Tugas Kecil 2 Strategi Algoritma
Mencari Pasangan Titik Terdekat dengan Algoritma Divide and Conquer

Disusun oleh:
Moch. Sofyan Firdaus (13521083)

Daftar Isi

I. Algoritma

Misalkan n adalah jumlah semua titik di dalam himpunan, berikut ini merupakan algoritma pencarian pasangan titik terdekat dalam bidang 3 dimensi dengan strategi algoritma *Divide and Conquer.*

1. Urutkan semua titik berdasarkan absisnya (x).
2. Jika jumlah titik dalam himpunan adalah dua, maka pasangan titik terdekat adalah kedua titik tersebut.
3. Jika jumlah titik lebih dari dua titik, maka bagi himpunan semua titik menjadi dua himpunan dengan masing-masing himpunan memiliki jumlah titik sebanyak ⌈n/2⌉ buah. Perhatikan bahwa jumlah titik sebanyak ⌈n/2⌉ akan menyebabkan kedua himpunan saling beririsan jika n adalah ganjil.
4. Lakukan rekursi mulai dari langkah kedua untuk masing-masing bagian.
5. Gabungkan kedua himpunan bagian dan bandingkan jarak dari pasangan titik terdekat pada masing-masing bagian, ambil pasangan dengan jarak yang terdekat (d).
6. Bisa jadi pasangan titik terdekat dipisahkan oleh kedua himpunan. Maka dari itu, selanjutnya himpun lah semua titik yang berada dalam jangkauan $x_{n/2} - d$ sampai $x_{n/2} + d$.
7. Lakukan rekursi dengan kembali ke langkah pertama, tetapi dengan mengganti semua x (absis) menjadi y (ordinat).
8. Dengan cara *brute force*, cari pasangan titik terdekat dari himpunan baru tersebut. Perhatikan bahwa untuk setiap titik yang diperiksa, tidak perlu memeriksa jarak terhadap titik dengan selisih koordinat (dalam hal ini koordinat z) > d.

II. Source Code

Algoritma di atas saya implementasikan dengan menggunakan bahasa pemrograman Go. Berikut ini merupakan struktur dari folder src.

```
src
├── algorithm
│   └── algorithm.go
├── banner
├── go.mod
├── go.sum
├── main.go
└── point
    └── point.go
```

- src/point/point.go

```go
package point

import "math"
```

```
type Point struct {
	dim    int
	coord []float64
}

var NumOfCalls uint32 = 0

func EuclideanDistance(a, b Point) float64 {
	if a.dim != b.dim {
		panic("Both points must have same dimension")
	}
	NumOfCalls++
	sum := 0.
	for i := 0; i < a.dim; i++ {
		delta := a.GetCoord()[i] - b.GetCoord()[i]
		sum += delta * delta
	}
	return math.Sqrt(sum)
}

func CreatePoint(coords ...float64) Point {
	return Point{len(coords), coords}
}

func (p Point) GetCoord() []float64 {
	return p.coord
}

func (p Point) GetDimension() int {
	return p.dim
}
```

