

LAPORAN TUGAS KECIL II
IF2211 STRATEGI ALGORITMA

**MEMBANGUN KURVA BEZIER DENGAN ALGORITMA TITIK
TENGAH BERBASIS DIVIDE AND CONQUER**



Disusun oleh :

13522021 Filbert
13522045 Elbert Chailes

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG 2023

BAB I

PENDAHULUAN

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 realistik, 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.

$$Q_0 = B(t) = (1 - t)P_0 + t.P_1, \quad t \in [0, 1]$$

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 kuadratik** terhadap titik P_0 dan P_2 . Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1 - t) P_0 + t \cdot P_1, \quad t \in [0, 1]$$

$$Q_1 = B(t) = (1 - t) P_1 + t \cdot P_2, \quad t \in [0, 1]$$

$$R_0 = B(t) = (1 - t) Q_0 + t \cdot Q_1, \quad t \in [0, 1]$$

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

$$R_0 = B(t) = (1 - t)^2 P_0 + (1 - t) t \cdot P_1 + t^2 \cdot P_2, \quad t \in [0, 1]$$

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 kuatrik, dan seterusnya. Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik.

Oleh karena itu, pada tugas kecil II IF2211 Strategi Algoritma kali ini, kami akan mencari titik-titik yang dapat menyusun kurva Bézier dengan titik mulai, titik kontrol, dan titik akhir kurva yang telah ditentukan dengan menggunakan algoritma *Divide and Conquer*. Kami juga akan melakukan perbandingan algoritma untuk mencapai tujuan tersebut antara algoritma *bruteforce* dengan algoritma *divide and conquer*. Hal yang kami bandingkan antara kedua algoritma tersebut adalah waktu eksekusinya untuk menentukan algoritma yang mana memiliki waktu eksekusi lebih cepat untuk menentukan algoritma mana yang lebih mangkus. Kami juga melakukan generalisasi khusus untuk algoritma *divide and conquer* yang tidak hanya dapat menerima satu titik kontrol, melainkan sebanyak n kontrol poin sehingga lebih fleksibel dalam melakukan penggambaran kurva Bézier. Maka dari itu, permasalahan untuk kurva Bézier linear, kuadratik, kuatrik, dan seterusnya dapat diselesaikan dengan menggunakan program yang telah kami buat menggunakan algoritma *divide and conquer*.

BAB II

ANALISIS DAN IMPLEMENTASI ALGORITMA PADA KURVA BEZIER

2.1 Analisis dan Implementasi Algoritma *Bruteforce* pada kurva Bézier

Algoritma *bruteforce* adalah pendekatan pemrograman yang sederhana namun efektif untuk menyelesaikan masalah dengan mencoba semua kemungkinan solusi hingga solusi yang tepat ditemukan dan tidak ada upaya untuk menghindari pengujian kandidat yang tidak mungkin menjadi solusi. Pendekatan ini sering digunakan ketika tidak ada solusi yang lebih efisien atau ketika iterasi atau perulangannya cukup kecil sehingga pendekatan ini masih praktis. Algoritma *bruteforce* sering digunakan meskipun sering kali tidak praktis karena inefisiensi waktu yang diperlukan.

Dalam pembuatan kurva Bezier, algoritma *bruteforce* dapat digunakan untuk menghitung titik-titik pada kurva dengan mencoba berbagai nilai parameter t dari 0 sampai 1 dan menghitung posisi titik yang sesuai. Nilai parameter t ini biasanya diambil dalam interval yang sama, sehingga kurva dapat digambarkan dengan presisi seperti yang diinginkan. Rumus yang digunakan untuk menghitung posisi titik pada kurva adalah di bawah ini :

$$R_0 = B(t) = (1 - t)^2 P_0 + (1 - t) t. P_1 + t^2. P_2, \quad t \in [0, 1]$$

Dimana:

- $B(t)$ adalah posisi titik pada kurva untuk nilai parameter t.
- P_0 adalah titik awal (start point).
- P_1 adalah titik kontrol (control point).
- P_2 adalah titik akhir (end point).
- t adalah parameter yang nilainya berada dalam rentang 0 hingga 1.

Dalam implementasi algoritma *bruteforce* untuk kurva Bézier, kita akan menghitung posisi titik-titik pada kurva dengan cara iteratif untuk setiap nilai t yang diambil dari interval yang telah ditentukan. Misalnya, jika kita ingin menghitung 100 titik pada kurva, maka kita akan menghitung posisi titik untuk $t = 0, t = 0.01, t = 0.02$, dan seterusnya hingga $t = 1$.

Proses ini akan menghasilkan serangkaian titik yang jika digambarkan akan membentuk kurva Bézier. Dengan meningkatkan jumlah titik yang dihitung (misalnya dari 100 titik menjadi 1000 titik), presisi kurva yang dihasilkan akan semakin tinggi, tetapi juga akan memerlukan waktu komputasi yang lebih lama.

2.2 Analisis dan Implementasi Algoritma *Divide and Conquer* pada kurva Bézier

Algoritma *divide and conquer* berasal dari dua kata, yaitu *divide* yang berarti persoalan yang besar dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, tetapi ukurannya lebih kecil dan *conquer* yang berarti upa-persoalan masing-masing diselesaikan secara rekursif jika masih berukuran besar dan diselesaikan secara langsung jika masalah sudah berukuran kecil (atau mencapai *base case* dalam kasus rekursi). Dengan algoritma ini, maka hasil solusi masing-masing upa-persoalan hasil *conquer* akan digabung menjadi satu solusi atas persoalan semula melalui *combine*.

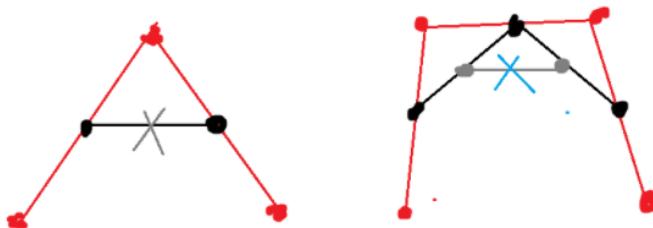
Dengan konsep seperti yang telah dijelaskan, maka sangat memungkinkan untuk menerapkan algoritma ini untuk menentukan titik-titik yang dapat menyusun kurva Bézier dengan menggunakan titik-titik kontrol yang telah ditentukan. Konsep ini juga dapat menyelesaikan persoalan penyusunan kurva Bézier sebanyak n titik kontrol karena penerapannya yang fleksibel dengan menggunakan algoritma *divide and conquer*. Prosedur untuk menentukan titik-titik untuk menyusun kurva Bézier pada persoalan yang telah disebutkan adalah sebagai berikut.

1. **Conquer** : Pertama, lakukan *conquer (solve)* terlebih dahulu terhadap titik-titik kontrol poin yang diterima dengan mencari titik tengah dari titik tengah yang terbentuk dari garis-garis yang menyambungkan antar titik kontrol. Titik tengah dari titik tengah yang dimaksud dapat dilihat pada ilustrasi yang terdapat pada **Gambar 2.2.1.** yang mana tanda X pada gambar menunjukkan hasil *conquer* yang didapatkan yaitu berupa titik tengah dari titik tengah yang terdapat pada titik kontrol yang tersedia.

Sebagai catatan analisis, bahwa pada tahap *conquer* ini dilakukan perulangan sebanyak dua kali atau bisa dikatakan menerapkan strategi *bruteforce* untuk menentukan titik tengah dari titik tengah yang tidak diketahui sesuai dengan banyaknya titik kontrol yang diberikan. Maka dari itu, untuk menentukan dua titik tengah dari titik-titik tengah

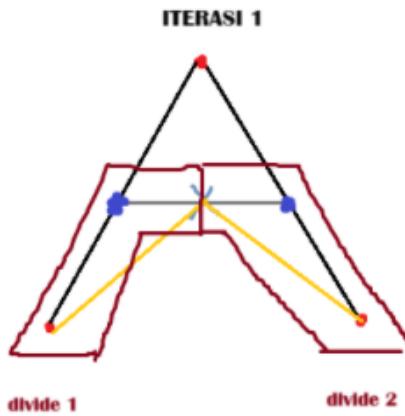
titik kontrol yang kemudian akan dicari titik tengah yang menjadi titik pembagi menjadi dua upa-persoalan dilakukan implementasi algoritma berupa perulangan sebanyak dua kali, dengan perulangan untuk menentukan titik tengah sebanyak jumlah titik kontrol dikurang (termasuk dengan titik mulai dan titik berakhir) dengan 1. Maka,

banyak langkah untuk tahapan *conquer* ini saja adalah $\left(\sum_{2}^{n-1} (n) \right) + 1$ langkah sehingga kompleksitas algoritmanya dapat dipastikan adalah $O(n^2)$.



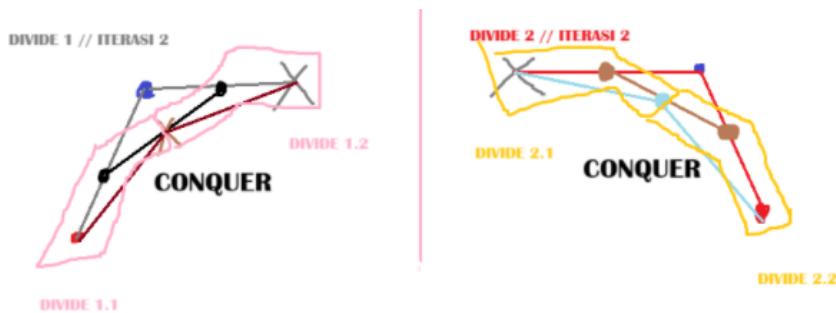
Gambar 2.2.1. Hasil *conquer* berupa titik tengah dari titik-titik tengah titik kontrol

2. **Divide :** Kemudian, hasil yang didapatkan berupa titik tengah tersebut yang ditandai dengan X pada **Gambar 2.2.1.** dapat digunakan sebagai titik pembelahan antara kurva tersebut sehingga persoalan terbagi menjadi upa-persoalan yang lebih kecil. Hasil pembagian antara kedua akan menghasilkan hasil *divide* yang dapat terlihat pada ilustrasi yang terdapat pada **Gambar 2.2.2.** yang mana ditandai dengan pembagian yang jelas dengan menggunakan tinta berwarna merah yang berhasil membagi persoalan utama menjadi dua upa-persoalan yang masing-masing mengandung tiga titik kontrol juga sehingga dapat memanfaatkan ulang fungsi rekursi sedang digunakan.



