LAPORAN TUGAS KECIL 2

Membangun Kurva Bézier dengan Algoritma Titik Tengah
berbasis Divide and Conquer
IF2211 – Strategi Algoritma

Oleh :
Kelompok 2

Mohamad Akmal Ramadan	13522161
Valentino Chryslie Triadi	13522164

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

DAFTAR ISI

BAB I
Analisis Algoritma

1.1 Analisis Implementasi Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah paradigma pemecahan masalah yang membagi masalah besar menjadi submasalah yang lebih kecil, menyelesaikan submasalah tersebut secara rekursif, dan kemudian menggabungkan solusi submasalah menjadi solusi untuk masalah asli. Algoritma ini sering digunakan dalam berbagai konteks, termasuk pencarian, pengurutan, dan pemecahan masalah geometri, seperti pada kurva Bezier.

Dalam konteks pembuatan kurva Bezier, algoritma *divide and conquer* digunakan untuk membagi kurva menjadi segmen-segmen yang lebih kecil dan kemudian membangun kurva dengan menghubungkan segmen-segmen tersebut. Proses ini dilakukan dengan mencari titik tengah antara titik kontrol yang ada dan menggunakan titik-titik tersebut untuk membagi kurva menjadi dua bagian yang lebih kecil. Ini diulang secara rekursif hingga mencapai tingkat iterasi yang diinginkan.

Terdapat dua algoritma yang diimplementasi untuk membentuk kurva Bezier ini, yaitu algoritma untuk tiga titik control dan algoritma untuk *n* buah titik control, kedua algoritma ini diimplementasikan dalam Bahasa *TypeScript* dengan *React Components (.tsx).*

1.1.1 Analisis Implementasi Algoritma Divide and Conquer Tiga Titik Kontrol

Fungsi *Bezier3Points* (pada bab 2.1) merupakan implementasi algoritma *divide and conquer* untuk membuat kurva Bezier dengan 3 titik kontrol. Pada setiap pemanggilan fungsi, algoritma mengambil titik kontrol awal, tengah, dan akhir, dan kemudian mencari titik tengah dari masing-masing titik kontrol.

Fungsi ini membagi dua iterasi menjadi *leftCurve* dan *rightCurve*, *leftCurve* akan menghasilkan point-point hasil pembagian dari iterasi kurva sebelah kiri (P0, Q0, R0) dan *rightCurve* dari sebelah kanan (R0, Q1, P2). Iterasi ini akan terus membagi dua buah garis dan mengambil titik tengahnya untuk membuat titik kontrol baru pada masing-masing bagaian (bagian kiri dan kanan) yang akan digunakan di iterasi selanjutnya, ketika iterasi selanjutnya dipanggil, jumlah iterasi akan dikurang satu. Hal ini akan terus dilakukan hingga jumlah iterasi sudah habis atau tersisa satu iterasi (hal ini merupakan salah satu solusi untuk mempersingkat algoritma), pada kondisi ini fungsi tidak akan melakukan pembagian lagi. Ketika sudah tidak ada operasi pembagian yang dilakukan, maka point-point yang dihasilkan dari variabel konstan *leftCurve* dan *rightCurve* akan dijadikan satu membentuk titik-titik yang merupakan titik-titik belok kurva Bezier. Jadi, dalam mengimplementasikan algoritma *divide and conquer* ke algoritma *Bezier3Points* kami

menerapkan pembagian tiga buah titik kontrol sebagai proses *divide* dan menggabungkan titik-titik kontrol baru sebagai proses *conquer*.

1.1.2 Analisis Implementasi Algoritma *Divide and Conquer* n Titik kontrol

Fungsi *drawBezierCurve* (pada bab 2.2) merupakan implementasi algoritma *divide and conquer* untuk membuat kurva Bezier dengan n titik kontrol. Fungsi *drawBezierCurve* menerima dua input, yaitu titik kontrol dan jumlah iterasi. Langkah pertama, fungsi akan mencari titik tengah dari masing-masing titik kontrol (P0, P1, P2, P3). Langkah kedua, dari titik tengah yang dihasilkan (titik tengah antara P0 dan P1 adalah Q0, titik tengah antara P1 dan P2 adalah Q1, titik tengah antara P2 dan P3 adalah Q2). Langkah ketiga, fungsi akan mencari titik tengah kembali dari titik-titik tengah yang sebelumnya didapatkan (R0 titik tengah antara Q0 dan Q1, R1 titik tengah antara Q1 dan Q2). Langkah terakhir, fungsi akan mengecek terlebih dahulu apakah ini iterasi yang terakhir atau tidak, apabila iya, maka hasil akhir adalah titik tengah dari langkah ketiga ditambah dengan *initial point* dan *final point* dari kurva, menjadi P0, R0, R1, P3. Namun apabila ini bukan iterasi terakhir, maka fungsi akan melakukan langkah pertama kembali, yaitu mencari titik tengah, namun dari titik-titik yang berbeda dari langkah pertama sebelumnya, yang mana titik-titik yang akan dicari tengahnya merupakan titik-titik gabungan yang didapat dari langkah kedua dan ketiga, yang menjadi P0, Q0, R0, P1, Q1, R1, P2, Q2, P3.

Fungsi akan terus melakukan langkah-langkah di atas sesuai dengan jumlah iterasi yang diberikan.

1.2 Analisis Implementasi Algoritma *Bruteforce*

Fungsi drawBezierCurveBruteForce ini mengimplementasikan metode brute force untuk menggambar kurva Bézier. Fungsi ini bekerja dengan menghitung posisi setiap titik di sepanjang kurva berdasarkan formula Bézier, yang merupakan kombinasi linier dari titik-titik kontrol dengan bobot yang ditentukan oleh polinomial Bernstein.

