Hole Filling

Questions:

1. If there are m boundary pixels and $n$pixels inside the hole, what's the complexity of the algorithm that fills the hole, assuming that the hole and boundary were already found? Try to also express the complexity only in terms of n.
2. Describe an algorithm that approximates the result in $O(n)$ to a high degree of accuracy. As a bonus, implement the suggested algorithm in your library in addition to the algorithm described above.

Answers:

1. At each iteration the algorithm searches for the new boundary pixels by iterating the image. I will annotate s as the size of the given image.
at worst case each $p \in H$ will be discovered in a new iteration (which is impossible because there is only a single hole). so the highest number of iterations is $(n-m)$. So, at worst case the time complexity for extracting the boundary is $O\big(s(n-m)\big) \in O(s^2)$.
For each $u \in H$, $I(u) = \frac{\sum_{v \in B} w(u,v) \cdot I(v)}{\sum_{v \in B} w(u,v)}$. the $w(u,v)$ function has a constant running time, and there is a constant number of neighbors $v \in B$ for u. so the complexity for $I(u)$ is $O(1)$. The calculation is done $(m+n)$ times so the time complexity will be $O\big((n-m)(m+n)\big) \in O(s^2)$. So the overall time complexity for the algorithm will be $O(\max(n^2-m^2),(n-m)S) \in O(s^2)$. When the number of iterations is relatively small the time complexity for the algorithm would be $O(n^2)$ but the accurate answer is $O(s^2)$.

2.
declare HashMap H.
for each $(x', y') = p \in boundary$:
    check if $H[x].y > y'$
        H[x]=y'
For each $u = (x, y)$ in Image:
    if u is a hole:
        calculate I(u)
    else:
        check if the key exist and if so declare $u = (x+1, H[x+1])$

it will be less accurate because the pixels inside the hole will be colored with less colored pixels surrounding them. Both the HashMap initialization and the holes iteration will take $O(n)$ time complexity.

another suggestion with $O(s)$ space complexity:
initialize boundary list
while boundary is not empty:
    for each $u \in boundary$:
        calculate I(u)
    for each $v \in neighbors(u)$:
        if $v \in H$:
            boundary.insert(v)