LAPORAN TUGAS KECIL 2

IF2211 Strategi Algoritma

Membangun Kurva Bezier dengan Algoritma Titik Tengah Berbasis *Divide and Conquer*

Disusun oleh:

Dhidit Abdi Aziz (13522040)

Nyoman Ganadipa Narayana (13522066)

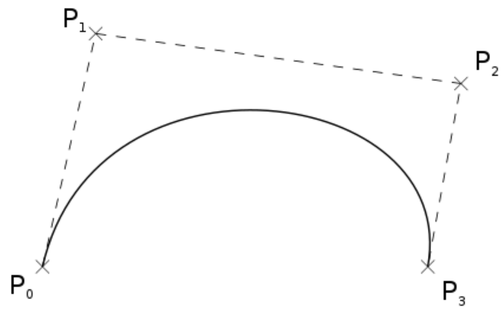
Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

1 Deskripsi Permasalahan



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 P0 sampai Pn , 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 P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 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 P0 dan P1 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 Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1. Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P0 dan P2. Berikut adalah uraian persamaannya.

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

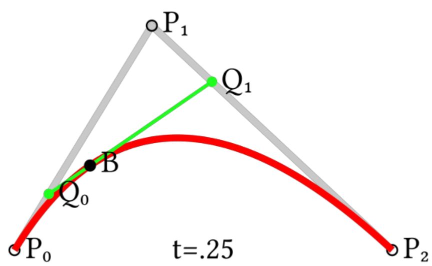
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

dengan melakukan substitusi nilai Q0 dan Q1, maka diperoleh persamaan sebagai berikut.

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

Berikut ini adalah ilustrasi dari proses di atas



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.

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

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

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 kami mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

2 Analisis dan Implementasi dalam Algoritma Brute Force

Langkah-langkah:

1. Dari tiga titik awal, didefinisikan titik pertama (P_0) terdiri atas (x_0, y_0), titik kedua (P_1) terdiri atas (x_1, y_1), dan titik ketiga (P_2) terdiri atas (x_2, y_2).
2. Kemudian, perlu dihitung terlebih dahulu berapa titik yang akan dihasilkan pada iterasi-n. Mari kita tinjau bersama-sama bagaimana polanya:
   a. Pada iterasi ke-1, dari yang awalnya tiga titik tetap menjadi tiga titik
   b. Pada iterasi ke-2, dari tiga titik menjadi lima titik, yaitu $(3 + 2^1)$
   c. Pada iterasi ke-3, dari lima titik menjadi sembilan titik, yaitu $(3 + 2^1 + 2^2)$
   d. Pada iterasi ke-4, dari sembilan titik menjadi tujuh belas titik, yaitu $(3 + 2^1 + 2^2 + 2^3)$

   Misalkan banyaknya titik adalah m. Dapat disimpulkan bahwa pada iterasi ke-n (berlaku sejak n=2), akan terdapat m sebesar $(3 + \sum_{i=1}^{n} 2^{n-1})$

2. Setelah diketahui akan ada berapa titik pada suatu iterasi, dicari berapa jarak antar tiap titik pada kurva tersebut (skala nol sampai satu). Misalkan jarak antar tiap titik adalah k, maka k = 1/(m-1)
3. Titik kontrol awal dan kontrol akhir pasti akan muncul kembali di akhir proses, sehingga kita cukup mencari titik titik yang berada di antara keduanya melalui iterasi sebanyak (m-2) kali.

   Misalkan titik ke-n terdiri atas (x_n, y_n). Pada tiap iterasinya, akan digunakan rumus bezier curve untuk mencari x_n dan y_n, yaitu:

   $$x_n = (1 - t)^2 x_0 + (1 - t)x_1 + t^2 x_2$$

   $$y_n = (1 - t)^2 y_0 + (1 - t)y_1 + t^2 y_2$$

   Dengan t adalah jarak dari titik ke-n ke titik awal, yaitu t = n*k

4. Setelah didapatkan tiap titik, maka dihubungkan satu garis lurus yang membentuk kurva

3 Analisis dan Implementasi dalam Algoritma Divide and Conquer

3.1 Quadratic Bezier Curve

Langkah-langkah:

1. Diberikan 3 control points, katakanlah point-point tersebut adalah p0, p1, dan p2 secara berurutan. dan sebuah angka jumlah iterasi, katakan K (K>=0).
2. (*EDGE CASE*) Jika K bernilai 0, kembalikan points semula yang diberikan.
3. Dibuat middle point dari segment P0P1 katakan Q0, dibuat middle point dari segment P1P2 katakan Q1, Sehingga terakhir dapat dibuat middle point dari segment Q0Q1 katakan R0
4. (*BASE CASE*) Sekarang, jika K bernilai 1, maka kita selesai, yaitu cukup kembalikan array yang memuat Q0, R0, dan Q1.
5. (*RECURSIVE CASE*) Jika K > 1, maka lakukan pembagian area conquering, yaitu titik-titik pada first half dan second half. Untuk first half, conquer titik-titik P0, Q0, R0 (menjadikannya P0, P1, dan P2 pada iterasi selanjutnya) dengan jumlah iterasi sebanyak K - 1. Sementara untuk second half, lakukan conquer titik-titik R0, Q1, P2 (menjadikannya P0, P1, dan P2 pada iterasi selanjutnya) dengan jumlah iterasi sebanyak K - 1.
6. Setelah meng-conquer titik-titik pada first half dan second half, lakukan penggabungan hasil conquer titik-titik tersebut.

3.2 N Control points Bezier Curve

Langkah-langkah:

1. Diberikan N control points katakanlah point-point tersebut adalah p0, p1, ..., pN-1 secara berurutan. dan sebuah angka jumlah iterasi, katakan K (K >= 0)(EDGE CASE) jika K bernilai 0, kembalikan points semula yang diberikan.
2. Buat dua buah array kosong: first dan second, untuk menyimpan titik-titik yang akan kita conquer pada iterasi selanjutnya
3. Lakukan perulangan yang dimulai pada titik-titik p0, p1, ..., pN
    a. Jika titik-titik yang diberikan berjumlah 1, maka lompat ke langkah nomor 3
    b. Diberikan titik-titik, katakanlah T0,.., Tc.
    c. Untuk setiap i yang mungkin, buatlah titik Mi yang merupakan middle point dari segment TiTi+1
    d. Dari langkah 3.b akan didapat c - 1 titik baru hasil konstruksi tersebut.
    e. Copy M0, masukkan ke array first. Kemudian copy M(c-1), masukkan ke array second.

    f. Lakukanlah kembali perulangan dengan titik-titik tersebut adalah M0, ... Mc-1.

4. Sekarang kita punya 1 titik, katakan M.
5. (*BASE CASE*) Jika K = 1, maka kita sudah selesai, kembalikan array yang memuat P0, M, P(N-1)
6. (*RECURSIVE CASE*) Jika K > 1, lakukan conquer terhadap titik-titik yang disimpan pada array first, dan lakukan juga conquer terhadap titik-titik yang disimpan array second namun urutannya dibalik.
7. Setelah melakukan conquer pada array first dan reversed second, lakukan penggabungan hasil conquer titik-titik tersebut.

4 Source Code

4.1 types.ts

Berisikan tipe struktur data yang kami gunakan, yaitu Points (titik), dan Segment (dua titik). Segment digunakan untuk memudahkan pencarian middle point dari dua titik.

```
export type Point = {
  x: number;
  y: number;
};

export type Segment = {
  start: Point;
  end: Point;
};
```

4.2 BezierCurve.ts

Terdapat beberapa tipe struktur data tambahan yang kami gunakan, yaitu QuadraticBezierCurveParams, yang berisikan tipe inputan metode perhitungan kurva yang diinginkan pengguna. Selain itu juga terdapat beberapa fungsi utama, yaitu:

a. Fungsi QuadraticBezierCurve( ) berfungsi menerima inputan dan mengirimkan inputan tersebut ke fungsi berdasarkan kategorinya, apakah divide and conquer atau bruteforce.
b. Fungsi QuadraticBezierCurveDnC( ) berfungsi untuk menjalankan divide and conquer untuk kurva kuadratic
c. Fungsi BezierCurve( ), berfungsi untuk menjalankan divide and conquer untuk n-titik

```
import { Point, Segment } from "../types";
import { QuadraticBezierBruteForce } from "./BezierBruteForce";

function middlePoint(s: Segment): Point {
  const xp = (s.start.x + s.end.x) / 2;
  const yp = (s.start.y + s.end.y) / 2;
  return {
    x: xp,
    y: yp,
  };
}
```

```
type QuadraticBezierCurveParams = {
  points: Point[];
  iteration: number;
} & (
  | {
      type: "DnC";
    }
  | {
      type: "Bruteforce";
    }
);

export function QuadraticBezierCurve(params: QuadraticBezierCurveParams):
{
  points: Point[];
  duration: number;
} {
  const { points, iteration, type } = params;
  let result;
  const startTime = performance.now();

  if (iteration === 0) {
    return {
      points,
      duration: performance.now() - startTime,
    };
  }

  if (type === "DnC") {
    result = QuadraticBezierDnC(points, iteration);
  } else {
    result = QuadraticBezierBruteForce(points, iteration);
  }

  const endTime = performance.now();

  return { points: result, duration: endTime - startTime };
}
function QuadraticBezierDnC(points: Point[], iteration: number): Point[] {
```

```
  if (iteration === 0) return points;
  const middle1 = middlePoint({ start: points[0], end: points[1] });
  const middle2 = middlePoint({ start: points[1], end: points[2] });
  const middle = middlePoint({ start: middle1, end: middle2 });

  if (iteration <= 1) {
    return [points[0], middle, points[2]];
  }

  let first = QuadraticBezierDnC([points[0], middle1, middle], iteration -
1);
  let second = QuadraticBezierDnC([middle, middle2, points[2]], iteration
- 1);

  return [...first, ...second.slice(1)];
}

export function DnCBezierCurve(points: Point[], iteration: number) {
  const startTime = performance.now();

  const result = BezierCurve(points, iteration);
  const endTime = performance.now();
  return {
    points: result,
    duration: endTime - startTime,
  };
}

export function BezierCurve(points: Point[], iteration: number): Point[] {
  if (iteration === 0) return points;
  const degree = points.length;
  let middles: Point[][] = [];

  for (let i = 0; i < degree; i++) {
    middles.push([]);
    middles[0].push(points[i]);
  }

  for (let i = 1; i < degree; i++) {
    for (let j = 0; j < degree - i; j++) {
```

```
      middles[i].push(
        middlePoint({
          start: middles[i - 1][j],
          end: middles[i - 1][j + 1],
        })
      );
    }
  }

  let first: Point[] = [],
    second: Point[] = [];

  for (let i = 0; i < degree; i++) {
    first.push(middles[i][0]);
  }

  for (let i = degree - 1; i >= 0; i--) {
    second.push(middles[i].at(-1) as Point);
  }

  if (iteration === 1) {
    return [points[0], middles[degree - 1][0], points[degree - 1]];
  } else {
    return BezierCurve(first, iteration - 1).concat(
      BezierCurve(second, iteration - 1).slice(1)
    );
  }
}
```

4.2 BezierCurveBruteforce.ts

QuadraticBezierBruteForce( ) merupakan fungsi utama yang menerima inputan list of titik dan jumlah iterasinya, kemudian mengeluarkan list of titik pada iterasi tersebut. Terdapat juga fungsi pembantu yaitu countNumSteps() untuk menghitung berapa titik yang akan dihasilkan pada suatu iterasi

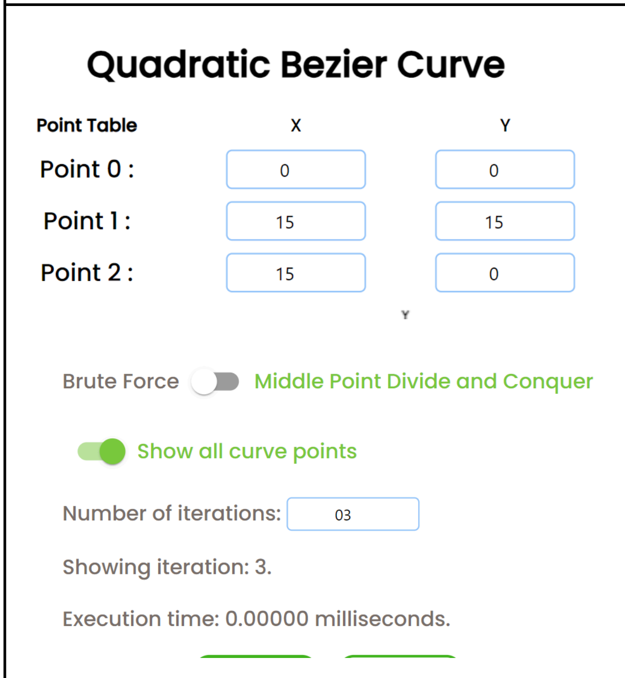
```
import { Point } from "../types";

export function QuadraticBezierBruteForce(
  points: Point[],
```

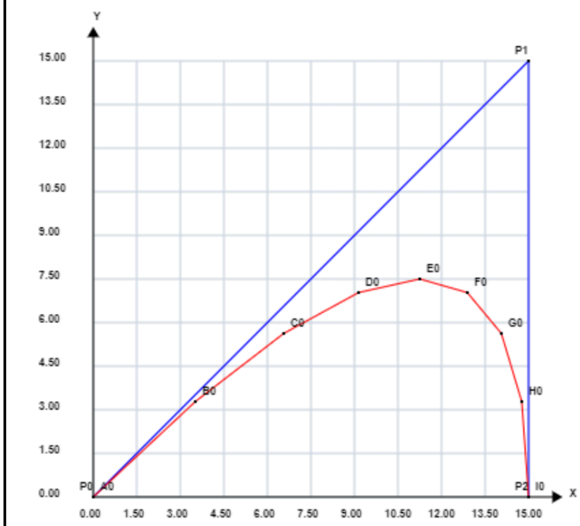
```
  iteration: number
): Point[] {
  let numSteps: number = countNumSteps(3, iteration); //Menghitung
banyaknya (titik) pada suatu iterasi
  const stepSize: number = 1.0 / (numSteps - 1); //Menghitung jarak antar
titik pada suatu iterasi
  let returnArr: Point[] = [];
  returnArr.push(points[0]);
  for (let i = 1; i <= numSteps - 2; i++) {
    const t: number = stepSize * i;
    const xTemp: number =
      (1 - t) ** 2 * points[0].x +
      2 * (1 - t) * t * points[1].x +
      t ** 2 * points[2].x;
    const yTemp: number =
      (1 - t) ** 2 * points[0].y +
      2 * (1 - t) * t * points[1].y +
      t ** 2 * points[2].y;
    const pointTemp: Point = { x: xTemp, y: yTemp };
    returnArr.push(pointTemp);
  }
  returnArr.push(points[2]);
  return returnArr;
  //3
  //3 + 2
  //3 + 2 + 4
}

function countNumSteps(nBezier: number, iteration: number) {
  //Mencari berapa banyak titik pada iterasi ke-sekian
  let difference: number = nBezier - 1;
  let temp: number = nBezier;
  for (let i = 1; i < iteration; i++) {
    temp = temp + difference ** i;
  }
  return temp;
}
```

