**Penyelesaian Persoalan Convex Hull dengan Divide and Conquer (Quick Hull)**

Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2017/2018

Oleh:

Hafizh Budiman 13516137



**TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2018**

**BAB I**

**PSEUDO CODE DAN KOMPLEKSITAS ALGORITMA**

Kompleksitas rata-rata = O(n log n)

Kompleksitas worst case = O(n^2)

**PSEUDO CODE:**

Using time

Using numpy

Using matplotlib.pyplot

function divide\_conquer (input: list\_of\_points)

// initialize min x and max x

initialize min x to infinity

initialize max x to 0

initialize min y to 0

initialize max y to 0

// get the leftmost and rightmost point

// in list of randomly generated points

for (x,y) in list\_of\_points:

if x is less than min x then

min x = x, min y = y

if x is more than max x then

max x = x, min y = y

set min equals to [min x, min y]

set max equals to [max x, max y]

// initial division

set hull\_points equals to quickhull(list\_of\_points, min, max)

set hull\_points equals to (hull\_points + quickhull(list\_of\_points, max, min))

returns hull\_points

// function to sort and do the quick hull algorithm

function quickhull (input: list\_of\_points, min, max)

//get all the points which situated in the left of the line

set left\_points equals to get\_left\_points(min, max, list\_of\_points)

//set the furthest point from current line as a new hull point

set hull\_point equals to max\_distance\_point(min, max, left\_points)

// return the point max as hull point

// if no points left in the left part of the line

if length of list hull\_point equals to 0 then

returns max

# divide recursively

set hullpts equals to quickhull(left\_points, min, hull\_point)

set hullpts equals to (hullpts + quickhull(left\_points, hull\_point, max))

returns hullpts

// function to get all points located at the left part

// of the currently checked line that joins p1 and p2

function get\_left\_points (input: p1, p2, points)

initialize pts as an empty array

for pt in points do

set val equals to ((p2.y-p1.y)\*(pt.x-p2.x) - (p2.x-p1.x)\*(pt.y-p2.y))

if (val not equal to 0) and (val less than 0):

insert pt to array pts

returns array pts

// returns the furthest point from the line joining p1 and p2

function max\_distance\_point (input: p1, p2, points)

initialize max\_dist as 0

initialize empty array max\_point

for point in points do

if (point.x not equals to p1.x or point.y not equals to p1.y) and (point.y not equals to p2.y or point.x not equals to p2.x):

set dist equals to line\_distance(p1, p2, point)

if dist more than max\_dist:

set max\_dist equals to dist

set max\_point equals to point

returns array max\_point

// function to calculate distance between a line joining p1 p2 and a point

function line\_distance (input p1, p2, pt)

set x1, y1 equals to p1

set x2, y2 equals to p2

set x0, y0 equals to pt

set top equals to absolute of ((y2 - y1) \* x0 - (x2 - x1) \* y0 + x2 \* y1 - y2 \* x1)

set bottom equals to ((y2 - y1)\*\*2 + (x2 - x1) \*\* 2) \*\* 0.5

return result of top divided by bottom

// function to print convex hull result and draw it using matplotlib.pyplot

function draw (input : points)

if length of points is less than 3 :

print "convex hull cant be created"

exit from function

start timer

set array quick\_hull equals to result of function divide\_conquer(points)

print elements inside array quick\_hull

stop timer, print time result

draw elements inside array quick\_hull using matplotlib.myplot

exit from function

// MAIN FUNCTION

main function ()

input value of n

initialize empty array points

random n points, append to array points

print elements inside array points

scatter elements inside array points using matplotlib.pyplot

draw(points)

show using matplotlib.pyplot

**BAB 2**

**KODE PROGRAM**

# Implementing Quick Hull algorithm to find convex hull

# Hafizh Budiman, February 22nd 2018

# Informatics Engineering, Bandung Institute of Technology, 2018

import time

import numpy as np

import matplotlib.pyplot as plt

def divide\_conquer(points):

# initialize hull with leftmost and rightmost point

# get the x min, and x max from the randomly generated points

min\_x = float('inf')

max\_x = 0

min\_y = 0

max\_y = 0

for x,y in points:

if x < min\_x:

min\_x = x

min\_y = y

if x > max\_x:

max\_x = x

max\_y = y

min, max = [min\_x,min\_y], [max\_x,max\_y]