- src/algorithm/algorithm.go

```
package algorithm

import (
	"math"
	"tucil/stima/pairit/point"
)

var sortKey = 0

func QuickSort[T any](data []T, compareFunc func(T, T) bool) {
	if len(data) <= 1 {
```

```
            return
      }
      // partisi
      pivotIdx := len(data) - 1
      pivot := data[pivotIdx]

      p := -1

      for q := 0; q < pivotIdx; q++ {
            if compareFunc(data[q], pivot) {
                  p++
                  data[p], data[q] = data[q], data[p]
            }
      }

      data[p+1], data[pivotIdx] = data[pivotIdx], data[p+1]

      QuickSort(data[:p+1], compareFunc)
      QuickSort(data[p+1:], compareFunc)
}

func BruteForceFCP(points []point.Point) (*point.Point,
*point.Point, float64) {
      p1 := &points[0]
      p2 := &points[1]
      min := point.EuclideanDistance(*p1, *p2)
      for i := 0; i < len(points); i++ {
            for j := i + 1; j < len(points); j++ {
                  a := &points[i]
                  b := &points[j]
                  d := point.EuclideanDistance(*a, *b)
                  if d < min {
                        min = d
                        p1 = a
                        p2 = b
                  }
            }
      }
      return p1, p2, min
}

func fcpImpl(sortedPoints []point.Point) (*point.Point,
*point.Point, float64) {
      n := len(sortedPoints)

      if n == 2 {
            a := &sortedPoints[0]
            b := &sortedPoints[1]
            return a, b, point.EuclideanDistance(*a, *b)
      }

      mid := int(math.Ceil(float64(n)/2))
```

```
	s1 := sortedPoints[:mid]
	var s2 []point.Point
	if n%2 == 1 {
		s2 = sortedPoints[mid-1:]
	} else {
		s2 = sortedPoints[mid:]
	}

	a1, b1, d1 := fcpImpl(s1)
	a2, b2, d2 := fcpImpl(s2)

	var (
		a, b *point.Point
		d    float64
	)

	if d1 < d2 {
		a, b, d = a1, b1, d1
	} else {
		a, b, d = a2, b2, d2
	}

	i := len(s1) - 1
	for i >= 0 && s1[i].GetCoord()[0] >
sortedPoints[n/2].GetCoord()[0]-d {
		i--
	}

	j := 0
	for j < len(s2) && s2[j].GetCoord()[0] <
sortedPoints[n/2].GetCoord()[0]+d {
		j++
	}

	var s []point.Point
	if n%2 == 1 {
		s = append(s1[i+1:], s2[1:j]...)
	} else {
		s = append(s1[i+1:], s2[:j]...)
	}

	if sortKey < a.GetDimension() - 1 {
		sortKey++
		a, b, d = fcpIntermediete(s)
	}

	for i = 0; i < len(s); i++ {
		for j = i + 1; j < len(s); j++ {
			a3 := &s[i]
			b3 := &s[j]
			m1 := a3.GetCoord()[a3.GetDimension() - 1]
```

```
                  m2 := b3.GetCoord()[b3.GetDimension() - 1]
                  delta := math.Abs(m1 - m2)
                  if delta < d {
                        d3 := point.EuclideanDistance(*a3, *b3)
                        if d3 < d {
                              a, b, d = a3, b3, d3
                        }
                  }
            }
      }
      return a, b, d
}

func FindClosestPairOfPoints(points []point.Point) (*point.Point,
*point.Point, float64) {
      point.NumOfCalls = 0
      a, b, d := fcpIntermediete(points)
      sortKey = 0
      return a, b, d
}

func fcpIntermediete(points []point.Point) (*point.Point,
*point.Point, float64) {
      QuickSort(points, func(a, b point.Point) bool {
            return a.GetCoord()[sortKey] < b.GetCoord()[sortKey]
      })
      return fcpImpl(points)
}
```