Pertama, fungsi ini menghitung jumlah titik yang akan dihasilkan, yang ditentukan oleh parameter iterate dan jumlah titik kontrol. Kemudian, untuk setiap titik yang akan dihasilkan, fungsi ini menghitung parameter t, yang mewakili posisi titik di sepanjang kurva.

Selanjutnya, fungsi ini menghitung koordinat x dan y dari titik di sepanjang kurva dengan mengambil rata-rata tertimbang dari koordinat titik kontrol. Bobot untuk rata-rata ini dihitung menggunakan polinomial Bernstein, yang merupakan fungsi dari t dan posisi titik kontrol dalam array.

Setelah koordinat x dan y dihitung, titik baru dengan koordinat ini ditambahkan ke array resPoints. Proses ini diulangi untuk setiap titik yang akan dihasilkan.

Sebagai perbandingan, metode divide and conquer, seperti yang diimplementasikan dalam fungsi Bezier3Points, bekerja dengan cara yang sedikit berbeda. Metode ini memecah kurva Bézier menjadi dua kurva Bézier yang lebih kecil pada setiap iterasi, dan menghitung titik-titik di sepanjang kedua kurva ini secara rekursif. Metode ini efisien karena memanfaatkan fakta

bahwa titik tengah dari segmen garis antara dua titik pada kurva Bézier juga berada di kurva Bézier. Selanjutnya akan dibahas di bab Analisis.

BAB II
Source Code Program

2.1 *Source code* algoritma *divide and conquer* tiga buah titik

```
function middlePoint(P0: Point, P1: Point) {
    return {
        x: (P0.x + P1.x) / 2,
        y: (P0.y + P1.y) / 2
    }
}

export const Bezier3Points = ({points, iterate} : nBezierProps) => {
    if (iterate === 0) return points;
    const P0 = points[0];
    const P1 = points[1];
    const P2 = points[2];

    const Q0 : Point = middlePoint(P0, P1); // Left
    const Q1 : Point = middlePoint(P1, P2); // right
    const R0 : Point = middlePoint(Q0, Q1); // middle

    if (iterate === 1) return [P0, R0, P2];

    const leftCurve : Point[] = Bezier3Points({points:[P0, Q0, R0], iterate: iterate - 1});
    const rightCurve : Point[] = Bezier3Points({points:[R0, Q1, P2], iterate: iterate - 1});

    return [...leftCurve, ...rightCurve];
}
```

2.2 *Source code* algoritma *divide and conquer* n buah titik

```ts
export const drawBezierCurve = ({ points, iterate }: nBezierProps) => {
  const start = performance.now();
  const resPoints: Point[] = [];
  let num = points.length;

  for (let i = 0; i < iterate; i++) {
    let temPoints = [];
    let temPoints2 = [];
    let k = 1;

    for (let j = 0; j < num - 1; j++) {
      const mid = midPoint(
        points[j].x,
        points[j].y,
        points[j + 1].x,
        points[j + 1].y
      );
      temPoints.push(mid);
      if (j > 0) {
        const mid2 = midPoint(
          temPoints[k - 1].x,
          temPoints[k - 1].y,
          temPoints[k].x,
          temPoints[k].y
        );
        if (i < iterate - 1) {
          temPoints2.push(temPoints[k - 1]);
          temPoints2.push(mid2);
          if (num - 2 == k) {
            temPoints2.push(temPoints[k]);
          }
        } else {
          resPoints.push(mid2);
        }
        k++;
      }
    }
    temPoints2.unshift(points[0]);
    temPoints2.push(points[points.length - 1]);
    num = temPoints2.length;
    points = temPoints2;
  }

  resPoints.unshift(points[0]);
  resPoints.push(points[points.length - 1]);
  const end = performance.now();
  return {
    pointsResult: resPoints,
    time: end - start
  };
};
```

2.3 *Source code* algoritma *bruteforce* n buah titik

```ts
export const drawBezierCurveBruteForce = ({
  points,
  iterate,
}: nBezierProps) => {
  const start = performance.now();
  const resPoints: Point[] = [];
  const n = points.length - 1;
  const numOfiterate = Math.pow(2, iterate) * (points.length - 2);

  for (let i = 0; i <= numOfiterate; i++) {
    const t = i / numOfiterate;
    let x = 0;
    let y = 0;

    for (let j = 0; j <= n; j++) {
      const blend =
        binomialCoefficient(n, j) * Math.pow(1 - t, n - j) * Math.pow(t, j);
      x += blend * points[j].x;
      y += blend * points[j].y;
    }

    resPoints.push({ x, y });
  }

  const end = performance.now();
  return {
    pointsResult: resPoints,
    time: end - start
  };
};
```

BAB III
Tangkapan Layar

3.1 Contoh 1

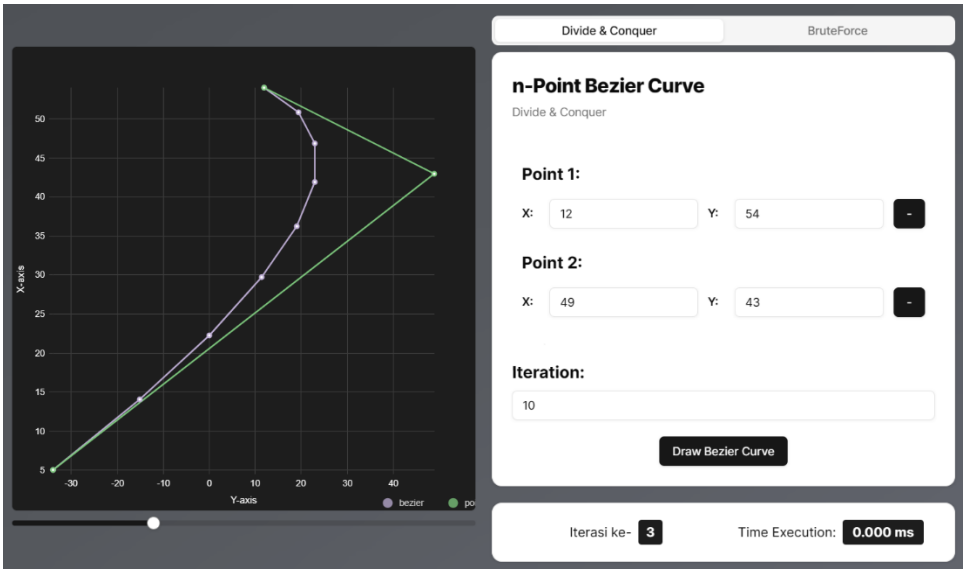
Input:

Point 1: (12,54)

Point 2: (49,43)

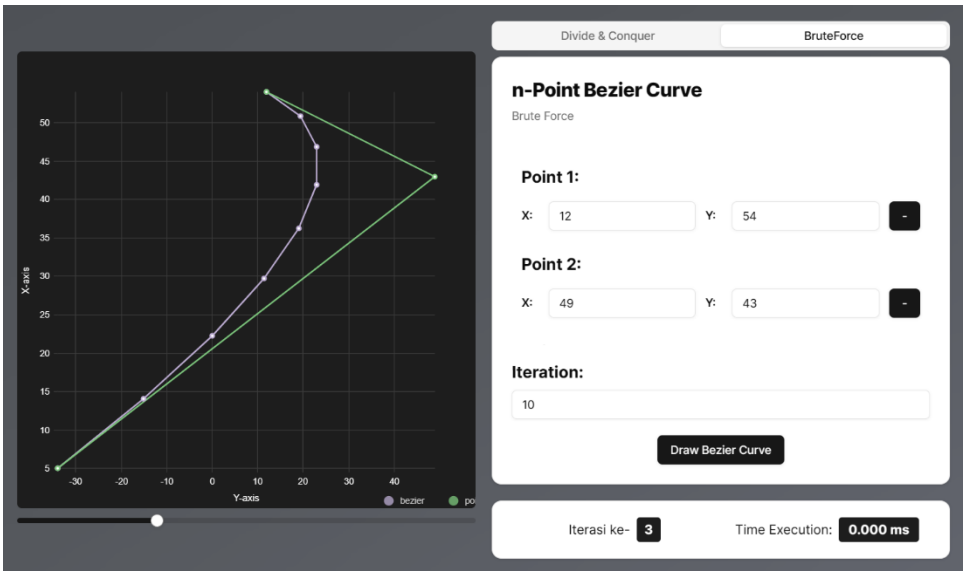
Point 3: (-34,5)

Output DNC:



Real Time execution: 0.010009765625 ms

Output Bruteforce:



Real Time Execution: 0.009033203125 ms

3.2 Contoh 2

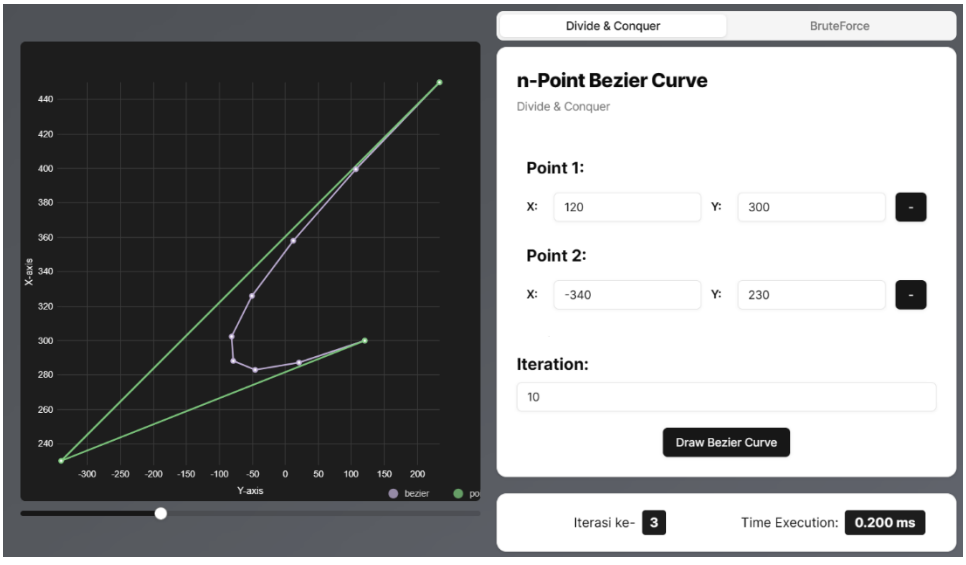
Input:

Point 1: (120,300)

Point 2: (-340,230)

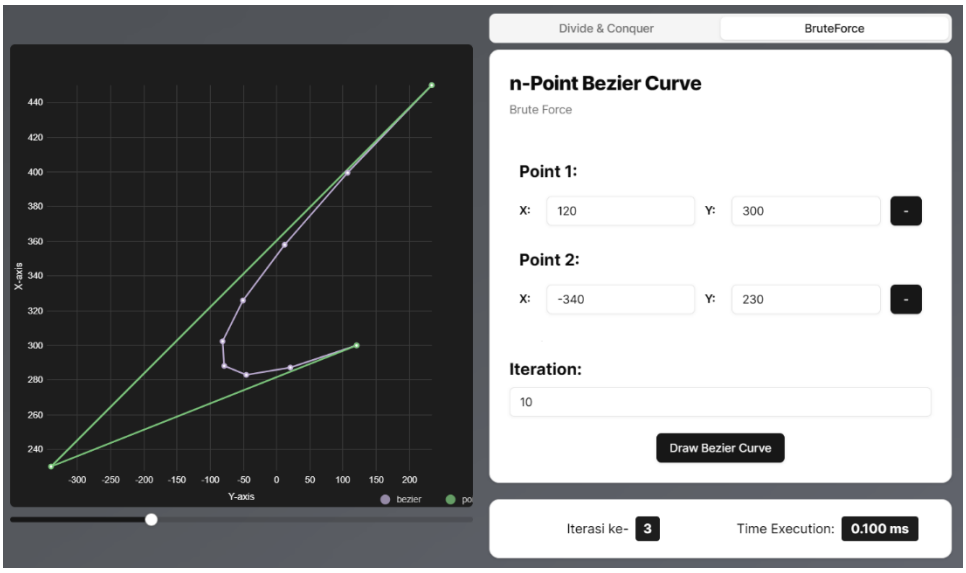
Point 3: (234,450)

