

# **Laporan Tugas Kecil 2 IF2211 Strategi**

## **Algoritma**



**Disusun oleh:**

Dewantoro Triatmojo (13522011)  
Farhan Nafis Rayhan (13522037)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT  
TEKNOLOGI BANDUNG**

**2023**

# **Daftar Isi**

<b>Daftar Isi</b>	<b>2</b>
<b>BAB 1: Deskripsi Singkat</b>	<b>3</b>
<b>BAB 2: Eksplorasi Solusi</b>	<b>4</b>
2.1. Solusi Brute Force	4
2.2 Solusi Divide and Conquer	4
<b>BAB 3: Uji Coba</b>	<b>6</b>
3.1. Uji Coba Brute Force	6
3.1.1 Test Case 1 Brute Force	6
3.1.2 Test Case 2 Brute Force	6
3.1.3 Test Case 3 Brute Force	6
3.1.4 Test Case 4 Brute Force	6
3.1.5 Test Case 5 Brute Force	6
3.1.6 Test Case 6 Brute Force	6
3.2. Uji Coba Divide and Conquer	7
3.1.1 Test Case 1 Divide and Conquer	7
3.1.2 Test Case 2 Divide and Conquer	7
3.1.3 Test Case 3 Divide and Conquer	7
3.1.4 Test Case 4 Divide and Conquer	7
3.1.5 Test Case 5 Divide and Conquer	7
3.1.6 Test Case 6 Divide and Conquer	7
<b>BAB 4: Analisis Perbandingan Algoritma</b>	<b>8</b>
<b>BAB 5: Penutup</b>	<b>9</b>
5.1. Kesimpulan	9
5.2. Lampiran	9

# BAB 1: Deskripsi Singkat

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.

# BAB 2: Eksplorasi Solusi

## 2.1. Solusi Brute Force

Implementasi pembentukan kurva bezier menggunakan brute force cukup sederhana dan straight forward. Program brute force hanya perlu mengimplementasikan rumus deret kurva bezier. Pertama program akan melakukan inisialisasi array kosong untuk menyimpan solusi dan memulai perhitungan durasi. Kemudian program akan melakukan looping sebanyak jumlah iterasi yang diinginkan. Dalam loop tersebut, kita inisialisasikan variable sementara untuk menghitung x, y pada nilai t tertentu. Lalu kita lakukan looping dari 0 sampai n untuk menghitung nilai deret tersebut. Setelah dihitung, buat object point baru lalu kita append ke array solusi. Setelah itu, mulai iterasi selanjutnya sampai iterasi sudah dilakukan sesuai penentuan jumlah iterasi. Setelah iterasi selesai solusi dan durasi perhitungan disimpan dalam atribut objek.

$$\mathbf{B}(t) = \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i$$

Gambar 1. Rumus Fungsi Parametrik Kurva Bezier

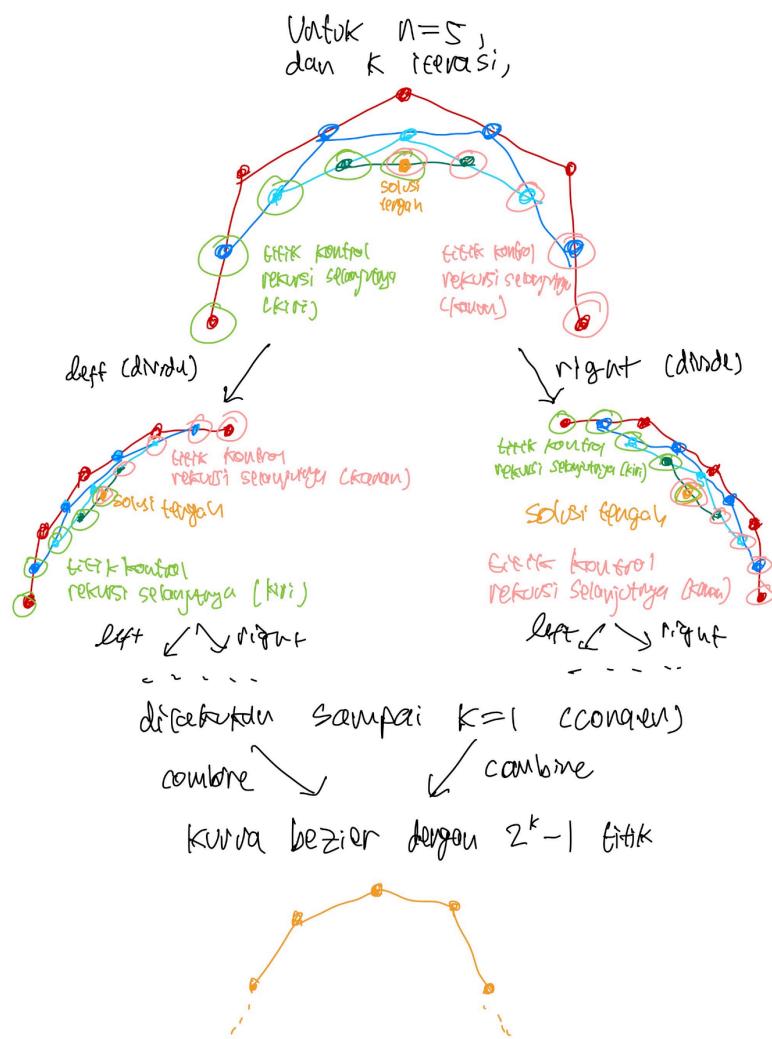
## 2.2 Solusi Divide and Conquer

Implementasi pembentukan kurva bezier menggunakan Divide and Conquer tidak sesederhana dan se-straight forward solusi brute force. Program yang kami buat dapat menentukan kurva bezier berorde n (sudah dirampatkan). Dalam solusi yang kami buat, kami membuat dua fungsi yaitu fungsi untuk memanggil solusi divide conquer dan fungsi rekursif untuk algoritma divide and conquer.

Pada fungsi pemanggil solusi, pertama tama fungsi memulai perhitungan durasi perhitungan. Kemudian fungsi memanggil fungsi divide and conquer untuk mendapatkan solusi kurva bezier dan menyimpannya di suatu variabel sementara. Setelah perhitungan selesai, solusi dan durasi perhitungan disimpan dalam atribut objek.

Pada fungsi divide and conquer fungsi menerima 2 parameter yaitu `remaining_depth` (jumlah sisa iterasi) dan `points` (list titik yang menjadi titik kontrol pada pemanggilan rekursi tersebut). Pertama fungsi mengcopy array `points` pada sebuah variabel sementara `temp` dan menginisialisasikan array kosong `left_points` dan `right_points` untuk menyimpan titik-titik pada bagian kiri dan kanan yang akan digunakan sebagai rekursi divide and conquer selanjutnya. Selanjutnya, fungsi akan melakukan looping selama isi array `temp` masih terdapat lebih dari satu elemen. Element pertama dari `temp` akan diappend ke array `left_points`. Elemen terakhir dari `temp` akan dimasukkan ke posisi awal array `right_points`. Kemudian

array `temp` akan diubah isinya menjadi titik tengah baru antara dua titik yang saling berdekatan dari titik kontrol sebelumnya. Setelah looping selesai, sisa elemen pada temp array merupakan titik yang dihasilkan untuk solusi kurva bezier dimana ini akan kita simpan dalam variabel `middle`. Program kemudian mengappend titik tersebut ke array `left_points` dan menambahkan juga di awal array `right_points`. Sekarang array `left_points` adalah titik kontrol selanjutnya untuk rekursi bagian kiri dan array `right_points` untuk bagian kanan. Jika `remaining_depth` adalah satu, artinya jumlah iterasi sudah mencapai titik terakhir kembalikan array yang sisa yang, maka program akan mengembalikan `middle` (`conquer`). Selain dari itu, maka program akan memanggil fungsi `divide` and `conquer` secara rekursif untuk sisi left dan right (`divide`) dengan parameter `remaining_depth` yang dikurangi satu. Kemudian, program akan mengembalikan gabungan dari left, middle dan right (`combine`).



Gambar 2. Ilustrasi Rekursi Divide and Conquer

## 2.3 Implementasi dalam Python

Kedua strategi kurva bezier diatas kami implementasikan dalam kode sumber menggunakan bahasa *Python*. Detail implementasi secara lengkap dapat dilihat dalam repository [github](#). Antara lain berikut adalah kode sumber program.

Kami juga mengimplementasikan spesifikasi bonus yaitu visualisasi proses penggambaran kurva. Dalam hal ini, kami memanfaatkan *library* dalam bahasa *Python* yaitu *turtle*. *Library* ini memungkinkan kami menggambar grafik 2 dimensi bagaikan sebuah robot dengan pena yang menggambar diatas selembar kertas.

