1. In the algorithm, I showed fuses as an int array. 1 means healthy fuse and whenever it becomes 0, it means that starting point of 0 is defective fuse.

Algorithm does binary search (it is decrease and conquer) and additionally cheks if finds 0, chek left index. If it is 1, this means we are at defective fuse. Returns id. Otherwise searchs for left sub-array. If it sees 1, directly searchs for right sub-array.

Best Case: O(1) Average Case: O(logn) Worst Case: O(logn)

2. The algorithm takes mxn int array and cheks all items one by one by calling helper function. Helper function cheks left, right, upper and lower indexes to see if the current index what we are looking for. So the helper function is O(1).

Best Case: O(1)

Average Case: O(mxn)

Worst Case: O(mxn)

3. The algorithm goes from start to end in any case. Finds possible highest range by comparing numbers in constant time. (Kadane's algorithm variant).

Best Case: $\Theta(n)$

Average Case: $\Theta(n)$

Worst Case: $\Theta(n)$

4. The algorithm use DFS to explore all paths from source to destination

Best Case: $O(V+E)$ → Graph is sparse like a tree

Average Case: $O(V!)$ → Depends on the graph structure but it is closer to $O(V!)$

Worst Case: $O(V!)$ → Graph is fully connected complete graph.

5. We need do look for every index because the array is not sorted. So the time complexity is $\Theta(n)$ always. It doesn't matter if we apply divide and conquer. Recurrence relation is $T(n) = 2T(n/2) + \Theta(1)$

Best Case: $\Theta(n)$

Average Case: $\Theta(n)$

Worst Case: $\Theta(n)$