Output DNC:



Real Time execution: 0.012939453125 ms

Output Bruteforce:



Real Time Execution: 0.01416015625 ms

3.3 Contoh 3

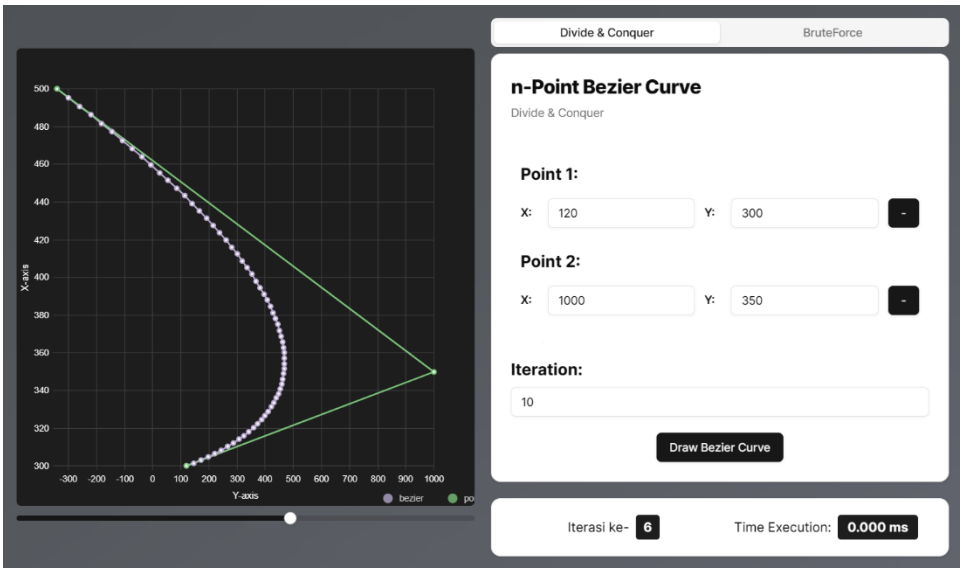
Input:

Point 1: (120,300)

Point 2: (1000,350)

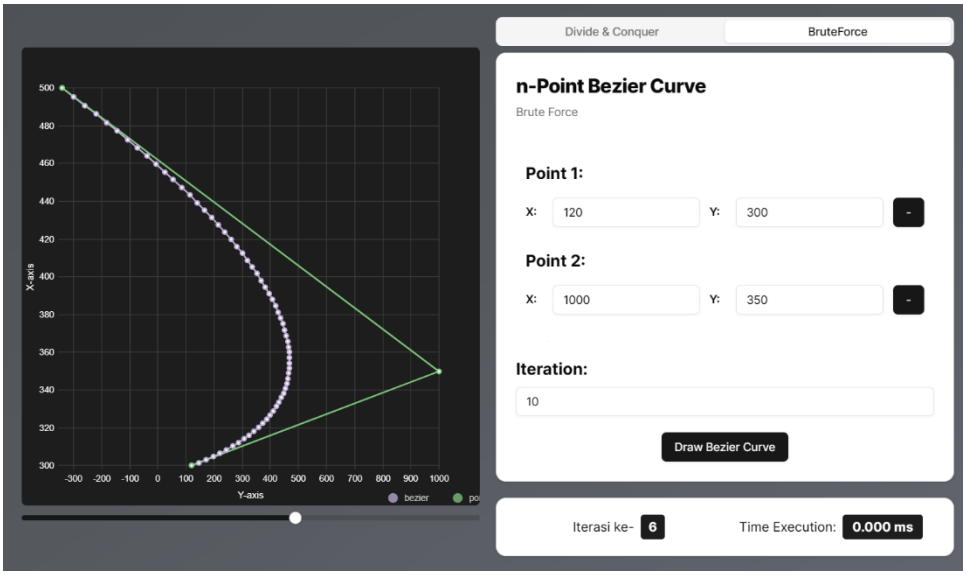
Point 3: (234,450)

Output DNC:



Real Time execution: 0.02099609375 ms

Output Bruteforce:



Real Time Execution: 0.031982421875 ms

3.4 Contoh 4

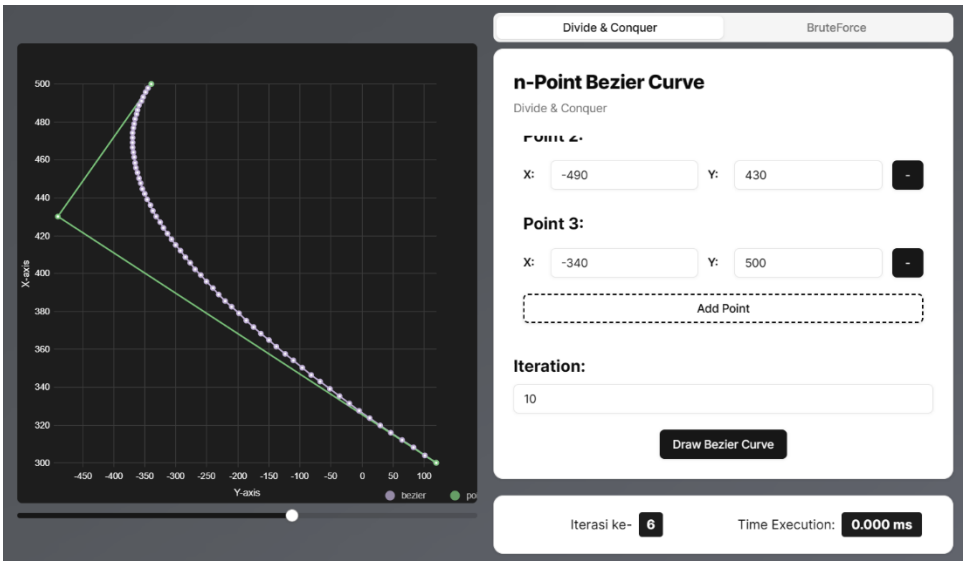
Input:

Point 1: (120,300)

Point 2: (-490,430)

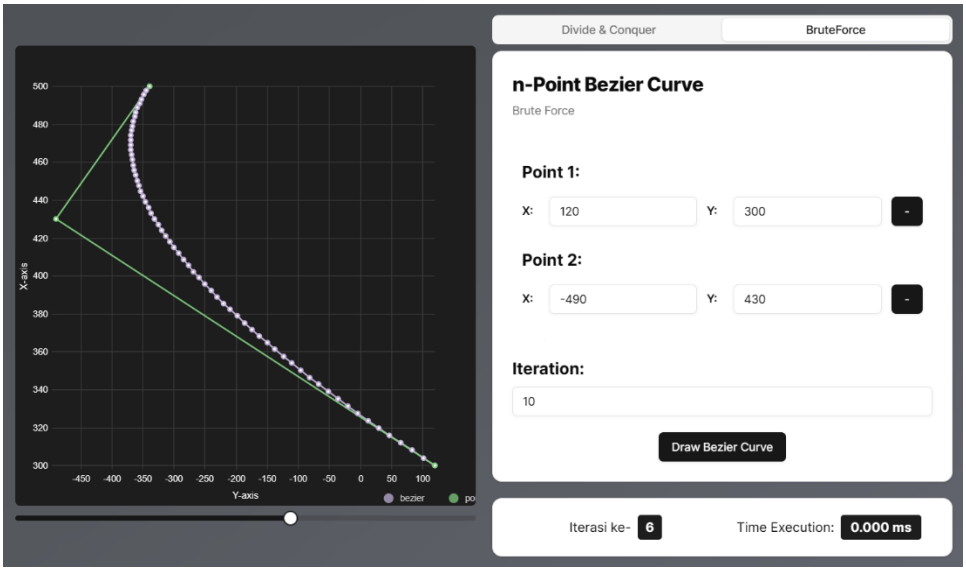
Point 3: (-340,500)

Output DNC:



Real Time execution: 0.0419921875 ms

Output Bruteforce:



Real Time Execution: 0.0390625 ms

3.5 Contoh 5

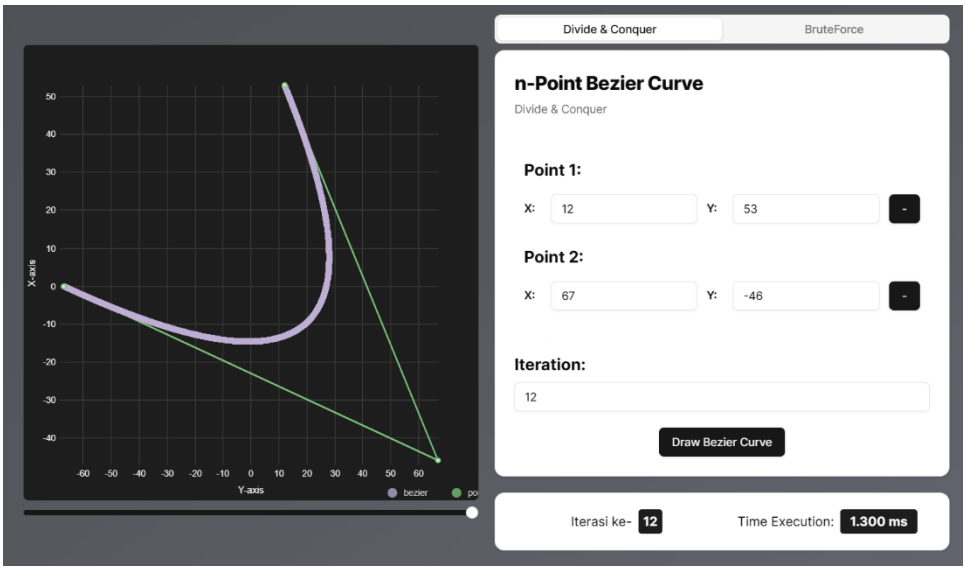
Input:

Point 1: (12,53)

Point 2: (67,-46)

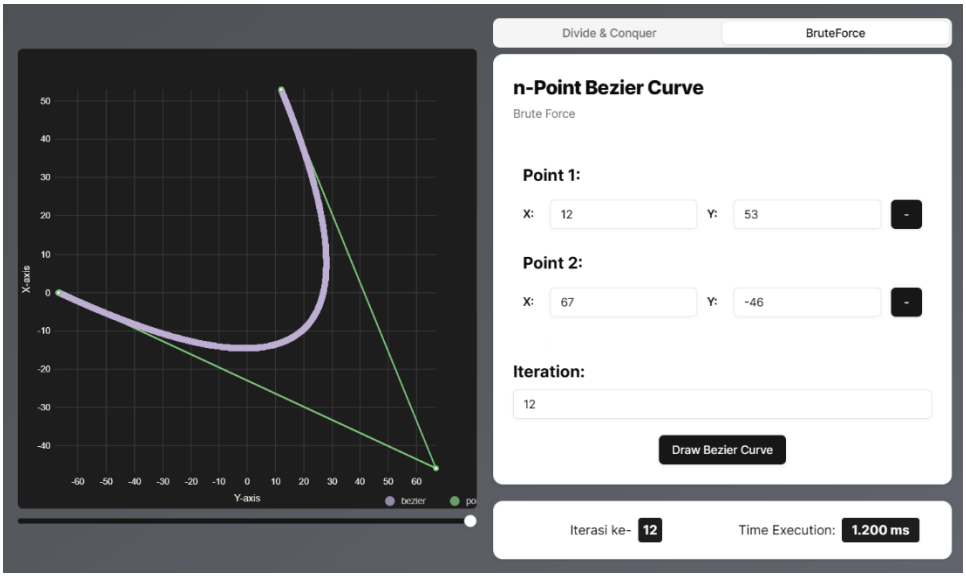
Point 3: (-67,0)

