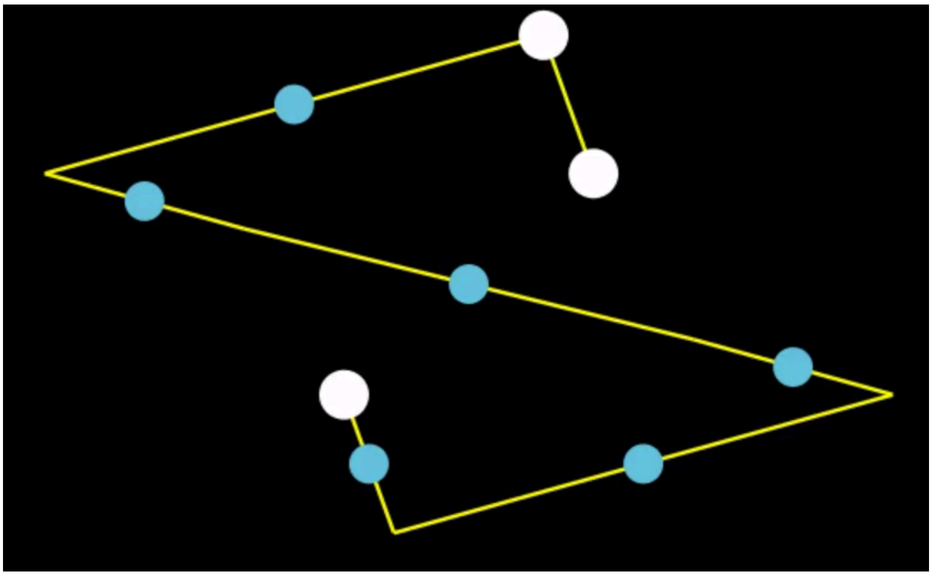
# Tugas Kecil
# IF2211 Strategi Algoritma
# Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer



Oleh:


Muhammad Rasheed Qais Tandjung 13522158



**PROGRAM STUDI TEKNIK INFORMATIKA**

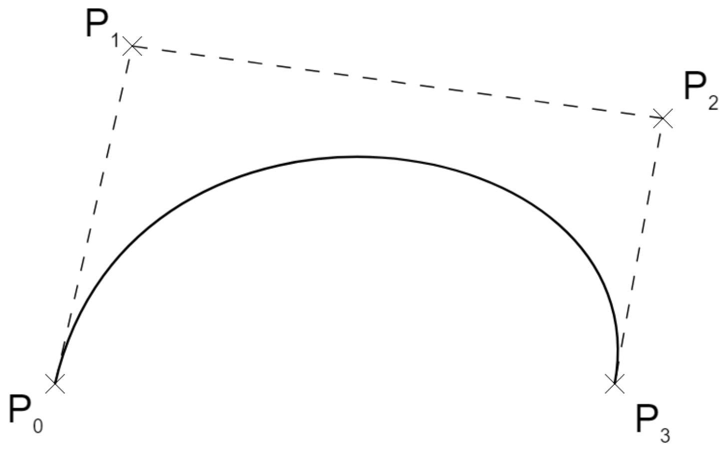
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2024**

# Bab I
# Deskripsi Tugas




*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.

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.

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

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

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 kuartik, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

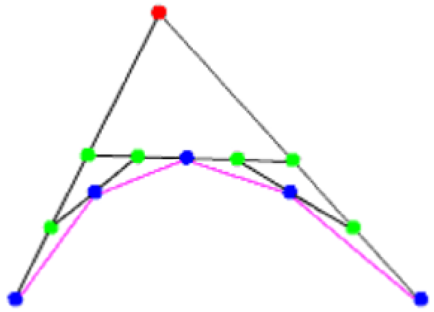
Idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik, $P_0$, $P_1$, dan $P_2$, dengan titik $P_1$ menjadi titik kontrol antara, maka:

a. Buatlah sebuah titik baru $Q_0$ yang berada di tengah garis yang menghubungkan $P_0$ dan $P_1$, serta titik $Q_1$ yang berada di tengah garis yang menghubunhkan $P_1$ dan $P_2$.
b. Hubungkan $Q_0$ dan $Q_1$ sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru $R_0$ yang berada di tengah $Q_0$ dan $Q_1$.
d. Buatlah sebuah garis yang menghubungkan $P_0$ - $R_0$ - $P_2$.

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah "kurva" yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

a. Buatlah beberapa titik baru, yaitu $S_0$ yang berada di tengah $P_0$ dan $Q_0$, $S_1$ yang berada di tengah $Q_0$ dan $R_0$, $S_2$ yang berada di tengah $R_0$ dan $Q_1$, dan $S_3$ yang berada di tengah $Q_1$ dan $P_2$.
b. Hubungkan $S_0$ dengan $S_1$ dan $S_2$ dengan $S_3$ sehingga terbentuk garis baru.
c. Buatlah dua buah titik baru, yaitu $T_0$ yang berada di tengah $S_0$ dan $S_1$, serta $T_1$ yang berada di tengah $S_2$ dan $S_3$.
d. Buatlah sebuah garis yang menghubungkan $P_0$ - $T_0$ - $R_0$ - $T_1$ - $P_2$.

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



*Gambar 2. Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2*

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

# Bab II
# Pendekatan dengan Algoritma *Bruteforce*

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1} (1-t)^{n-1} t \mathbf{P}_1 + \cdots + \binom{n}{n-1} (1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

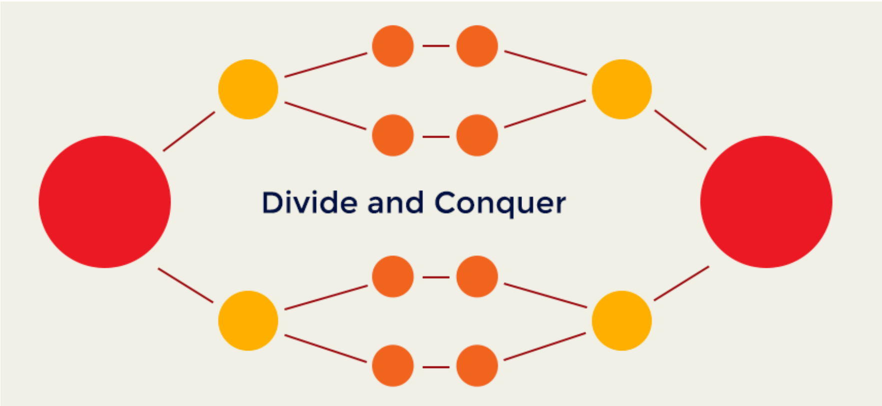
*Gambar 3. Rumus untuk melakukan generasi kurva Bezier dengan algoritma brute force*

Metode yang umum digunakan untuk melakukan generasi kurva *Bezier* adalah dengan algoritma *bruteforce* yang memanfaatkan rumus koefisien binomial. Pada rumus tersebut, n adalah jumlah *control point*, i adalah kontrol point ke-i [0 <= i <= n], dan t merupakan jarak *midpoint* antar dua titik yang berada dalam rentang [0..1].