### 2.3.1 Class Point.py

```
class Point:
    x: float
    y: float

    def __init__(self, X: float, Y: float):
        self.x = X
        self.y = Y

    def getX(self) -> float:
        return self.x

    def getY(self) -> float:
        return self.y

    def print(self):
        print(f"({self.x}, {self.y})")

    @staticmethod
    def midpoint(p: "Point", q: "Point") -> "Point":
        return Point((p.getX() + q.getX()) / 2.0, (p.getY() + q.getY()) /
2.0)

    def scale(self, scale : float):
        self.x *= scale
        self.y *= scale
```

## 2.3.2 Class Input.py

```
from point import Point
import sys


class Input:
    type: int # 1 for brute force, 2 for divide and conquer
    order: int # Order of Bezier Curve
    control_points: list[Point] # Array of control points
    steps: int # Steps for brute force / DNC
    all_positive: bool
    scale: float

    def __init__(self):
        self.all_positive = True

        # Type of algorithm
        print("Choose algorithm:")
        print("1. Brute Force")
        print("2. Divide and Conquer")
        type = int(input("Enter number of algorithm: "))
        # Validation: Type must be 1 or 2
        while type != 1 and type != 2:
            print("Type of algorithm must be 1 or 2")
            type = int(input("Enter number of algorithm: "))

        # Order of bezier curve
        order = int(input("Enter order of Bezier Curve (n): "))
        # Validation: Minimal order = 2
        while order < 1:
            print(
                "Order of Bezier Curve must be greater than or equal than 2
                (Linear Bezier Curve)"
            )
            order = int(input("Enter order of Bezier Curve (n): "))

        # Array of control points
        control_points: list[Point] = []
```

```

xMax = sys.float_info.min
xMin = sys.float_info.max
yMax = sys.float_info.min
yMin = sys.float_info.max
for i in range(order + 1):
    print("Input coordinate (x, y) of control point P_", i)
    x, y = list(map(float, input().split()))
    if (x < 0 or y < 0):
        self.all_positive = False
    xMax = max(x, xMax)
    xMin = min(x, xMin)
    yMax = max(y, yMax)
    yMin = min(y, yMin)
    point = Point(x, y)
    control_points.append(point)

# Calculate scale
self.scale = min(390/(xMax-xMin), 390/(yMax-yMin))

# Steps
steps = int(input("Enter number of steps/iteration for brute force / DNC: "))
# Validation: Minimal steps = 1
while steps < 1:
    print("Number of steps/iteration must be greater than 0")
    steps = int(
        input("Enter number of steps/iteration for brute force / DNC: ")
    )

# Set the input data
self.type = type
self.order = order
self.control_points = control_points
self.steps = steps

def get_type(self) -> int:
    return self.type

def get_order(self) -> int:
    return self.order

```

```

def get_control_points(self) -> list[Point]:
    return self.control_points

def get_steps(self) -> int:
    return self.steps

def is_brute_force(self) -> bool:
    return self.type == 1

```

### 2.3.3 Class Draw.py

```

from turtle import *
from point import Point

class Draw():
    t = Turtle()
    fullSize : bool

    def __init__(self, all_positive):
        Draw.t.speed(20)
        Draw.t.hideturtle()
        self.fullSize = all_positive
        if (self.fullSize):
            Draw.t.screen.setworldcoordinates(0,0,400,400)
        Draw.t.screen.title("Bezier Gesserit")
        Draw.t.screen.bgcolor("#000000")

    @staticmethod
    def drawDotsFromList(l : list[Point], color : str, size : int):
        Draw.t.hideturtle()
        Draw.t.penup()
        for i in range(len(l)):
            Draw.t.setposition(l[i].getX(), l[i].getY())
            Draw.t.dot(size, color)

    @staticmethod
    def drawLinesFromList(l : list[Point], color : str, size : int):
        Draw.t.penup()
        Draw.t.setposition(l[0].getX(), l[0].getY())
        Draw.t.pendown()
        Draw.t.pencolor(color)
        Draw.t.pensize(size)

```

```

        for i in range(1, len(l)):
            Draw.t.setposition(l[i].getX(), l[i].getY())

    @staticmethod
    def drawControlPoints(l : list[Point]):
        Draw.drawLineFromList(l, "#FFFFFF", 5)
        Draw.drawDotsFromList(l, "#08637E", 9)

    @staticmethod
    def drawBruteForceCurve(l : list[Point]):
        Draw.t.hideturtle()
        Draw.drawLineFromList(l, "#8F1D26", 9)
        Draw.drawDotsFromList(l, "#AE787E", 9) # Drive pink : "#E84298"

    @staticmethod
    def drawDNCCurve(l : list[Point]):
        Draw.t.hideturtle()
        Draw.drawLineFromList(l, "#8F1D26", 9)
        Draw.drawDotsFromList(l, "#AE787E", 9) # Drive pink : "#E84298"

```

## 2.3.4 Class Solve.py

```

from math import comb

from input import Input
from point import Point
from draw import Draw

class Solve:
    solution: list[Point] # Array of result points

    def __init__(self):
        self.solution = []

    def solve_brute_force(self, input: Input):
        # Initialize
        order = input.get_order()
        steps = input.get_steps()
        control_points = input.get_control_points()
        solution: list[Point] = []

```

```

# Iterate 0 <= t <= 1 in steps
for t in [i / steps for i in range(steps)]:
    # Use formula to calculate brute force
    x, y = 0.0, 0.0

    # Calculate B(t) = sum from 0 to n nCi * (1-t)^(n-i) * t^i * P_i
    for i in range(order+1):
        coefficient = comb(order, i) * ((1 - t) ** (order - i)) *
        (t**i)
        x += coefficient * control_points[i].x
        y += coefficient * control_points[i].y

    # Create new point
    bt = Point(x, y)

    # Append to solution
    solution.append(bt)

# Set the solution
self.solution = solution

def solve_brute_force_optimized(self, input:Input):
    # Initialize
    order = input.get_order()
    steps = input.get_steps()
    control_points = input.get_control_points()
    end = input.control_points[input.order]
    solution: list[Point] = []
    combination : list[int] = [None for i in range (order//2+1)]

    # Precompute
    for i in range(order//2 + 1):
        combination[i] = comb(order, i)

    if (order%2==1):
        combination.append(combination[len(combination)-1])

    for i in range (order//2-1, -1, -1):
        combination.append(combination[i])

    # Iterate 0 <= t <= 1 in steps

```

```

        for t in [i / steps for i in range(steps)]:
            # Use formula to calculate brute force
            x, y = 0.0, 0.0
            coefficient = (1-t)**order
            multiply = t/(1-t)

            for i in range(order+1):
                x += combination[i] * coefficient * control_points[i].x
                y += combination[i] * coefficient * control_points[i].y
                coefficient *= multiply

            # Create new point
            bt = Point(x, y)

            # Append to solution
            solution.append(bt)

        # Append end point
        solution.append(end)

        # Set the solution
        self.solution = solution

    @staticmethod
    def dnc(remaining_depth: int, points: list[Point]) -> list[Point]:
        # Left & right next points
        left_points = []
        right_points = []

        # Get middle point from points and fill left & right points
        temp = points.copy()
        while len(temp) > 1:
            # Draw in GUI
            Draw.drawDotsFromList(temp, "#138757", 7)
            Draw.drawLinesFromList(temp, "#2d3030", 3)
            left_points.append(temp[0])
            right_points.insert(0, temp[-1])
            temp = [Point.midpoint(temp[i], temp[i + 1]) for i in
range(len(temp) - 1)]

        middle = temp

```

```

        Draw.drawDotsFromList(temp, "#AE787E", 9)
        left_points.append(middle[0])
        right_points.insert(0, middle[0])

    # If current remaining depth = 1 (means current is last), return middle
    point
    if remaining_depth == 1:
        # CONQUER
        return middle
    else:
        # DIVIDE
        left = Solve.dnc(remaining_depth - 1, left_points)
        right = Solve.dnc(remaining_depth - 1, right_points)

    # COMBINE
    return left + middle + right

def solve_dnc(self, input: Input):
    # Initialize
    steps = input.get_steps()
    control_points = input.get_control_points()
    solution: list[Point] = Solve.dnc(steps, control_points)
    start = control_points[0]
    end = control_points[input.get_order()]

    solution.insert(0, start)
    solution.append(end)
    self.solution = solution

def get_solution(self) -> list[Point]:
    return self.solution

def print_solution(self, scale : float):
    # Print points
    print("Result:")
    for i in range(len(self.solution)):
        t = i / len(self.solution)
        print(f"B({t}) = ({self.solution[i].x / scale}, {self.solution[i].y / scale})")

```

## 2.3.5 Class Main.py

```
import time
from input import Input
from solve import Solve
from draw import Draw


# Main Program
def main():
    # Create input
    input = Input()

    # Create GUI window
    draw = Draw(input.all_positive)

    if input.is_brute_force():
        for i in range(len(input.control_points)):
            input.control_points[i].scale(input.scale)

    # Start time
    timeStart = time.time()

    # Solve the curve
    result = Solve()

    # Print solution
    result.print_solution(input.scale)

    # Draw in GUI
    Draw.drawControlPoints(input.control_points)
    result.solve_brute_force_optimized(input)
    Draw.drawBruteForceCurve(result.solution)
    Draw.t.hideturtle()

    # End time
    timeEnd = time.time()
    duration = timeEnd - timeStart
    print("Calculation duration:", duration, "seconds")

else:
    for i in range(len(input.control_points)):
        input.control_points[i].scale(input.scale)
```

```
# Start time
timeStart = time.time()

# Solve the curve
result = Solve()

# Draw in GUI
Draw.drawControlPoints(input.control_points)
result.solve_dnc(input)
Draw.drawDNCCurve(result.solution)
Draw.t.hideturtle()

# Print the Solution
result.print_solution(input.scale)

# End time
timeEnd = time.time()
duration = timeEnd - timeStart
print("Calculation duration:", duration, "seconds")

draw.t.screen.mainloop()

# Run main program
main()
```