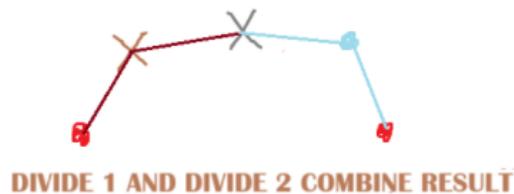
Gambar 2.2.2. Hasil *divide* yang dilakukan terhadap persoalan utama dengan memanfaatkan hasil *conquer*

3. **Conquer upa-persoalan** : Kemudian, masing-masing upa-persoalan atau hasil *divide* yang dihasilkan seperti pada **Gambar 2.2.2.** dilakukan *solve* atau *conquer* lagi sama seperti *conquer* yang dilakukan pada poin pertama. Hal tersebut dengan tujuan untuk menentukan titik tengah dari titik-titik tengah titik kontrol. *Conquer* pada upa-persoalan 3 ini tidak akan dilakukan jika jumlah iterasi yang hendak dilakukan oleh pengguna sudah tercapai sehingga *conquer* ini bersifat opsional atau dinamis tergantung dengan jumlah iterasi yang dibutuhkan oleh pengguna. Tentu saja, semakin banyak iterasi akan menyebabkan hasil penyusunan kurva Bézier semakin bagus karena konsentrasi titik yang didapatkan semakin tinggi. Ilustrasi *conquer upa-persoalan* dapat dilihat pada **Gambar 2.2.3.**



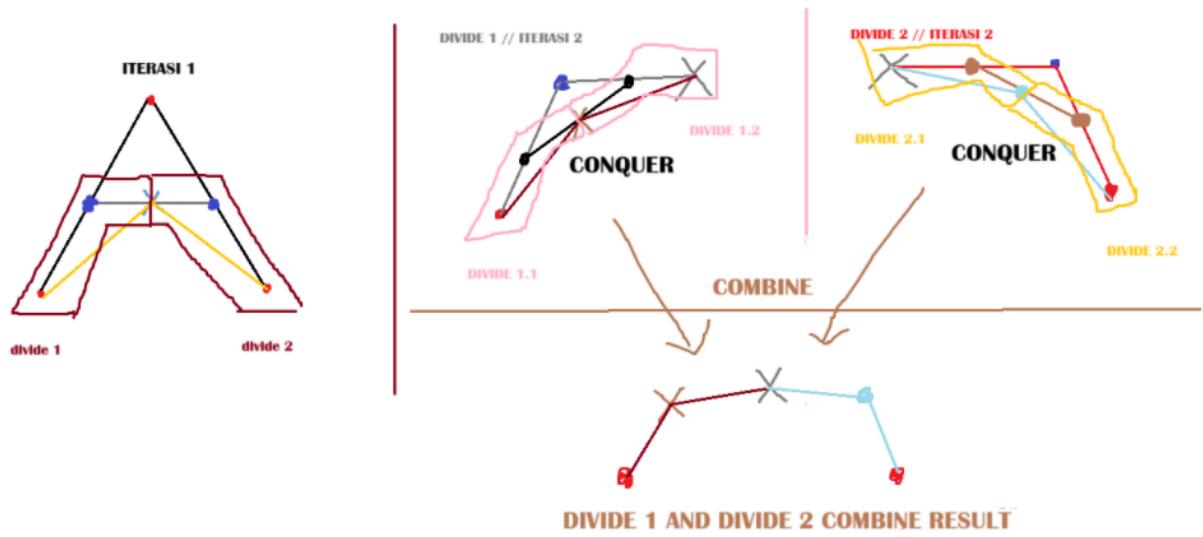
Gambar 2.2.3. Ilustrasi *conquer upa-persoalan* yang dilakukan oleh masing-masing *divide* yang dihasilkan daripada *divide* pada poin kedua

4. **Combine** : Pada akhirnya, setelah mencapai banyak iterasi yang telah ditentukan, yang mana pada ilustrasi ini diasumsikan bahwa iterasi masukan yang diberikan pengguna adalah dua. Maka, akan dilakukan tahap akhir dari algoritma *divide and conquer*, setelah dilakukan *divide*, maka akan dilakukan *combine* atau penggabungan untuk menghasilkan satu solusi yang tepat untuk menyelesaikan persoalan utama. Hasil dari *combine* dapat dilakukan dengan menyimpan titik-titik tengah yang telah dihasilkan pada setiap *conquer* yang telah dilakukan sehingga ketika kembali pada rekursi awal untuk menyelesaikan persoalan utama, titik tersebut masih tersimpan sehingga dapat divisualisasikan menjadi satu kurva Bézier yang telah melewati algoritma *divide and conquer*. Ilustrasi tahap *combine* dapat dilihat pada **Gambar 2.2.4**.



Gambar 2.2.4. Ilustrasi hasil *combine* antara solusi kedua upa-persoalan yang merupakan hasil *divide* persoalan utama

Dari penjelasan secara prosedural yang telah dijelaskan pada paragraf sebelumnya, ilustrasi prosedur algoritma *divide and conquer* untuk memperoleh kurva Bézier dari titik-titik kontrol yang diberikan dapat dilihat secara lengkap pada **Gambar 2.2.5**.



Gambar 2.2.5. Ilustrasi perlakuan *divide and conquer* secara lengkap yang meliputi *divide*, *combine*, dan *conquer* untuk menghasilkan solusi atas persoalan utama

BAB III

SOURCE CODE

3.1 File function.py

```
1 class Point:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"Point({self.x}, {self.y})"
8
9 def find_midpoint(point_a, point_b):
10    return Point(
11        (point_a.x + point_b.x) / 2,
12        (point_a.y + point_b.y) / 2
13    )
14
15 def generate_bezier_bruteforce(start_point, control_point, end_point, iterations):
16     num_points = 2 ** iterations
17
18     points = [start_point]
19     for i in range(1, num_points):
20         t = i / num_points
21         x = (1 - t)**2 * start_point.x + 2 * (1 - t) * t * control_point.x + t**2 * end_point.x
22         y = (1 - t)**2 * start_point.y + 2 * (1 - t) * t * control_point.y + t**2 * end_point.y
23         points.append(Point(x, y))
24     points.append(end_point)
25
26     return points
```

Gambar 3.1.1. Cuplikan kode program yang berisi kelas *Point*, fungsi *find_midpoint*, dan *generate_bezier_bruteforce*

```

● ● ●
1 def dnc_n_curve(control_points, depth, max_depth, points_list):
2     if depth < max_depth:
3         middle_point_useful_left = [control_points[0]]
4         middle_point_useful_right = [control_points[-1]]
5         middle_of_middle_point = [find_midpoint(control_points[i], control_points[i + 1])
6                                     for i in range(len(control_points) - 1)]
7
8         # Append the first and last midpoints to the left and right lists, respectively
9         middle_point_useful_left.append(middle_of_middle_point[0])
10        middle_point_useful_right.insert(0, middle_of_middle_point[-1])
11
12        # Calculate the successive midpoints until only one or two points remain
13        while len(middle_of_middle_point) > 2:
14            middle_of_middle_point = [find_midpoint(middle_of_middle_point[i], middle_of_middle_point[i + 1])
15                                      for i in range(len(middle_of_middle_point) - 1)]
16            middle_point_useful_left.append(middle_of_middle_point[0])
17            middle_point_useful_right.insert(0, middle_of_middle_point[-1])
18
19        # Handle the case when two points remain
20        if len(middle_of_middle_point) == 2:
21            new_middle_point = find_midpoint(middle_of_middle_point[0], middle_of_middle_point[1])
22            middle_point_useful_left.append(new_middle_point)
23            middle_point_useful_right.insert(0, new_middle_point)
24
25        # Recursive calls for the left and right halves
26        dnc_n_curve(middle_point_useful_left, depth + 1, max_depth, points_list)
27        points_list.append(new_middle_point)
28        dnc_n_curve(middle_point_useful_right, depth + 1, max_depth, points_list)

```

Gambar 3.1.2. Cuplikan kode program yang berisi fungsi *dnc_n_curve*

```

● ● ●
1 def generate_bezier_dnc_n_curve(start_point, control_points, end_point, max_iterations):
2     all_iterations_points = []
3
4     for iterations in range(1, max_iterations + 1):
5         bezier_curve_points = []
6         dnc_n_curve([start_point] + control_points + [end_point], 0, iterations, bezier_curve_points)
7         all_iterations_points.append([start_point] + bezier_curve_points + [end_point])
8
9     return all_iterations_points

```

Gambar 3.1.3. Cuplikan kode program yang berisi fungsi *generate_bezier_dnc_n_curve*

3.2 File main.py

```
● ● ●
1  from matplotlib.figure import Figure
2  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
3  from matplotlib.animation import FuncAnimation
4  from function import *
5  from tkinter import TK, Canvas, messagebox, Radiobutton, IntVar
6  import tkinter as tk
7  import time
8
9  def close_window(event=None):
10     window.destroy()
11
12 animation_in_progress = False
13 def generate_button():
14     global ani, execution_time_var, ax, canvas, bezier_line, control_line, animation_in_progress
15
16     if animation_in_progress:
17         messagebox.showerror("Error",
18             "An animation is already in progress. Please wait until it finishes before starting a new one.")
19         return
20
21     animation_in_progress = True
22
23     try:
24         num_points = int(number_of_points.get())
25         if num_points < 3:
26             messagebox.showerror("Error", "You must input minimum 3 points.")
27             return
28     except ValueError:
29         messagebox.showerror("Error", "Number of points must be an integer.")
30     return
31
32     try:
33         points_str = input_points.get().strip()
34         if not all(s.strip().count(',') == 1 for s in points_str.split(',')[:-1]):
35             raise ValueError("Invalid format for points."
36                             "Format should be (x,y),(x,y),... with no spaces between numbers and commas")
37
38         points_list = [Point(float(point_part.split(',')[0].strip(' ')), float(point_part.split(',')[1].strip(' ')))
39                         for point_part in points_str.split(',')[:-1]]
40         if len(points_list) != num_points:
41             raise ValueError("The number of input points does not match the number specified.")
42
43         start_point = points_list[0]
44         control_points = points_list[1:-1]
45         end_point = points_list[-1]
```