Dengan memilih sebuah nilai t, maka rumus tersebut akan menghasilkan sebuah titik baru yang dapat disambungkan ke titik awal dan titik akhir untuk membentuk sebuah kurva sederhana tiga titik. Cara kerja pembuatan kurva *Bezier* dengan menggunakan algoritma *bruteforce* adalah dengan mengambil banyak nilai t yang akan dipakai. Setiap nilai t akan menghasilkan sebuah titik baru, sehingga dengan memasukkan banyak sekali nilai t maka bentuk kurva aproksimasi akan semakin dekat ke bentuk kurva aslinya.

# Bab III
# Pendekatan dengan Algoritma *Divide and Conquer*

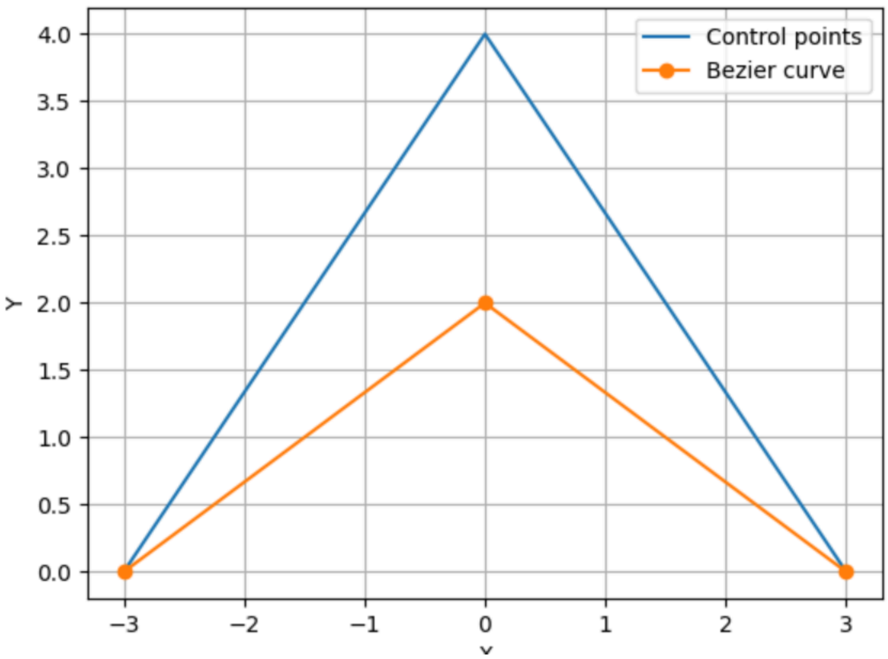
## 3.1 Metodologi Algoritma *Divide and Conquer* pada Permasalahan




*Gambar 4. Ilustrasi cara kerja algoritma divide and conquer*
*Sumber:*
*https://mobisoftinfotech.com/resources/blog/divide-and-conquer-approach-to-quality-assurance/*

Inti dari algoritma *divide and conquer* adalah pemecahan suatu masalah menjadi masalah-masalah yang lebih kecil yang sifatnya **sama seperti masalah sebelumnya**, sehingga ketika kita memperkecil suatu masalah dengan *divide and conquer*, kita bisa mempermudah solusi penyelesaian masalah dengan menyelesaikan masalah terkecil dari masalah utama, yang solusi tersebut bisa digunakan sebagai fondasi untuk menyelesaikan masalah utama tersebut.

Karena submasalah yang dihasilkan sifatnya sama seperti masalah awal, solusi yang digunakan untuk masalah utama dapat dimanfaatkan untuk menyelesaikan submasalah tersebut, sehingga pada umumnya permasalahan *divide and conquer* diselesaikan dengan sebuah solusi **rekursif**.




*Gambar 5. Permasalahan Generasi Kurva Bezier dengan Algoritma Titik Tengah*

Permasalahan utama yang dibahas pada laporan ini adalah penyusunan algoritma untuk pembentukan *Bezier curve* dari N buah titik, dikhususkan pada tiga buah terlebih dahulu. Pendekatan yang digunakan untuk membuat kurva adalah dengan pertama menyusun kumpulan titik menjadi sebuah sekuens yang linier, lalu mencari titik yang berada di tengah dua titik yang "bersebelahan".

Setelah didapatkan Q[N - 1] titik yang berada di tengah setiap pasangan Q[N] titik pada sekuens tersebut, kita lanjutkan dengan mencari titik tengah dari masing-masing pasangan N - 1 titik tengah pada Q. Prosedur ini akan terus diulangi sampai terhasilkan sekuens titik X[1], yang hanya memiliki 1 titik.

Ditemukannya titik tengah yang merupakan anggota dari X[1] tersebut menandakan akhir dari sebuah iterasi pembuatan *Bezier curve*, yang hasilnya merupakan kurva yang melewati titik awal, titik akhir, dan titik tengah X[1] yang didapatkan tersebut. Namun tentunya 3 titik saja tidak cukup untuk menghasilkan sebuah kurva yang halus dan kontinu, sehingga dapat dilakukan iterasi selanjutnya untuk mendapatkan kurva yang lebih halus. Iterasi kedua dari pembuatan kurva *Bezier* didapatkan dengan menerapkan algoritma yang sama pada sisi kiri dan sisi kanan dari kumpulan titik yang didapatkan pada iterasi sebelumnya.

Ide pertama yang mungkin didapatkan seseorang yang mencoba untuk menyelesaikan permasalahan ini adalah bahwa algoritma *divide and conquer* sudah terpakai di algoritma penyelesaian, karena pada setiap langkah algoritma akan menggunakan dua titik untuk menghasilkan titik yang paling dekat ke kurva hasil akhir. Ide lain adalah bahwa penerapan algoritma *divide and conquer* terletak pada pengurangannya jumlah titik yang dipakai pada setiap iterasi, dari N titik, menjadi N - 1, N -2, … sampai tersisa 1 titik.

Walaupun algoritma yang dipakai lumayan mirip dengan sebuah algoritma *divide and conquer*, solusi tersebut masih belum cocok disebut sebagai solusi *divide and conquer*.

Pengurangan jumlah titik dari N titik menjadi N - 1 titik tidak mengurangi masalah yang perlu diselesaikan, melainkan menambah masalah yang harus diselesaikan.

Yang menarik untuk diperhatikan adalah ketika sebuah iterasi sudah diselesaikan, permasalahan pada iterasi selanjutnya terpisah menjadi dua, yaitu bagian kiri dan bagian kanan, yang ukuran masalahnya cukup sama. Pada iterasi selanjutnya, bagian kiri dan bagian kanan akan dipisah lagi menjadi bagian kiri dan bagian kanannya masing-masing. Dengan pemahaman atas konsep ini, dapat diformulasikan sebuah prosedur yang pada setiap iterasi akan membelah permasalahan menjadi bagian kiri dan bagian kanan yang akan diselesaikan secara rekursif.