5 Testing

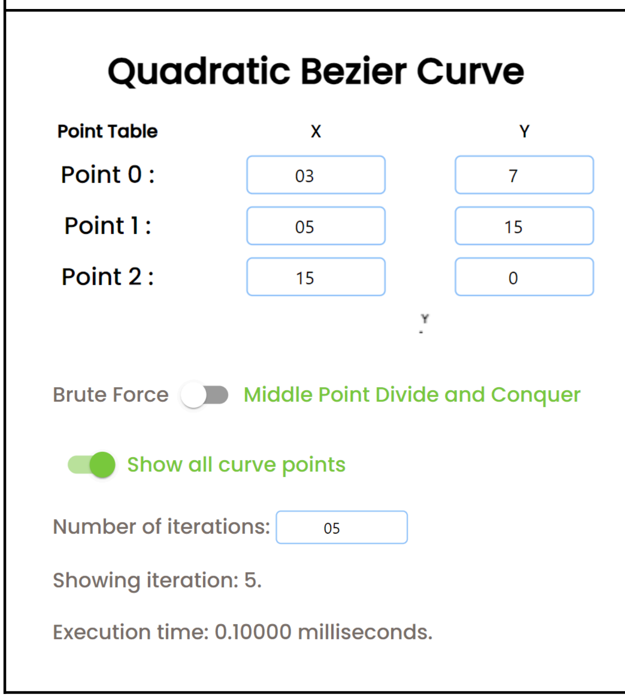
5.1 Testing Algoritma Brute Force

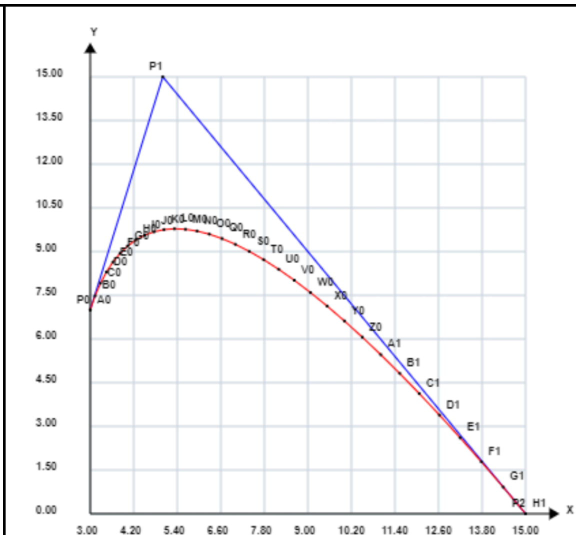
Testing ke-1

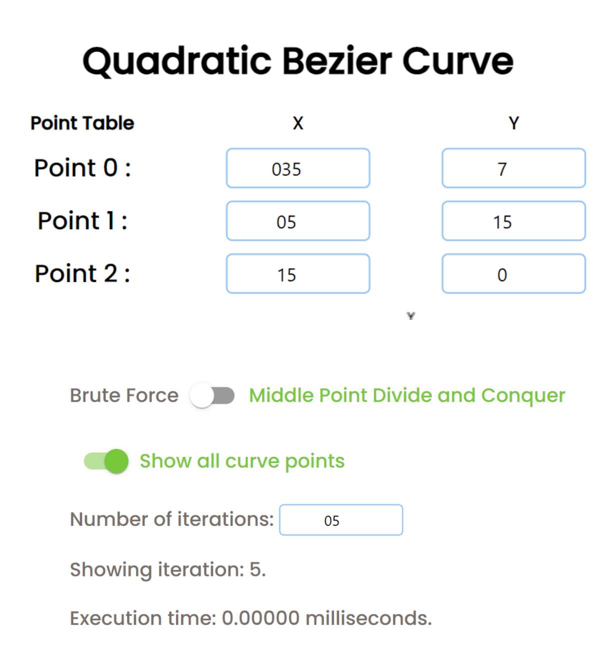
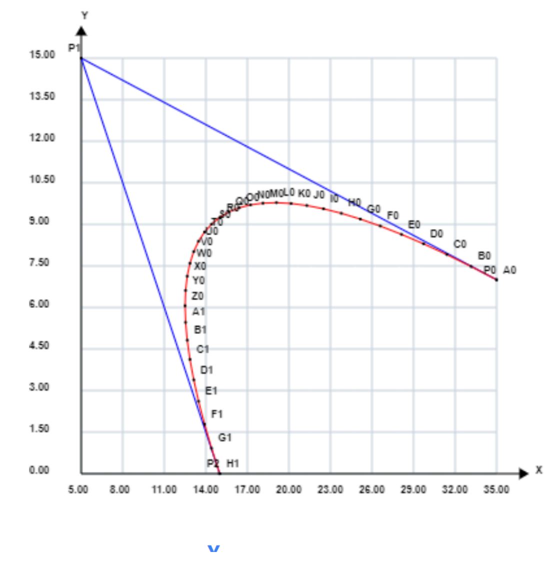
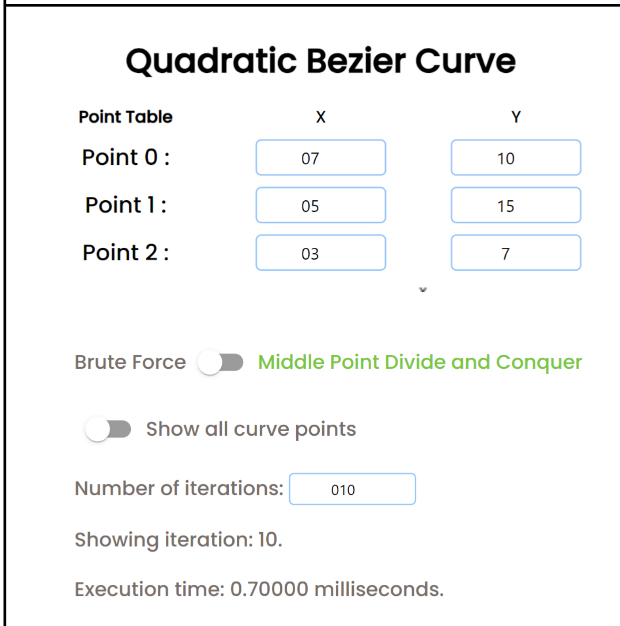
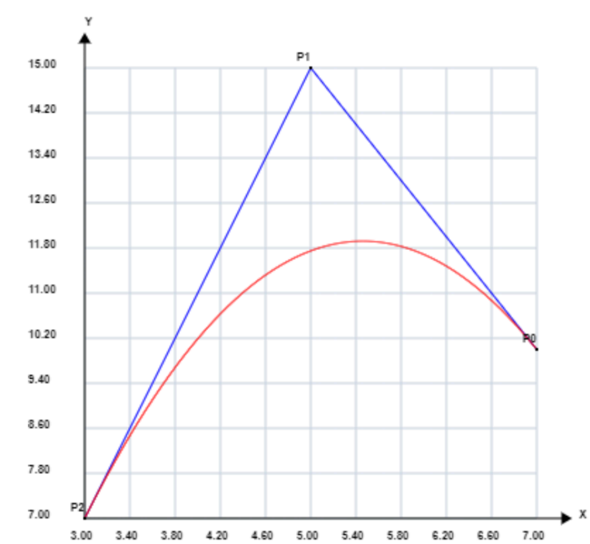


