LAPORAN TUGAS KECIL II

IF2211 STRATEGI ALGORITMA

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

Disusun oleh:

Adril Putra Merin 13522068

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

Daftar Isi

BAB I
DASAR TEORI

A. Algoritma Brute Force

Algoritma *brute force* adalah algoritma yang memiliki pendekatan sederhana, tetapi cukup mampu untuk menyelesaikan berbagai persoalan. Sesuai dengan namanya, algoritma *brute force* atau, diterjemahkan secara bebas, *tenaga kasar*, menyelesaikan persoalan dengan mencoba semua kemungkinan solusi hingga ditemukan solusi yang benar atau optimal. Secara umum, algoritma *brute force* tidak selalu menjadi pilihan yang efisien karena cenderung membutuhkan waktu yang sangat besar seiring meningkatnya ukuran ruang solusi. Karena itu, algoritma yang lebih cerdas dan efisien seringkali lebih dipilih untuk menyelesaikan persoalan tertentu.

B. Algoritma Divide and Conquer

Algoritma *divide and conquer* adalah sebuah metode pemecahan masalah yang terdiri dari tiga langkah utama: *divide*, *conquer*, dan *combine*. Metode ini sering digunakan untuk memecahkan persoalan kompleks dengan membagi persoalan tersebut menjadi subpersoalan yang lebih kecil, menyelesaikan subpersoalan tersebut secara rekursif, dan kemudian menggabungkan solusi dari subpersoalan tersebut menjadi solusi untuk masalah asli. Berikut adalah langkah-langkah umum dari algoritma *divide and conquer*.

1. *Divide*: Persoalan utama dibagi menjadi dua atau lebih subpersoalan yang lebih kecil dan serupa dengan persoalan asli. Pembagian dilakukan hingga submasalah memiliki ukuran yang cukup kecil untuk diselesaikan (*base case*) atau hingga iterasi tertentu.
2. *Conquer*: Setiap subpersoalan diselesaikan secara rekursif. Hal ini dapat dilakukan dengan menerapkan algoritma *divide and conquer* pada setiap subpersoalan, atau dengan menggunakan algoritma lain bergantung pada jenis persoalan dan kompleksitas submasalah.
3. *Combine*: Solusi dari berbagai subpersoalan digabungkan hingga memberikan solusi untuk masalah utama atau asli.

C. Bezier Curve

Bezier curve adalah kurva parametrik yang banyak digunakan dalam grafika computer dan banyak bidang lainnya. *Bezier curve* didefinisikan oleh sekumpulan titik-titik kontrol yang melalui P_0 hingga P_n dengan n adalah orde dari kurva tersebut ($n = 1$ linear, $n = 2$ kuadratik, $n = 3$ kubik, dan seterusnya). Titik kontrol P_0 dan P_n berturut-turut adalah titik awal dan titik akhir dari kurva yang dihasilkan, sedangkan titik kontrol diantara kedua titik tersebut biasanya tidak terletak pada kurva yang dihasilkan. Misalkan P_0 dan P_1 adalah dua titik yang berbeda, *linear bezier curve* dalam kasus ini adalah garis yang menghubungkan kedua titik tersebut sehingga fungsi parameter dari kurva ini adalah

$$B(t) = (1 - t)P_0 + tP_1$$

Bezier curve dapat didefinisikan secara rekursif untuk sebarang derajat n dengan mengekspresikan fungsi parameter sebagai interpolasi linear antara pasangan titik terkait pada dua *bezier curve* dengan derajat $n - 1$. Misalkan $B_{P_0P_1\dots P_k}$ adalah fungsi parameter *bezier curve* dengan titik-titik kontrol $P_0P_1\dots P_k$, maka definisi rekursif dari *bezier curve* adalah

$$B(t) = (1-t)B_{P_0P_1\ldots P_{n-1}}(t) + tB_{P_1P_2\ldots P_n}(t)$$

$$B_{P_0}(t) = P_0$$

BAB II
ANALISIS DAN IMPLEMENTASI ALGORITMA

A. Algoritma *Brute Force*

Salah satu pendekatan yang digunakan untuk memecahkan masalah ini adalah algoritma *brute force*. Pada program ini, *brute force* dilakukan dengan rekursif sesuai dengan definisi general *bezier curve* secara rekursif, yaitu:

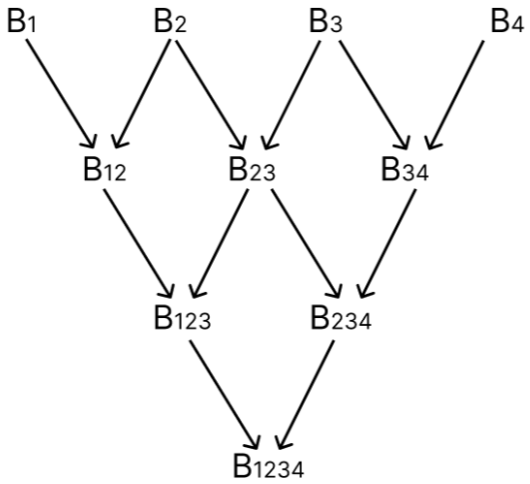
$$B(t) = (1-t)B_{P_0P_1\dots P_{n-1}}(t) + tB_{P_1P_2\dots P_n}(t)$$

$$B_{P_0}(t) = P_0$$

dengan $P_0P_1\dots P_n$ adalah titik-titik kontrol dari sebuah kurva bezier dengan derajat n. Jadi derajat yang digunakan pada algoritma ini sudah digeneralisasi untuk sebarang orde n dengan n bilangan bulat positif. Agar proses kalkulasi lebih optimal, proses untuk fungsi rekursi diatas dilakukan secara iteratif (*bottom-up*).

Adapun langkah-langkah algoritma *brute force* pada program ini adalah sebagai berikut.

1. Jika banyak titik kontrol kurang dari satu, maka kembalikan titik kontrol tersebut sebagai titik hasil. Jika tidak, lanjutkan pada langkah ke-2.
2. Hitung berapa banyak titik yang dihasilkan dimana $banyak\ titik = 2^k - 1$ dengan k adalah banyak iterasi. Banyak titik ini akan menentukan banyak t yang digunakan pada fungsi parameter $B(t)$ dengan $t \in [0, 1]$. Perhatikan bahwa banyak titik yang dihitung di awal adalah jumlah titik hasil kurva yang bukan merupakan titik kontrol awal dan akhir sehingga pada akhirnya, banyak titik yang dihasilkan adalah $2^k + 1$. Jadi, parameter t akan memiliki bentuk $t = i/2^k$ dengan $i \in [0, 2^k]$ dan i bilangan bulat. Misalkan banyak iterasi adalah 2, maka banyak titik (tanpa titik kontrol) adalah 3 sehingga $t = \{0, 0.25, 0.5, 0.75, 1\}$.
3. Untuk setiap t, akan dilakukan kalkulasi menggunakan fungsi rekursi diatas secara iteratif. Lakukan kalkulasi dimulai dari *bezier curve* orde 1 hingga orde ke-n. Sebagai ilustrasi, berikut adalah contoh urutan perhitungan untuk kasus $n = 4$.



4. Masukkan setiap titik hasil kalkulasi *t* ke dalam *list of points* hasil.

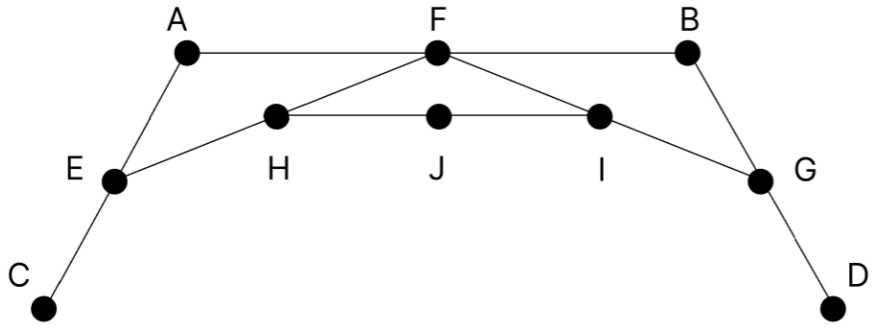
Alasan mengapa banyak titik yang dihasilkan adalah perpangkatan dua dari banyak iterasi adalah untuk mempermudah perbandingan algoritma *brute force* dengan *divide and conquer*. Hal ini karena banyak titik yang dihasilkan dengan algoritma *divide and conquer* untuk *k* iterasi adalah $2^k + 1$.

B. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* dapat digunakan untuk menyelesaikan persoalan *bezier curve* dengan *n* buah titik kontrol dan *k* buah iterasi. Algoritma ini akan menghasilkan sebanyak $2^k + 1$ titik yang akan digunakan untuk mengkontruksi *bezier curve*.

Berdasarkan permasalahan yang diberikan, penulis sudah melakukan generalisasi algoritma *middle point* dengan *divide and conquer* untuk sebarang orde *n*. Secara umum, untuk *k* buah iterasi dan n buah titik, langkah-langkah dari algoritma ini adalah sebagai berikut.

1. Inisialisasi *kedalaman* saat ini sebagai 0.
2. Lakukan operasi terhadap titik-titik kontrol saat ini sehingga dihasilkan *list of point left, mid*, dan *right*. Operasi tersebut dilakukan dengan cara mencari titik tengah dari segmen garis yang dibentuk oleh n titik kontrol yang berurutan sehingga dihasilkan $n - 1$ titik tengah. Ulangi operasi tersebut dengan titik tengah yang didapatkan sebelumnya sebagai titik kontrol baru hingga ditemukan satu titik tengah. Berikut adalah ilustrasi operasi ini untuk banyak titik kontrol awal sebanyak n.



Pada awalnya, titik-titik kontrol pada operasi tersebut adalah A, B, C, dan D. Selanjutnya, diperoleh titik-titik tengah dari titik kontrol tersebut sebagai E, F, G. Lakukan operasi tersebut hingga ditemukan 1 titik tengah (*mid*), yang dalam hal ini adalah J. Selain *mid,* operasi ini juga akan menghasilkan *list left* dan *right* yang berisi titik kontrol baru untuk *kedalaman* selanjutnya. *List left* berisi titik kontrol awal pertama dan kumpulan titik tengah paling awal dari setiap iterasi pada operasi pencarian titik tengah, sedangkan *right* berisi titik kontrol terakhir dan kumpulan titik tengah paling akhir dari setiap iterasi pada operasi pencarian titik tengah. Misalkan, pada contoh diatas, *left* akan berisi elemen {C, E, H, J}, sedangkan *right* akan berisi elemen {J, I, G, D}.
3. Ulangi Langkah (2) dengan titik kontrol *left* dan *right* hingga *kedalaman* sama dengan banyak iterasi yang diinginkan.
4. Gabungkan hasil titik tengah pada titik kontrol *left* dan *right* dengan mid pada langkah (2). Penggabungan dilakukan

BAB III
SOURCE CODE

Algoritma pada program ini menggunakan bahasa Go, sedangkan bagian *frontend* menggunakan bahasa *typescript* menggunakan *React*. Karena *source code* untuk program ini cukup panjang, penulis hanya akan menampilkan *source code* yang berkaitan dengan algoritma pada laporan ini.

