Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Implementasi Convex Hull untuk Visualisasi Tes Linear Separability Dataset dengan Algoritma Divide and Conquer

Disusun oleh:

Malik Akbar Hashemi Rafsanjani

13520105

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

TAHUN AJARAN 2021/2022

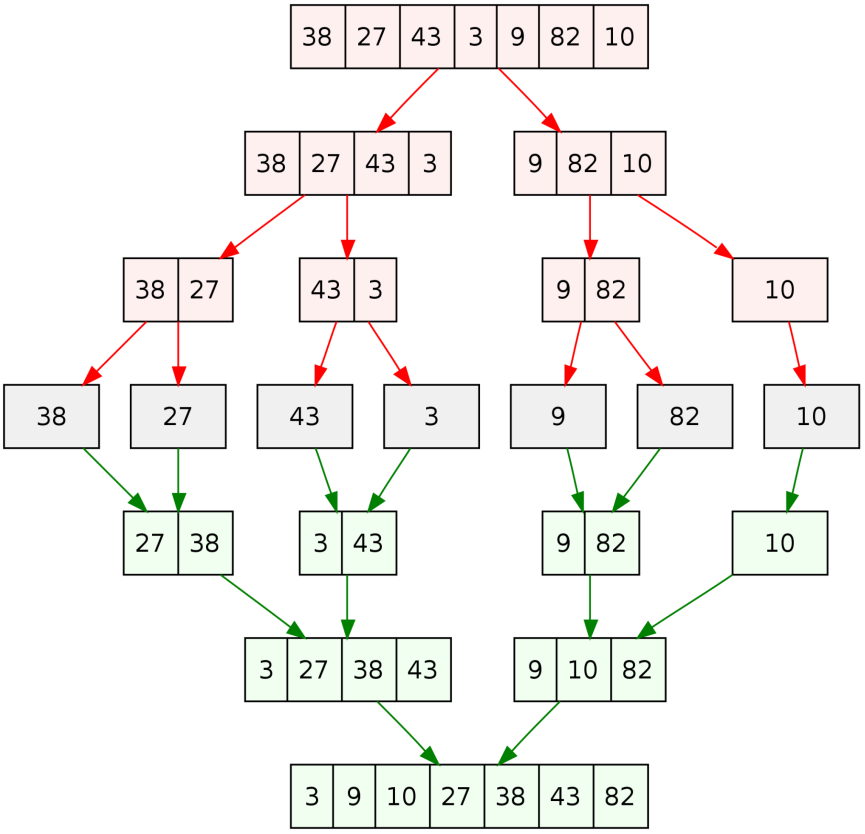
BAB I
Algoritma *Divide and Conquer*

Definisi Algoritma *Divide and Conquer*

Dalam ilmu komputer, algoritma *divide and conquer* merupakan salah satu paradigma dalam mendesain suatu algoritma. Sebuah algoritma divide and conquer secara rekursif memecah masalah menjadi dua atau lebih sub-masalah dari jenis yang sama atau terkait sampai cukup sederhana untuk diselesaikan secara langsung. Solusi untuk upa masalah kemudian digabungkan untuk memberikan solusi untuk masalah asli.

Algoritma divide and conquer terdiri atas 3 tahap yaitu:

1. Divide: memecah masalah menjadi beberapa upa masalah yang mempunyai kemiripan dengan masalah semula tetapi berukuran lebih kecil
2. Conquer: menyelesaikan upa masalah yang telah dibuat secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar
3. Combine: menggabungkan solusi dari masing-masing upa masalah sehingga membentuk solusi masalah semula



Gambar 1: Pendekatan divide and conquer dalam mengurutkan suatu senarai menggunakan algoritma *merge sort*
Dikutip dari: https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

Teknik divide and conquer merupakan dasar dari algoritma yang efisien untuk banyak masalah, seperti pengurutan (misalnya: quicksort dan merge sort), mengalikan bilangan besar (misalnya: algoritma Karatsuba), menemukan pasangan titik terdekat, dan masih banyak lagi. Pada laporan kali ini, akan dibahas lebih mendalam salah satu algoritma divide and conquer dalam implementasi convex hull untuk Visualisasi Tes Linear Separability Dataset.

Skema umum Algoritma Divide and Conquer

```
procedure divideAndConquer(input p: problem, n: integer)
{ Menyelesaikan persoalan p dengan algoritma divide and conquer }
Masukan: persoalan p berukuran n
Luaran: solusi dari persoalan p

Deklarasi
    r: integer

Algoritma
    if n ≤ n_o then
        SOLVE persoalan p berukuran n ini
    else
        DIVIDE menjadi r upa persoalan P_1, P_2, ..., P_r yang masing-masing berukuran n_1, n_2, ..., n_r
        for masing-masing P_1, P_2, ..., P_r do
            divideAndConquer(P_i, n_i)
        endfor
        COMBINE solusi dari P_1, P_2, ..., P_r menjadi solusi persoalan semula
    endif
```

Kompleksitas Algoritma Divide and Conquer

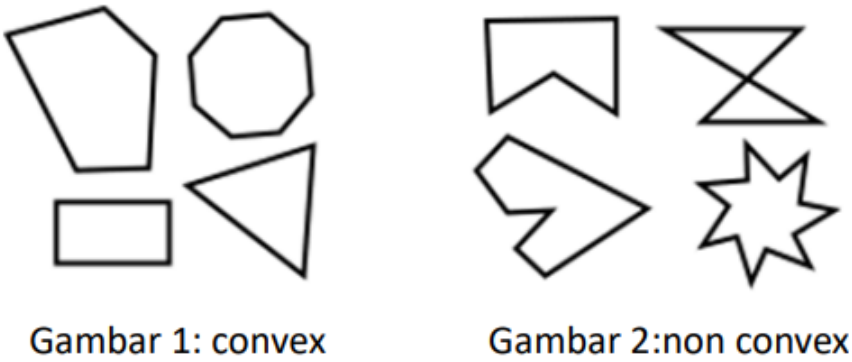
Algoritma divide and conquer pada umumnya memiliki kompleksitas yang dapat direpresentasikan dalam formula sebagai berikut