# BAB 3: Uji Coba

## 3.1. Uji Coba Brute Force

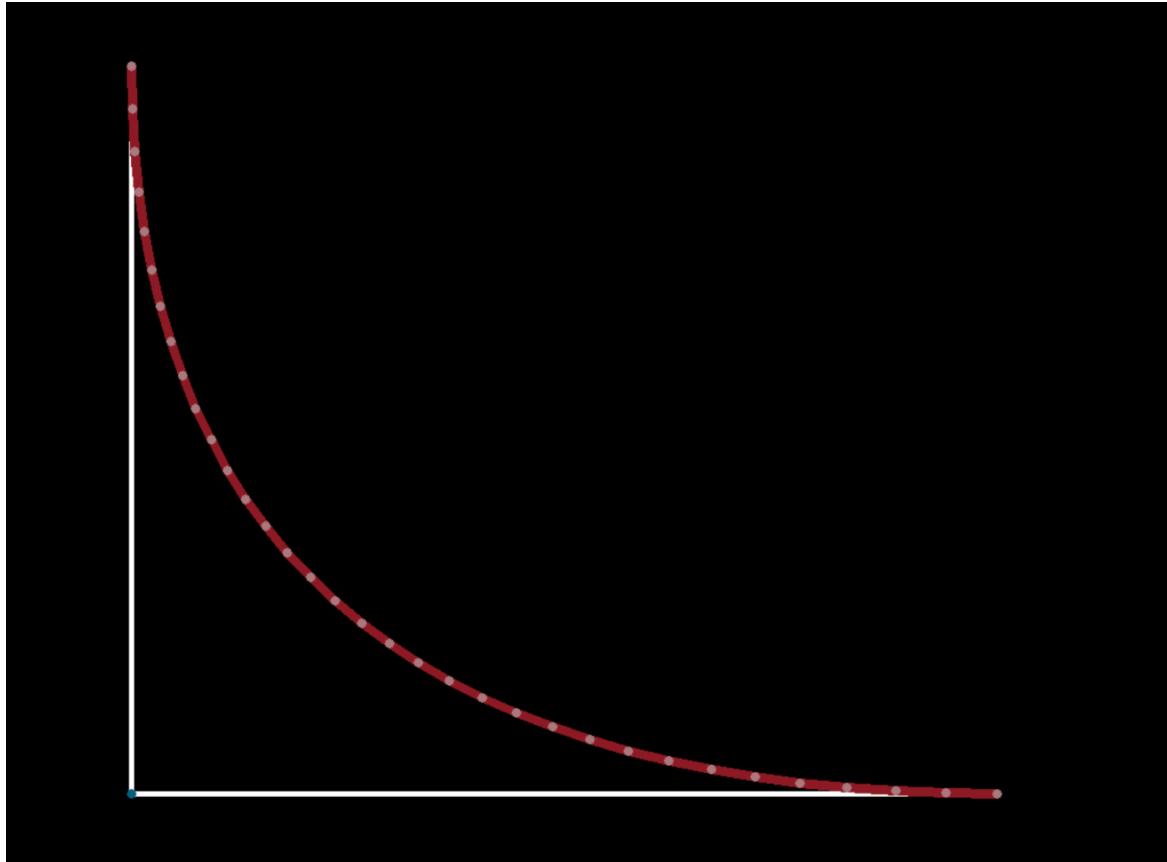
### 3.1.1 Test Case 1 Brute Force

Input:

```
Choose algorithm:  
1. Brute Force  
2. Divide and Conquer  
Enter number of algorithm: 1  
Enter order of Bezier Curve (n): 2  
Input coordinate (x, y) of control point P_ 0  
300 0  
Input coordinate (x, y) of control point P_ 1  
0 0  
Input coordinate (x, y) of control point P_ 2  
0 300  
Enter number of steps/iteration for brute force / DNC: 33  
Result:
```

Output:

```
Result:  
B(0.0) = (300.0, 0.0)  
B(0.029411764705882353) = (282.0936639118458, 0.2754820936639119)  
B(0.058823529411764705) = (264.73829201101927, 1.1019283746556474)  
B(0.08823529411764706) = (247.93388429752062, 2.4793388429752063)  
B(0.11764705882352941) = (231.68044077134985, 4.40771349862259)  
B(0.14705882352941177) = (215.97796143250693, 6.887052341597797)  
B(0.17647058823529413) = (200.8264462889917, 9.917355371900825)  
B(0.20588235294117646) = (186.22589531680438, 13.498622589531685)  
B(0.23529411764705882) = (172.1763085399449, 17.63085399449036)  
B(0.2647058823529412) = (158.6776859504132, 22.31404958677685)  
B(0.29411764705882354) = (145.7300275482094, 27.548209366391188)  
B(0.3235294117647059) = (133.3333333333337, 33.3333333333333)  
B(0.35294117647058826) = (121.48766330578511, 39.669421487603316)  
B(0.38235294117647056) = (110.19283746556475, 46.55647382920109)  
B(0.4117647058823529) = (99.44903581267215, 53.99449035812672)  
B(0.4411764705882353) = (89.25619834710741, 61.983471074380155)  
B(0.47058823529411764) = (79.61432506887051, 70.52341597796145)  
B(0.5) = (70.52341597796145, 79.61432506887051)  
B(0.5294117647058824) = (61.98347107438017, 89.2561983471074)  
B(0.5588235294117647) = (53.9944903581267222)  
B(0.5882352941176471) = (46.556473829201096, 110.19283746556475)  
B(0.6176470588235294) = (39.6694214876033, 121.48766330578511)  
B(0.6470588235294118) = (33.3333333333334, 133.33333333333331)  
B(0.6764705882352942) = (27.548209366391173, 145.7300275482094)  
B(0.7058823529411765) = (22.314049586776854, 158.6776859504132)  
B(0.7352941176470589) = (17.630853994490362, 172.1763085399449)  
B(0.7647058823529411) = (13.498622589531685, 186.22589531680435)  
B(0.7941176470588235) = (9.91735537190082, 200.82644628099177)  
B(0.8235294117647058) = (6.887052341597793, 215.97796143250693)  
B(0.8529411764705882) = (4.4077134986225985, 231.68044077134988)  
B(0.8823529411764706) = (2.479338842975208, 247.93388429752068)  
B(0.9117647058823529) = (1.1019283746556454, 264.73829201101927)  
B(0.9411764705882353) = (0.27548209366391135, 282.0936639118458)  
B(0.9705882352941176) = (0.0, 300.0)  
Calculation duration: 16.461461782455444 seconds
```



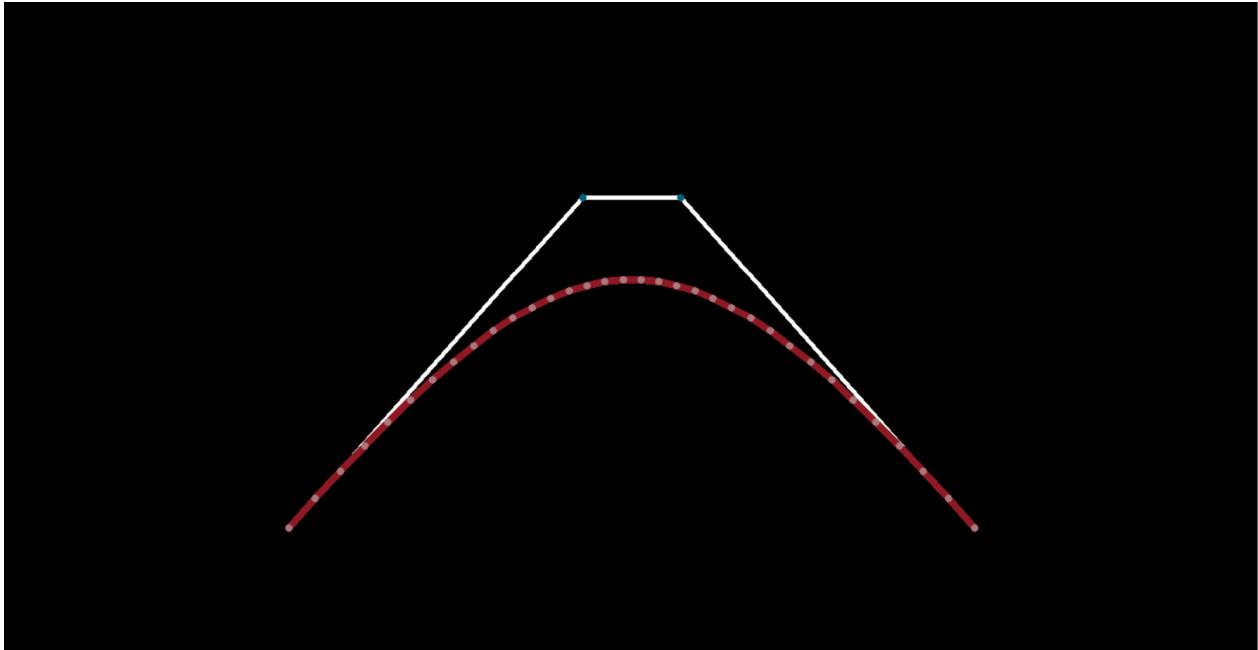
### 3.1.2 Test Case 2 Brute Force

Input:

```
Choose algorithm:  
1. Brute Force  
2. Divide and Conquer  
Enter number of algorithm: 1  
Enter order of Bezier Curve (n): 3  
Input coordinate (x, y) of control point P_ 0  
100 100  
Input coordinate (x, y) of control point P_ 1  
400 500  
Input coordinate (x, y) of control point P_ 2  
500 500  
Input coordinate (x, y) of control point P_ 3  
800 100  
Enter number of steps/iteration for brute force / DNC: 33  
Result:  
400 500  
Input coordinate (x, y) of control point P_ 2  
500 500  
Input coordinate (x, y) of control point P_ 3  
800 100  
Enter number of steps/iteration for brute force / DNC: 33
```