A. Struktur Data dan Metode

bezier_points.go

```go
package models

type BezierPoints struct {
  Points    []Point `json:"points"`
  Neff      int     `json:"neff"`
  Iteration int     `json:"iteration"`
}

func (bp *BezierPoints) InsertBefore(newBp BezierPoints) {
  bp.Points = append(newBp.Points, bp.Points...)
  bp.Iteration = newBp.Iteration
  bp.Neff += newBp.Neff
}

func (bp *BezierPoints) InsertAfter(newBp BezierPoints) {
  bp.Points = append(bp.Points, newBp.Points...)
  bp.Iteration = newBp.Iteration
  bp.Neff += newBp.Neff
}
```

point.go

```go
package models

type Point struct {
  X float64 `json:"x"`
  Y float64 `json:"y"`
}
```

response.go

```go
package models
```

```
type Response struct {
  Result BezierPoints `json:"result"`
  Time   float64      `json:"time"`
}
```

B. Algoritma Brute Force

bruteforce.go

```
package algorithms

import (
  "bezier/backend/models"
  "math"
)

func calculateBezier(t float64, bp models.BezierPoints) models.Point {
  res := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points:
[]models.Point{}}
  res.InsertAfter(bp)
  for i := 0; i < bp.Neff-1; i++ {
    for j := 0; j < res.Neff-1; j++ {
      res.Points[j] = models.Point{X: (1-t)*res.Points[j].X +
t*res.Points[j+1].X, Y: (1-t)*res.Points[j].Y + t*res.Points[j+1].Y}
    }
    res.Neff--
    res.Points = res.Points[:res.Neff]
  }
  return res.Points[0]
}

func GetPointsBruteforce(bp models.BezierPoints) models.BezierPoints {
  if bp.Neff < 2 {
    return bp
  }
  iter := int(math.Pow(2, float64(bp.Iteration))) - 1
  result := models.BezierPoints{Iteration: bp.Iteration, Neff: 2 +
iter}
  for i := 0; i < iter+2; i++ {
```

```
    result.Points = append(result.Points,
calculateBezier(float64(i)/float64(1+iter), bp))
  }
  return result
}
```

C. Algoritma *Divide and Conquer*

dnc.go

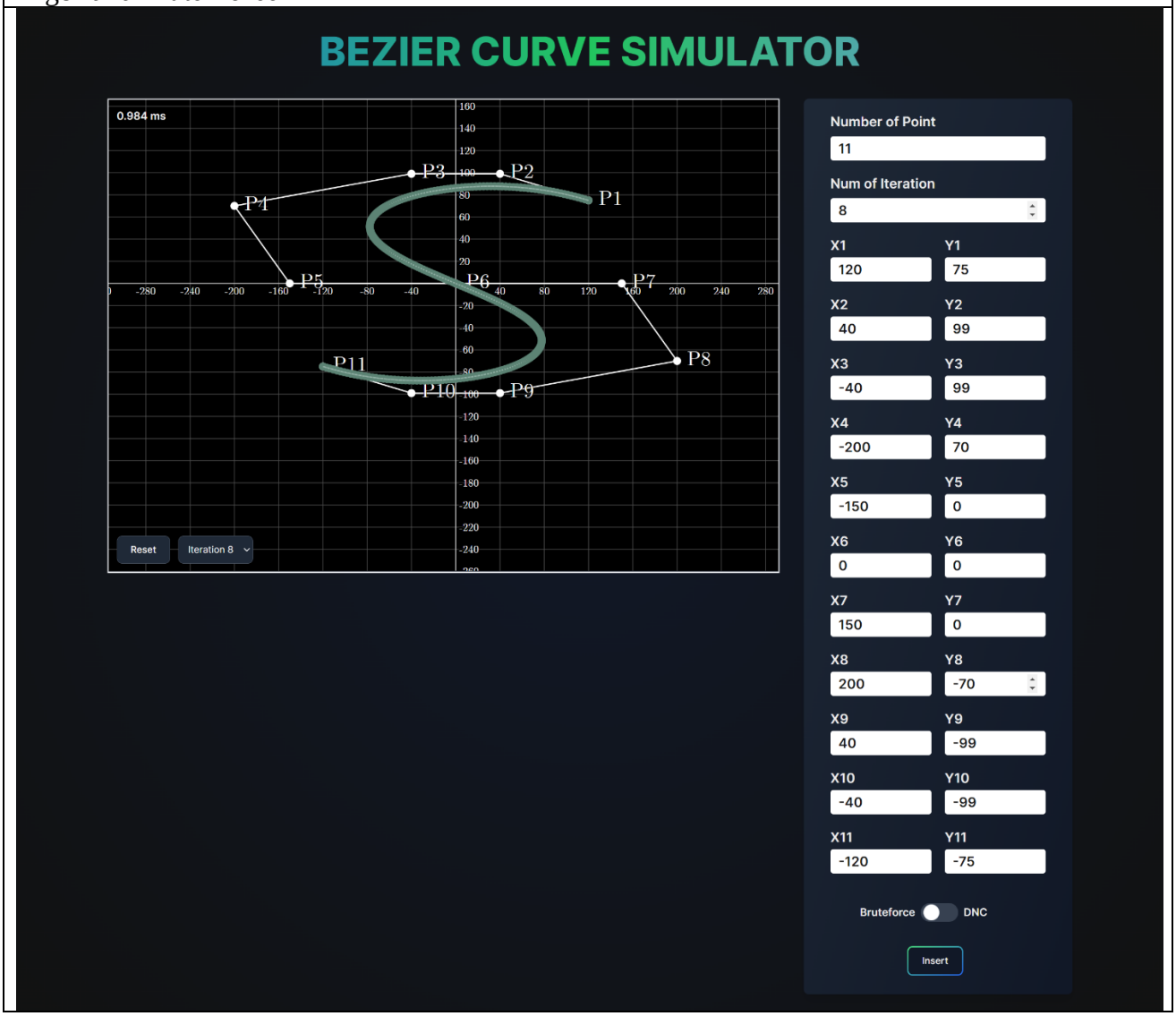
```
package algorithms

import (
  "bezier/backend/models"
)

func getMidPoints(bp models.BezierPoints) (models.BezierPoints,
models.BezierPoints, models.BezierPoints) {
  mid := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points:
[]models.Point{}}
  left := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points:
[]models.Point{}}
  right := models.BezierPoints{Iteration: bp.Iteration, Neff: 0,
Points: []models.Point{}}

  left.InsertAfter(models.BezierPoints{Neff: 1, Points:
[]models.Point{bp.Points[0]}, Iteration: bp.Iteration})
  right.InsertAfter(models.BezierPoints{Neff: 1, Points:
[]models.Point{bp.Points[bp.Neff-1]}, Iteration: bp.Iteration})
  mid.InsertAfter(bp)
  for i := 0; i < bp.Neff-1; i++ {
    for j := 0; j < mid.Neff-1; j++ {
      mid.Points[j] = models.Point{X: (mid.Points[j].X +
mid.Points[j+1].X) / 2, Y: (mid.Points[j].Y + mid.Points[j+1].Y) / 2}
    }
    mid.Neff--
    mid.Points = mid.Points[:mid.Neff]
    left.InsertAfter(models.BezierPoints{Neff: 1, Points:
[]models.Point{mid.Points[0]}, Iteration: bp.Iteration})
```

```
    right.InsertBefore(models.BezierPoints{Neff: 1, Points:
[]models.Point{mid.Points[mid.Neff-1]}, Iteration: bp.Iteration})
  }
  return left, mid, right
}

func findPoints(bp models.BezierPoints, dep int) models.BezierPoints {
  if dep >= bp.Iteration {
    return models.BezierPoints{Iteration: bp.Iteration}
  }
  left, mid, right := getMidPoints(bp)

  leftMidPoints := findPoints(left, dep+1)
  rightMidPoints := findPoints(right, dep+1)

  leftMidPoints.InsertAfter(mid)
  leftMidPoints.InsertAfter(rightMidPoints)
  return leftMidPoints
}

func GetPointsDnc(bp models.BezierPoints) models.BezierPoints {
  if bp.Neff < 2 {
    return bp
  }
  result := models.BezierPoints{Iteration: bp.Iteration, Neff: 2}
  result.Points = append(result.Points, bp.Points[0])
  result.InsertAfter(findPoints(bp, 0))
  result.Points = append(result.Points, bp.Points[bp.Neff-1])
  return result
}
```

