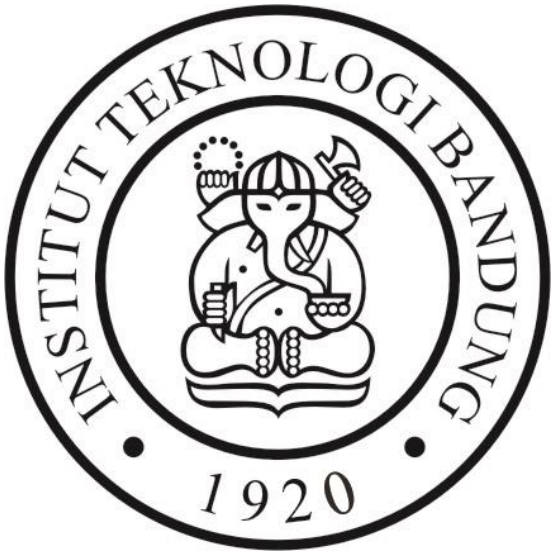
LAPORAN TUGAS KECIL 2

IF 2211 STRATEGI ALGORITMA

PENYELESAIAN PERSOALAN CONVEX HULL DENGAN DIVIDE AND CONQUER (QUICK HULL)



Oleh:

Haifa Fadhila Ilma

13516076

A. *Pseudo-Code* Algoritma

Berikut adalah *pseudo-code* algoritma quick-hull yang telah dibuat:

```
arrPoint <- Array of Input Points
convexHull <- Array of Solved Points

Procedure getConvexHull (arrPoint)
        Sort ( semua titik pada arrPoint )   {Berdasarkan absis. Jika absis sama, berdasarkan ordinat}
        P1 <- arrPoint[Idx pertama]  {P1 adalah titik minimum}
        Pn <- arrPoint[Idx kedua]  {Pn adalah titik maksimum}

        Masukkan P1 dan Pn kedalam array convexHull

        quickHull(P1,Pn,arrTitikdiKananGaris) {Proses titik sebelah kiri garis P1-Pn}
        quickHull(P1,Pn,arrTitikdiKananGaris) {Proses titik sebelah kanan garis P1-Pn}

        return convexHull


Procedure quickHull (P1,Pn,arrTitik)
        If (Masih ada titik di sisi kanan/kiri garis P1-Pn)
                titikMax <- Titik dengan jarak terjauh dari garis P1-Pn
                Masukkan titikMax kedalam array convexHull

{ Memproses secara rekursif titik yang berada disebelah kiri atau kanan garis yang dibentuk P1-titikMax dan titikMax-Pn }
                quickHull(P1,titikMax,arrTitikBaru)
                quickHull(titikMax,Pn,arrTitikBaru)
```

B. Kompleksitas Algoritma

Pada algoritma *convex-hull* secara *divide and conquer* ini, kompleksitas algoritmanya dapat dilihat sebagai berikut:

Kompleksitas waktu untuk Algoritma *quick hull* adalah:

$$\mathbf{T(n) = 2\ T(n/2) + O(n).}$$

Dengan teorema master, akan didapat kompleksitas algoritma dengan notasi Big-O sebagai berikut:

Pada kompleksitas waktu dengan bentuk $T(n) = aT(n/b) + f(n)$, didapat $\mathbf{a = 2}$, $\mathbf{b = 2}$, dan $\mathbf{d = 1}$. Oleh karena itu, dapat dikatakan bahwa $\mathbf{a = b^d}$**.** Oleh karena itu, Kompleksitas algoritmanya adalah:

$$\mathbf{O(n^d \log n)}$$

$$\mathbf{= O(n \log n)}$$

C. *Source Code* Program

Program ini dibuat dengan menggunakan Bahasa *Python.*

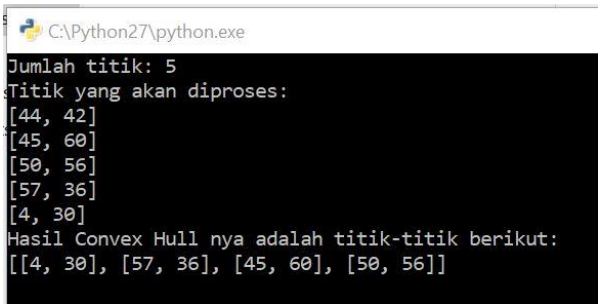
```
#   TUGAS KECIL II IF2211 STRATEGI ALGORITMA
#   Penyelesaian Persoalan Convex Hull dengan Divide and Conquer
#   NAMA    :   Haifa Fadhila Ilma
#   NIM     :   13516076
#   KELAS   :   K-01

import matplotlib.pyplot as plt
import numpy as np
import random

LEFT = 1
RIGHT = -1
INLINE = 0

convexHull = []

def main():
    # Menerima masukan jumlah titik dan men-generate koordinat secara
random
    n = input("Jumlah titik: ")
    while (n<3):
        print("Minimal diperlukan 3 titik untuk membentuk Convex
Hull\n")
        n = input("Jumlah titik: ")

    print("Titik yang akan diproses:")
    for i in range (n):
        x = random.randint(0, 100)
        y = random.randint(0, 100)
        arrTitik.append([x,y])
        print(arrTitik[i])

    arrPoints = arrTitik
    # Pemrosesan kumpulan titik secara Quick-Hull
    quickHull(arrTitik,n)

    print ("Hasil Convex Hull nya adalah titik-titik berikut:")
    print (convexHull)

    # Menggambar semua titik
    absis = []
    ordinat = []
    for i in range(len(arrPoints)):
        point = arrPoints[i]
        absis.append(point[0])
        ordinat.append(point[1])
    plt.plot(absis,ordinat,'ro')
    # Menggambar hasil quick Hull
    drawPoints()

    plt.show()
```

