Project 2

**Code/complexity**

A screenshot of a computer program

Description automatically generated

This was my classes created to store my points and my hull. Allowed for easy cleanup with garbage collection when reassigning CW and CCW nodes.

A screenshot of a computer program

Description automatically generated

getSlope happened in O(1) constant time.

**solveHull:**

* Time: o(nlog(n)
* Space: o(n)

The time is due to the function being called log(n) times, due to it being split in half. The ‘n’ part of the time comes from the combine function, which will be explained later.

As for space, it is o(n), due to having to create a node at the base case of len(points) = 1. Thus, worse case would be having a node stored for each point, and making space be O(n)

**combine:**

* Time: o(n)
* Space: o(1)

The time is due to the worst case of having to iterate through each node to find upper and lower tangent lines, especially if each point was needed in the final hull. The space is o(1), because very little data is stored when performing the while loops and are rewritten constantly within getUpper and getLower

A computer screen with text on it

Description automatically generated

**getTangents:**

* Time: o(n)
* Space: o(1)

Time is a result of the while loops having to iterate for all of n, and space is due to the variables always being updated in each loop.

A screen shot of a computer program

Description automatically generated

**compareCW/CCW**

* Time: o(1)
* Space: o(1)

Time and space are both O(1) due to each point only being recured once when it is called in the while loop in getTangentLine functions.

**getPoints/getPolygon**

* Time: O(n)
* Space: o(n)

Both have to recurse through a data structure of n length and return it, thus being space and time complexity of o(n)

A screen shot of a computer program

Description automatically generated

**Called functions in computeHull class**

**Plot/Analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 100 | 1000 | 10000 | 100000 | 500000 | 1000000 |
| 1 | 0 | .001 | .014 | .149 | 1.500 | 8.250 | 16.085 |
| 2 | .001 | .003 | .014 | .165 | 1.604 | 7.993 | 15.602 |
| 3 | .001 | .002 | .016 | .147 | 1.498 | 8.161 | 15.178 |
| 4 | .000 | .002 | .014 | .148 | 1.558 | 7.599 | 15.253 |
| 5 | .000 | .002 | .014 | .151 | 1.528 | 8.454 | 14.817 |
| Averages | .001 | .002 | .0144 | .152 | 1.548 | 8.091 | 15.387 |

A graph with a line

Description automatically generated

**g(n) = k = 2.46 \* 10-6 n \*log(n)**

**k = 2.46 \* 10-6**

My algorithm and data plot matches with knlog(n) time. When g(10, 100, 1000…) is plugged in, a value very close to my averages for each n is given, as n increases. This tells me that my theoretical analysis of being O(nlogn) time is very close to my empirical data and results. It seems that as N increases, time does increase at a rate that would be expected. This confirmed my hypothesis that my algorithm and functions are in the correct time complexity of O(nlog(n)). My x for my plot is in logratmic scale to help with the graph display my points even and spaced out.

**Screenshots**

100

A screenshot of a computer

Description automatically generated

1000

A screenshot of a computer

Description automatically generated