Gambar 3.2.1. Cuplikan kode program yang berisi fungsi *close_window* dan *generate_button* untuk keperluan GUI

```

● ● ●
1 except ValueError as e:
2     messagebox.showerror("Error", f"Invalid points input: {e}")
3     return
4
5     try:
6         iterations = int(number_of_iterations.get())
7     except ValueError:
8         messagebox.showerror("Error", "Number of iterations must be an integer.")
9     return
10
11 ax.clear()
12 ax.set_title('Bezier Curve Generator\nIterations more than 4 may take times')
13 ax.set_axis_on()
14
15 all_x = [p.x for p in points_list]
16 all_y = [p.y for p in points_list]
17 x_min, x_max = min(all_x), max(all_x)
18 y_min, y_max = min(all_y), max(all_y)
19 ax.set_xlim(x_min - 1, x_max + 1)
20 ax.set_ylim(y_min - 1, y_max + 1)
21
22 bezier_line, = ax.plot([], [], 'b-', label='Bezier Curve', lw=2, marker='o', markersize=5)
23
24 lines = []
25 def init():
26     global control_line, bezier_line
27     lines.clear()
28     control_line, = ax.plot([], [], 'ro--', label='Control Points', lw=1)
29     bezier_line, = ax.plot([], [], 'b-', label='Bezier Curve', lw=2, marker='o', markersize=5)
30     lines.append(control_line)
31     lines.append(bezier_line)
32     return lines
33
34 def animate_bruteforce(i):
35     global control_line, bezier_line, animation_in_progress
36     if i < len(x_points):
37         x = x_points[:i + 1]
38         y = y_points[:i + 1]
39         bezier_line.set_data(x, y)
40         control_line.set_data(x_control, y_control)
41
42     if i == len(x_points) - 1 :
43         animation_in_progress = False
44
45     return bezier_line,control_line
46
47 x_control = [p.x for p in [start_point] + control_points + [end_point]]
48 y_control = [p.y for p in [start_point] + control_points + [end_point]]
49 control_line, = ax.plot(x_control, y_control, 'ro--', label='Control Points', lw=1,markersize=5)
50

```

Gambar 3.2.2. Cuplikan kode program yang berisi fungsi *init* dan *animate_bruteforce* yang terdapat pada fungsi *generate_button*

```

● ○ ● ●
1 exec_time = 0
2     try:
3         if method_var.get() == 1:
4             if len(points_list) != 3:
5                 raise ValueError("Brute force method requires exactly 3 points.")
6
7         start_bf = time.time()
8         bezier_points = generate_bezier_bruteforce(start_point, control_points[0], end_point, iterations)
9         end_bf = time.time()
10
11        exec_time = (end_bf - start_bf) * 1000
12        x_points = [point.x for point in bezier_points]
13        y_points = [point.y for point in bezier_points]
14        ani = FuncAnimation(fig, animate_bruteforce, init_func=init, frames=len(x_points), interval=500, blit=True, repeat=False)
15    else:
16        control_line.set_data(x_control, y_control)
17        bezier_line.set_data([], [])
18        intermediate_lines = []
19        def animate_dnc(i):
20            global animation_in_progress
21            nonlocal intermediate_lines
22
23            for line in intermediate_lines:
24                line.remove()
25            intermediate_lines = []
26
27            if i < len(all_iterations_points):
28                current_points = all_iterations_points[i]
29
30                x = [p.x for p in current_points]
31                y = [p.y for p in current_points]
32                line, = ax.plot(x, y, 'b--', lw=1, marker='o', markersize=5)
33                intermediate_lines.append(line)
34
35            if i == len(all_iterations_points) - 1 or i == len(all_iterations_points):
36                animation_in_progress = False
37                line.set_color('b')
38                line.set_linestyle('-')
39                line.set_linenwidth(2)
40
41            return intermediate_lines
42
43        start_dnc = time.time()
44        all_iterations_points = generate_bezier_dnc_n_curve(start_point, control_points, end_point, iterations)
45        end_dnc = time.time()
46        exec_time = (end_dnc - start_dnc)*1000
47
48        ani = FuncAnimation(fig, animate_dnc, init_func=init, frames=len(all_iterations_points), interval=500, blit=True, repeat=False)
49    except ValueError as e:
50        messagebox.showerror("Error", str(e))
51        animation_in_progress = False
52        return
53
54    execution_time_var.set(f"Execution time: {exec_time:.4f} ms")
55
56    control_line.set_data(x_control, y_control)
57    ax.legend(handles=[control_line, bezier_line], loc='best')
58    window.ani = ani
59    canvas.draw()

```

Gambar 3.2.3. Cuplikan kode program pemanggilan algoritma *Bruteforce* dan *Divide and Conquer*

```

● ● ●
1 window = Tk()
2 window.geometry("1142x618")
3 window.configure(bg = "#130202")
4
5 canvas = Canvas(
6     window,
7     bg = "#130202",
8     height = 618,
9     width = 1142,
10    bd = 0,
11    highlightthickness = 0,
12    relief = "ridge"
13 )
14
15 canvas.place(x = 0, y = 0)
16 canvas.create_text(
17     400.0,
18     24.0,
19     anchor="nw",
20     text="Bezier Curve Generator",
21     fill="#1EFF6A",
22     font=("Times New Roman", 40 * -1)
23 )
24
25 canvas.create_rectangle(
26     570.0,
27     94.0,
28     571.0,
29     573.0,
30     fill="#1EFF6A",
31     outline="")
32
33
34 method_var = tk.IntVar(value=2) # Default to DNC
35
36 # Create radio buttons for method selection
37 brute_force_rb = Radiobutton(window, text="Brute Force", variable=method_var, value=1, bg="#000000", fg="#0c88e9",font=("Times New Roman", 20))
38 dnc_rb = Radiobutton(window, text="Divide and Conquer", variable=method_var, value=2, bg="#000000", fg="#0c88e9",font=("Times New Roman", 20))
39 brute_force_rb.place(x=90.0, y=90.0)
40 dnc_rb.place(x=290.0, y=90.0)

```

Gambar 3.2.4. Cuplikan kode program GUI untuk menentukan ukuran *canvas*

```

1 # Create 'Number of Points' label and entry
2 number_of_points_label = tk.Label(window, text="Number of Points :", bg="#130202", fg="#1EFF6A", font=("Times New Roman", 32 * -1))
3 number_of_points_label.place(x=98.0, y=131.0)
4 number_of_points = tk.Entry(window, bg="#1EFF6A",fg="#000000", insertbackground="#000000",font=("Times New Roman", 20))
5 number_of_points.place(x=98.0, y=177.0, width=377.0, height=52.0)
6
7 # Create 'Input Points (x,y)' label and entry
8 input_points_label = tk.Label(window, text="Input Points (x,y) :", bg="#130202", fg="#1EFF6A", font=("Times New Roman", 32 * -1))
9 input_points_label.place(x=98.0, y=240.0)
10 input_points = tk.Entry(window, bg="#1EFF6A",fg="#000000", insertbackground="#000000",font=("Times New Roman", 20))
11 input_points.place(x=98.0, y=289.0, width=377.0, height=52.0)
12
13 # Create 'Number of Iterations' label and entry
14 number_of_iterations_label = tk.Label(window, text="Number of Iterations :", bg="#130202", fg="#1EFF6A", font=("Times New Roman", 32 * -1))
15 number_of_iterations_label.place(x=98.0, y=354.0)
16 number_of_iterations = tk.Entry(window, bg="#1EFF6A",fg="#000000", insertbackground="#000000",font=("Times New Roman", 20))
17 number_of_iterations.place(x=98.0, y=403.0, width=377.0, height=52.0)
18
19 # Create 'Execution time' label and entry
20 execution_time_var = tk.StringVar(window)
21 execution_time_var.set("Execution time: ")
22 execution_times_label = tk.Label(window, textvariable=execution_time_var, bg="#130202", fg="#1EFF6A", font=("Times New Roman", 32 * -1))
23 execution_times_label.place(x=575.0, y=90.0)
24
25 # Create a Matplotlib figure and axes
26 fig = Figure()
27 ax = fig.add_subplot(111)
28
29 canvas = FigureCanvasTkAgg(fig, master=window)
30 mpl_canvas_widget = canvas.get_tk_widget()
31 mpl_canvas_widget.place(x=580, y=135, width=550, height=440)
32
33 button_border = tk.Frame(window, highlightbackground="#1EFF6A", highlightthickness=2, bd=0)
34 button_border.place(x=153.0, y=504.0, width=264.0, height=69.0)
35
36 generate_button = tk.Button(button_border, text="Generate", command=generate_button, bg="#1EFF6A", fg="#000000", font=("Times New Roman", 32))
37 generate_button.pack(expand=True, fill=tk.BOTH)
38
39 window.bind('<Escape>', close_window)
40 window.resizable(False, False)
41 window.mainloop()
42

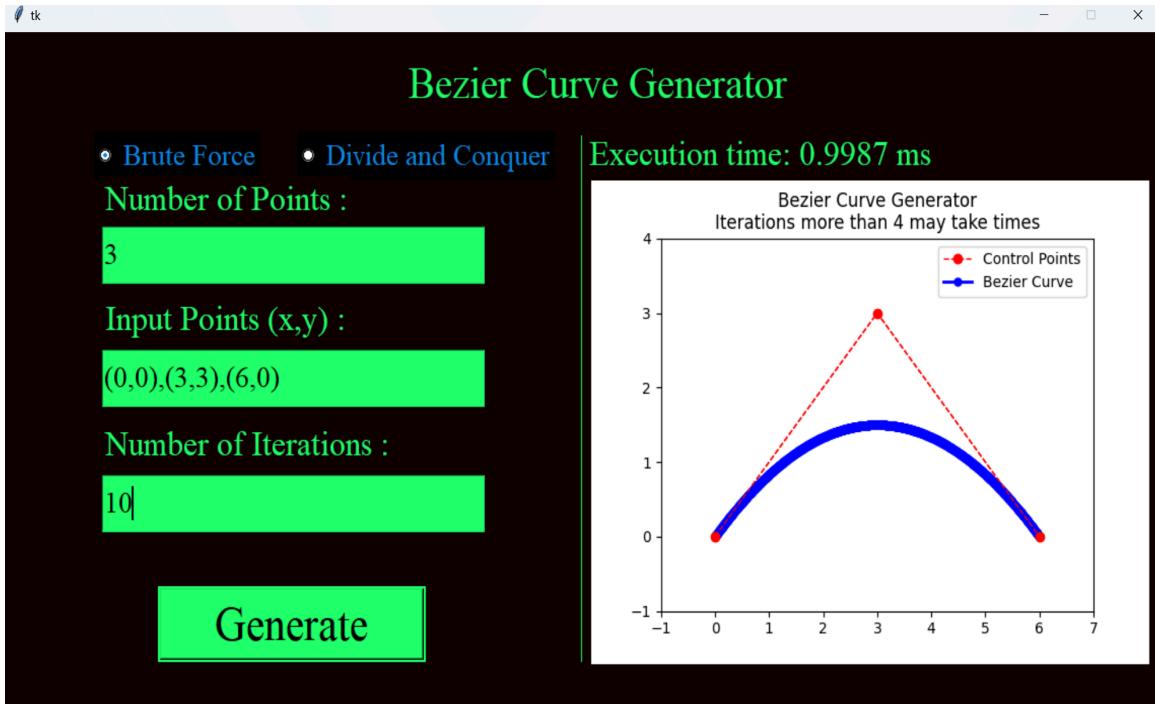
```

