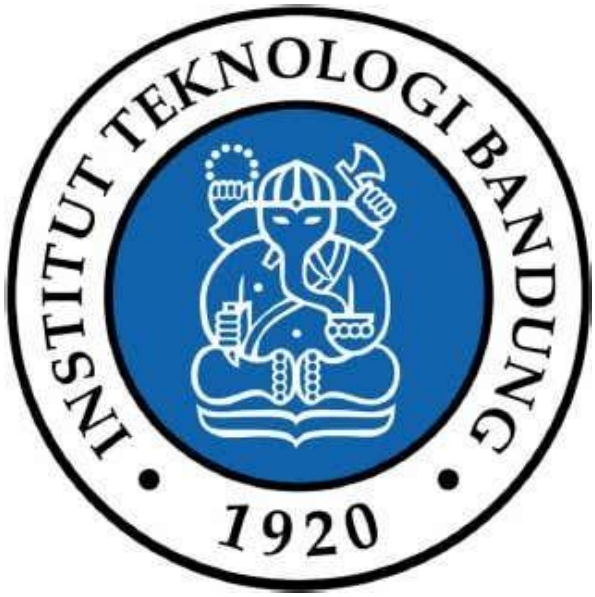
Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

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

Disusun oleh :

Ghazian Tsabit Alkamil

13520165



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*

Algoritma *divide and conquer* memiliki definisi sebagai berikut :

1. *Divide* : membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama)
2. *Conquer (solve)* : menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar)
3. *Combine* : menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Objek persoalan yang dibagi pada algoritma *divide and conquer* adalah masukan atau instances persoalan yang berukuran *n* seperti :

1. Tabel (larik)
2. Matriks
3. Polinom
4. Eksponen
5. dll

Tiap-tiap upa-persoalan memiliki karakteristik yang sama dengan karakteristik persoalan semula, sehingga metode *divide and conquer* lebih natural diungkapkan dalam skema rekursif.

Skema Umum Algoritma *Divide and Conquer*

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

Deklarasi
        r : integer
Algoritma
        if n ≤ n0 then
                SOLVE persoalan P yang berukuran n ini
        else
                DIVIDE menjadi r upa-persoalan, P1, P2,..., Pr, yang masing-masing berukuran
        n1, n2, ..., nr
                for masing-masing P1, P2, ..., Pr, do
                        DIVIDEandCONQUER(Pi, ni)
                endfor
```

COMBINE solusi dari $P_1, P_2, \ldots, P_r$ menjadi solusi persoalan semula

endif

Kompleksitas Algoritma *Divide and Conquer*

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

Penjelasan:

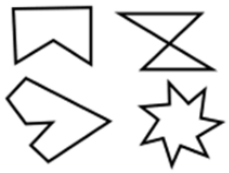
1. T(n) : kompleksitas waktu penyelesaian persoalan P yang berukuran *n*
2. g(n) : kompleksitas waktu untuk SOLVE jika *n* sudah berukuran kecil
3. $T(n_1) + T(n_2) \ldots + T(n_r)$: kompleksitas waktu untuk memproses setiap upa-persoalan
4. f(n) : kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan

Convex Hull

Salah satu hal penting dalam komputasi geometri adalah menentukan *convex hull* dari kumpulan titik. Himpunan titik pada bidang planar disebut *convex* jika untuk sembarang dua titik pada bidang tersebut (misal p dan q), seluruh segmen garis yang berakhir di p dan q berada pada himpunan tersebut.

Contoh gambar 1 adalah poligon yang *convex*, sedangkan gambar 2 menunjukkan contoh yang non-convex.

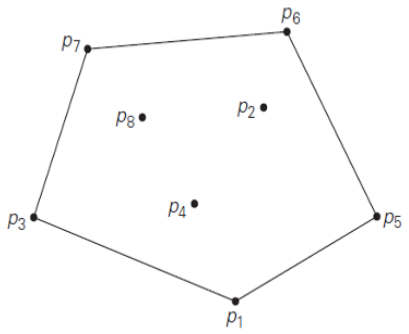
Gambar 1: convex



Gambar 2:non convex

Convex hull dari himpunan titik S adalah himpunan *convex* terkecil (*convex polygon*) yang mengandung S. Untuk dua titik, maka *convex hull* berupa garis yang menghubungkan dua titik tersebut. Untuk tiga titik yang terletak pada satu garis, maka *convex hull* adalah sebuah garis yang menghubungkan dua titik terjauh. Sedangkan *convex hull* untuk tiga titik yang tidak terletak pada satu garis adalah sebuah segitiga yang menghubungkan ketiga titik tersebut. Untuk titik yang lebih banyak dan tidak terletak pada satu garis, maka *convex hull* berupa poligon *convex* dengan sisi berupa garis yang menghubungkan beberapa titik pada S.

Gambar tiga menunjukkan contoh *convex hull* untuk delapan titik.



Gambar 3 Convex Hull untuk delapan titik

BAB II
Source Program

Program *convex hull* untuk visualisasi tes linear separability dataset dengan algoritma *divide and conquer* diimplementasikan menggunakan bahasa pemrograman python. Selain itu implementasi dari program ini juga didukung dengan penggunaan pustaka *numpy* untuk mengolah data yang berasal dari dataset masukan.

Program *convex hull* untuk visualisasi tes linear separability dataset dengan algoritma *divide and conquer* diimplementasikan dengan pembuatan beberapa fungsi yang memenuhi konsep-konsep algoritma *divide and conquer*, fungsi-fungsi tersebut adalah :

1. fungsi *bagi_dua()*

