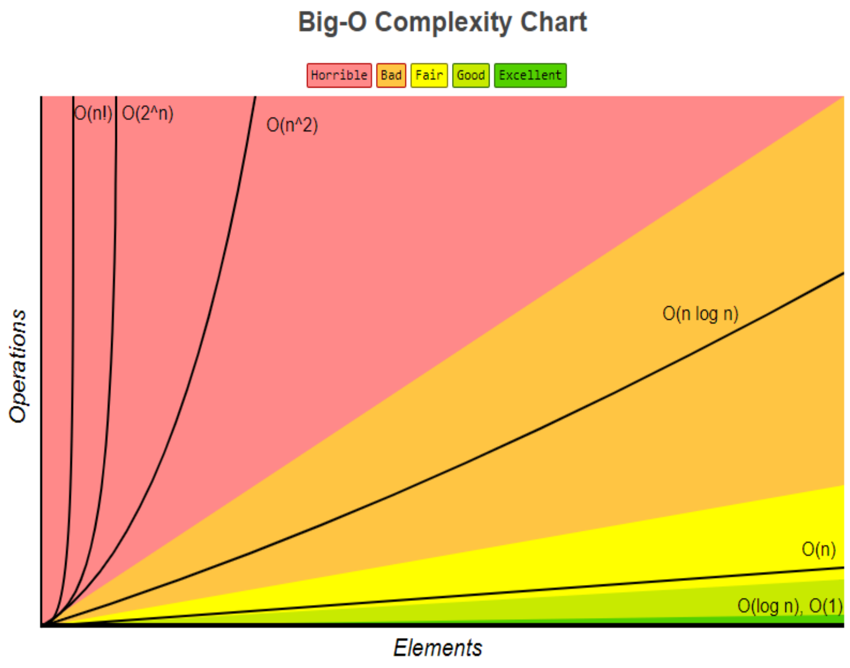
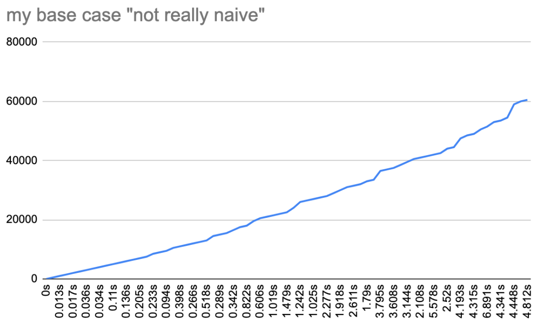
Convex Hull Algorithm: Benchmark and Analysis

Benchmarks

n	“naive”	dncs
0	0s	0.000s
500	0.002s	0.012s
1000	0.013s	0.031s
1500	0.009s	0.051s
2000	0.017s	0.067s
2500	0.044s	0.085s
3000	0.036s	0.119s
3500	0.02s	0.146s
4000	0.034s	0.297s
4500	0.12s	0.336s
5000	0.11s	0.289s
5500	0.024s	1.206s
6000	0.136s	0.453s
6500	0.074s	0.526s





I didn't expect my base case to be faster than divide and conquer.We can see that my convex hull base case algorithm graph resulting from my benchmark falls into the range between **O(nlogn) and O(n)**. I predict O(n) for the following reasons:

For that algorithm, I rely on the fact that we know that the points with the highest and lowest x, y values are part of the convex hull.

<u>Finding a maximum or minimum value is O(n)</u> as we need to iterate through the whole list in order to compare their values.

From the first hull point, (in this case the lowest with the highest y called hp1), we have a segment from that point perpendicular to the y axis where all other points of the list are above that line. The intersection point to that line is the point T

We iterate through all points and search for the point with the lowest (T, hp1, p) angle.

<u>Finding that second point (hp2) takes O(n - 1) ⇔ O(n).</u>

We append that point to our hull points list, pop it from the initial list and repeat the process on the left side, leading to the following 3 hull points (hp1, hp2, hp3)

Starting from the segment [hp1, hp2] we iterate through the list of points in clockwise order and find the point with the point forming the widest (hp1, hp2, p) angle which becomes a hull point.

By Yann Youbi
Prof. Dr. Daniels

For every hull segment we repeat that process until the next hull point found is hp3. From i=0 to incrementing by 1 until i=number of hull points the total number of iteration is **:**

$$\sum(n - i) + 1 + C \Leftrightarrow O(n)$$

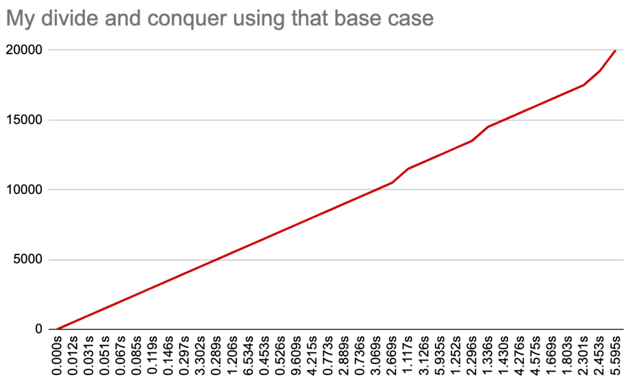
The best case is when the number of hull points of a convex hull is closer or equal to 0.

The worst case is when the number of hull points of a convex hull is closer or equal to n.

The **+ 1** is because we keep hp3 (the left adjacent hull point to the bottom hull point) on the list of points to iterate through for termination. Closure of the convex hull path from the right to the left into a circular path.

The constant **+ C** for all other computations such as the angle, comparisons, pops and appends.