Gambar 3.2.5. Cuplikan kode program GUI untuk menampilkan titik-titik dan garis hasil dari titik-titik penyusun kurva Bézier yang dihasilkan oleh fungsi algoritma

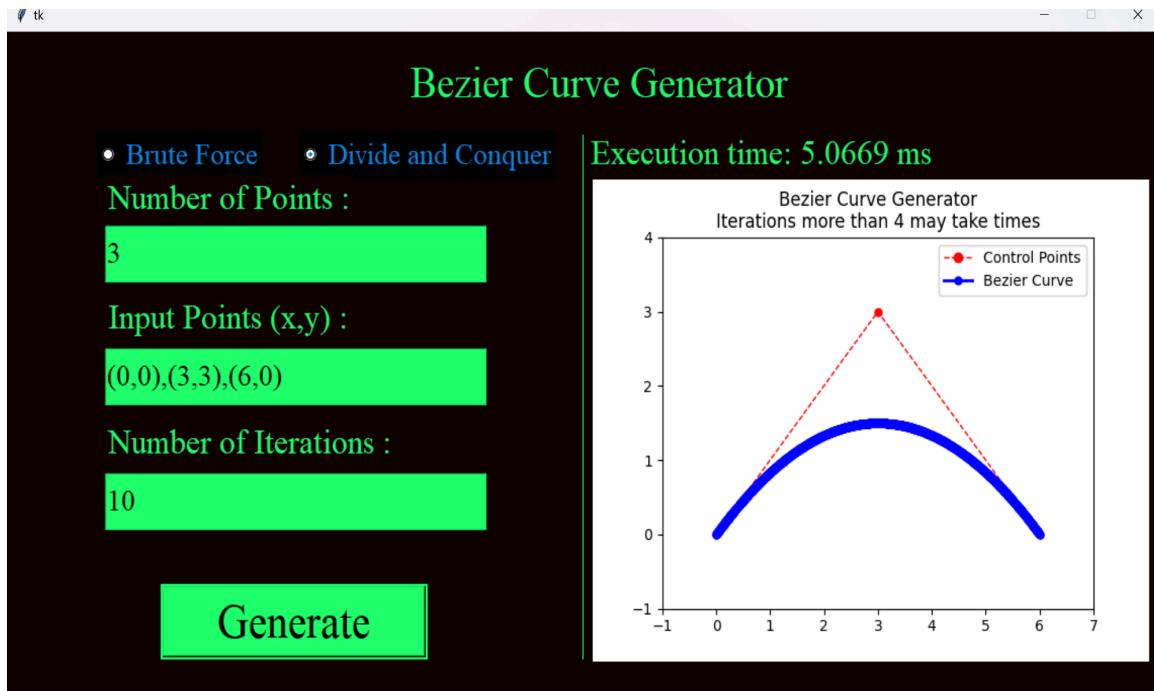
BAB IV

TESTING PROGRAM

4.1 Test Case 1



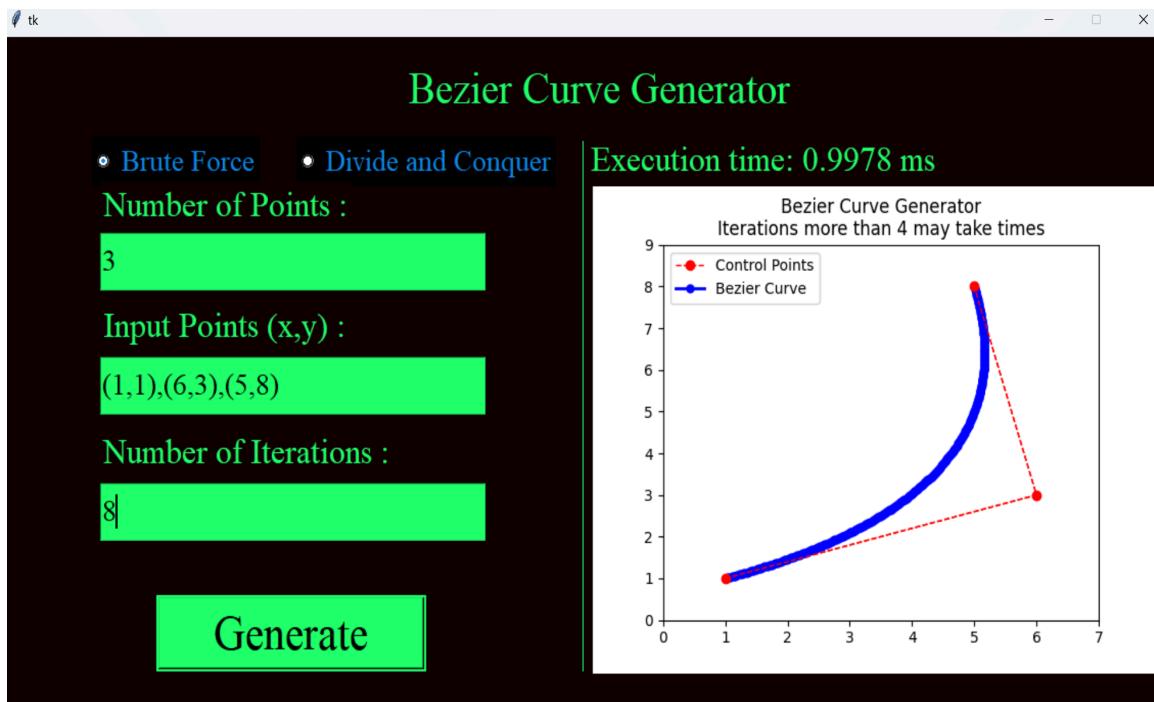
Gambar 4.1.1. Test Case 1 dengan Algoritma Brute force



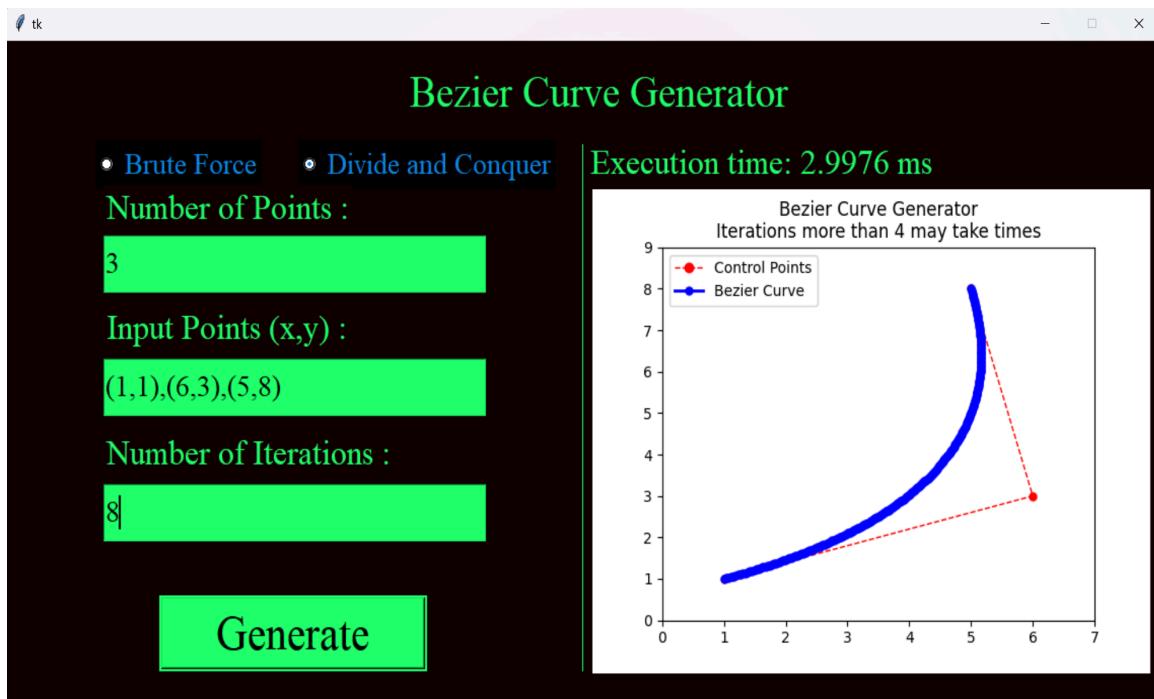
Gambar 4.1.2. Test Case 1 dengan Algoritma *Divide and Conquer*

Dengan menggunakan tiga titik kontrol yaitu (0,0), (3,3), dan (6,0), serta melakukan iterasi sebanyak 10 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat dibandingkan dengan algoritma *divide and conquer*.