```python
def bagi_dua(start, end, points):
    # apabila ukuran data tidak sesuai
    if points is None or points.shape[0] < 1:
        return None, None
    S1, S2 = [], []

    # menentukan apakah sebuah titik berada di sebelah kiri garis pembagi
    # atau disebelah kanan garis pembagi
    for _, point in enumerate(points):
        dis = posisi_titik(start, end, point)
        if dis > 0:
            S1.append(point)
        else:
            S2.append(point)
    S1 = np.vstack(S1) if len(S1) else None
    S2 = np.vstack(S2) if len(S2) else None
    return S1, S2
```

Gambar 2.1 fungsi *bagi_dua()*

Fungsi *bagi_dua()* berfungsi untuk mengembalikan dua buah array yang berisi beberapa point yang berada di sebelah kiri garis pembagi dan beberapa point yang berada di sebelah kanan garis pembagi. Point yang berada pada kiri garis pembagi akan disimpan dalam array *S1* dan point yang berada pada kanan garis pembagi akan disimpan dalam array *S2*. Garis pembagi dalam hal ini adalah garis yang dihasilkan dari menghubungkan dua buah titik atau point yang memiliki jarak terjauh. Dalam konsep algoritma *divide and conquer* fungsi ini termasuk ke dalam bagian *divide*.

2. fungsi *posisi_titik()*

```
def posisi_titik(start, end, point, epsilon = 1e-8):

    return np.cross(end - start, point-start)/(np.linalg.norm(end-start)+epsilon)
```

Gambar 2.2 fungsi *posisi_titik()*

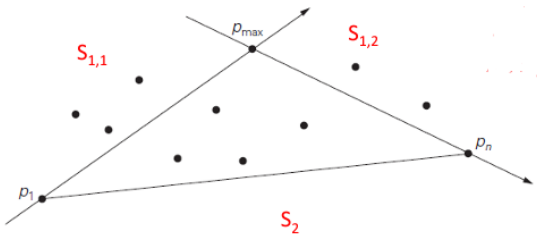
Fungsi *posisi_titik()* berfungsi untuk menentukan apakah sebuah point akan dimasukkan ke dalam array *S1* atau array *S2* pada fungsi *bagi_dua()*. Apabila fungsi *posisi_titik()* mengembalikan nilai positif maka point tersebut akan dimasukkan ke dalam array *S1* dan apabila fungsi *posisi_titik()* mengembalikan nilai negatif maka point akan dimasukkan ke dalam array *S2*.

3. fungsi *pembagian_segitiga()*

```
def pembagian_segitiga(points, P, C, Q):
    if points is None:
        return None, None

    S1, S2 = [], []

    for _, point in enumerate(points):
        disPC = posisi_titik(P, C, point)
        disCQ = posisi_titik(C, Q, point)
        if disPC > 0 and disCQ < 0:
            S1.append(point)
        elif disPC < 0 and disCQ > 0:
            S2.append(point)

