Report - Joseph Koetting

1. Code:

```
class ConvexHullSolverThread(QThread):

    def __init__( self, unsorted_points, demo):
        self.points = unsorted_points
        self.pause = demo
        QThread.__init__(self)

    def __del__(self):
        self.wait()

    show_hull = pyqtSignal(list,tuple)
    display_text = pyqtSignal(str)

# some additional thread signals you can implement and use for debugging, if you like
    show_tangent = pyqtSignal(list,tuple)
    erase_hull = pyqtSignal(list)
    erase_tangent = pyqtSignal(list)

    ############################################
    # split_list
    # Split the list into lower and upper lists and return the new lists
    #
    # Time Complexity: O(n)
    # Space Complexity: O(n)
    ##########################################
    def split_list(self, a_list):
        half = len(a_list) // 2
        return a_list[:half], a_list[half:]

    ###########################################
    # divide_and_conquer
    # Splits the list into 2 recursively until their are 6 or less elements in
    # each list. Divide the list, and create shape objects with the elements.
    # Append the shape together, then recursively append each other shape
    # on each return call.
    #
    # Time Complexity: O(log n)
    # Space Complexity: O(less than 7) I create one or two arrays to store 7
    # or less points
    ###########################################
```

```
def divide_and_conquer(self, points):
    if len(points) > 6:
        # Recursively divide array in half
        A, B = self.split_list(points)
        shape_A = self.divide_and_conquer(A)
        shape_B = self.divide_and_conquer(B)
    else:
        # Create shape objects and append items
        firstArray = []
        secondArray = []
        if len(points) > 3:
            for i in range(len(points)):
                if i < 3:
                    firstArray.append(points[i])
                else:
                    secondArray.append(points[i])

            shape_1 = Shape(self, firstArray)
            shape_2 = Shape(self, secondArray)

            shape_1.append_array(shape_2)

        else:
            for i in range(len(points)):
                firstArray.append(points[i])
            shape_1 = Shape(self, firstArray)
        return shape_1

    # Recursively return bigger and bigger shapes
    shape_A.append_array(shape_B)
    return shape_A

def run( self):
    assert( type(self.points) == list and type(self.points[0]) == QPointF )

    n = len(self.points)
    print( 'Computing Hull for set of {} points'.format(n) )

    t1 = time.time()

    # SORT THE POINTS BY INCREASING X-VALUE
    self.points.sort(key = lambda p: p.x())

    t2 = time.time()
    print('Time Elapsed (Sorting): {:3.3f} sec'.format(t2-t1))
```

```
                t3 = time.time()

                # COMPUTE THE CONVEX HULL USING DIVIDE AND CONQUER
                circle = self.divide_and_conquer(self.points)

                t4 = time.time()

                # DISPLAY HULL
                circle.show_shape()

                # send a signal to the GUI thread with the time used to compute the hull
                self.display_text.emit('Time Elapsed (Convex Hull): {:3.3f} sec'.format(t4-t3))
                print('Time Elapsed (Convex Hull): {:3.3f} sec'.format(t4-t3))
```