- src/main.go

```
package main

import (
      _ "embed"
      "fmt"
      "io"
      "math/rand"
      "os"
      "os/exec"
      "os/signal"
      "syscall"
      "time"
      "tucil/stima/pairit/algorithm"
      . "tucil/stima/pairit/point"

      . "github.com/klauspost/cpuid/v2"
)

type fcpFunction func([]Point) (*Point, *Point, float64)
```

```
var (
	dim, n     int
	upperBound float64
	points     []Point
	p1, p2     *Point
	plotData   *os.File
)

//go:embed banner
var banner string

func main() {
	// handle SIGTERM and SIGINT for deleting temporary file
(plotData)
	sig := make(chan os.Signal, 1)
	signal.Notify(sig, syscall.SIGINT, syscall.SIGTERM)
	go func() {
		<-sig
		deleteTempFile()
		os.Exit(1)
	}()
	defer deleteTempFile()

	fmt.Print(banner)

inputDim:
	fmt.Print("Dimension\t: ")
	_, err := fmt.Scanf("%d\n", &dim)
	if err != nil || dim < 1 {
		fmt.Println("Invalid input!")
		goto inputDim // Is it bad practice? I don't think so
	}

inputN:
	fmt.Print("Number of points: ")
	_, err = fmt.Scanf("%d\n", &n)
	if err != nil || n < 2 {
		fmt.Println("Invalid input!")
		goto inputN
	}

	points = generatePoints(dim, n, float64(n))
	performFcpAlgorithm("Divide and Conquer",
algorithm.FindClosestPairOfPoints)
	performFcpAlgorithm("Brute Force", algorithm.BruteForceFCP)

	path, err := exec.LookPath("gnuplot")
	if dim > 1 && dim <= 3 {
		if err != nil {
			fmt.Println("Gnuplot not found in your PATH.
Visualization is not performed")
```

```
            } else {
                  if dim == 3 {
                        process3D(points)
                        runGnuplot(path)
                  } else {
                        process2D(points)
                        runGnuplot(path)
                  }
            }
      }
}

func performFcpAlgorithm(title string, algo fcpFunction) {
      NumOfCalls = 0
      var d float64
      start := time.Now()
      p1, p2, d = algo(points)
      executionTime := time.Since(start)
      fmt.Println()
      fmt.Printf("\x1b[93m====== %s ======\x1b[0m\n", title)
      fmt.Printf("Point 1\t\t: %v\n", p1.GetCoord())
      fmt.Printf("Point 2\t\t: %v\n", p2.GetCoord())
      fmt.Printf("Distance\t: %f\n", d)
      timeSec := float64(executionTime.Nanoseconds())/1e9
      fmt.Printf("Execution time\t: %.9f s (%s)\n", timeSec,
CPU.BrandName)
      fmt.Printf("The Euclidean distance function is called %dx\n",
NumOfCalls)
}

func runGnuplot(path string) {
      proc := exec.Command(path)
      stdin, _ := proc.StdinPipe()
      if err := proc.Start(); err != nil {
            err_str := fmt.Sprintln("**", err.Error())
            panic(err_str)
      }
      var format string
      if dim == 3 {
            format = "splot '%s' u 1:2:3:4 t '' w p pt 7 ps 1 lc
variable"
      } else {
            format = "plot '%s' u 1:2:3 t '' w p pt 7 ps 1 lc
variable"
      }
      cmd := fmt.Sprintf(format, plotData.Name())
      plotCmd(stdin, "set term qt title 'PairIt'")
      plotCmd(stdin, fmt.Sprintf("set xrange [0:%f]", upperBound))
      plotCmd(stdin, fmt.Sprintf("set yrange [0:%f]", upperBound))
      if dim == 3 {
            plotCmd(stdin, fmt.Sprintf("set zrange [0:%f]",
upperBound))
```

```
	}
	plotCmd(stdin, cmd)
	plotCmd(stdin, "pause mouse close")
	plotCmd(stdin, "q")
	if err := proc.Wait(); err != nil {
		err_str := fmt.Sprintln("**", err.Error())
		panic(err_str)
	}
}

func process3D(points []Point) {
	var err error
	plotData, err = os.CreateTemp(".", "gnuplot-data-")
	if err != nil {
		err_str := fmt.Sprintln("**", err.Error())
		panic(err_str)
	}
	for i := 0; i < len(points); i++ {
		x := points[i].GetCoord()[0]
		y := points[i].GetCoord()[1]
		z := points[i].GetCoord()[2]
		color := 0
		if &points[i] == p1 || &points[i] == p2 {
			color = 7
		}
		line := fmt.Sprintf("%v %v %v %v\n", x, y, z, color)
		if _, err := plotData.WriteString(line); err != nil {
			err_str := fmt.Sprintln("**", err.Error())
			panic(err_str)
		}
	}
	plotData.Close()
}

func process2D(points []Point) {
	var err error
	plotData, err = os.CreateTemp(".", "gnuplot-data-")
	if err != nil {
		err_str := fmt.Sprintln("**", err.Error())
		panic(err_str)
	}
	for i := 0; i < len(points); i++ {
		x := points[i].GetCoord()[0]
		y := points[i].GetCoord()[1]
		color := 0
		if &points[i] == p1 || &points[i] == p2 {
			color = 7
		}
		line := fmt.Sprintf("%v %v %v\n", x, y, color)
		if _, err := plotData.WriteString(line); err != nil {
			err_str := fmt.Sprintln("**", err.Error())
			panic(err_str)
```

```
            }
      }
      plotData.Close()
}

func plotCmd(stdin io.Writer, command string) {
      if _, err := io.WriteString(stdin, command+"\n"); err != nil
{
            fmt.Println("**", err.Error())
      }
}

func generatePoints(dim, n int, ub float64) []Point {
      rand.Seed(time.Now().UnixNano())
      upperBound = ub
      points := make([]Point, n)
      for i := 0; i < n; i++ {
            coord := make([]float64, dim)
            for j := 0; j < dim; j++ {
                  coord[j] = rand.Float64() * upperBound
            }
            points[i] = CreatePoint(coord...)
      }
      return points
}

func deleteTempFile() {
      if plotData != nil {
            os.Remove(plotData.Name())
      }
}
```