4.2 Test Case 2



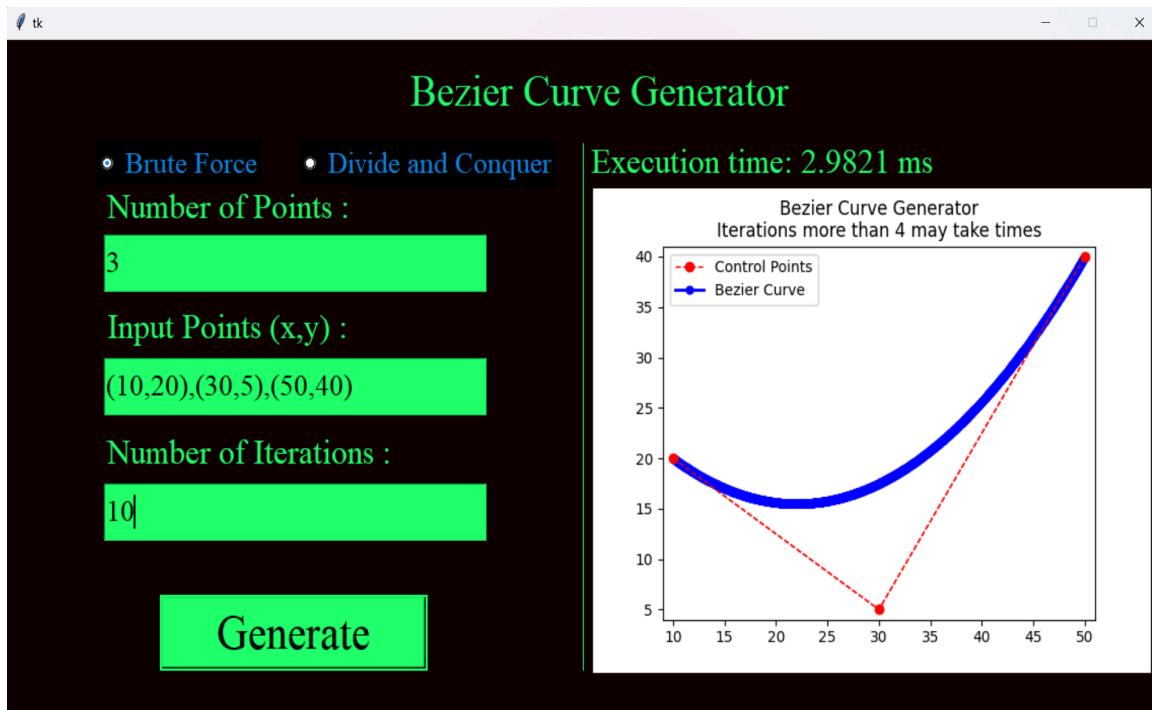
Gambar 4.2.1. Test Case 2 dengan Algoritma Brute force



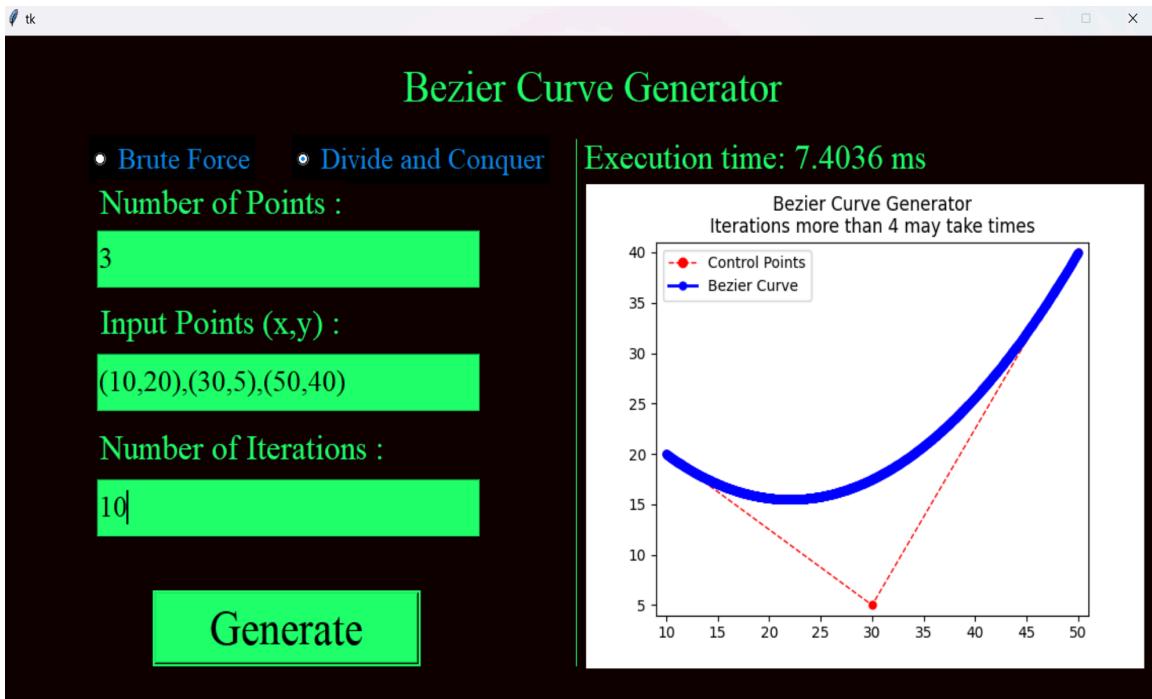
Gambar 4.2.2. Test Case 2 dengan Algoritma Divide and Conquer

Dengan menggunakan tiga titik kontrol yaitu (1,1), (6,3), dan (5,8), serta melakukan iterasi sebanyak 8 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat dibandingkan dengan algoritma *divide and conquer*.

4.3 Test Case 3



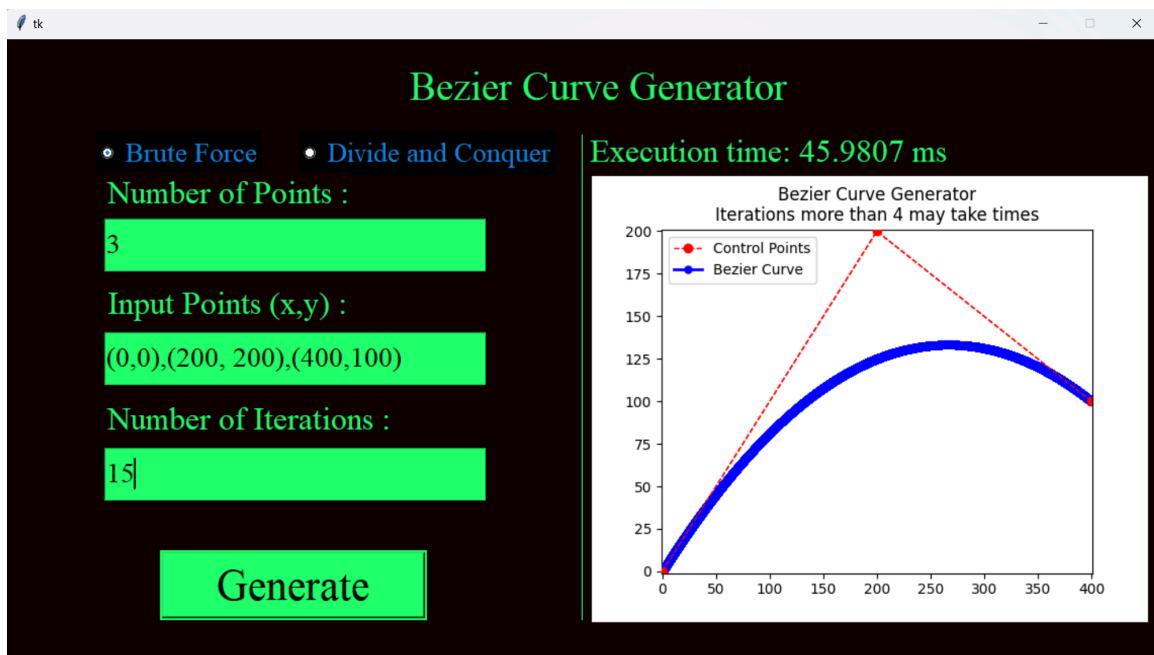
Gambar 4.3.1. Test Case 3 dengan Algoritma *Bruteforce*



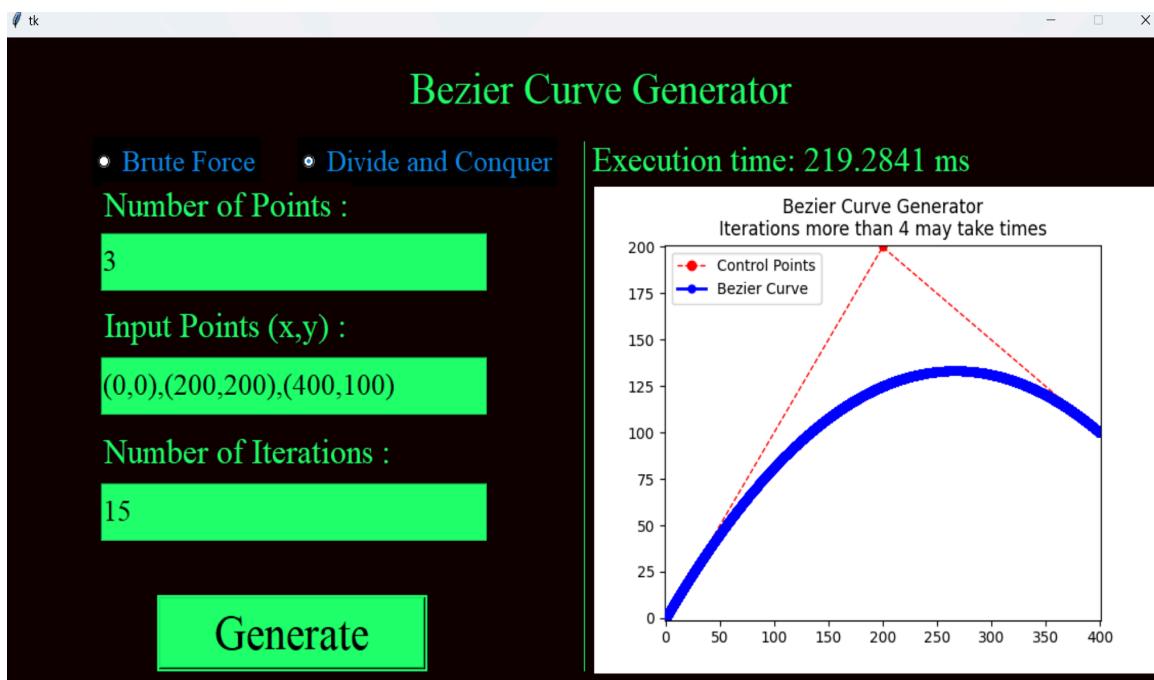
Gambar 4.3.2 Test Case 3 dengan Algoritma *Divide and Conquer*

Dengan menggunakan tiga titik kontrol yaitu (10,20), (30,5), dan (50,40), serta melakukan iterasi sebanyak 10 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat dibandingkan dengan algoritma *divide and conquer*.