Output DNC:



Real Time execution: 1.18603515625 ms

Output Bruteforce:



Real Time Execution: 1.156982421875 ms

3.6 Contoh 6

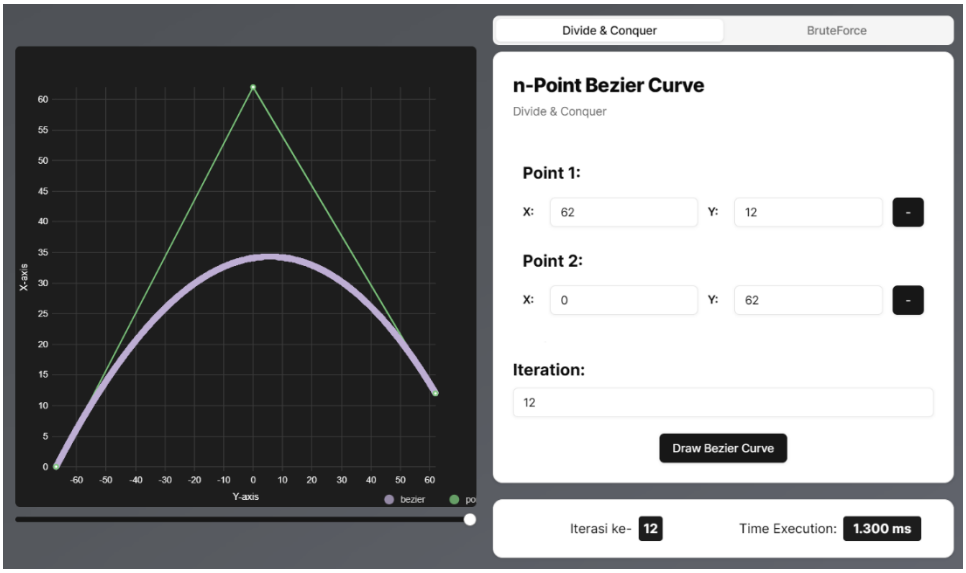
Input:

Point 1: (62,12)

Point 2: (0,62)

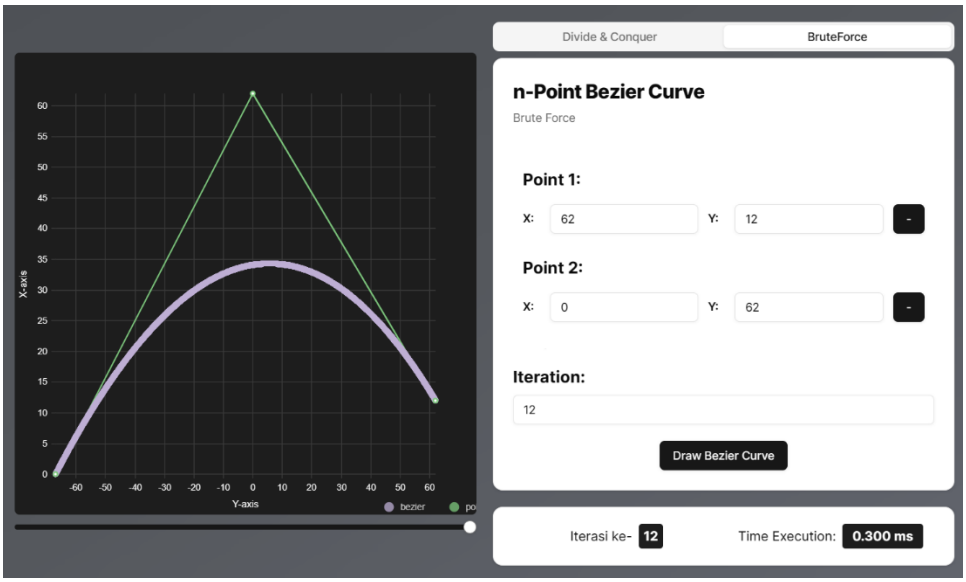
Point 3: (-67,0)

Output DNC:



Real Time execution: 1.337890625 ms

Output Bruteforce:



Real Time Execution: 0.37890625 ms

3.7 Contoh Bonus 1

Input:

Point 1: (62,12)

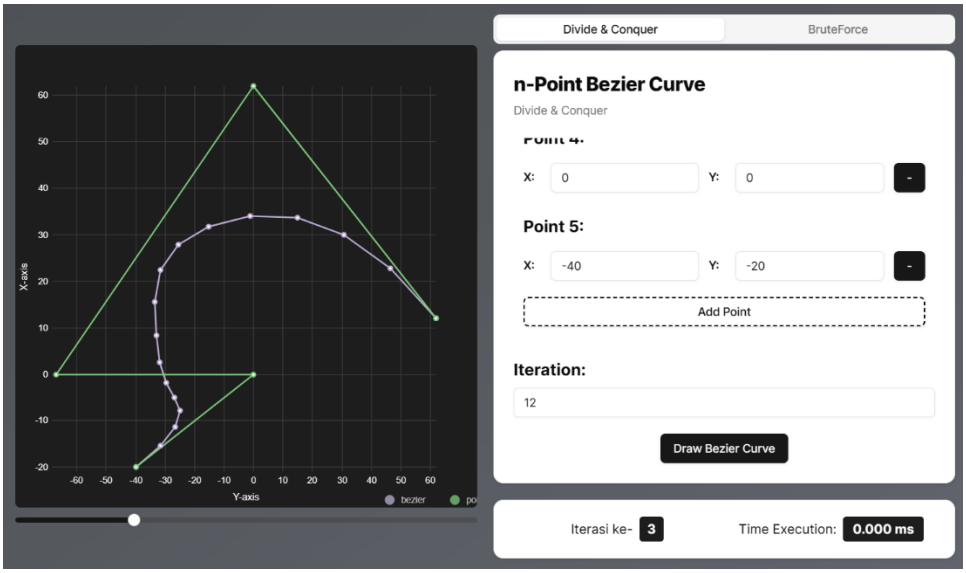
Point 2: (0,62)

