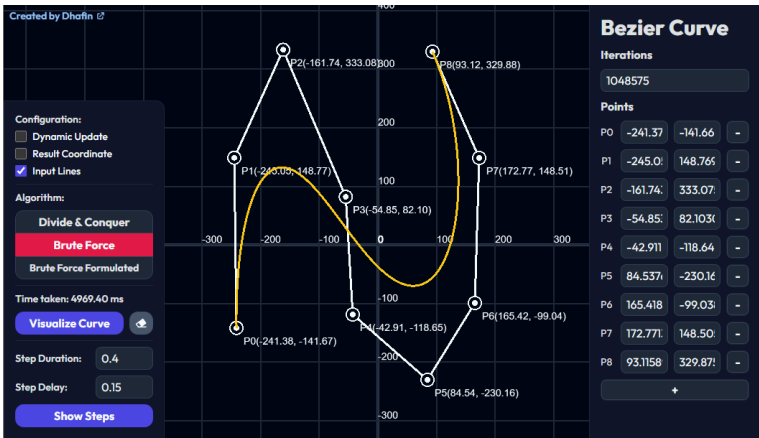
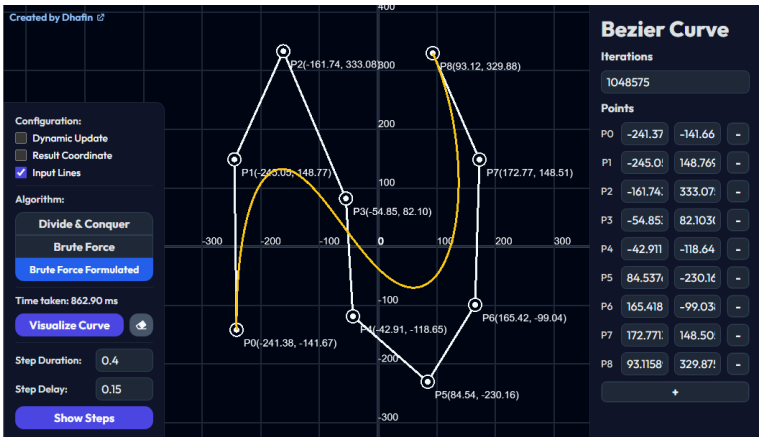
-

+

	Brute Force	71 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div></div><div>Time taken: 71.00 ms</div><div>Visualize Curve</div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div><div></div></div>
	Brute Force dengan Turunan Rumus	32.9 ms	<div><div>Created by Dhaifin</div><div><div>Configuration:<div><div><input type="checkbox"/> Dynamic Update</div><div><input type="checkbox"/> Result Coordinate</div><div><input checked="" type="checkbox"/> Input Lines</div></div><div>Algorithm:<div><div>Divide & Conquer</div><div>Brute Force</div><div>Brute Force Formulated</div></div></div><div>Time taken: 32.90 ms</div><div>Visualize Curve</div><div>Step Duration: 0.4</div><div>Step Delay: 0.15</div><div>Show Steps</div></div></div><div></div></div>

6	Iterasi Divide and Conquer	Input	Output
---	----------------------------	-------	--------

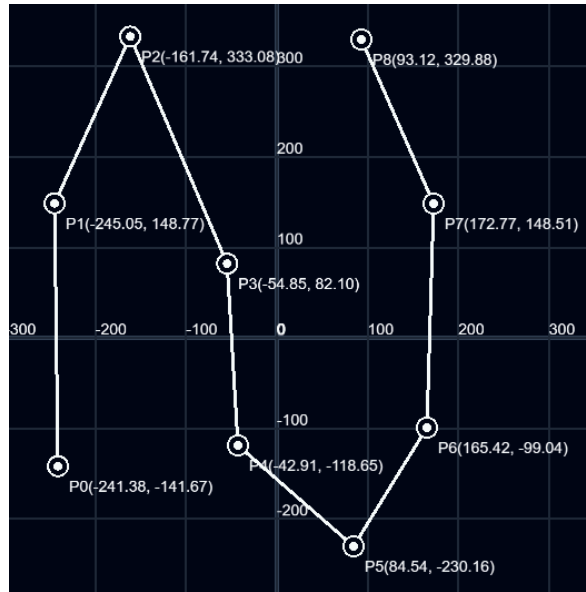
[illegible]