Output:

```
Result:  
B(0.0) = (100.0, 100.0)  
B(0.029411764705882353) = (126.73289367504243, 135.2617079889807)  
B(0.058823529411764705) = (152.43064251328715, 168.31955922865018)  
B(0.08823529411764706) = (177.16003005259202, 199.17355371900823)  
B(0.11764705882352941) = (200.98783983081503, 227.8236914600551)  
B(0.14705882352941177) = (223.9808553858141, 254.26997245179066)  
B(0.17647058823529413) = (246.205860255447, 278.51239669421483)  
B(0.20588235294117646) = (267.7296379775719, 300.55096418732785)  
B(0.23529411764705882) = (288.61897209004644, 320.3856749311294)  
B(0.2647058823529412) = (308.9406461307288, 338.0165289256198)  
B(0.29411764705882354) = (328.7614436374768, 353.443526170799)  
B(0.3235294117647059) = (348.14814814814815, 366.66666666666663)  
B(0.35294117647058826) = (367.167543200601, 377.6859504132231)  
B(0.38235294117647056) = (385.8864123326933, 386.5013774104682)  
B(0.4117647058823529) = (404.37153908228277, 393.11294765840205)  
B(0.4411764705882353) = (422.6897069872275, 397.5206611570247)  
B(0.47058823529411764) = (440.90769958538556, 399.724517906336)  
B(0.5) = (459.09230041461444, 399.724517906336)  
B(0.5294117647058824) = (477.31029301277226, 397.52066115702473)  
B(0.5588235294117647) = (495.6284609177171, 393.11294765840216)  
B(0.5882352941176471) = (514.1135876673067, 386.5013774104683)  
B(0.6176470588235294) = (532.8324567993989, 377.68595041322305)  
B(0.6470588235294118) = (551.8518518518517, 366.66666666666663)  
B(0.6764705882352942) = (571.2385563625234, 353.44352617079886)  
B(0.7058823529411765) = (591.0593538692711, 338.01652892561975)  
B(0.7352941176470589) = (611.3810279099534, 320.3856749311295)  
B(0.7647058823529411) = (632.270362022428, 300.5509641873278)  
B(0.7941176470588235) = (653.794139744553, 278.5123966942149)  
B(0.8235294117647058) = (676.0191446141861, 254.26997245179066)  
B(0.8529411764705882) = (699.0121601691851, 227.82369146005507)  
B(0.8823529411764706) = (722.839969947408, 199.1735537190083)  
B(0.9117647058823529) = (747.5693574867128, 168.31955922865004)  
B(0.9411764705882353) = (773.2671063249575, 135.26170798898065)  
B(0.9705882352941176) = (800.0, 100.0)  
Calculation duration: 13.317183017730713 seconds
```



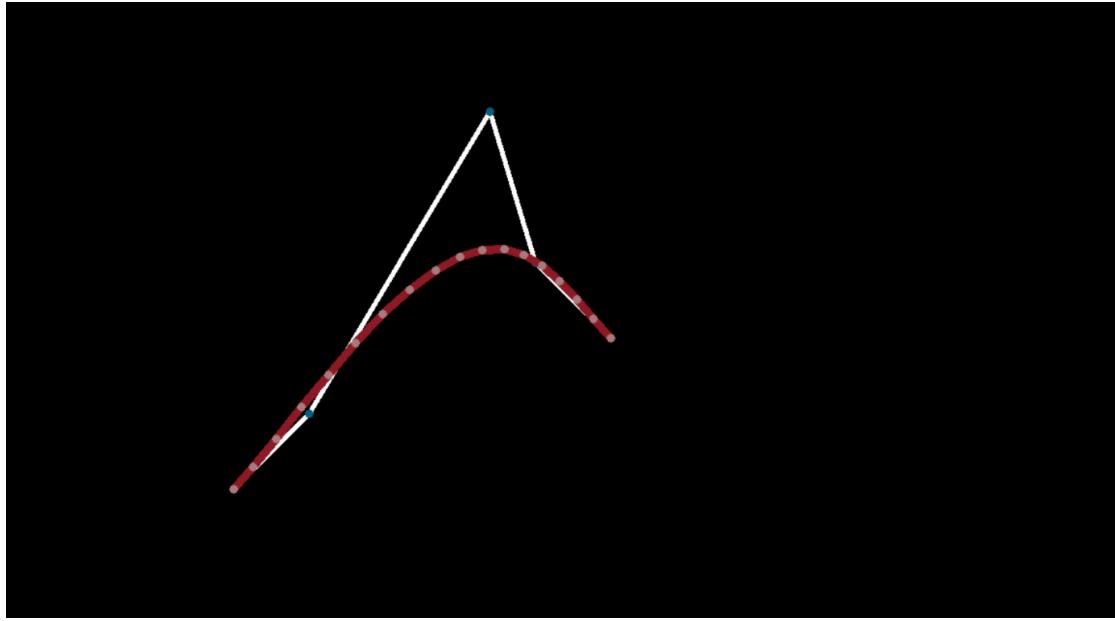
### 3.1.3 Test Case 3 Brute Force

Input:

```
Enter number of algorithm: 1
Enter order of Bezier Curve (n): 4
Input coordinate (x, y) of control point P_ 0
-50 -50
Input coordinate (x, y) of control point P_ 1
0 0
Input coordinate (x, y) of control point P_ 2
120 200
Input coordinate (x, y) of control point P_ 3
150 100
Input coordinate (x, y) of control point P_ 4
200 50
Enter number of steps/iteration for brute force / DNC: 17
```

Output:

```
Result:
B(0.0) = (-50.0, -50.0)
B(0.0555555555555555) = (-36.909040840028254, -35.477903760730825)
B(0.1111111111111111) = (-21.64784904395301, -16.791585349792264)
B(0.1666666666666666) = (-4.881646532009906, 4.205529148357902)
B(0.2222222222222222) = (12.801930053519483, 25.8898959543109)
B(0.2777777777777778) = (30.892829348307625, 46.86785359370699)
B(0.333333333333333) = (48.95858526598099, 65.97562289723541)
B(0.3888888888888889) = (66.64431699812023, 82.27930700063456)
B(0.4444444444444444) = (83.67272901425987, 95.07489134469174)
B(0.5) = (99.84411106188863, 103.88824367524333)
B(0.5555555555555556) = (115.03633816644916, 108.47511404317476)
B(0.6111111111111112) = (129.20487063133828, 108.82113480442047)
B(0.6666666666666666) = (142.3827540379066, 105.1418206199638)
B(0.7222222222222222) = (154.6806192454592, 97.88256845583746)
B(0.7777777777777778) = (166.28668239125489, 87.7186575831228)
B(0.833333333333334) = (177.46674489050653, 75.55524957795045)
B(0.8888888888888888) = (188.5641934363812, 62.52738832149997)
B(0.9444444444444444) = (200.0, 50.0)
Calculation duration: 3.129777431488037 seconds
```



### 3.1.4 Test Case 4 Brute Force

Input:

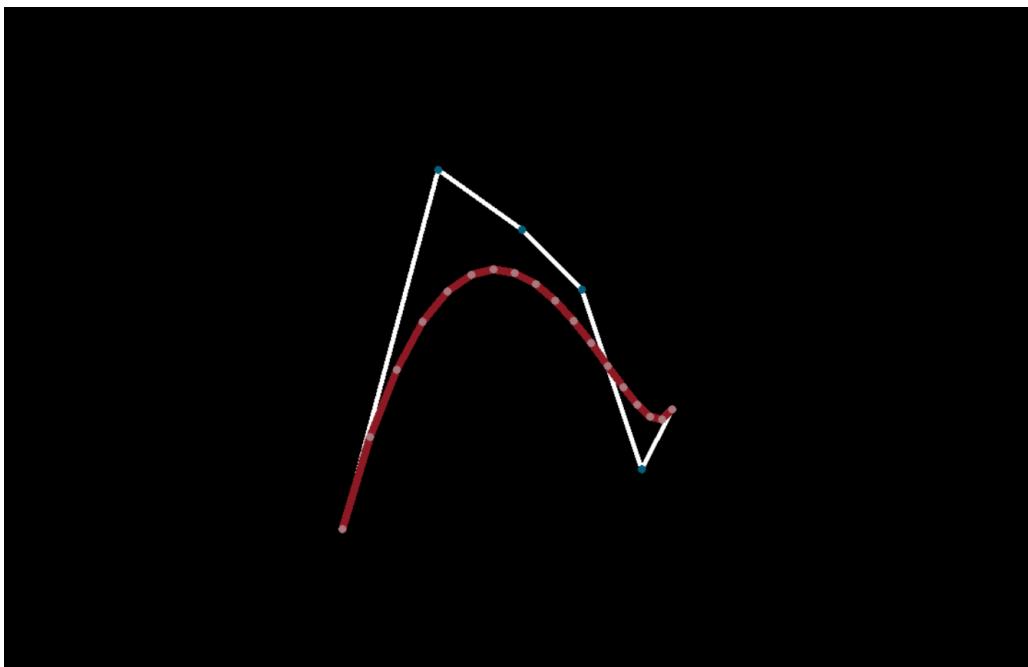
```
Enter number of algorithm: 1
Enter order of Bezier Curve (n): 5
Input coordinate (x, y) of control point P_ 0
-200 -200
Input coordinate (x, y) of control point P_ 1
-120 100
Input coordinate (x, y) of control point P_ 2
-50 50
Input coordinate (x, y) of control point P_ 3
0 0
Input coordinate (x, y) of control point P_ 4
50 -150
Input coordinate (x, y) of control point P_ 5
75 -100
Enter number of steps/iteration for brute force / DNC: 17
```

Output:

```

Result:
B(0.0) = (-200.0, -200.0)
B(0.0555555555555555) = (-176.83522002567864, -123.18937752182084)
B(0.1111111111111111) = (-154.46104783791606, -66.68502532297265)
B(0.1666666666666666) = (-132.9428773460989, -27.092587492965826)
B(0.222222222222222) = (-112.31567686041619, -1.5205756636055587)
B(0.2777777777777778) = (-92.59032775835874, 12.491469211336083)
B(0.3333333333333333) = (-73.75996315121873, 17.04481507644783)
B(0.3888888888888889) = (-55.80630655058926, 13.953376995007245)
B(0.4444444444444444) = (-38.70601053486373, 4.815555369308325)
B(0.5) = (-22.436995415735527, -8.913925839010547)
B(0.555555555555556) = (-6.984787904697442, -25.85218088863878)
B(0.6111111111111112) = (7.651140220458834, -44.61632403826585)
B(0.6666666666666666) = (21.45103344914312, -63.751631326253275)
B(0.722222222222222) = (34.36851387146733, -81.6597023503071)
B(0.7777777777777778) = (46.32424251174589, -96.5266220471498)
B(0.833333333333334) = (57.199580661996244, -106.25112247219262)
B(0.8888888888888888) = (66.83025121543933, -108.37274457920766)
B(0.9444444444444444) = (75.0, -100.0)
Calculation duration: 3.1763195991516113 seconds