## 3.2 Spesifikasi Algoritma

Dengan memanfaatkan pemisahan ruang lingkup masalah menjadi kurva kiri dan kurva kanan, maka dapat diformulasikan sebuah prosedur untuk penyelesaian generasi titik kontrol menjadi kurva Bezier dengan langkah-langkah seperti berikut:

1. Anggap permasalahan sebagai larik arr[N], dengan N adalah jumlah *control points*.
2. Untuk setiap titik di arr[N], hitung titik yang berada tepat di tengah-tengah titik ke-i dan ke-(i + 1), dengan i dalam batas [0 .. N - 2]
3. Masukkan semua titik-titik tengah tersebut ke sebuah larik baru, new_arr[N - 1].
4. Gantikan arr[N] menjadi new_arr[N - 1] (arr[N] ← new_arr[N - 1]), lalu ulangi kembali langkah 1-4 hingga ukuran N = 1.
5. Ketika N = 1, maka iterasi sudah selesai. Jika jumlah iterasi yang dilakukan sudah cukup, maka berhenti, dan kurva yang dihasilkan adalah titik awal, titik akhir, dan titik tengah yang didapatkan ketika N = 1. Jika jumlah iterasi belum cukup, pecahkan masalah menjadi dua bagian, yaitu bagian kiri dan bagian kanan, dan selesaikan masing-masing bagian dengan mengulangi langkah-langkah yang sudah dilakukan di atas.

# Bab IV
# Implementasi Algoritma dalam Bahasa Python

## *A. Imported Libraries*

```
from typing import List
import matplotlib.pyplot as plt
from termcolor import colored
from PIL import Image
from datetime import datetime
from numpy import linspace
from math import comb
import time
import sys
import os
import shutil
import subprocess
```

## *B. class Point*

```
class Point():
    def __init__(self, x: float, y: float) -> None:
        self.x: float = x
        self.y: float = y
```

## *C. class BezierCurve*

```
class BezierCurve():
    def __init__(self, iterations: int) -> None:
        # Number of iterations for the Bezier curve
        self.iterations: int = iterations

        # Stores previous iterations of the Bezier curve
        self.iterationsList: List[List[Point]] = [[] for _ in
range(iterations + 1)]

    # Gets the midpoint of two points
    def midpoint(self, A: Point, B: Point) -> Point:
        new_x: float = (A.x + B.x) / 2
        new_y: float = (A.y + B.y) / 2
```

```
        return Point(new_x, new_y)
```

### *D. class BezierCurve → method reduce()*

```
    # Calculates the points of the Bezier curve using a decrease and conquer algorithm
    def reduce(self, arr: List[Point], left: List[Point], right: List[Point], iter: int) -> List[Point]:
        # If at 0th iteration -> append leftmost and rightmost point
        if iter == 0:
            self.iterationsList[iter] += left + right

        # If number of iterations = 0 -> return leftmost and rightmost as control points
        if self.iterations == 0:
            return []

        # Initialize left array and right array for recursion
        left_arr: List[Point] = [arr[0]]
        right_arr: List[Point] = [arr[len(arr) - 1]]

        # Reduce the number of points from N -> (N - 1) -> ... -> 1
        while len(arr) > 1:
            # Initialize new array of midpoints
            new_arr: List[Point] = []

            # Append midpoints to new_arrs
            for i in range(len(arr) - 1):
                new_arr.append(self.midpoint(arr[i], arr[i + 1]))

            # Add leftmost and rightmost to left_arr and right_arr
            left_arr += [new_arr[0]]
            right_arr = [new_arr[len(new_arr) - 1]] + right_arr
            arr = new_arr

        # Add points to iterationsList
        self.iterationsList[iter + 1] += left + arr + right

        # If current iteration is enough, finish recursion.
```

```
        if iter == self.iterations - 1:
            return arr
        else: # Else, call reduce method to left side and right side
            iter += 1
            return self.reduce(left_arr + arr, left, arr, iter) + arr +
self.reduce(arr + right_arr, [], right, iter)
```

### *E. Class BezierCurve → method bruteforce()*

```
    # Calculates the points of the Bezier curve using a bruteforce algorithm
    def bruteforce(self, control: List[Point]):
        # Get results for all iterations from 0 to N
        for iter in range(self.iterations + 1):
            n = len(control) - 1              # Number of control points
            point_count = (2**iter) + 1       # Number of curve points

            t_values = list(linspace(0, 1, num=point_count))    # Number of
t values

            # For all t values, calculate midpoint using the explicit Bezier
curve formula
            for t in t_values:
                x = 0
                y = 0

                for i in range(n + 1):
                    point_i_x = control[i].x
                    point_i_y = control[i].y

                    x += (comb(n, i) * ((1 - t)**(n - i)) * (t**i) *
point_i_x)
                    y += (comb(n, i) * ((1 - t)**(n - i)) * (t**i) *
point_i_y)

                new_point = Point(x, y)
                self.iterationsList[iter] += [new_point]
```

### *F. function plot_graph*

```
def plot_graph(control: List[Point], curve: List[Point], iter: int):
    # Get x and y values for control points
    x_control = [p.x for p in control]
    y_control = [p.y for p in control]

    # Get x and y values for curve
    x_curve = [p.x for p in curve]
    y_curve = [p.y for p in curve]

    # Plot graph
    plt.plot(x_control, y_control, label='Control points')
    plt.plot(x_curve, y_curve, '-o', label='Bezier curve')

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title(f'Bezier Curve - Iteration {iter}')
    plt.legend()
    plt.grid(True)
```

### *G. function get_datetime*

```
def get_datetime():
    return str(datetime.now().time())[:8].replace(':', '_')
```

### *H. function create_gif*

```
def create_gif(bez: BezierCurve, control: List[Point]):
    # Initialize variables
    datetime = get_datetime()
    filename = f'{datetime}'
    foldername = f'plot/{datetime}'
    os.makedirs(foldername)
    n = bez.iterations

    # Iterate through bezier curves and save iteration as image
    images: List[Image.Image] = []
    for i in range(n + 1):
        filename_i = f'{foldername}/{filename}_iter{i}'
```

```
        curve_i = bez.iterationsList[i]

        plot_graph(control, curve_i, i)

        plt.savefig(filename_i)
        images.append(Image.open(filename_i + '.png'))
        plt.close('all')

    # Compile all images as gif
    images[0].save(f'{foldername}/{filename}.gif', save_all=True,
append_images=images[1:], loop=0, duration=1000)

    # Return filename
    return f'{foldername}/{filename}.gif'
```

### *I. function write_to_file*

```
def write_to_file(control: List[Point], iterations: int):
    with open('manim.txt', 'w') as file:
        # Write iterations
        file.write(str(iterations) + '\n')

        # Write control points
        for i in range(len(control)):
            x = control[i].x
            y = control[i].y

            file.write(f'{x} {y}\n')
```

### *J. function render_manim*

```
def render_manim(filename: str):
    # Render Manim animation
    cmd = f'manim -ql dnc_manim.py PointAnimation -o {filename}.mp4'
    subprocess.run(cmd, shell=True)
```

### *K. function slowprint*

```
def slowprint(string: str, sleep: float = 0.03, endl=True):
    for c in string:
        sys.stdout.write(c)
        sys.stdout.flush()
        time.sleep(sleep)
    time.sleep(0.5)
    if endl:
        print()
```

