Algorithmes et Complexité

TP1 Le plus grand rectangle

Chia-Ling Bragagnolo

The first file of code(tp1_n4_complexity) is on O(n3) complexity.

The function contains() :
use a for loop to verify if a point is contained inside a rectangle,
and we use an if condition there to detect if there is any point inside a rectangle,
by comparing the value of x and y position.
if yes, we return True ; if no, we return False
It's complexity is O(n)

The function surface() :
for calculating the surface of a rectangle.
It's complexity is O(1)

The function currentBasesFrom() :
using a lambda function to substitute x from each point given. Since we go through n points so it's complexity is O(n)

Finally the main function solveProblem() :
We substitute x position of the currentBase by using the x position of all the points since we are calculating the surface base on x = 0. We always compare the points on the right side of currentBase points and check first if there is any point in the middle of the rectangle with an if condition before comparing the size of surfaces.
It has 3 for nested loops, therefore the complexity is O(n3) + O(n) + O(n) + O(1) = O(n3).

The second file of code(tp1_n4_complexity) is also on O(n3) complexity, but there is a small change in the end.

The function contains() :
use a for loop to verify if a point is contained inside a rectangle,
and we use an if condition there to detect if there is any point inside a rectangle,
by comparing the value of x and y position.
if yes, we return True ; if no, we return False
It's complexity is O(n)

The function surface() :
for calculating the surface of a rectangle.
It's complexity is O(1)

The function currentBasesFrom() :
using a lambda function to substitute x from each point given. Since we go through n points so it's complexity is O(n)

Finally the main function solveProblem() :
We substitute x position of the currentBase by using the x position of all the points since we are calculating the surface base on x = 0. We always compare the points on the right side of currentBase points. However in this version we calculate the surface first and find the max surface then verify if this surface contains any points Therefore in this version the complexity is O(n3) + O(n) the worse and O(n3) the best case, even though in either case the Complexity is O(n3)

The Divide and conquer version, file of code name recursive_complexity is on O(n logn) complexity.

The function surface() calculates the surface or rectangle by passing pivot and a point. It has the complexity of O(1).

The function pickPivot() is for searching for the pivot in each iteration.
In our case, the pivot is the one with lowest y-value in each divided section.
Its complexity is O(n).

Split the list, by the function splitList(), to the right list and the left list by the giving point of pivot. Its complexity is O(n)

Lastly, using the function solveBranch() to find the lowest y-value (height) in each divided section. If there is no point within the section then the maximum height is considered as h, which is 20 in our example. The solveBranch() function and solveProblem() function inter-calls each other, forming a second degree recursivity.

The algorithm solves problems by dividing them into two subproblems of half the size in each recursion, and then combining the solution in linear time. if we apply master theorem to calculate the complexity :
$T(n) = 2*T(n/2)+O(n)$, where $a = 2$, $b = 2$, $d = 1$,
if $d = \log_b a = 1$, then $T(n) = O(n \log n)$