# initial division

hullpts = quickhull(points, min, max)

hullpts = hullpts + quickhull(points, max, min)

return hullpts

'''

Does the sorting for the quick hull sorting algorithm

'''

def quickhull(points, min, max):

# get all points which situated in the

# left part of the newly formed triangle

left\_points = get\_left\_points(min, max, points)

# set the furthest point from the line as a new hull point

hull\_point = max\_distance\_point(min, max, left\_points)

if len(hull\_point) < 1: # return the point max as hull points

return [max]

# divide recursively

hullPts = quickhull(left\_points, min, hull\_point)

hullPts = hullPts + quickhull(left\_points, hull\_point, max)

return hullPts

'''

Returns all points that a LEFT of a line

that joins the point p1 and p2.

'''

def get\_left\_points(p1, p2, points):

pts = []

for pt in points:

val = ((p2[1]-p1[1])\*(pt[0]-p2[0]) - (p2[0]-p1[0])\*(pt[1]-p2[1]))

if (val != 0) and (val < 0):

pts.append(pt)

return pts

'''

Returns the furthest point from

a line joining the p1 and p2.

'''

def max\_distance\_point(p1, p2, points):

max\_dist = 0

max\_point = []

for point in points:

if (point[0]!=p1[0] or point[1]!=p1[1]) and (point[1]!=p2[1] or point[0]!=p2[0]):

dist = line\_distance(p1, p2, point)

if dist > max\_dist:

max\_dist = dist

max\_point = point

return max\_point

'''

Returns a value proportional to the distance

between the point pt and the line joining p1 and p2.

'''

def line\_distance(p1, p2, pts): # pt is the point

x1, y1 = p1

x2, y2 = p2

x0, y0 = pts

top = abs((y2 - y1) \* x0 - (x2 - x1) \* y0 + x2 \* y1 - y2 \* x1)

bottom = ((y2 - y1)\*\*2 + (x2 - x1) \*\* 2) \*\* 0.5

result = top / bottom

return result

'''

Prints the hull results and draw it

'''

def draw(points):

if (len(points) < 3):

return

start\_time = time.time() # start timer

quick\_hull = divide\_conquer(points) # call divide and conquer function

# print convex hull points result

print "\nQuick hull result:"

for x in quick\_hull: print x

# print timestamp

print("\nProccess done in: ")

print("%s seconds" % (time.time() - start\_time))

# draw convex hull result using matplotlib.pyplot

n = len(quick\_hull)

for i in range(n):

plt.plot([quick\_hull[i][0], quick\_hull[(i+1)%n][0]], [quick\_hull[i][1],quick\_hull[(i+1)%n][1]],'k-',lw=2)

plt.pause(0.08)

return

def main():

# Entering number of points

n = input("Enter the value for n: ")

points = np.random.randint(100,size=(n,2))

points[points[:0].argsort()] # sorting points

print ("Randomly Generated Points: \n")

for x in points:

print x

plt.scatter(x[0],x[1])

draw(points.tolist())

plt.show()

if \_\_name\_\_=="\_\_main\_\_":

main()