```



### 3.1.5 Test Case 5 Brute Force

Input:

```

Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 1
Enter order of Bezier Curve (n): 6
Input coordinate (x, y) of control point P_ 0
-100 -150
Input coordinate (x, y) of control point P_ 1
-25 0
Input coordinate (x, y) of control point P_ 2
-10 25
Input coordinate (x, y) of control point P_ 3
0 100
Input coordinate (x, y) of control point P_ 4
15 38
Input coordinate (x, y) of control point P_ 5
59 124
Input coordinate (x, y) of control point P_ 6
148 0
Enter number of steps/iteration for brute force / DNC: 17

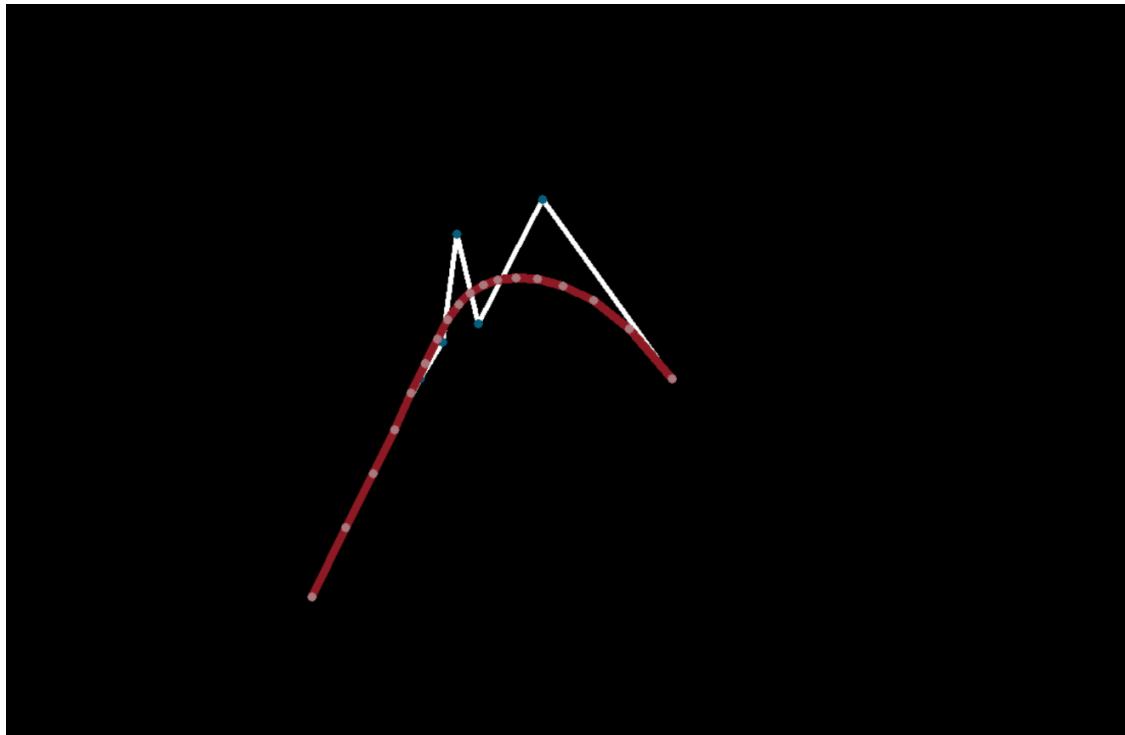
```

### Output:

```

Result:
B(0.0) = (-100.0, -150.0)
B(0.0555555555555555) = (-76.42753866389774, -102.8958871541703)
B(0.1111111111111111) = (-57.845949938040576, -65.30264626069012)
B(0.1666666666666666) = (-43.167217875172085, -34.80135021053693)
B(0.2222222222222222) = (-31.442687703968847, -9.813529357492452)
B(0.2777777777777778) = (-21.839187699473783, 10.676402416498542)
B(0.3333333333333333) = (-13.617567204054383, 27.340584878286617)
B(0.3888888888888889) = (-6.113650798885342, 40.66901186279363)
B(0.4444444444444444) = (1.278391374044336, 51.072694934605884)
B(0.5) = (9.120257139399582, 58.92869029188482)
B(0.5555555555555556) = (17.941309665443107, 64.56698891259512)
B(0.6111111111111112) = (28.247958690454716, 68.1992699430502)
B(0.6666666666666666) = (40.53062559862594, 69.78951732877489)
B(0.7222222222222222) = (55.26829234543046, 68.86649968768603)
B(0.7777777777777778) = (72.93063423246969, 64.27811342558977)
B(0.8333333333333334) = (93.97773653179406, 53.887589093997)
B(0.8888888888888888) = (118.85739495969959, 34.2115609902555)
B(0.9444444444444444) = (148.0, 0.0)
Calculation duration: 2.9417812824249268 seconds

```



### 3.1.6 Test Case 6 Brute Force

Input:

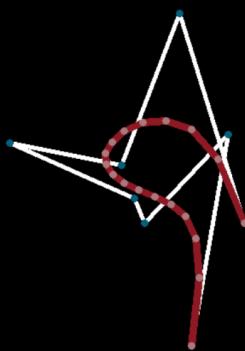
```
o py"ents\ITB\Semester 4\Stima\Tucil2_13522011_13522037>
Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 1
Enter order of Bezier Curve (n): 7
Input coordinate (x, y) of control point P_ 0
100 0
Input coordinate (x, y) of control point P_ 1
35 210
Input coordinate (x, y) of control point P_ 2
-23 58
Input coordinate (x, y) of control point P_ 3
-135 80
Input coordinate (x, y) of control point P_ 4
-10 25
Input coordinate (x, y) of control point P_ 5
0 0
Input coordinate (x, y) of control point P_ 6
84 89
Input coordinate (x, y) of control point P_ 7
47 -123
Enter number of steps/iteration for brute force / DNC: 17
Result:
```

Output:

```

Result:
B(0.0) = (100.0, 0.0)
B(0.0555555555555555) = (73.44279628013514, 63.67122928479128)
B(0.1111111111111111) = (46.95700896317905, 93.5079907420766)
B(0.1666666666666666) = (21.360474470803773, 102.85577051861254)
B(0.2222222222222222) = (-1.6226730645005565, 100.54819370145807)
B(0.2777777777777778) = (-20.166057623820414, 92.20120455231864)
B(0.3333333333333333) = (-32.848137660181, 81.24134209499668)
B(0.3888888888888889) = (-38.91497051022534, 69.68568736634775)
B(0.4444444444444444) = (-38.351332378559384, 58.691058641699115)
B(0.5) = (-31.812861414112923, 48.89003094524312)
B(0.5555555555555556) = (-20.47189139786489, 40.53135615613787)
B(0.6111111111111112) = (-5.8296435612833255, 33.44236002098687)
B(0.6666666666666666) = (10.45255594517167, 26.830892383375215)
B(0.7222222222222222) = (26.650365414131937, 18.944406941141526)
B(0.7777777777777778) = (40.94915625464332, 6.603746842062839)
B(0.833333333333334) = (51.23971345737623, -15.370787571367897)
B(0.8888888888888888) = (54.68424033237543, -54.823512557394295)
B(0.9444444444444444) = (47.0, -123.0)
Calculation duration: 3.439054250717163 seconds

```



## 3.2. Uji Coba Divide and Conquer

### 3.1.1 Test Case 1 Divide and Conquer

Input:

```

Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 2
Input coordinate (x, y) of control point P_ 0
300 0
Input coordinate (x, y) of control point P_ 1
0 0
Input coordinate (x, y) of control point P_ 2
0 300
Enter number of steps/iteration for brute force / DNC: 5

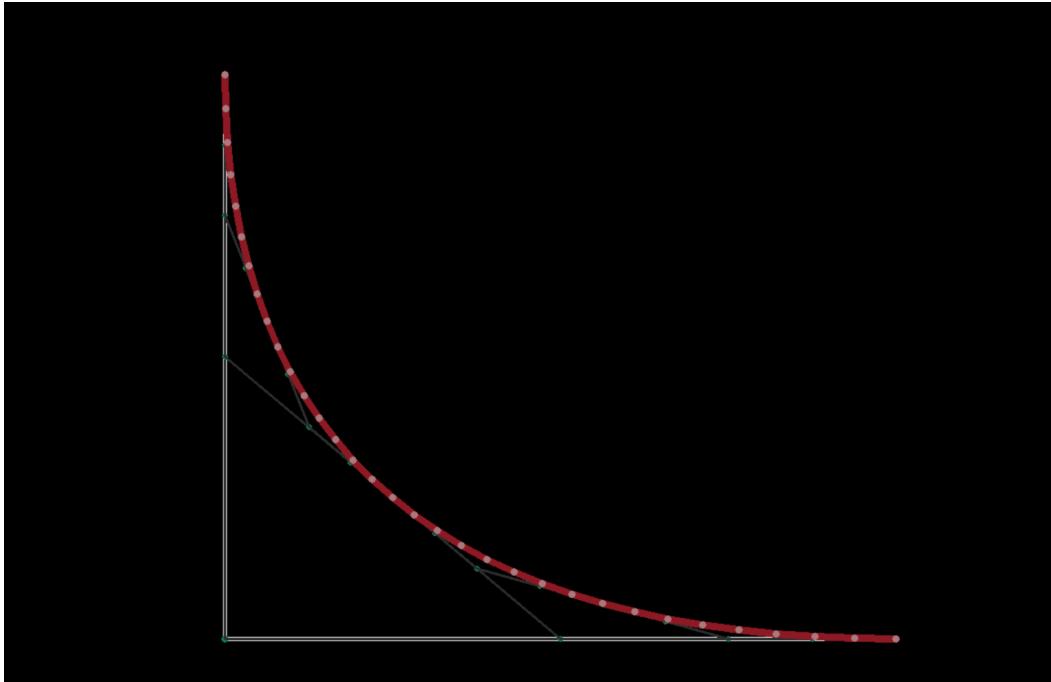
```