class Shape:

```
    def __init__(self, hull, points):
        self.hull = hull
        self.points = points
        self.size = len(points)
        self.top_index = 0
        self.make_clockwise()


    def show_shape(self):
        polygon = [QLineF(self.points[i], self.points[(i + 1) % len(self.points)]) for i in range(len(self.points))]
        self.hull.show_hull.emit(polygon, (255, 0, 0))

    def show_line(self, points):
        polygon = [QLineF(points[i], points[(i + 1) % len(points)]) for i in range(len(points))]
        self.hull.show_hull.emit(polygon, (0, 255, 0))

    def show_line_red(self, points):
        polygon = [QLineF(points[i], points[(i + 1) % len(points)]) for i in range(len(points))]
        self.hull.show_hull.emit(polygon, (0, 0, 255))

    ##########################################
    # make_clockwise
    # Sorts the array in a clockwise fashion
    #
    # Time Complexity: 0(1) Function contains easy calculations and if statements
    # Space Complexity: 0(1) simple assignments
    ##########################################
```

```
def make_clockwise(self):
    if self.size == 1:
        self.top_index = 0

    elif self.size == 2:
        self.top_index = 1

    # Point with greater slope is the next item in the array to keep it clockwise
    elif self.size == 3:
        if self.slope(self.points[0].x(),
                      self.points[0].y(),
                      self.points[1].x(),
                      self.points[1].y()) < \
                self.slope(self.points[0].x(),
                           self.points[0].y(),
                           self.points[2].x(),
                           self.points[2].y()):
            self.points[1], self.points[2] = self.points[2], self.points[1]
            self.top_index = 1
        else:
            self.top_index = 2

##########################################
# index_big
# Returns the right most (x) number in an array
#
# Time Complexity: O(n) loops through array
# Space Complexity: O(1) assigns a number to variable big
##########################################
def index_big(self, points):
    big = 0
    for i in range(len(points)):
        if points[i].x() > points[big].x():
            big = i
    return big

##########################################
# slope
# Returns the slope of two points
#
# Time Complexity: O(1) does one calculation
# Space Complexity: 0(1) assigns a number to variable m
##########################################
def slope(self, x1, y1, x2, y2):
     m = (y2 - y1) / (x2 - x1)
     return m
    ##########################################
    # get_upper_common_tangent
    # Returns the upper common tangent of two shapes
    #
    # Time Complexity: O (n + m) methods that this function calls loops through 2 arrays
    # Space Complexity: 0(2) stores two variables
    ##########################################
    def get_upper_common_tangent(self, lhs, rhs):

        top_left_index = lhs.top_index
        top_right_index = 0

        # While the common tangent is not found
        while True:

            # Find potential right and left index
                        temp_top_right_index  =  self.get_top_right(lhs,  rhs,  top_left_index,
top_right_index)
                         temp_top_left_index  =  self.get_top_left(lhs,  rhs,  top_left_index,
temp_top_right_index)

             # If same as original indexes, the common tangent has been found -> return new
values
                  if  (temp_top_left_index  ==  top_left_index  and  temp_top_right_index  ==
top_right_index):
                return top_left_index, top_right_index

            # otherwise recalculate the index using newly found values
            top_right_index = temp_top_right_index
            top_left_index = temp_top_left_index


    ##########################################
    # get_top_right
    # Return the top right index with highest slope
    #
    # Time Complexity: O(n) loops through right shape
    # Space Complexity: O(1) assigns slopes to highest slope variable
    ##########################################
    def get_top_right(self, lhs, rhs, index_left, index_right):

        highest_slope = 0

        # loop throught the right shape and determine index with greatest slope
        for i in range(rhs.size + 1):
            '''
            points_temp = []
            points_temp.append( QPointF(lhs.points[index_left].x(), lhs.points[index_left].y()))
            points_temp.append( QPointF(rhs.points[(index_right + i) % rhs.size].x(),
rhs.points[(index_right + i) % rhs.size].y()))
            self.show_line_red(points_temp)
            '''

            slope = self.slope(lhs.points[index_left].x(),
                               lhs.points[index_left].y(),
                               rhs.points[(i + index_right) % rhs.size].x(),
                               rhs.points[(i + index_right) % rhs.size].y())

            if highest_slope == 0:
                highest_slope = slope
                continue

            # set highest slope to any other higher slopes
            if slope > highest_slope:
                highest_slope = slope
            # When highest slope is found return the index
            else:
                return index_right + i - 1
        print("nooooo")

    ########################################
    # get_top_left
    # Return the top left index with lowest slope
    #
    # Time Complexity: O(n) loops through left shape
    # Space Complexity: assigns slopes to lowest slope variable
    ########################################
    def get_top_left(self, lhs, rhs, index_left, index_right):

        lowest_slope = 0

        for i in range(lhs.size + 1):

            '''
            points_temp = []
            points_temp.append(QPointF(lhs.points[(index_left - i) % lhs.size].x(),
lhs.points[(index_left - i) % lhs.size].y()))
            points_temp.append(QPointF(rhs.points[index_right].x(),
rhs.points[index_right].y()))
            self.show_line(points_temp)
            '''

            slope = self.slope(lhs.points[(index_left - i) % lhs.size].x(),
                               lhs.points[(index_left - i) % lhs.size].y(),
                               rhs.points[index_right].x(),
                               rhs.points[index_right].y())

            if lowest_slope == 0:
                lowest_slope = slope
