Answers

May 19, 2021

1 The Time Complexity of the Given Algorithm

For each black pixel in the hole we go over every “gray” pixel in the outer boundary. Therefore the time complexity is $O(n \cdot m)$.

The outer boundary cannot exceed $O(n)$ as every hole pixel in the inner boundary can lead to a maximum of 8 outer boundary pixels in 8-connectivity or 4 in 4-connectivity.

However in the worst case it might be that $m = O(n)$, as in a hole consisting of a streight line, in which case the inner boundary is the entire hole, so in conclusion the time complexity is $O\left(n^2\right)$.

2 Approximation in $O(n)$

We can go by “layers”:

In the first iteration we take the inner boundary of the hole i.e. taking the hole pixels which connect to the outer boundary.

This means traversing the pixels in the outer boundary, and for each one, checking 8 or 4 pixels connected to it, i.e. a constant number of pixels for each pixel in the outer boundary, i.e. $O(m)$ and as we stated in the first question, this equals $O(n)$.

Now, For each pixel in the inner boundary we apply the base algorithm while taking into account only the outer boundary pixels connected to it and the pixels in the current inner boundary which were already filled, i.e. a constant number of pixels for each pixel in the inner boundary, which means $O(\text{size of the current inner boundary})$.

Next we consider the current inner boundary, which we just filled, as the current outer boundary and keep going for the next inner boundary, continuing this way iteratively/recursively until the inner most boundary.

Each iteration we need to find the next inner boundary and apply the base algorithm on it. Each time it takes $O(\text{size of the current outer boundary} + \text{size of current inner boundary})$ and in total it takes

$O\left(\text{sum of the sizes of all outer boundaries + sum of the sizes of all inner boundaries}\right)$ which equals $O(n)$ as all the outer boundaries and inner boundaries together are twice the hole itself plus the initial outer boundary.

In conclusion we remain with $O(n)$ as we wanted.

Note: from my testing the best result is achieved when considering 4-connectivity in the stage of collecting an inner boundary, and either 4-connectivity or 8-connectivity in the stage of filling the pixels.

3 $O(n \log n)$ Bonus – try

Sort the hole pixels lexicographically – $O(n \log n)$.

Take $\lfloor \log n \rfloor$ hole pixels with equal distances between them starting from the first one – these will be the hole "representatives".

Taking these pixels in equal distances takes $O(\log n)$ because we just jump by $\left\lfloor \frac{n}{\log n} \right\rfloor$ indices and take modulo in the size of the sorted array every time we look for the next hole pixel, i.e. $O(1)$ for each of the $\lfloor \log n \rfloor$ representatives, i.e. $O(\log n)$ in total.

Apply the base algorithm just on these hole pixels. As seen in the first question, the time complexity of this step is $O(n \log n)$, since the boundary is $O(n)$ (discussed as well in question 1) and the current hole pixels are $O(\log n)$.

Now, apply the base algorithm on the rest of the hole pixels where the "boundary" are the representatives.

The pixels filled in this step are $O(n)$ and the "boundary" is $O(\log n)$ so again we remain with $O(n \log n)$.

Note: I also tried to take random $\lfloor \log n \rfloor$ from the hole pixels as the representatives, but the result differred significantly from the base solution.

4 $O(n \log n)$ Bonus – Another Try

"Boundary Line Space" Algorithm:

Sort the boundary and the hole pixels lexicographically – $O(n \log n)$.

Traverse the hole and the boundary pixels lexicographically: Take $\lfloor \log n \rfloor$ boundary pixels with equal distances between them starting from the first one and apply the base algorithm using the chosen $\lfloor \log n \rfloor$ boundary pixels.
Taking these pixels in equal distances takes $O(\log n)$ because we just jump by $\left\lfloor \frac{n}{\log n} \right\rfloor$ indices and take modulo in the size of the sorted array every time we look for the next boundary pixel, i.e. $O(1)$ for each of the $\lfloor \log n \rfloor$ representatives, i.e. $O(\log n)$ in total.

Apply the base algorithm where the boundary consists only of the representatives. As seen in the first question, the time complexity of this step is $O\left(n \log n\right)$, since the boundary is $O\left(\log n\right)$ and the hole is $O\left(n\right)$.

So in total we have $O\left(n \log n\right)$.

Note: I also tried to take different $\lfloor \log n \rfloor$ pixels on the boundary for each hole pixel by choosing the starting boundary pixel to be one pixel further from the last one, lexicographically, but the result differred significantly from the base solution. Furthermore I tried to take random $\lfloor \log n \rfloor$ from the boundary and use the same ones throughout the process, but again it wasn't good enough. These results can be seen in the tests class.