Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II 2022-2023

PENYELESAIAN PERMASALAHAN CLOSEST PAIR OF POINTS MENGGUNAKAN DIVIDE AND CONQUER

Disusun oleh :

Addin Munawwar Yusuf	13521085
Fakih Anugerah Pratama	13521091

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

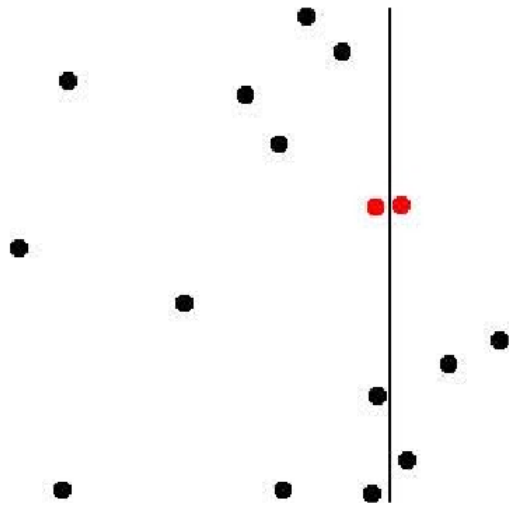
2023

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
IF2211 – Semester II 2022/2023

1. Deskripsi Persoalan

Closest Pair Problem adalah salah satu persoalan di bidang komputasional geometri dan strategi algoritma. Persoalan ini meminta untuk menemukan sepasang titik terdekat di dalam sebuah bidang (2D), atau pada dimensi yang lebih tinggi.

Secara formal, persoalan ini dirumuskan sebagai berikut: diberikan sebuah set n titik dalam ruang dua dimensi atau dimensi yang lebih tinggi. Carilah pasangan titik yang memiliki jarak terpendek satu sama lain. Masalah ini memiliki banyak aplikasi praktis di bidang seperti grafika komputer, pengenalan pola, dan sistem informasi geografis.



Gambar 1. Ilustrasi Persoalan *Closest Pair*

[Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/]

Solusi kasar (brute force) untuk masalah ini akan memerlukan pemeriksaan setiap pasangan titik dalam set. Untuk setiap titik, akan dihitung jarak Akan tetapi, terdapat algoritma yang lebih efisien yang dapat menyelesaikan masalah ini dalam waktu O(n log n) atau bahkan lebih cepat.

Algoritma-algoritma ini biasanya menggunakan teknik divide and conquer, dan seringkali bergantung pada konsep diagram Voronoi, triangulasi Delaunay, atau struktur data geometris lainnya. Pada laporan Tugas Kecil 2 ini sendiri, strategi algoritma yang akan digunakan adalah *divide and conquer*.

2. Spesifikasi Tugas

Mencari pasangan titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/ Ruby/Perl (pilih salah satu) untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

Masukan program:

- n
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program:

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)

- Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
- Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di Rn, setiap vektor dinyatakan dalam bentuk $\mathbf{x}$ = (x1, x2,..., xn)

3. Teori Singkat

3.1. Divide and Conquer

Algoritma divide and conquer adalah salah satu pendekatan algoritmik untuk menyelesaikan sebuah permasalahan dengan membaginya menjadi beberapa upa-permasalahan secara rekursif.

3.2. Quicksort

Quicksort adalah salah satu pemanfaatan algoritma divide and conquer yang berupa algoritma pengurutan yang terbukti cepat, mengalahkan algoritma pengurutan yang lain. Quicksort bekerja dengan melakukan partisi pada barisan yang akan diurutkan menjadi upa-barisan secara rekursif lalu menggabungkannya kembali menjadi sebuah barisan nilai yang terurut.

3.3. Closest Pair Problem

Closest Pair Problem merupakan sebuah permasalahan terkenal yang digunakan untuk mendemonstrasikan bagaimana *divide and conquer* dimanfaatkan dalam melakukan penyelesaian terhadap suatu masalah. Ketika diselesaikan menggunakan dua algoritma berbeda, yaitu *bruteforce* dan *divide and conquer*, permasalahan ini akan menunjukkan bagaimana pengaruh pemanfaatan *divide and conquer* mempengaruhi kompleksitas waktu dan ruang penyelesaian permasalahan ini secara signifikan.

4. Implementasi Teori

4.1. Flow Program

4.1.1. Program meminta masukan beberapa nilai penting kepada pengguna

4.1.2. Program akan melakukan inisialisasi awal berupa proses generasi titik-titik yang akan digunakan dalam penyelesaian permasalahan berdasarkan masukan pengguna sebelumnya

4.1.3. Program melakukan pengurutan awal terhadap titik-titik yang telah tergenerasi melalui proses sebelumnya (proses ini sangat krusial untuk menyelesaikan permasalahan menggunakan algoritma *divide and conquer*)