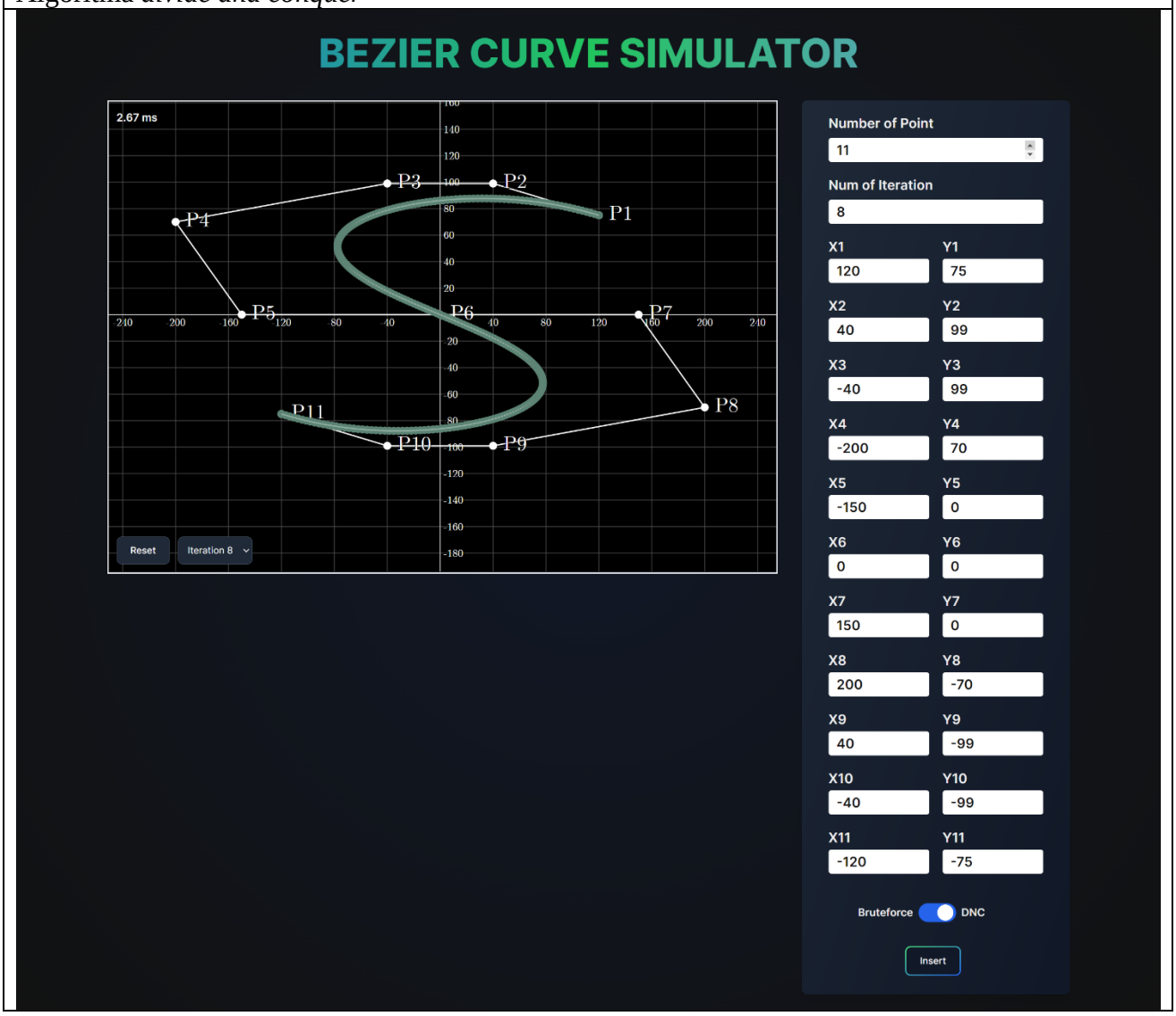
BAB IV
EKSPERIMEN

A. Uji kasus 1

Algoritma *Brute Force*

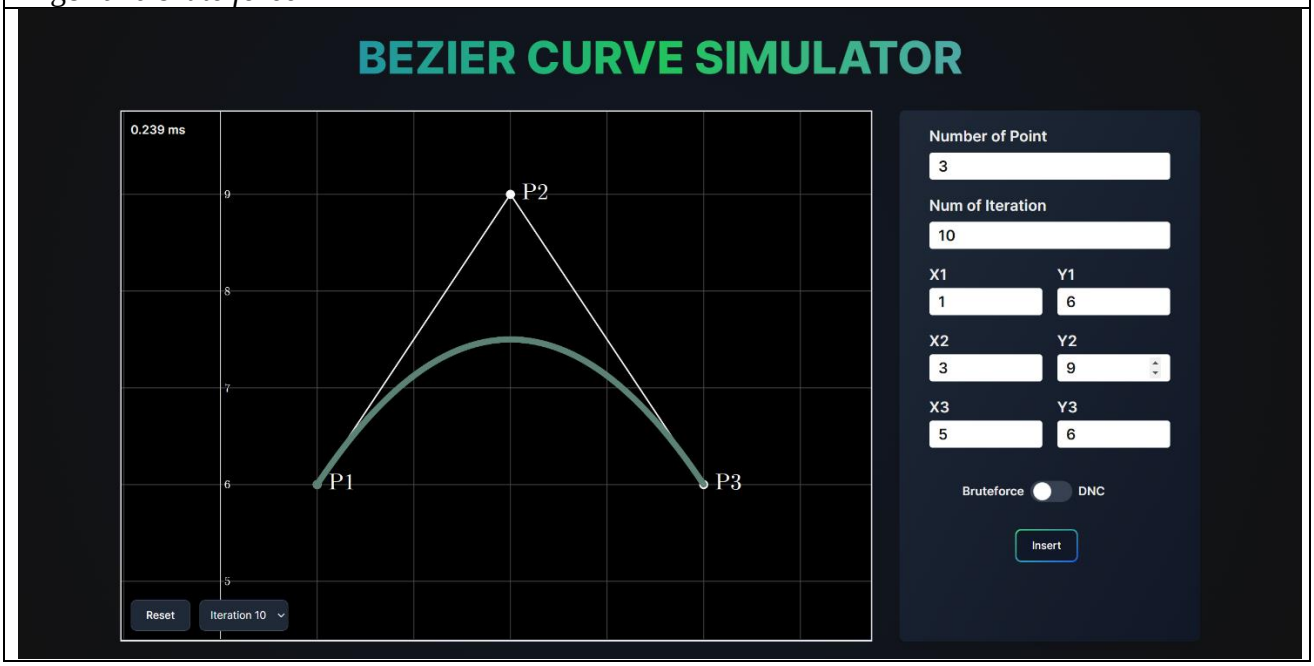


Algoritma *divide and conquer*

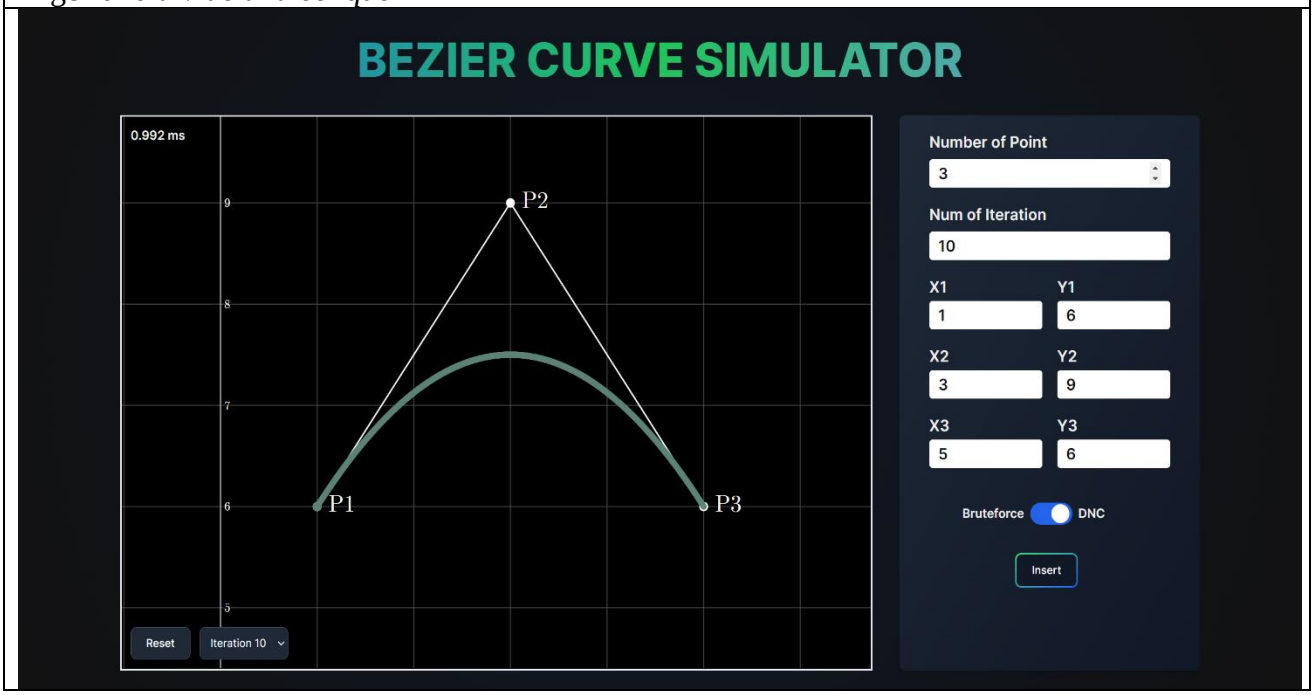


B. Uji Kasus 2

Algoritma *brute force*

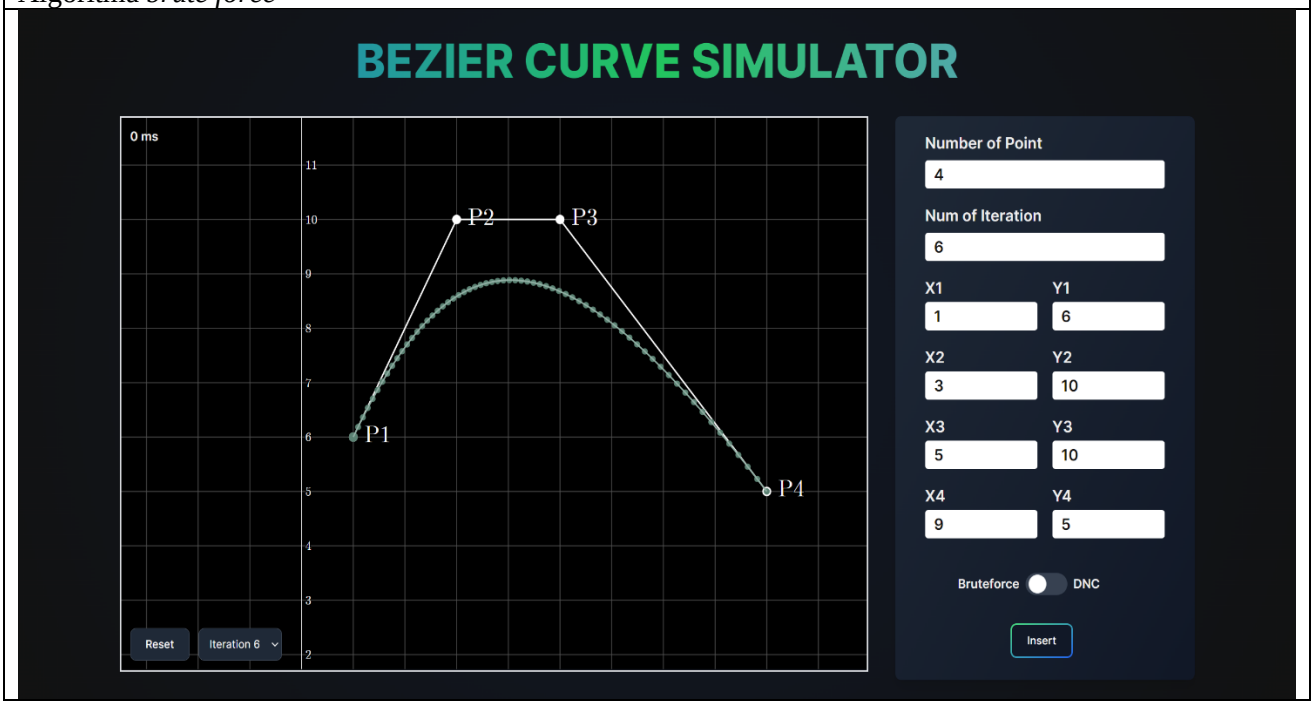


Algoritma *divide and conquer*

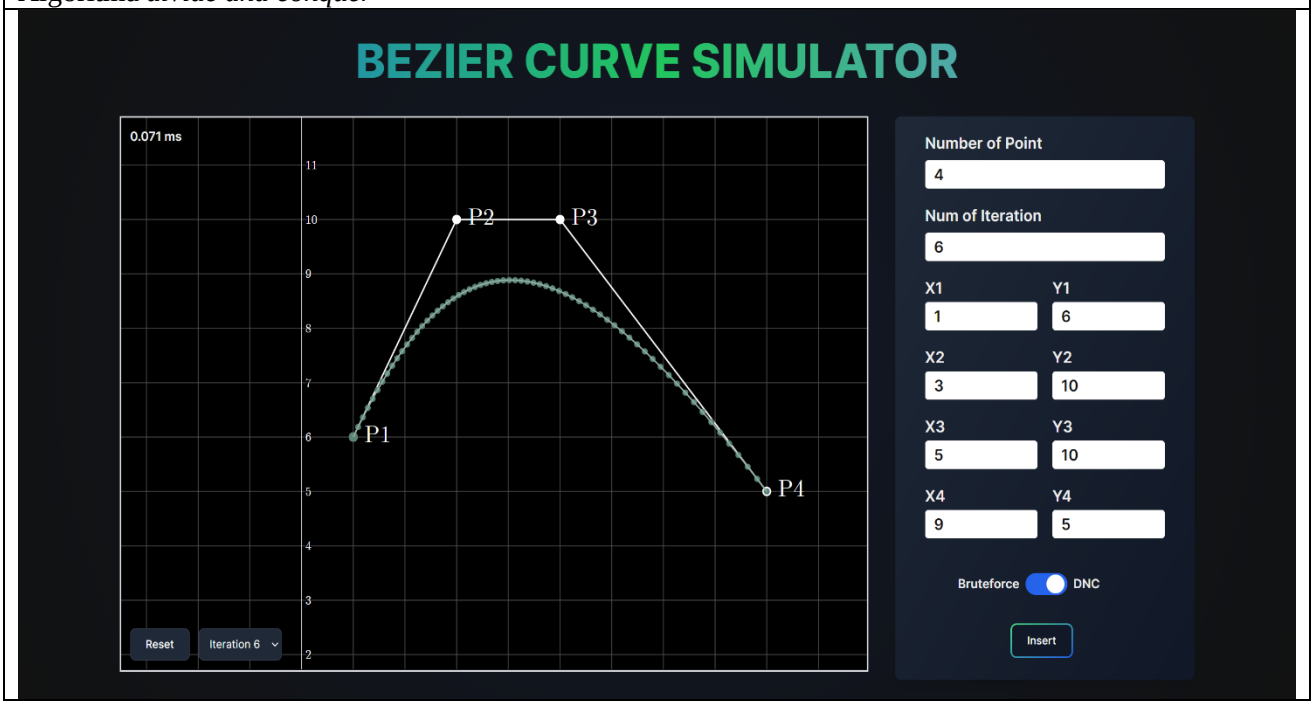


C. Uji Kasus 3

Algoritma *brute force*

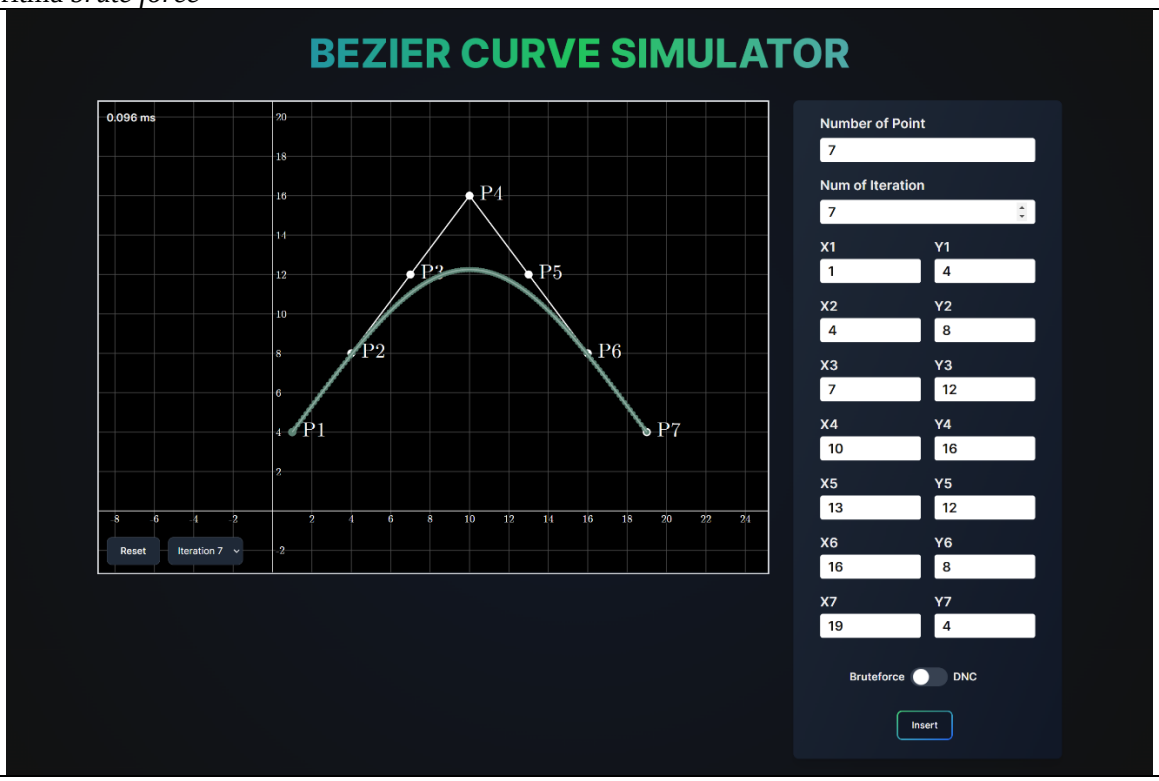


Algoritma *divide and conquer*

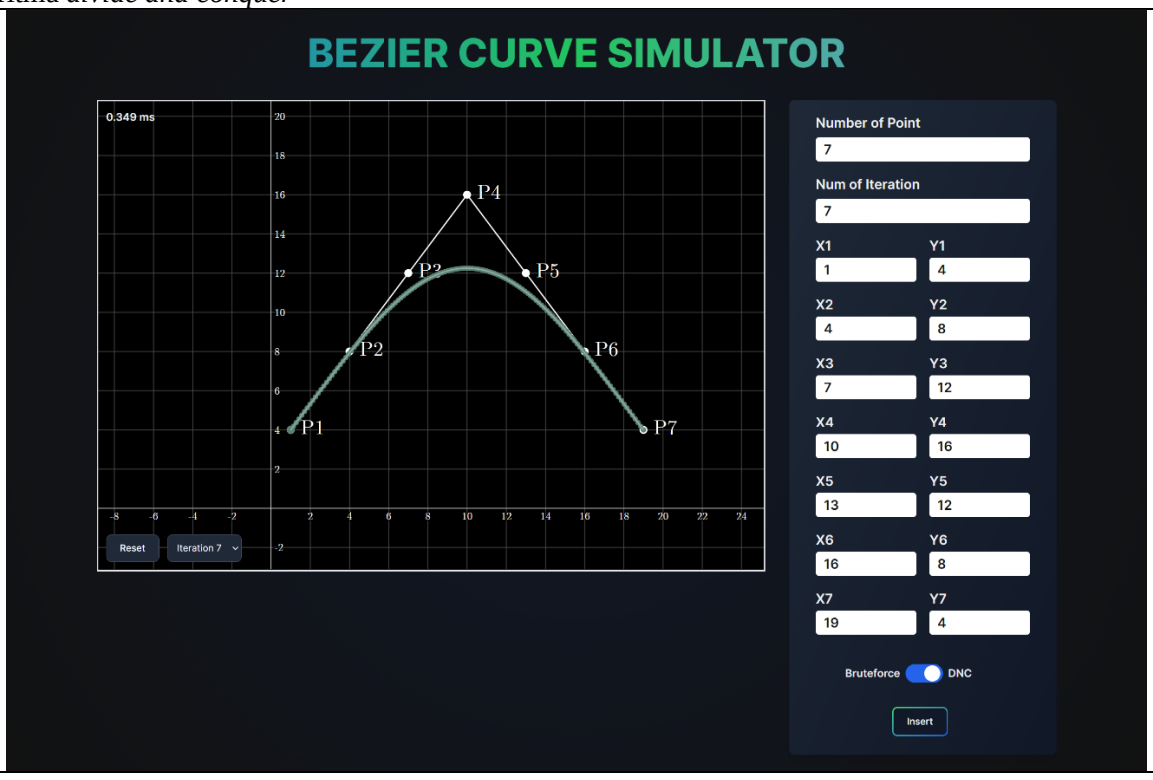


D. Uji Kasus 4

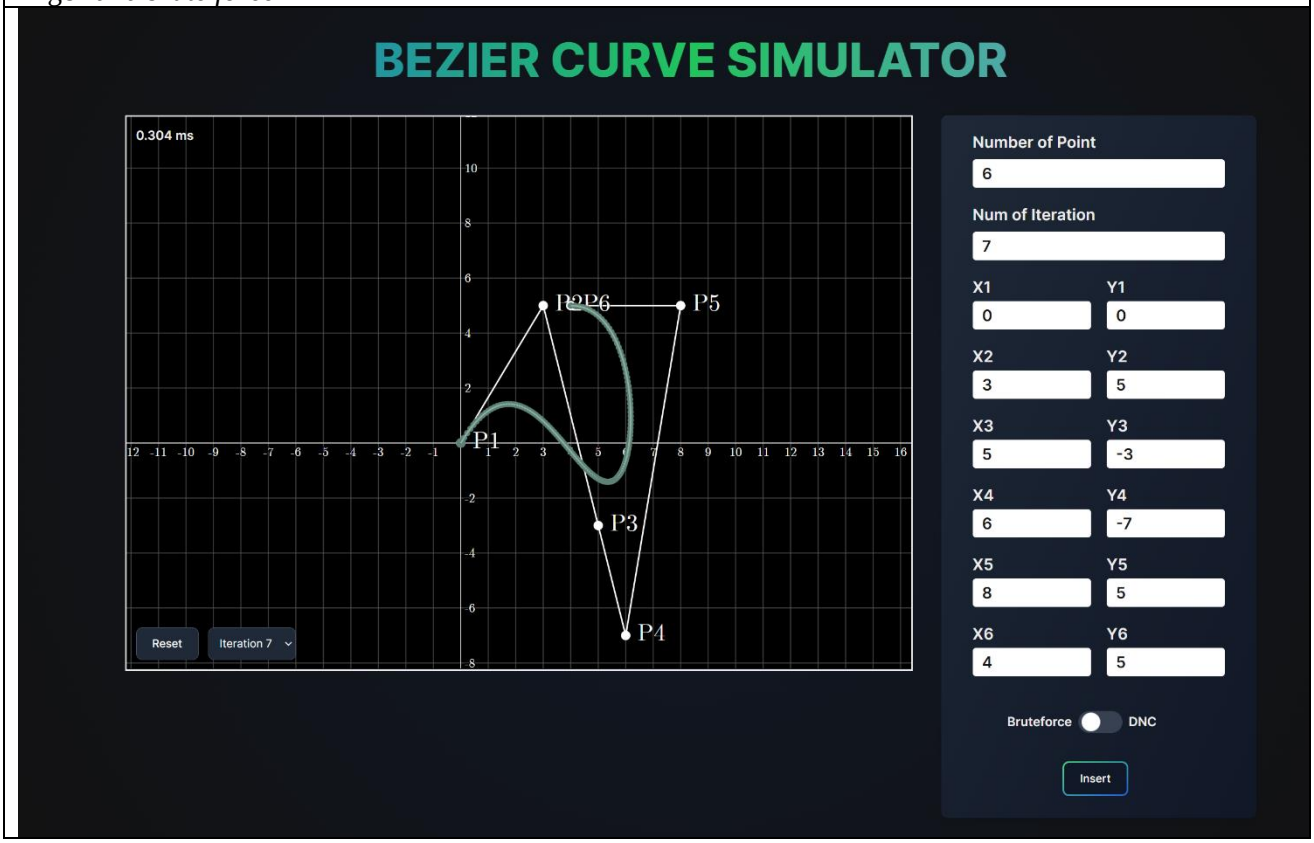
Algoritma *brute force*



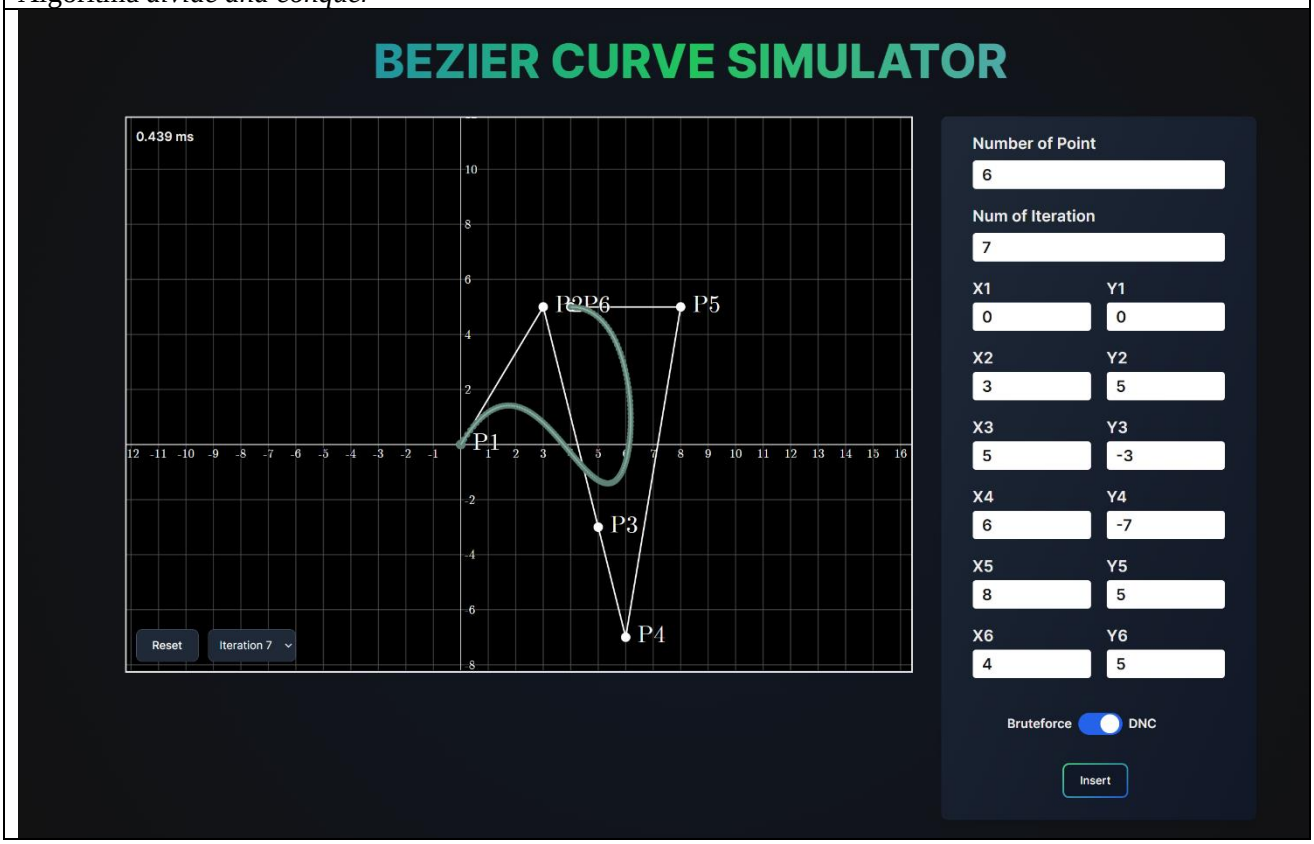
Algoritma *divide and conquer*



E. Uji Kasus 5

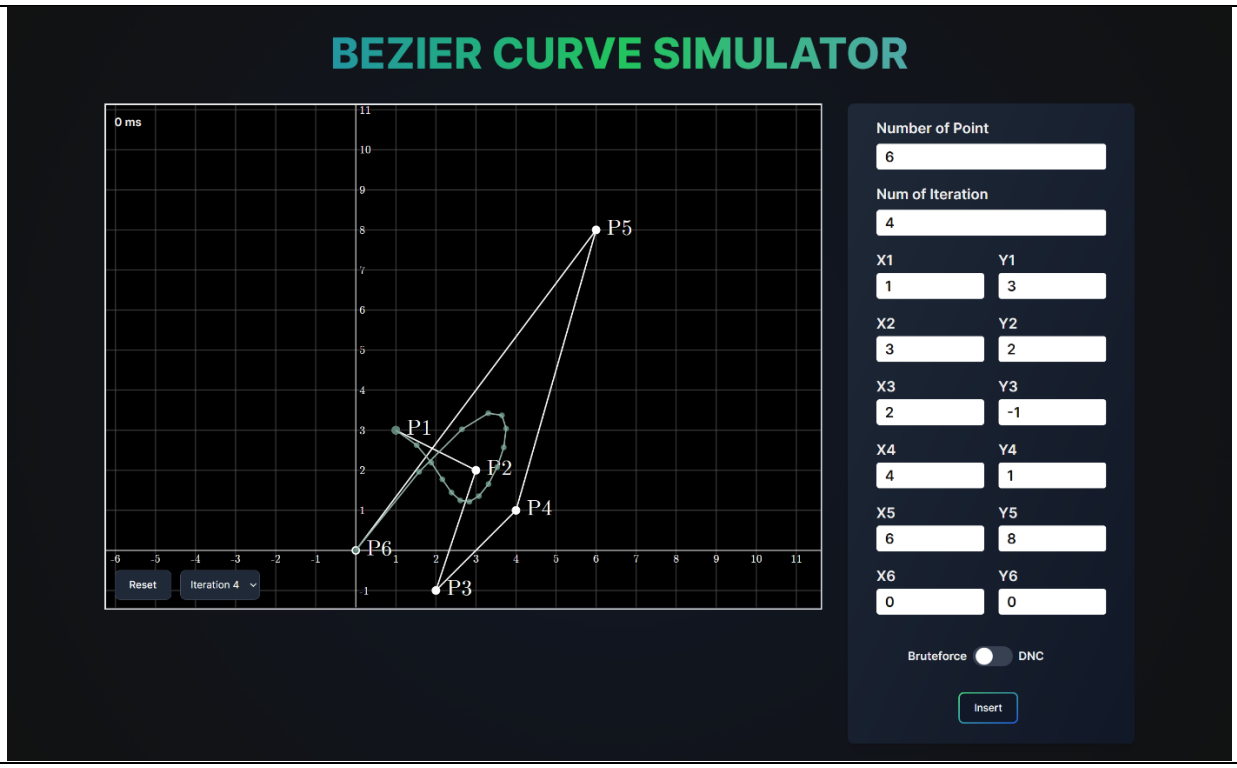
Algoritma *brute force*


Algoritma *divide and conquer*

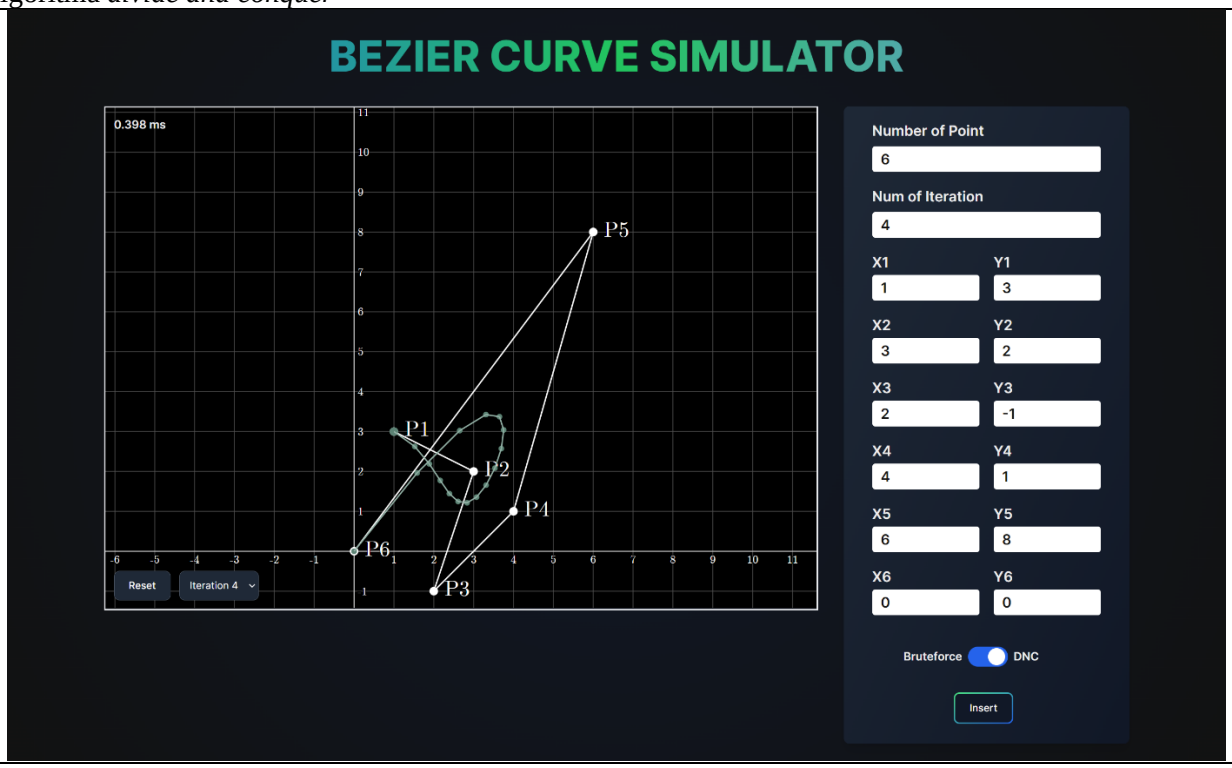


F. Uji Kasus 6

Algoritma *brute force*

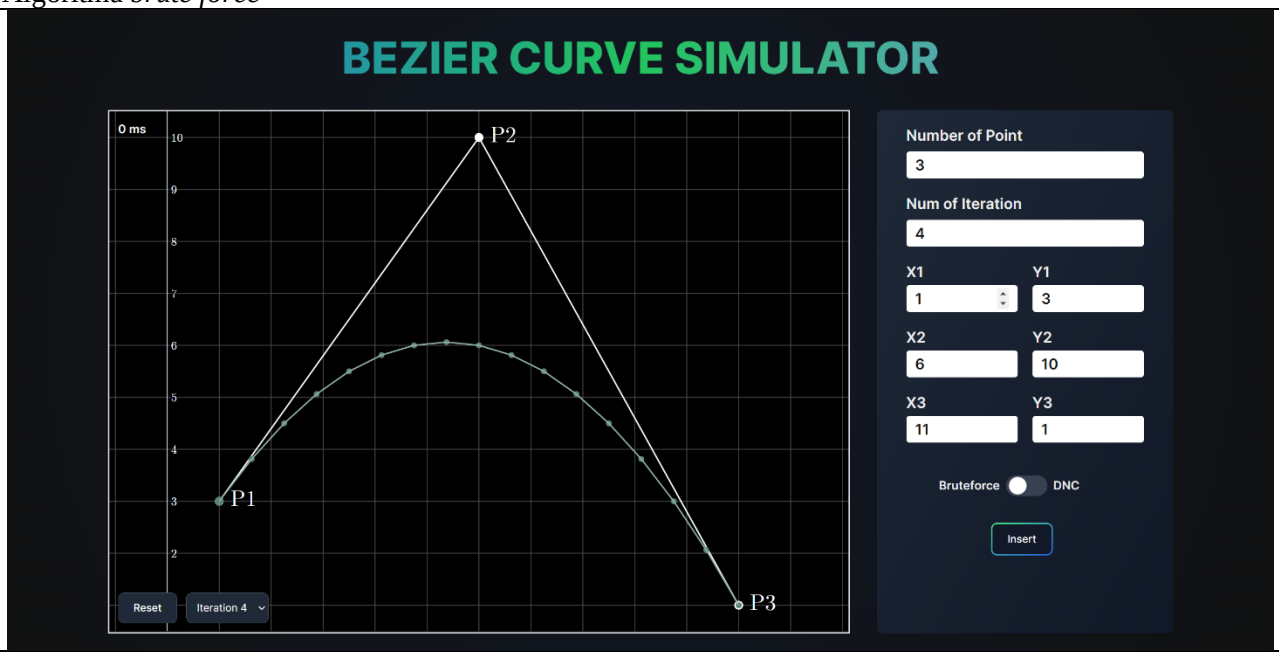


Algoritma *divide and conquer*

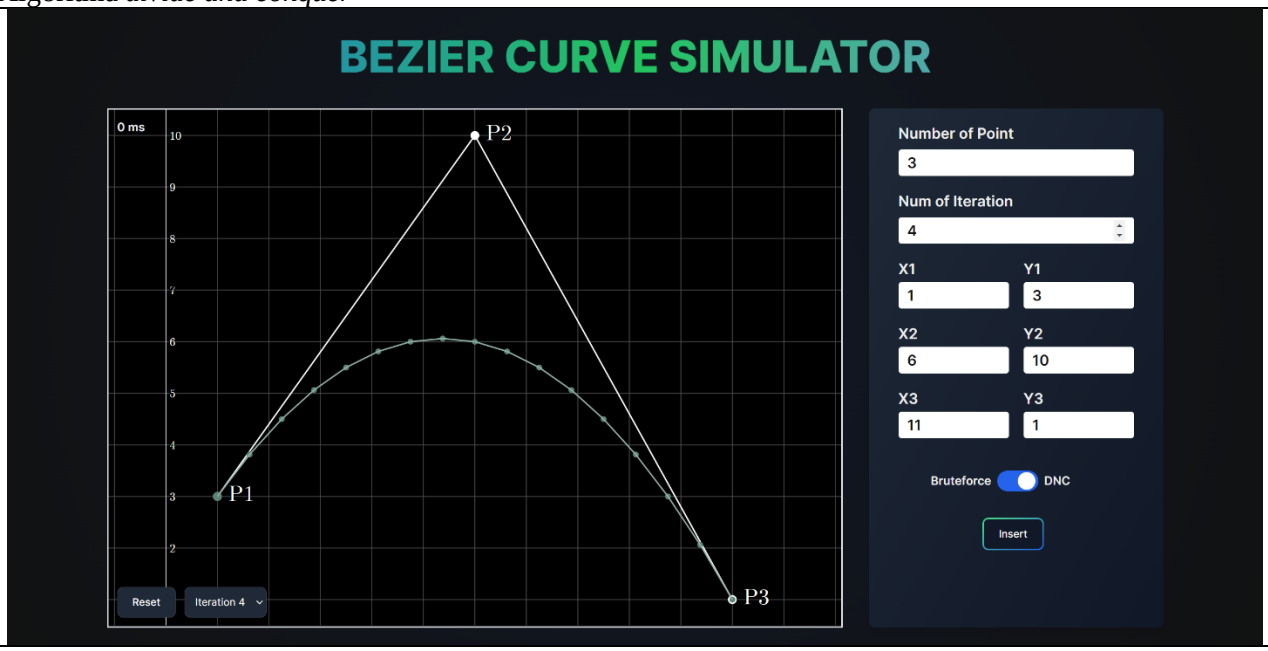


G. Uji Kasus 7