4.4 Test Case 4



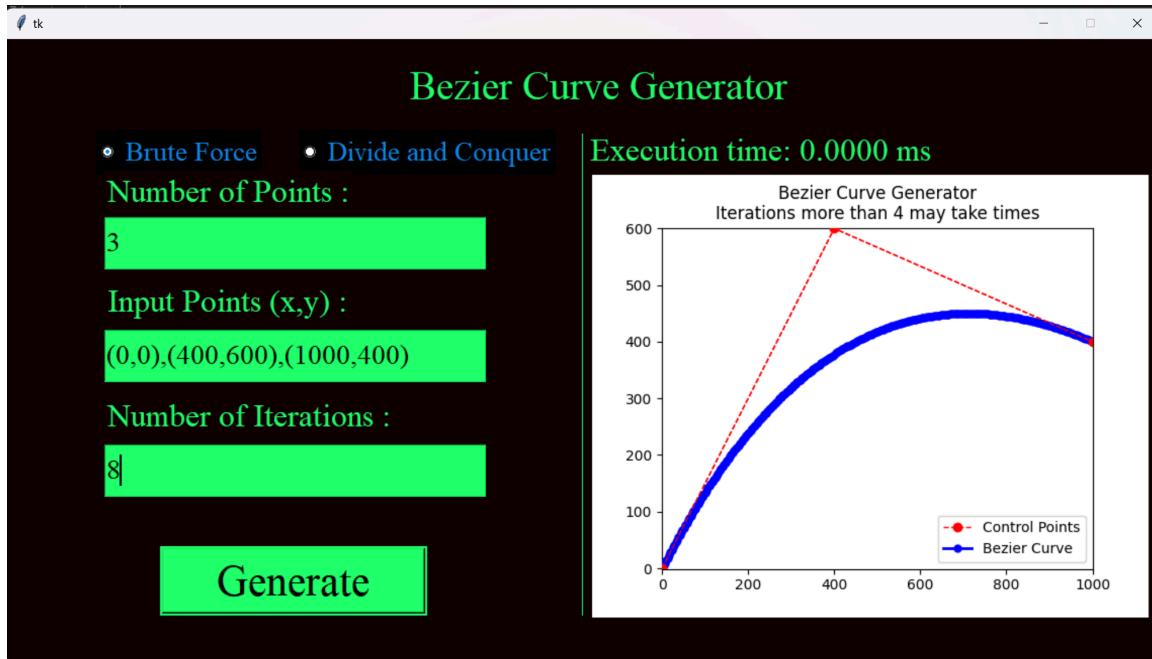
Gambar 4.4.1. Test Case 4 dengan Algoritma Bruteforce



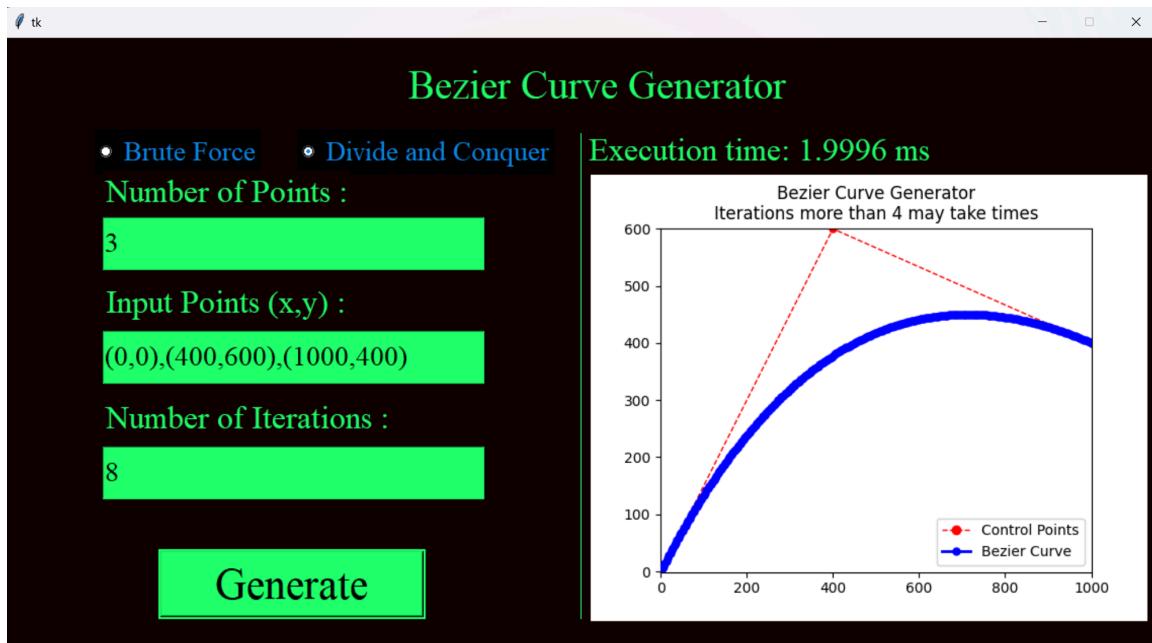
Gambar 4.4.2. Test Case 4 dengan Algoritma Divide and Conquer

Dengan menggunakan tiga titik kontrol yaitu (0,0), (200,200), dan (400,100), serta melakukan iterasi sebanyak 10 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat dibandingkan dengan algoritma *divide and conquer*.

4.5 Test Case 5



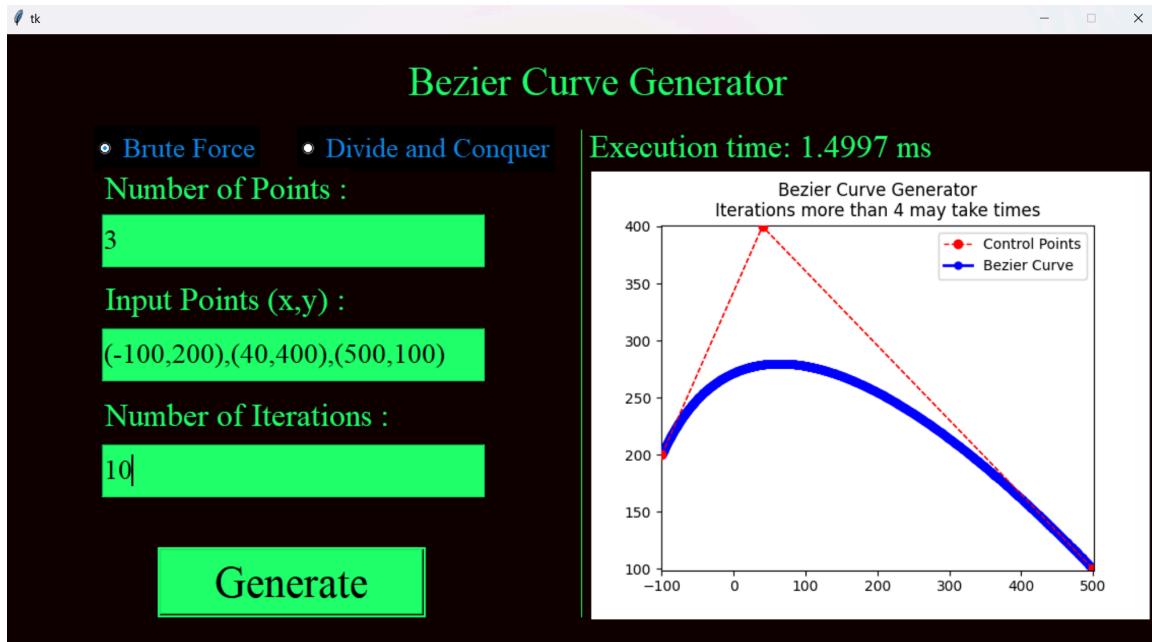
Gambar 4.5.1. Test Case 5 dengan Algoritma *Bruteforce*



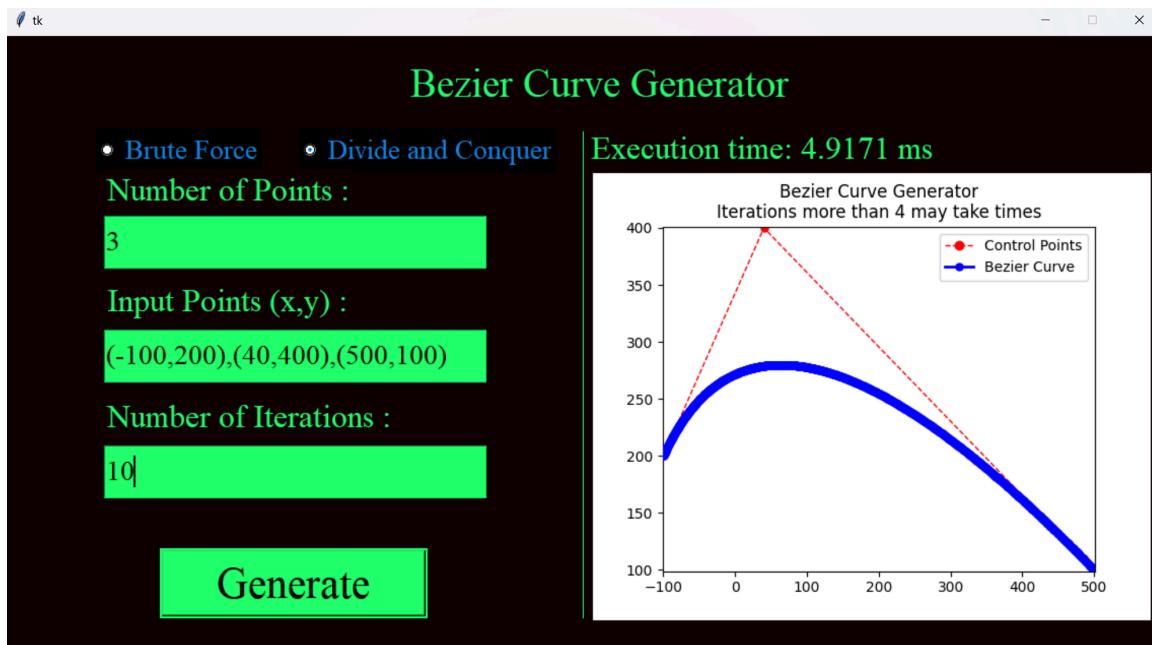
Gambar 4.5.2. Test Case 5 dengan Algoritma *Divide and Conquer*

Dengan menggunakan tiga titik kontrol yaitu (0,0), (400,600), dan (1000,400), serta melakukan iterasi sebanyak 8 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat dibandingkan dengan algoritma *divide and conquer*.