                continue

            # set lowest slope to any other lower slopes
            if slope < lowest_slope:
                lowest_slope = slope
            # When lowest slope is found return the index
            else:
                return index_left - i + 1

        print("nooooo")

    ##########################################
    # get_lower_common_tangent
    # Returns the upper common tangent of two shapes
    #
    # Time Complexity: O (n + m) methods that this function calls loops through 2 arrays
    # Space Complexity: 0(2) stores two variables
    ##########################################
    def get_lower_common_tangent(self, lhs, rhs):

        bottom_left_index = lhs.top_index
        bottom_right_index = 0

        # While the common tangent is not found
        while True:

            # Find potential right and left index
            temp_bottom_right_index = self.get_bottom_right(lhs, rhs, bottom_left_index,
bottom_right_index)
            temp_bottom_left_index = self.get_bottom_left(lhs, rhs, bottom_left_index,
temp_bottom_right_index)

            # If same as original indexes, the common tangent has been found -> return new
values
            if (temp_bottom_left_index == bottom_left_index and temp_bottom_right_index
== bottom_right_index):
                return bottom_left_index, bottom_right_index
            # otherwise recalculate the index using newly found values
            bottom_right_index = temp_bottom_right_index
            bottom_left_index = temp_bottom_left_index


    ########################################
    # get_bottom_right
    # Return the bottom right index with lowest slope
    #
    # Time Complexity: O(n) loops through right shape
    # Space Complexity: assigns slopes to lowest slope variable
    ########################################
    def get_bottom_right(self, lhs, rhs, index_left, index_right):

        lowest_slope = 0

        for i in range(rhs.size + 1):

            '''
            points_temp = []
                                            points_temp.append( QPointF(lhs.points[index_left %
lhs.size].x(),lhs.points[lhs.top_index % lhs.size].y()))
                     points_temp.append( QPointF( rhs.points[-i % rhs.size].x(), rhs.points[-i %
rhs.size].y()))
            self.show_line(points_temp)
            '''

            slope = self.slope(lhs.points[index_left].x(),
                               lhs.points[index_left].y(),
                               rhs.points[(index_right - i) % rhs.size].x(),
                               rhs.points[(index_right - i) % rhs.size].y())

            if lowest_slope == 0:
                lowest_slope = slope
                continue

            # set lowest slope to any other lower slopes
            if slope < lowest_slope:
                lowest_slope = slope
            # When lowest slope is found return the index
            else:
                return ((index_right - (i - 1)) % rhs.size)
        print("nooooo")

    ########################################
    # get_bottom_left
    # Return the bottom left index with lowest slope
    #
    # Time Complexity: O(n) loops through left shape
    # Space Complexity: 0(1) assigns slopes to highest slope variable
    ########################################
    def get_bottom_left(self, lhs, rhs, index_left, index_right):

        highest_slope = 0

        for i in range(lhs.size + 1):

            '''
            points_temp = []
                           points_temp.append(QPointF(lhs.points[(index_left + i) % lhs.size].x(),
lhs.points[(index_left+ i) % lhs.size].y()))
                                             points_temp.append(QPointF(rhs.points[index_right].x(),
rhs.points[index_right].y()))
            self.show_line(points_temp)
            '''

            slope = self.slope(lhs.points[(index_left + i) % lhs.size].x(),
                               lhs.points[(index_left + i) % lhs.size].y(),
                               rhs.points[index_right].x(),
                               rhs.points[index_right].y())

            if highest_slope == 0:
                highest_slope = slope
                continue

            # set higher slope to any other higher slopes
            if slope > highest_slope:
                highest_slope = slope
            # When highest slope is found return the index
            else:
                return (index_left + i - 1) % lhs.size
        print("nooooo")

    ########################################
    # append_array
    # Returns the combination of two shapes
    #
    # Time Complexity: O (n + m) method calls function get common tangent which has
    # as O(n + m). Also I loop through both arrays to add them to one bigger array
    # which is also O(n + m). simplified the algorithm is O(n + m)
    # Space Complexity: (n + m) I make an array that has n + m elements in it.
    ##########################################
    def append_array(self, rhs):

        # Get upper and lower common tangents
        index_top_left, index_top_right = self.get_upper_common_tangent(self, rhs)
        index_bottom_left, index_bottom_right = self.get_lower_common_tangent(self, rhs)

        '''
        print("top left: ", index_top_left)
        print("bottom left: ", index_bottom_left)
        print("top right: ", index_top_right)
        print("bottom right: ", index_bottom_right)
        '''

        # Create new array
        array = []

        # Add points from first array clockwise until first upper left index
        for i in range(index_top_left + 1):
            array.append(self.points[i])

        # Add points from second array clockwise from first upper right index until bottom right index
        j = index_top_right
        while j % rhs.size != (index_bottom_right) % rhs.size:
            array.append(rhs.points[j % rhs.size])
            j = j + 1
        array.append(rhs.points[j % rhs.size])

        # Add the rest of points clockwise from first array at the bottom left index till start of the array
        k = index_bottom_left
        while k % self.size > 0:
            array.append(self.points[k])
            k = k + 1

        # Refresh integral values of shape
        self.points = array
        self.top_index = self.index_big(self.points)
        self.size = len(array)
```