Algoritma *brute force*

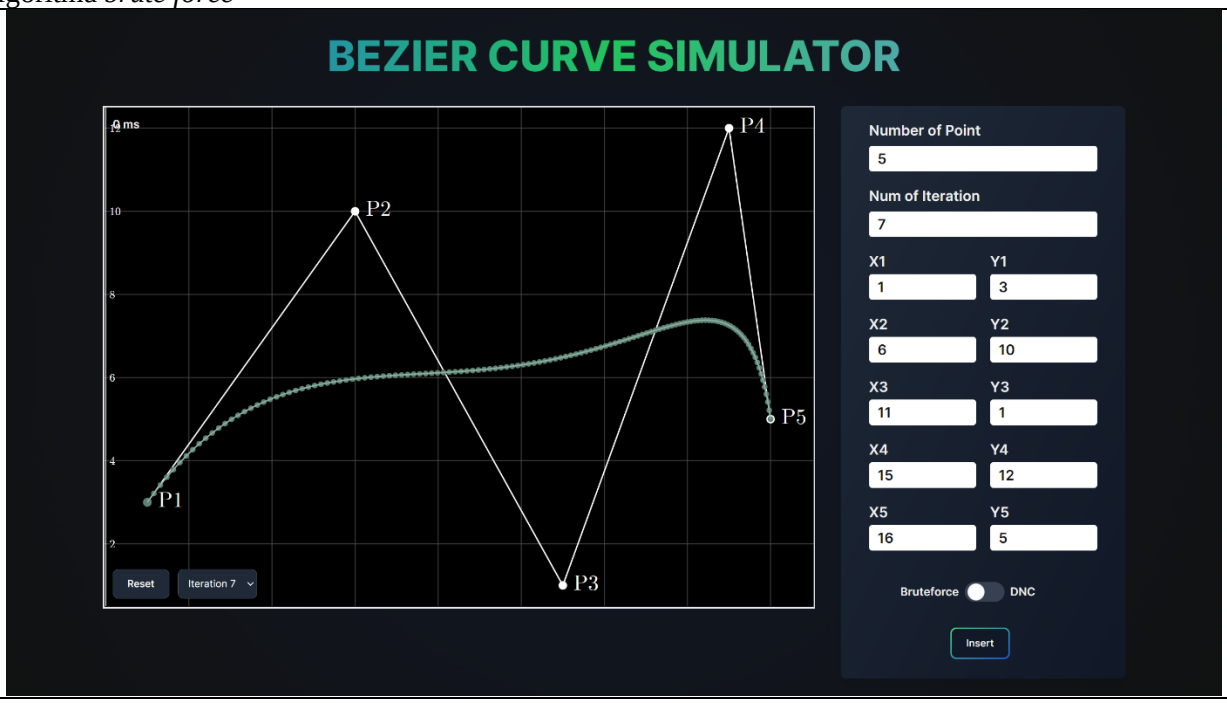


Algoritma *divide and conquer*

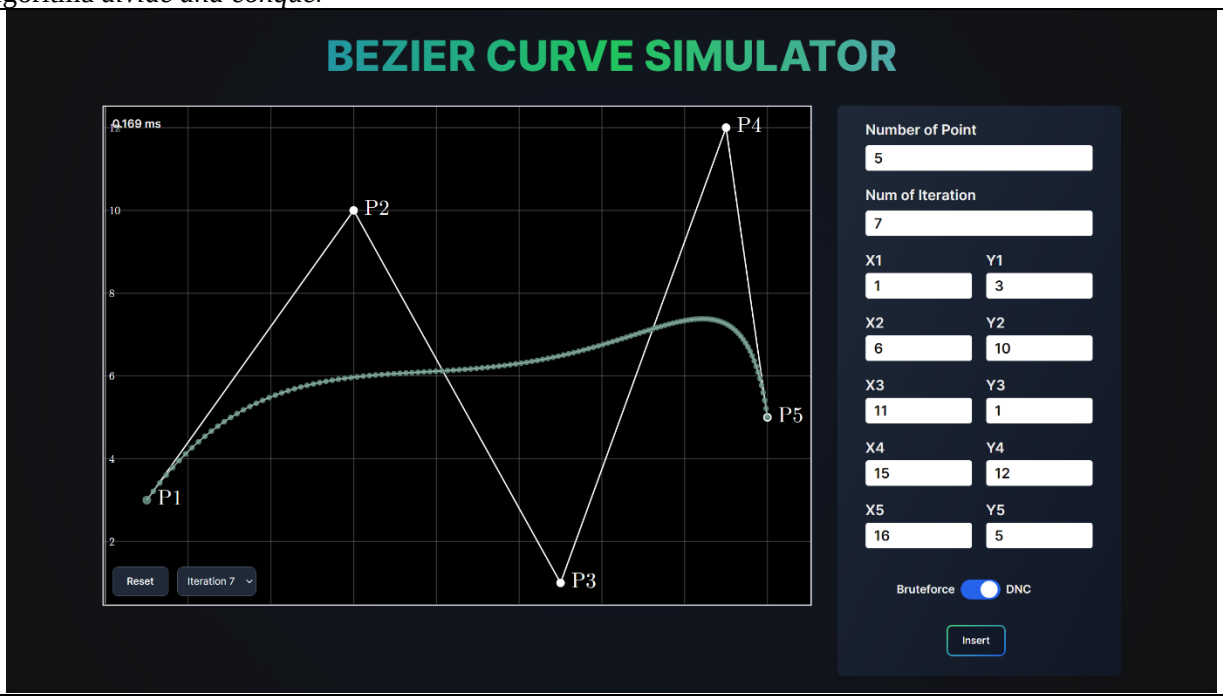


H. Uji Kasus 8

Algoritma *brute force*

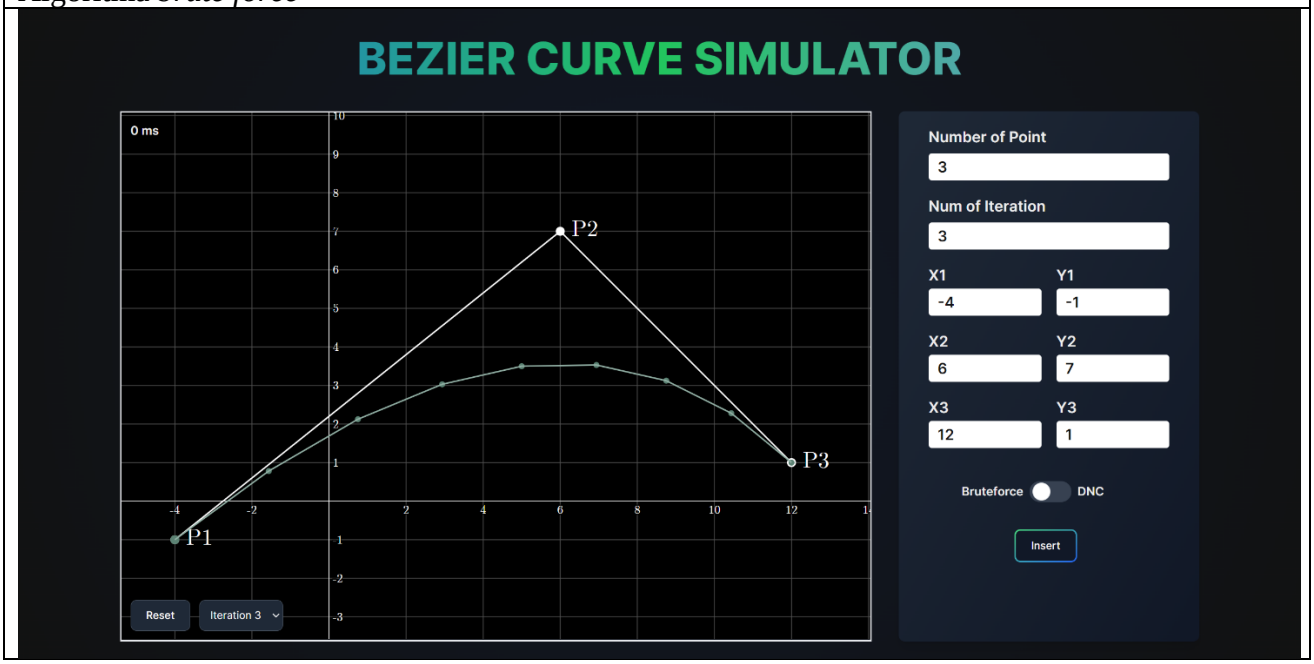


Algoritma *divide and conquer*

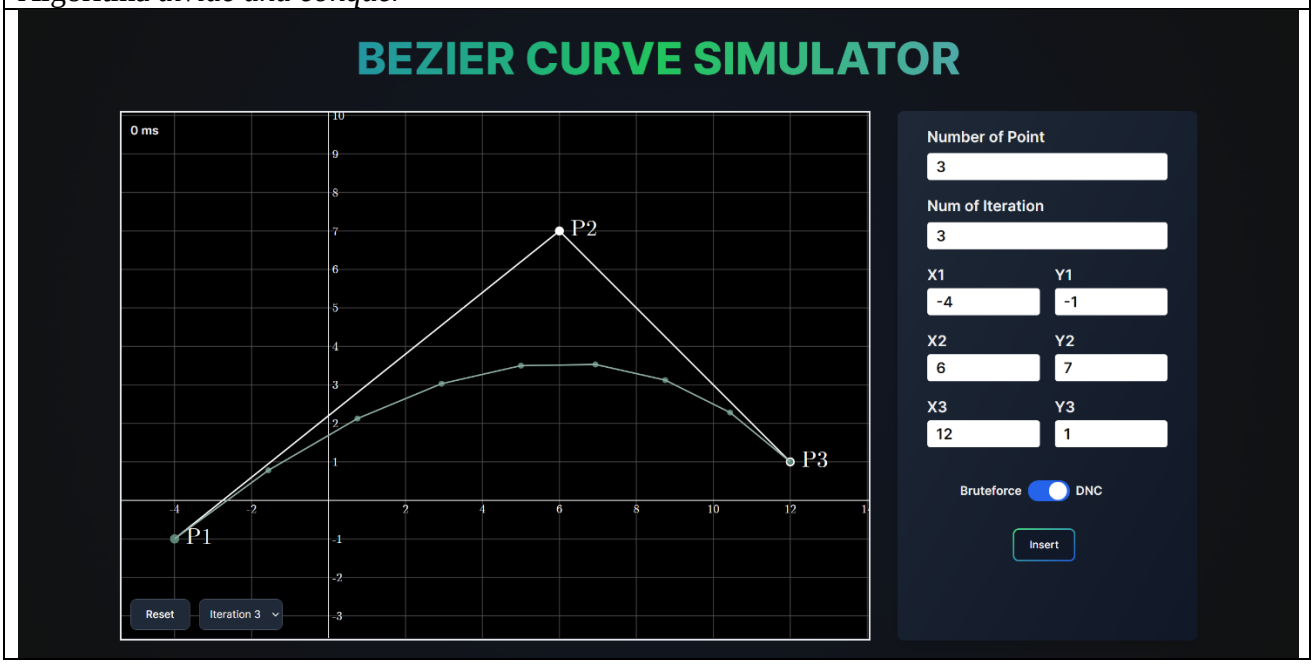


I. Uji Kasus 9

Algoritma *brute force*

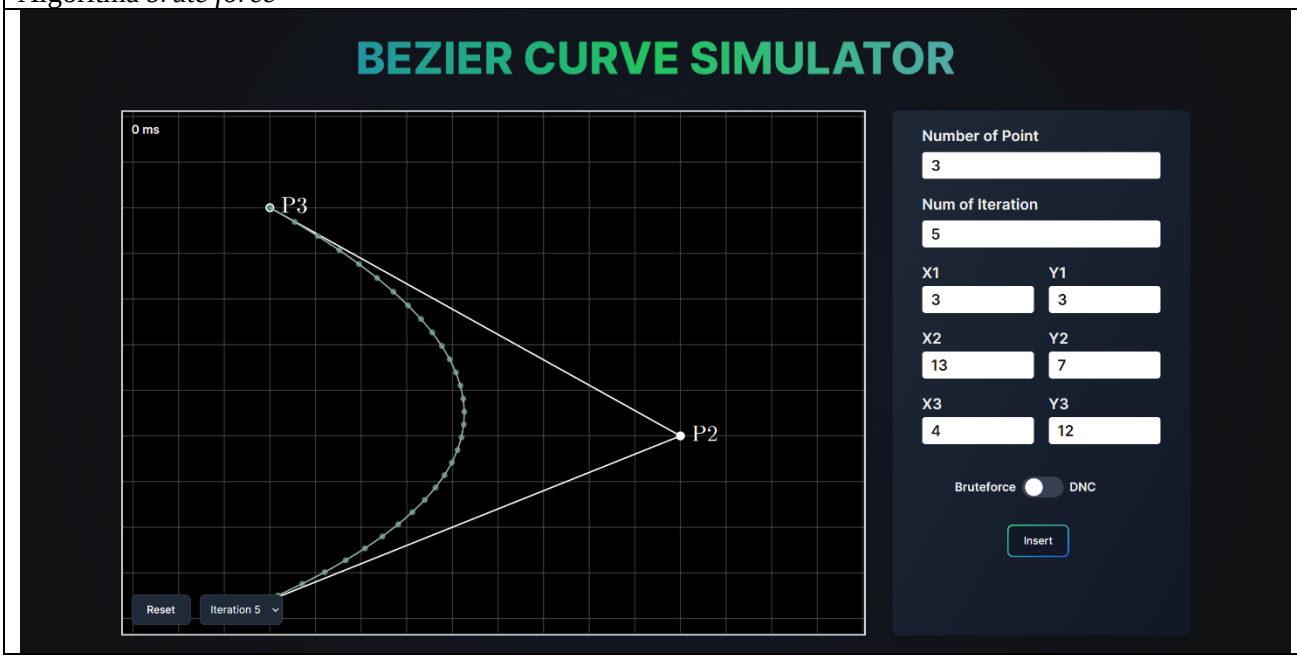


Algoritma *divide and conquer*

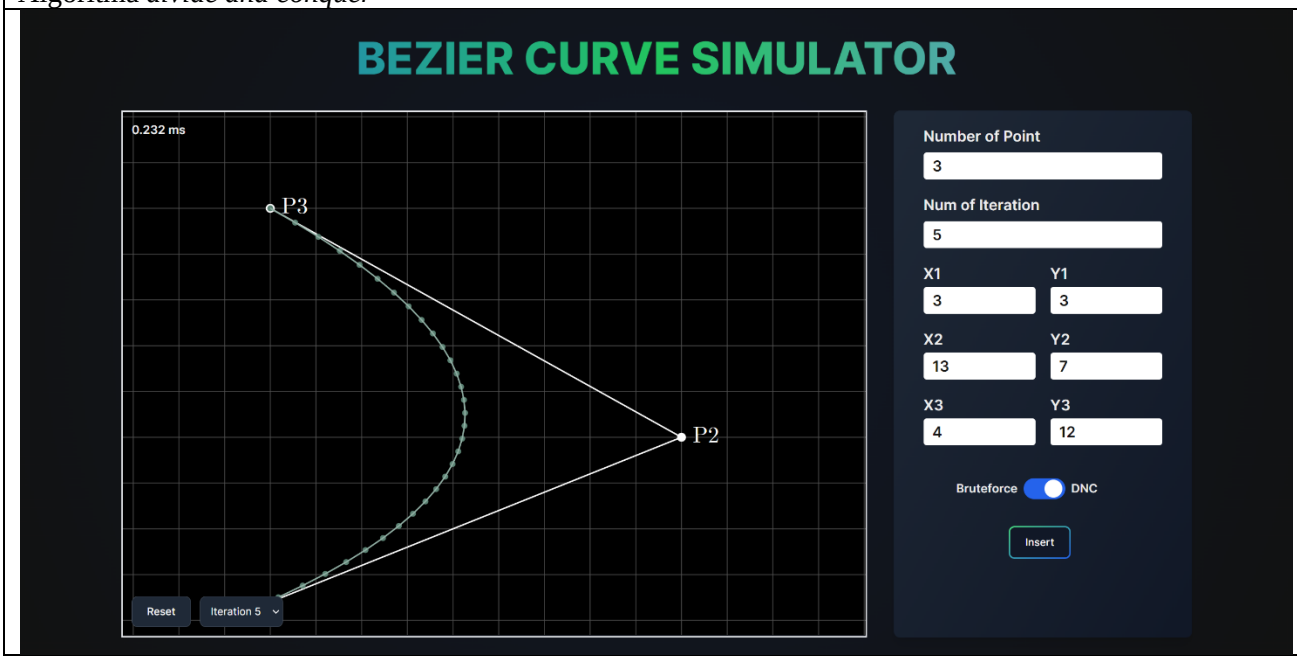


J. Uji Kasus 10

Algoritma *brute force*

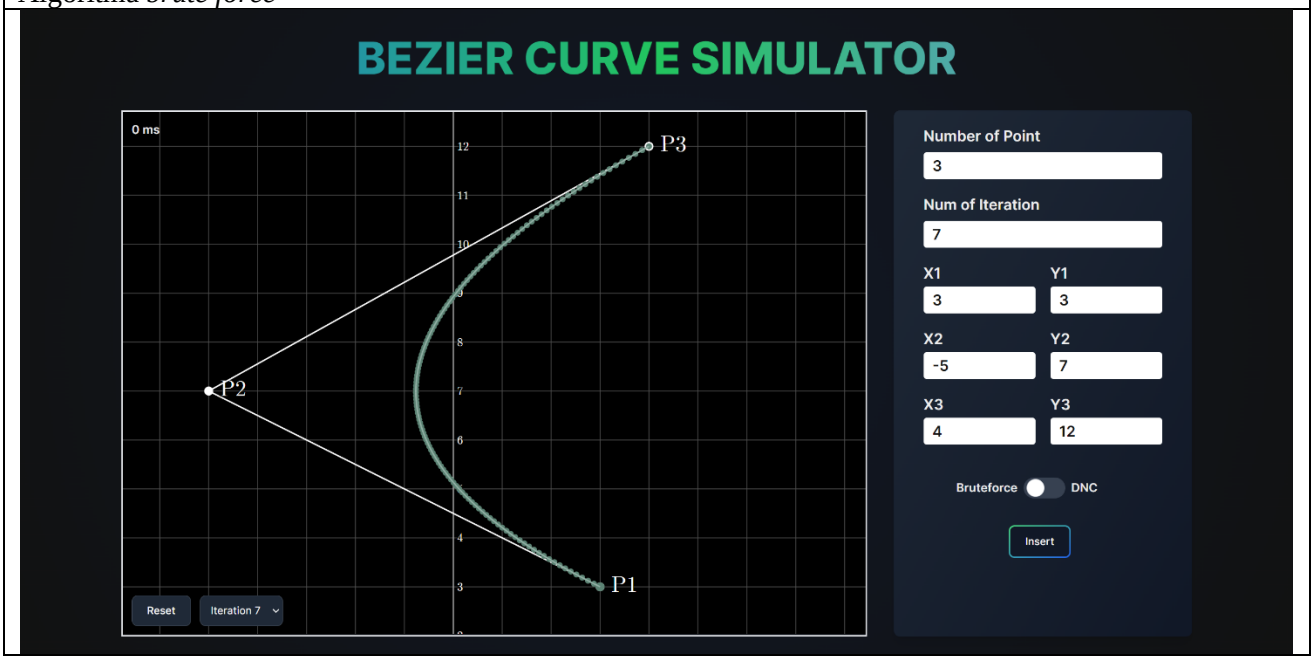


Algoritma *divide and conquer*

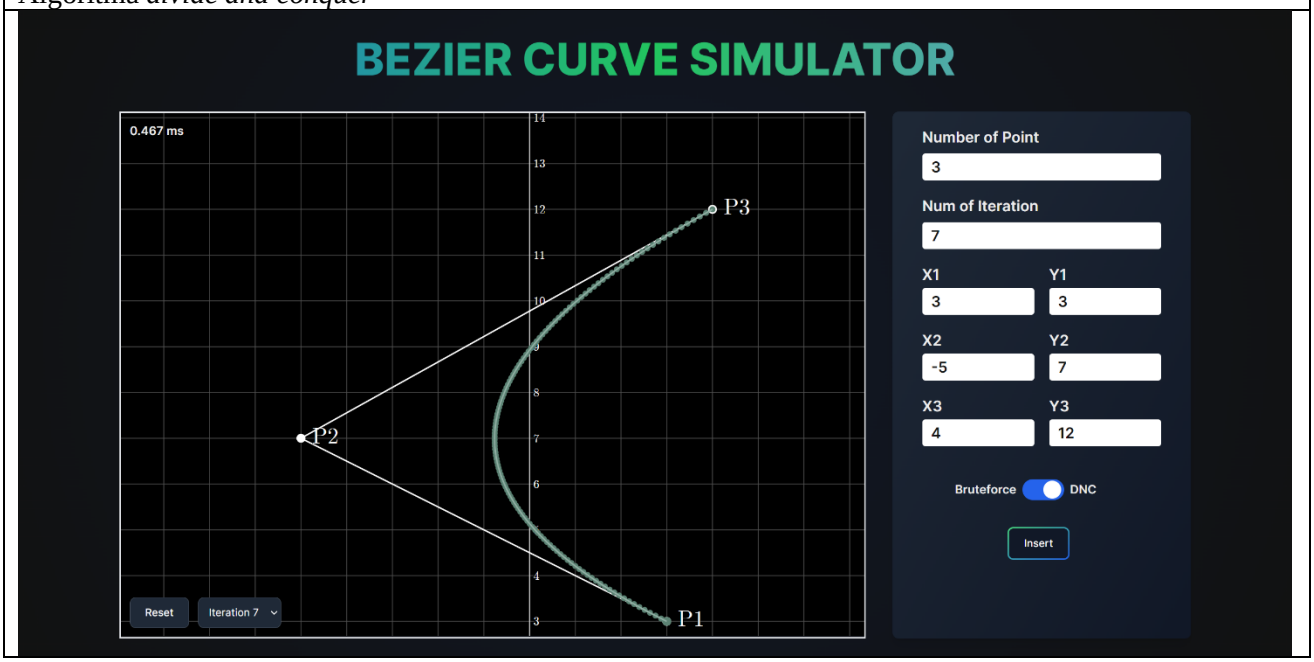


K. Uji Kasus 11

Algoritma *brute force*

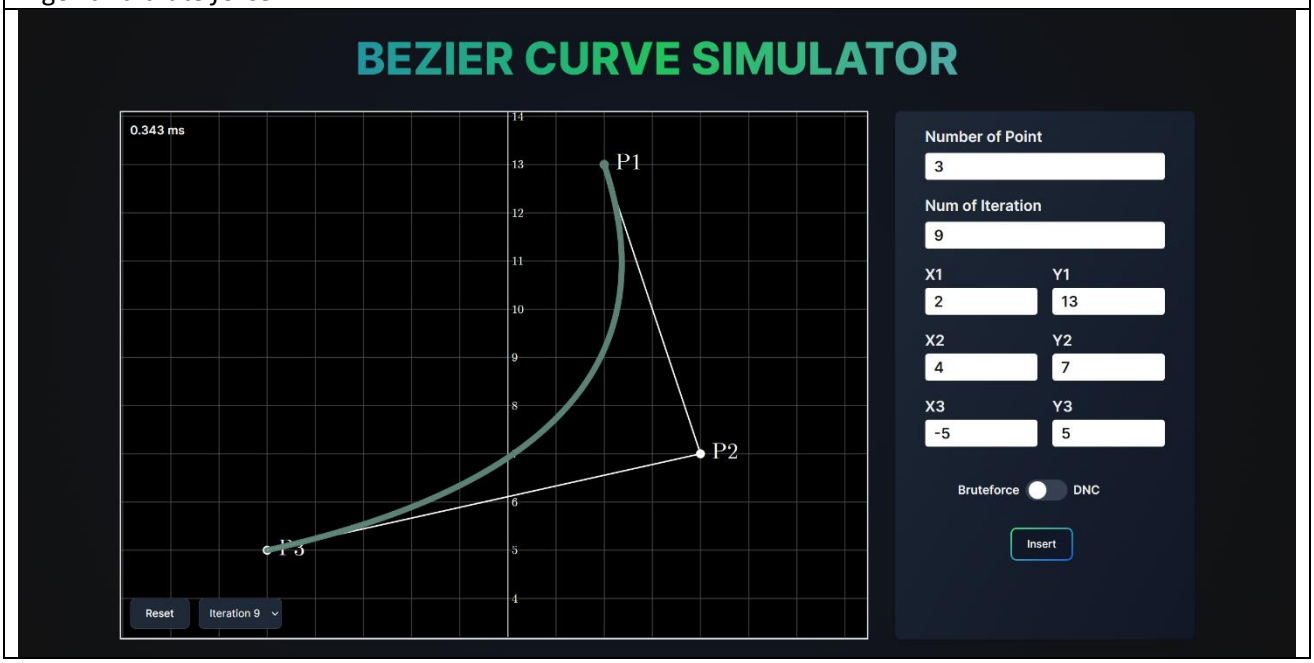


Algoritma *divide and conquer*

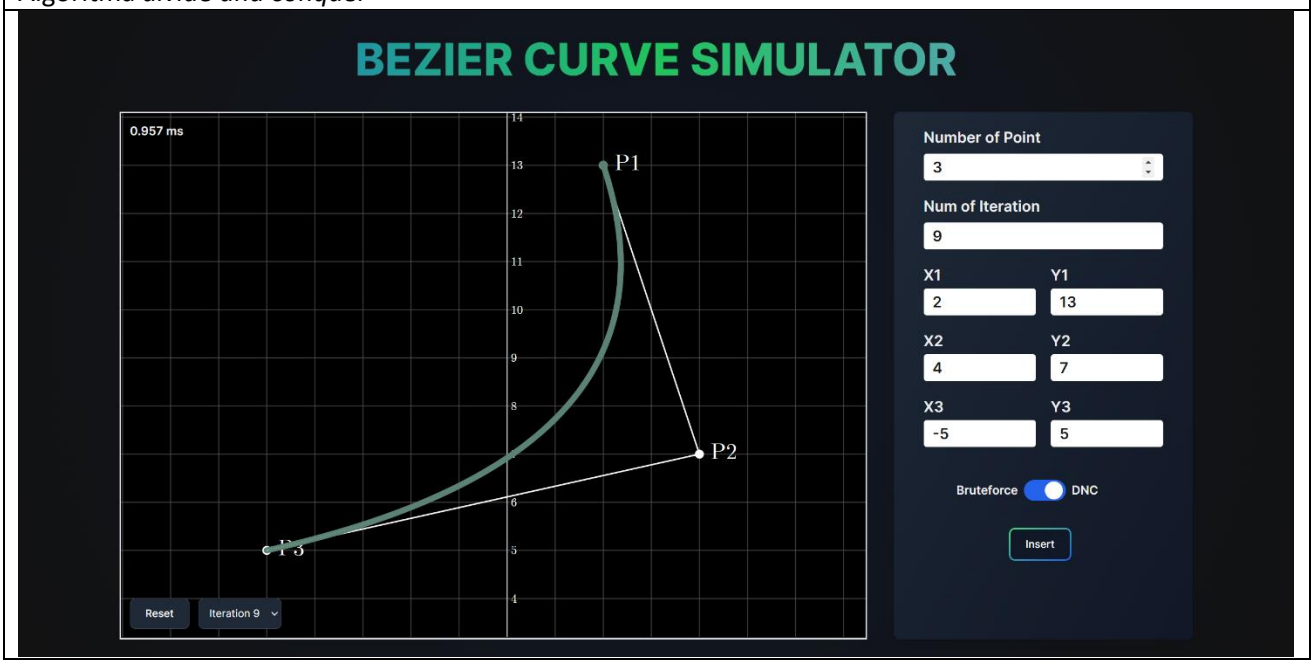


L. Uji Kasus 12

Algoritma *brute force*



Algoritma *divide and conquer*



BAB V
ANALISIS PERBANDINGAN SOLUSI

A. Analisis Kompleksitas Algoritma *Brute Force*