4.6 Test Case 6



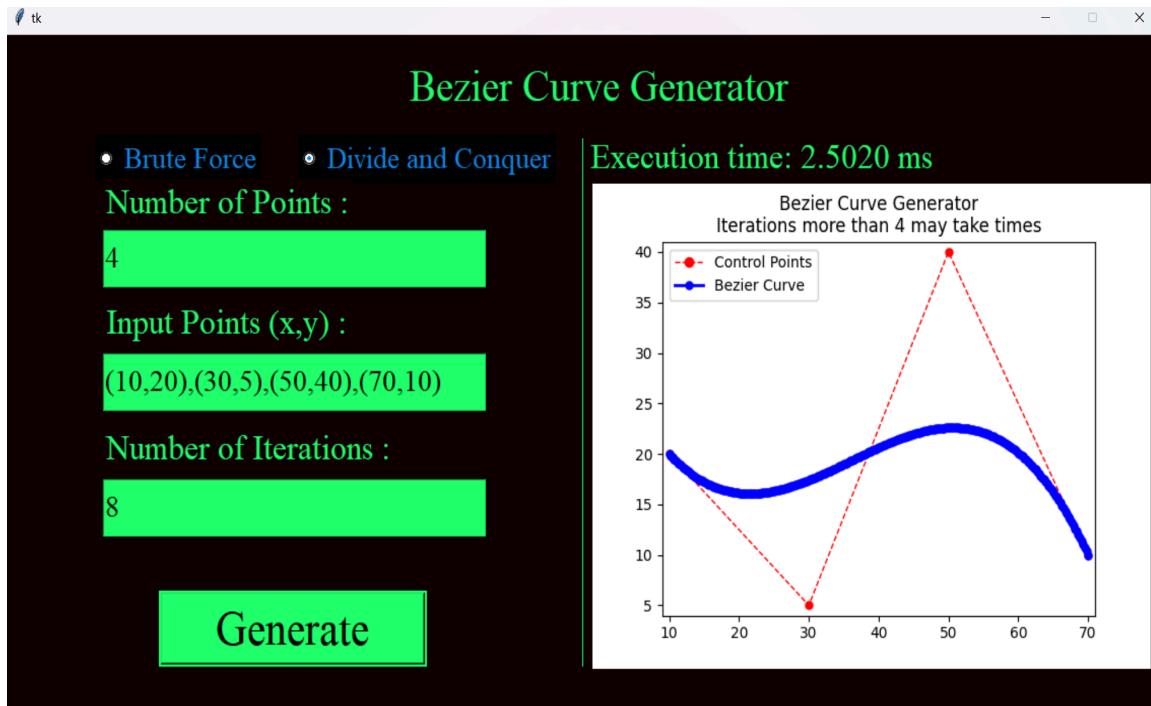
Gambar 4.6.1. Test Case 6 dengan Algoritma *Bruteforce*



Gambar 4.6.2. Test Case 6 dengan Algoritma *Divide and Conquer*

Dengan menggunakan tiga titik kontrol yaitu (-100,200), (40,400), dan (500,100), serta melakukan iterasi sebanyak 10 kali, terlihat bahwa algoritma *bruteforce* mampu beroperasi lebih cepat 3 kali lipat dibandingkan dengan algoritma *divide and conquer*.

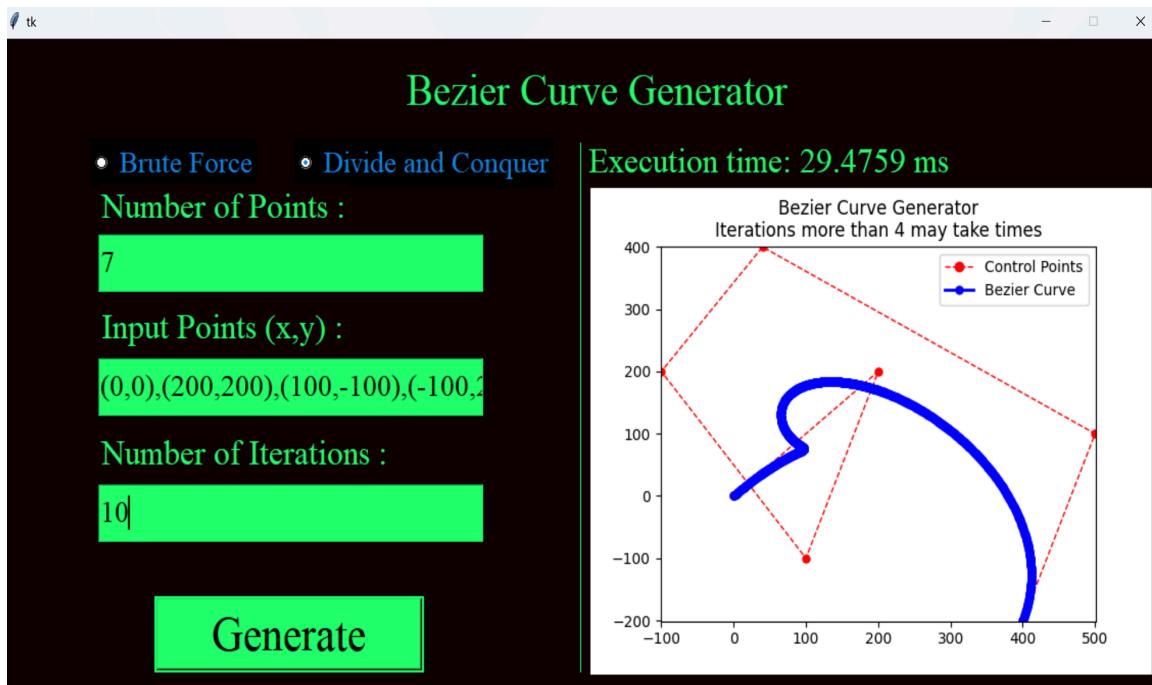
4.7 Test Case 7



Gambar 4.7. Test Case 7 dengan Algoritma *Divide and Conquer*

Dengan menggunakan empat titik kontrol yaitu (10,20), (30,5), (50,40) dan (70,10) serta melakukan iterasi sebanyak 8 kali, terlihat bahwa algoritma *divide and conquer* dapat beroperasi dengan cukup cepat.

4.8 Test Case 8



Gambar 4.8. Test Case 8 dengan Algoritma Divide and Conquer

Dengan menggunakan tujuh titik kontrol yaitu $(0,0)$, $(200,200)$, $(100,-100)$, $(-100,200)$, $(40,400)$, $(500,100)$ dan $(400,-200)$ serta melakukan iterasi sebanyak 10 kali, dibandingkan dengan **Gambar 4.7.** yang dimana hanya melakukan pengoperasian 4 titik dan iterasi sebanyak 8 kali, perbedaan waktu yang diperlukan juga bisa dikatakan sangat jauh karena perbedaanya sebesar 10 kali lipat.

BAB V

HASIL ANALISIS

5.1. Analisis Kompleksitas Algoritma *Bruteforce*

Terdapat aspek penting yang pasti akan dibahas jika berbicara tentang *bruteforce* yaitu kompleksitas algoritma yang diberikan. Dalam proses penyusunan kurva Bézier menggunakan algoritma *bruteforce*, total langkah yang dilakukan oleh algoritma tergantung oleh dua variabel, yaitu banyaknya iterasi dan banyaknya titik kontrol yang akan diproses oleh algoritma ini. Sehingga, banyak langkah dapat diperkirakan dan digambarkan seperti berikut

$$T(n, i) = 2^i \cdot n^2$$

Dengan catatan bahwa n merupakan banyak titik kontrol dan i merupakan banyak iterasi yang dilakukan. Alasan dilakukan operasi n^2 adalah karena ingin mencari nilai segitiga pascal diperlukan kompleksitas waktu sebesar n^2 dan dikali 2 untuk mencari nilai koordinat x dan koordinat y untuk setiap iterasi yang dilakukan.

5.2. Analisis Kompleksitas Algoritma *Divide and Conquer*

Sebagai hasil akhir dari perjalanan algoritma *divide and conquer* untuk menyusun kurva Bezier, sama juga seperti algoritma *bruteforce* bahwa terdapat dua faktor yang mempengaruhi banyak langkah yang ditempuh untuk menyelesaikan persoalan ini, yaitu banyak iterasi (i) dan banyak titik kontrol (n). Banyak langkah yang ditempuh oleh algoritma ini dapat dituliskan seperti berikut.

$$\begin{aligned} T(n, i) &= a \text{ jika } i = 0 \\ T(n, i) &= 2 \cdot T(n, i - 1) + (n - 1) \cdot (n - 2) \\ T(n, i) &= 2 \cdot T(n, i - 1) + c \cdot n^2 \end{aligned}$$

Kita ambil kasus jika banyak iterasi yang dilakukan sebanyak i kali dan banyak kontrol sebanyak n titik kontrol. Maka, perhitungan akan berlangsung seperti berikut.

$$\begin{aligned} T(n, i) &= 2 \cdot (2 \cdot T(n, i - 2) + c \cdot n^2) + c \cdot n^2 \\ T(n, i) &= 2 \cdot (2 \cdot (2 \cdot T(n, i - 3) + c \cdot n^2)) + 2c \cdot n^2 + c \cdot n^2 \\ T(n, i) &= 8 \cdot T(n, i - 3) + 4c \cdot n^2 + 2c \cdot n^2 + c \cdot n^2 \end{aligned}$$

$$T(n, i) = 8 \cdot T(n, i - 3) + 4c \cdot n^2 + 2c \cdot n^2 + c \cdot n^2$$

$$T(n, i) = 2^i \cdot T(n, 0) + (2^i - 1) \cdot c \cdot n^2$$

$$T(n, i) = 2^i \cdot a + (2^i - 1) \cdot c \cdot n^2$$

Sebagai informasi tambahan untuk menanggapi hasil implementasi atau percobaan yang dilakukan pada *Test case 7* dan *Test case 8*, hal tersebut terjadi disebabkan karena banyaknya langkah yang dilakukan oleh algoritma *divide and conquer* berkembang secara eksponensial bergantung dengan banyak titik kontrol n dan jumlah iterasi i . Hal tersebut ditandai dengan kompleksitas algoritma yang berkembangan sebesar $2^i * n^2$ langkah untuk setiap perubahan nilai variabelnya.