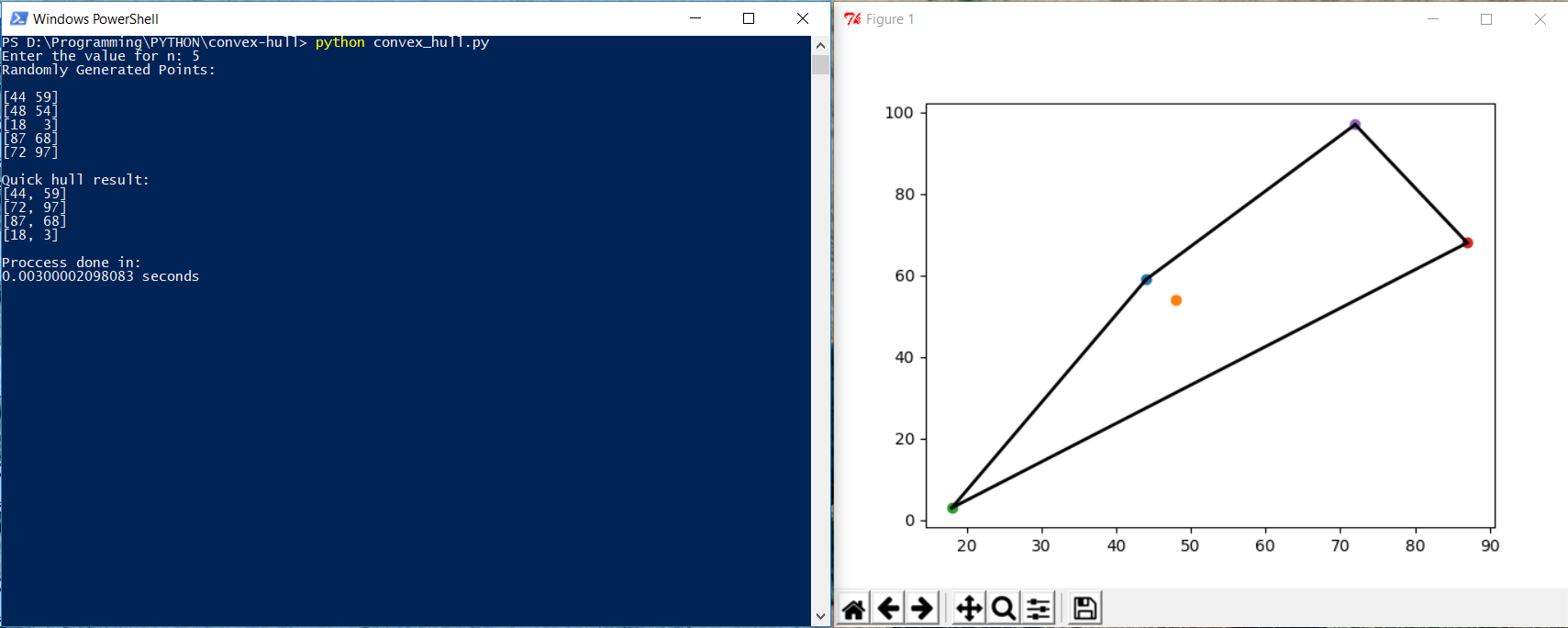
**BAB 3**

**INPUT/OUTPUT PROGRAM**

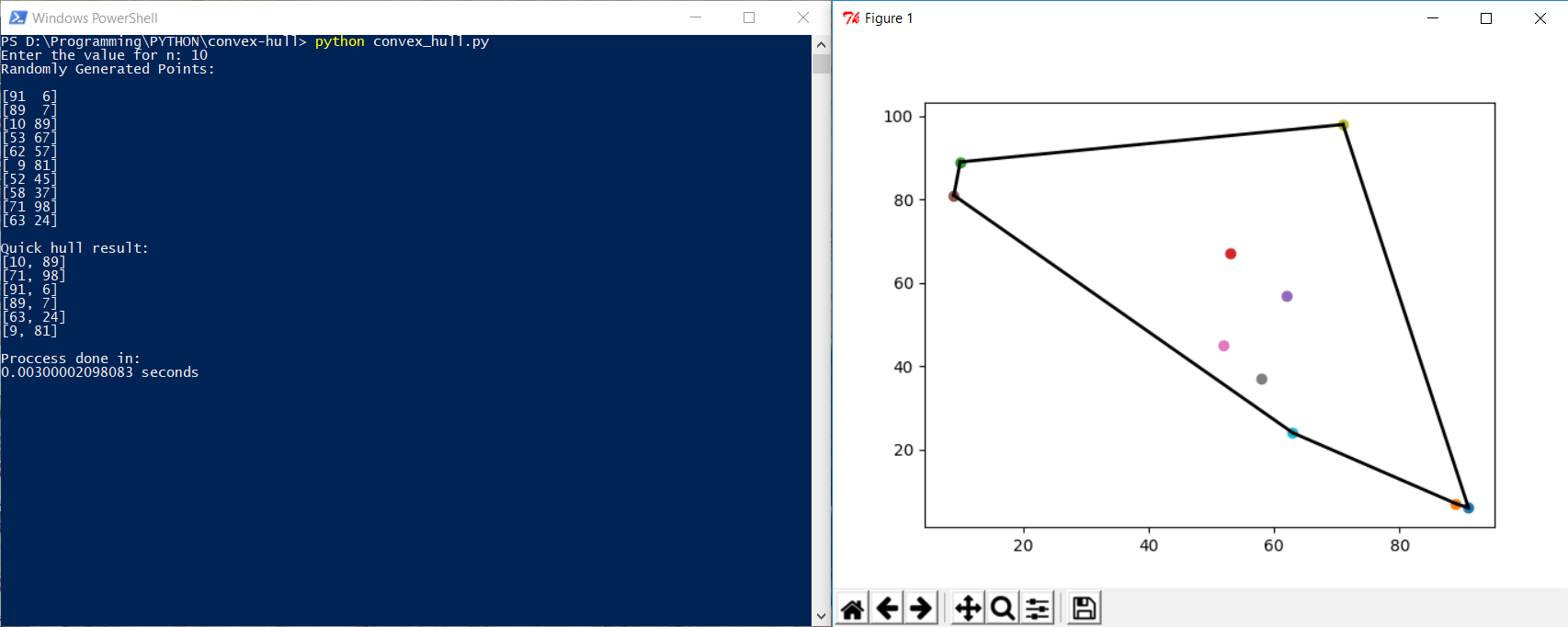
Spesifikasi komputer yang digunakan :