	Brute Force	4969.4 ms	
	Brute Force dengan Turunan Rumus	862.9 ms	

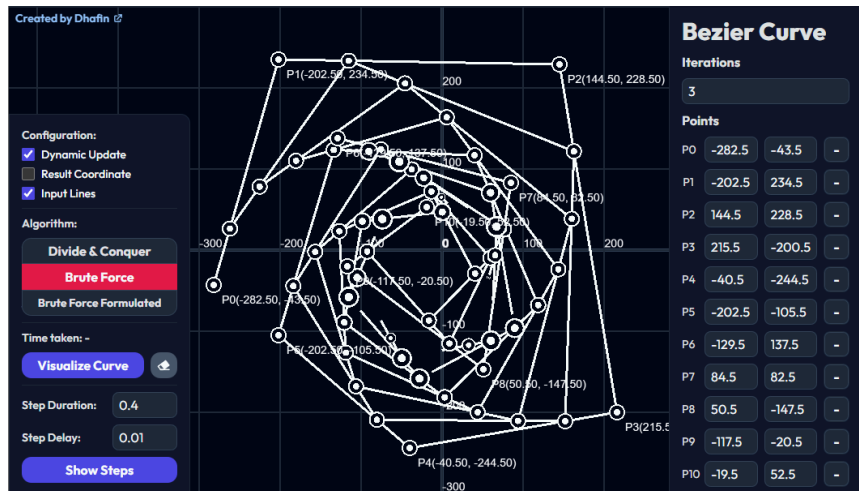
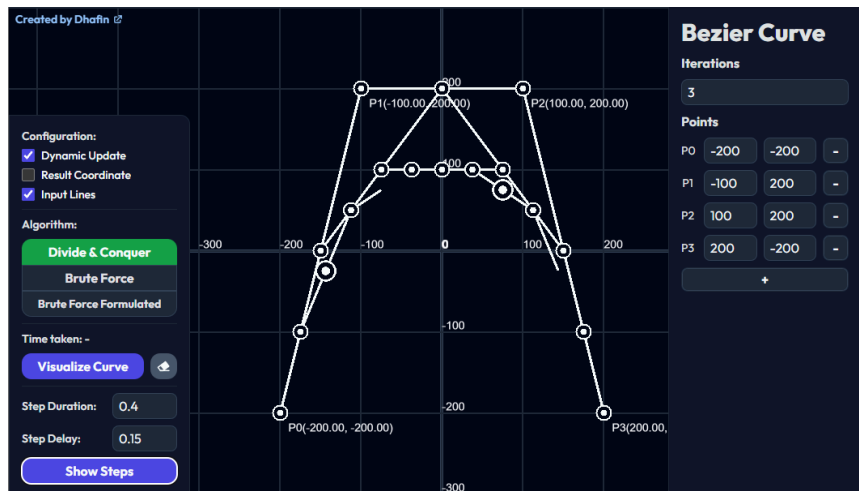
D. Pengujian Fitur Tambahan

Nama Fitur	Tangkapan Layar
------------	-----------------

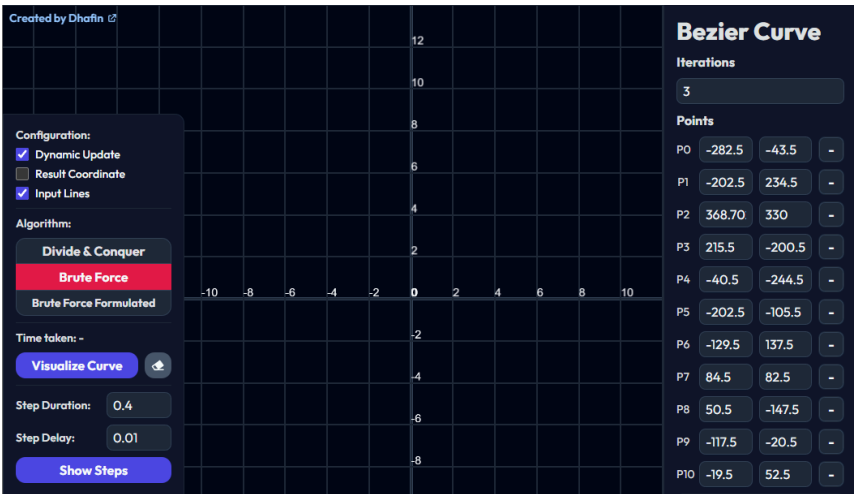
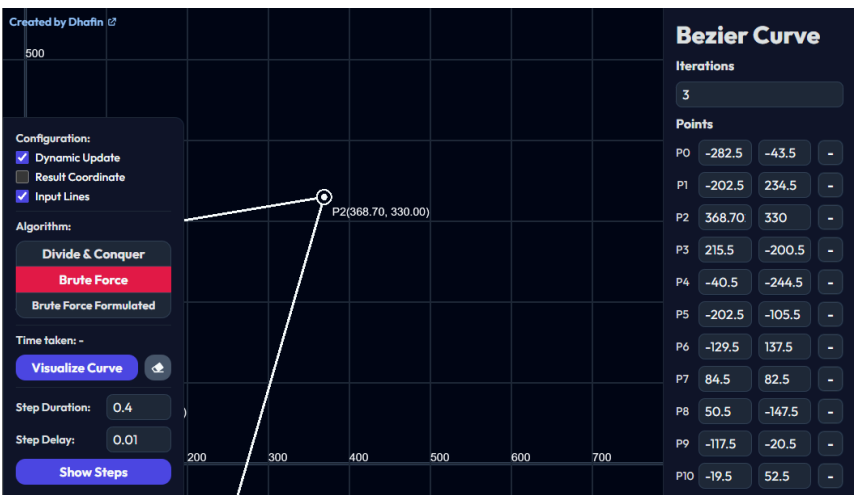
Kurva n titik kontrol