Output:

```

Result:
B(0.0) = (300.0, 0.0)
B(0.030303030303030304) = (281.54296875, 0.29296875)
B(0.06060606060606061) = (263.671875, 1.171875)
B(0.09090909090909091) = (246.38671875, 2.63671875)
B(0.12121212121212122) = (229.6875, 4.6875)
B(0.15151515151515152) = (213.57421875, 7.32421875)
B(0.181818181818182) = (198.046875, 10.546875)
B(0.21212121212121213) = (183.10546875, 14.35546875)
B(0.24242424242424243) = (168.75, 18.75)
B(0.2727272727272727) = (154.98046875, 23.73046875)
B(0.30303030303030304) = (141.796875, 29.296875)
B(0.3333333333333333) = (129.19921875, 35.44921875)
B(0.36363636363636365) = (117.1875, 42.1875)
B(0.3939393939393939) = (105.76171875, 49.51171875)
B(0.42424242424242425) = (94.921875, 57.421875)
B(0.45454545454545453) = (84.66796875, 65.91796875)
B(0.48484848484848486) = (75.0, 75.0)
B(0.5151515151515151) = (65.91796875, 84.66796875)
B(0.5454545454545454) = (57.421875, 94.921875)
B(0.5757575757575758) = (49.51171875, 105.76171875)
B(0.6060606060606061) = (42.1875, 117.1875)
B(0.6363636363636364) = (35.44921875, 129.19921875)
B(0.6666666666666666) = (29.296875, 141.796875)
B(0.696969696969697) = (23.73046875, 154.98046875)
B(0.7272727272727273) = (18.75, 168.75)
B(0.7575757575757576) = (14.35546875, 183.10546875)
B(0.7878787878787878) = (10.546875, 198.046875)
B(0.8181818181818182) = (7.32421875, 213.57421875)
B(0.8484848484848485) = (4.6875, 229.6875)
B(0.8787878787878788) = (2.63671875, 246.38671875)
B(0.9090909090909091) = (1.171875, 263.671875)
B(0.9393939393939394) = (0.29296875, 281.54296875)
B(0.9696969696969697) = (0.0, 300.0)
Calculation duration: 91.7638430595398 seconds
□

```



### 3.1.2 Test Case 2 Divide and Conquer

Input:

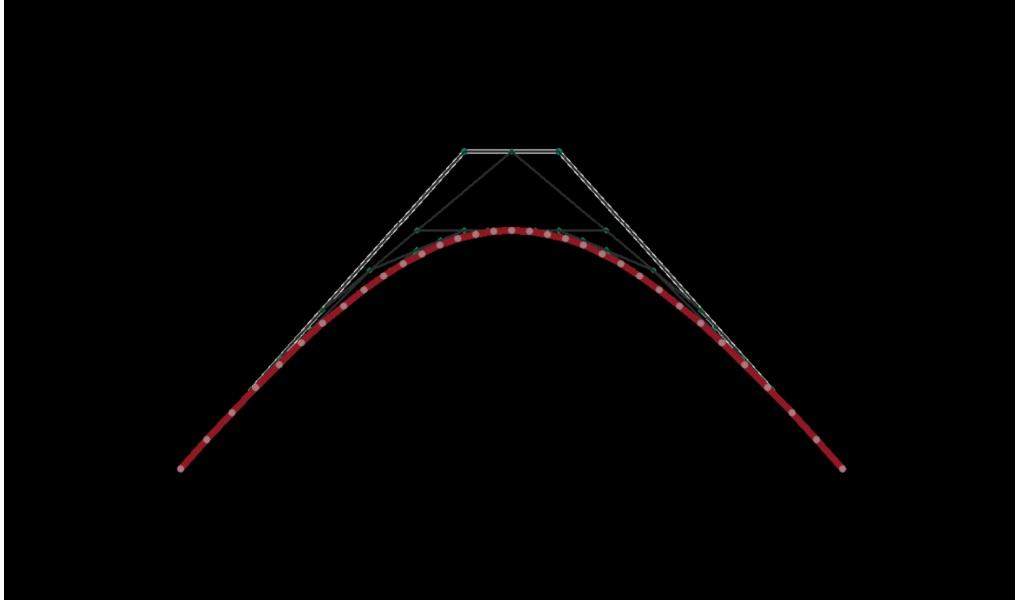
```
Bezier_100_200_100_200_7_3_3_main.py
Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 3
Input coordinate (x, y) of control point P_ 0
100 100
Input coordinate (x, y) of control point P_ 1
400 500
Input coordinate (x, y) of control point P_ 2
500 500
Input coordinate (x, y) of control point P_ 3
800 100
Enter number of steps/iteration for brute force / DNC: 5
```

Output:

```

Result:
B(0.0) = (100.0, 100.0)
B(0.0303030303030304) = (127.55126953124999, 136.32812500000003)
B(0.0606060606060601) = (154.00390625, 170.3125)
B(0.0909090909090901) = (179.43115234375, 201.95312500000003)
B(0.1212121212121212) = (203.90624999999994, 231.25)
B(0.151515151515152) = (227.50244140624994, 258.203125)
B(0.181818181818182) = (250.29296874999994, 282.8125)
B(0.21212121212121213) = (272.3510742187499, 305.0781249999999)
B(0.2424242424242423) = (293.75, 324.9999999999994)
B(0.2727272727272727) = (314.56298828125, 342.57812499999994)
B(0.303030303030304) = (334.86328125, 357.8124999999994)
B(0.333333333333333) = (354.72412109375, 370.7031249999999)
B(0.363636363636365) = (374.2187499999994, 381.25)
B(0.393939393939393) = (393.42041015625, 389.4531249999999)
B(0.42424242424242425) = (412.40234375, 395.3124999999999)
B(0.454545454545453) = (431.23779296874994, 398.8281249999999)
B(0.484848484848486) = (450.0, 399.9999999999994)
B(0.51515151515151) = (468.76220703124994, 398.8281249999999)
B(0.545454545454545) = (487.5976562499999, 395.3124999999999)
B(0.5757575757575758) = (506.57958984374994, 389.4531249999999)
B(0.6060606060606061) = (525.78125, 381.25)
B(0.6363636363636364) = (545.27587890625, 370.7031249999999)
B(0.6666666666666666) = (565.1367187500001, 357.8124999999994)
B(0.696969696969697) = (585.4370117187501, 342.5781249999999)
B(0.7272727272727273) = (606.25, 324.9999999999994)
B(0.7575757575757576) = (627.64892578125, 305.0781249999999)
B(0.78787878787878) = (649.70703125, 282.8125)
B(0.81818181818182) = (672.4975585937499, 258.203125)
B(0.848484848484845) = (696.09375, 231.25)
B(0.8787878787878788) = (720.56884765625, 201.95312500000003)
B(0.9090909090909091) = (745.99609375, 170.3125)
B(0.9393939393939394) = (772.44873046875, 136.32812500000003)
B(0.9696969696969697) = (800.0, 100.0)
Calculation duration: 119.09569048881531 seconds

```



### 3.1.3 Test Case 3 Divide and Conquer

Input:

```

Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 4
Input coordinate (x, y) of control point P_ 0
-50 -50
Input coordinate (x, y) of control point P_ 1
0 0
Input coordinate (x, y) of control point P_ 2
120 200
Input coordinate (x, y) of control point P_ 3
150 100
Input coordinate (x, y) of control point P_ 4
200 50
Enter number of steps/iteration for brute force / DNC: 4
Result:

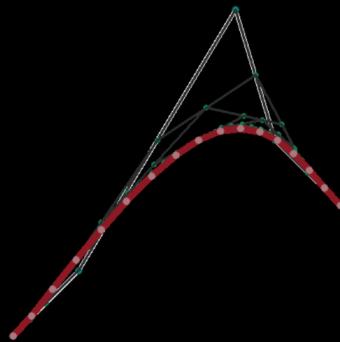
```

Output:

```

Result:
B(0.0) = (-50.0, -50.0)
B(0.058823529411764705) = (-36.011505126953125, -34.41162109375)
B(0.11764705882352941) = (-19.621582031249996, -14.2578125)
B(0.17647058823529413) = (-1.6194152832031206, 8.26416015625)
B(0.23529411764705882) = (17.304687500000007, 31.25)
B(0.29411764705882354) = (36.55929565429689, 53.08837890625)
B(0.35294117647058826) = (55.651855468750014, 72.4609375)
B(0.4117647058823529) = (74.18869018554688, 88.34228515625)
B(0.47058823529411764) = (91.87500000000001, 100.0)
B(0.5294117647058824) = (108.51486206054688, 106.99462890625)
B(0.5882352941176471) = (124.01123046875, 109.1796875)
B(0.6470588235294118) = (138.36593627929688, 106.70166015625)
B(0.7058823529411765) = (151.6796875, 100.0)
B(0.7647058823529411) = (164.1520690917969, 89.80712890625)
B(0.8235294117647058) = (176.08154296875, 77.1484375)
B(0.8823529411764706) = (187.86544799804688, 63.34228515625)
B(0.9411764705882353) = (200.0, 50.0)
Calculation duration: 38.88990020751953 seconds

