

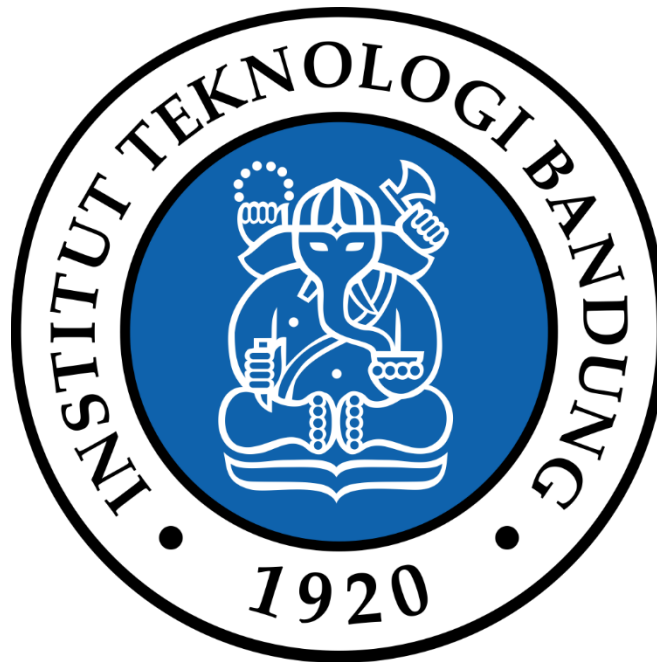
# **Penyelesaian Persoalan Convex Hull dengan Divide and Conquer (Quick Hull)**

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 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
```

```
set array quick_hull equals to result of function divide_conquer(points)
```

```
print elements inside array quick_hull
```

```
draw elements inside array quick_hull using matplotlib.pyplot
```

```
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 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 are 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

    quick_hull = divide_conquer(points)