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

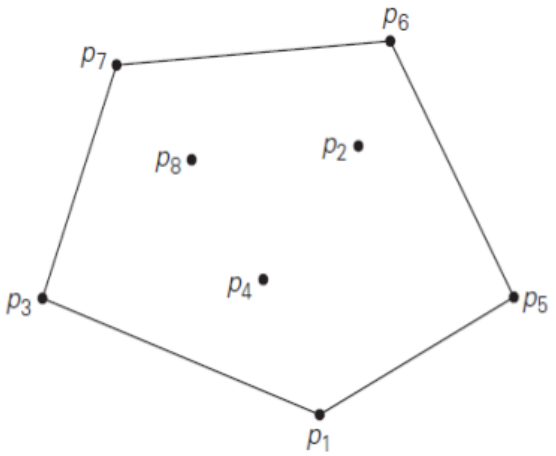
- T(n): kompleksitas waktu penyelesaian persoalan p yang berukuran n
- g(n): kompleksitas waktu untuk menyelesaikan persoalan jika n sudah cukup kecil
- T(n1) + T(n2) + … + T(nr): kompleksitas waktu untuk memproses setiap upa-persoalan
- f(n): kompleksitas waktu untuk menggabungkan solusi dari masing-masing upa persoalan

Divide and conquer pada pembuatan convex hull

Himpunan titik pada bidang planar disebut sebagai convex jika untuk sembarang dua titik pada bidang tersebut (misal p1 dan p2), seluruh segmen garis yang berakhir di p1 dan p2 berada pada himpunan tersebut. Sedangkan convex hull merupakan himpunan convex terkecil dari suatu himpunan titik (misal S) yang mengandung S. Convex hulls memiliki aplikasi yang luas dalam matematika, statistik, optimasi kombinatorial, ekonomi, pemodelan geometris, dan etologi.



Gambar 2: ilustrasi perbandingan antara poligon convex dan poligon non convex
Dikutip dari:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf



Gambar 3: contoh convex hull
Dikutip dari:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf

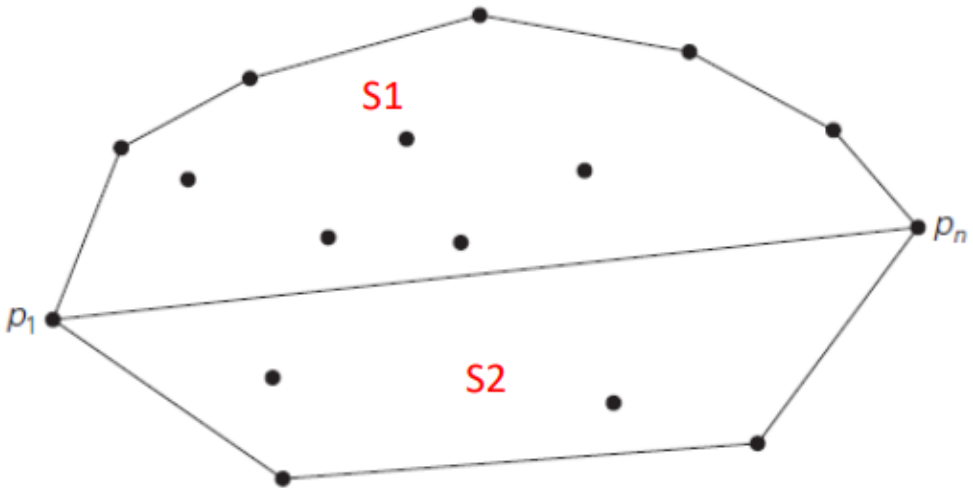
Algoritma untuk menemukan titik-titik terluar yang membentuk convex hull menggunakan algoritma divide and conquer dapat dijabarkan sebagai berikut.

1. Misalkan S merupakan himpunan titik sebanyak n, dengan n>1, yaitu titik P1 (X1, Y1) hinggan Pn (Xn,Yn) pada bidang kartesian dua dimensi

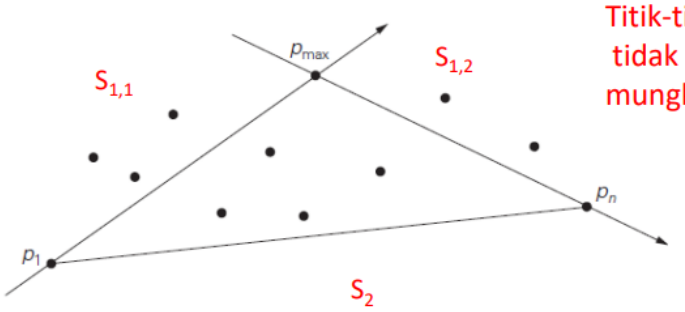
2. Urutkan titik-titik tersebut secara menaik berdasarkan nilai absis. Jika terdapat nilai absis yang sama, maka urutkan secara menaik berdasarkan nilai ordinat.
3. Didapatkan P1 dan Pn, yaitu titik pertama dan titik terakhir dari himpunan titik sebagai dua titik ekstrim yang akan membentuk convex hull dari kumpulan titik tersebut.
4. Garis yang menghubungkan P1 dan Pn membagi S menjadi dua bagian yaitu S1 (kumpulan titik disebelah kiri atas garis) dan S2 (kumpulan titik di sebelah kanan bawah garis).
5. Buat dua himpunan kosong, misal S1 dan S2. Lalu iterasi titik-titik yang ada dari 2 sampai n-1, dan masukkan titik tersebut sesuai dengan posisi relatif terhadap garis P1Pn
6. Untuk menentukan apakah suatu titik berada di sebelah kiri atas atau kanan bawah, dapat digunakan formula determinan sebagai berikut.

$$x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

   Jika bernilai positif maka titik berada di sebelah kiri atas, jika bernilai negatif maka titik berada di sebelah kanan bawah, dan jika bernilai 0, maka titik berada di garis. Titik-titik yang ada digaris tidak perlu dimasukkan ke dalam himpunan yang baru karena tidak mungkin membentuk convex hull.
7. Selesaikan upa persoalan S1 dan S2 dengan divide and conquer secara rekursif.