### *L. Main Program Initialization*

```
# Start interface
slowprint('Welcome to ' + colored('Bezier Simulator!', 'yellow'))

# Input parameters
print()
slowprint(colored('Please input the following parameters: ', 'blue'))
print()

# Input control points
slowprint(colored('Control Points', 'yellow') + ' (Input empty line to
finish): ')
slowprint(colored('Format:', 'blue') + ' x y')
print()
```

### *M. Input Control Points*

```
control = []
while True:
    # Get user input
    i = len(control) + 1
    slowprint(colored(f'Point {i}: ', 'green'), endl=False)
    inp = input().split(' ')

    # Break loop if user input is empty
    if inp == ['']:
        break
```

```
    # Check if input is two numbers
    if len(inp) != 2:
        slowprint(colored('Only input two numbers (x y)!', 'red'))
    else:
        try:
            # Create control point and put in control array
            point = Point(float(inp[0]), float(inp[1]))
            control.append(point)
        except:
            slowprint(colored('Invalid point input!', 'red'))
```

## *N. Input Iteration Number*

```
# Input iteration number
iterations = 0
while True:
    # Print prompt
    print()
    slowprint(colored('Iterations: ', 'yellow'), endl=False)

    # Get user input
    inp = input().split(' ')

    # Check if input is one number
    if len(inp) > 1:
        slowprint(colored('Only input one number!', 'red'))
        continue
    try:
        # Check if input is negative
        if int(inp[0]) < 0:
            slowprint(colored('Only input positive values!', 'red'))
            continue

        # Set number of iterations
        iterations = int(inp[0])
        break
    except:
        slowprint(colored('Invalid input!', 'red'))
print()
```

## O. Main Program

```
# Get starting time
start = time.time()

# Initialize Bezier curve generation
bez = BezierCurve(iterations)
curve = [control[0]] + bez.reduce(control, [control[0]],
[control[len(control) - 1]], 0) + [control[len(control) - 1]]

# Get calculation duration
duration = time.time() - start
slowprint('Time taken: ' + colored(str(duration) + ' sec', 'yellow'))

# Create gif
gif = create_gif(bez, control)

# Print output
slowprint(colored('Bezier curve successfully generated at ', 'blue') +
colored(gif[0], 'green'))
print()

# Write coordinates to folder
write_to_file(control, iterations)
shutil.copy('manim.txt', gif[0] + '/coordinates.txt')

# Create bruteforce curve
bez_bruteforce = BezierCurve(iterations)

start_bruteforce = time.time()
bez_bruteforce.bruteforce(control)
duration_bruteforce = time.time() - start_bruteforce
slowprint('Time taken for bruteforce method: ' +
colored(str(duration_bruteforce) + ' sec', 'yellow'))

gif_bruteforce = create_gif(bez_bruteforce, control, bruteforce=True,
file=gif[1])
slowprint(colored('Bezier curve (bruteforce) successfully generated at ',
'blue') + colored(gif_bruteforce[0], 'green'))
```

## *P. Create Manim Animation*

```
# Create Manim animation
time.sleep(0.2)
print()
slowprint(colored('Do you want to render your Bezier curve as a Manim 
animation?', 'blue'))
slowprint(colored('(Note: Rendering an animation will take significantly 
longer to finish.) [Y/y to proceed]: '), endl=False)

choice = input()
iterations = 0
print()
if choice != 'Y' and choice != 'y':
    slowprint('Program finished successfully.')
else:
    slowprint('The process of rendering the animation will take time.')
    slowprint(colored('It is highly recommended that the Bezier curve is 4 
iterations or less.', 'yellow'))
    slowprint(colored('You should not animate more than 5 iterations unless 
you are ready to wait for the rendering process.', 'yellow'))

    print()

    while True:
        slowprint(colored('Insert the amount of iterations you want to 
animate: ', 'yellow'), endl=False)
        inp = input().split(' ')

        # Check if input is one number
        if len(inp) != 1:
            slowprint(colored('Input only a single number!', 'red'))
        else:
            try:
                iterations = int(inp[0])
                write_to_file(control, iterations)
                render_manim(gif[1])
                break
            except:
                slowprint(colored('Invalid point input!', 'red'))
```

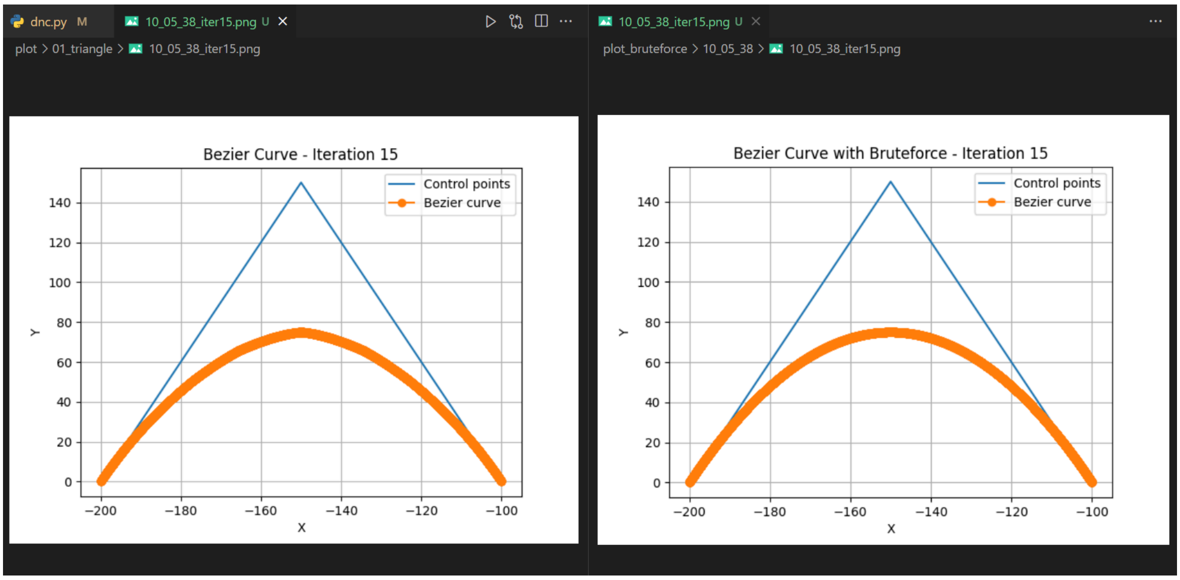
# Bab V
# Hasil Pengujian

## 5.1 Triangle - 3 Titik, 15 Iterasi

Input:

```
Please input the following parameters:

Control Points (Input empty line to finish):
Format: x y

Point 1: -200 0
Point 2: -150 150
Point 3: -100 0
Point 4:

Iterations: 15

Time taken: 2.19267201423645 sec
Bezier curve successfully generated at plot/10_05_38

Time taken for bruteforce method: 0.2982034683227539 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_05_38
```

Output:

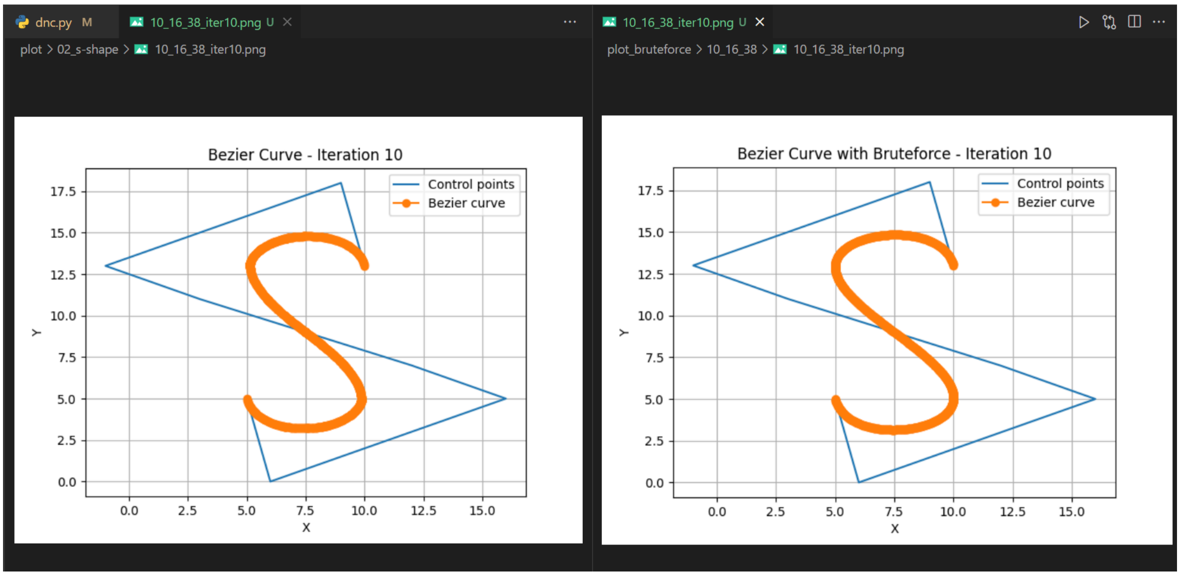


## 5.2 Huruf S - 8 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 5 5
Point 2: 6 0
Point 3: 16 5
Point 4: 12 7
Point 5: 3 11
Point 6: -1 13
Point 7: 9 18
Point 8: 10 13
Point 9:

Iterations: 10

Time taken: 0.0584409236907959 sec
Bezier curve successfully generated at plot/10_16_38

Time taken for bruteforce method: 0.022874116897583008 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_16_38
```

Output:

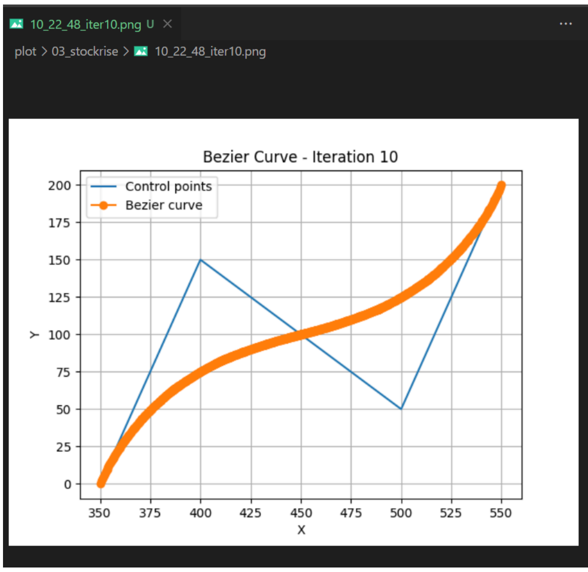


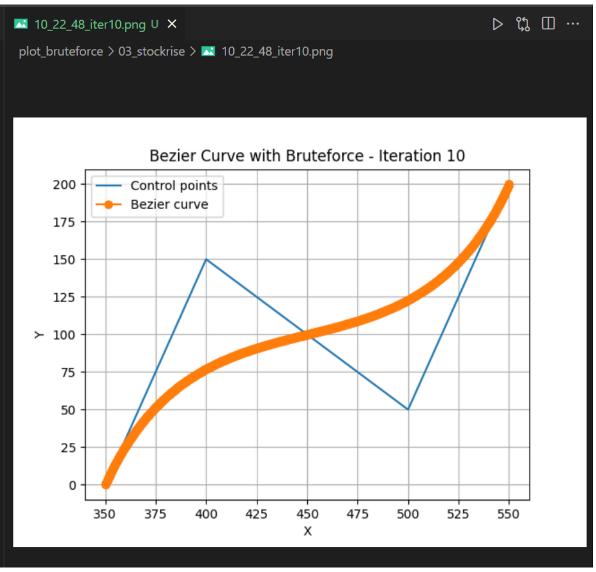
## 5.3 Stock Rising - 4 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 350 0
Point 2: 400 150
Point 3: 500 50
Point 4: 550 200
Point 5:

Iterations: 10

Time taken: 0.03393101692199707 sec
Bezier curve successfully generated at plot/10_22_48

Time taken for bruteforce method: 0.011670589447021484 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_22_48
```

Output:



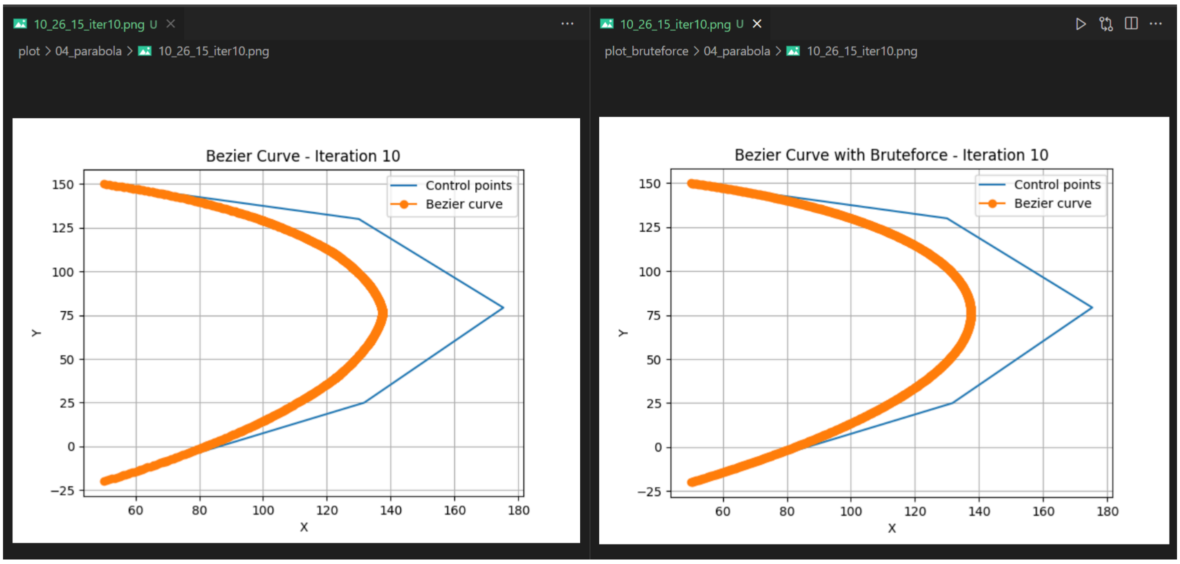



## 5.4 Parabola - 5 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 50 150
Point 2: 130 130
Point 3: 175.3 79.3
Point 4: 131.7 25.0
Point 5: 50 -20
Point 6:

Iterations: 10

Time taken: 0.038168191909790004 sec
Bezier curve successfully generated at plot/10_26_15

Time taken for bruteforce method: 0.015563726425170898 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_26_15
```

Output:

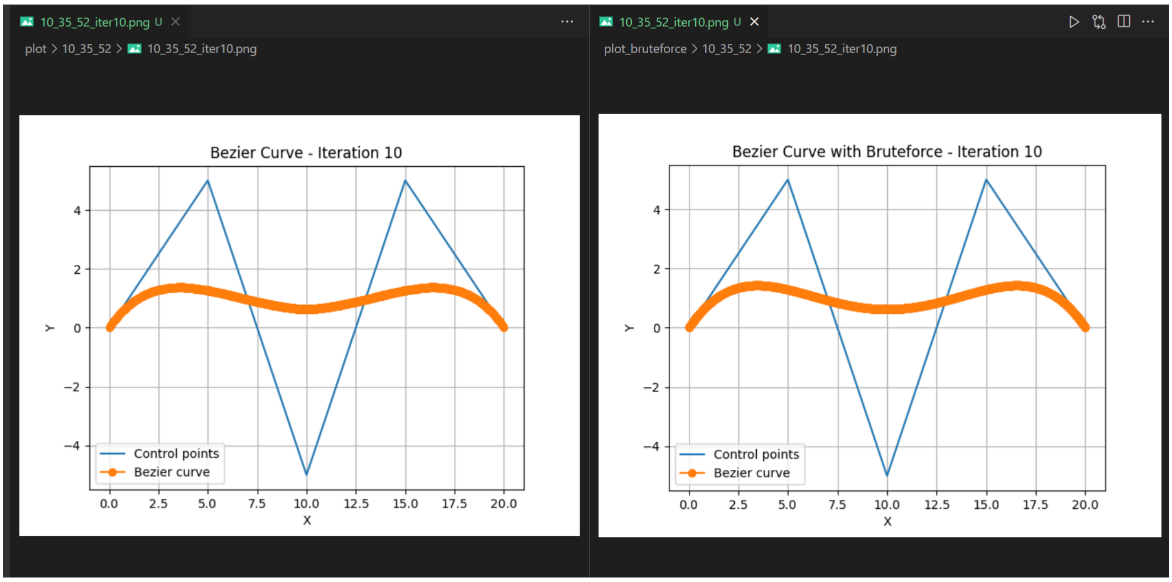


## 5.5 Moustache - 5 Titik, 10 Iterasi

Input:

```
Please input the following parameters:

Control Points (Input empty line to finish):
Format: x y

Point 1: 0 0
Point 2: 5 5
Point 3: 10 -5
Point 4: 15 5
Point 5: 20 0
Point 6:

Iterations: 10

Time taken: 0.0397818088531494l4 sec
Bezier curve successfully generated at plot/10_35_52

Time taken for bruteforce method: 0.025006055831909l8 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_35_52
```

Output:

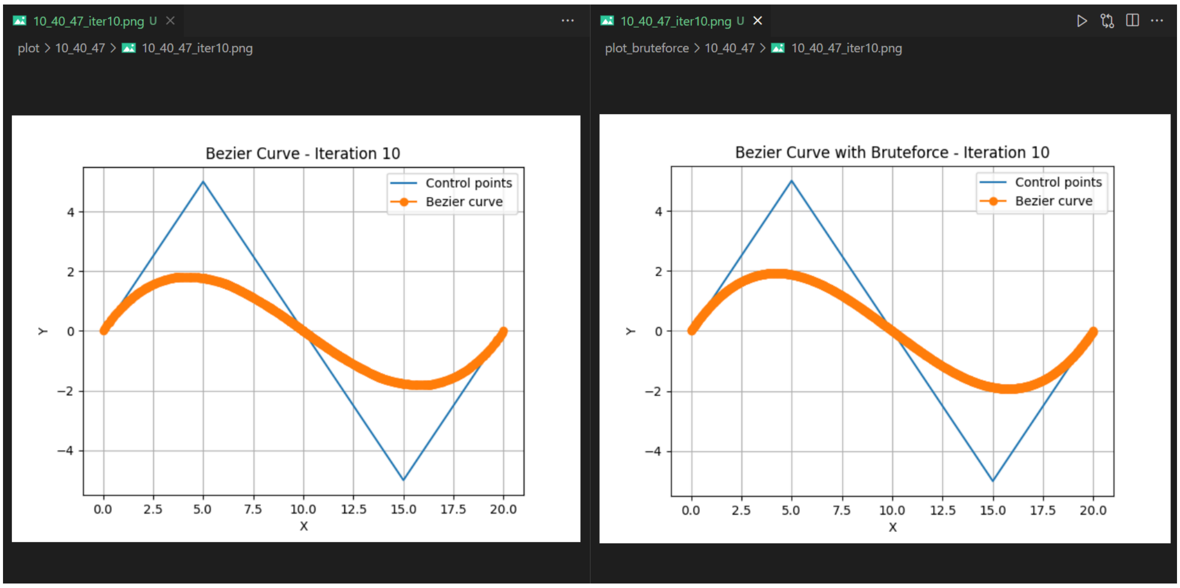


## 5.6 Sine Wave - 5 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 0 0
Point 2: 5 5
Point 3: 10 0
Point 4: 15 -5
Point 5: 20 0
Point 6:

Iterations: 10

Time taken: 0.039263486862182 62 sec
Bezier curve successfully generated at plot/10_40_47

Time taken for bruteforce method: 0.012866973876953125 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_40_47
```

Output:

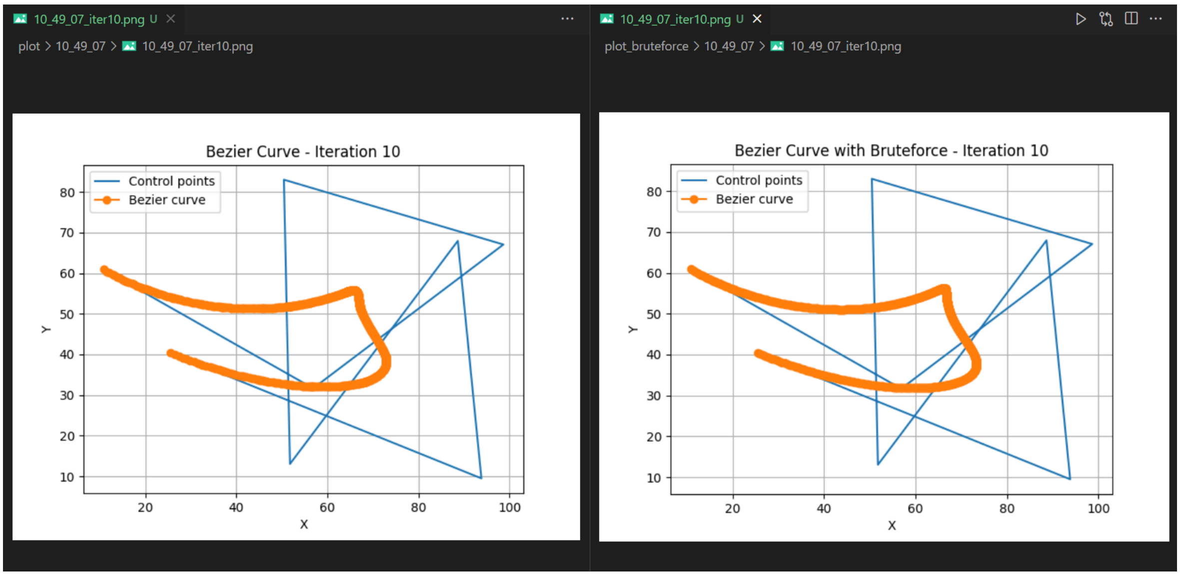


## 5.7 Literally Random - 8 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 10.822 60.904
Point 2: 56.874
Only input two numbers (x y)!
Point 2: 56.874 31.664
Point 3: 98.653 67.059
Point 4: 50.456
Only input two numbers (x y)!
Point 4: 50.456 83.018
Point 5: 51.783 13.048
Point 7: 93.823 9.493
Point 8: 25.472 40.401
Point 9:

Iterations: 10

Time taken: 0.05561804477142334 sec
Bezier curve successfully generated at plot/10_49_07

Time taken for bruteforce method: 0.021991014480590082 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_49_07
```

Output:

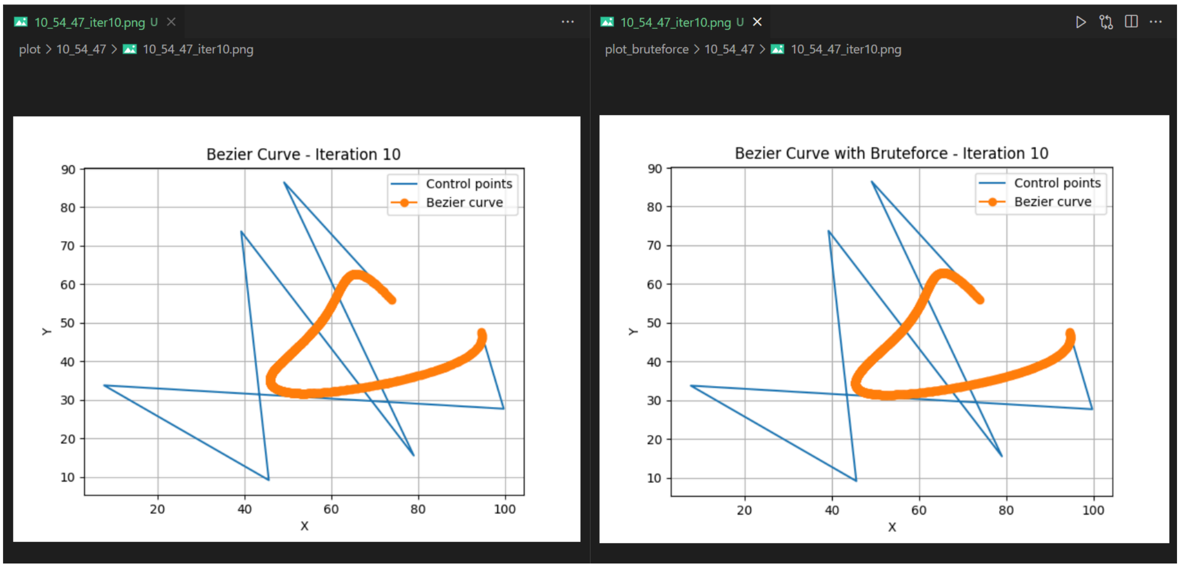


## 5.8 Literally Random [2] - 8 Titik, 10 Iterasi

Input:

```
Control Points (Input empty line to finish):
Format: x y

Point 1: 73.836 55.983
Point 2: 49.152 86.457
Point 3: 79.037 15.530
Point 4: 39.259 73.736
Point 5: 45.639 9.138
Point 6: 7.671 33.764
Point 7: 99.764 27.686
Point 8: 94.553 47.575
Point 9:

Iterations: 10

Time taken: 0.05734562873840332 sec
Bezier curve successfully generated at plot/10_54_47

Time taken for bruteforce method: 0.021279096603393555 sec
Bezier curve (bruteforce) successfully generated at plot_bruteforce/10_54_47
```

Output:



# Bab VI
# Perbandingan Solusi *Bruteforce* dengan *Divide and Conquer*

Algoritma *decrease and conquer* digunakan sebagai pengganti dari algoritma *brute force*, dikarenakan kompleksitas algoritma *bruteforce* yang umumnya sangat besar, bernilai di sekitar O(n!). Namun jika dilihat dari hasil-hasil percobaan, didapatkan bahwa algoritma *bruteforce* untuk pembentukan *Bezier curve* tidak lebih lambat dari algoritma *decrease and conquer*, bahkan di setiap percobaan waktu yang dibutuhkan algoritma ***lebih cepat*** dari algoritma DNC.

```python
def reduce(self, arr: List[Point], left: List[Point], right: List[Point], iter: int) -> List[Point]:
    # Initialize left array and right array for recursion
    left_arr: List[Point] = [arr[0]]
    right_arr: List[Point] = [arr[len(arr) - 1]]

    # Reduce the number of points from N -> (N - 1) -> ... -> 1
    while len(arr) > 1:
        # Initialize new array of midpoints
        new_arr: List[Point] = []

        # Append midpoints to new_arrs
        for i in range(len(arr) - 1):
            new_arr.append(self.midpoint(arr[i], arr[i + 1]))

        # Add leftmost and rightmost to left_arr and right_arr
        left_arr += [new_arr[0]]
        right_arr = [new_arr[len(new_arr) - 1]] + right_arr
        arr = new_arr

    # Add points to iterationsList
    self.iterationsList[iter + 1] += left + arr + right

    # If current iteration is enough, finish recursion.
    if iter == self.iterations - 1:
        return arr
    else: # Else, call reduce method to left side and right side
        iter += 1
        return self.reduce(left_arr + arr, left, arr, iter) + arr + self.reduce(arr + right_arr, [], right, iter)
```

$O(c)$ $O(c)$ $O(2^n)$

*Gambar 6. Kompleksitas Algoritma Decrease and Conquer*

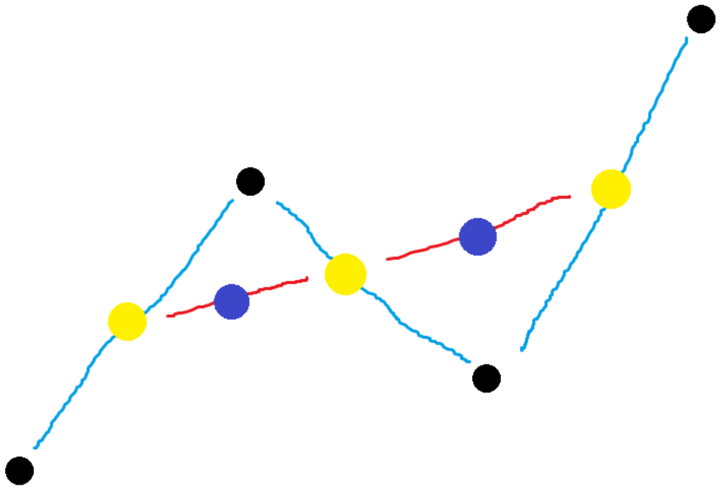
```python
# Calculates the points of the Bezier curve using a bruteforce algorithm
def bruteforce(self, control: List[Point]):
    # Get results for all iterations from 0 to N
    for iter in range(self.iterations + 1):
        n = len(control) - 1            # Number of control points
        point_count = (2**iter) + 1     # Number of curve points