4.1.4. Program melakukan pemecahan masalah menggunakan algoritma *bruteforce* dan menampilkan hasilnya

4.1.5. Program melakukan pemecahan masalah menggunakan algoritma *divide and conquer* dan menampilkan hasilnya

4.1.6. Program akan menampilkan visualisasi semua titik-titik beserta dua titik yang menjadi hasil pemecahan masalah dalam ruang tiga dimensi (hanya jika pengguna memasukkan nilai dimensi masalah sebagai 3 (tiga))

4.2. Algoritma

4.2.1. Quicksort

1. Nilai terakhir dipilih sebagai *pivot*
2. Periksa nilai-nilai dalam barisan dan buat *pivot* berada pada tempat yang tepat, yaitu di setelah semua nilai yang lebih kecil darinya dan sebelum semua nilai yang lebih besar darinya
3. Lakukan secara rekursif untuk masing-masing baris nilai di sebelum dan setelah pivot

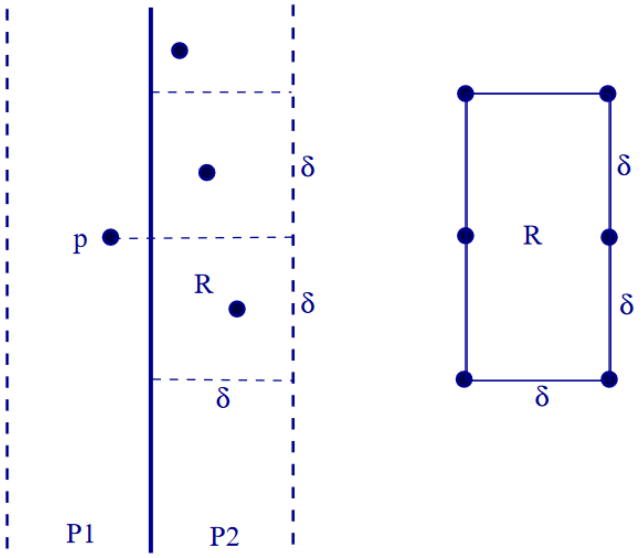
4.2.2. Bruteforce Closest Pair

1. 2 titik paling awal dalam barisan titik ditetapkan sebagai nilai hasil (inisialisasi)
2. Untuk setiap titik dalam barisan akan dicari nilai jaraknya dengan titik lain dalam barisan tepat satu kali

3. Jika jarak yang dihasilkan lebih kecil dari nilai hasil, tetapkan 2 titik tersebut sebagai hasil dan jaraknya sebagai jarak terdekat

4.2.3. Divide and Conquer Closest Pair Dimensi Dua

1. Urutkan titik-titik pada himpunan S berdasarkan absisnya (sumbu-x).
2. Bagi himpunan titik S menjadi dua bagian yang sama besar (S1 dan S2). Pemisahan dilakukan berdasarkan garis median dari absis / sumbu-x (*selanjutnya akan dikenal dengan istilah strip*).
3. Hitung jarak titik terdekat secara rekursif untuk S_1 dan S_2 sehingga didapatkan jarak minimum dari S1 (δ1) dan S2(δ2). *Basis dari rekursif adalah ketika jumlah titik ≤ 3. Pada kondisi tersebut, closest pair dihitung secara brute-force.*
4. Cari nilai jarak terdekat antar hasil dari himpunan S1 dan himpunan S2 (δ = min(δ1,δ2)). Simpan pasangan titik dan jarak tersebut.
5. Cari titik-titik yang berada sejauh δ dari strip. Titik-titik ini akan menjadi kandidat-kandidat untuk titik terpendek.
6. Untuk setiap titik di sebelah kiri strip, pilihlah kandidat-kandidat titik di sebelah kanan yang berkemungkinan menjadi pasangan terdekat. Titik-titik tersebut pasti berada pada persegi panjang R yang berukuran 2δ x δ. (Gambar 4.2.3)



Gambar 4.2.3. Pemilihan kandidat pasangan terpendek di sekitar strip
[Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/]

7. Untuk setiap titik kandidat yang terpilih, lakukan perhitungan jarak euclidean. Jika jarak yang dihitung lebih pendek, perbarui δ dan simpan pasangan titik tersebut.
8. Kembalikan pasangan titik terdekat yang ditemukan di akhir.

Kompleksitas algoritmanya adalah T(n) = 2T(n/2) + cn untuk n > 2, dengan teorema master, didapatkan kompleksitasnya adalah O(n log(n)).

4.2.4. Divide and Conquer Closest Pair Dimensi K (K ≥ 3)

1. Urutkan titik berdasarkan sumbu utamanya (dalam hal ini, penulis memilih sumbu-x).
2. Bagi himpunan titik S menjadi dua bagian yang sama besar (S1 dan S2). *Pemisahan dilakukan berdasarkan garis median dari absis / sumbu-x (selanjutnya akan dikenal dengan istilah strip).*

Dalam kasus dimensi ≥ 3, strip dapat berbentuk bidang ataupun dimensi yang lebih tinggi.