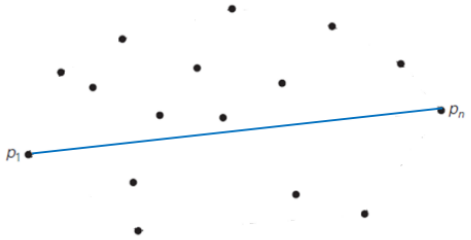


8. Untuk sebuah bagian (misal S1), terdapat dua kemungkinan:
   a. Jika tidak ada titik lain, maka masukkan P1 dan P2 sebagai solusi pembentuk convex hull
   b. Jika S1 tidak kosong, pilih suatu titik yang memiliki jarak terjauh dari garis P1Pn (misal Pmax). Jika terdapat lebih dari satu titik yang memiliki jarak terjauh yang sama, pilih titik yang memaksimalkan sudut Pmax P1 Pn
9. Tentukan kumpulan titik yang berada di sebelah kiri atas garis P1Pmax (menjadi bagian S1,1) dan di sebelah kanan bawah PnPmax

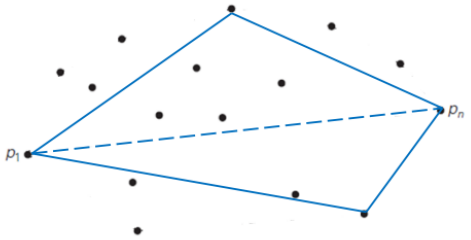


Titik-titik di dalam segitiga tidak diproses, karena tidak mungkin membentuk convex hull

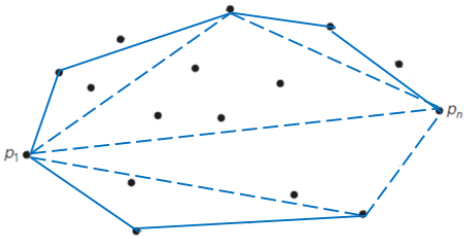
10. Lakukan hal yang sama untuk bagian S2 hingga bagian kiri atas dan kanan bawah kosong.



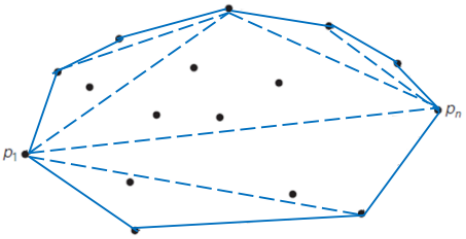
(i)



(ii)



(iii)



(iv)

Algoritma beserta ilustrasi yang dijelaskan diatas dikutip dengan penyesuaian dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf

BAB II
Source Code Program

Source Code Pustaka Implementasi MyConvexHull

```
import sys
import math


class MyConvexHull:
    """A class for solving the convex hull classification problem using
    divide and conquer algorithm.

    Attributes
    ----------
    simplces : list
        list of simplices, list of lists of two elements, containing the
        indices of the two connected points on convex hull
    _points : list
        list of points for classification, list of lists of two
        elements, containing the x and y coordinates of the point
    _sorted_points : list
        list of points sorted ascending by x coordinate then by y
        coordinate, list of lists of two elements, containing the x and
        y coordinates of the points
    """

    simplices = []
    _points = []
    _sorted_points = []

    def __init__(self, points):
        """Constructor for MyConvexHull class

        @type points: list
        @param points: list of points for classification
        """

        if (len(points) <= 1):
            raise Exception("The number of points must be greater than 1")

        self.simplices = []
        self._sorted_points = []

        # add index to points
        self._points = [[x[0], x[1], idx] for idx, x in enumerate(points)]

        self._sorted_points = self._qsort(self._points)
        self._solve_convex_hull()
```

```
    def _solve_convex_hull(self):
        """Solve the convex hull classification using divide and conquer
        algorithm.

        For more details about the algorithm, see:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf
        """

        n = len(self._sorted_points)
        p1 = self._sorted_points[0]
        pn = self._sorted_points[n - 1]

        upper_points = []
        lower_points = []

        for i in range(1, n-1):
            det = self._compare(p1, pn, self._sorted_points[i])

            if (det > 0):
                upper_points.append(self._sorted_points[i])
            elif (det < 0):
                lower_points.append(self._sorted_points[i])

        self._create_hull(upper_points, p1, pn, False)
        self._create_hull(lower_points, p1, pn, True)

    def _get_dist(self, p1, pn, point):
        """Calculate the distance between a point and the line defined
        by two points

        @type p1: list
        @param p1: first point of the line
        @type pn: list
        @param pn: second point of the line
        @type point: list
        @param point: a point whose distance is calculated from the line
        @rtype: list
        @returns: distance between the point and the line
        """

        num = abs((pn[0] - p1[0])*(p1[1] - point[1]) -
                  (p1[0] - point[0]) * (pn[1] - p1[1]))
        denom = ((pn[0] - p1[0])**2 + (pn[1] - p1[1])**2) ** 0.5

        if (denom <= sys.float_info.min and denom >= -sys.float_info.min):
            return num / denom
        else:
            # handle case where denom is very close to zero
```

```
            return (num + sys.float_info.min) / (denom + sys.float_info.min)

    def _length_square(self, p1, p2):
        """Calculate the square of the distance between two points

        @type p1: list
        @param p1: first point
        @type p2: list
        @param p2: second point
        @rtype: float
        @returns: square of the distance between the two points
        """

        xDiff = p1[0] - p2[0]
        yDiff = p1[1] - p2[1]
        return xDiff * xDiff + yDiff * yDiff

    def _calc_angle(self, p1, pn, point):
        """Calculate the angle on p1 between three point

        @type p1: list
        @param p1: point of whose angle is to be calculated
        @type p2: list
        @param p2: second point
        @type point: list
        @param point: third point
        @rtype: float
        @returns: angle between point, p1, and pn
        """

        a2 = self._length_square(pn, point)
        b2 = self._length_square(p1, pn)
        c2 = self._length_square(p1, point)

        b = math.sqrt(b2)
        c = math.sqrt(c2)

        try:
            alpha = math.acos((b2 + c2 - a2) / (2 * b * c))
        except ValueError:
            # handle case where angle is very close to 0 degree
            if ((b2 + c2 - a2) / (2 * b * c) > 1):
                alpha = math.acos(1)
            else:
                alpha = math.acos(-1)

        return alpha * 180 / math.pi

    def _create_hull(self, points, p1, pn, isLower):
        """Create the convex hull recursively. If points is empty, add
        p1 and pn to simplices as solutions for convex hull.
```

```
        @type points: list
        @param points: list of points for classification
        @type p1: list
        @param p1: most left and bottom point
        @type pn: list
        @param pn: most right and top point
        """

        if (len(points) != 0):
            idx_extreme_point = 0
            cur_dist = self._get_dist(p1, pn, points[0])
            cur_angle = self._calc_angle(p1, pn, points[0])

            for i in range(1, len(points)):
                new_dist = self._get_dist(p1, pn, points[i])
                if (new_dist > cur_dist):
                    cur_dist = new_dist
                    idx_extreme_point = i
                elif (new_dist == cur_dist):
                    # compare angle (between current pmax with pmax candidate)
                    # and p1 and pn
                    new_angle = self._calc_angle(p1, pn, points[i])
                    if (new_angle > cur_angle):
                        cur_angle = new_angle
                        idx_extreme_point = i

            upper_points = []
            lower_points = []

            for i in range(len(points)):
                if i != idx_extreme_point:
                    det1 = self._compare(
                        p1, points[idx_extreme_point], points[i])
                    det2 = self._compare(
                        points[idx_extreme_point], pn, points[i])

                    if (isLower):
                        det1 = -1 * det1
                        det2 = -1 * det2

                    if det1 > 0:
                        upper_points.append(points[i])
                    elif det2 > 0:
                        lower_points.append(points[i])

            # solve recursively
            self._create_hull(upper_points, p1,
                              points[idx_extreme_point], isLower)
            self._create_hull(
                lower_points, points[idx_extreme_point], pn, isLower)
```

```
        else:
            self.simplices.append([p1[2], pn[2]])

    def _compare(self, a, b, c):
        """Calculate the determinant of the matrix from three points.
        If the determinant is positive, point c is on the left side of
        the line defined by points a and b.

        @type a: list
        @param a: first point of the line
        @type b: list
        @param b: second point of the line
        @type c: list
        @param c: a point whose relative position from the line is calculated
        @rtype: float
        @returns: determinant of the matrix
        """

        return (a[0] * b[1] + b[0] * c[1] + c[0] * a[1]) -
               (a[1] * b[0] + b[1] * c[0] + c[1] * a[0])

    def _qsort(self, lst):
        """Sort the list of points using quick sort algorithm

        @type lst: list
        @param lst: list of points to be sorted
        @rtype: list
        @returns: sorted list of points
        """

        if len(lst) == 0:
            return []
        else:
            pivot = lst[0]
            lesser = self._qsort(
                [x for x in lst[1:]
                 if (x[0] < pivot[0])
                 or (x[0] == pivot[0] and x[1] < pivot[1])
                 ])
            greater = self._qsort(
                [x for x in lst[1:]
                 if (x[0] > pivot[0])
                 or (x[0] == pivot[0] and x[1] >= pivot[1])
                 ])
            return lesser + [pivot] + greater
```