Berdasarkan penjelasan dan *source code* algoritma *brute force* pada bab sebelumnya, kita dapat melakukan analisis terhadap kompleksitas waktunya. Untuk proses *calculateBezier* dengan banyak titik *n*, dan banyak iterasi *k*, kompleksitas waktu yang diperoleh adalah

$$T(n) = T(n-1) + n - 1$$

$$T(0) = 0$$

Dengan menyederhanakan ekspresi tersebut, diperoleh

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + n - 2 + n - 1 \\ &= T(0) + 1 + \ldots + n - 2 + n - 1 \\ &= \frac{n(n-1)}{2} \end{aligned}$$

Selanjutnya, proses *calculateBezier* dilakukan sebanyak $2^k + 1$ kali sehingga kompleksitas waktu total adalah $T(n,k) = (2^k + 1)\frac{n(n-1)}{2} = O((2^k + 1)\frac{n(n-1)}{2}) = O(2^k n^2)$.

B. Analisis Kompleksitas Algoritma *Divide and Conquer*

Berdasarkan penjelasan dan *source code* algoritma *divide and conquer* pada bab sebelumnya, kita dapat melakukan analisis terhadap kompleksitas waktunya. Untuk n buah titik dan k buah iterasi diperoleh kompleksitas waktu sebagai berikut

$$T(n,k) = \frac{n(n-1)}{2} + 2T(n,k-1)$$

$$T(n,0) = 0$$

Dengan melakukan penyederhanaan terhadap persamaan diatas, diperoleh kompleksitas waktu sebagai berikut.