2. TIME COMPLEXITY

Whole Algorithm = O(n * log(n))

Master Theorem

A = subproblems = 2
B = Size of each subproblem = 2
D = runs in n ^ 1 time = 1

D ? Log b (A)

1 ? Log 2 (2)

1 == 1

So time complexity is

n ^ (d) * log (n) ==

n * log (n)

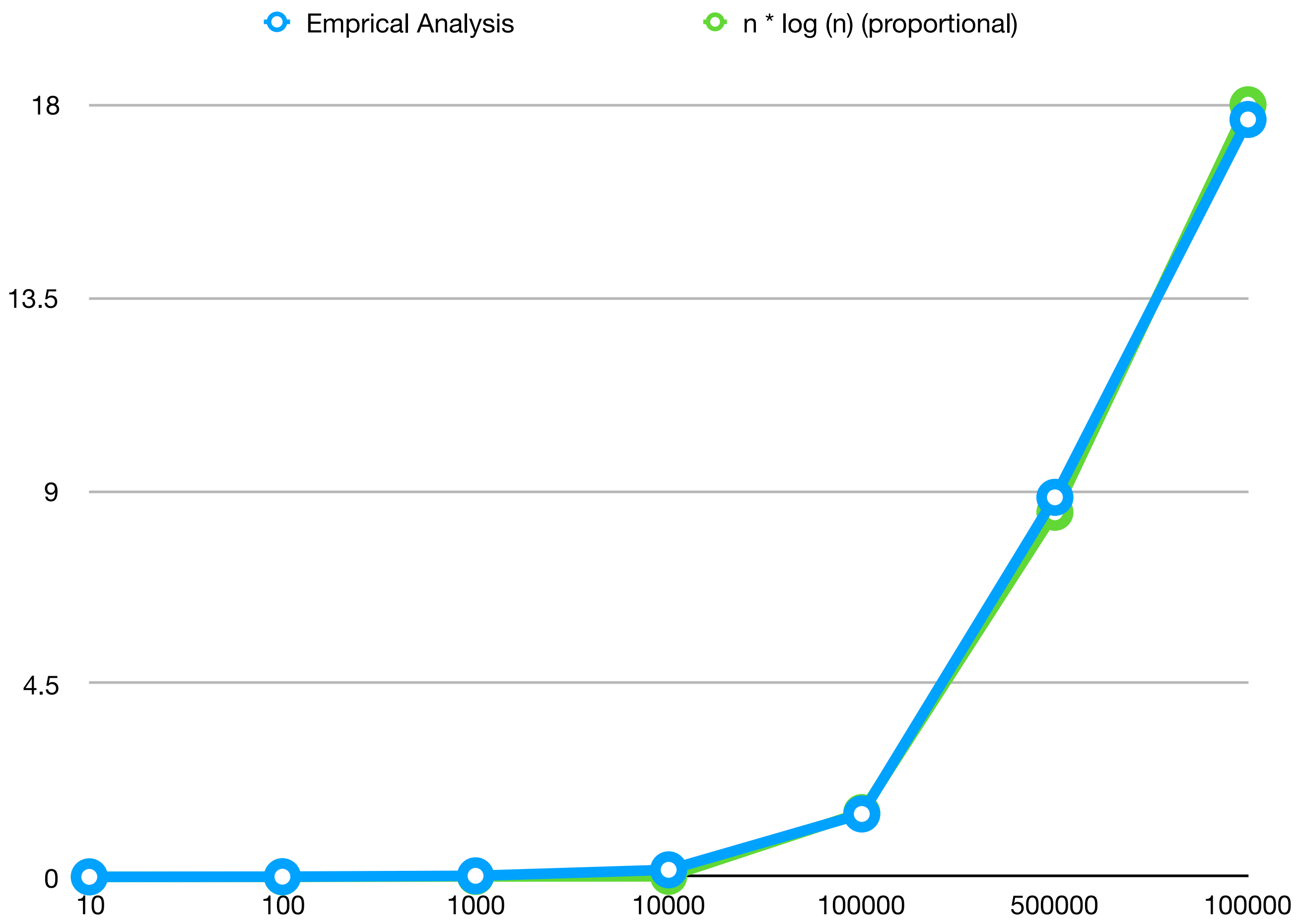
```
#################################
# divide_and_conquer
# Splits the list into 2 recursively until their are 6 or less elements in
# each list. Divide the list, and create shape objects with the elements.
# Append the shape together, then recursively append each other shape
# on each return call.
#
# Time Complexity: O( n * log (n) ) * see description above *
# Space Complexity: O(less than 7) I create one or two arrays to store 7
# or less points
#################################
```

```
#################################
# split_list
# Split the list into lower and upper lists and return the new lists
#
# Time Complexity: O(n)
# Space Complexity: O(n)
#################################

#########################################
# make_clockwise
# Sorts the array in a clockwise fashion
#
# Time Complexity: 0(1) Function contains easy calculations and if statements
# Space Complexity: 0(1) simple assignments
#########################################

#########################################
# index_big
# Returns the right most (x) number in an array
#
# Time Complexity: O(n) loops through array
# Space Complexity: O(1) assigns a number to variable big
#########################################


#########################################
# slope
# Returns the slope of two points
#
# Time Complexity: O(1) does one calculation
# Space Complexity: 0(1) assigns a number to variable m
#########################################
```

```
#########################################
# get_upper_common_tangent
# Returns the upper common tangent of two shapes
#
# Time Complexity: O (n + m) methods that this function calls loops through 2 arrays
# Space Complexity: 0(2) stores two variables
#########################################