3. Hitung jarak titik terdekat secara rekursif untuk S1 dan S2 sehingga didapatkan jarak minimum dari S1 (δ1) dan S2(δ2). Basis dari rekursif adalah ketika jumlah titik ≤ 3. Pada kondisi tersebut, closest pair dihitung secara brute-force, *masih sama seperti 2D*.
4. Cari nilai jarak terdekat antar hasil dari himpunan S1 dan himpunan S2 (δ = min(δ1,δ2)). Simpan pasangan titik dan jarak tersebut.
5. Cari titik-titik yang berada sejauh δ dari strip. Titik-titik ini akan menjadi kandidat-kandidat untuk titik terpendek. Pisah menjadi bagian sebelah kiri dan kanan.
6. Selanjutnya, untuk setiap titik di bagian kiri, pilih kandidat titik-titik di sebelah kanan yang berkemungkinan menjadi pasangan terdekat. Kandidat-kandidat ini dapat dipilih dengan cara melakukan proyeksi titik ke dimensi yang lebih rendah. Proyeksi dilakukan dengan melakukan pemampatan pada sumbu utamanya (jika sebelumnya yang dipilih adalah sumbu-x, maka mampatkan pada sumbu-x). Lakukan secara rekursif hingga dimensi satu. Pada setiap pemilihan kandidat, lakukan perhitungan euclidean untuk melihat apakah jarak titik tersebut ≤ δ.
7. Setelah kandidat terpilih, lakukan perhitungan jarak euclidean. Jika jarak yang dihitung lebih pendek, perbarui δ dan simpan pasangan titik tersebut.
8. Kembalikan pasangan titik terdekat yang ditemukan di akhir.

Kompleksitas algoritmanya adalah $T(n) = 2T(n/2, d) + T(n, d-1) + cn$. Jika persamaan tersebut diselesaikan, didapatkan kompleksitas dari algoritma ini adalah $O(n(\log n)^{d-1})$

4.3. Struktur Program

Program ini dibuat menggunakan bahasa pemrograman *golang* dengan paradigma pemrograman post-object-oriented. Untuk memperjelas modularitas, program dibagi menjadi beberapa file .go yang memiliki tanggung jawab masing-masing. Seluruh file yang berkaitan dengan *flow code* program diletakkan dalam folder *src/*

4.3.1. File main.go

File ini bertanggungjawab sebagai entry point sekaligus pembawa flow program.

4.3.2. File Point.go

File ini berisikan definisi post-object **point** serta method-method yang dimilikinya. **Point** bertugas untuk mensimulasikan sebuah titik dari titik-titik yang akan dicari pasangan dengan jarak terdekatnya.

4.3.3. File Solver.go

File ini berisikan definisi post-object **solver** serta method-method lainnya. **Solver** bertugas untuk menjadi “otak” dalam pencarian pasangan titik terdekat.

4.3.4. File Visualizer.go

File ini berisikan wrapper method yang memanfaatkan package ***go-echarts*** untuk melakukan visualisasi titik-titik dalam persoalan pencarian pasangan titik yang memiliki dimensi tiga.

4.3.5. File ExecTimer.go

File ini berisikan definisi post-object **ExecTimer** serta method-method yang dimilikinya. **ExecTimer** bertugas untuk mencatat waktu mulai dan berakhirnya eksekusi suatu proses, serta menampilkannya kepada pengguna.

4.3.6. IOHandler.go

File ini berisikan method-method yang digunakan untuk melakukan handling terhadap masukan pengguna dan keluaran.

4.4. Kode Program

4.4.1. main.go

```
package main

import (
    "github.com/shirou/gopsutil/v3/cpu"
    "github.com/shirou/gopsutil/v3/mem"
    "github.com/shirou/gopsutil/v3/host"
    "github.com/shirou/gopsutil/v3/disk"
    "strconv"
)

func main() {
    sortTimer := NewExecTimer("QuickSort")
    bruteforceTimer := NewExecTimer("Bruteforce")
    divideAndConquerTimer := NewExecTimer("Divide and Conquer")

    // get user inputs
    IOHandler := NewIOHandler()
    IOHandler.PrintLine("\nUSER
CONFIGURATION\n-----------------------")
    n, d, lb, ub := IOHandler.readUserConfig()

    // construct solver
    solver := NewSolver()
    solver.GeneratePoints(n, d, lb, ub)

    // tell specifications
    IOHandler.PrintLine("COMPUTER
SPECIFICATIONS\n-----------------------")
    val, _ := cpu.Info()
    IOHandler.PrintLine("CPU: " + val[0].ModelName)
    // get host
    host, _ := host.Info()
    IOHandler.PrintLine("Host: " + host.Hostname)
    // get memory
    v, _ := mem.VirtualMemory()
    IOHandler.PrintLine("Memory: " +
strconv.FormatUint(v.Total/1024/1024, 10) + " MB")
    // get disk
    disk, _ := disk.Usage("/")
```

```
    IOHandler.PrintLine("Disk: " +
strconv.FormatUint(disk.Total/1024/1024, 10) + " MB\n")

    // sort points by x-axis
    sortTimer.Start()

    solver.Sort(0)

    sortTimer.Finish()
    sortTimer.Tell()

    // with bruteforce
    IOHandler.PrintLine("\nWITH
BRUTEFORCE\n-----------------------")
    bruteforceTimer.Start()

    solver.SolveByForce()

    solver.Describe()

    bruteforceTimer.Finish()
    bruteforceTimer.Tell()

    // with divide and conquer
    IOHandler.PrintLine("\nWITH DIVIDE AND
CONQUER\n-----------------------")
    divideAndConquerTimer.Start()

    solver.Solve()

    solver.Describe()

    divideAndConquerTimer.Finish()
    divideAndConquerTimer.Tell()

    // visualize if 3 dimensions
    if d == 3 {
        if IOHandler.askToVisualize() {
            visualizer := NewVisualizer(solver.points,
solver.solutionPoints)
            visualizer.visualize()
        }
    }

    if d != 3 {
        IOHandler.PrintLine("\nUse 3 dimension input for full
experience with the visualizer:D")
    }
```

```
    IOHandler.PrintLine("\nThank you for using this program.")
}
```