$$\begin{aligned} T(n,k) &= \frac{n(n-1)}{2} + 2(\frac{n(n-1)}{2} + 2T(n,k-2))) \\ &= \frac{n(n-1)}{2}(1 + 2 + 4 + 8 + \ldots + 2^{k-1}) \end{aligned}$$

Jadi,

$$T(n,k) = \frac{n(n-1)}{2}2^k = O(n^2 2^k)$$

Namun, kompleksitas waktu tersebut belum mempertimbangkan proses penggabungan hasil proses *left*, *mid*, dan *right* yang pada kode di atas menggunakan *InsertAfter* (untuk lebih jelasnya, silahkan lihat pada bagian *source code* pada bab 3). Untuk *k* buah iterasi, dilakukan *insertion* titik pada *list* sebanyak 2^{k-1} kali sehingga kompleksitas waktu total menjadi.

$$T(n,k) = \frac{n(n-1)}{2}2^{k-1} + 2T(n,k-1)$$

$$T(n,0) = 0$$

Dengan menyederhanakan ekspresi tersebut diperoleh hasil sebagai berikut.

$$T(n,k) = \frac{n(n-1)}{2}2^{k-1} + 2(\frac{n(n-1)}{2}2^{k-2} + 2T(n,k-2)))$$

$$= \frac{n(n-1)}{2}2^{k-1} + 2(\frac{n(n-1)}{2}2^{k-2}) + 4(\frac{n(n-1)}{2}2^{k-3} + 2T(n,k-3)))$$

$$= \frac{n(n-1)}{2}k2^{k-1}$$

Jadi kompleksitas waktu total dalam notasi big-O adalah