Source Code Program Utama untuk Menjalankan Visualisasi Dataset

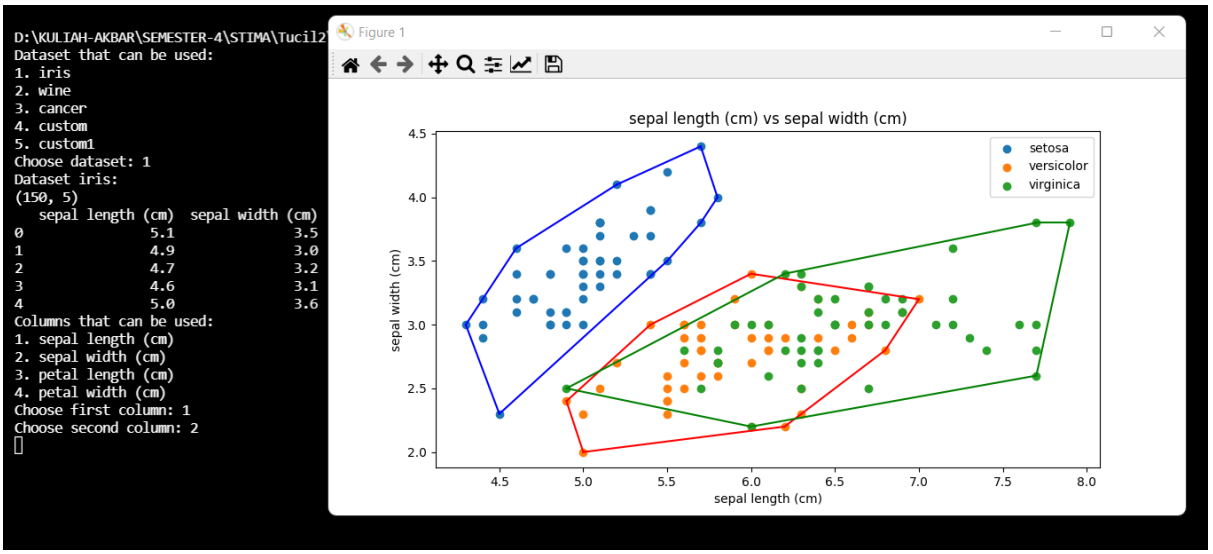
```
from scipy.spatial import ConvexHull
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from MyConvexHull import MyConvexHull
from CustomDataset import CustomDataset

datasets_dict = {
    'iris': datasets.load_iris(),
    'wine': datasets.load_wine(),
    'cancer': datasets.load_breast_cancer(),
    'custom': CustomDataset({
        'data': [[0, 0], [1, 0], [0, 1], [1, 1], [0, 0], [1, 0], [0, 1], [1, 1],
                 [0.5, 0.5], [0.5, 0.5], [-1, -1]],
        'target': [0 for i in range(11)],
        'target_names': ['uniform'],
        'feature_names': ['X', 'Y'],
    }),
    'custom1': CustomDataset({
        'data': [[0, 0], [1, 1], [2, 1]],
        'target': [0 for i in range(3)],
        'target_names': ['uniform'],
        'feature_names': ['X', 'Y'],
    }),
}

def choose_dataset():
    print("Dataset that can be used:")
    lst_datasets = [x for x in datasets_dict]
    for i in range(len(lst_datasets)):
        print(f"{i+1}. {lst_datasets[i]}")

    idx = input("Choose dataset: ")
    return lst_datasets[int(idx)-1]

def choose_colums(feature_names):
    print("Columns that can be used:")
    for i in range(len(feature_names)):
        print(f"{i+1}. {feature_names[i]}")

    idx1 = input("Choose first column: ")
    idx2 = input("Choose second column: ")
    return (int(idx1)-1, int(idx2)-1)

def main():
    dataset = choose_dataset()
    data = datasets_dict[dataset]
```

```
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df['Target'] = pd.DataFrame(data.target)