#########################################
# get_top_right
# Return the top right index with highest slope
#
# Time Complexity: O(n) loops through right shape
# Space Complexity: O(1) assigns slopes to highest slope variable
#########################################

#########################################
# get_top_left
# Return the top left index with lowest slope
#
# Time Complexity: O(n) loops through left shape
# Space Complexity: assigns slopes to lowest slope variable
#########################################

#########################################
# get_lower_common_tangent
# Returns the upper common tangent of two shapes
#
# Time Complexity: O (n + m) methods that this function calls loops through 2 arrays
# Space Complexity: 0(2) stores two variables
#########################################
```

```
#########################################
# get_bottom_right
# Return the bottom right index with lowest slope
#
# Time Complexity: O(n) loops through right shape
# Space Complexity: assigns slopes to lowest slope variable
#########################################

#########################################
# get_bottom_left
# Return the bottom left index with lowest slope
#
# Time Complexity: O(n) loops through left shape
# Space Complexity: 0(1) assigns slopes to highest slope variable
#########################################

#########################################
# append_array
# Returns the combination of two shapes
#
# Time Complexity: O (n + m) method calls function get common tangent which has
# as O(n + m). Also I loop through both arrays to add them to one bigger array
# which is also O(n + m). simplified the algorithm is O(n + m)
# Space Complexity: (n + m) I make an array that has n + m elements in it.
#########################################
```