        t_values = list(linspace(0, 1, num=point_count))    # Number of t values

        # For all t values, calculate midpoint using the explicit Bezier curve formula
        for t in t_values:
            x = 0
            y = 0

            for i in range(n + 1):
                point_i_x = control[i].x
                point_i_y = control[i].y

                x += (comb(n, i) * ((1 - t)**(n - i)) * (t**i) * point_i_x)
                y += (comb(n, i) * ((1 - t)**(n - i)) * (t**i) * point_i_y)

            new_point = Point(x, y)
            self.iterationsList[iter] += [new_point]
```

$O(c)$ $O(2^n)$ $O(n)$

*Gambar 7. Kompleksitas Algoritma Bruteforce*

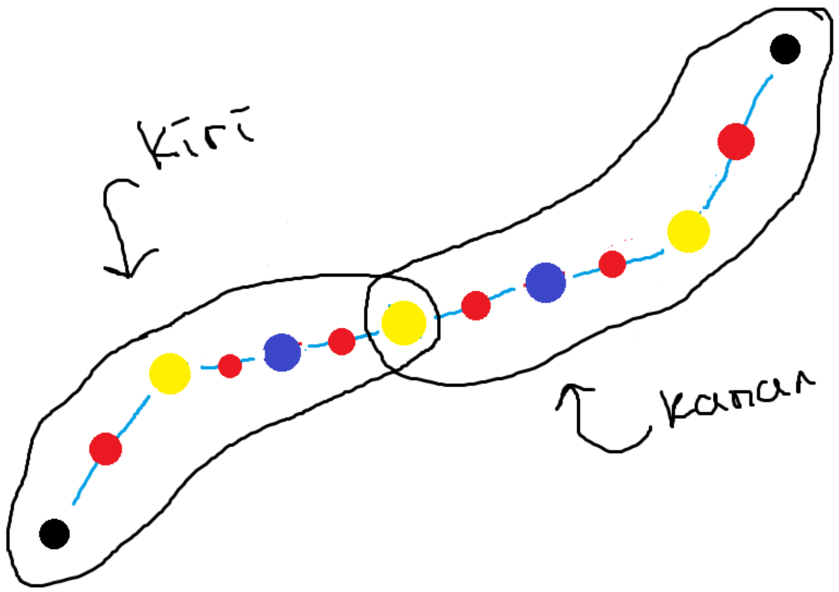
Alasan utama dari perbedaan ini berasal dari implementasi algoritma *bruteforce* yang tidak mengecek semua kemungkinan titik, melainkan melakukan aproksimasi dengan hanya $2^n + 1$ titik. Karena hal tersebut, kompleksitas algoritma *bruteforce* tidak dalam jangkauan faktorial, melainkan bernilai $O(2^n . nc)$ dalam notasi *Big-O*. Kompleksitas algoritma tersebut tidak jauh berbeda dari nilai kompleksitas algoritma *decrease and conquer*, yang bernilai $O(2^n c^2)$. Sehingga tidak terlalu asing bahwa algoritma *bruteforce* dalam permasalahan ini tidak kalah cepat, bahkan lebih cepat dari algoritma *decrease and conquer*.

# Bab VII
# Bonus - Kurva *Bezier* N Titik dan Visualisasi Pembuatan Kurva



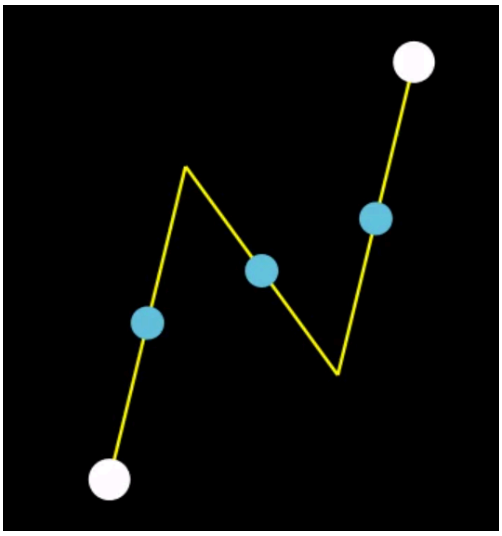
*Gambar 8. Ilustrasi pencarian titik tengah di iterasi pertama*

Generasi kurva *Bezier* sebanyak N titik didapatkan dengan menggunakan fungsi *decrease and conquer* secara rekursif. Ketika diberikan sebanyak N *control point*, algoritma akan melakukan reduksi *midpoint* hingga didapatkan N - 1 titik. Setelah itu, N - 1 titik tersebut dianggap sebagai kumpulan titik baru dan dilakukan reduksi kembali. Reduksi dilakukan sampai hanya tersisa 1 titik, yaitu titik tengah. Didapatkannya titik tengah menandakan berakhirnya satu iterasi.




*Gambar 9. Ilustrasi pemecahan masalah menjadi dua submasalah*

Ketika titik tengah sudah didapatkan, iterasi selanjutnya akan memecah kumpulan titik menjadi *Bezier kiri* dan *Bezier kanan.* Di tahap ini dimana aplikasi algoritma *divide and conquer* digunakan, yaitu bagian kiri dan bagian kanan akan diselesaikan secara rekursif dengan memanggil kembali fungsi reduksi terhadap titik-titik baru di bagian kiri dan bagian kanan. Masing-masing dari pemanggilan rekursif ini akan menghasilkan titik tengahnya masing-masing, yang jika ingin dilanjutkan ke iterasi selanjutnya maka akan dipanggil kembali fungsi rekursif terhadap bagian kiri dan bagian kanan.



*Gambar 10. Visualisasi pembangkitan titik-titik tengah menggunakan Manim*

Visualisasi pembentukan kurva *Bezier* dilakukan dengan *library Manim* di *python*. *Manim* merupakan sebuah *library* yang memberikan fungsionalitas untuk melakukan generasi dan *rendering* animasi berbasis matematika hanya dengan menulis kode *python*. Karena kode awal sudah ditulis dalam bahasa *python*, maka membuat animasi dari pembuatan kurva tersebut cukup mudah, hanya perlu mentranslasikan algoritma tersebut ke ke sintaks *manim* dan melakukan generasi objek menjadi objek *manim* setiap program mendapatkan *midpoint* dari dua titik.

# Bab VIII
# Kode Program

**Repository program dapat dilihat pada tautan berikut:**
https://github.com/trimonuter/Tucil2_13522158