Point 3: (-67,0)

Point 4: (0,0)

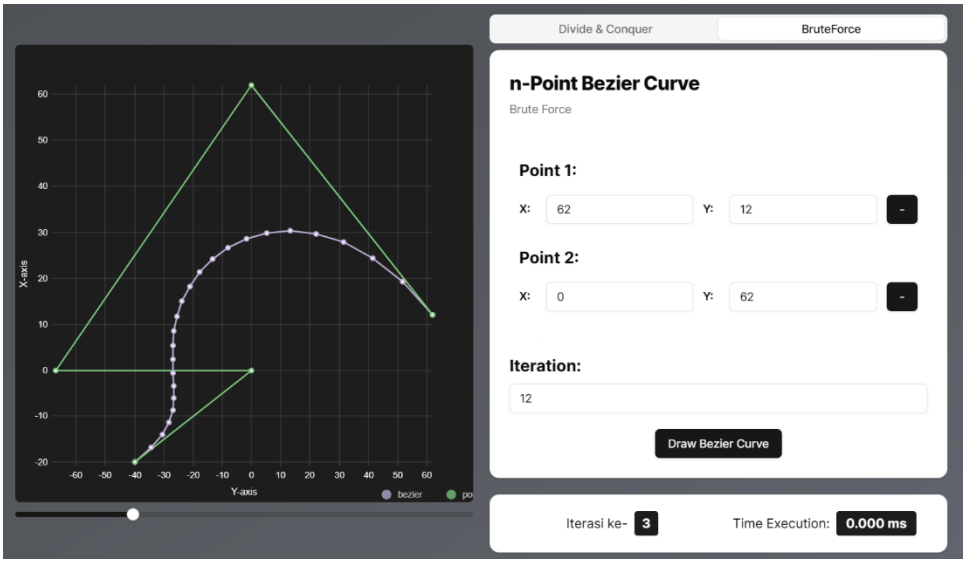
Point 5: (-40,-20)

Output DNC:



Real Time execution: 0.010986328125 ms

Output Bruteforce:



Real Time Execution: 0.01806640625 ms

3.8 Contoh Bonus 2

Input:

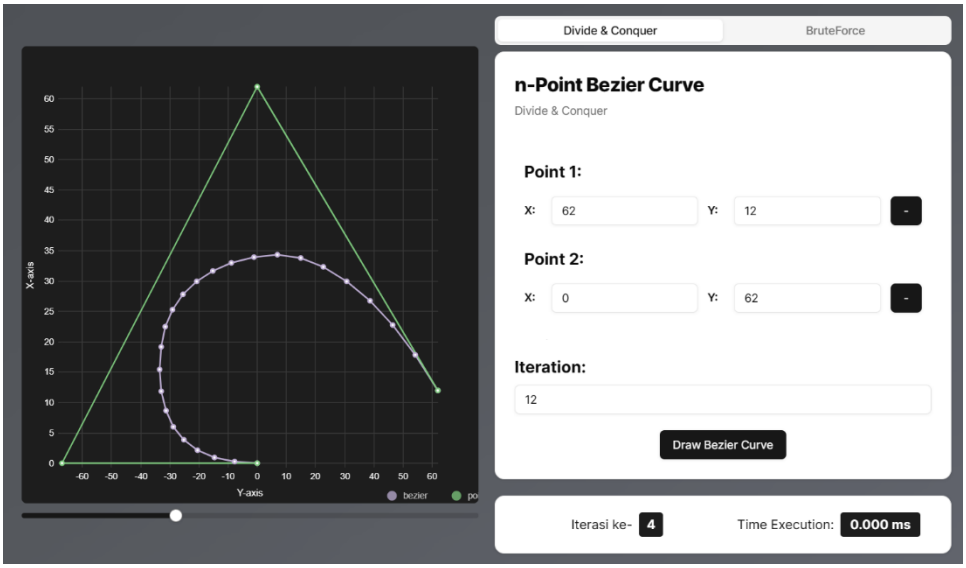
Point 1: (62,12)

Point 2: (0,62)

Point 3: (-67,0)