    S1 = np.vstack(S1) if len(S1) else None
    S2 = np.vstack(S2) if len(S2) else None

    return S1, S2
```

Gambar 2.3 fungsi *pembagian_segitiga()*

Fungsi *pembagian_segitiga()* berfungsi untuk membagi lagi daerah yang sebelumnya sudah dibagi pada fungsi *bagi_dua()*.



Dapat dilihat pada gambar ilustrasi diatas bahwa point point yang terdapat pada daerah atau array S1 nilai nya di update menjadi point point yang berada pada daerah $S_{1,1}$ dan $S_{1,2}$. Fungsi *posisi_titik()* juga digunakan kembali pada fungsi ini untuk menentukan posisi point yang sesuai yang akan di *assign* ke array *S1* yang baru. Dalam konsep algoritma *divide and conquer* fungsi ini termasuk ke dalam bagian *divide*.

4. fungsi *sudut_terbesar()*

```
def sudut_terbesar(x):
    x0, y0 = x[:,0].mean(), x[:,1].mean()
    theta = np.arctan2(x[:,1] - y0, x[:,0] - x0)
    index = np.argsort(theta)
    x = x[index]

    return x
```

Gambar 2.4 fungsi *sudut_terbesar()*

Fungsi *sudut_terbesar()* berfungsi untuk mengurutkan titik yang berada pada array *S1* atau array *S2* berdasarkan point yang memberikan sudut terbesar.

5. *class myConvexHull*

```
class myConvexHull:

    # class myConvexHull memilik
    def __init__(self):
        self.points = None
        self.convex_hull = []
```

Gambar 2.5 *class myConvexHull*

Implementasi dari fungsi utama program *convex hull* adalah dengan membuat sebuah *class* yang bernama *myConvexHull* yang memiliki atribut *points* dan atribut convex_hull. Atribut *points* berguna untuk menyimpan seluruh point yang akan dicari point *convex hull* nya, atribut *convex hull* yang berupa array berguna untuk menyimpan titik/point yang merupakan titik/point *convex hull*. Selain itu *class myConvexHull* juga memiliki beberapa *method* yang nantinya akan berguna untuk mencari titik atau point yang merupakan titik/point convex hull.

6. fungsi *quickHull()* dan fungsi *findHull()*

```
def quickHull(self):
    # mengurutkan points
    self.points = self.points[np.lexsort(np.transpose(self.points)[::-1])]

    # mencari point paling kiri dan point paling kanan
    point_paling_kiri, point_paling_kanan = self.points[0], self.points[-1]

    self.points = self.points[1:-1] # mendapatkan sisa dari points
    self.convex_hull.append(point_paling_kiri) # point paling kiri sudah pasti
    self.convex_hull.append(point_paling_kanan) # point paling kiri sudah pasti

    self.points_kanan, self.points_kiri = bagi_dua(start=point_paling_kiri,
    end=point_paling_kanan, points = self.points)

    self.findHull(self.points_kanan, point_paling_kiri, point_paling_kanan)
    self.findHull(self.points_kiri, point_paling_kanan, point_paling_kiri)

    self.convex_hull = np.stack(self.convex_hull)
    self.convex_hull = sudut_terbesar(self.convex_hull)

    if self.convex_hull.shape[0] >= 3:
        return self.convex_hull
```

Gambar 2.5 fungsi *quickHull()*

```
def findHull(self, points, P, Q):
    if points is None:
        return None
    distance = 0.0
    C, index = None, None
    for i, point in enumerate(points):
        distance_1 = abs(posisi_titik(P,Q,point))
        if distance_1 > distance:
            C = point
            index = i
            distance = distance_1
    if C is not None:
        self.convex_hull.append(C)
        # point yang sudah dimasukkan kedalam array convex hull
        # array point tersebut
        points = np.delete(points, index, axis=0)
    else:
        return