    # print convex hull points result
    print "\nQuick hull result:"
    for x in quick_hull: print x

    # 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()

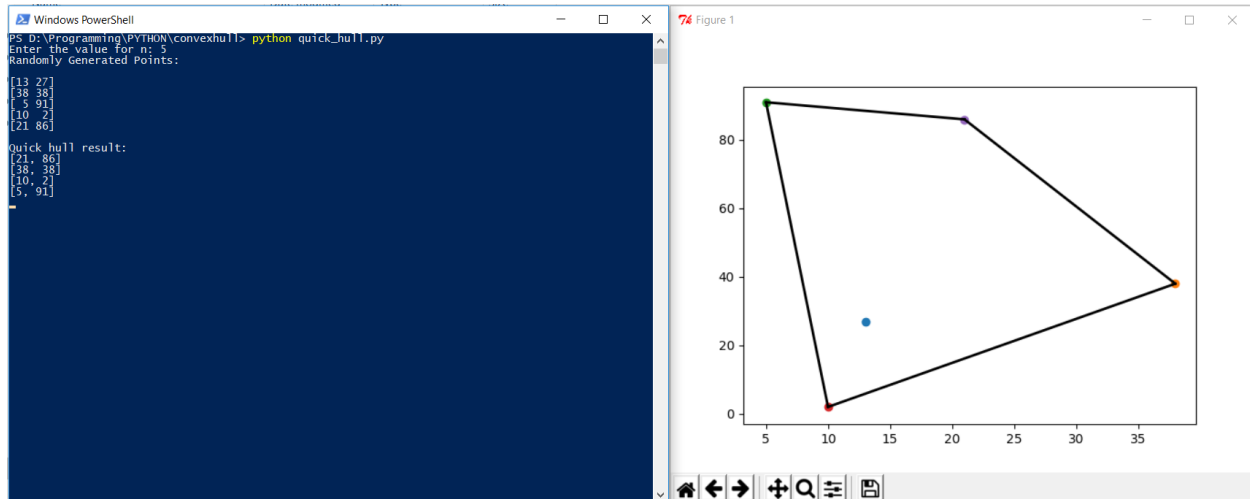
```



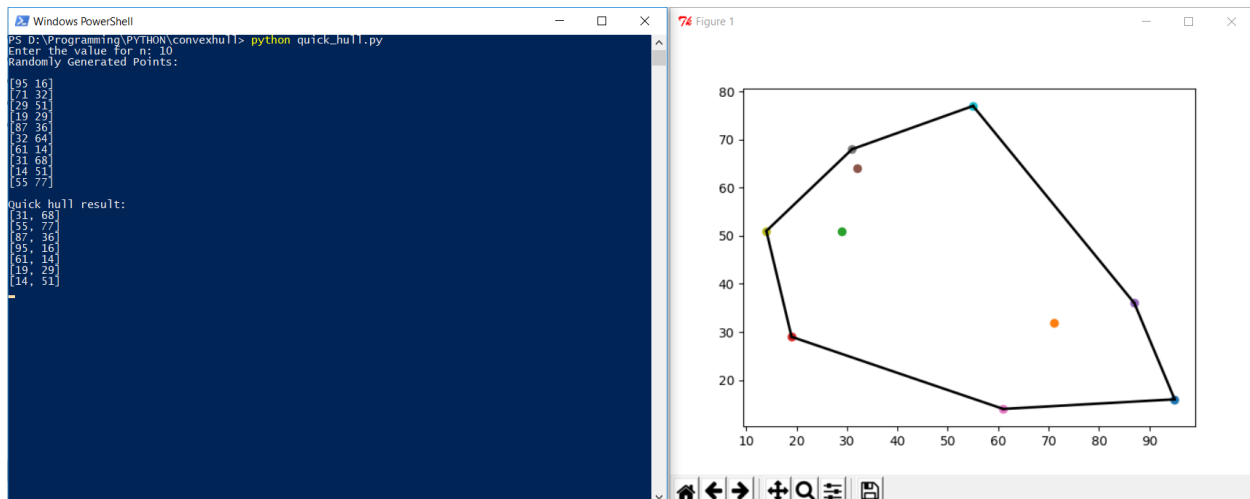
# BAB 3

## INPUT/OUTPUT PROGRAM

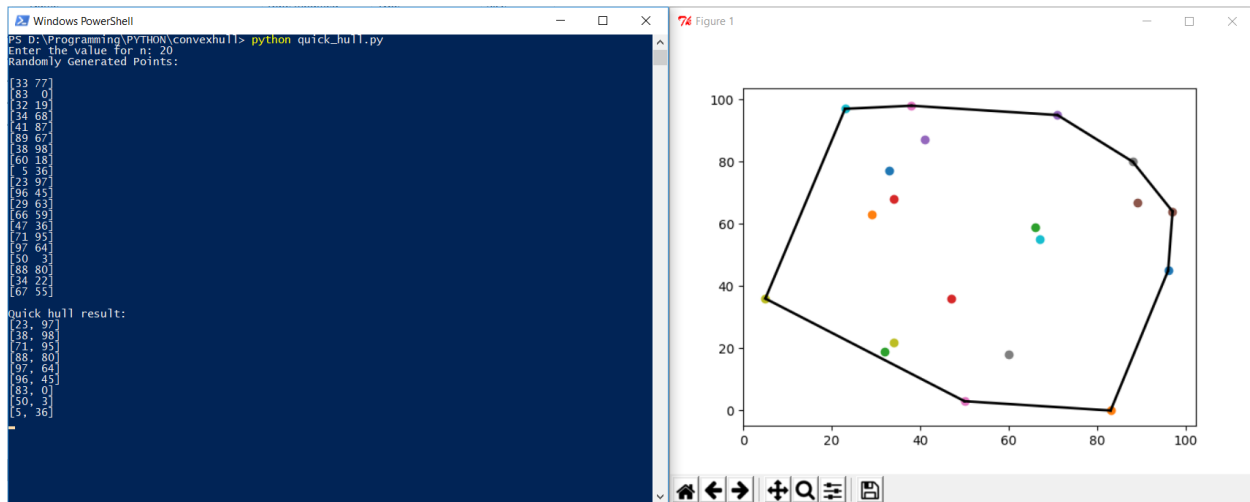
Input n = 5:



Input n = 10:



Input 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	✓	