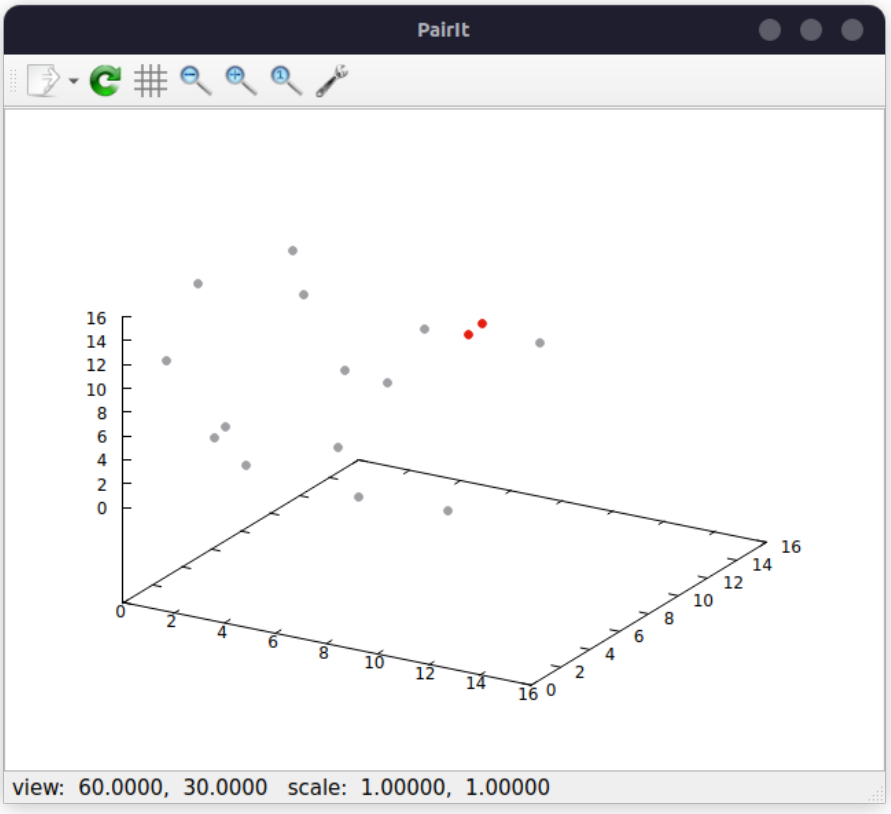
III. Contoh Input/Output

- n = 16

```
└─ bin/pairit
'########::::'###::::'####:'########::'####:'########::::
 ##.... ##:::'## ##:::. ##:: ##.... ##:. ##:::... ##..:::::
 ##:::: ##::'##:. ##::: ##:: ##:::: ##:: ##::::: ##::::::::
 ########::'##:::. ##:: ##:: ########::: ##::::: ##::::::::
 ##.....::: #########:: ##:: ##.. ##:::: ##::::: ##::::::::
 ##:::::::: ##.... ##:: ##:: ##::. ##::: ##::::: ##::::::::
 ##:::::::: ##:::: ##:'####: ##:::. ##:'####:::: ##::::::::
..:::::::::..:::::..::....::..:::::..::....:::::..:::::::::
Dimension       : 3
Number of points: 16

====== Divide and Conquer ======
Point 1         : [8.627099616837572 8.584810649252036 11.810708471347]
Point 2         : [9.06317707008489 8.75975064262682 12.844263361407572]
Distance        : 1.135343
Execution time  : 0.000015000 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 32x

====== Brute Force ======
Point 1         : [8.627099616837572 8.584810649252036 11.810708471347]
Point 2         : [9.06317707008489 8.75975064262682 12.844263361407572]
Distance        : 1.135343
Execution time  : 0.000007752 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 121x
```