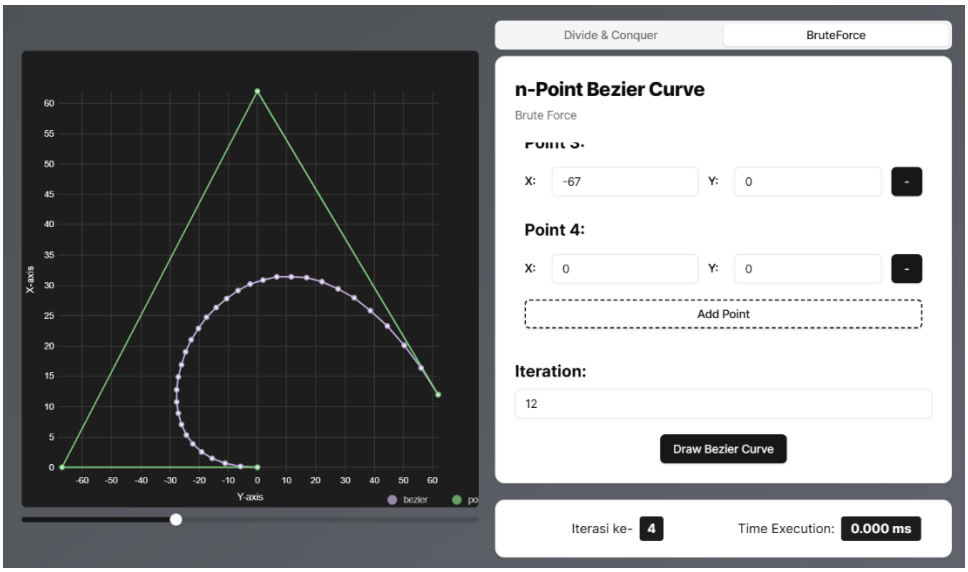
Point 4: (0,0)

Output DNC:



Real Time execution: 0.01416015625 ms

Output Bruteforce:



Real Time Execution: 0.02197265625 ms

BAB IV
Hasil Analisis

Dalam proyek ini, terdapat dua pendekatan yang digunakan untuk menghitung titik-titik kurva Bézier, yaitu brute force dan *divide and conquer*. Meskipun kedua pendekatan tersebut menghasilkan output yang sama, namun kompleksitas algoritma dan efisiensi keduanya berbeda.

Pendekatan brute force menggunakan rumus binomial yang melibatkan perhitungan titik-titik kurva secara langsung tanpa melakukan iterasi yang banyak. Dalam implementasinya, pendekatan ini menggunakan fungsi `drawBezierCurveBruteForce` yang memanfaatkan rumus binomial untuk menghitung setiap titik kurva. Kompleksitas waktu dari pendekatan ini adalah O(n * m), di mana n adalah jumlah titik kontrol dan m adalah jumlah titik yang dihitung pada kurva. Meskipun kompleksitas waktunya lebih tinggi secara teoritis, pendekatan ini cenderung lebih efisien untuk kasus-kasus dengan jumlah titik kontrol yang banyak dan jumlah titik yang dihitung pada kurva tidak terlalu sedikit.

Di sisi lain, pendekatan *divide and conquer* menggunakan algoritma rekursif yang membagi masalah menjadi sub-masalah yang lebih kecil, menyelesaikan sub-masalah tersebut, dan kemudian menggabungkan solusinya untuk mendapatkan solusi akhir. Dalam implementasinya, pendekatan ini menggunakan fungsi `drawBezierCurve` yang melakukan iterasi berdasarkan jumlah titik kontrol. Kompleksitas waktu dari pendekatan ini adalah O(n log n), di mana n adalah jumlah titik kontrol. Meskipun kompleksitas waktunya lebih baik secara teoritis, pendekatan ini melibatkan operasi array yang lebih banyak, seperti pembuatan array baru, penyisipan, dan penggabungan, yang dapat menjadi *overhead* yang signifikan untuk kasus-kasus dengan jumlah titik kontrol yang banyak.

Berdasarkan proyek ini dan program yang telah dibuat, pendekatan brute force lebih efisien untuk kasus-kasus dengan jumlah titik kontrol yang besar, sedangkan pendekatan divide and conquer menjadi lebih efisien saat jumlah titik kontrol semakin kecil, seperti yang sudah dibahas pada paragraf sebelumnya. Namun, perlu diingat bahwa efisiensi juga bergantung pada implementasi dan optimasi yang dilakukan pada masing-masing algoritma. Dalam proyek ini, kedua pendekatan tersebut diimplementasikan secara terpisah, sehingga memungkinkan untuk membandingkan kinerja dan efisiensi keduanya secara langsung.

Hal penting yang perlu diperhatikan juga, *time execution* pada website tidak bisa dijadikan parameter untuk menghitung *real processing time* dari tiap algoritma, karena pada program yang telah dibuat, *time execution* sangat bergantung terhadap *performance* website saat dijalankan. Apabila *performance* website sedang bagus, *time execution* akan menunjukan hasil yang tepat sesuai dengan *processing time* tiap algoritma, namun ketika *performance* sedang buruk, *time execution* akan menghitung variabel-variabel selain *processing time* sebagai waktu eksekusi, seperti *rendering time, event loop* karena javaScript menggunakan single-threaded

yang artinya ketika fungsi algoritma tidak selalu langsung berjalan ketika *execution time* dimulai, melainkan akan menjalankan proses lain terlebih dahulu yang ada pada queue. Ini membuat hasil *time execution* cenderung lebih lama pada website dibanding dengan waktu sebenarnya untuk melakukan kalkulasi pada fungsi algoritma tersebut. Maka dari itu, jika ingin melihat *real processing time* dari tiap algoritma, dapat melihat melalui konsol pada website.

BAB V
Bonus

4.1 Bonus Implementasi Algoritma Divide and Conquer dengan n buah titik

Implementasi algoritma *divide and conquer* dengan n buah titik telah dibahas pada bab 1.1.2.

4.2 Bonus Tampilan Proses Iterasi Pembentukan Kurva Bezier

Tampilan proses iterasi pembentukan kurva bezier yang telah kami buat, memanfaatkan *library* tambahan dalam kerangka kerja *Next.js* bernama *Nivo*. Tampilan yang kami buat mampu menampilkan proses pembentukan kurva bezier dengan algoritma *divide and conquer* secara bertahap, yakni ketika pembentukan kurva bezier pada iterasi pertama dan dilanjutkan dengan menampilkan kurva bezier pada iterasi ke-n yang dapat disesuaikan dengan keinginan pengguna.

BAB VI
Lampiran

Pranala repository : https://github.com/akmalrmn/Tucil2_13522161_13522164

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bezier.	✓	
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.	✓	