$O(n^2)$ time complexity happens when there are two nested loops. The first loop iterates over n number of loops and the second loop which is the inner loops will run n times for the number of first loop.
$n*n=n^2$
Swaps are performed first as n-1 then n-1 and so forth for reverse case which is the worst outcome. The total number of swaps is n*(n-1)/2 and time complexity is $O(n^2)$

$O(n^3)$ time complexity happens with three nested loops. The first loops iterates over n number of loops, second loop runs n times over the first loop and third loop runs over n number of second loop. All loops are running n number of times.
$n*n*n=n^3$

O(n*logn) occurs when the dataset is divided in half based on some conditions and then works on the elements individually. Seen in recursive sorting. The input size keeps decreasing. This increases the speed.
Whenever the array gets divided in half it is represented using a logarithmic function log n and number of steps as log n + 1.
A single step operation to find the middle element O(1).
To merge the arrays which were divides it requires O(n) linear time.
The total time is O(n*logn)

O(1) space complexity means that the algorithm only uses the fixed amount of space in the memory.
O(n) indicates extra space is required to store the merged array
O(logn) as for each recursive call the algorithm needs to create extra space for stack frames.