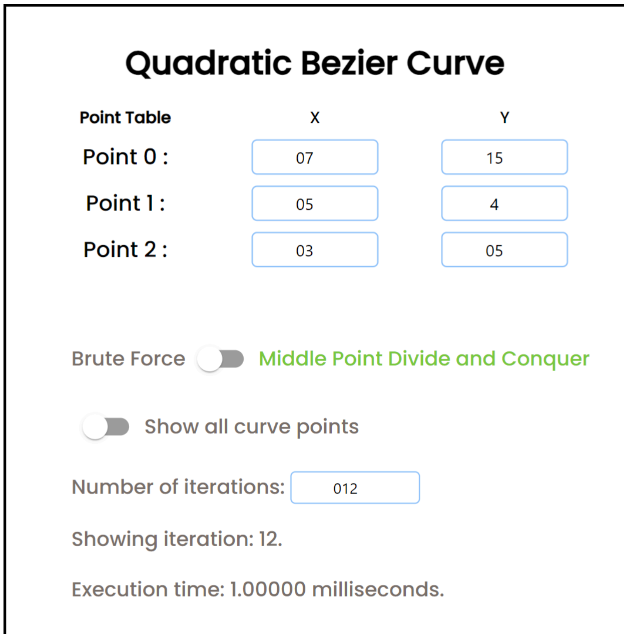


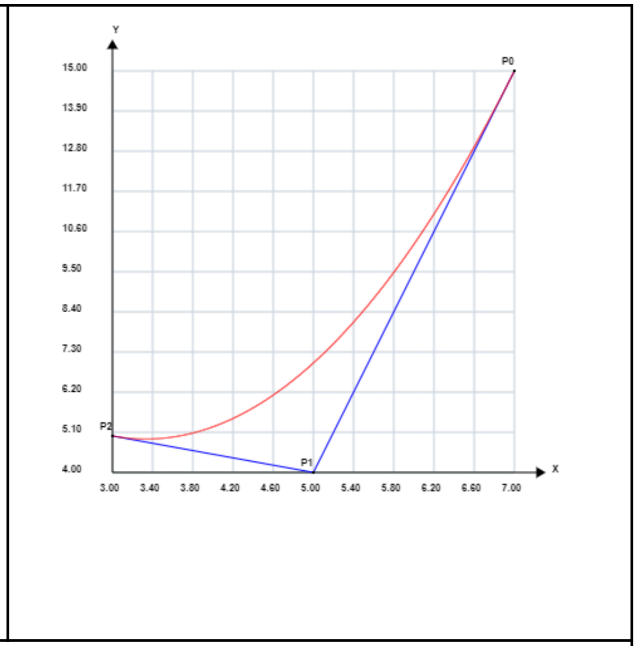
Testing ke-2



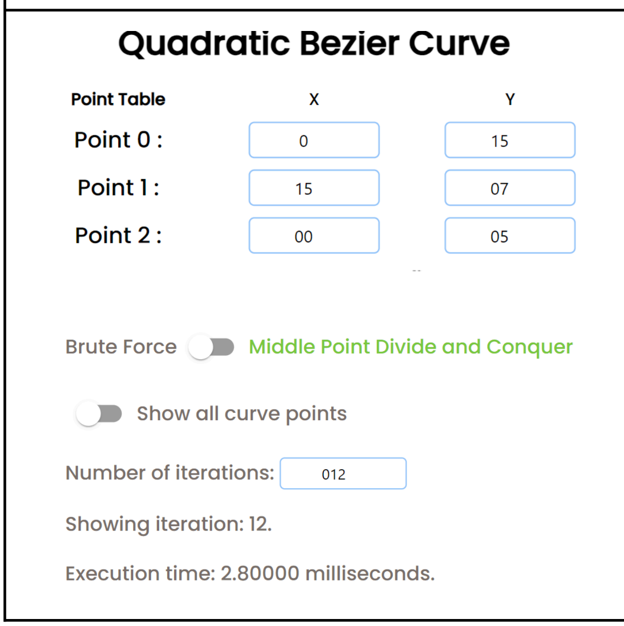


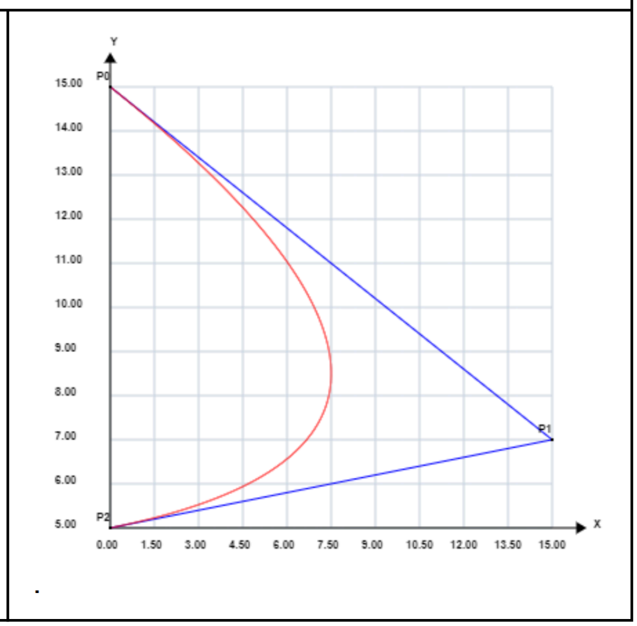
<table>
<tr><td colspan="2">Testing ke-3</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">Testing ke-4</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">Testing ke-5</td></tr>
</table>