```



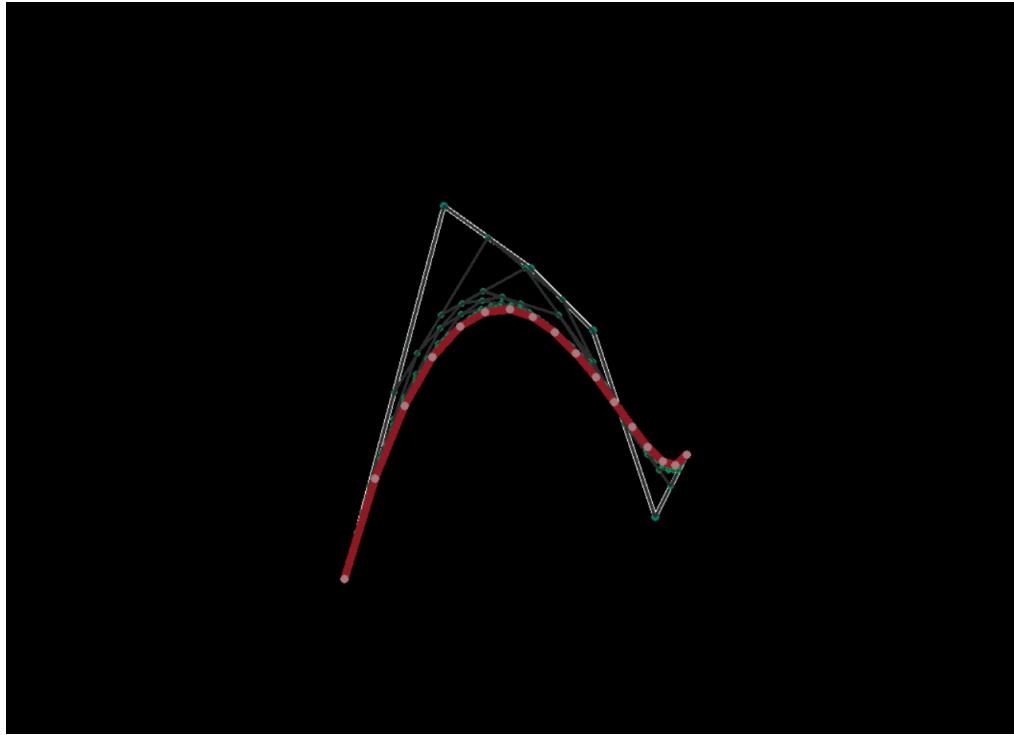
### 3.1.4 Test Case 4 Divide and Conquer

Input:

```
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 5
Input coordinate (x, y) of control point P_ 0
-200 -200
Input coordinate (x, y) of control point P_ 1
-120 100
Input coordinate (x, y) of control point P_ 2
-50 50
Input coordinate (x, y) of control point P_ 3
0 0
Input coordinate (x, y) of control point P_ 4
50 -150
Input coordinate (x, y) of control point P_ 5
75 -100
Enter number of steps/iteration for brute force / DNC: 4
DNC
```

Output:

```
Result:
B(0.0) = (-200.0, -200.0)
B(0.058823529411764705) = (-175.41282176971436, -119.10090446472168)
B(0.11764705882352941) = (-151.72348022460938, -60.87493896484375)
B(0.17647058823529413) = (-129.00679111480713, -21.309518814086914)
B(0.23529411764705882) = (-107.2998046875, 2.978515625)
B(0.29411764705882354) = (-86.61038875579834, 14.840173721313477)
B(0.35294117647058826) = (-66.92581176757812, 16.69158935546875)
B(0.4117647058823529) = (-48.22132587432861, 10.611295700073242)
B(0.47058823529411764) = (-30.46875, -1.5625)
B(0.5294117647058824) = (-13.645052909851074, -18.134641647338867)
B(0.5882352941176471) = (2.259063720703125, -37.45574951171875)
B(0.6470588235294118) = (17.230582237243652, -57.8249454498291)
B(0.7058823529411765) = (31.2255859375, -77.392578125)
B(0.7647058823529411) = (44.16067600250244, -94.06294822692871)
B(0.8235294117647058) = (55.904388427734375, -105.39703369140625)
B(0.8823529411764706) = (66.26861095428467, -108.51521492004395)
B(0.9411764705882353) = (75.0, -100.0)
Calculation duration: 50.61630344390869 seconds
```



### 3.1.5 Test Case 5 Divide and Conquer

Input:

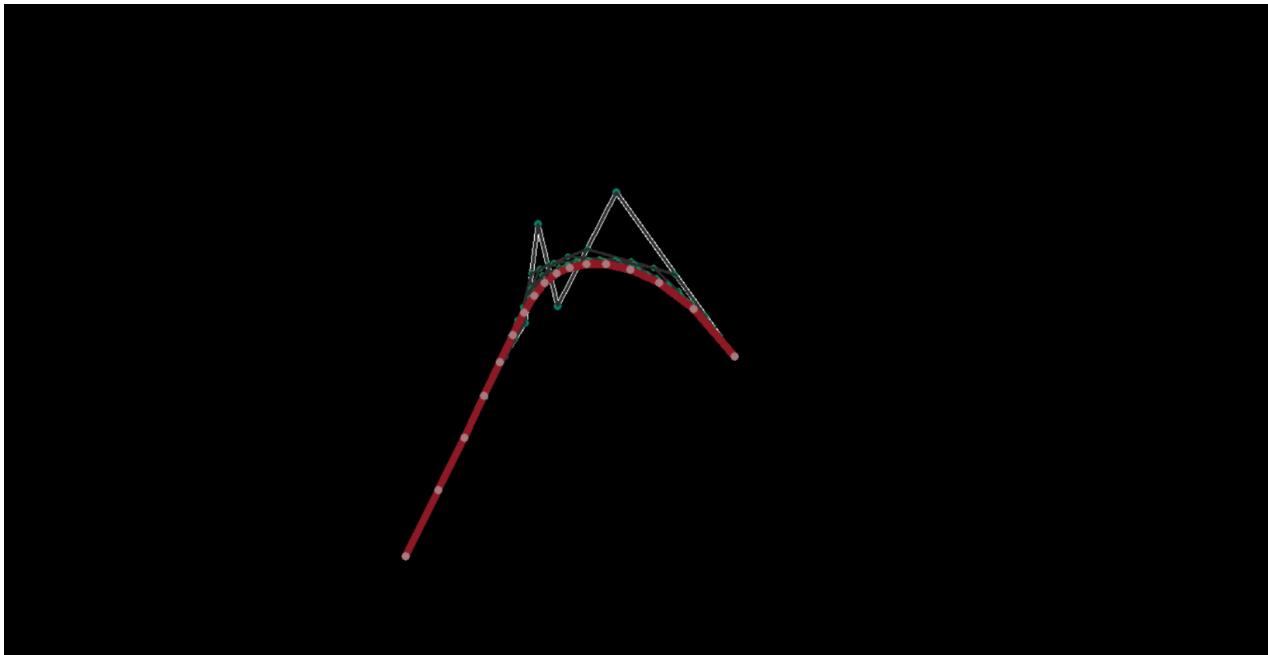
```
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 6
Input coordinate (x, y) of control point P_ 0
-100 -150
Input coordinate (x, y) of control point P_ 1
-25 0
Input coordinate (x, y) of control point P_ 2
-10 25
Input coordinate (x, y) of control point P_ 3
0 100
Input coordinate (x, y) of control point P_ 4
15 38
Input coordinate (x, y) of control point P_ 5
59 124
Input coordinate (x, y) of control point P_ 6
148 0
Enter number of steps/iteration for brute force / DNC: 4
Result.
```

Output:

```

Result:
B(0.0) = (-100.0, -150.0)
B(0.058823529411764705) = (-75.13203722238539, -100.29792577028275)
B(0.11764705882352941) = (-55.818363189697266, -61.14134597778321)
B(0.17647058823529413) = (-40.770405352115624, -29.73299914598465)
B(0.23529411764705882) = (-28.873291015625, -4.300048828125003)
B(0.29411764705882354) = (-19.154025614261627, 16.269473731517788)
B(0.35294117647058826) = (-10.753147125244139, 32.706584930419915)
B(0.4117647058823529) = (-2.8998566269874573, 45.557622134685516)
B(0.47058823529411764) = (5.109374999999999, 55.29687499999999)
B(0.5294117647058824) = (13.92872422933578, 62.35557478666304)
B(0.5882352941176471) = (24.177455902099606, 67.06724166870116)
B(0.6470588235294118) = (36.45088809728622, 69.52939003705978)
B(0.7058823529411765) = (51.32788085937499, 69.381591796875)
B(0.7647058823529411) = (69.3748487830162, 65.49989765882492)
B(0.8235294117647058) = (91.14629745483398, 55.60761642456054)
B(0.8823529411764706) = (117.18188375234605, 35.80245226621627)
B(0.9411764705882353) = (148.0, 0.0)
Calculation duration: 70.5945258140564 seconds

```



### 3.1.6 Test Case 6 Divide and Conquer

Input:

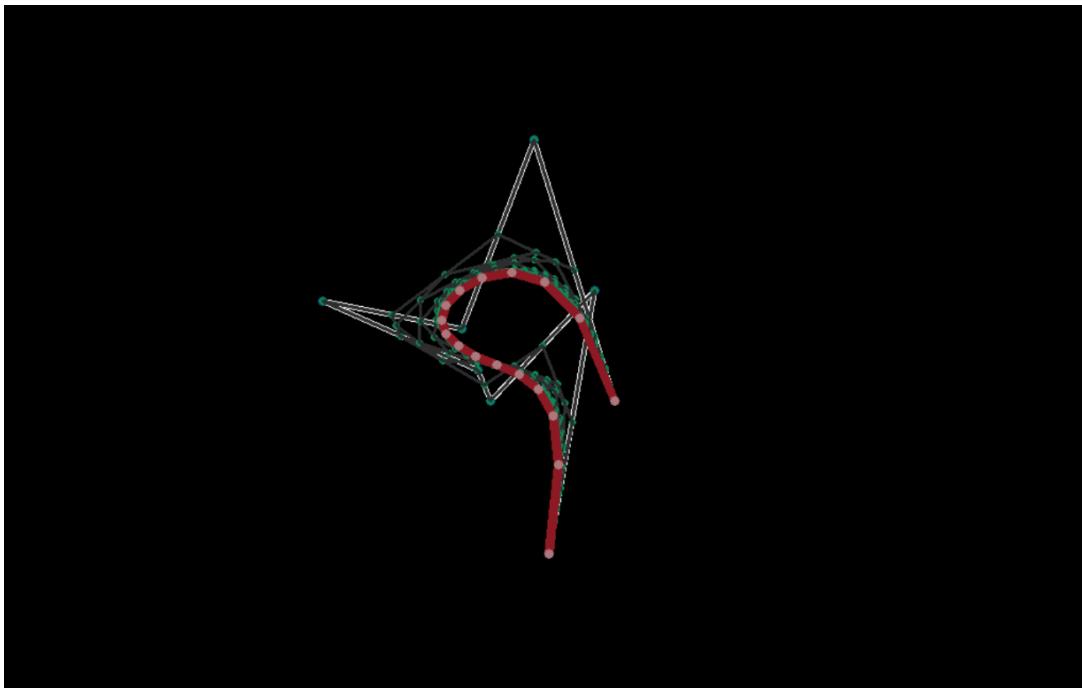
```

Choose algorithm:
1. Brute Force
2. Divide and Conquer
Enter number of algorithm: 2
Enter order of Bezier Curve (n): 7
Input coordinate (x, y) of control point P_ 0
100 0
Input coordinate (x, y) of control point P_ 1
35 210
Input coordinate (x, y) of control point P_ 2
-23 58
Input coordinate (x, y) of control point P_ 3
-135 80
Input coordinate (x, y) of control point P_ 4
-10 25
Input coordinate (x, y) of control point P_ 5
0 0
Input coordinate (x, y) of control point P_ 6
84 89
Input coordinate (x, y) of control point P_ 7
47 -123
Enter number of steps/iteration for brute force / DNC: 4

```

## Output:

```
Enter number of steps, iteration for brace 10,000, 0.001  
Result:  
B(0.0) = (100.0, 0.0)  
B(0.058823529411764705) = (71.78443294763565, 66.36176201328635)  
B(0.11764705882352941) = (43.678241729736335, 95.57825040817261)  
B(0.17647058823529413) = (16.79460102319718, 103.10448967292902)  
B(0.23529411764705882) = (-6.739990234374998, 98.86322021484374)  
B(0.29411764705882354) = (-24.808846414089203, 88.94013492390513)  
B(0.35294117647058826) = (-35.9151725769043, 76.90129709243773)  
B(0.4117647058823529) = (-39.45959085226059, 64.75960737839341)  
B(0.47058823529411764) = (-35.765625, 53.6171875)  
B(0.5294117647058824) = (-25.933652341365814, 44.01054834946991)  
B(0.5882352941176471) = (-11.603832244873043, 35.985410213470466)  
B(0.6470588235294118) = (5.291479647159578, 28.92804278805852)  
B(0.7058823529411765) = (22.705322265625004, 21.17999267578125)  
B(0.7647058823529411) = (38.50321584939957, 9.463066052645445)  
B(0.8235294117647058) = (50.259235382080085, -11.858565807342531)  
B(0.8823529411764706) = (54.74056154489518, -51.652270462363965)  
B(0.9411764705882353) = (47.0, -123.0)  
Calculation duration: 89.9988181591034 seconds
```





# BAB 4: Analisis Perbandingan Algoritma

## 4.1. Analisis Kompleksitas Algoritma

### 4.1.1 Kompleksitas Algoritma Brute Force

Misalkan banyaknya titik kontrol dilambangkan dengan  $N$  dan banyaknya langkah yang diinginkan untuk mengestimasi kurva adalah sebanyak  $K$ . Amati bahwa dalam implementasi strategi ini terdapat 2 buah looping. Loop pertama adalah untuk setiap langkah yang diambil, sehingga kompleksitasnya adalah  $O(K)$ . Sedangkan looping kedua merupakan bagian dari rumus general kurva bezier yaitu perhitungan sigma untuk suatu nilai  $i$  dari 0 sampai senilai orde kurvanya, sehingga kompleksitas waktu loop ini  $O(N)$ . Sadari bahwa karena menggunakan rumus langsung, maka perhitungan di dalam kedua loop ini dapat dilakukan secara langsung dalam  $O(1)$ . Jadi, kompleksitas waktu total algoritma ini adalah  $O(NK)$ .

### 4.1.1 Kompleksitas Algoritma Divide and Conquer

Asumsikan banyaknya titik kontrol adalah  $N$  dan  $K$  adalah banyaknya titik yang digunakan untuk mengestimasi kurva bezier. Tinjau kompleksitas waktu untuk melakukan iterasi pertama untuk suatu kurva dengan  $N$  titik kontrol, misalkan  $T(N)$ . Amati bahwa kita dapat membentuk  $N-1$  titik tengah dari setiap 2 pasangan titik kontrol, yang kemudian dapat menjadi titik kontrol suatu kurva dengan orde 1 lebih kecil. Hal ini dapat kita rekursikan hingga memperoleh suatu kurva dengan orde 1 dan diperoleh 1 titik hasil iterasi pertama. Relasi rekurens tersebut dapat dinyatakan sebagai

$$T(N) = \begin{cases} 1 & N = 2 \\ T(N-1) + N - 1 & N \geq 2 \end{cases}$$

Dengan sedikit aljabar akan diperoleh

$$T(N) = N - 1 + N - 2 + \dots + 2 + T(2) = \frac{N(N-1)}{2}$$

Akibatnya, bisa disimpulkan bahwa kompleksitas waktu untuk melakukan iterasi pertama bagi kurva dengan  $N$  titik kontrol adalah  $O(N^2)$ .

Sekarang kita tinjau kompleksitas waktu untuk membentuk  $K$  buah titik yang akan mengestimasi kurva, sebut saja  $T(K)$ . Amati bahwa pada setiap iterasi, kita melakukan kembali proses mencari titik iterasi pertama secara rekursif pada sebelah kanan dan kiri titik yang baru ditemukan pada langkah sebelumnya. Hal ini mengakibatkan kita banyaknya titik yang perlu dibentuk pada masing-masing proses menjadi hanya setengahnya. Sedangkan, waktu yang diperlukan untuk menyatukan kembali kedua hasil tersebut hanyalah  $O(1)$  karena kita hanya perlu menyatukan array. Sadari juga bahwa apabila kita hanya perlu membuat 1 titik, maka

kompleksitasnya adalah  $T(N)$  dengan  $N$  banyaknya titik kontrol, seperti yang telah kita hitung sebelumnya. Jadi, relasi rekurens kasus ini dapat ditulis dengan

$$T(K) = \begin{cases} T(N) & K = 1 \\ 2T\left(\frac{K}{2}\right) + 1 & K > 1 \end{cases}$$

Dengan menggunakan teorema master, diperoleh

$$a = 2, b = 2, c = 1, d = 0$$

Dimana kita punya  $a > b^d$ . Jadi, bisa disimpulkan bahwa kompleksitas waktunya adalah  $O(K)$ , dan secara keseluruhan strategi membutuhkan waktu  $O(KN^2)$ .

## 4.2. Analisis Hasil Kedua Strategi

Perlu ditegaskan bahwa pada implementasi program, waktu yang dikeluarkan dalam output melibatkan waktu untuk menggambar keseluruhan kurva. Amati bahwa pada setiap testcase, strategi divide and conquer memerlukan waktu yang jauh lebih banyak dibandingkan strategi brute force. Hal ini didukung dengan analisis kompleksitas waktu sebelumnya, dimana  $O(KN^2)$  jelas lebih buruk dibandingkan dengan  $O(NK)$ . Alasan lain mengapa hal ini dapat terjadi adalah akibat implementasi menggunakan *library* turtle. Dimana karena turtle perlu menggambarkan keseluruhan proses pembuatan kurva, yang memperlambat jalannya program karena banyaknya perjalanan turtle. Di lain pihak, pada strategi brute force, turtle tidak perlu banyak berpindah dan menggambar karena tidak ada proses yang perlu ditampilkan.

Namun, dengan analisis lebih lanjut dapat disadari bahwa kedua strategi memberikan solusi yang berbeda dengan perbedaan sangat kecil. Dengan menggunakan kalkulator kurva bezier, bisa disimpulkan bahwa divide and conquer dapat memberikan hasil yang lebih akurat. Hal ini terjadi akibat dalam implementasi strategi brute force, banyak dilakukan perhitungan yang melibatkan float, juga konversi antara integer menjadi float. Hal ini dapat mengakibatkan pembulatan, yang akhirnya mengurangi tingkat keakuratan solusi.

# BAB 5: Penutup

## 5.1. Kesimpulan

Dari hasil eksplorasi menggunakan 6 testcase, kami menyimpulkan bahwa strategi brute force masih lebih cepat dibandingkan strategi divide and conquer yang diimplementasikan. Hal ini didukung dengan kompleksitas waktu yang dihitung. Namun, secara akurasi strategi divide and conquer lebih unggul karena tidak banyak melibatkan perhitungan tipe data float yang kompleks. Jadi, kedua strategi ini memiliki kelebihan dan kekurangannya masing-masing.

## 5.2. Lampiran

Link Repository GitHub

[https://github.com/dewodt/Tucil2\\_13522011\\_13522037](https://github.com/dewodt/Tucil2_13522011_13522037)

Tabel Poin

No	Poin	Ya	Tidak
1.	Program berhasil dijalankan.	✓	
2.	Program dapat melakukan visualisasi kurva Bézier.	✓	
3.	Solusi yang diberikan program optimal.	✓	
4.	<b>[Bonus]</b> Program dapat membuat kurva untuk n titik kontrol.	✓	
5.	<b>[Bonus]</b> Program dapat melakukan visualisasi proses pembuatan kurva.	✓	