4.4.2. Point.go

```
package main

import (
    "math/rand"
)

var (
    num_of_points int = 0
)

type Point struct {
    ID        int
    dimension int
    val       []float64
}

// Constructor
func NewPoint(val ...float64) *Point {
    p := new(Point)
    p.dimension = len(val)
    p.val = val

    num_of_points++
    p.ID = num_of_points

    return p
}

func NewRandomPoint(dimension int, lowerBound float64, upperBound
float64) *Point {
    p := new(Point)
    p.dimension = dimension

    p.val = make([]float64, dimension)

    for i, _ := range p.val {
        p.val[i] = rand.Float64()*(upperBound-lowerBound) +
lowerBound // [lowerBound, upperBound]
    }

    num_of_points++
    p.ID = num_of_points
```

```
    return p
}

func (p Point) GetAxisValue(axis int) float64 {
    return p.val[axis]
}

func (p1 Point) Equals(p2 Point) bool {
    return p1.dimension == p2.dimension && p1.ID == p2.ID
}
```

4.4.3. Solver.go

```
package main

import (
    "fmt"
    "math"
)

var euclidOpsCount int

type Solver struct {
    points    []Point
    dimension int

    isInitialized bool

    solutionFound  bool
    solutionPoints [2]Point
    solutionDist   float64
}

func NewSolver(points ...Point) *Solver {
    s := new(Solver)

    if len(points) == 0 {
        s.dimension = 0
        s.isInitialized = false
    } else {
        s.dimension = points[0].dimension
        s.isInitialized = true
    }

    s.points = points
    s.solutionFound = false
```

```
    return s
}

func (s *Solver) GeneratePoints(nPoints, nDimension int,
lowerBound, upperBound float64) {
    var tempPoints []Point

    for i := 0; i < nPoints; i++ {
        tempPoints = append(tempPoints, *NewRandomPoint(nDimension,
lowerBound, upperBound))
    }

    s.dimension = tempPoints[0].dimension
    s.isInitialized = true

    s.points = tempPoints
    s.solutionFound = false
}

// sort utility
// we use quicksort
func getPartition(slice []Point, a, b, axis int) int {
    curPiv := slice[b].GetAxisValue(axis)

    i := a - 1

    for j := a; j < b; j++ {
        if slice[j].GetAxisValue(axis) <= curPiv {
            i = i + 1

            slice[i], slice[j] = slice[j], slice[i]
        }
    }

    slice[i+1], slice[b] = slice[b], slice[i+1]

    return i + 1
}

func (s *Solver) quickSort(a, b, axis int) {
    if a < b {
        p := getPartition(s.points, a, b, axis)

        s.quickSort(a, p-1, axis)
        s.quickSort(p+1, b, axis)
    }
}
```

```
func (s *Solver) Sort(axis int) {
    s.quickSort(0, len(s.points)-1, axis)
}

// Print prints points saved on solver
func (s *Solver) Print() {
    for _, point := range s.points {
        for _, val := range point.val {
            fmt.Print(val, " ")
        }

        fmt.Println()
    }
}

func getEuclideanDistance(p1, p2 Point) float64 {
    // assuming both points have the same order of dimension

    var d float64

    euclidOpsCount++

    for i := 0; i < p1.dimension; i++ {
        d +=
math.Pow(float64(p2.GetAxisValue(i))-float64(p1.GetAxisValue(i)),
2)
    }

    return math.Sqrt(d)
}
func getEuclideanDistanceAxis(p1, p2 Point, axis int) float64 {
    // assuming both points have the same order of dimension

    var d float64

    euclidOpsCount++

    for i := axis; i < p1.dimension; i++ {
        d +=
math.Pow(float64(p2.GetAxisValue(i))-float64(p1.GetAxisValue(i)),
2)
    }

    return math.Sqrt(d)
}

func getClosestByForce(points ...Point) (Point, Point, float64) {
    delta := getEuclideanDistance(points[0], points[1])
```

```
    idA, idB := 0, 1

    if len(points) > 3 {
        fmt.Print("\nWARNING BRUTEFORCE USED FOR A SLICE OF ",
len(points), " POINTS", "\n\n")
    }

    for id1, p1 := range points {
        for id2, p2 := range points[id1+1:] {
            if p1.ID == p2.ID {
                continue
            }

            dist := getEuclideanDistance(p1, p2)

            if delta > dist {
                delta = dist