Testing ke-6

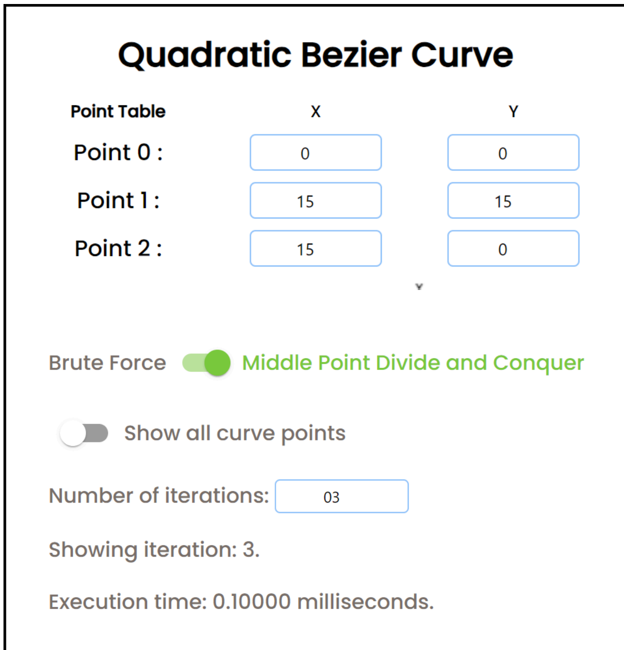


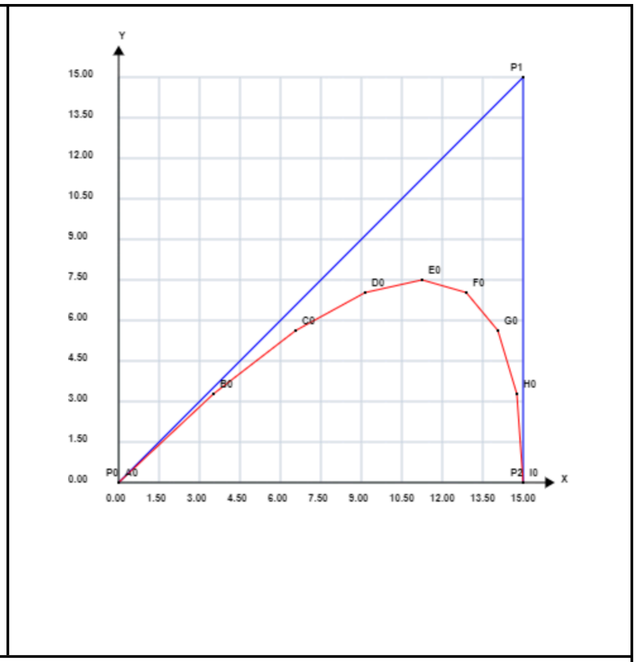


5.2 Testing Algoritma Divide and Conquer

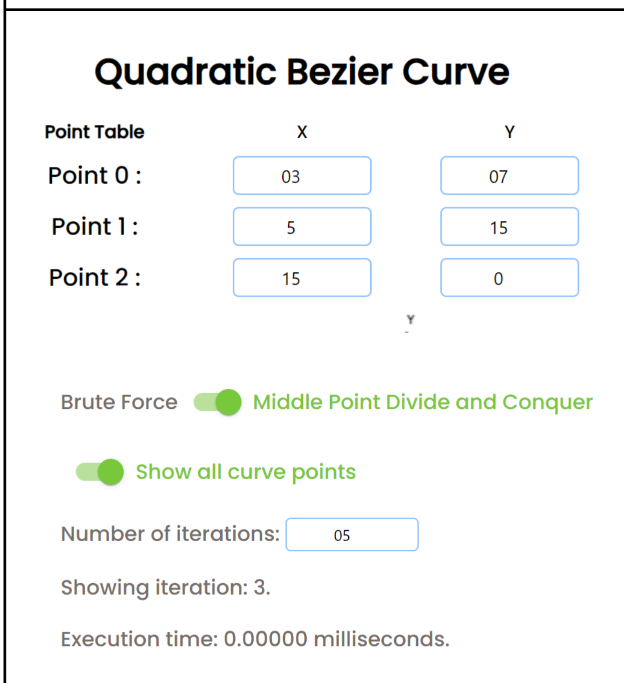
5.2.1 Quadratic Bezier Curve

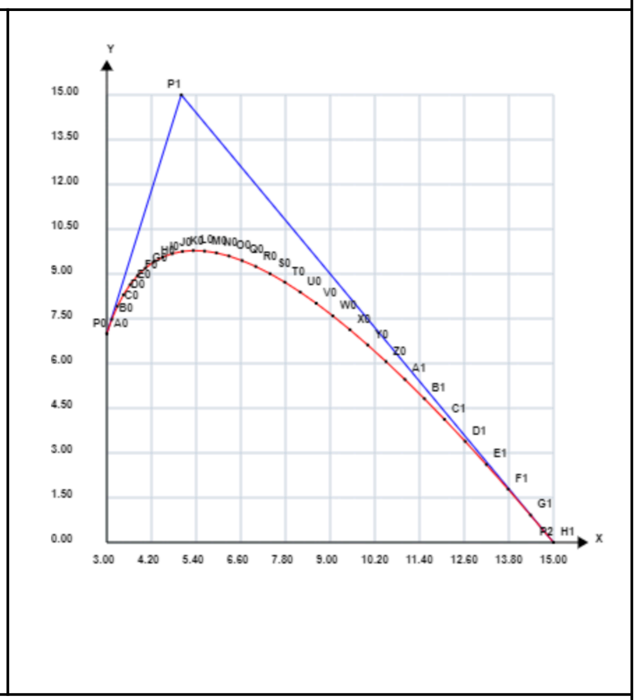
Testing ke-1



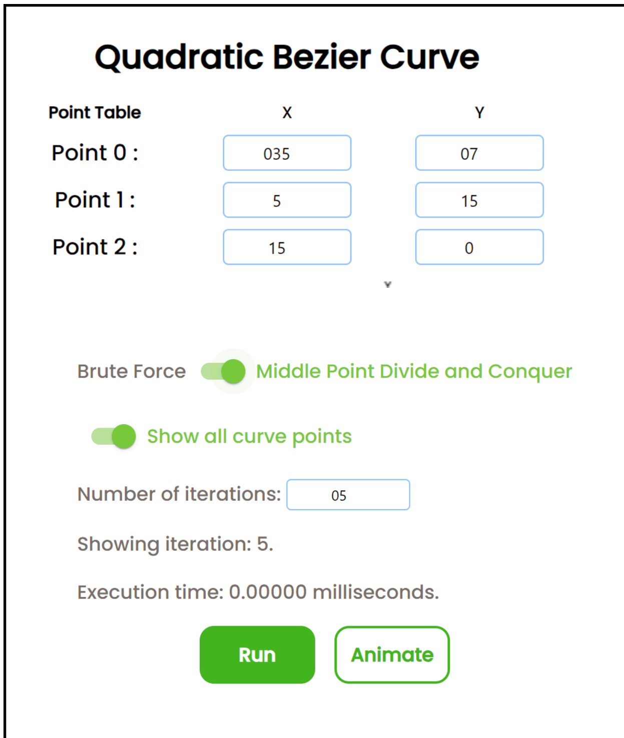


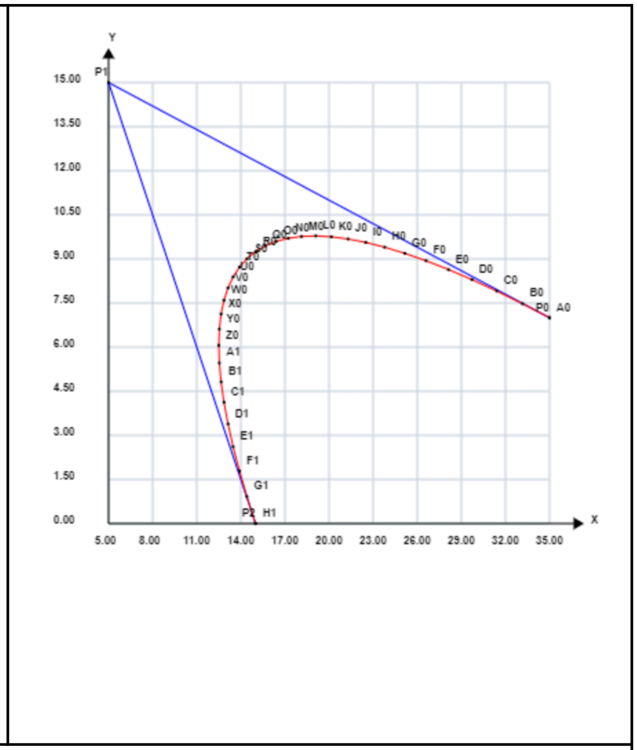
Testing ke-2



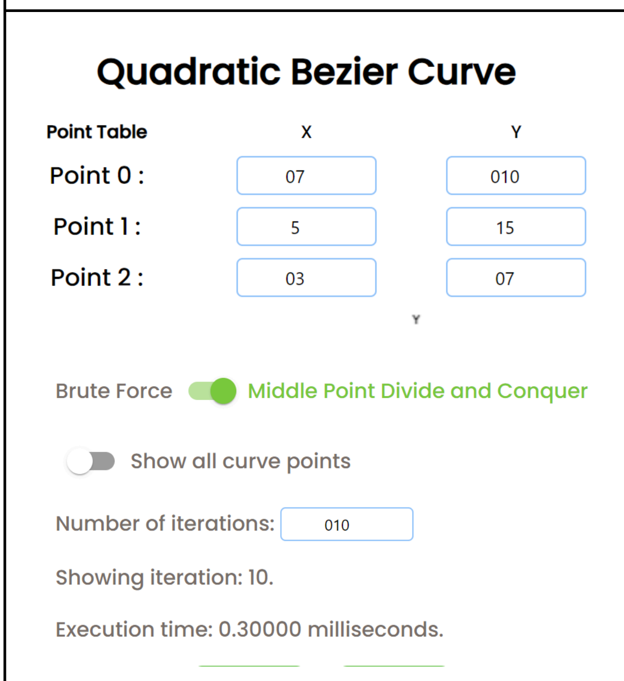


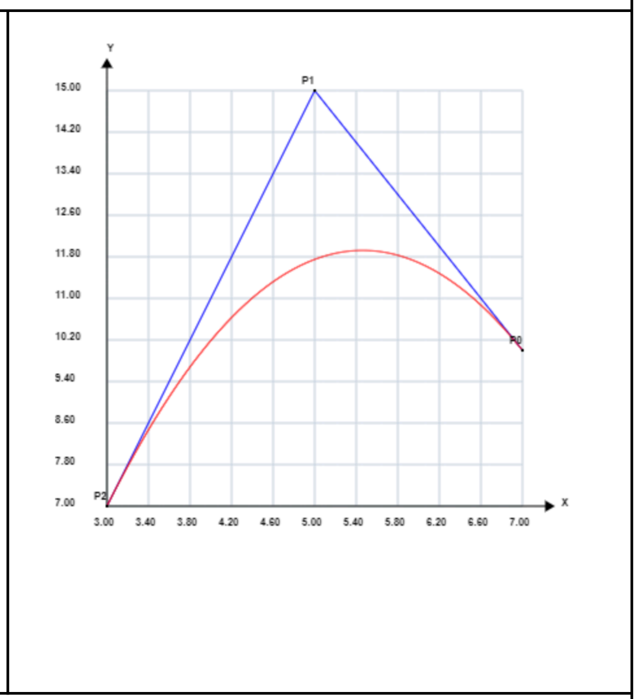
Testing ke-3



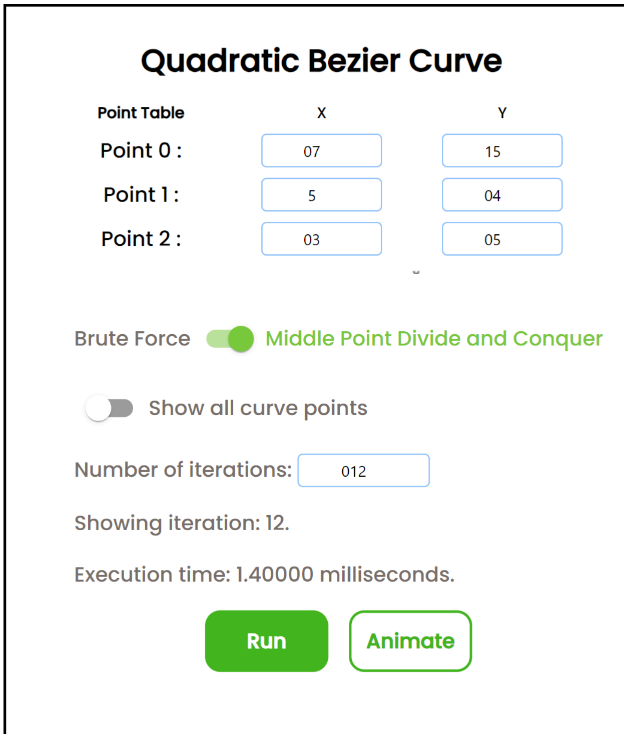


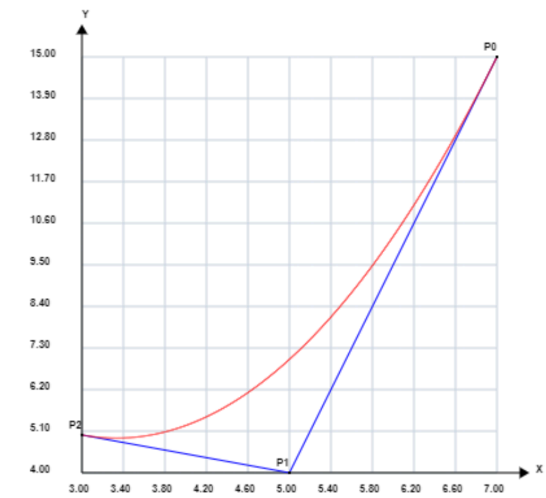
Testing ke-4



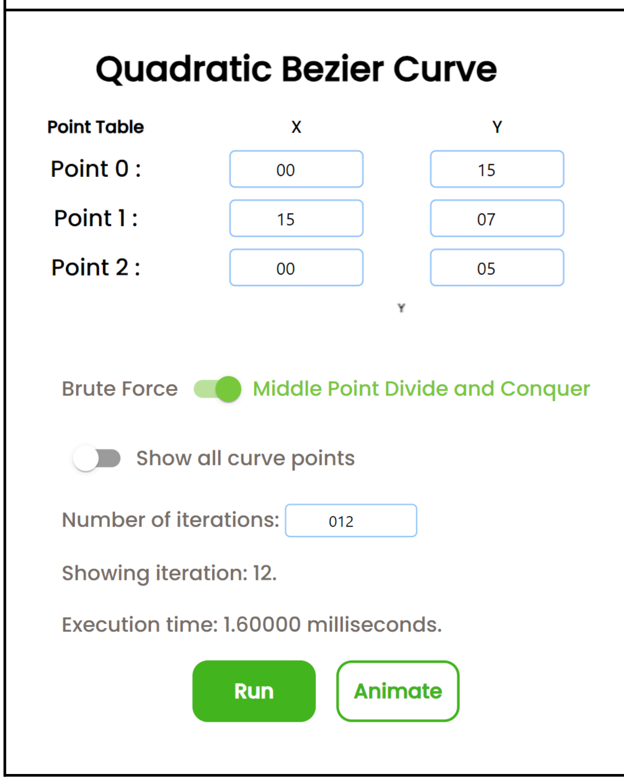


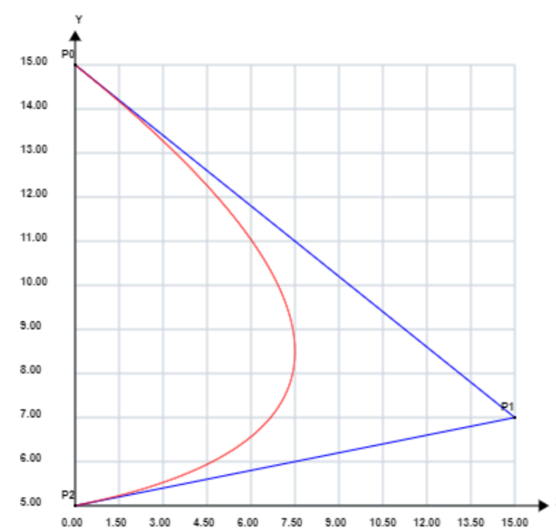
Testing ke-5


Quadratic Bezier Curve
Point Table
X
Y
Point 0 :
07
15
Point 1 :
5
04
Point 2 :
03
05
Brute Force
Middle Point Divide and Conquer