    print(f"Dataset {dataset}:")
    print(df.shape)
    print(df.head())

    COLUMNS_USED = choose_colums(data.feature_names)

    plt.figure(figsize=(10, 6))
    colors = ['b', 'r', 'g', 'c', 'm', 'y', 'k', 'w']
    plt.title(
        f'{data.feature_names[COLUMNS_USED[0]]} vs
          {data.feature_names[COLUMNS_USED[1]]}')
    plt.xlabel(data.feature_names[COLUMNS_USED[0]])
    plt.ylabel(data.feature_names[COLUMNS_USED[1]])
    for i in range(len(data.target_names)):
        bucket = df[df['Target'] == i]
        bucket = bucket.iloc[:, [COLUMNS_USED[0], COLUMNS_USED[1]]].values
        hull = MyConvexHull(bucket)
        plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
        for simplex in hull.simplices:
            plt.plot(bucket[simplex, 0], bucket[simplex, 1],
                     colors[i % len(colors)])
    plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

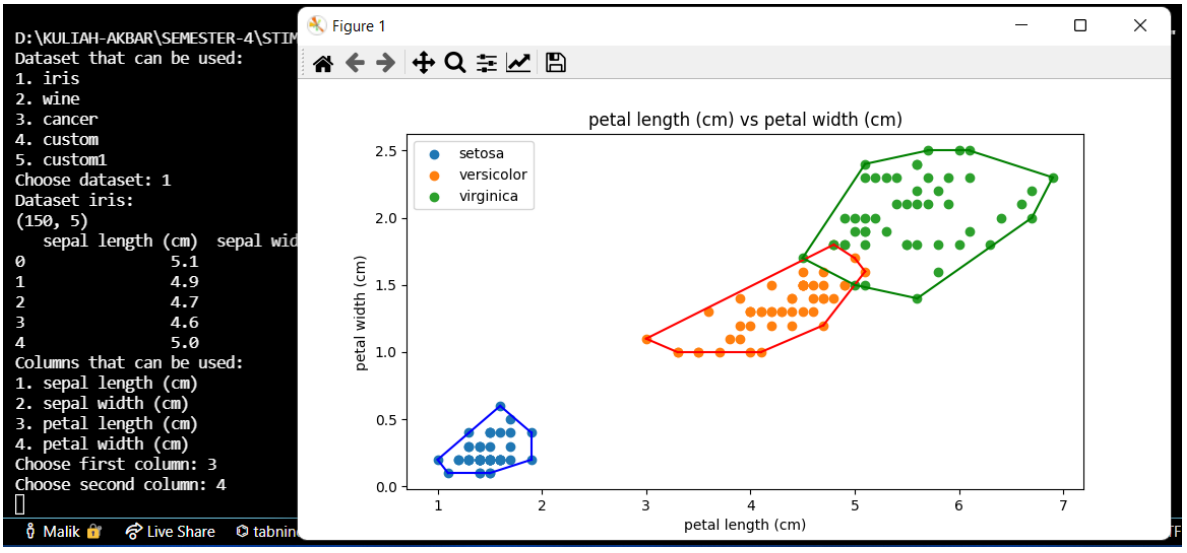
BAB III
Skrinsut Input Output Program

Program ini telah diuji dengan beberapa dataset yang telah disediakan. Berikut merupakan beberapa skrinsut input dan output dari program untuk beberapa dataset dan atribut yang berbeda.

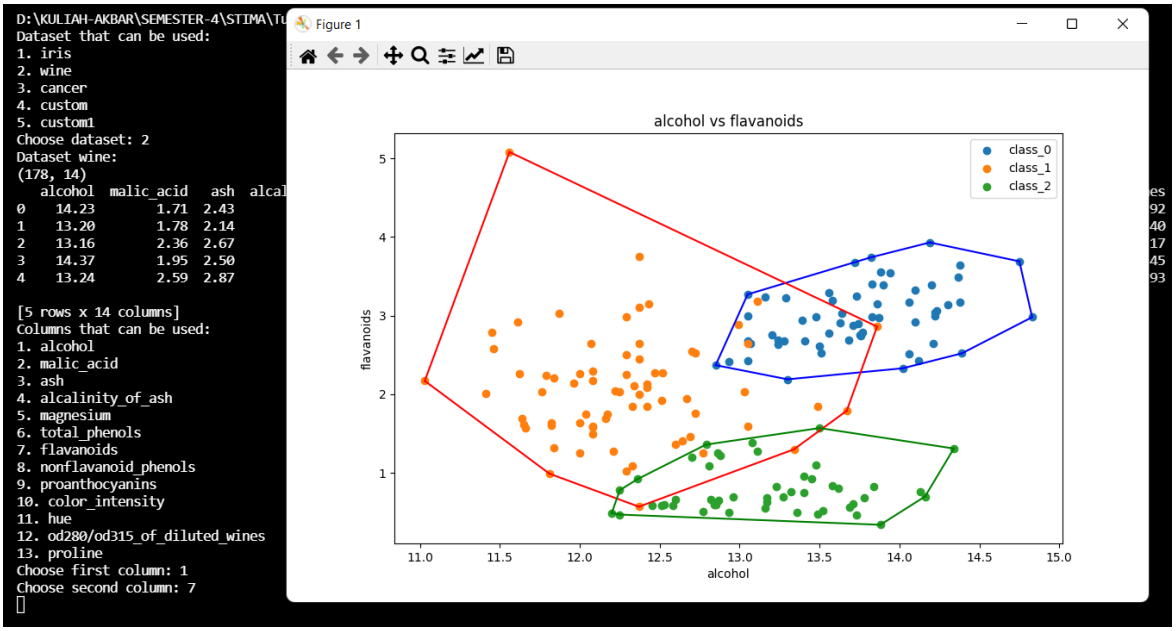
1. Dataset iris (sepal width vs sepal length)



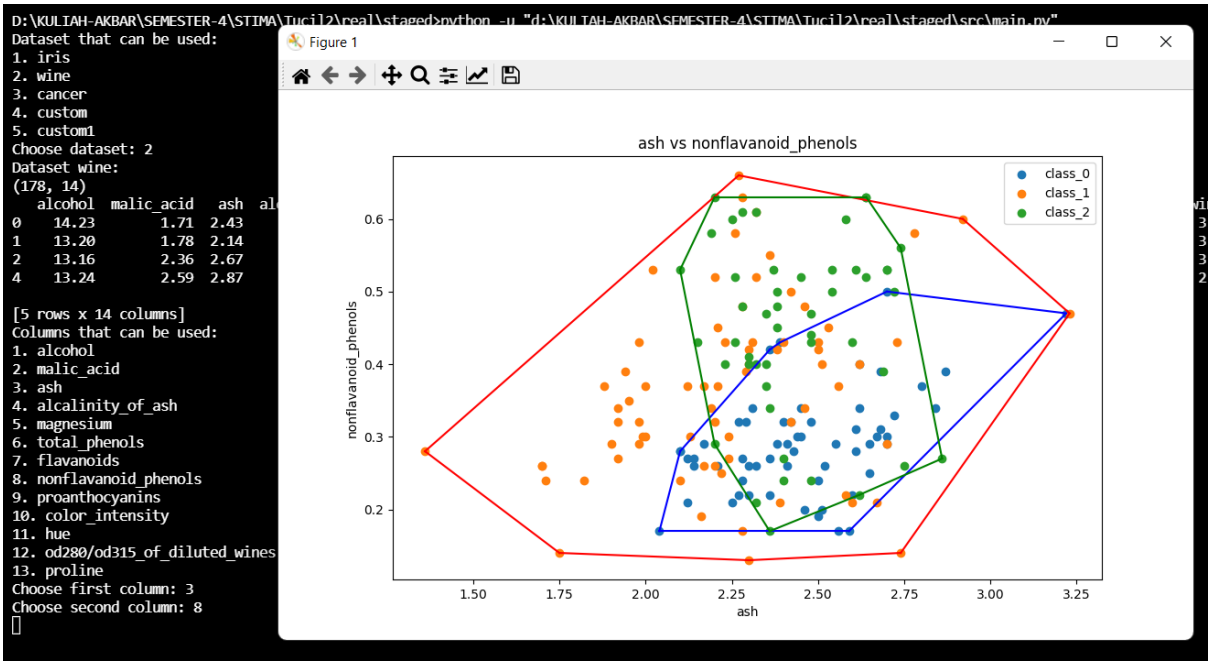
2. Dataset iris (petal length vs petal width)



3. Dataset wine (alcohol vs flavonoids)

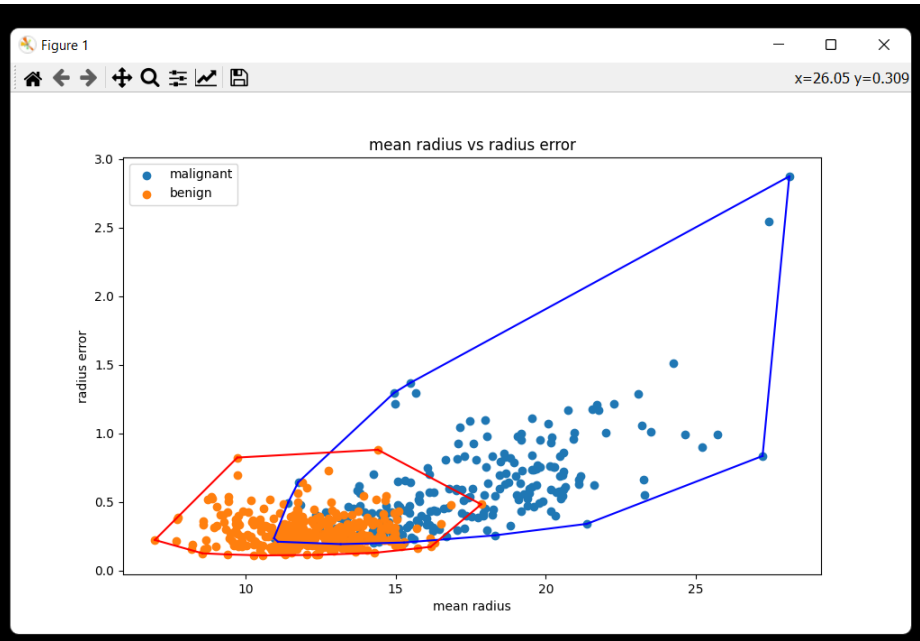


4. Dataset wine (ash vs nonflavonoid_phenols)



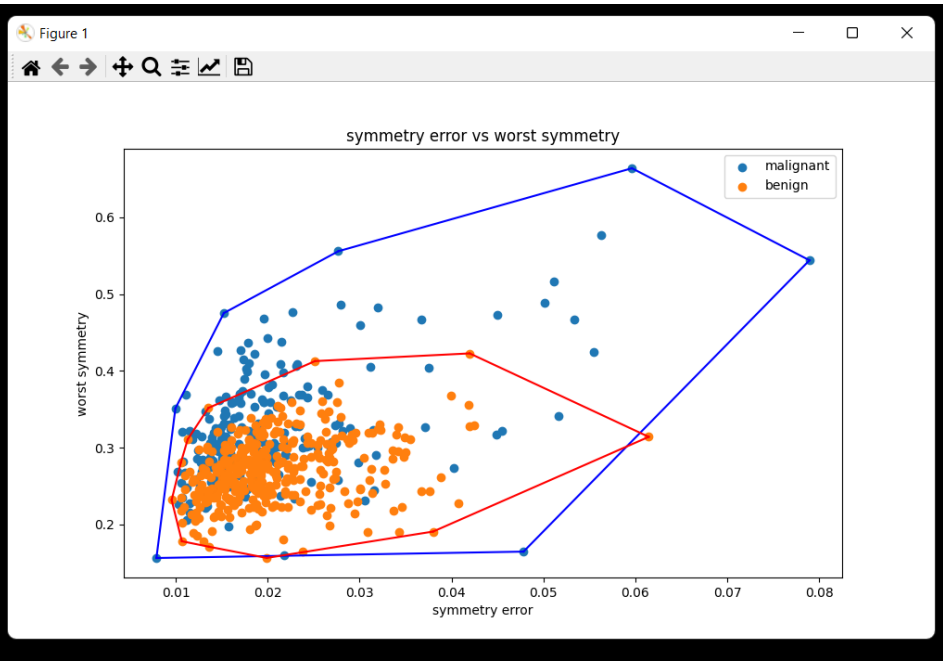
5. Dataset breast cancer (mean radius vs radius error)