                idA, idB = id1, id2+id1+1
            }
        }
    }

    return points[idA], points[idB], delta
}

func (p Point) getPointsInRange(points []Point, d float64,
curr_dimension int) []Point {
    var pointsInRange []Point
    if curr_dimension == 1 {
        for _, other := range points {
            if p.ID == other.ID { // dont compare with itself
                continue
            }
            axis := p.dimension - curr_dimension
            if math.Abs(p.GetAxisValue(axis) -
other.GetAxisValue(axis)) <= d {
                pointsInRange = append(pointsInRange, other)
            }
        }
    } else {
        candidates := p.getPointsInRange(points, d, curr_dimension
- 1)
        axis := p.dimension - curr_dimension
        for _, candidate := range candidates {
            if getEuclideanDistanceAxis(p, candidate, axis) <= d {
                pointsInRange = append(pointsInRange, candidate)
            }
        }
```

```
    }

    return pointsInRange
}

func getClosestPair(P []Point, n int) (Point, Point, float64) {
    if n <= 3 {
        return getClosestByForce(P...)
    }
    mid := n / 2

    var a, b, a1, a2, b1, b2 Point
    var d, dl, dr float64

    a1, b1, dl = getClosestPair(P[:mid], mid) // note that slices
arent inclusive on both sides -> [lo:hi)
    a2, b2, dr = getClosestPair(P[mid:], n-mid)

    if dl < dr {
        d = dl
        a, b = a1, b1
    } else {
        d = dr
        a, b = a2, b2
    }

    // TODO : optimize this part
    midX := (P[mid-1].GetAxisValue(0) + P[mid].GetAxisValue(0)) / 2

    var strip_l, strip_r []Point
    for _, point := range P {
        if math.Abs(point.GetAxisValue(0) - midX) < d {
            if point.GetAxisValue(0) < midX {
                strip_l = append(strip_l, point)
            } else {
                strip_r = append(strip_r, point)
            }
        }
    }

    for _, point := range strip_l {
        candidates := point.getPointsInRange(strip_r, d,
point.dimension)
        for _, candidate := range candidates {
            dist := getEuclideanDistance(point, candidate)
            if dist < d {
                d = dist
                a, b = point, candidate
```

```
            }
        }
    }

    return a, b, d
}

func (s *Solver) Solve() {
    if !s.isInitialized {
        fmt.Println("Solver has not been initialized!")
        return
    }

    euclidOpsCount = 0

    s.solutionPoints[0], s.solutionPoints[1], s.solutionDist =
getClosestPair(s.points, len(s.points))
    s.solutionFound = true
}

func (s *Solver) SolveByForce() {
    if !s.isInitialized {
        fmt.Println("Solver has not been initialized!")
        return
    }

    euclidOpsCount = 0

    s.solutionPoints = [2]Point{s.points[0], s.points[1]}
    s.solutionDist = getEuclideanDistance(s.points[0], s.points[1])
    s.solutionFound = true

    for i, _ := range s.points {
        for j, _ := range s.points[i+1:] {
            tempDist := getEuclideanDistance(s.points[i],
s.points[j+i+1])

            if s.solutionDist > tempDist {
                s.solutionDist = tempDist
                s.solutionPoints = [2]Point{s.points[i],
s.points[j+i+1]}
            }
        }
    }
}

func (s *Solver) Describe() {
    if !s.solutionFound {
```

```
        fmt.Println("This solver hasn't found any solution")
        return
    }

    fmt.Println("[Closest Pair]")
    fmt.Println("Dimension:", s.dimension)
    fmt.Println("Point 1")
    fmt.Println("ID:", s.solutionPoints[0].ID)
    fmt.Println("Position:", s.solutionPoints[0].val)
    fmt.Println("Point 2")
    fmt.Println("ID:", s.solutionPoints[1].ID)
    fmt.Println("Position:", s.solutionPoints[1].val)
    fmt.Println()
    fmt.Println("Delta:", s.solutionDist)
    fmt.Println("Number of Operations:", euclidOpsCount)
    fmt.Println()

}

func (s *Solver) getSolutionPoints() [2]Point {
    return s.solutionPoints
}
```