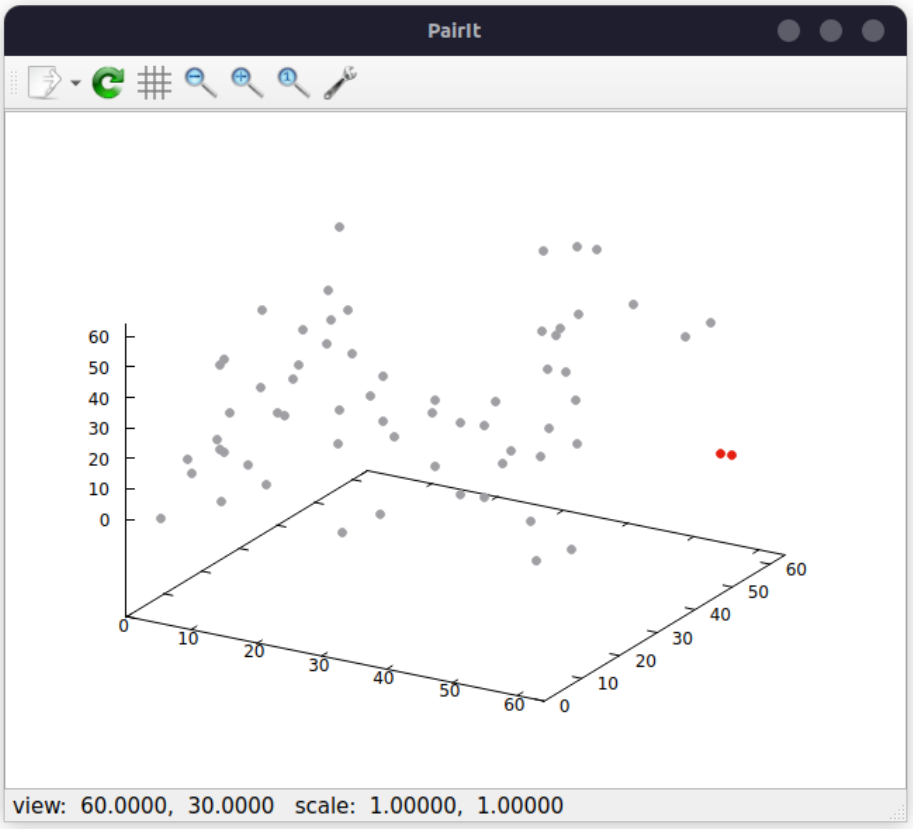


- n = 64

```
└─ bin/pairit
'########::::'###::::'####:'########::'####:'########::::
 ##.... ##::::'## ##:::. ##:: ##.... ##:. ##:::... ##..:::::
 ##:::: ##::'##:. ##::: ##:: ##:::: ##:: ##::::: ##:::::::
 ########::'##:::. ##:: ##:: ########::: ##::::: ##:::::::
 ##.....::: #########:: ##:: ##.. ##:::: ##::::: ##:::::::
 ##:::::::: ##.... ##:: ##:: ##::. ##::: ##::::: ##:::::::
 ##:::::::: ##:::: ##:'####: ##:::. ##:'####:::: ##:::::::
..:::::::::..:::::..::....::..:::::..::....:::::..::::::::
Dimension       : 3
Number of points: 64

====== Divide and Conquer ======
Point 1         : [62.958646800915204 48.91258341757296 12.057363284425714]
Point 2         : [63.47847158230521 50.79713061342768 10.271366490043468]
Distance        : 2.647928
Execution time  : 0.000114066 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 183x

====== Brute Force ======
Point 1         : [62.958646800915204 48.91258341757296 12.057363284425714]
Point 2         : [63.47847158230521 50.79713061342768 10.271366490043468]
Distance        : 2.647928
Execution time  : 0.000090890 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 2017x
```