3. ANALYSIS OF DATA



	10	100	1000	10000	100000	500000	1000000
Trial: 1	0	0.004	0.024	0.170	1.433	8.879	18.888
Trial: 2	0	0.004	0.020	0.167	1.466	8.921	17.620
Trial: 3	0	0.004	0.028	0.161	1.496	8.799	17.625
Trial: 4	0	0.004	0.023	0.164	1.499	8.766	17.721
Trial: 5	0	0.004	0.021	0.158	1.436	8.879	17.435
Mean	0	0.004	0.024	0.164	1.466	8.848	17.657
n * log(n)	10	200	3000	40000	500000	2849485	6000000
Porportion n * log(n) * (1/333,333)	0	0	0.009	0.120	1.5	8.5	18

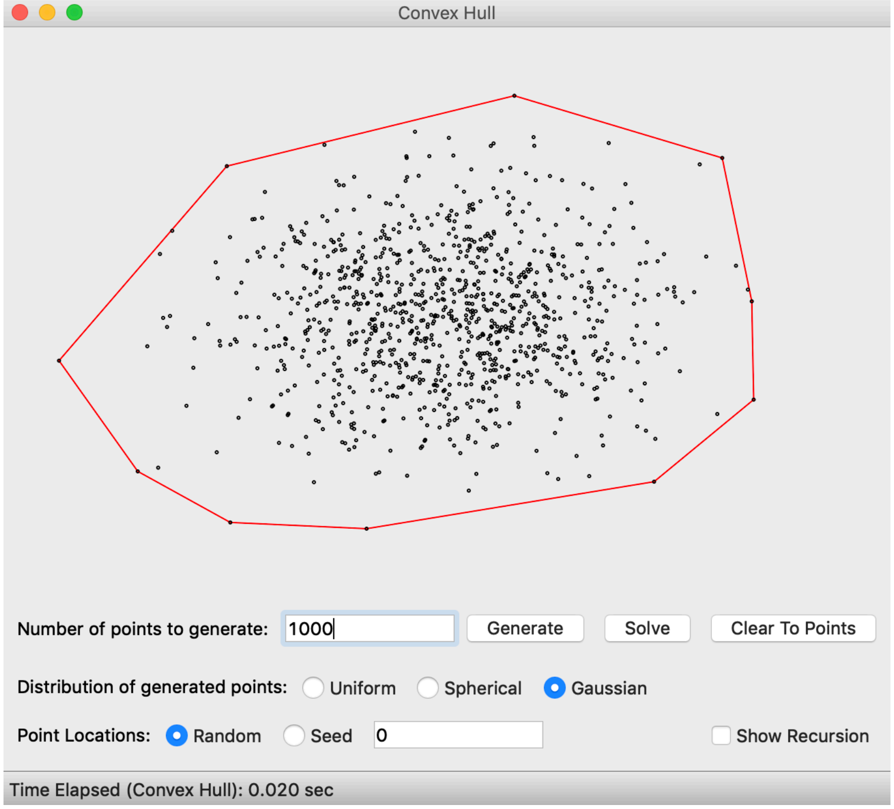
n * log(n) curve is a great fit with the experimental data obtained because it fits my experimental data almost perfectly. I divided 1,000,000 * log (1,000,000) by 18 to get a proportional constant of about 333,333. Then I multiplied each other n * log (n) by a 1/333,333 to get a proportion number for each set of points.