4.4.4. Visualizer.go

```
package main

import (
    "net/http"

    "github.com/go-echarts/go-echarts/v2/charts"
    "github.com/go-echarts/go-echarts/v2/opts"
    "github.com/icza/gox/osx"
)

type Visualizer struct{
    points []Point
    solutionPoints [2]Point
}

func NewVisualizer(points []Point, solution [2]Point) *Visualizer {
    v := new(Visualizer)
    v.points = points
    v.solutionPoints = solution

    return v
}
```

```
func (v Visualizer) visualize() {
    scatter3d := charts.NewScatter3D()
    scatter3d.Chart3D.SetGlobalOptions(
        charts.WithInitializationOpts(opts.Initialization{
            PageTitle: "Closest Pair Visualization",
            Width: "90vw",
            Height: "90vh",
            BackgroundColor: "#F2F4F7",
        }),
        charts.WithTitleOpts(opts.Title{
            Title:    "Closest Pair Problem : 3D Visualization",
            Subtitle: "Author : Addin Munawwar Yusuf and Fakih
Anugerah Pratama",
            Top:      "5%",
            Bottom:   "5%",
        }),
    )

    scatter3d.AddSeries("scatter3d", v.genScatter3dData(v.points,
v.solutionPoints))
    http.HandleFunc("/", func(w http.ResponseWriter, _
*http.Request) {
        scatter3d.Render(w)
    })
    osx.OpenDefault("http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}

func (v Visualizer) genScatter3dData(points []Point, solutionPoints
[2]Point) []opts.Chart3DData {
    var data []opts.Chart3DData
    for i, point := range points {
        if point.Equals(solutionPoints[0]) ||
point.Equals(solutionPoints[1]) {}
        data = append(data, opts.Chart3DData{
            Name:  "Point " + string(i),
            Value: []interface{}{point.GetAxisValue(0),
point.GetAxisValue(1), point.GetAxisValue(2)},
            ItemStyle: &opts.ItemStyle{
                Color:   "#1ECBE1",
                Opacity: 1,
            },
        })
    }

    for i, point := range solutionPoints {
        data = append(data, opts.Chart3DData{
            Name:  "Solution Point " + string(i),
```

```
            Value: []interface{}{point.GetAxisValue(0),
point.GetAxisValue(1), point.GetAxisValue(2)},
            ItemStyle: &opts.ItemStyle{
                Color:   "#E1341E",
                Opacity: 1,
            },
        })
    }

    return data
}
```

4.4.5. ExecTimer.go

```
package main

import (
    "fmt"
    "time"
)

type ExecTimer struct {
    name        string
    startTime   time.Time
    elapsedTime time.Duration
}

func NewExecTimer(name string) *ExecTimer {
    et := new(ExecTimer)

    et.name = name

    return et
}

func (et *ExecTimer) Start() {
    et.startTime = time.Now()
}

func (et *ExecTimer) Finish() {
    et.elapsedTime = time.Since(et.startTime)
}

func (et *ExecTimer) Tell() {
    fmt.Println(et.name, "took", et.elapsedTime)
}
```

4.4.6. IOHandler.go

```go
package main

import (
    "fmt"
    "strconv"
)

type IOHandler struct{}

func NewIOHandler() *IOHandler {
    return new(IOHandler)
}

func (i IOHandler) readUserConfig() (int, int, float64, float64) {
    var n, dimension int
    var lowerBound, upperBound float64

    for {
        n = i.GetInt("Enter number of points : ")

        if n >= 2 {
            break
        }

        fmt.Println("Masukan harus >= 2 !")
    }

    for {
        dimension = i.GetInt("Enter dimension : ")

        if dimension >= 1 {
            break
        }

        fmt.Println("Masukan harus >= 1 !")
    }

    lowerBound = i.GetFloat64("Enter lower bound : ")

    upperBound = i.GetFloat64("Enter upper bound : ")

    if lowerBound > upperBound {
        fmt.Println("\nNilai batas lower lebih tinggi dari upper!")
        fmt.Println("Menukar nilai...\n")

        lowerBound, upperBound = upperBound, lowerBound
    }
```

```
    fmt.Println()

    return n, dimension, lowerBound, upperBound
}

func (i IOHandler) askToVisualize() bool {
    var visualize string

    fmt.Printf("\nVisualize your result? (y/n) : ")
    fmt.Scan(&visualize)

    for visualize != "y" && visualize != "n" {
        fmt.Printf("\nInvalid input. Visualize? (y/n) : ")
        fmt.Scan(&visualize)
    }

    if visualize == "y" {
        fmt.Println("\nVisualizing... (Ctrl-C to exit))")
        return true
    } else {
        return false
    }
}

func (i IOHandler) PrintLine(msg string) {
    fmt.Println(msg)
}

func (i IOHandler) GetInt(msg string) int {
    var temp string
    var res int64
    var err error

    for {
        fmt.Printf(msg)
        fmt.Scan(&temp)
        res, err = strconv.ParseInt(temp, 10, 64)

        if err == nil {
            break
        }

        fmt.Println("Masukan salah!")
    }

    return int(res)
}
```

```go
func (i IOHandler) GetFloat64(msg string) float64 {
    var temp string
    var res float64
    var err error

    for {
        fmt.Printf(msg)
        fmt.Scan(&temp)
        res, err = strconv.ParseFloat(temp, 64)

        if err == nil {
            break
        }

        fmt.Println("Masukan salah!")
    }

    return res
}
```