$$T(n,k) = \frac{n(n-1)}{2}k2^{k-1} = O(kn^2 2^k)$$

C. Perbandingan Hasil Uji Kasus

No	Banyak Titik Kontrol	Banyak iterasi	*Brute Force* (ms)	DNC (ms)
1	11	8	0.984	2.67
2	3	10	0.239	0.992
3	4	6	0	0.071
4	7	7	0.096	0.349
5	6	7	0.304	0.439
6	6	4	0	0.398
7	3	4	0	0
8	5	7	0	0.169
9	3	3	0	0
10	3	5	0	0.232
11	3	7	0	0.467
12	3	9	0.343	0.957

Berdasarkan tabel dari uji sampel tersebut, dapat dilihat bahwa algoritma *brute force* secara konsisten memiliki *elapsed time* yang lebih baik dibandingkan algoritma *divide and conquer*. Hal ini sesuai dengan prediksi secara teoritis dari kompleksitas waktu masing-masing algoritma dengan kompleksitas dari algoritma *brute force* adalah $O(2^k n^2)$, sedangkan kompleksitas waktu dari algoritma *divide and conquer* adalah $O(k2^k n^2)$. Terdapat perbedaan sebesar orde k pada kedua algoritma tersebut.

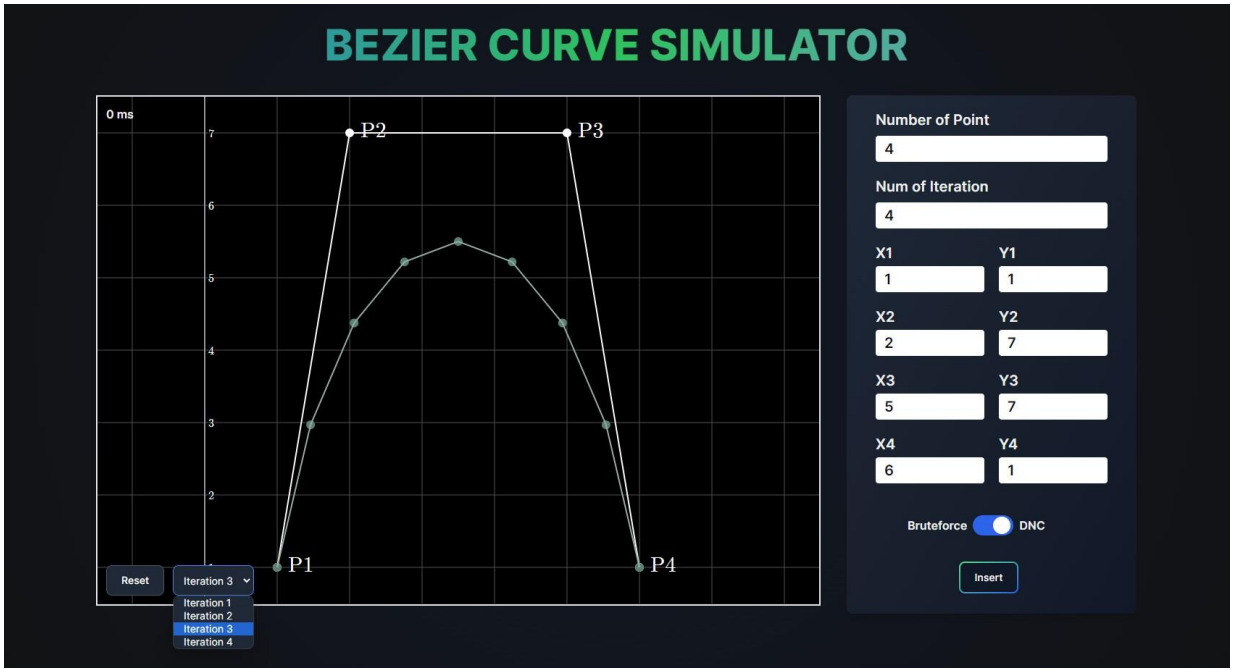
BAB VI
IMPLEMENTASI BONUS

A. Bonus Generalisasi Kurva

Analisis dan implementasi dari *bezier curve* dengan *n* buah titik kontrol sudah dijelaskan pada bab-bab sebelumnya dimana penulis melakukan analisis untuk kurva secara *general*.

B. Bonus Visualisasi Proses Pembuatan Kurva

Program ini mengimplementasikan visualisasi proses pembuatan kurva sehingga pengguna dapat memilih untuk melihat iterasi pembentukan kurva hingga iterasi *k*. Berikut adalah contoh tampilan dari bonus ini.



BAB VII
KESIMPULAN

Berdasarkan analisis secara teoritis, ditemukan bahwa kompleksitas waktu dari algoritma *brute force* adalah $O(2^k n^2)$, sedangkan kompleksitas waktu dari algoritma *divide and conquer* adalah $O(k2^k n^2)$. Jadi, algoritma *brute force* memiliki kompleksitas waktu yang lebih baik dibandingkan algoritma *divide and conquer*. Hasil tersebut juga didukung oleh data hasil pengujian dari 12 titik sampel dimana algoritma *brute force* secara konsisten memiliki *elapsed time* yang lebih baik dibandingkan algoritma *divide and conquer*.

LAMPIRAN

1. Checklist

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.	✓	

2. Pranala *Repository*

https://github.com/ninoaddict/Tucil2_13522068