    S1, S2 = pembagian_segitiga(points, P, C, Q)
    self.findHull(S1, P, C)
    self.findHull(S2, C, Q)
```

Gambar 2.6 fungsi *findHull()*

Fungsi *quickHull()* dan fungsi *findHull()* bersama-sama memiliki fungsi untuk mencari titik/point yang merupakan convex hull. Untuk mencari seluruh titik/point yang merupakan convex hull, pertama-tama titik/point masukan harus diurutkan secara

menaik berdasarkan nilai absis, apabila terdapat nilai absis yang sama maka akan diurutkan secara menaik berdasarkan nilai ordinat. Setelah point diurutkan maka langkah selanjutnya adalah kita ambil titik/point pada urutan pertama dan terakhir, kemudian point pertama dan terakhir tersebut dimasukkan ke dalam atribut convex_hull, karena point/titik pertama dan terakhir sudah pasti merupakan titik/point *convex hull*. Setelah itu, point atau titik yang tersisa akan dikelompokkan kedalam dua array menggunakan fungsi *bagi_dua()* yang sudah diimplementasikan sebelumnya. Kemudian dari masing masing array tersebut akan dicari lagi titik/point yang merupakan titik/point convex hull, proses tersebut akan terus dilakukan sampai tidak ada titik/point yang perlu dicari lagi. Dalam konsep algoritma *divide and conquer* proses ini termasuk ke dalam bagian *conquer*. Kemudian apabila seluruh proses pencarian titik/point telah selesai maka langkah selanjutnya adalah mengkombinasikan hasil pencarian ke dalam satu array *convex_hull*. Dalam konsep algoritma *divide and conquer* proses ini masuk ke dalam bagian *combine*.

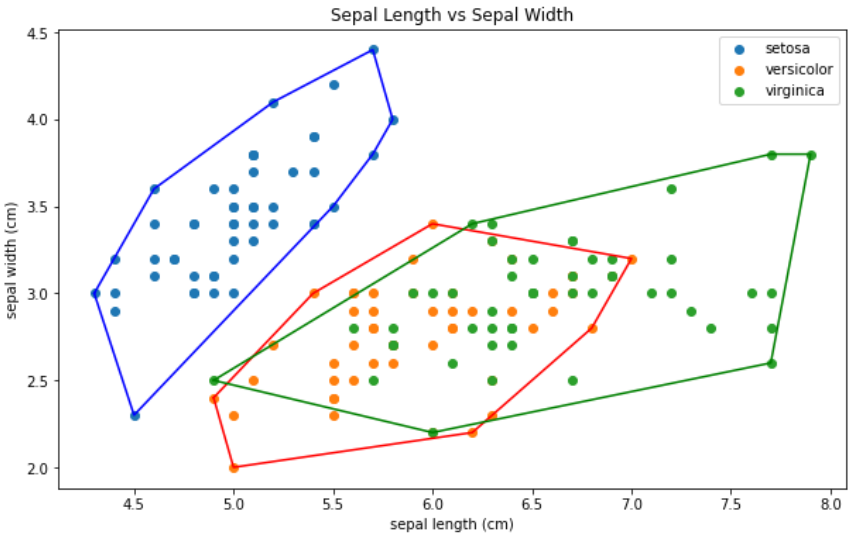
BAB III
Hasil Program

1. Data set iris (sepal-width, sepal-length)

Input :

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

plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Sepal Length vs Sepal Width')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])

model = myConvexHull()

for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[0,1]].values

    hull = model(bucket) #bagian ini diganti dengan hasil implementasi Con

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])

    hull = np.vstack([hull, hull[0]])
    plt.plot(hull[:,0], hull[:,1], colors[i])
plt.legend()
plt.show()
```

Output :



2. Data set iris (petal-width, petal-length)

Input :

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

plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Petal Length vs Petal Width')
plt.xlabel(data.feature_names[2])
plt.ylabel(data.feature_names[3])

model = myConvexHull()

for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[2,3]].values
    # print(bucket)

    hull = model(bucket) #bagian ini diganti dengan hasil implementasi C

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])