4.5. Keluaran Program

4.5.1. Percobaan 1

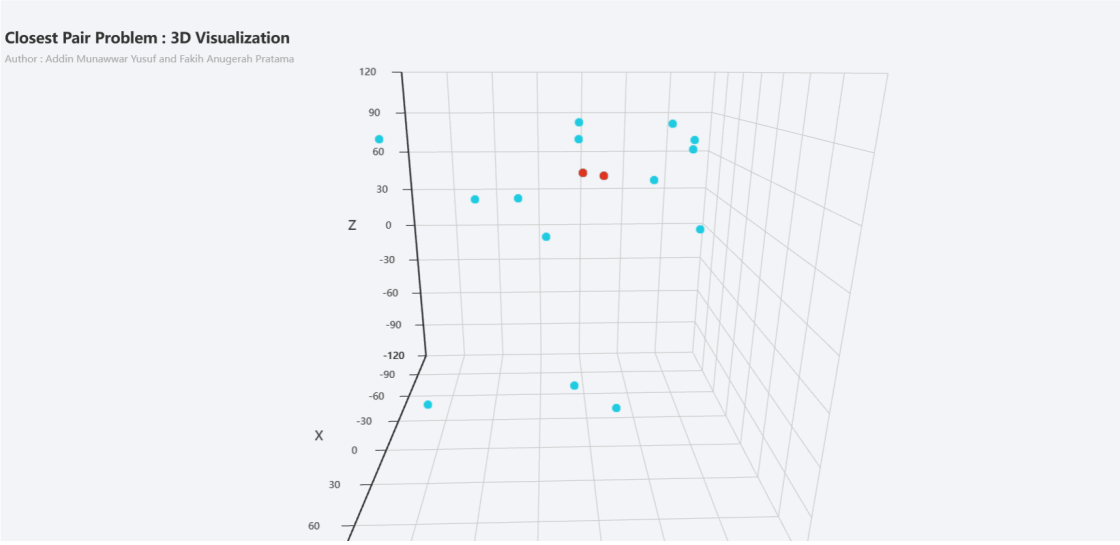
n = 16, dimensi = 3, lowerbound = -100, upperbound = 100

Output

```
USER CONFIGURATION
----------------------
Enter number of points : 16
Enter dimension : 3
Enter lower bound : -100
Enter upper bound : 100

COMPUTER SPECIFICATIONS
----------------------
CPU: AMD Ryzen 7 5700U with Radeon Graphics
Host: DESKTOP-H0NGB0H
Memory: 15706 MB
Disk: 387855 MB

QuickSort took 0s
```

```
WITH BRUTEFORCE
-----------------------
[Closest Pair]
Dimension: 3
Point 1
ID: 3
Position: [1.5360147138335947 -6.190356220782462 65.98981074648981]
Point 2
ID: 4
Position: [30.33358642041844 1.50838497206216 69.85743096262317]

Delta: 30.058763086436908
Number of Operations: 121

Bruteforce took 522.3µs

WITH DIVIDE AND CONQUER
-----------------------
[Closest Pair]
Dimension: 3
Point 1
ID: 3
Position: [1.5360147138335947 -6.190356220782462 65.98981074648981]
Point 2
ID: 4
Position: [30.33358642041844 1.50838497206216 69.85743096262317]

Delta: 30.058763086436908
Number of Operations: 40

Divide and Conquer took 1.5493ms

Visualize your result? (y/n) : y

Visualizing... (Ctrl-C to exit))
```

Visualisasi



4.5.2. Percobaan 2

n = 64, dimensi = 4, lowerbound = -500, upperbound = 500

Output

```
USER CONFIGURATION
----------------------
Enter number of points : 64
Enter dimension : 4
Enter lower bound : -500
Enter upper bound : 500

COMPUTER SPECIFICATIONS
----------------------
CPU: AMD Ryzen 7 5700U with Radeon Graphics
Host: DESKTOP-H0NGB0H
Memory: 15706 MB
Disk: 387855 MB

QuickSort took 0s

WITH BRUTEFORCE
----------------------
[Closest Pair]
Dimension: 4
Point 1
ID: 13
Position: [-458.0141191257709 -41.703531289583395 -222.69718272611783 -222.12675923397597]
```

```
Point 2
ID: 64
Position: [-418.716968434769 -4.57874532451882 -127.71377465766773 -316.53156044474645]

Delta: 144.41824706499511
Number of Operations: 2017

Bruteforce took 2.1071ms

WITH DIVIDE AND CONQUER
-----------------------
[Closest Pair]
Dimension: 4
Point 1
ID: 13
Position: [-458.0141191257709 -41.703531289583395 -222.69718272611783 -222.12675923397597]
Point 2
ID: 64
Position: [-418.716968434769 -4.57874532451882 -127.71377465766773 -316.53156044474645]

Delta: 144.41824706499511
Number of Operations: 521

Divide and Conquer took 1.5558ms

Use 3 dimension input for full experience with the visualizer:D

Thank you for using this program.
```