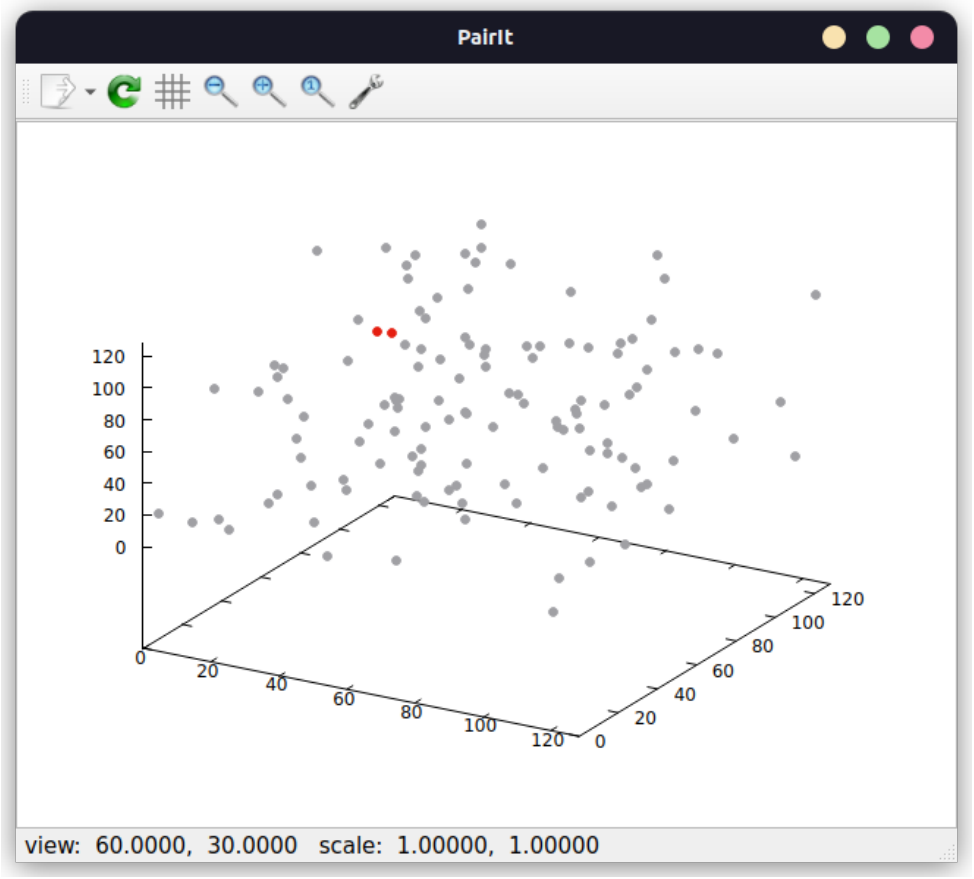


- n = 128

```
└─ bin/pairit
'########::::'###::::'####:'########::'####:'########::::
 ##.... ##::::'## ##:::. ##:: ##.... ##:. ##::... ##..:::::
 ##:::: ##::'##:. ##::: ##:: ##:::: ##:: ##::::: ##:::::::
 ########::'##:::. ##:: ##:: ########::: ##::::: ##:::::::
 ##.....::: #########:: ##:: ##.. ##::::: ##::::: ##:::::::
 ##:::::::: ##.... ##:: ##:: ##::. ##::: ##::::: ##:::::::
 ##:::::::: ##:::: ##:'####: ##:::. ##:'####:::: ##:::::::
..:::::::::..:::::..::....::..:::::..::....:::::..::::::::
Dimension       : 3
Number of points: 128

====== Divide and Conquer ======
Point 1         : [9.230556106319066 102.9089872877552 62.30821730489828]
Point 2         : [12.844487821514347 104.30705296182728 62.018414694129795]
Distance        : 3.885753
Execution time  : 0.000134429 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 471x

====== Brute Force ======
Point 1         : [9.230556106319066 102.9089872877552 62.30821730489828]
Point 2         : [12.844487821514347 104.30705296182728 62.018414694129795]
Distance        : 3.885753
Execution time  : 0.000248724 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 8129x
```