5.3. Analisis Perbandingan Kompleksitas Algoritma antara Algoritma *Bruteforce* dengan Algoritma *Divide and Conquer* dengan 3 titik kontrol

Jika dilakukan algoritma *bruteforce* untuk kasus ketika jumlah kontrol poin (n) sebanyak tiga dan iterasi (i) yang dilakukan sebanyak dua kali. Maka, banyak langkah yang dilakukan oleh algoritma *bruteforce* tersebut adalah sebagai berikut.

$$T(n = 3, i = 2) = 2^2 \cdot 3^2 = 36 \text{ langkah}$$

Kemudian, jika diambil kasus titik kontrol yang digunakan adalah sebanyak tiga titik kontrol dan banyak iterasi yang dilakukan adalah dua. Maka, banyak langkah yang dilakukan algoritma *divide and conquer* ini dapat dilihat pada proses berikut.

$$T(n = 3, i = 2) = 4 \cdot a + 3 \cdot c \cdot n^2 \approx 3n^2 = 27 \text{ langkah}$$

Dari hasil perhitungan analisis yang dilakukan secara teoretis berdasarkan kedua perhitungan banyak langkah di atas, dapat disimpulkan bahwa algoritma *bruteforce* menempuh langkah yang lebih banyak untuk menyelesaikan sebuah persoalan dibandingkan algoritma

divide and conquer. Namun, jika dilihat dari waktu eksekusi yang dilakukan pada **Gambar 4.2.1.** dan **Gambar 4.2.2.** didapatkan informasi bahwa algoritma *bruteforce* lebih cepat dibandingkan dengan algoritma *divide and conquer*. Kedua hasil analisis ini berkontradiksi antara analisis secara teoretis dan praktik melalui kompilasi program.

Kontradiksi ini dapat terjadi dikarenakan untuk ukuran n atau titik kontrol yang berjumlah kecil karena hasil perhitungan kompleksitas algoritma kedua algoritma yang dilakukan diatas merupakan hasil generalisasi kompleksitas algoritma sehingga tidak akurat untuk ukuran data yang kecil dan analisis kompleksitas tersebut hanya bersifat untuk mengukur secara perkiraan untuk jumlah data yang besar. Oleh karena itu, algoritma *bruteforce* dapat dipastikan akan lebih lambat waktu eksekusinya dibandingkan algoritma *divide and conquer* untuk ukuran titik kontrol yang besar.

Dari hasil analisis tersebut pula, didapatkan bahwa untuk memenuhi kebutuhan penyusunan persoalan kurva Bezier pengguna, dapat digunakan algoritma *divide and conquer* untuk banyak titik kontrol n yang banyak karena sifatnya lebih fleksibel dan memiliki kompleksitas algoritma yang lebih baik dibandingkan algoritma *bruteforce*. Namun, jika sebatas untuk banyak titik kontrol (n) yang sedikit, maka algoritma *bruteforce* dapat dijadikan pertimbangan karena memiliki waktu eksekusi yang lebih baik dibandingkan algoritma *divide and conquer*.

Dengan demikian, tidak dapat disimpulkan mana yang lebih baik diantara kedua algoritma, algoritma *divide and conquer* maupun algoritma *bruteforce*. Namun, dari segi fleksibilitas dan kompleksitas algoritma dengan jumlah titik kontrol (n) yang besar, algoritma *divide and conquer* akan lebih unggul dibandingkan dengan algoritma *bruteforce*. Namun, jika kebutuhan pengguna hanya berupa penyusunan kurva dengan 3 titik kontrol, algoritma *bruteforce* juga dapat dijadikan sebagai pilihan.

BAB VI

IMPLEMENTASI BONUS

6.1. Bonus Generalisasi Algoritma *divide and conquer* dengan N titik kontrol

Sebenarnya, konsep yang diterapkan untuk menerapkan algoritma *divide and conquer* ini sama seperti *divide and conquer* yang dilakukan terhadap jumlah titik tertentu, atau kurva Bezier dengan satu titik kontrol. Namun, perbedaannya hanya terdapat pada proses *conquer*-nya karena semakin banyak titik kontrol, maka akan semakin banyak pula proses yang dilakukan untuk menemukan titik tengah akhir dari titik-titik tengah dari titik kontrol untuk menyusun kurva Bezier. Cara untuk menyimpan titik tengah yang disimpan sebagai result dan implementasi rekursif sama seperti algoritma *divide and conquer* dengan titik tertentu, yang mana bisa dilihat perbandingannya pada fungsi bernama *dnc_n_curve*.

Alasan bisa ditemukan cara *bruteforce* untuk menentukan titik tengah akhir dari titik-titik tengah dari titik kontrol untuk menyusun kurva Bezier adalah karena ditemukan pola seperti yang dijelaskan pada Bab II analisis algoritma *divide and conquer*, yaitu langkah yang diperlukan untuk menembus level akhir dapat ditemukan dengan pendekatan $\left(\sum_{2}^{n-1} (n) \right) + 1$.

Maka dari itu, didapatkan algoritma yang dapat menyelesaikan permasalahan *conquer* tersebut. Sisanya, untuk *divide* dan *combine* sama seperti algoritma untuk menyusun kurva Bezier dengan titik kontrol tertentu.

6.2. Bonus Visualisasi Proses Pembentukan Kurva

Animasi kurva Bézier pada tugas ini diimplementasikan dengan menggunakan library matplotlib dan tkinter. Matplotlib menyediakan fungsionalitas visualisasi data, dan tkinter adalah library GUI standar Python yang digunakan untuk membuat aplikasi desktop.

Proses animasi diawali dengan pengambilan input dari pengguna melalui antarmuka grafis tkinter. Pengguna dapat memasukkan jumlah titik kontrol, koordinat titik kontrol, dan jumlah iterasi yang diinginkan untuk membentuk kurva Bézier. Setelah tombol *Generate* ditekan, fungsi *generate_button* akan dipanggil.

Dalam fungsi *generate_button*, data yang diinput diolah untuk menghasilkan titik-titik kurva Bézier. Ada dua metode yang dapat dipilih: *bruteforce* dan *divide and conquer*. Metode *bruteforce* menghitung titik-titik kurva dengan mencoba setiap kemungkinan secara langsung, sedangkan metode *divide and conquer* memecah masalah menjadi upa persoalan yang lebih kecil, memprosesnya secara rekursif, dan menggabungkannya kembali untuk mendapatkan hasil akhir.

Setelah titik-titik kurva dihitung, animasi dibuat menggunakan *FuncAnimation* dari *matplotlib*. Ini memungkinkan kita untuk menganimasikan proses pembentukan kurva Bézier langkah demi langkah. Animasi dimulai dengan menampilkan titik-titik kontrol dan menampilkan titik-titik kontrol dari setiap iterasi. Kemudian, pada setiap frame, bagian berikut dari kurva Bézier ditambahkan. Jika metode *Divide and Conquer* dipilih, setiap iterasi rekursif yang menampilkan titik-titik kontrol yang terlibat dalam membagi kurva akan ditunjukkan satu per satu, yang memberikan visualisasi yang jelas tentang bagaimana kurva tersebut dibangun secara bertahap.

Waktu eksekusi dari proses pembuatan kurva juga diukur dan ditampilkan, memberikan informasi tambahan mengenai efisiensi dari metode yang dipilih. Setelah semua iterasi animasi selesai, kurva Bézier akhir ditampilkan sebagai hasil akhir.

Perlu juga diperhatikan bahwa terdapat format tertentu yang perlu dipenuhi guna membentuk visualisasi kurva Bézier dan dapat menjalankan program dengan sempurna. Penginputan titik kontrol dan iterasi harus memenuhi format berikut:

- Input titik kontrol dalam format $(x_1,y_1),(x_2,y_2),(x_3,y_3),\dots,(x_n,y_n)$ tanpa spasi dibelakang maupun di depan inputan dengan x_n dan y_n merupakan integer.
- Panjang dari control points harus minimal berjumlah 3 dan harus berupa integer
- Inputan jumlah iterasi harus berupa integer lebih dari 0 dan tanpa memiliki spasi didepan maupun dibelakang inputan jumlah iterasi.

Jika format diatas tidak dipenuhi, maka tampilan kurva tidak akan muncul, melainkan akan memberikan pesan error berdasarkan errornya masing-masing.

BAB VII

KESIMPULAN DAN SARAN

7.1. Kesimpulan

Dalam proses menyusun kurva Bezier, algoritma *Bruteforce* dan *Divide and Conquer* dapat dijadikan pilihan. Berdasarkan analisis dan implementasi yang dilakukan menggunakan kedua algoritma ini, terungkap bahwa pencarian solusi menggunakan algoritma *Divide and Conquer* cenderung lebih efisien berdasarkan analisis kompleksitas algoritma dibandingkan *Bruteforce* ketika titik kontrol yang digunakan untuk menyusun kurva Bezier berjumlah banyak. Namun, untuk kasus dengan jumlah titik kontrol yang relatif kecil seperti yang ditemukan pada contoh implementasi yang terdapat pada Bab yang telah dibahas, algoritma *Bruteforce* dapat memberikan solusi yang lebih cepat dibandingkan dengan *Divide and Conquer*, seperti ketika banyak titik kontrol adalah tiga. Oleh karena itu, meskipun algoritma *divide and conquer* mengungguli secara fleksibilitas dan performa berdasarkan kompleksitas algoritma, *bruteforce* juga dapat dipertimbangkan dalam penggunaan.

7.2. Saran

Dalam proses pembuatan tugas kecil ini, sangat disarankan untuk membaca spesifikasi tugas dengan teliti dan menyeluruh, terutama bagian yang berkaitan dengan bonus yang cenderung ambigu dan kurang jelas. Hal ini penting untuk menghindari kebutuhan akan revisi kode yang berulang-ulang, yang dapat menghambat proses penyelesaian tugas. Penting pula untuk memahami permasalahan secara mendalam sebelum memulai pengembangan program. Dengan demikian, tugas kecil dapat dikerjakan dengan lebih efektif dan tidak dilakukan pekerjaan yang berulang-ulang akibat ketidaktelitian pembacaan spesifikasi kebutuhan persoalan.

LAMPIRAN

Pranala repository

https://github.com/ChaiGans/Tucil2_13522021_13522045

Checklist

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

DAFTAR PUSTAKA

- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 1. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian1.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 2. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian2.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 3. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian3.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 4. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian4.pdf).