Visualisasi
Tidak ada visualisasi untuk non-3D

4.5.3. Percobaan 3

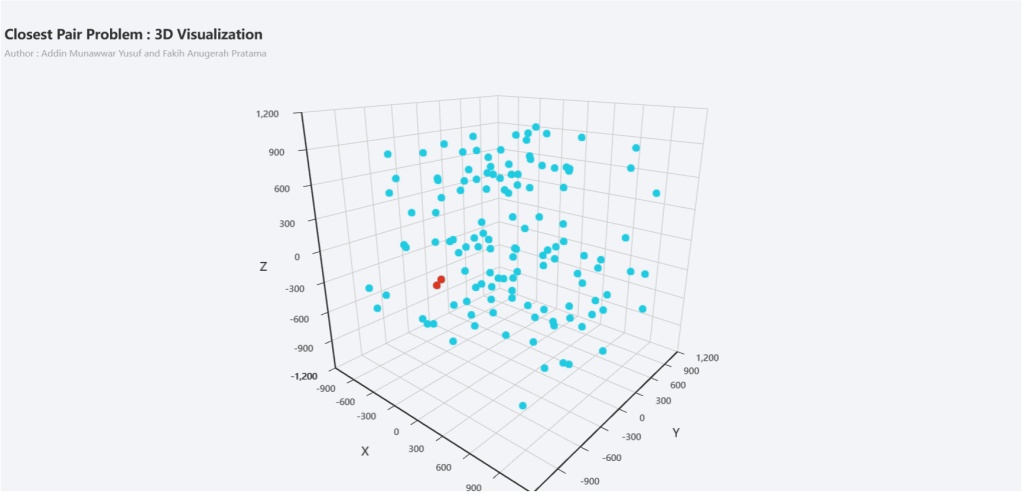
n = 128, dimensi = 3, lowerbound = -1000, upperbound = 1000

Output

```
USER CONFIGURATION
------------------------
Enter number of points : 128
Enter dimension : 3
Enter lower bound : -1000
Enter upper bound : 1000

COMPUTER SPECIFICATIONS
------------------------
```

```
CPU: AMD Ryzen 7 5700U with Radeon Graphics
Host: DESKTOP-H0NGB0H
Memory: 15706 MB
Disk: 387855 MB

QuickSort took 0s

WITH BRUTEFORCE
-----------------------
[Closest Pair]
Dimension: 3
Point 1
ID: 53
Position: [-491.24740063703484 -484.83713316504804 -368.3841933801823]
Point 2
ID: 105
Position: [-467.2823598192804 -448.5396790481459 -314.2446885578381]

Delta: 69.4472054093891
Number of Operations: 8129

Bruteforce took 2.0637ms

WITH DIVIDE AND CONQUER
-----------------------
[Closest Pair]
Dimension: 3
Point 1
ID: 53
Position: [-491.24740063703484 -484.83713316504804 -368.3841933801823]
Point 2
ID: 105
Position: [-467.2823598192804 -448.5396790481459 -314.2446885578381]

Delta: 69.4472054093891
Number of Operations: 429

Divide and Conquer took 1.0343ms

Visualize your result? (y/n) : y

Visualizing... (Ctrl-C to exit))
```

Visualisasi



4.5.4. Percobaan 4

n = 1000, dimensi = 7, lowerbound = -1000, upperbound = 1000

Output

```
USER CONFIGURATION
----------------------
Enter number of points : 1000
Enter dimension : 7
Enter lower bound : -1000
Enter upper bound : 1000

COMPUTER SPECIFICATIONS
----------------------
CPU: AMD Ryzen 7 5700U with Radeon Graphics
Host: DESKTOP-H0NGB0H
Memory: 15706 MB
Disk: 387855 MB

QuickSort took 0s
```

```
WITH BRUTEFORCE
-----------------------
[Closest Pair]
Dimension: 7
Point 1
ID: 559
Position: [-179.68456226141768 171.45083838957976 -896.2456166573804 965.942865659661 -773.0361631676053 -164.29096720630412 -687.1970078605266]
Point 2
ID: 75
Position: [-39.47964859021897 322.3704664718698 -949.175061646349 956.2544463992047 -798.6044084739708 -229.5311817082769 -751.1369188278486]

Delta: 233.08341112887172
Number of Operations: 499501

Bruteforce took 103.2004ms

WITH DIVIDE AND CONQUER
-----------------------
[Closest Pair]
Dimension: 7
Point 1
ID: 559
Position: [-179.68456226141768 171.45083838957976 -896.2456166573804 965.942865659661 -773.0361631676053 -164.29096720630412 -687.1970078605266]
Point 2
ID: 75
Position: [-39.47964859021897 322.3704664718698 -949.175061646349 956.2544463992047 -798.6044084739708 -229.5311817082769 -751.1369188278486]

Delta: 233.08341112887172
Number of Operations: 72562

Divide and Conquer took 18.0035ms


Use 3 dimension input for full experience with the visualizer:D

Thank you for using this program.
```

Visualisasi
Tidak ada visualisasi untuk non-3D

5. Lampiran

a. Pranala *Repository* Github

https://github.com/fakihap/Tucil2_13521085_13521091

b. Tabel Ketercapaian

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 benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	