Show all curve points
Number of iterations:
012
Showing iteration: 12.
Execution time: 1.40000 milliseconds.
Run
Animate

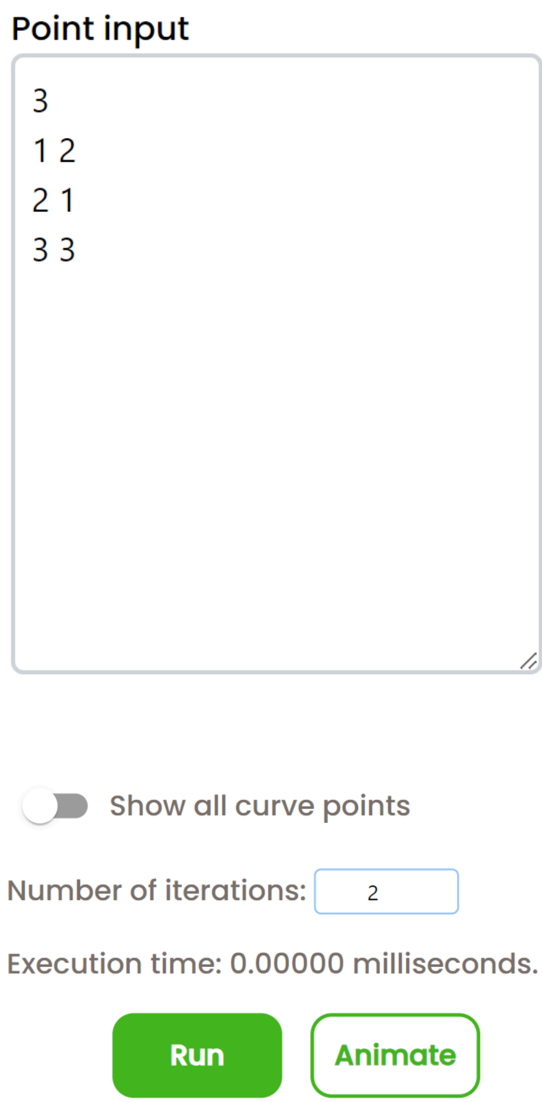
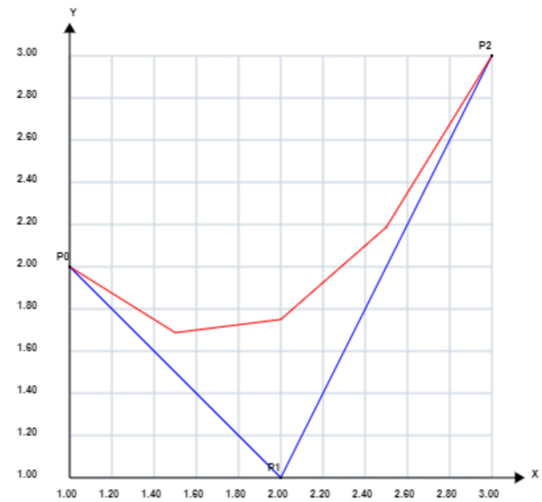

Y
X
P0
P1
P2
15.00
13.90
12.80
11.70
10.60
9.50
8.40
7.30
6.20
5.10
4.00
3.00
3.40
3.80
4.20
4.60
5.00
5.40
5.80
6.20
6.60
7.00

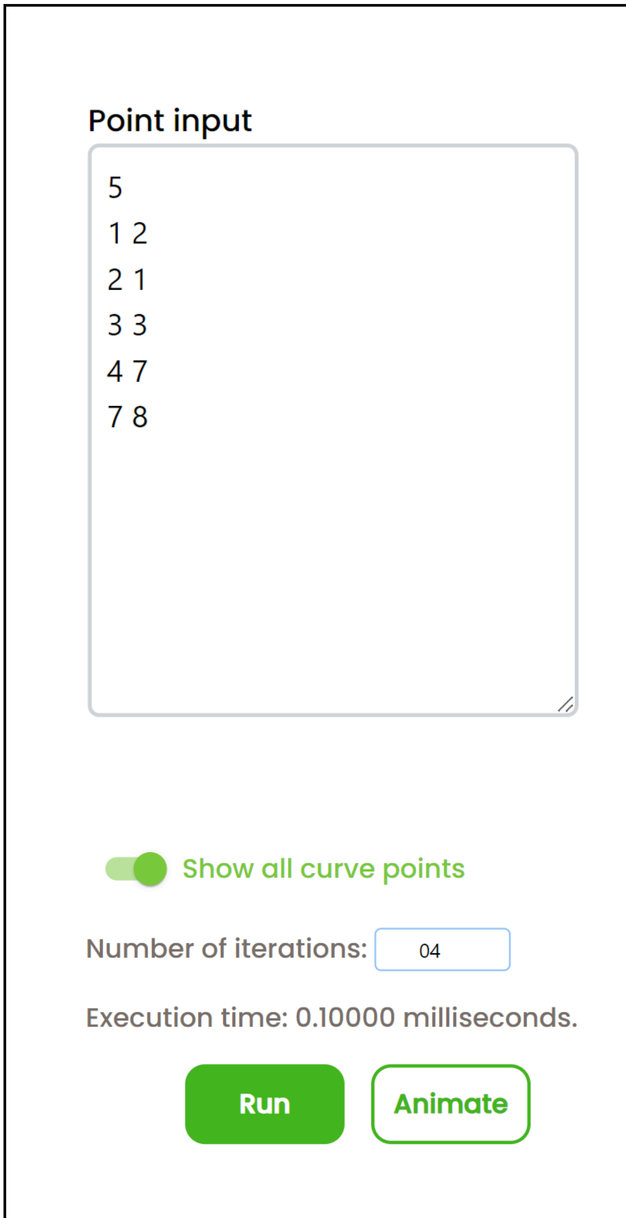
Testing ke-6

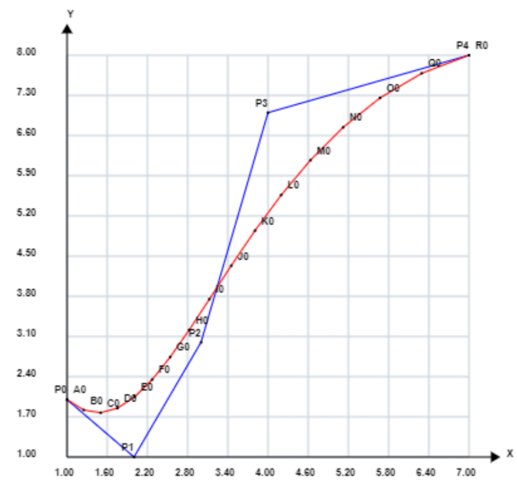

Quadratic Bezier Curve
Point Table
X
Y
Point 0 :
00
15
Point 1 :
15
07
Point 2 :
00
05
Brute Force
Middle Point Divide and Conquer
Show all curve points
Number of iterations:
012
Showing iteration: 12.
Execution time: 1.60000 milliseconds.
Run
Animate


Y
X
P0
P1
P2
15.00
14.00
13.00
12.00
11.00
10.00
9.00
8.00
7.00
6.00
5.00
0.00
1.50
3.00
4.50
6.00
7.50
9.00
10.50
12.00
13.50
15.00