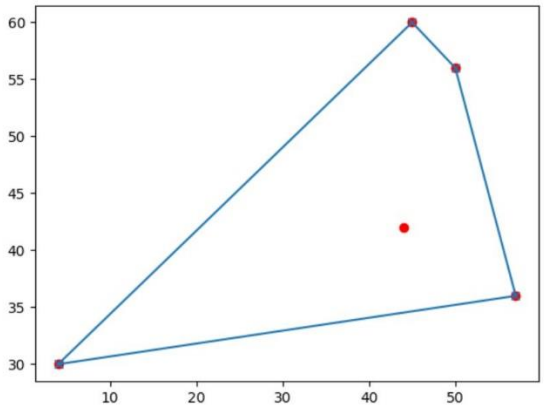
```
def isLeftRight(x1, xn, xc):
# Mengecek apakah titik xc ada disebelah kanan atau kiri garis x1-xn
    det1 = xn[0]*xc[1] + x1[0]*xn[1] + xc[0]*x1[1]
    det2 = xn[0]*x1[1] + xc[0]*xn[1] + x1[0]*xc[1]
    det = det1 - det2

    if det>0:
        return LEFT
    elif det<0:
        return RIGHT
    else:
        return INLINE

def pointToLine(p1, pn, pc):
# Mengembalikan jarak dari titik pc ke garis p1-pn
    dist = ((pc[1] - p1[1]) * (pn[0] - p1[0]) -(pn[1] - p1[1]) * (pc[0]
- p1[0]));
    return abs(dist)

def processLeft(p1, pn, arrKiri):
# Pemrosesan kumpulan titik yang berada dibagian kiri
    if (len(arrKiri)!= 0):
        # Mengambil titik dengan jarak terjauh
        arrKiri.sort(key=lambda x:pointToLine(p1, pn, x), reverse=True)
        maxPoint = arrKiri[0]
        convexHull.append(maxPoint)
        arrKiri.remove(maxPoint)

        # Pemrosesan bagian-bagiannya kembali secara rekursif jika
masih ada titik
        titikKiri1 = getTitik(p1,maxPoint,arrKiri,LEFT, len(arrKiri))
        processLeft(p1,maxPoint,titikKiri1)
        titikKiri2 = getTitik(maxPoint,pn,arrKiri,LEFT, len(arrKiri))
        processLeft(maxPoint,pn,titikKiri2)

def processRight(p1, pn, arrKanan):
# Pemrosesan kumpulan titik yang berada dibagian kanan
    if (len(arrKanan)!= 0):
        # Mengambil titik dengan jarak terjauh
        arrKanan.sort(key=lambda x:pointToLine(p1, pn, x),
reverse=True)
        maxPoint = arrKanan[0]
        convexHull.append(maxPoint)
        arrKanan.remove(maxPoint)

        # Pemrosesan bagian-bagiannya kembali secara rekursif jika
masih ada titik
        titikKanan1 = getTitik(p1,maxPoint,arrKanan,RIGHT,
len(arrKanan))
        processRight(p1,maxPoint,titikKanan1)
        titikKanan2 = getTitik(maxPoint,pn,arrKanan,RIGHT,
len(arrKanan))
        processRight(maxPoint,pn,titikKanan2)

def getTitik(p1, pn, arrTitik, sisi, n):
# Mengembalikan list yang berisi titik di bagian sisi tertentu
```

```
    arr = []
    for i in range(n):
        if (isLeftRight(p1, pn, arrTitik[i])==sisi):
            arr.append(arrTitik[i])
    return arr

def quickHull(arrTitik,n):
# IMPLEMENTASI QUICK HULL DENGAN DIVIDE AND CONQUER

    # Diurutkan, diambil P1 dan Pn
    arrTitik.sort(key=lambda k: [k[0], k[1]])

    P1 = arrTitik[0]
    Pn = arrTitik[n-1]

    convexHull.append(P1)
    convexHull.append(Pn)

    # Divide and conquer = Membagi titik ke sebelah kiri dan kanan
berdasarkan garis P1-Pn
    arrKiri = getTitik(P1,Pn,arrTitik,LEFT, n)
    arrKanan = getTitik(P1,Pn,arrTitik,RIGHT, n)
    processLeft(P1,Pn,arrKiri)
    processRight(P1,Pn,arrKanan)

def makePolygon(arr):
# Mengurutkan titik agar dapat digambar berbentuk poligon
    arr.sort(key=lambda k: [k[0], k[1]])
    leftmost = arr[0]
    rightmost = arr[len(arr)-1]

    arrLeft =  getTitik(leftmost, rightmost, arr, LEFT, len(arr))
    arrRight =  getTitik(leftmost, rightmost, arr, RIGHT, len(arr))

    arrLeft.sort(key=lambda k: [k[0], k[1]])
    arrRight.sort(key=lambda k: [k[0], k[1]], reverse=True)

    return [leftmost]+arrLeft+[rightmost]+arrRight

def drawPoints():
# Menggambar titik dan garis hasil pemrosesan quickHull
    absis = []
    ordinat = []
    arrPolygon = convexHull
    arrPolygon = makePolygon(arrPolygon)
    for i in range(len(arrPolygon)):
        point = arrPolygon[i]
        absis.append(point[0])
        ordinat.append(point[1])

    point = arrPolygon[0]
    absis.append(point[0])
    ordinat.append(point[1])
    plt.plot(absis,ordinat,marker = 'x')

main()
```