K = (1/333,333)

4. Further Analysis

The n log (n) curve is below the set of points on the left side and above the set of points on the right side. The curve is more distinguishably farther from the points the more it goes to the left side. The curve still works well because it changes from being below to above near the middle of the set of points. When looking at the graph above, almost no visible changes between the graphs are seen.

5. SCREENSHOTS

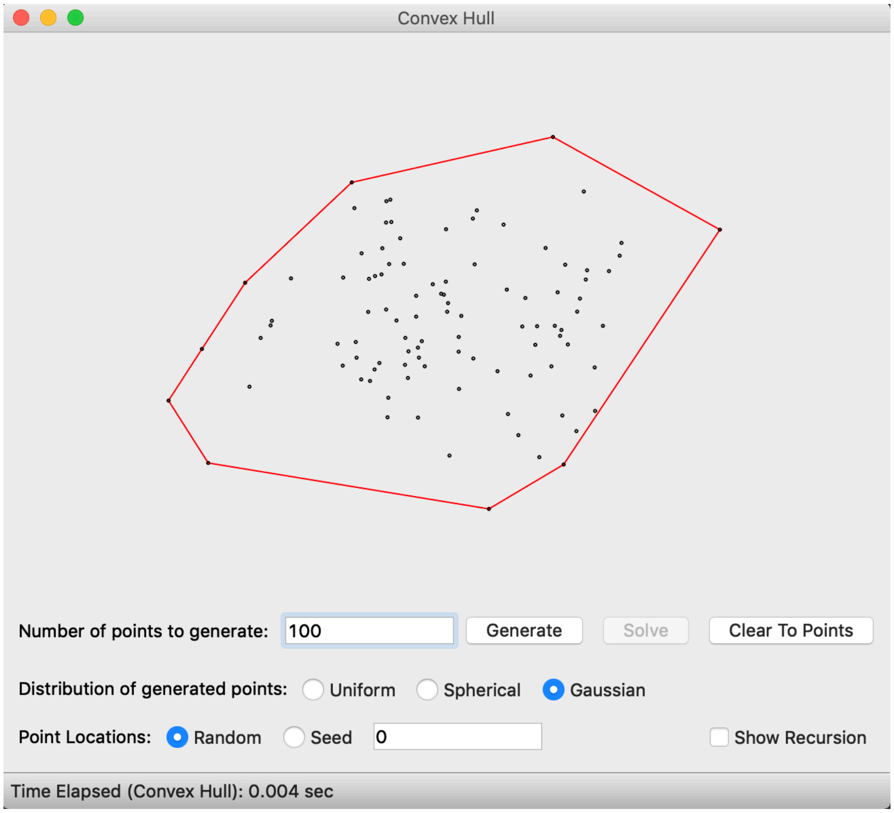


Number of points to generate: 1000

Distribution of generated points: ○ Uniform ○ Spherical ● Gaussian

Point Locations: ● Random ○ Seed 0 □ Show Recursion

Time Elapsed (Convex Hull): 0.020 sec



Number of points to generate: 100

Distribution of generated points: ○ Uniform ○ Spherical ● Gaussian

Point Locations: ● Random ○ Seed 0 □ Show Recursion

Time Elapsed (Convex Hull): 0.004 sec