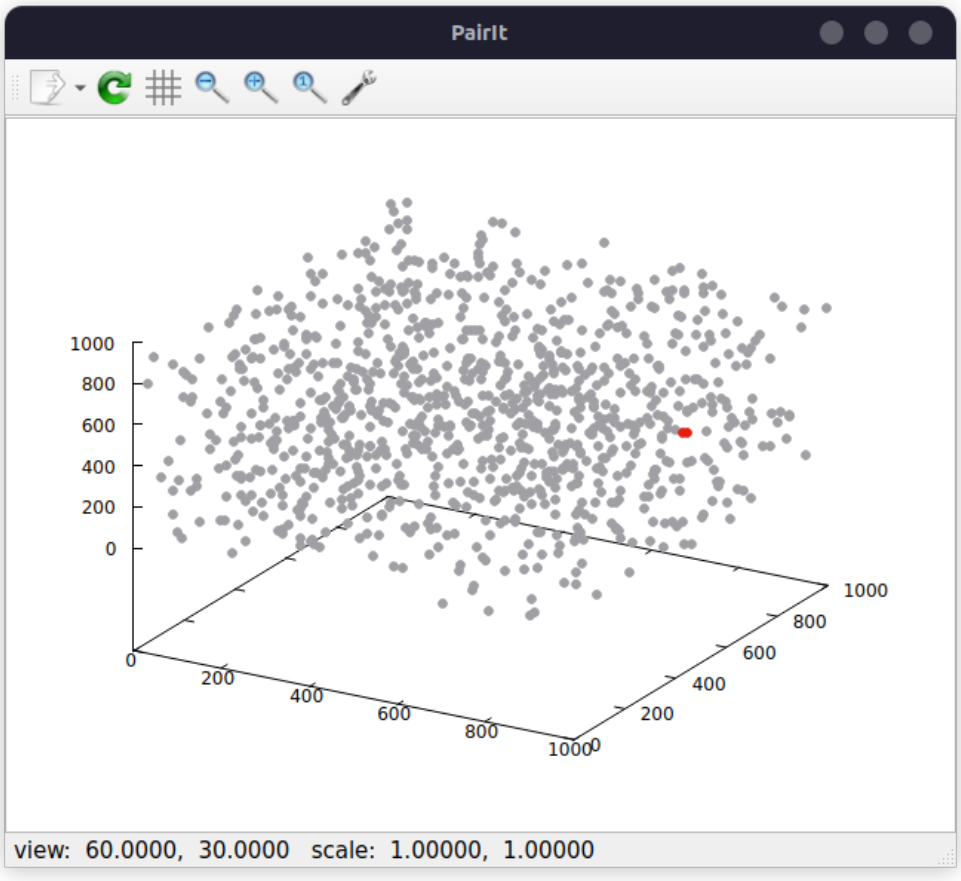


- n = 1000

```
└─ bin/pairit
'########::::'###::::'####:'########::'####:'########::::
 ##.... ##:::'## ##::::. ##:: ##.... ##:. ##:::... ##..:::::
 ##:::: ##::'##:. ##::::: ##:: ##:::: ##:: ##::::: ##:::::::
 ########::'##:::. ##:::: ##:: ########::: ##::::: ##:::::::
 ##.....::: #########:: ##:: ##.. ##:::: ##::::: ##:::::::
 ##:::::::: ##.... ##:: ##:: ##::. ##::: ##::::: ##:::::::
 ##:::::::: ##:::: ##:'####: ##:::. ##:'####:::: ##:::::::
..:::::::::..:::::..::....::..:::::..::....:::::..::::::::
Dimension       : 3
Number of points: 1000

====== Divide and Conquer ======
Point 1         : [736.4463953692687 889.6342678547612 209.79759893927897]
Point 2         : [741.2992005042274 895.6173535673297 210.06617424025166]
Distance        : 7.708383
Execution time  : 0.002136940 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 4592x

====== Brute Force ======
Point 1         : [736.4463953692687 889.6342678547612 209.79759893927897]
Point 2         : [741.2992005042274 895.6173535673297 210.06617424025166]
Distance        : 7.708383
Execution time  : 0.005367947 s (Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz)
The Euclidean distance function is called 499501x
```



IV. Lampiran

Link github: https://github.com/msfir/Tucil2_13521083

Checklist program:

Poin	**Ya**	**Tidak**
1. Program berhasil dikompilasi tanpa ada kesalahan.	✔️	
2. Program berhasil *running*	✔️	
3. Program dapat menerima masukan dan menuliskan luaran.	✔️	
4. Luaran program sudah benar (solusi *closest pair*)	✔️	
5. Bonus 1 dikerjakan	✔️	
6. Bonus 2 dikerjakan	✔️	