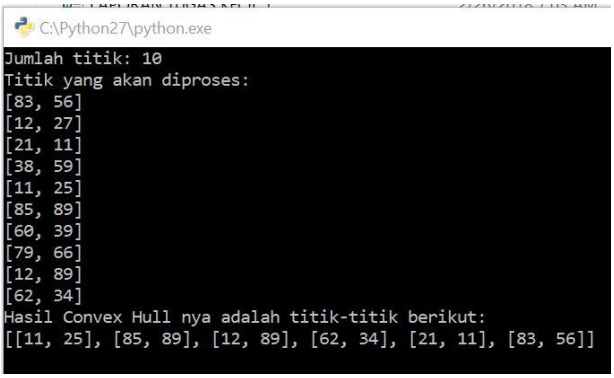
D. Hasil Input dan Output

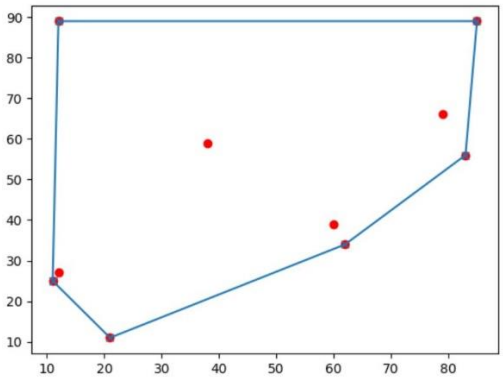
1. Jumlah Titik = 5



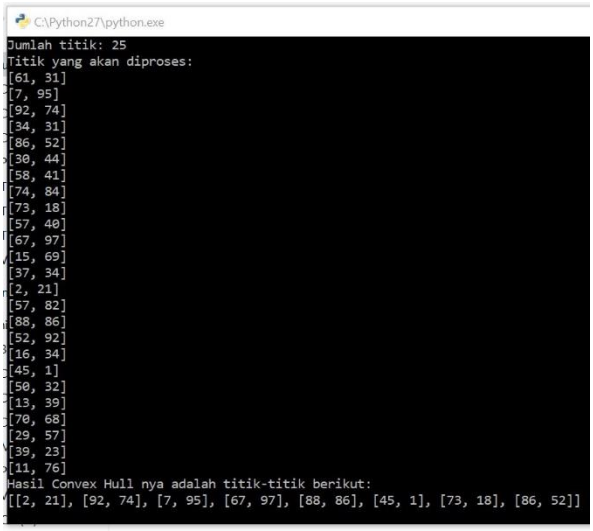


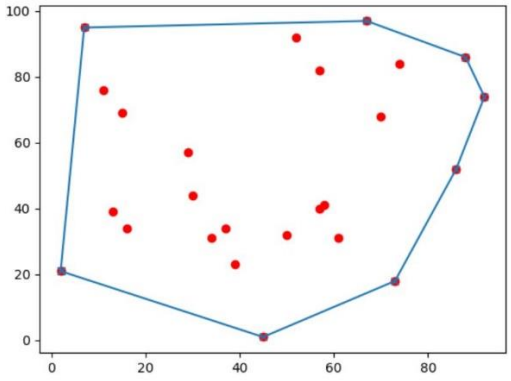
2. Jumlah Titik = 10



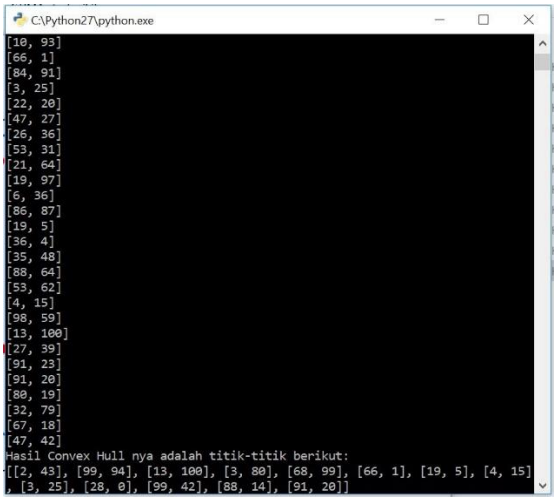


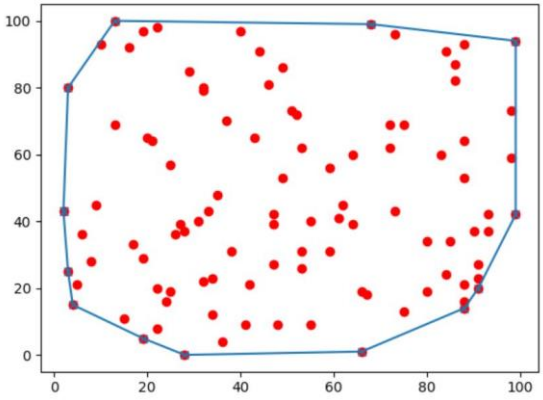
3. Jumlah Titik = 25





4. Jumlah Titik = 100





5. Tabel Laporan

Poin	Ya	Tidak
1. Program berhasil dikompilasi	√	
2. Program berhasil *running*	√	
3. Program dapat menerima input dan mengeluarkan output.	√	
4. Luaran sudah benar untuk semua *n*	√	