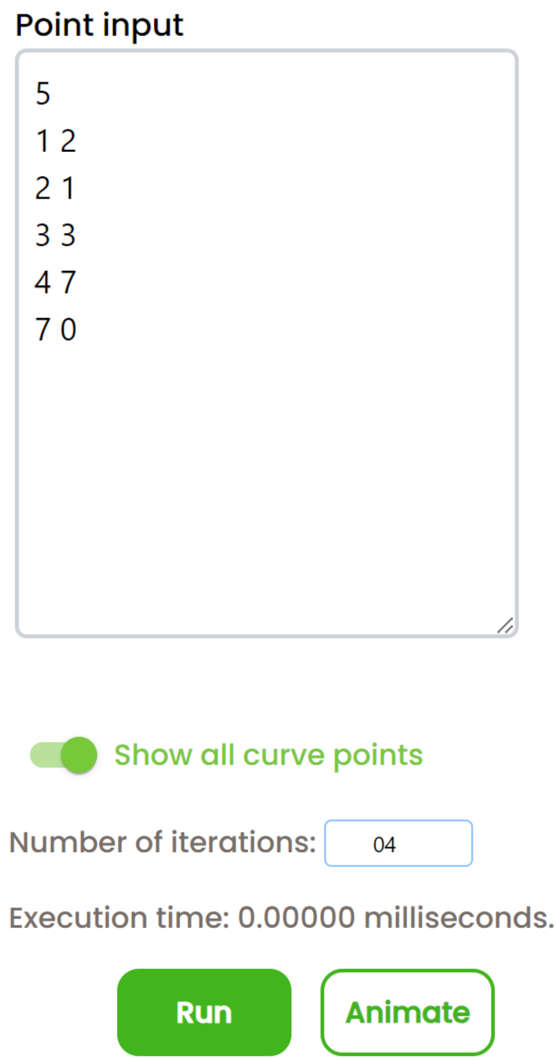
5.2.2 Generalized Bezier Curve

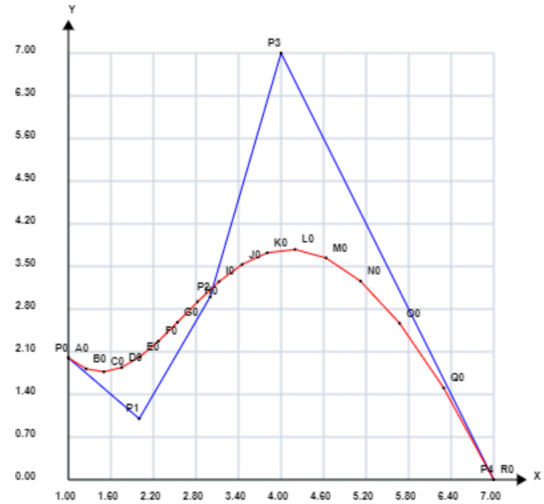
<table>
<tr><td colspan="2">Testing ke-1</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">Testing ke-2</td></tr>
</table>


Point input
5
1 2
2 1
3 3
4 7
7 8
Show all curve points
Number of iterations:
04
Execution time: 0.10000 milliseconds.
Run
Animate

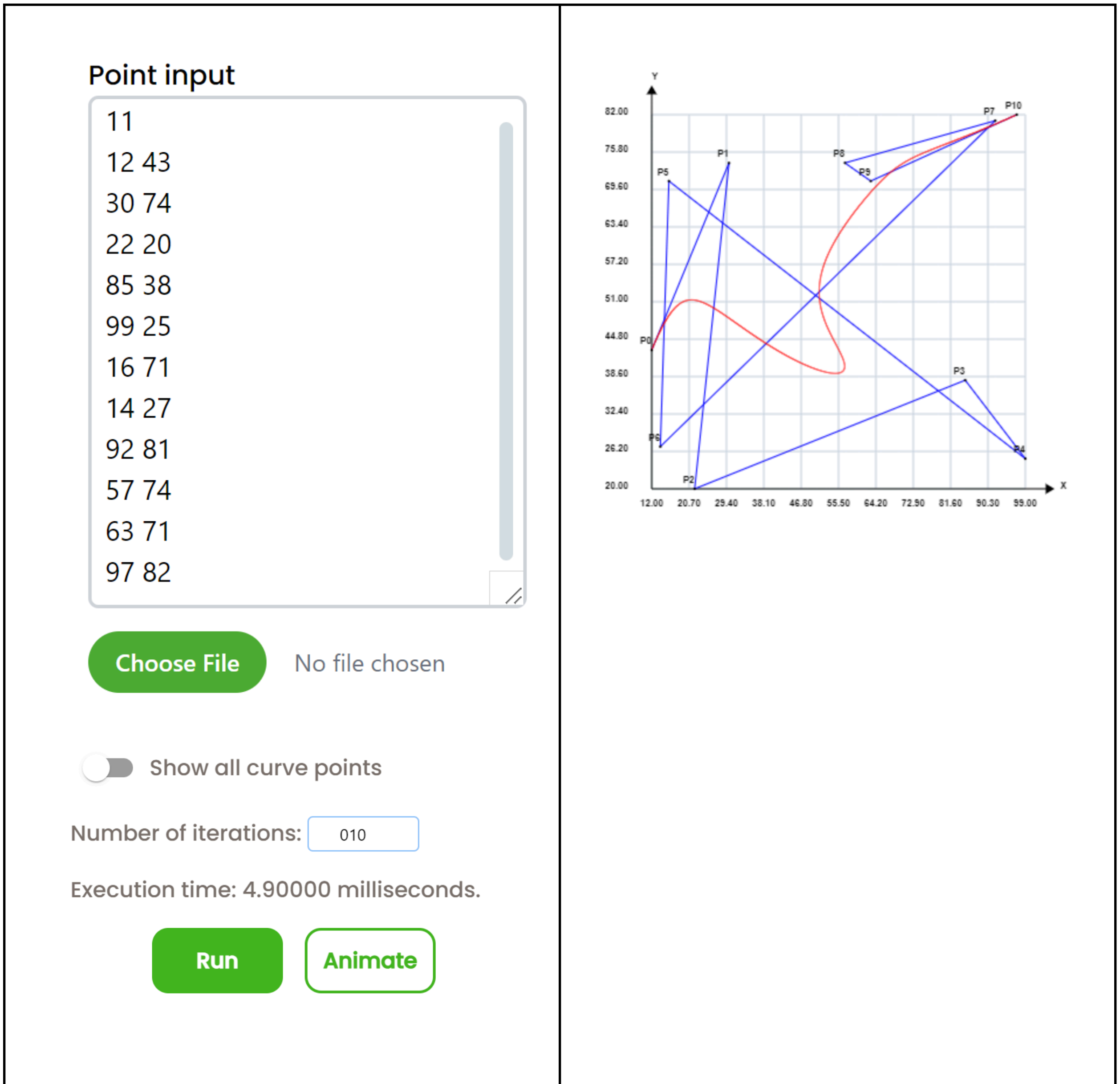

Y
X
8.00
7.30
6.60
5.90
5.20
4.50
3.80
3.10
2.40
1.70
1.00
1.00
1.60
2.20
2.80
3.40
4.00
4.60
5.20
5.80
6.40
7.00
P0
P1
P2
P3
P4
A0
B0
C0
D0
E0
F0
G0
H0
I0
J0
K0
L0
M0
N0
O0
Q0
R0

Testing ke-3

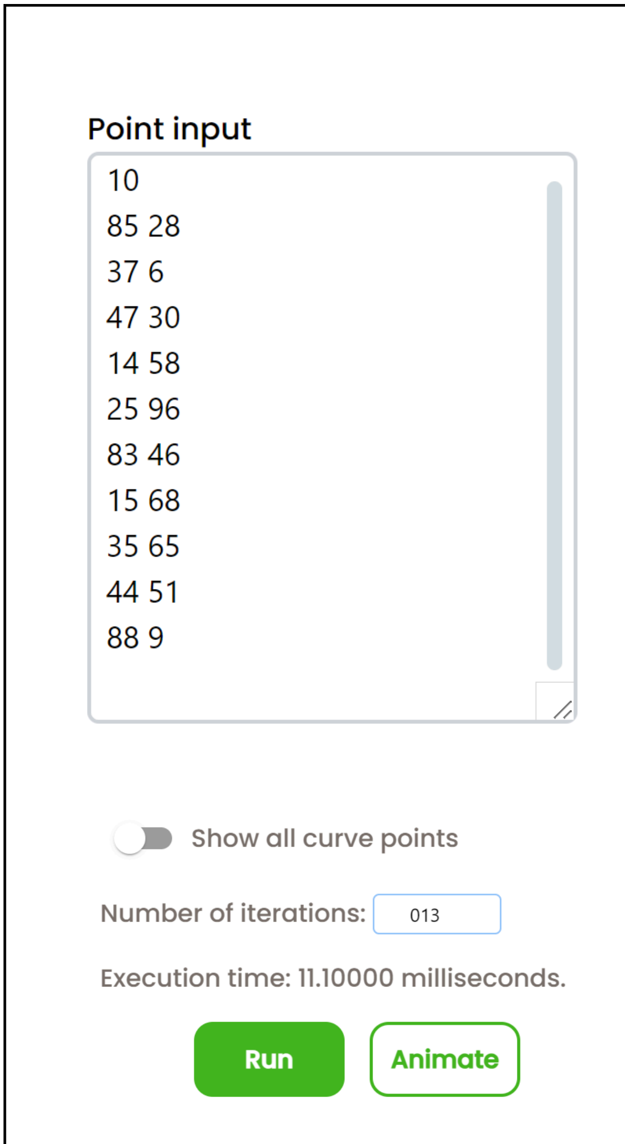

Point input
5
1 2
2 1
3 3
4 7
7 0
Show all curve points
Number of iterations: 04
Execution time: 0.00000 milliseconds.
Run
Animate