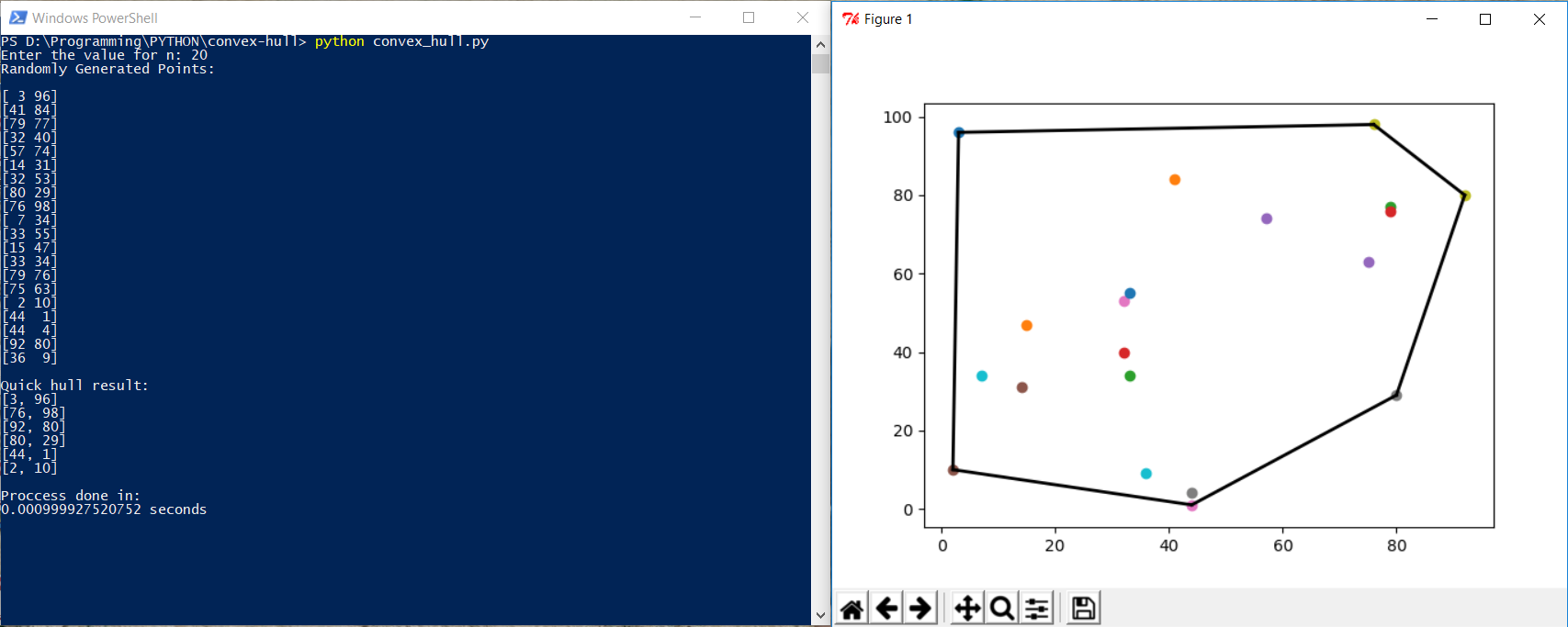
Input/output untuk n = 5:



Input/output untuk n = 10:



Input/output untuk n = 20:



Cek List Asisten

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | ✓ |  |
| 2. Program berhasil running | ✓ |  |
| 3. Program dapat menerima input dan menuliskan output. | ✓ |  |
| 4. Luaran sudah benar untuk semua n | ✓ |  |