LAPORAN TUGAS KECIL 2
IF2211 Strategi Algoritma Semester II tahun 2023/2024
Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun Oleh:

Ariel Herfrison	13522002
Kristo Anugrah	13522024

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

Daftar Isi

1. Analisis dan Implementasi Algoritma Brute Force

1.1 Analisis

Algoritma Brute Force dapat dilakukan dengan mencari persamaan dari kurva bezier. Secara langkah per langkah, persamaan linier, kuadratik, kubik, dan seterusnya dapat dicari dengan menggunakan rekursif. Namun, proses dapat dipersingkat dengan memanfaatkan binomial expansion untuk mencari persamaan derajat n akhir. Setelah persamaan didapatkan, titik-titik pada kurva dapat diperoleh dengan memasukkan nilai "t" ke dalam persamaan.

1.2 Implementasi

Langkah-langkah:

1. Menerima n buah poin
2. Membuat persamaan dengan derajat n. Persamaan dibuat berdasarkan rumus binomial expansion sebagai berikut:

$$(x + y)^n = \sum_{i=0}^{n} ({}^nC_i x^{n-i} y^i)$$

3. Menghitung dan mengumpulkan titik-titik f(t) sebagai titik pada kurva bezier.

2. Analisis dan Implementasi Algoritma Divide and Conquer

2.1 Analisis

Diberikan n buah titik kontrol, program pertama akan menyalin n titik tersebut ke sebuah *list* baru A. Jika n sama dengan jumlah iterasi yang diinginkan, maka program akan mengembalikan *list* yang berisi titik pertama dan terakhir. Jika tidak, selama panjang *list* A lebih dari 1, program akan mencari titik tengah dari garis yang dibuat oleh dua titik yang bersebelahan pada *list* A. Titik yang telah dibuat tersebut kemudian dimasukkan ke dalam *list* lain, misalkan *list* B. Kemudian program akan mengambil titik pertama dan terakhir dari *list* A, dan memasukkan kedua titik tersebut ke dalam *list* C. Program kemudian akan menukar *list* A dan B, dan proses tersebut diulangi hingga panjang *list* A sama dengan 1.

Ketika panjang *list* A sama dengan 1, maka titik yang ada dalam *list* tersebut adalah salah satu titik solusi. Program akan membagi *list* C menjadi dua bagian kiri dan kanan. Program kemudian akan memanggil fungsi Divide and Conquer lagi secara rekursif untuk *list* C bagian kiri dan kanan. Hasil dari pemanggilan rekursif tersebut kemudian akan digabungkan dan dikembalikan sebagai hasil fungsi.

2.2 Implementasi

Langkah-langkah:

1. Terima n buah poin, jumlah iterasi sekarang, jumlah iterasi yang diinginkan
2. Jika n sama dengan jumlah iterasi yang diinginkan, kembalikan poin pertama dan terakhir
3. Jika tidak, lanjutkan ke langkah 4
4. Buat *list* kosong A, B, C
5. Misal u adalah panjang *list* A saat ini, lakukan langkah 6 sebanyak u - 1 kali

6. Hitung titik tengah garis yang dibentuk oleh titik dengan indeks saat ini dengan titik dengan indeks setelahnya, kemudian masukan titik ke *list* B.
7. Ambil titik pertama dan terakhir pada *list* A, kemudian masukkan ke *list* C.
8. Tukar *list* A dan B
9. Jika panjang A sama dengan 1, lanjut ke langkah 10. Jika tidak, kembali ke langkah 5
10. Bagi *list* C ke bagian kiri dan kanan, kemudian panggil fungsi ini kembali dengan argumen iterasi yang ditambah 1, jumlah iterasi yang diinginkan, serta *list* C bagian kiri dan kanan
11. Gabungkan dua hasil dari langkah 10, kemudian kembalikan hasil sebagai hasil fungsi

3. Source Code

3.1 Algoritma Brute Force

Header

```
struct Point {
   double x,y;
};

double factorial(double);

double combination(double, double);

struct Persamaan {
   Persamaan(vector <Point> &p);
   Point func(double x);
   void printInfo();

   std::vector <double> coefficients;
   std::vector <Point> points;
};

vector <Point> brute_force(int it,vector<Point> &p);
```

Implementasi

```
double factorial(double x){
   if (x==0 || x == 1) return 1;
   else return x*factorial(x-1);
}

double combination(double n, double r){
   return factorial(n)/(factorial(r)*factorial(n-r));
}
```

```
Persamaan::Persamaan(vector <Point> &p): coefficients(p.size()) {
   double n = p.size();
   for (double i=0;i<n;i++) {
      coefficients[i] = combination(n-1,i);
   }

   points = p;
}

Point Persamaan::func(double t) {
   double x,y;
   Point result;

   double n = coefficients.size();
   for (double i=0;i<n;i++) {
      x += coefficients[i]*pow((1-t),n-1-i)*pow(t,i)*points[i].x;
      y += coefficients[i]*pow((1-t),n-1-i)*pow(t,i)*points[i].y;
   }

   result.x = x;
   result.y = y;

   return result;
}

void Persamaan::printInfo() {
   for (auto& i:coefficients) {
      cout << i << " ";
   }
   cout << endl;

   for (auto& i:points) {
      cout << "(" << i.x << "," << i.y << ") ";
   }
   cout << endl;
}

vector <Point> brute_force(int it,vector <Point> &p) { // it = iteration
   int n = p.size();
   double steps = 1/(pow(2,it));
   Persamaan f(p);

   vector<Point> new_p;
   new_p.push_back(p[0]);
   for (int i=1;i<pow(2,it);i++){
```

```
        new_p.push_back(f.func(i*steps));
    }
    new_p.push_back(p.back());

    return new_p;
}
```

3.2 Algoritma Divide and Conquer

Header

```
struct Point {
    double x,y;
};

vector <Point> recurse(int it, int mIt, vector <Point> &p);
```

Implementasi

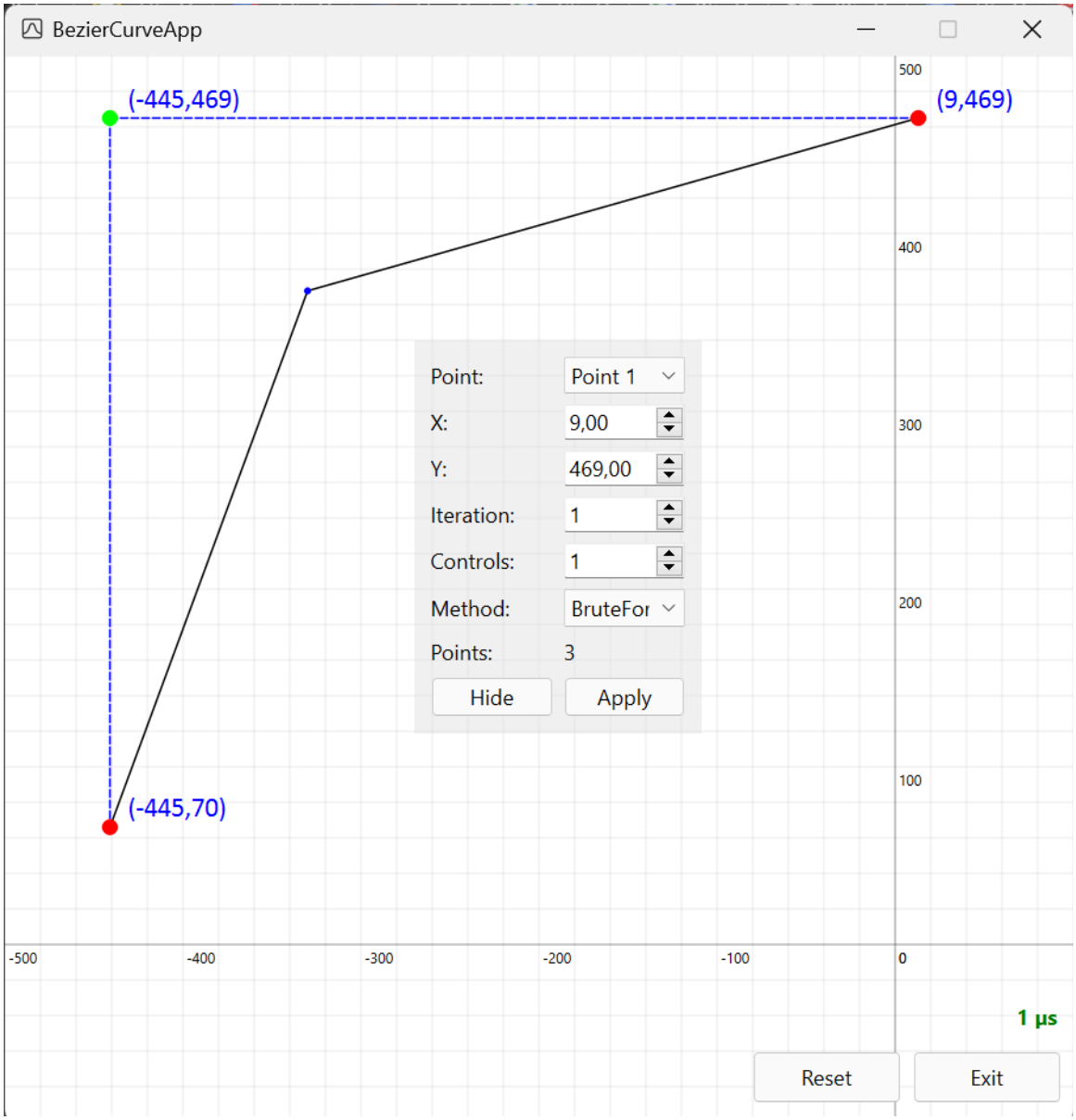
```
vector <Point> recurse(int it, int mIt, vector <Point> &p) {
    if(it == mIt) return {p[0], p.back()};
    vector<Point> pref, suf;
    while(p.size() >= 2){
        int n = p.size();
        pref.push_back(p[0]);
        suf.push_back(p.back());
        vector<Point> newP;
        for (int i=0;i<n-1;i++) {
            // find middle of 2 points
            double x = (p[i+1].x+p[i].x)/2;
            double y = (p[i+1].y+p[i].y)/2;
            Point new_p = {x, y};
            newP.push_back(new_p);
        }
        p.swap(newP);
    }

    double x = (pref.back().x+suf.back().x)/2;
    double y = (pref.back().y+suf.back().y)/2;
    Point midPoint = {x, y};
    suf.push_back(midPoint);
    reverse(suf.begin(), suf.end());
    pref.push_back(midPoint);
    vector<Point> lef = recurse(it + 1, mIt, pref);
    vector<Point> rig = recurse(it + 1, mIt, suf);
```

```
    lef.pop_back();
    for(auto &x: rig) lef.push_back(x);
    return lef;
}
```

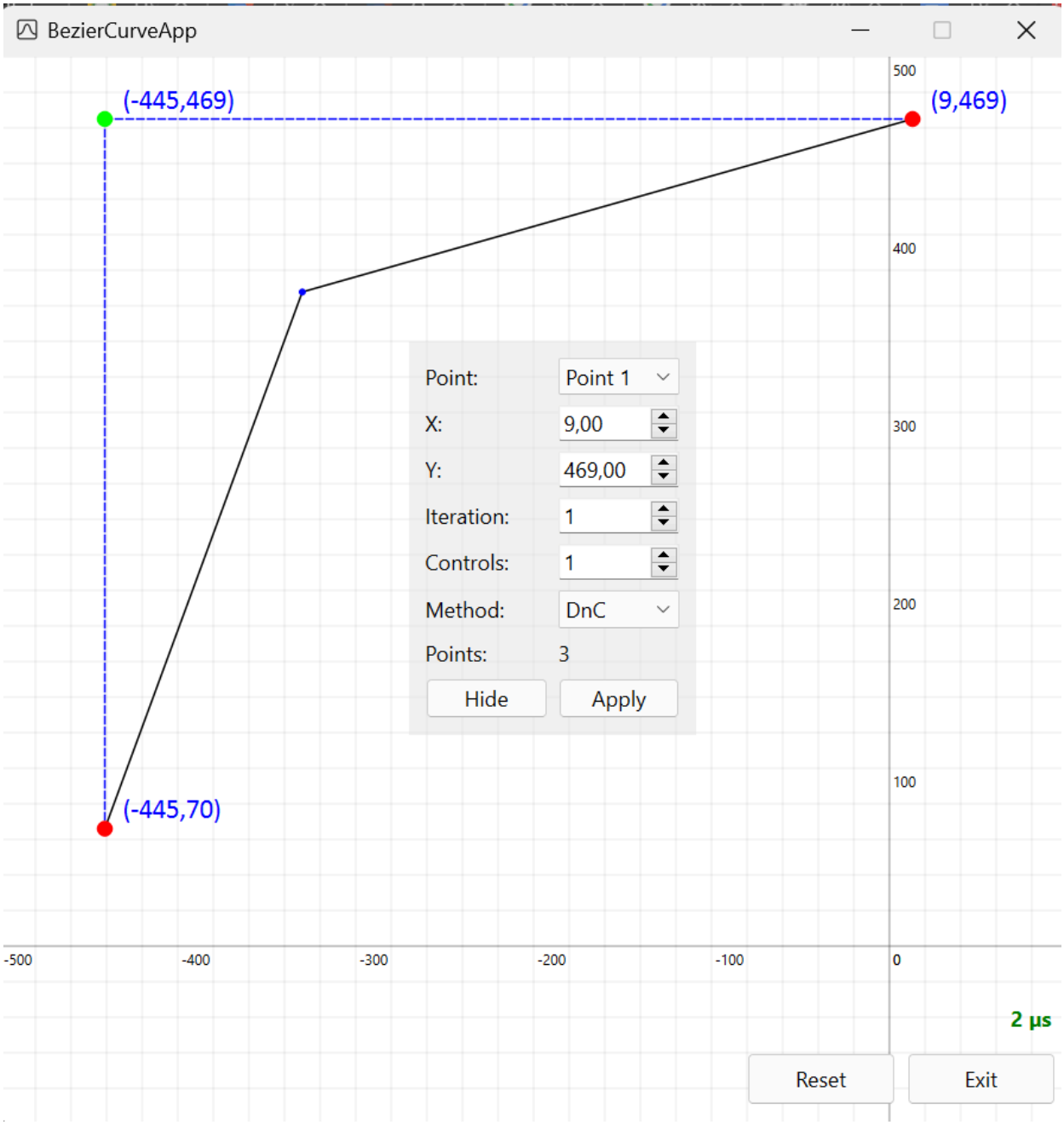
4. Uji Coba

1. Uji Coba 1

Brute Force:

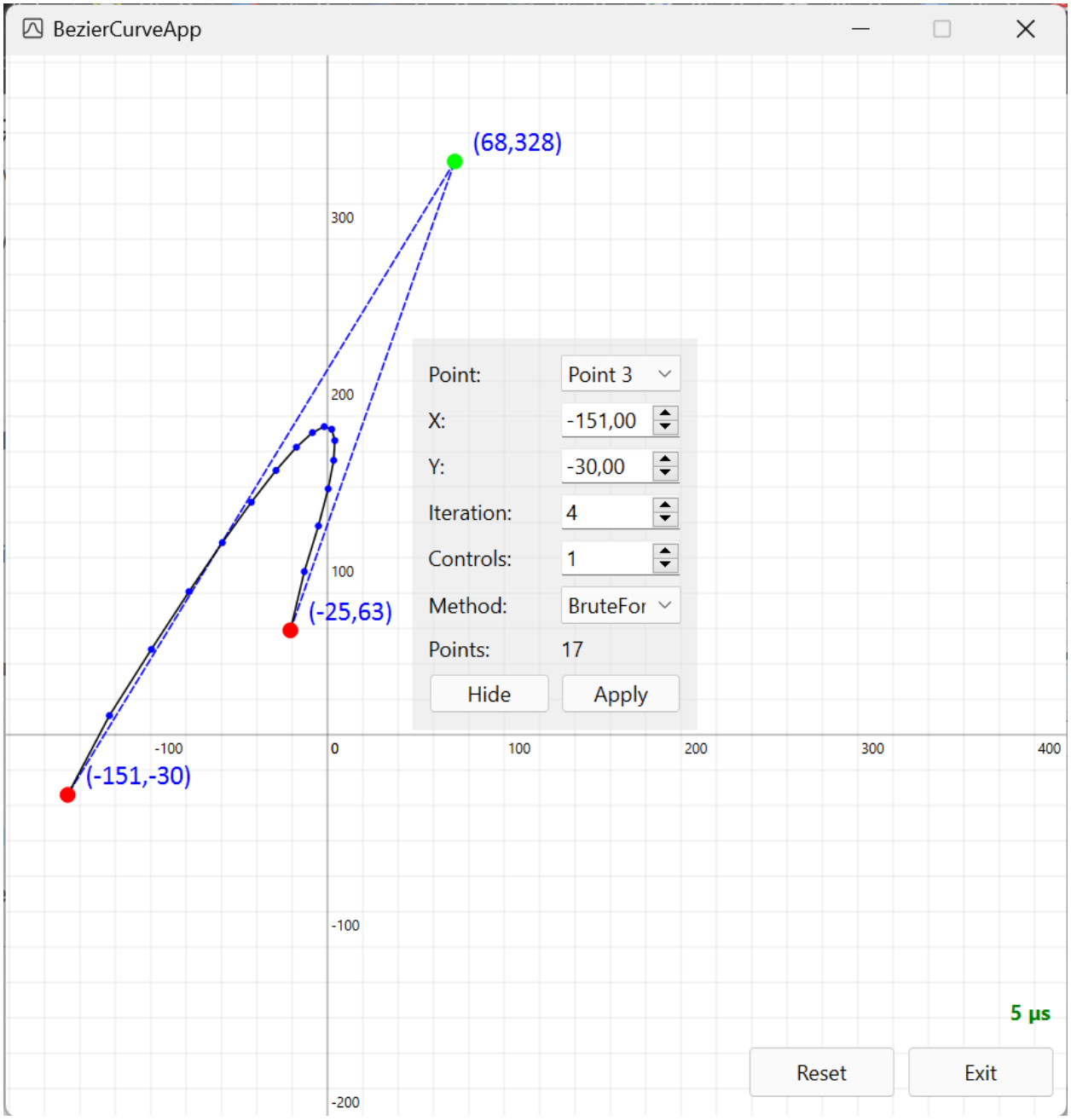


Divide and Conquer:

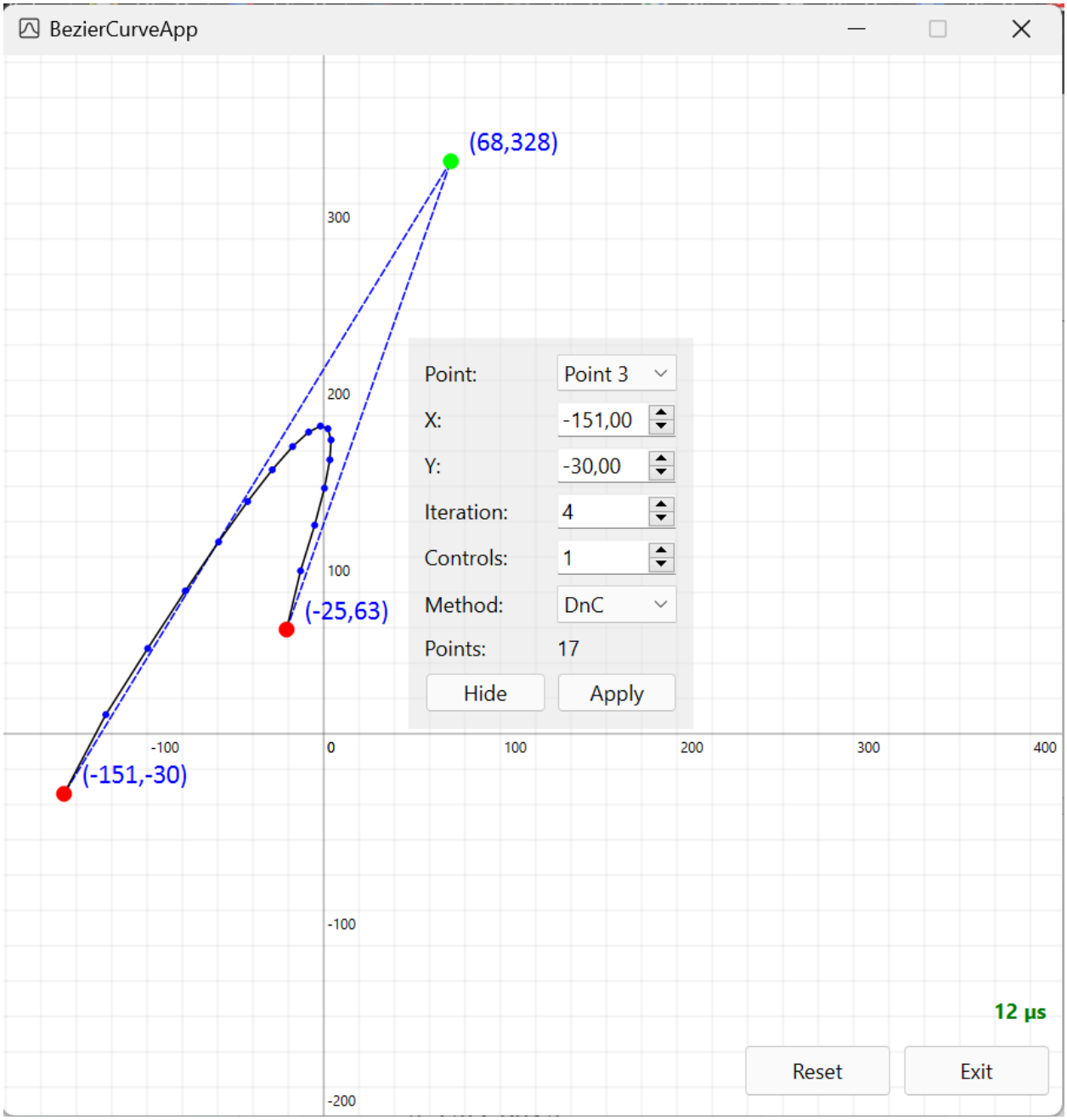


2. Uji Coba 2

Brute Force:

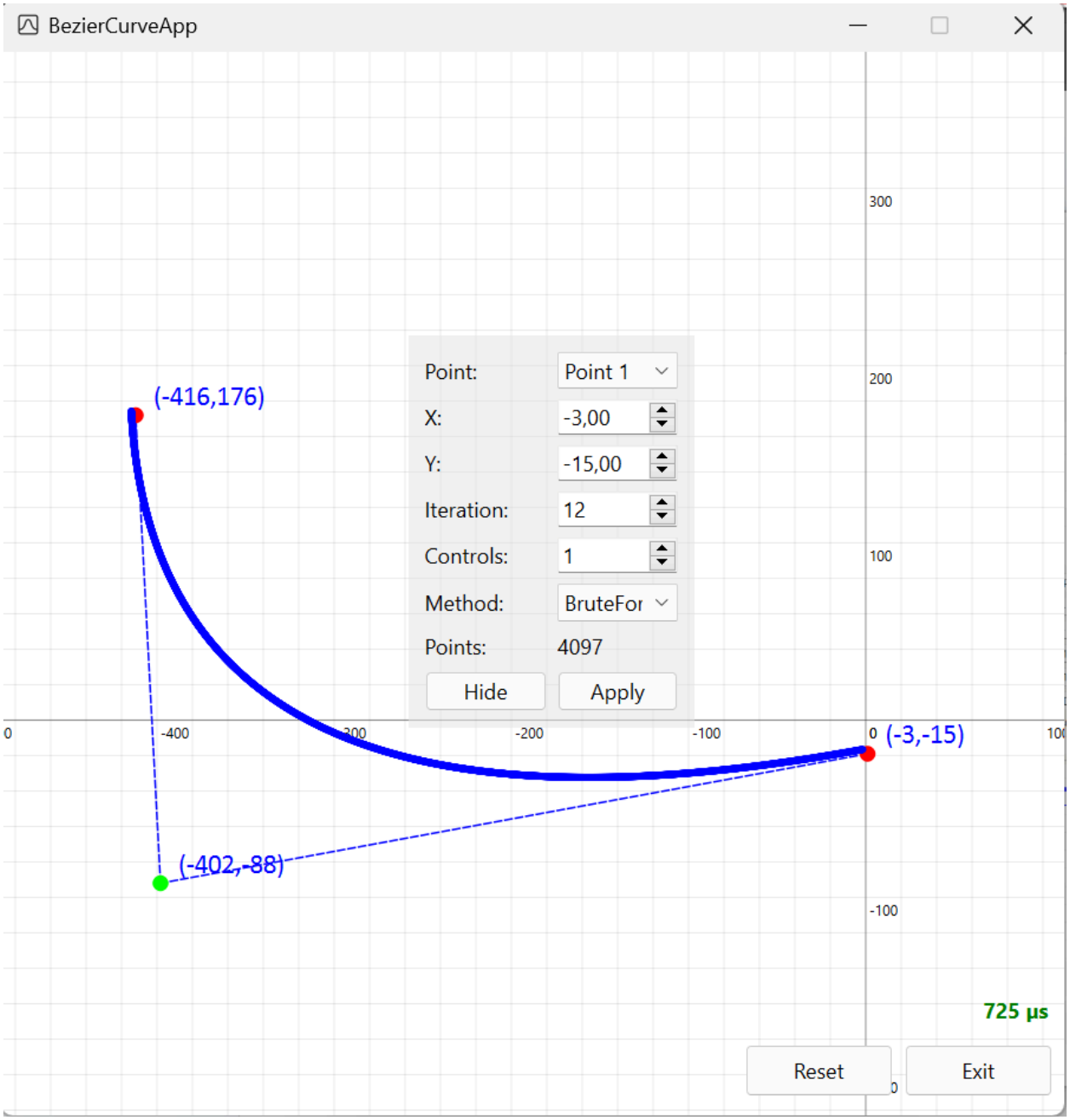


Divide and Conquer:

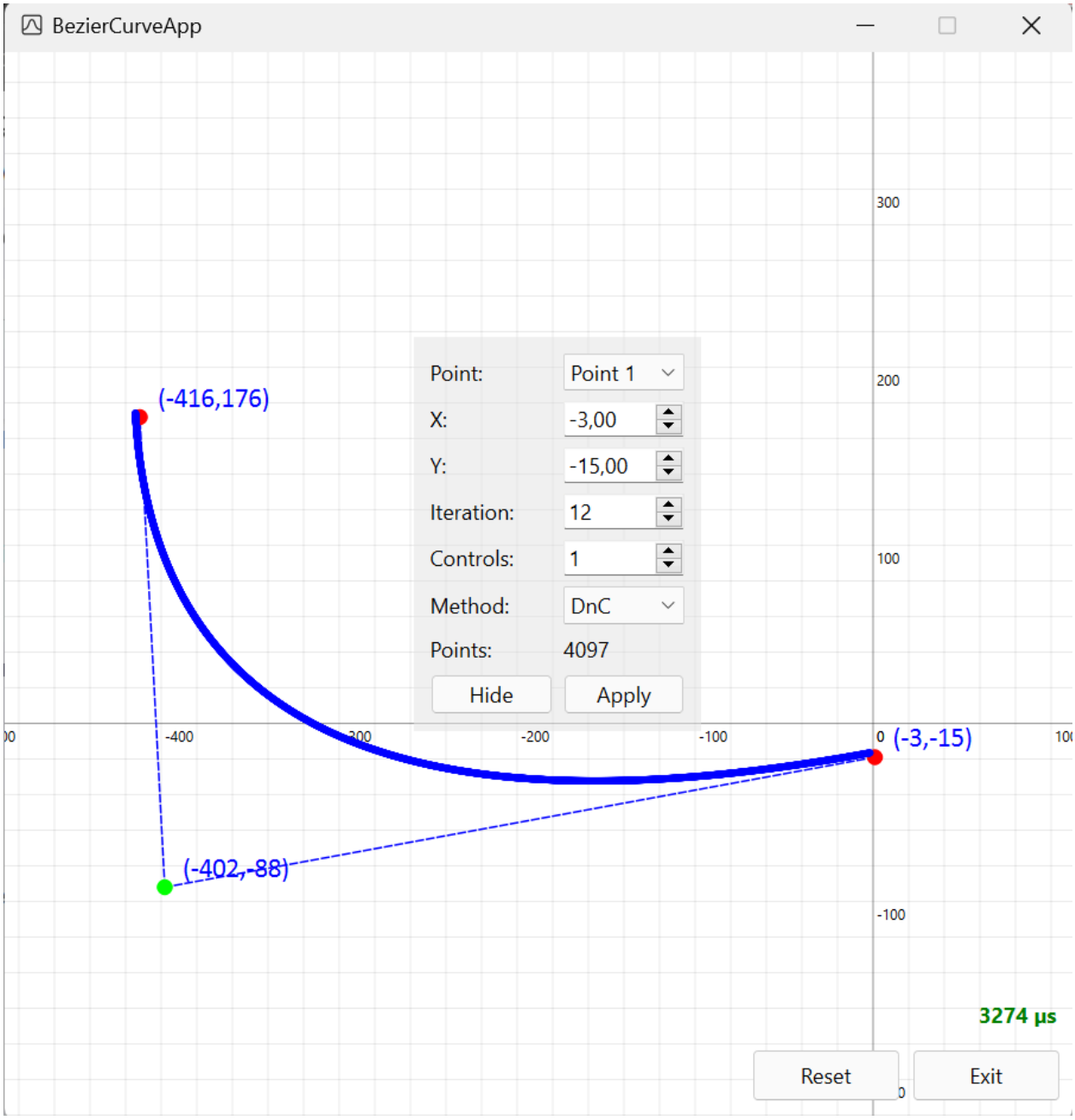


3. Uji Coba 3

Brute Force:

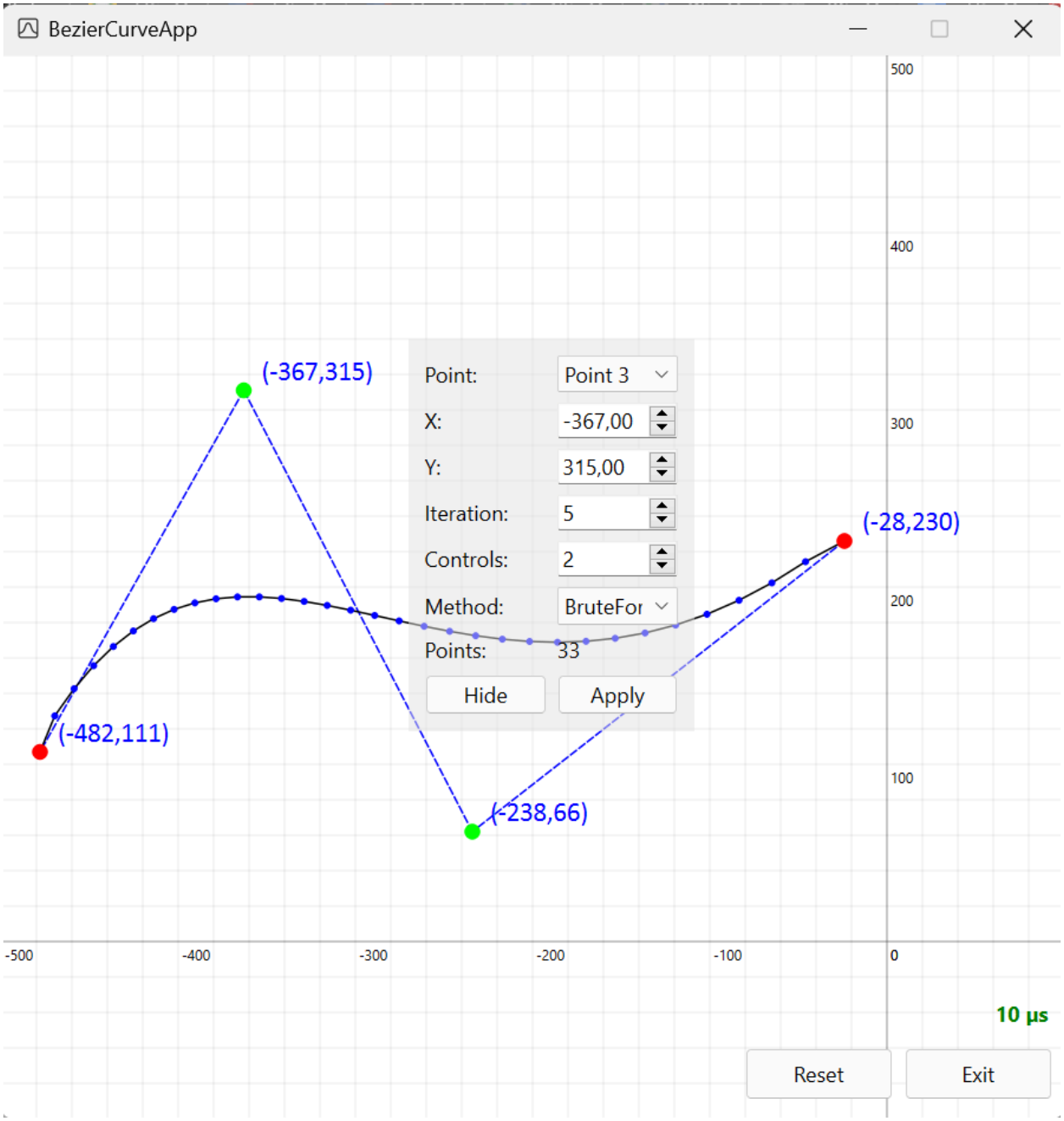


Divide and Conquer:

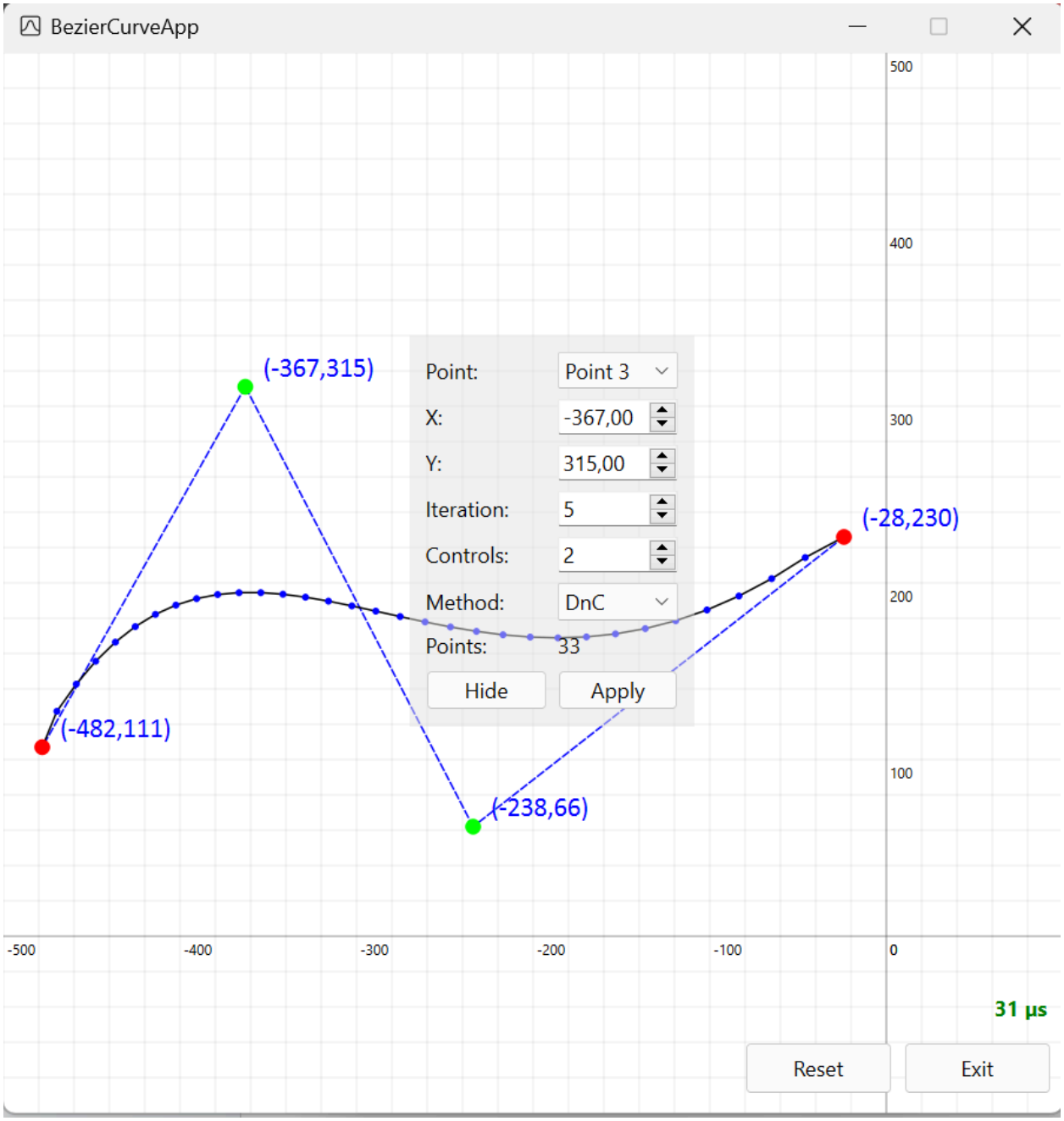


4. Uji Coba 4

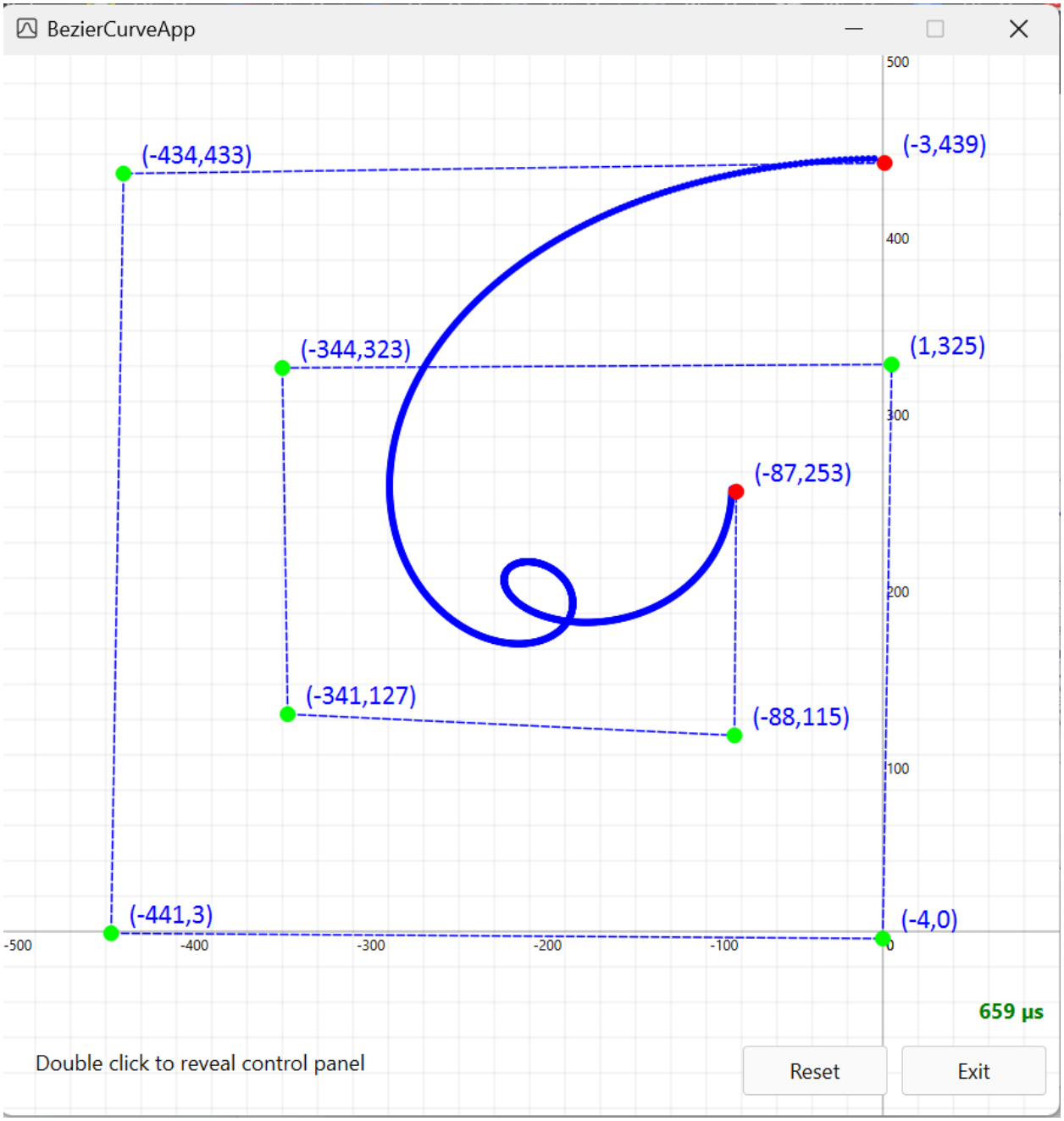
Brute Force:



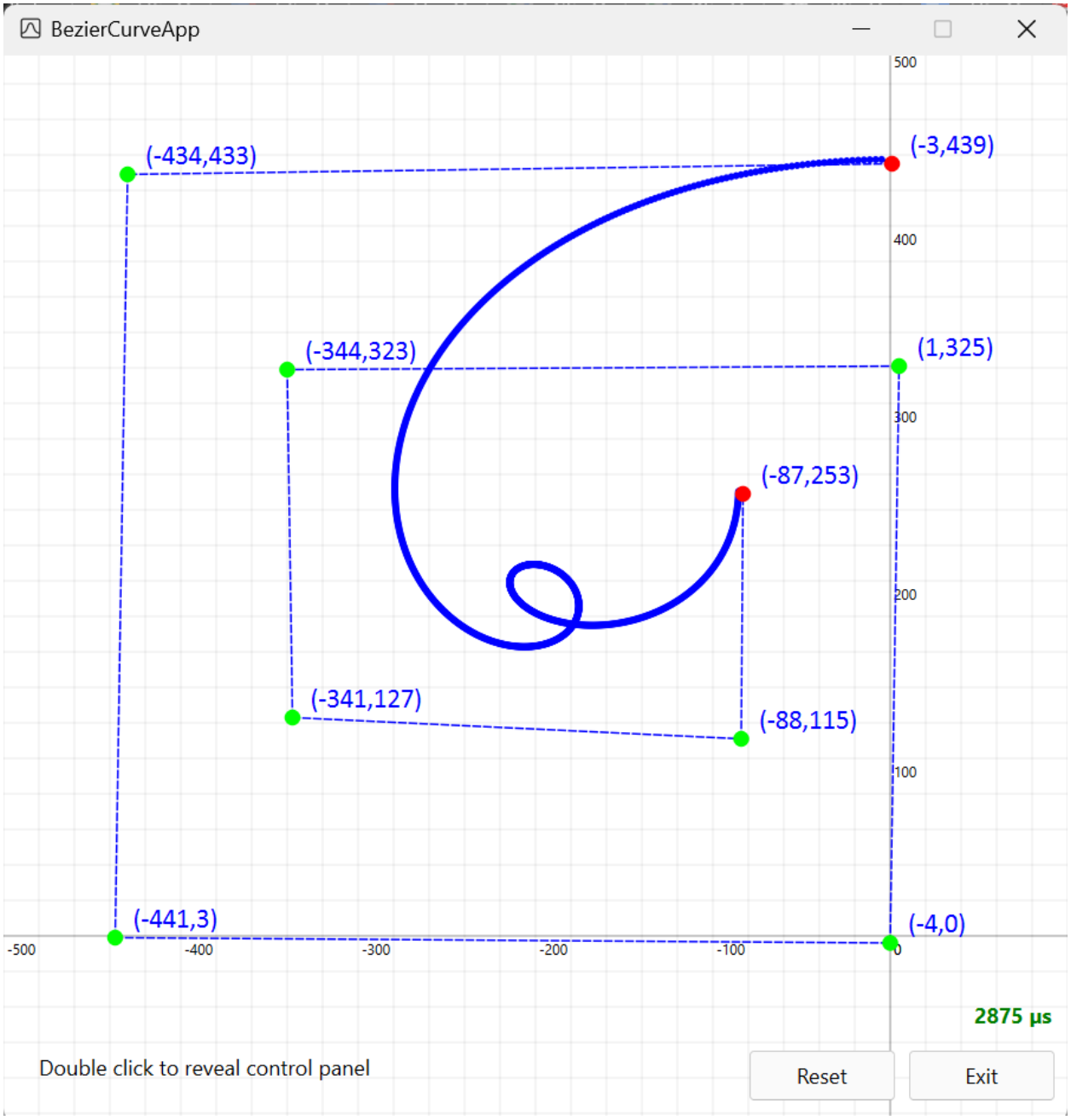
Divide and Conquer:



5. Uji Coba 5
Brute Force:

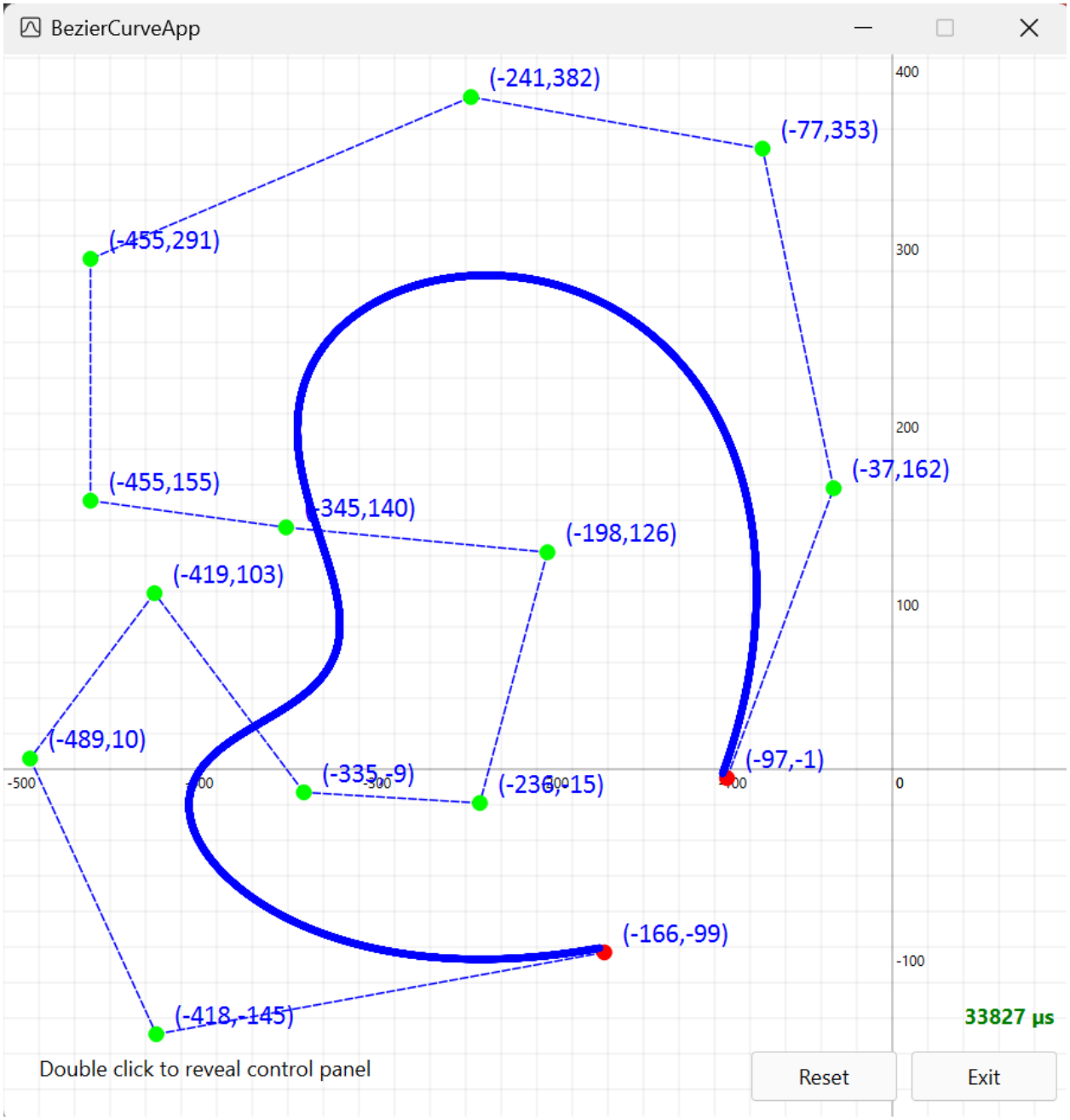


Divide and Conquer:

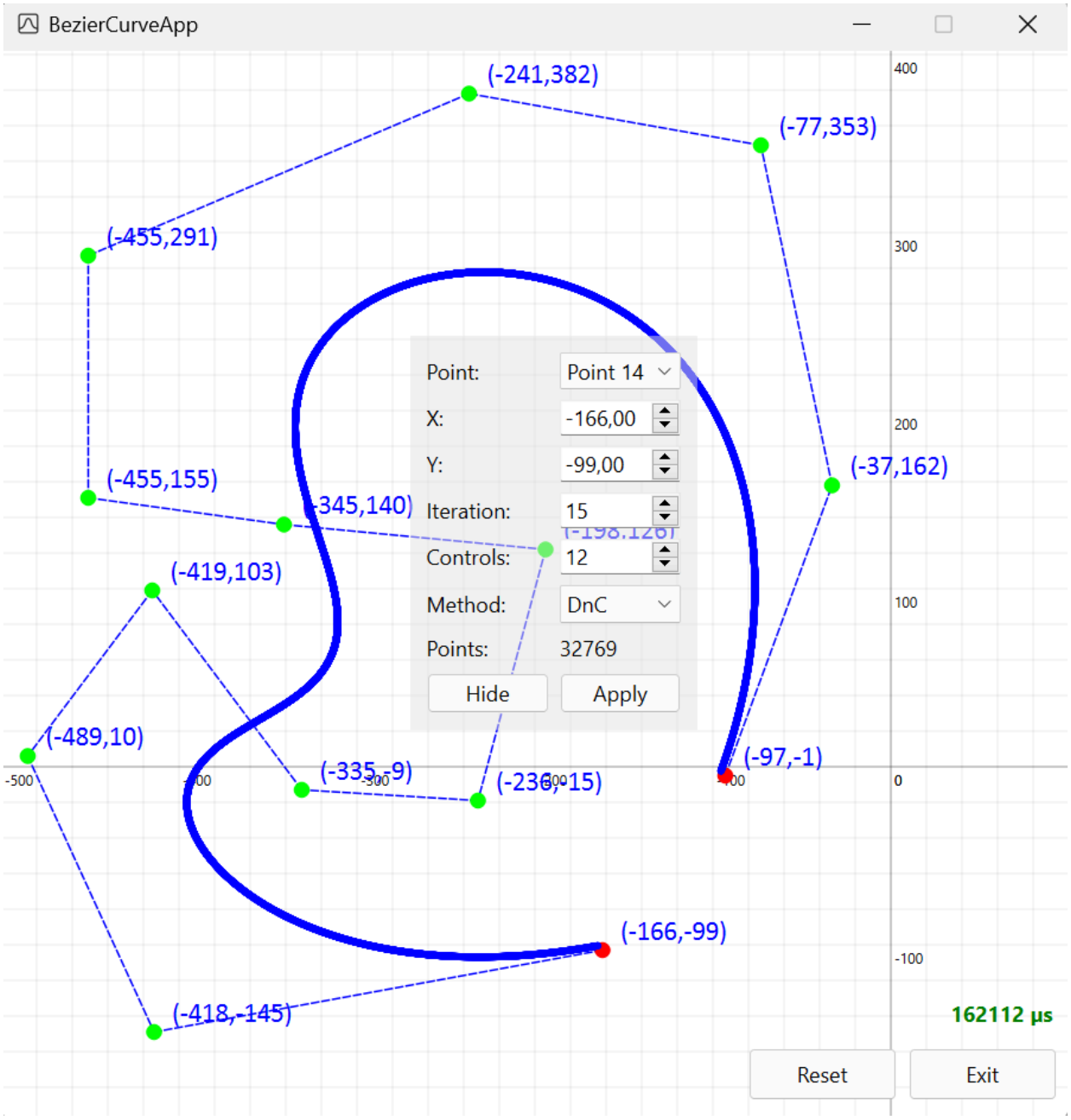


6. Uji Coba 6

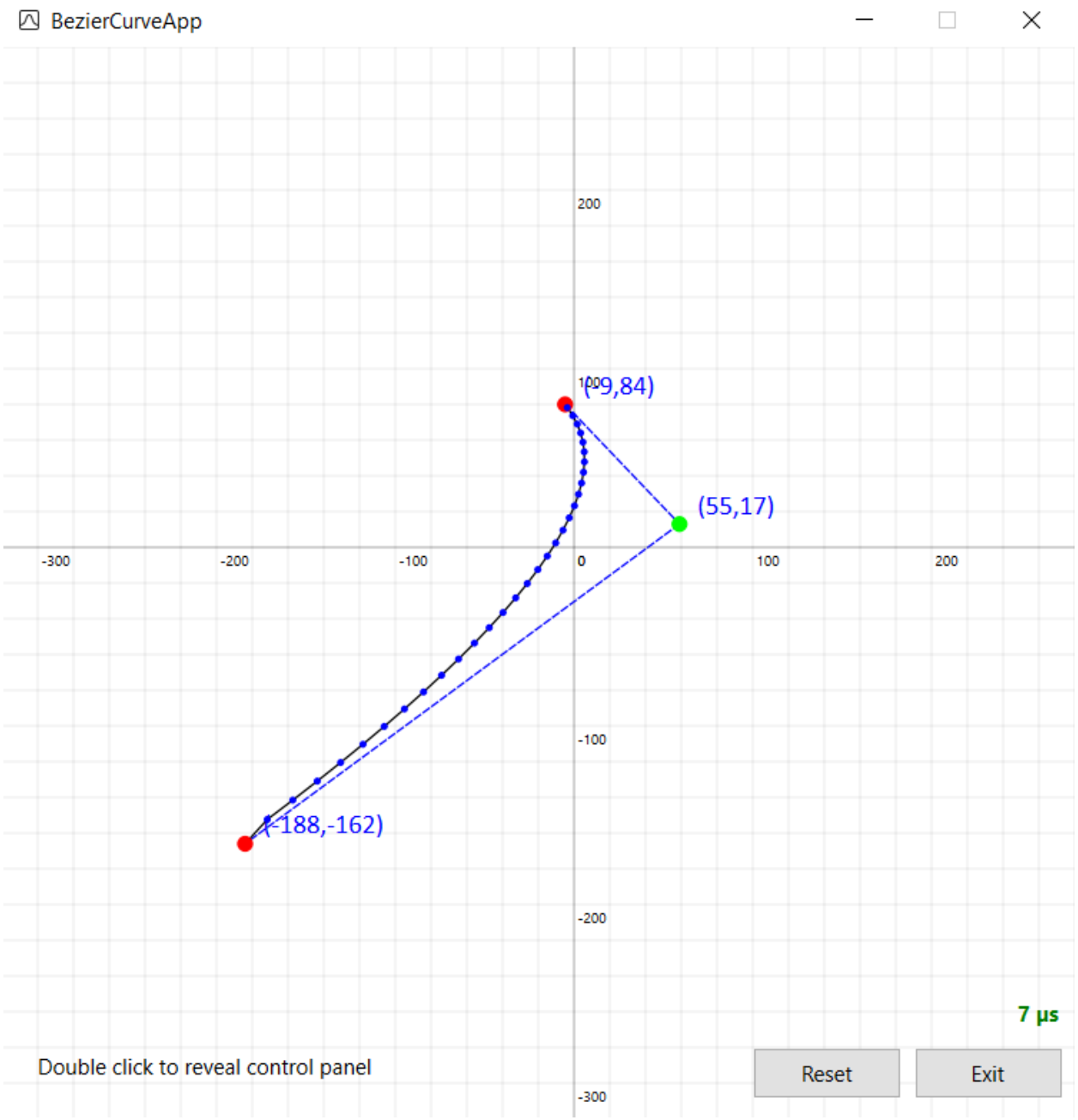
Brute Force:



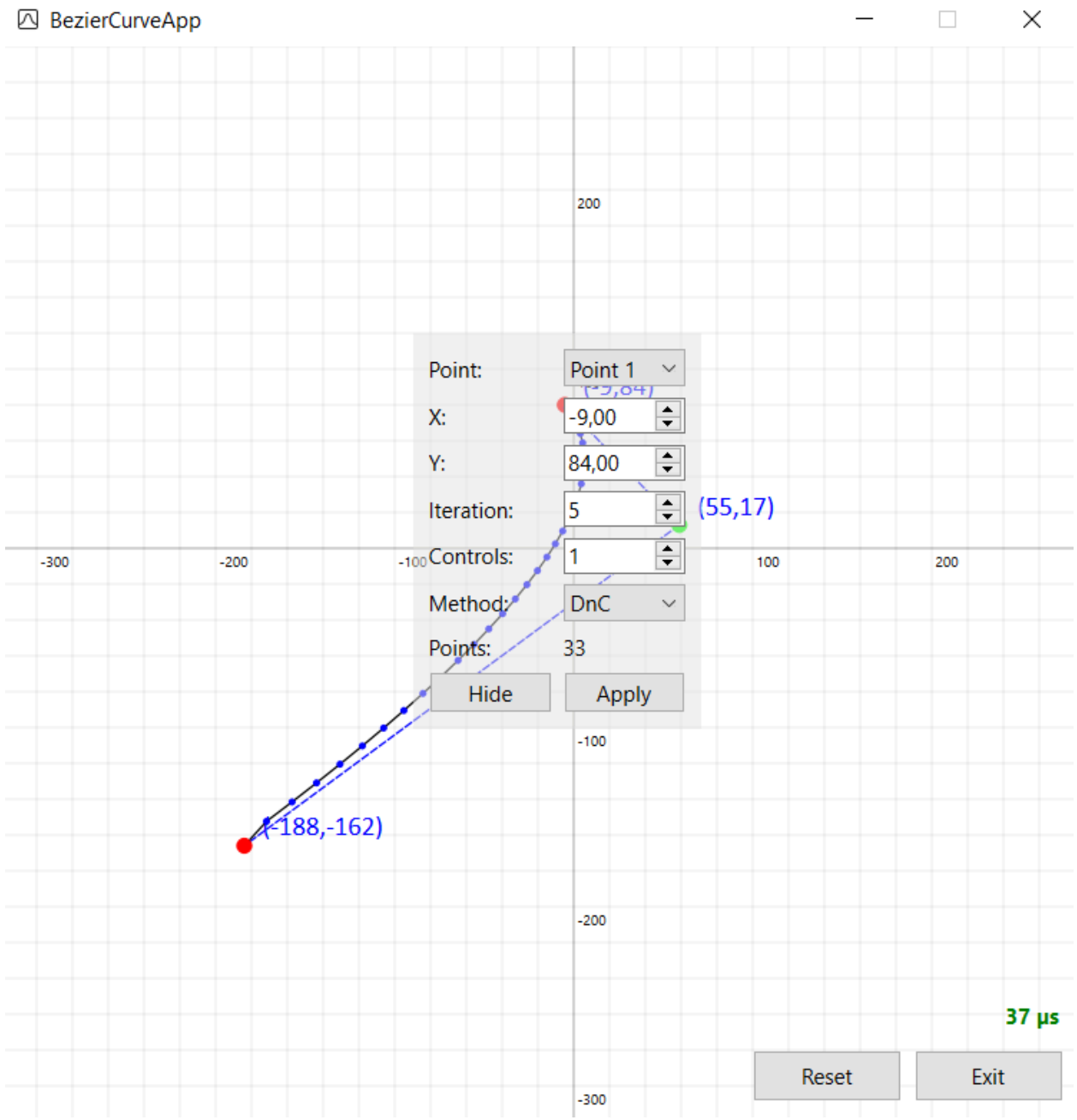
Divide and Conquer:



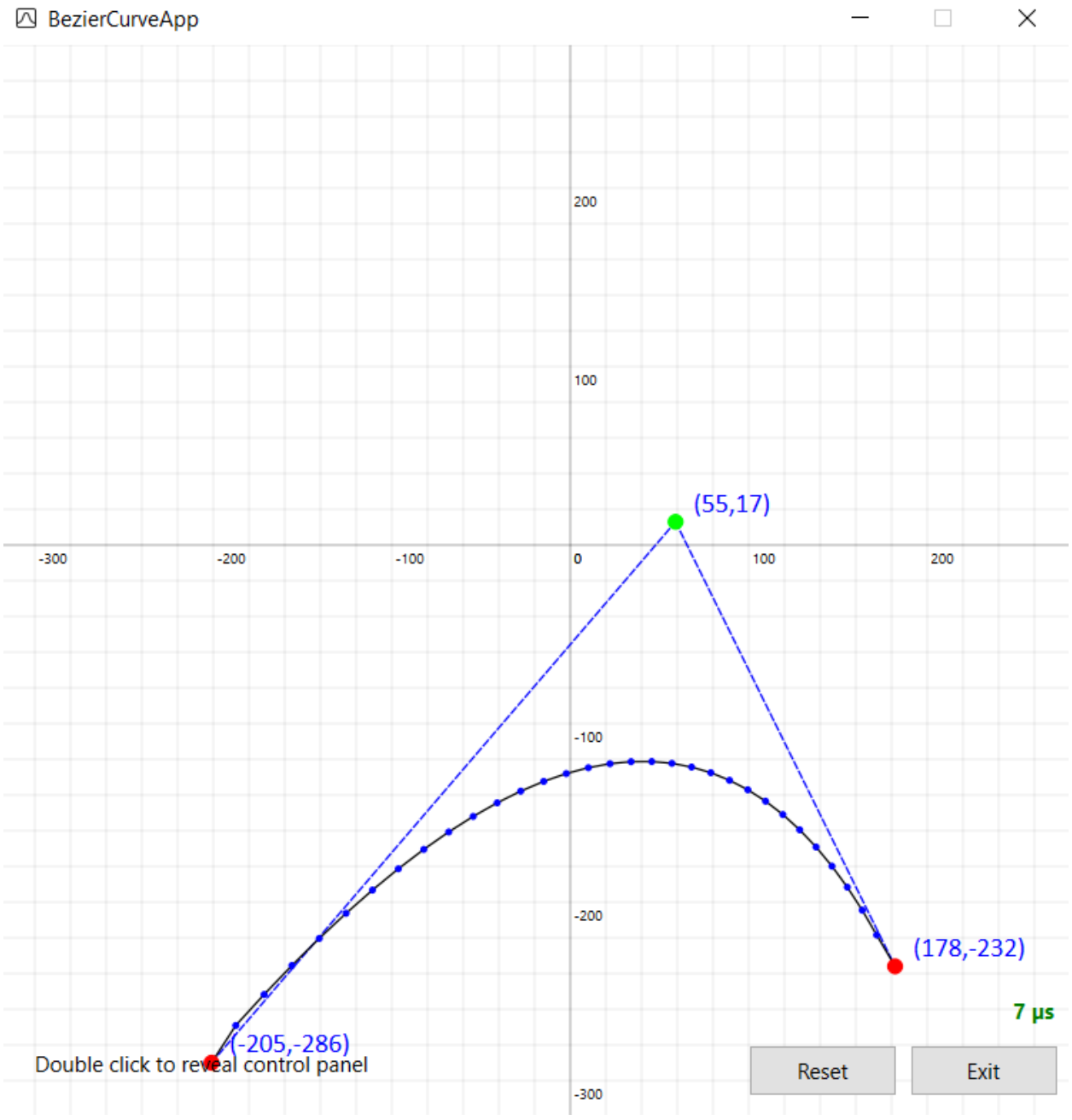
7. Uji Coba 7
Brute Force:



Divide and Conquer:

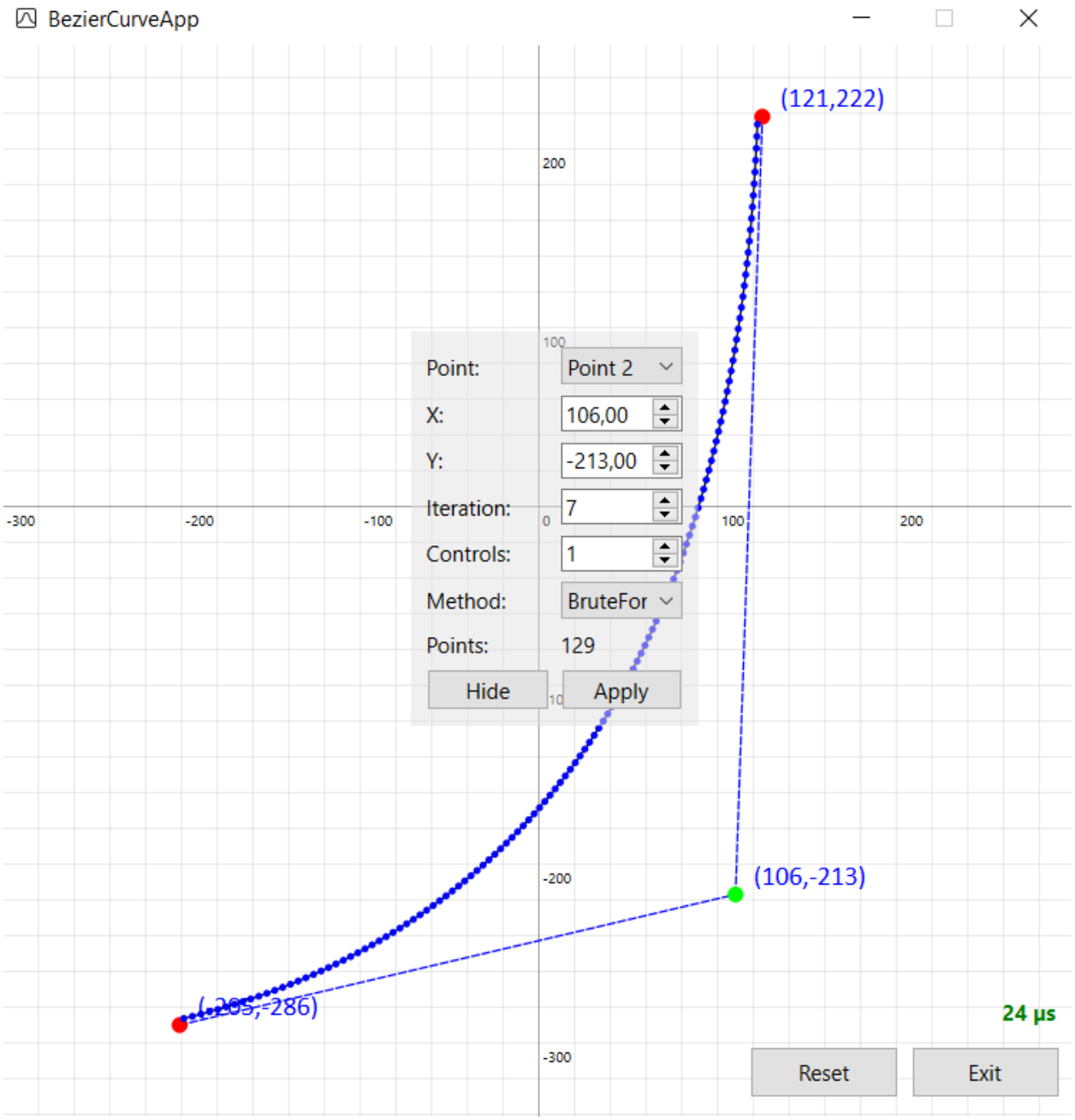


8. Uji Coba 8
Brute Force:

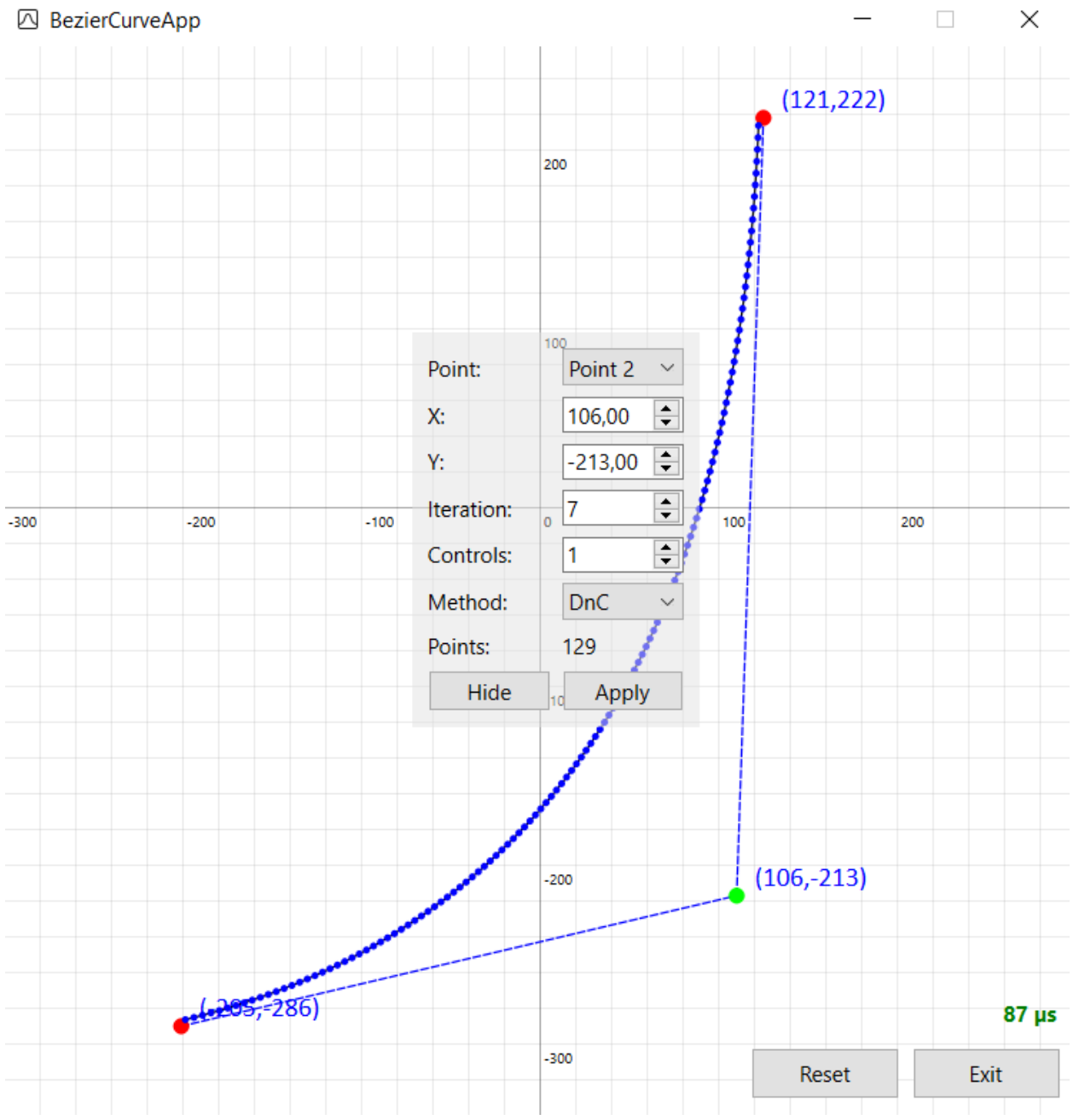


Divide and Conquer:

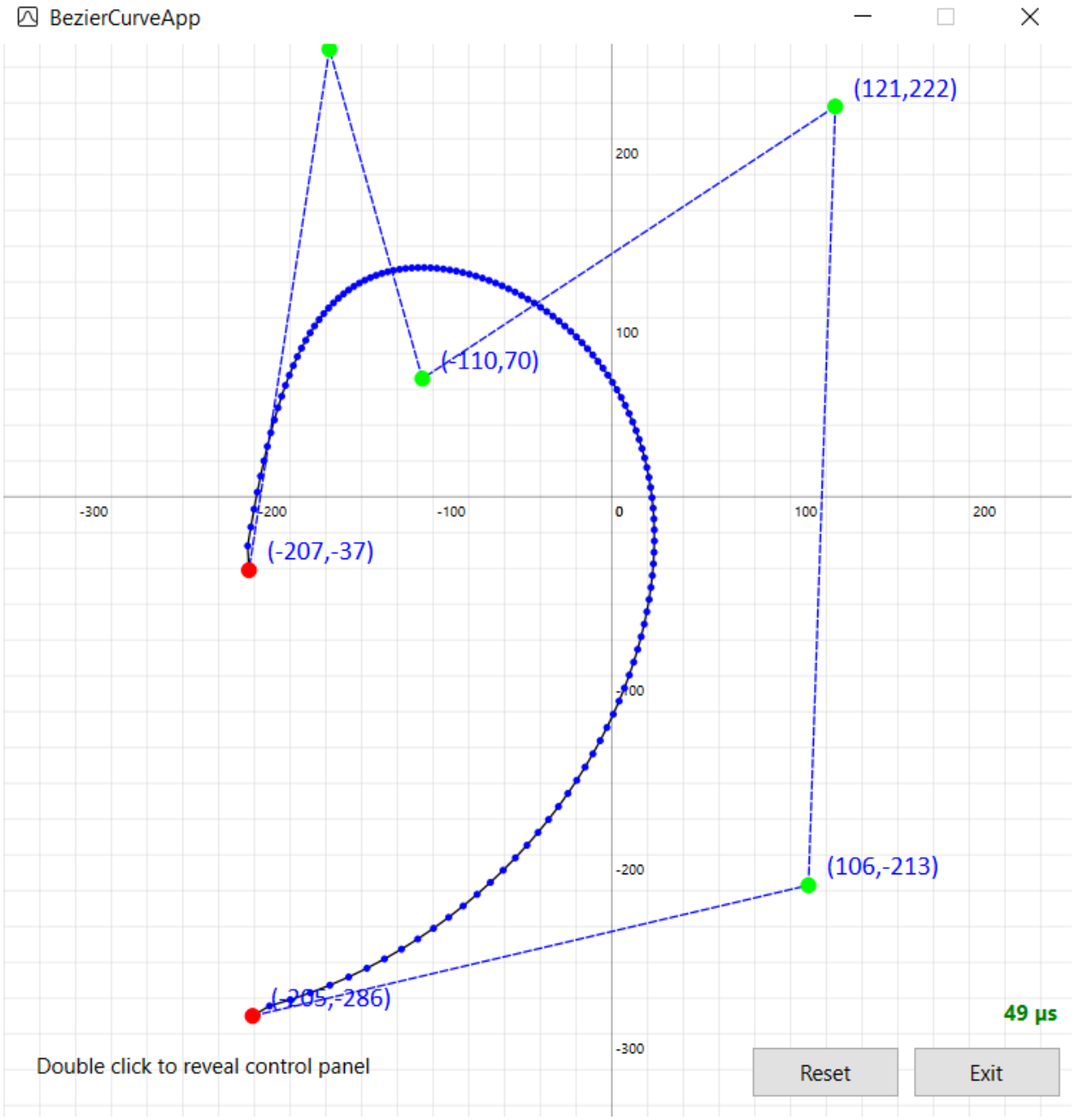
9. Uji Coba 9
Brute Force:



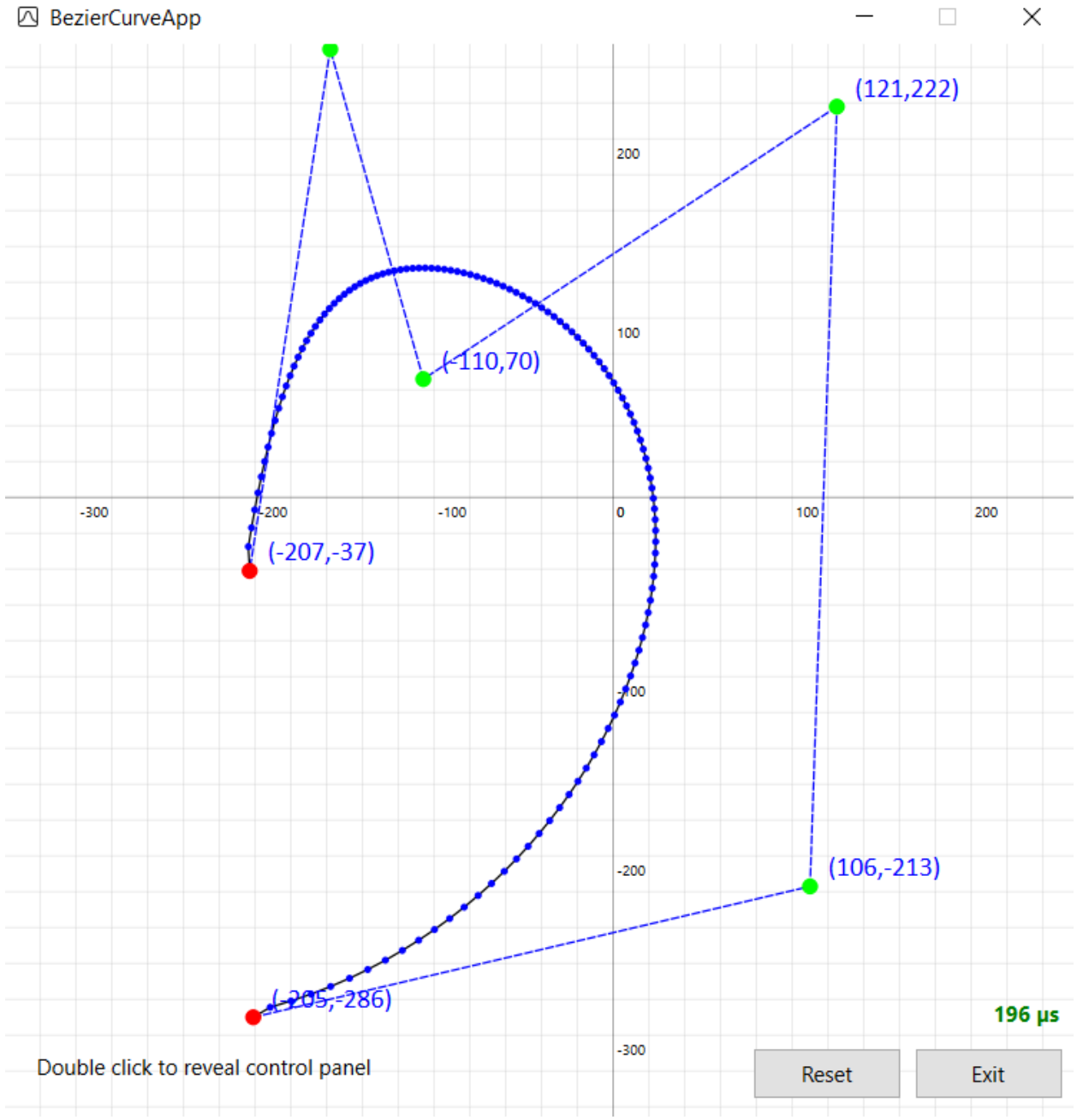
Divide and Conquer:



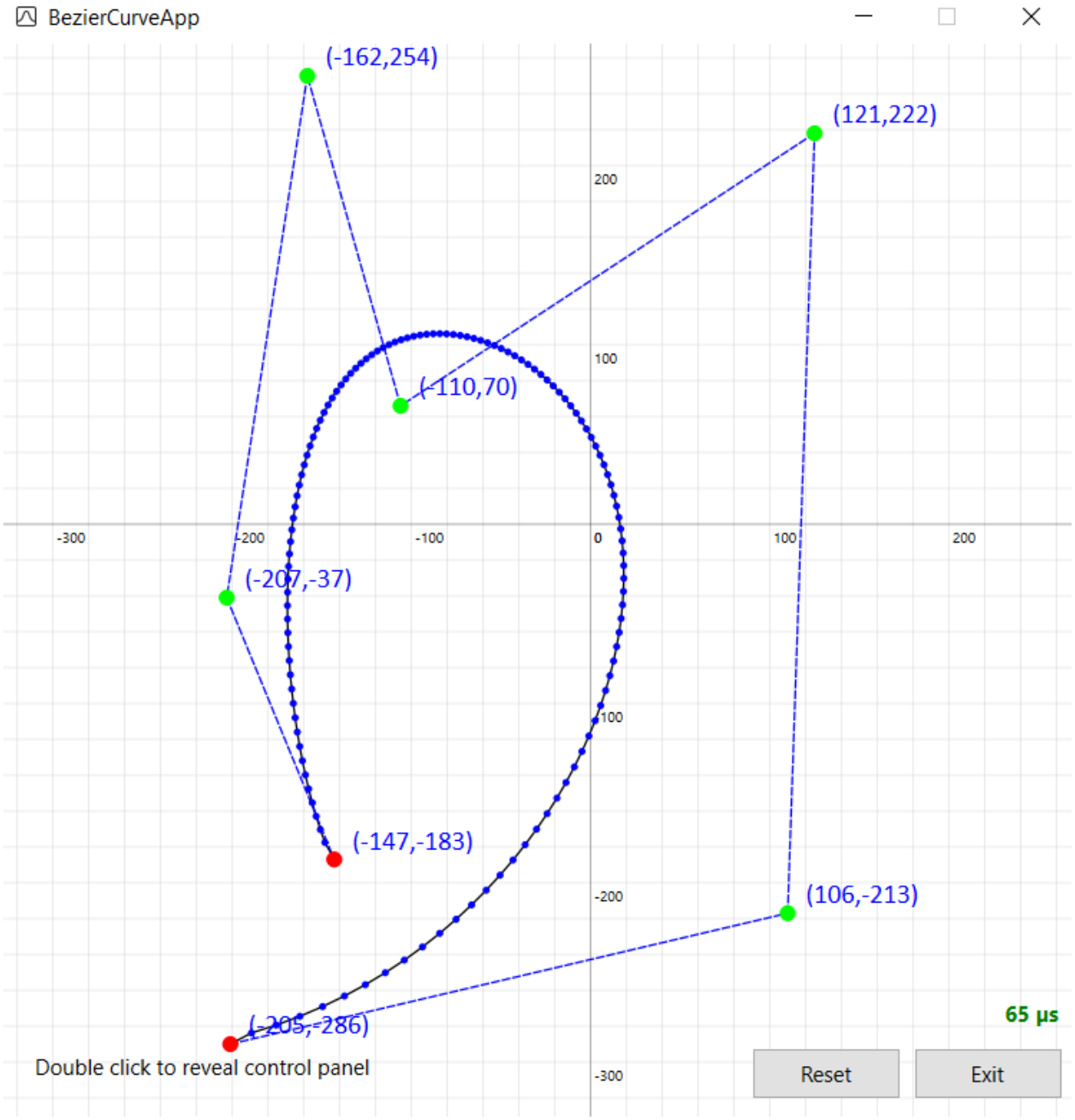
10. Uji Coba 10
Brute Force:



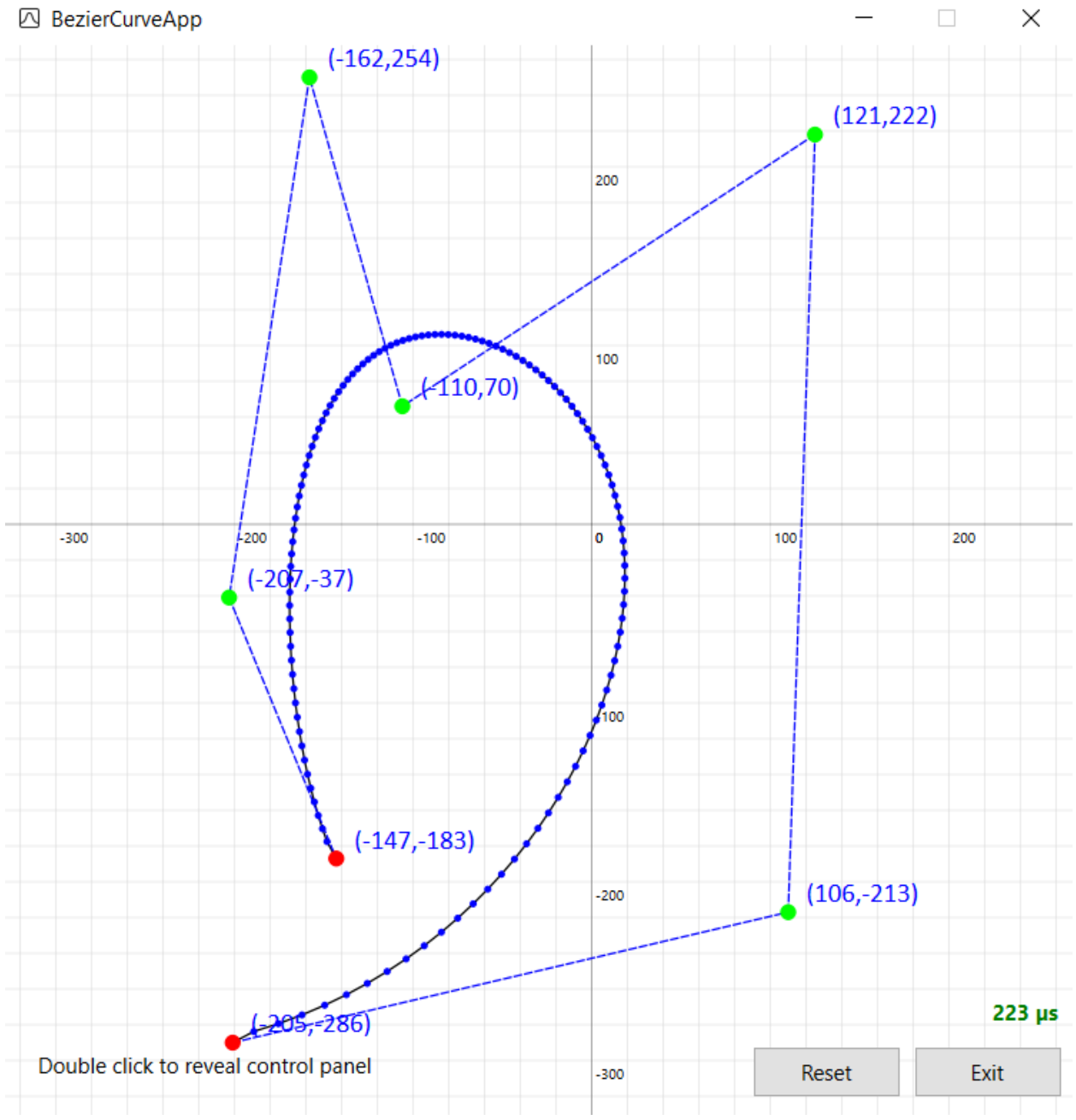
Divide and Conquer:



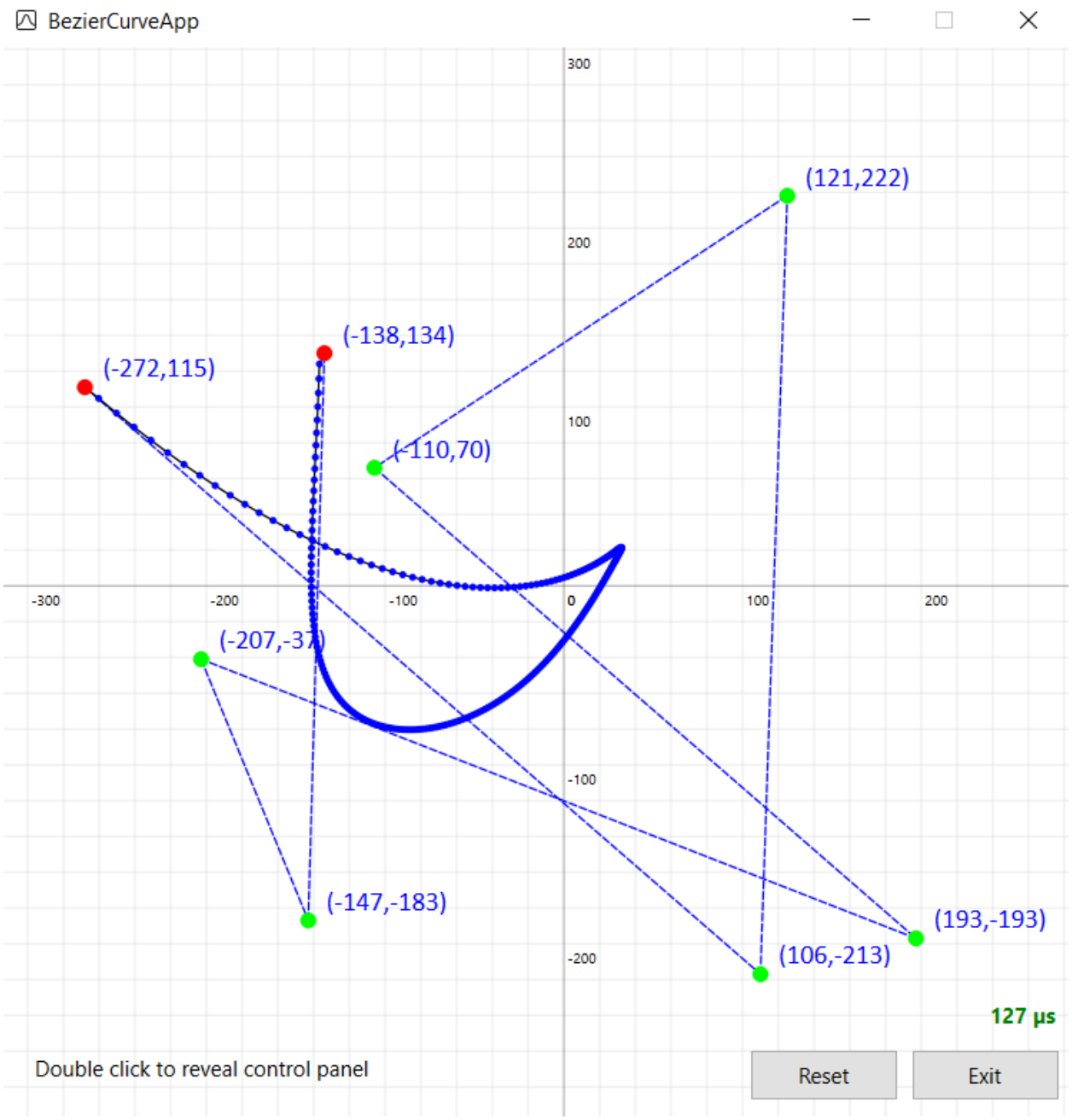
11. Uji Coba 11
Brute Force:



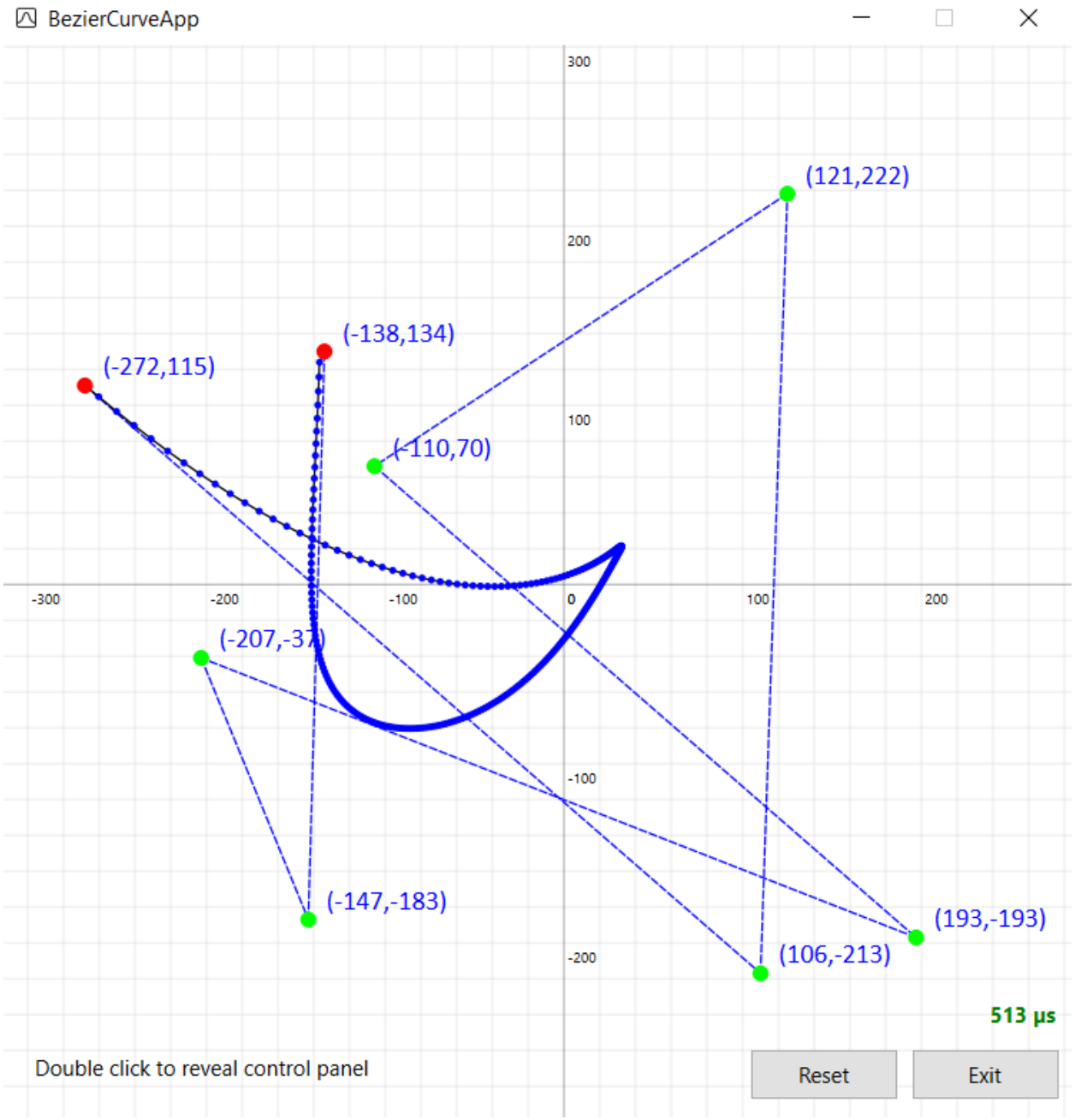
Divide and Conquer:



12. Uji Coba 12
Brute Force:



Divide and Conquer:



5. Analisis Perbandingan Solusi

Berdasarkan hasil uji coba, terlihat bahwa algoritma *brute force* dan algoritma *divide and conquer* dapat memberikan hasil titik yang sama. Namun, perbedaan yang signifikan terletak pada waktu eksekusi kedua algoritma. Terlihat bahwa algoritma *brute force* cukup lebih cepat daripada algoritma *divide and conquer*. Hal ini dikarenakan algoritma *brute force* tidak harus mencari titik-titik tengah seperti pada algoritma *divide and conquer*. Perbedaan kesangkilan tersebut dapat terlihat pada analisis kompleksitas algoritma sebagai berikut:

Catatan: Analisis berdasarkan operasi perkalian dan penjumlahan

Kompleksitas Algoritma Brute Force

```
double factorial(double x){ Total = O(n)
    if (x==0 || x == 1) return 1;
    else return x*factorial(x-1);
}

double combination(double n, double r){ Total = O(1)*O(n) = O(n)
    return factorial(n)/(factorial(r)*factorial(n-r));    O(n)
}

Persamaan::Persamaan(vector <Point> &p): coefficients(p.size()) { Total = O(n)*O(n) = O(n^2)
    double n = p.size();
    for (double i=0;i<n;i++) {                    O(n)
        coefficients[i] = combination(n-1,i);    O(n)
    }

    points = p;
}

Point Persamaan::func(double t) { Total = O(n)*O(1) = O(n)
    double x,y;
    Point result;

    double n = coefficients.size();
    for (double i=0;i<n;i++) {    O(n)
        x += coefficients[i]*pow((1-t),n-1-i)*pow(t,i)*points[i].x;    O(1)
        y += coefficients[i]*pow((1-t),n-1-i)*pow(t,i)*points[i].y;    O(1)
    }

    result.x = x;
    result.y = y;

    return result;
}
```

```
vector <Point> brute_force(int it,vector <Point> &p) {
   int n = p.size();
   double steps = 1/(pow(2,it));    O(1)
   Persamaan f(p);    O(n^2)

   vector<Point> new_p;
   new_p.push_back(p[0]);
   for (int i=1;i<pow(2,it);i++){  O(2^k) dengan k = jumlah iterasi
      new_p.push_back(f.func(i*steps));    O(n)
   }
   new_p.push_back(p.back());

   return new_p;
}
```

Total = $O(n^2) + O(2^k)*O(n) = O(2^k \times n)$

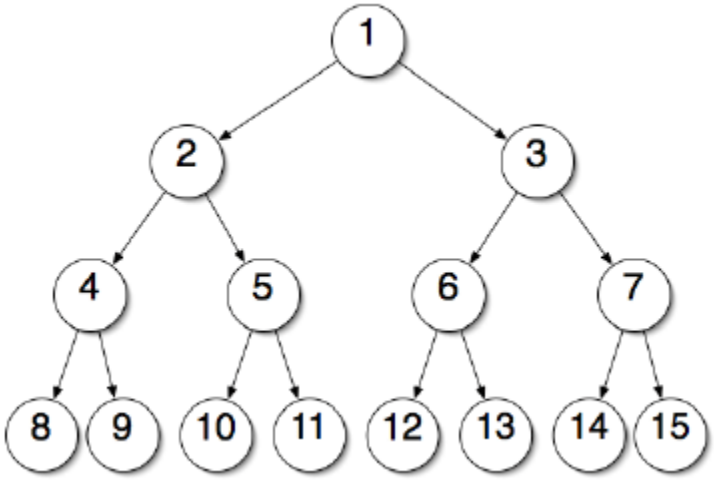
Maka, kompleksitas Algoritma Brute Force = $\mathbf{O(2^k \times n)}$

Kompleksitas Algoritma Divide and Conquer

```
vector <Point> recurse(int it, int mIt, vector <Point> &p) {
   if(it == mIt) return {p[0], p.back()};
   vector<Point> pref, suf;
   while(p.size() >= 2){ O(n), karena panjang list berkurang 1 tiap iterasi
      int n = p.size();
      pref.push_back(p[0]);
      suf.push_back(p.back());
      vector<Point> newP;
      for (int i=0;i<n-1;i++) { O(n)
         // find middle of 2 points
         double x = (p[i+1].x+p[i].x)/2;
         double y = (p[i+1].y+p[i].y)/2;
         Point new_p = {x, y};
         newP.push_back(new_p);
      }
      p.swap(newP);
   } Total = O(n) * O(n) = O(n^2)

   double x = (pref.back().x+suf.back().x)/2;
   double y = (pref.back().y+suf.back().y)/2;
   Point midPoint = {x, y};
   suf.push_back(midPoint);
   reverse(suf.begin(), suf.end());
```

```
    pref.push_back(midPoint);
    vector<Point> lef = recurse(it + 1, mIt, pref); rekursi kiri
    vector<Point> rig = recurse(it + 1, mIt, suf); rekursi kanan
    lef.pop_back();
    for(auto &x: rig) lef.push_back(x); menggabungkan hasil
    return lef;
}
```

Untuk mengetahui kompleksitas algoritma divide and conquer, perlu dilakukan analisis menggunakan teorema Master. Misalkan k adalah banyak iterasi yang diinginkan dan n adalah banyak titik kontrol, mudah dilihat bahwa algoritma ini akan menghasilkan $2^k + 1$ titik, dengan setiap pemanggilan rekursi menghasilkan satu titik. Fungsi *divide and conquer* ini akan memiliki diagram rekursi seperti berikut



Gambar Ilustrasi Rekursi Divide and Conquer

Algoritma ini membutuhkan operasi sebanyak $T(n,k) = T(n,k-1) + O(n^2)$. Karena fungsi akan dipanggil sebanyak $1 + 2 + \ldots + 2^{k-2} + 2^{k-1} = 2^k + 1$ kali, dan setiap pemanggilan rekursi membutuhkan orde $O(n^2)$, maka $O(n,k) = O(2^k * n^2)$

Karena kompleksitas algoritma *divide and conquer* lebih besar dibandingkan kompleksitas *brute force*, waktu eksekusi algoritma *brute force* akan lebih cepat dibandingkan algoritma *divide and conquer*. Hal ini karena pada algoritma *divide and conquer*, program perlu mencari titik tengah sebanyak n * (n + 1) / 2 kali, sedangkan pada algoritma *brute force*, proses evaluasi hanya membutuhkan orde n.

6. Penjelasan Implementasi Bonus

Pada program kami, diterapkan bonus generalisasi algoritma. Untuk algoritma *brute force*, generalisasi dilakukan dengan memanfaatkan binomial expansion untuk membentuk persamaan

dengan derajat sesuai dengan jumlah titik yang diberikan. Untuk algoritma *divide and conquer*, generalisasi dilakukan dengan memperhatikan fakta setiap pemanggilan rekursi akan memiliki jumlah poin yang sama. Langkah-langkah algoritma secara lengkap dapat dilihat pada bagian 2.2.

Pranala Repository

https://github.com/Ariel-HS/Tucil2_13522002_13522024.git

Referensi

Binomial Expansion Formulas - Derivation, Examples. (n.d.). Cuemath. Retrieved March 17, 2024, from https://www.cuemath.com/binomial-expansion-formula/

https://forum.qt.io/topic/2095/interactive-content-fixed-in-on-view-while-it-scrolls-over-scene?page=1. Diakses 17 Maret 2024.

https://forum.qt.io/topic/82834/creating-exit-button-for-a-window. Diakses 15 Maret 2024

https://www.qtcentre.org/threads/44880-QGraphicsScene-Z-Ordering. Diakses 16 Maret 2024

https://stackoverflow.com/questions/57982597/qt-how-to-move-view-with-mousemoveevent. Diakses 15 Maret 2024.

https://www.qtcentre.org/threads/20712-QGraphicsView-per-item-antialiasing-specification. Diakses 15 Maret 2024

https://doc.qt.io/qt-6/qgraphicsscene.html. Diakses 15 Maret 2024

https://doc.qt.io/qt-6/qgraphicsview.html. Diakses 15 Maret 2024

https://doc.qt.io/qt-6/qpointf.html#x. Diakses 15 Maret 2024

https://doc.qt.io/qt-6/qcolor.html. Diakses 16 Maret 2024

https://doc.qt.io/qt-6/qgraphicslineitem.html#QGraphicsLineItem-1. Diakses 15 Maret 2024