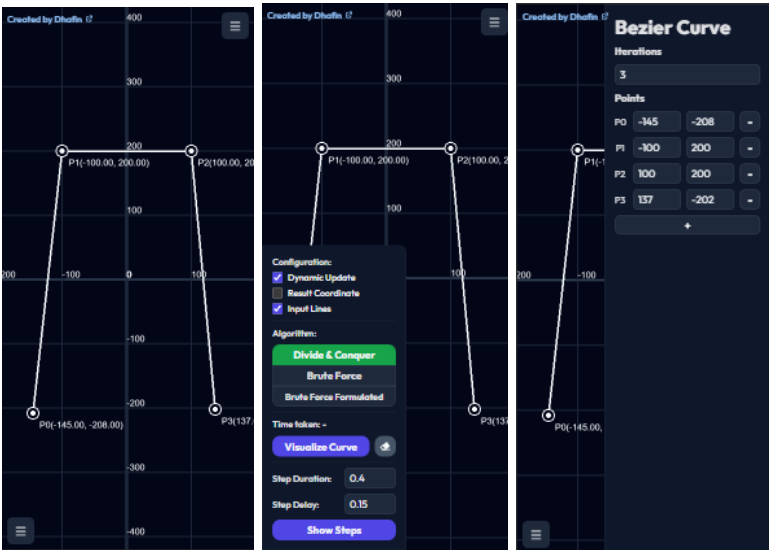
Visualisasi Proses Pembuatan Kurva (Tangkapan layar merupakan tangkapan di tengah-tengah animasi berlangsung)



Navigation (zoom, drag)
dengan auto scaling
koordinat.



Responsif untuk mobile



BAB V Kesimpulan

Berbagai persoalan pemrograman dapat diselesaikan dengan berbagai algoritma. Salah satunya adalah algoritma Divide and Conquer. Algoritma ini merupakan sebuah algoritma yang dilakukan dengan membagi persoalan menjadi upa persoalan yang lebih kecil lalu menyelesaikan persoalan yang lebih kecil tersebut dan menggabungkan kembali solusi tersebut kembali menjadi persoalan awal. Salah satu masalah yang dapat diselesaikan dengan algoritma ini adalah persoalan pembangkitan kurva bezier. Melalui perbandingan dengan algoritma Brute Force, dapat disimpulkan bahwa algoritma brute force tidak selamanya lebih lambat dibanding algoritma Divide and Conquer. Kedua algoritma ini memiliki kompleksitas yang sama untuk menghasilkan kurva bezier dengan jumlah titik akhir yang sama. Namun walaupun memiliki notasi Big O yang sama, kompleksitas waktunya akan berbeda. Tapi walaupun begitu, perhitungan kompleksitas tersebut juga tidak dapat dilakukan dengan cukup adil. Pada algoritma Brute Force, terdapat banyak operasi aritmatika yang terjadi. Sedangkan pada algoritma Divide and Conquer, terdapat banyak manipulasi array yang terjadi. Masing-masing dari hal ini akan mengakibatkan hasil yang akan berbeda-beda tergantung dari implementasi akhirnya. Akan banyak pula faktor lain seperti caching, abstraksi implementasi array pada suatu bahasa pemrograman, tingkah laku memori stack saat pemanggilan fungsi yang banyak, ketersediaan sumber daya, dan sebagainya yang akan memengaruhi waktu pembangkitan kurva bezier. Oleh sebab itu, diperlukan pengujian implementasi secara langsung. Hal ini juga sebenarnya belum dapat menyimpulkan algoritma mana yang lebih cepat. Namun akan memberi kesimpulan algoritma mana, di lingkungan mana dan dengan sumber daya apa yang lebih cepat.

BAB VI Lampiran

Source code: https://github.com/DhafinFawwaz/Tucil2_13522084

Link hasil: <https://beziercurvegenerator.vercel.app/>

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