```
[5 rows x 31 columns]
Columns that can be used:
1. mean radius
2. mean texture
3. mean perimeter
4. mean area
5. mean smoothness
6. mean compactness
7. mean concavity
8. mean concave points
9. mean symmetry
10. mean fractal dimension
11. radius error
12. texture error
13. perimeter error
14. area error
15. smoothness error
16. compactness error
17. concavity error
18. concave points error
19. symmetry error
20. fractal dimension error
21. worst radius
22. worst texture
23. worst perimeter
24. worst area
25. worst smoothness
26. worst compactness
27. worst concavity
28. worst concave points
29. worst symmetry
30. worst fractal dimension
Choose first column: 1
Choose second column: 11
```



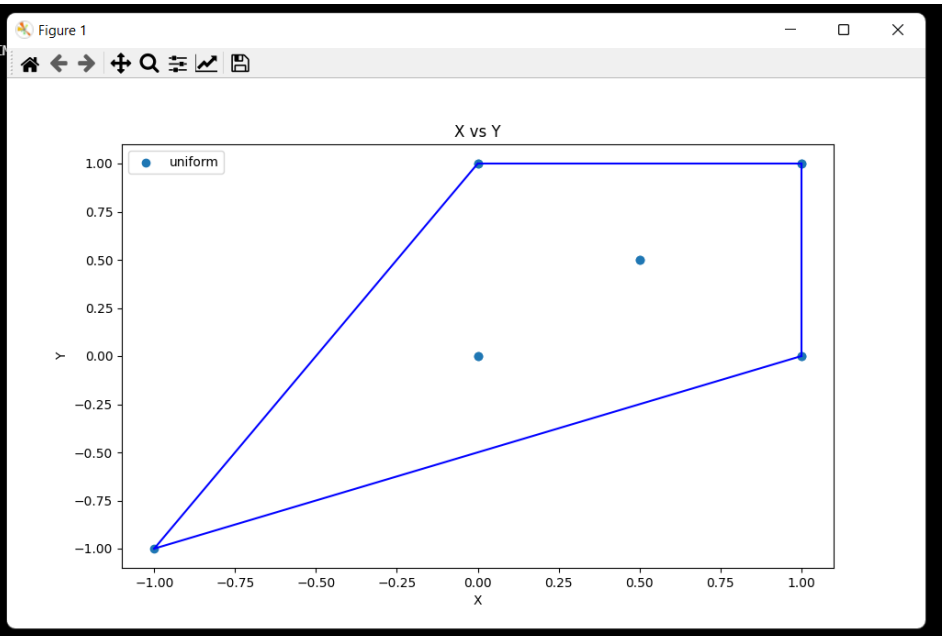
6. Dataset breast cancer (symmetry error vs worst symmetry)

