Question 1:

The DFS solution to this problem has a O(V+E) time complexity. But why is not just O(V)? Yes we visit every vertex and every edge, but every edge just leads to another vertex, it's not an additional step. For example, if we have 2 vertex, with one edge between them, then we visit 2 vertex, period. We don't visit 3 things (2 vertex + edge). Give me an example of DAG where 'V+E' results in more visit than just 'V'

To strengthen my argument, the time complexity of a DFS on a binary tree is O(N), where N is number of nodes. No one says it's O(N+E).

Question 2:

The reason why the algorithm remains O(log*N*) even with the extra calls for the odd number case is because the number of extra calls is bounded by a constant. In the worst case, *N/2* is odd at each iteration, but this would only double the number of extra calls (the constant is 2). That is, at worst, there will be 2log*N* calls to complete the algorithm.

Question 3:

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

Time complexity: O(9(N*N)), For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)).

Space Complexity: O(N*N), To store the output array a matrix is needed.

Question 4:

Insertion Sort

$array$ = {6, 8, 9, 8, 3, 3, 12}

6 8 9 **8** 3 3 12

6 8 **8** 9 **3** 3 12

6 8 8 **3** 9 3 12

6 8 **3** 8 9 3 12

6 **3** 8 8 9 3 12

3 6 8 8 9 **3** 12

3 6 8 8 **3** 9 12

3 6 8 **3** 8 9 12

3 6 **3** 8 8 9 12

3 **3** 6 8 8 9 12

This a stable sorting algorithm as items maintained their relative position. When 3 & 3 was sorted , the left 3 stayed on the left side of the after the final sort result.

Quick Sort:

$array$ = {6, 8, 9, 8, 3, 3, 12}

Firstly select pivot -> Third index: 8

6 8 9 **8** 3 3 12

6 8 9 12 3 3 **8**

6 **3** 9 12 3 **8** 8

6 3 **3** 12 **9** 8 8

6 **3 3 8** 9 8 **12**

6 **3 3** 8 9 8 12

3 3 6 8 9 8 12

3 6 **3** 8 9 8 12

3 **3** 6 8 9 **8 12**

3 3 6 8 9 **12 8**

3 3 6 8 **8** 12 9

3 3 6 8 8 9 12

This is also a stable sorting algorithm. When we were swapping items, for the equal items the relative position was always maintained as we didn't perform any swap when items were equal.

Bubble Sort

$array$ = {6, 8, 9, 8, 3, 3, 12}

6 8 **9 8** 3 3 12

6 8 **8 9 3** 3 12

6 8 8 **3 9 3** 12

6 8 **8 3 3 9** 12

6 8 **3 8 3** 9 12

6 8 **3 3 8** 9 12

6 **3** 8 **3 8** 9 12

6 **3 3** 8 **8** 9 12

3 6 **3** 8 **8** 9 12

3 3 6 8 **8** 9 12

Bubble Sort is a stable sorting algorithm. We swap elements only when A is less than B. If A is equal to B, we do not swap them, hence relative order between equal elements will be maintained.

Question 5:

a. The primary distinction between brute force and exhaustive search is the fact that brute force is a method used when the issue is of finite size (pattern search). Exhaustive search, on the other hand, is a technique for solving large (permutational or combinational-related) problems. Furthermore, a thorough search takes less time than a forceful one.
The brute force algorithm is a method that programmers created to retrieve strings; it is also used to solve the eight-queen conundrum (a puzzle to place eight queens on eight by eight chessboard).
It's an intuitive approach, but to address the issue, many comparisons are required.
The exhaustive search is a kind of brute force search that is employed to address permutation and combination-related issues. The main objective is to look at every possibility for the best solution while satisfying the constraints. Additionally, issues with travelling salespeople and knapsacks can be resolved.
The non-uniform approach uses the Brute Force search technique. On the other hand, a uniform approach is an exhaustive search algorithm.
A technique for finding the string in the text is the brute force search algorithm. On the other hand, a thorough search algorithm looks for permutations and combinations' solutions.
The Brute Force method functions by finding matches among every letter in a string. The process of scrutinising each node of the flowchart until the restrictions are met, however, is followed by an exhaustive search.The Brute Force approach works well when there is less data and requires more timing. On the other hand, a comprehensive algorithm can be used in complicated situations.

b. Caesar cyphers are extremely susceptible to a "brute force" assault, in which the decoder merely tries every conceivable letter combination, because there are only 25 potential keys. Caesar cyphers are not seen to be a safe way to encrypt electronic communications since it could take some patience for a human to accomplish it, but computers nowadays can decipher the code in a matter of milliseconds.
Against brute-force assaults, AES is secure. A threat actor uses a brute-force assault when they try every key combination until they find the right one. Therefore, the key size used for AES encryption needs to be sufficiently big to prevent it from being cracked by contemporary computers, even taking Moore's law-based improvements in processor performance into account.
A 128-bit encryption key is much easier for brute-force attacks to guess than a 256-bit key, but since the latter takes so long to guess, even with a lot of computing power, it is unlikely to be a problem in the near future because a malicious actor would need to use quantum computing to produce the necessary brute force.
Explanation
One of the earliest and best-known cyphers is called the Caesar encryption after the Roman Emperor Julius Caesar. It is a straightforward "substitution cypher" in which each letter of the alphabet is replaced with a different letter by moving the entire alphabet by a predetermined number of letters (wrapping around to the beginning once you reach the end)
The United States government selected the symmetric block cypher known as the Advanced Encryption Standard (AES) to safeguard sensitive data.
To encrypt sensitive data, AES is used in hardware and software around the globe. For government computer security, cybersecurity, and the protection of electronic data, it is crucial.

c. When you refer to an algorithm as having a polynomial running time, what you really mean is that the method's running time is a polynomial in the input length. In general, the input length is the quantity of keystrokes needed to convey the input.
Consider what occurs when you add a new digit to the right end of your input to see why O(vn) is not polynomial in the input length. This only needs one additional keystroke. As a result, the input length increases by one. But take note that the input's value increases by a factor of 10.
So, in order to determine the answer, you must now do V10 = 3 times as many divisions. One extra character in the input length causes the effort to multiply by three. Clearly, this is exponential.
This is the reason why the naive solution to primality testing grows exponentially.