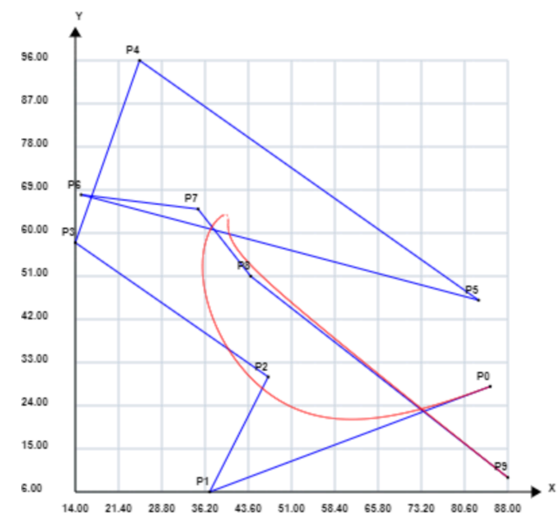

Y
X
7.00
6.30
5.60
4.90
4.20
3.50
2.80
2.10
1.40
0.70
0.00
1.00
1.60
2.20
2.80
3.40
4.00
4.60
5.20
5.80
6.40
7.00
P0
P1
P2
P3
P4

Testing ke-4


Point input
11
12 43
30 74
22 20
85 38
99 25
16 71
14 27
92 81
57 74
63 71
97 82
Choose File
No file chosen
Show all curve points
Number of iterations:
010
Execution time: 4.90000 milliseconds.
Run
Animate

Testing ke-5


Point input
10
85 28
37 6
47 30
14 58
25 96
83 46
15 68
35 65
44 51
88 9
Show all curve points
Number of iterations:
013
Execution time: 11.10000 milliseconds.
Run
Animate


Y
X
96.00
87.00
78.00
69.00
60.00
51.00
42.00
33.00
24.00
15.00
6.00
14.00
21.40
28.80
36.20
43.60
51.00
58.40
65.80
73.20
80.60
88.00
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9

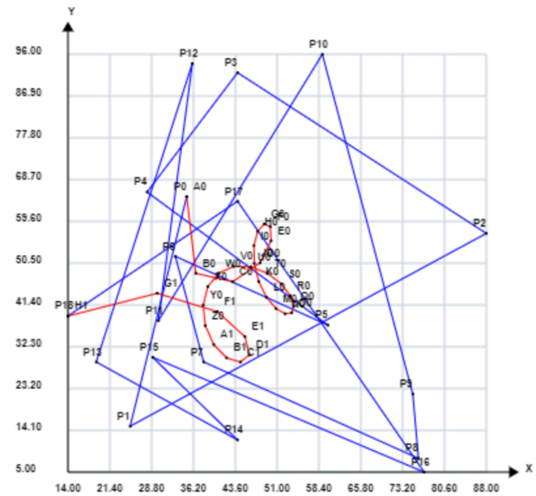
Testing ke-6

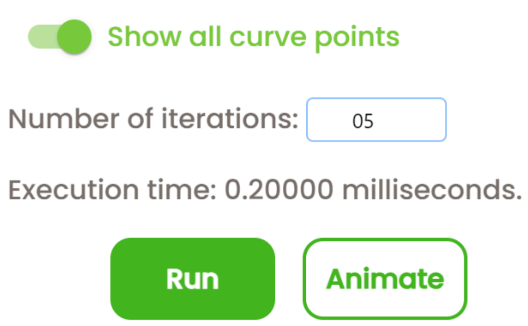
Point input

19
35 65
25 15
88 57
44 92
28 66
60 37
33 52
38 29
76 8
75 22
59 96

Point input

38 29
76 8
75 22
59 96
30 38
36 94
19 29
44 12
29 30
77 5
44 64
14 39




Show all curve points
Number of iterations:
05
Execution time: 0.20000 milliseconds.
Run
Animate

6 Analisis Perbandingan Solusi Brute Force dengan Divide and Conquer

Dalam algoritma yang telah diimplementasikan, baik menggunakan bruteforce maupun menggunakan middle point divide and conquer, telah dilakukan test untuk mencari yang manakah algoritma yang lebih efisien. Dari waktu yang diperlukan untuk setiap eksekusi algoritma, disadari bahwa implementasi menggunakan bruteforce lebih cepat dibandingkan dengan implementasi menggunakan divide and conquer.

Dengan tidak menghitung operasi matematika sebagai jumlah operasi, akan dianalisis jumlah eksekusi masing-masing implementasi dan kompleksitasnya. Jumlah eksekusi quadratic bezier curve dengan implementasi bruteforce berjumlah T(k) = 2^k dengan keterangan variabel k adalah jumlah iterasi. hal ini disebabkan karena algoritma bruteforce langsung mendapatkan titik dari ekspresi polinomial menggunakan matematika.

Sementara kuadratik bezier curve dengan implementasi divide and conquer berjumlah T(k) = 2*T(k - 1) + 9 karena untuk setiap iterasi diperlukan (setengah dari) 3^2 = 9 untuk mencari titik tengah kemudian menconquer masing-masing first-half dan second half dengan jumlah iterasi berkurang 1.

Jadi, untuk bruteforce, kompleksitasnya adalah O(2^k) sementara untuk divide and conquer kompleksitasnya adalah O(2^k x 9).

Untuk eksplorasi lebih lanjut, telah dilakukan eksplorasi middle point divide and conquer untuk bezier curve dengan N control points, yaitu O(2^k x N^2). Kesimpulannya, untuk kasus bezier curve, implementasi algoritma divide and conquer tidak membuat pencarian lebih efektif.

7 Lampiran

7.1 Pemenuhan Spesifikasi Tugas Kecil

No	Poin	Ya	Tidak
1	Program berhasil dijalankan	V	
2	Program dapat melakukan visualisasi kurva Bezier	V	
3	Solusi yang diberikan program optimal	V	
4	[Bonus] Program dapat membuat kurva untuk n titik kontrol	V	
5	[Bonus] Program dapat melakukan visualisasi proses pembuatan kurva	V	

7.2 Penjelasan Implementasi Bonus

Langkah-langkah:

1. Diberikan N control points katakanlah point-point tersebut adalah p0, p1, ..., pN-1 secara berurutan. dan sebuah angka jumlah iterasi, katakan K (K >= 0)(EDGE CASE) jika K bernilai 0, kembalikan points semula yang diberikan.
2. Buat dua buah array kosong: first dan second, untuk menyimpan titik-titik yang akan kita conquer pada iterasi selanjutnya
3. Lakukan perulangan yang dimulai pada titik-titik p0, p1, ..., pN
    a. Jika titik-titik yang diberikan berjumlah 1, maka lompat ke langkah nomor 3
    b. Diberikan titik-titik, katakanlah T0,.., Tc.
    c. Untuk setiap i yang mungkin, buatlah titik Mi yang merupakan middle point dari segment TiTi+1
    d. Dari langkah 3.b akan didapat c - 1 titik baru hasil konstruksi tersebut.
    e. Copy M0, masukkan ke array first. Kemudian copy M(c-1), masukkan ke array second.
    f. Lakukanlah kembali perulangan dengan titik-titik tersebut adalah M0, ... Mc-1.
4. Sekarang kita punya 1 titik, katakan M.
5. (*BASE CASE*) Jika K = 1, maka kita sudah selesai, kembalikan array yang memuat P0, M, P(N-1)
6. (*RECURSIVE CASE*) Jika K > 1, lakukan conquer terhadap titik-titik yang disimpan pada array first, dan lakukan juga conquer terhadap titik-titik yang disimpan array second namun urutannya dibalik.

7. Setelah melakukan conquer pada array first dan reversed second, lakukan penggabungan hasil conquer titik-titik tersebut.

Sementara untuk bonus visualisasinya pembuatannya, digunakan animasi hasil akhir per iterasi sampai dengan iterasi yang diinginkan seperti apa yang ditanyakan pada *qna* (nomor 8).

7.3 Pranala Github

https://github.com/ganadipa/Tucil2_13522040_13522066