```
[5 rows x 31 columns]
Columns that can be used:
1. mean radius
2. mean texture
3. mean perimeter
4. mean area
5. mean smoothness
6. mean compactness
7. mean concavity
8. mean concave points
9. mean symmetry
10. mean fractal dimension
11. radius error
12. texture error
13. perimeter error
14. area error
15. smoothness error
16. compactness error
17. concavity error
18. concave points error
19. symmetry error
20. fractal dimension error
21. worst radius
22. worst texture
23. worst perimeter
24. worst area
25. worst smoothness
26. worst compactness
27. worst concavity
28. worst concave points
29. worst symmetry
30. worst fractal dimension
Choose first column: 19
Choose second column: 29
```



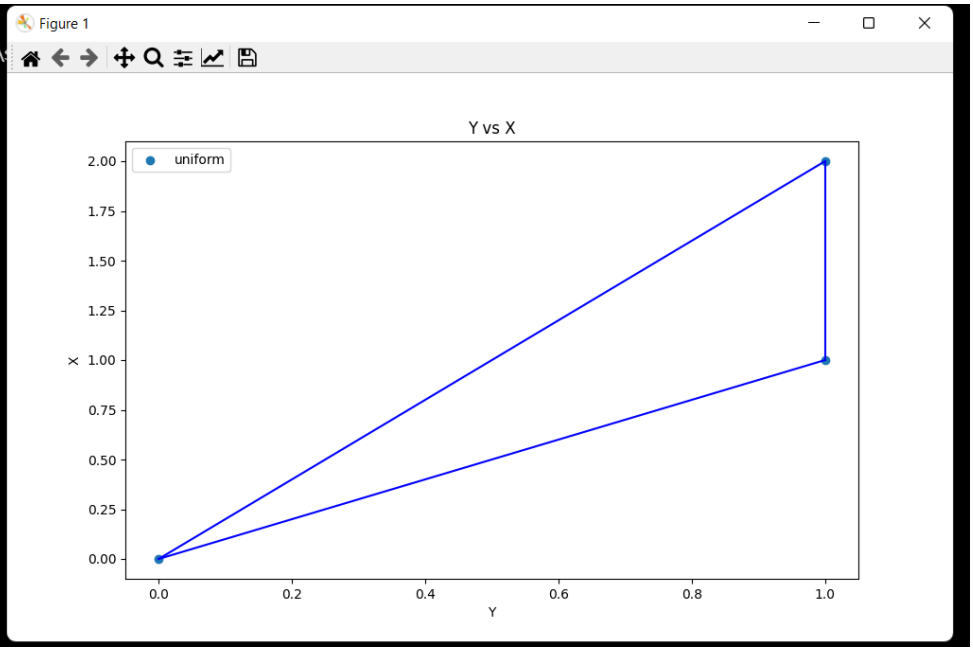
7. Dataset custom (X vs Y)

```
D:\KULIAH-AKBAR\SEMESTER-4\STIN
Dataset that can be used:
1. iris
2. wine
3. cancer
4. custom
5. custom1
Choose dataset: 4
Dataset custom:
(11, 3)
     X    Y  Target
0  0.0  0.0       0
1  1.0  0.0       0
2  0.0  1.0       0
3  1.0  1.0       0
4  0.0  0.0       0
Columns that can be used:
1. X
2. Y
Choose first column: 1
Choose second column: 2
```



8. Dataset custom1 (Y vs X)

```
D:\KULIAH-AKBAR\SEMESTER-4\S
Dataset that can be used:
1. iris
2. wine
3. cancer
4. custom
5. custom1
Choose dataset: 5
Dataset custom1:
(3, 3)
   X  Y  Target
0  0  0       0
1  1  1       0
2  2  1       0
Columns that can be used:
1. X
2. Y
Choose first column: 2
Choose second column: 1
```



BAB IV
Kesimpulan

Kesimpulan yang dapat diambil ialah algoritma divide and conquer dalam membentuk convex hull merupakan algoritma yang cukup sangkil dan mangkus. Hal ini dapat dilihat dari hasil visualisasi convex hull di atas. Selain itu, algoritma ini dapat dimanfaatkan untuk menguji dengan baik untuk berbagai dataset yang disediakan dan dataset yang dibuat sendiri.

Source code program implementasi pustaka MyConvexHull dapat dilihat pada:

- Github: https://github.com/malikrafsan/Convex-Hull-Library-Tucil-2-Stima
- Google Drive: https://drive.google.com/drive/folders/1YL3vPGczCahpoVNhICTf0gbWxn4WsFko

Poin	Ya	Tidak
1. Pustaka myConvexHull berhasil dibuat dan tidak ada kesalahan.	✓	
2. Convex hull yang dihasilkan sudah benar.	✓	
3. Pustaka myConvexHull dapat digunakan untuk menampilkan convex hull setiap label dengan warna yang berbeda.	✓	
4. Bonus: program dapat menerima input dan menuliskan output untuk dataset lainnya.	✓	