    hull = np.vstack([hull, hull[0]])
    plt.plot(hull[:,0], hull[:,1], colors[i])
plt.legend()
plt.show()
```

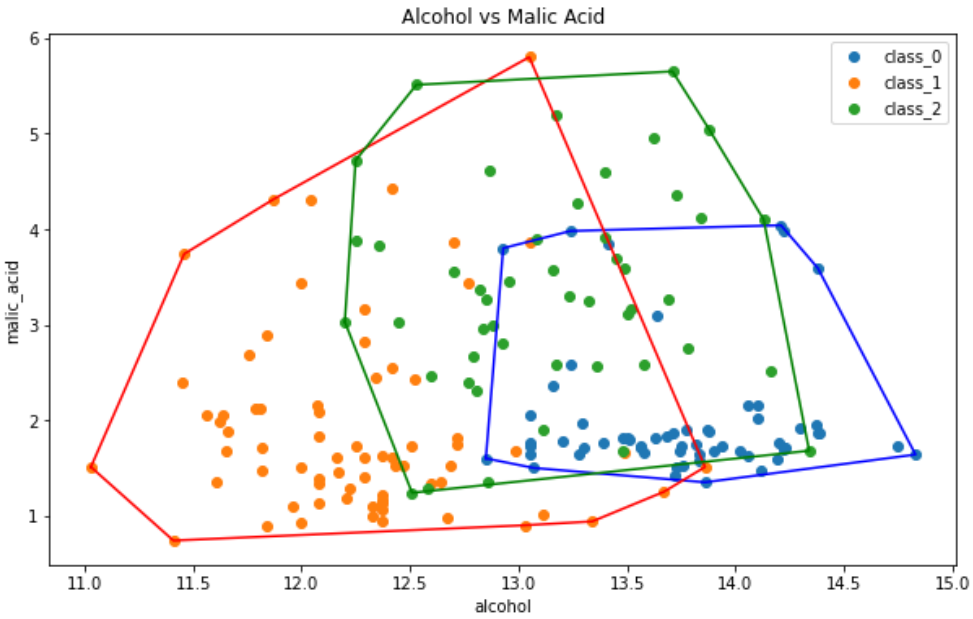
Output :

3. Data set wine (alcohol, malic_acid) *Bonus*

Input :

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

plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Alcohol vs Malic Acid')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])

model = myConvexHull()

for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[0,1]].values

    hull = model(bucket) #bagian ini diganti dengan hasil implementasi C

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])

    hull = np.vstack([hull, hull[0]])
    plt.plot(hull[:,0], hull[:,1], colors[i])
plt.legend()
plt.show()
```

Output :

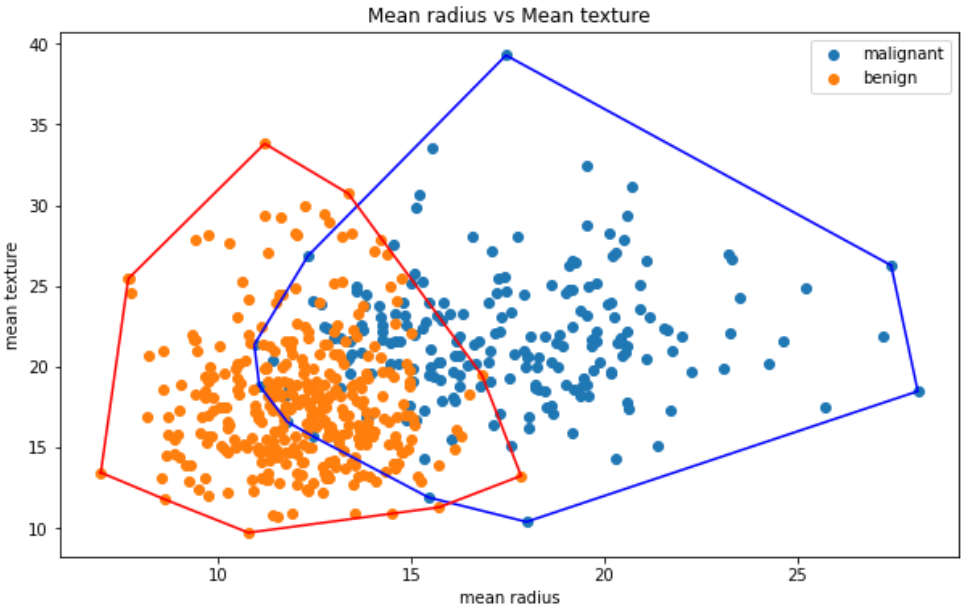


4. Data set breast_cancer (mean radius, mean texture) *Bonus*

Input :

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

plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Mean radius vs Mean texture')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])

model = myConvexHull()

for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[0,1]].values

    hull = model(bucket)

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])

    hull = np.vstack([hull, hull[0]])
    plt.plot(hull[:,0], hull[:,1], colors[i])
plt.legend()
plt.show()
```

Output :



BAB IV
Kesimpulan

Kesimpulan yang dapat diambil dari hasil implementasi convex hull untuk visualisasi tes linear separability dataset adalah algoritma divide and conquer dapat dimanfaatkan dan diimplementasikan dengan baik untuk dataset iris maupun dataset wine.

Source code program implementasi *convex hull* untuk visualisasi tes linear separability dataset dengan algoritma divide and conquer : https://github.com/ZianTsabit/ConvexHull_Stima

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	✓	

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

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