From looking up graham scan and Jarvis's March my algorithm is more like the latter.



We can see that our divide and conquer graph shows a higher curve, due to a slower running time.

The graph falls into the **O(nlogn)** range.

We sort the elements based on their x values one time for a time of **O(nlogn).**

We keep on splitting each lists by 2 until it reaches a length

Then **O(k)** with k a constant for the computation of the split index using **O(1)** to get the length with the python len function. Splitting the list is just **O(1),**

Once that length is reached we hit our base case and compute the convex hull with **O(n)**

Our merge of each half involves clockwise sorting: **O(nlogn)**

For hp/2 convex hull and hp/2 convex hull to find the tangents we iterate k * n times to find one tangent (top/bot) **O(n)**

From i=0 to k incrementing by 1 until i=k

$$\sum(hp - 2) + C \Leftrightarrow O(hp)$$

With k the number of times we need to raise the tangent on the left and right side until we obtain the same y intercept value. HP the number of points on the convex hulls

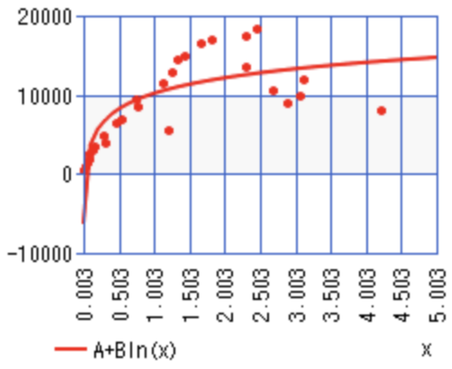
The constant **+ C** for all other computations such as the y value, comparisons and value declarations.

O(nlogn) being the biggest run time for dnc algorithm *(invariants on my code convex.py)*

By Yann Youbi
Prof. Dr. Daniels

function	value
mean of x	0.6071539545
mean of y	8,803.571429
correlation coefficient r	0.816890926
A	10,204.99547
B	2,808.617608

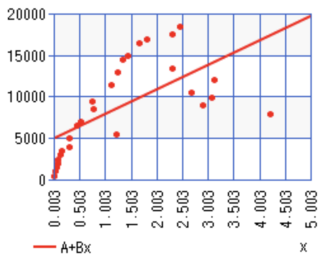
We can indeed see that there is a strong correlation between my divide and conquer and a logarithmic function such as n log n



function	value
mean of x	1.300571429
mean of y	8,803.571429
correlation coefficient r	0.6323414861
A	4,969.866191
B	2,947.708333

Guidelines for interpreting correlation coefficient r :

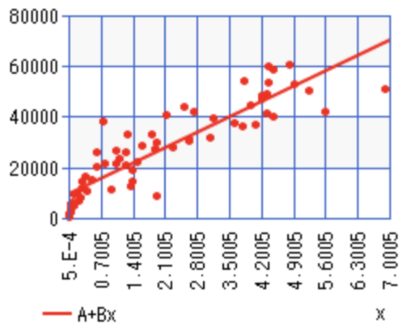
0.7＜|r|≦1	strong correlation
0.4＜|r|＜0.7	moderate correlation
0.2＜|r|＜0.4	weak correlation
0≦|r|＜0.2	no correlation

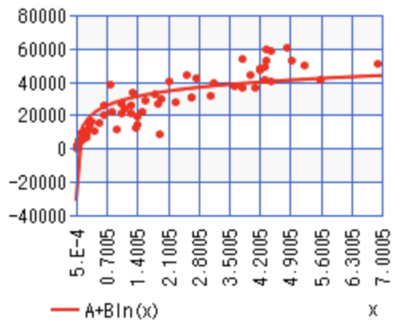


We can see from R squared that my base case is closer to a linear function than a logarithmic function.

function	value
mean of x	1.976106061
mean of y	26,765.15152
correlation coefficient r	0.8950780147
A	9,614.171122
B	8,679.180098

function	value
mean of x	0.7852294774
mean of y	26,765.15152
correlation coefficient r	0.8464246642
A	28,644.80486
B	7,774.25335

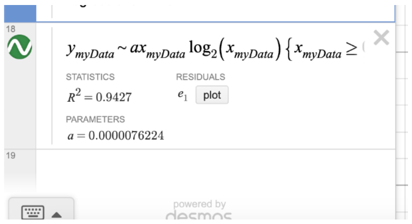




My benchmark and analysis of my base case and my divide and algorithm therefore matches with my r squared measures. We can conclude that our analysis was accurate.

By Yann Youbi
Prof. Dr. Daniels

We have a r squared of 0.9427 for divide and conquer for O(nlogn) which proves what we’ve mentioned before



$y_{myData} \sim a x_{myData} \log_2(x_{myData}) \{x_{myData} \geq$

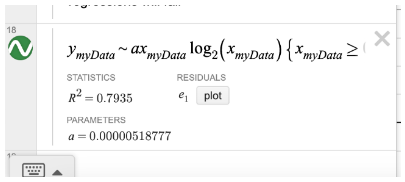
STATISTICS
$R^2 = 0.9427$

RESIDUALS
e_1 plot

PARAMETERS
$a = 0.0000076224$

log-linear

We have a r squared of 0.7935 for our base caseO(nlogn)which proves what we’ve mentioned before and a strong r squared for linear O(n) with 0.7786 as well.



$y_{myData} \sim a x_{myData} \log_2(x_{myData}) \{x_{myData} \geq$

STATISTICS
$R^2 = 0.7935$

RESIDUALS
e_1 plot

PARAMETERS
$a = 0.00000518777$

log-linear

By Yann Youbi
Prof. Dr. Daniels