CSE321 Homework 3

Eren Çakar 1901042656

Question 1

a) To topologically order given graph the outdegrees must come last, and they will always be where all the connections lead to. So, DFS algorithm is applied in the reverse direction of the edges starting from each outdegree. A dictionary used to prevent visiting the same node more than once. Therefore, every node is visited only once. So, if adding the visited node to the topological order is taken as the basic operation, the worst-case time complexity is linear. **But** if checking if the node is visited before is taken as the basic operation (which is the most done) then its worst case is exponential. The worst case is having n/2 outdegree nodes and having the rest n/2 many nodes having edges with all outdegrees. This would result in checking n/2 nodes for n/2 outdegree nodes which then would have $\frac{n^2}{4}$ many checks. To summarize it has exponential time complexity.

b)

Question 2

Whenever $a^n = a^{2k}$ it can be calculated as $a^k \cdot a^k$, and if the n is odd then it can be represented as $a^{2k} \cdot a$. Therefore, there is cases of latter only as many as the first case. And if only the first case happens throughout the process, then it has logarithmic time complexity, and if the latter case happens whenever possible it only makes it two times of the mentioned logarithmic time complexity which still logarithmic.

Question 3

The algorithm fills the entire 9x9 grid with numbers arranged from 1 to 9, and whenever the whole grid is filled it's checked if the given sudoku is solved. There are 9 possible values for each square and there are 81 squares in total. Therefore, the grid is checked if it's solved 9^{81} times.

The checking process looks at all the original values of the given sudoku grid and the checked sudoku grid if they are equal, in total 81 checks are done. The rest of the checking process does not have more complex checks as far as time is concerned.

So, if the rules of sudoku could be expanded such that for a grid with n^2 squares that it can be solved by placing numbers from 1 to n then the algorithm mentioned above would have time complexity of O($n^2 \cdot n^{n^2}$) which shows how horrendously slow it is.

Question 4

Given that array = {6, 8, 9, 8, 3, 3, 12} sort it using: (for ease of writing commas won't be used)

- Insertion sort
  - 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
  - 3 3 6 8 8 9 12
- Quick sort
  - 6 8 9 8 3 3 12
  - 6 3 9 8 3 8 12
  - 6 3 3 8 9 8 12
  - 3 3 6 8 9 8 12
  - 3 3 6 8 8 9 12
- Bubble sort
  - 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

Insertion and bubble sort are stable since they only swap elements in case of inequality and direction of swaps are always in one way. Quick sort is not stable since it may swap the pivot element with another element that is equal to it.

Question 5

a) Brute force is the design that directly follows definitions of the problem. An exhaustive search is a special case of brute force that checks the entire set of potential solutions.

b) Caesar's Cipher can be brute forced since the number of letters in an alphabet is generally small. AES can't be brute forced since the set of keys available that affects the encryption is huge.

c) Well, if what is mentioned is checking if *one* value is prime or not the given algorithm is linear and not exponential. But if finding the primes until given n is what is meant then it is indeed exponential since n many integers are checked and the checking algorithm is linear time.