CSE321 – Introduction to Algorithm Design

Yusuf Arslan
200104004112

1. (a) The code performs a depth-first search over the graph, which takes O(V + E) time, where V is the number of nodes and E is the number of edges. The function dfs() visits each node at most once and visits each edge at most once, resulting in a time complexity of O(V + E).

1. (b) The code starts by looping over all the nodes (V) and counting the number of incoming edges (E). This takes O(V + E) time. Then, it iterates through the nodes with no incoming edges and adds them to the topological order one by one. This takes O(V) time. For each node, it iterates over all the edges (E) and removes the node from the edge's list if it is present. This takes O(E) time. Finally, it checks if there are any edges left in the graph, which takes O(E) time. Therefore, the total time complexity is O(V + E).

2. The code given above is an efficient solution to finding the power of a number. It uses a while loop to keep dividing the exponent by 2 until it reaches 0. Every time the exponent is divided by 2, the base is squared, and if the exponent is odd, the result is multiplied by the base. This way, the time complexity is reduced to O(log n) since the number of loop iterations is halved each time.

3. The time complexity of this code is O(n*m^2) where n is the number of empty cells in the board, and m is the possible value for each cell (1-9). The code uses a recursive backtracking approach to solve the sudoku board, traversing each empty cell until a valid value is found. The code will loop over each empty cell, and for each cell, loop over each possible value, so the time complexity is O(n*m^2).

Algorithm of insertion sort:

1. Begin with an empty sorted array and the given unsorted array.
2. Iterate through the unsorted array and for each element, insert it into the correct position in the sorted array.
3. To insert an element into the correct position in the sorted array, compare it to each element in the sorted array from right to left. If the element is smaller than any of the elements it is compared to, it should be inserted to the left of that element.

Apply this algorithm to the array [6, 8, 9, 8, 3, 3, 12]:

1. Start with an empty sorted array and the given unsorted array [6, 8, 9, 8, 3, 3, 12].
2. Take the first element from the unsorted array, which is 6, and insert it into the sorted array. The sorted array is now [6] and the unsorted array is [8, 9, 8, 3, 3, 12].
3. Take the next element from the unsorted array, which is 8, and insert it into the sorted array. The sorted array is now [6, 8] and the unsorted array is [9, 8, 3, 3, 12].
4. Take the next element from the unsorted array, which is 9, and insert it into the sorted array. The sorted array is now [6, 8, 9] and the unsorted array is [8, 3, 3, 12].
5. Take the next element from the unsorted array, which is 8, and insert it into the sorted array. The sorted array is now [6, 8, 8, 9] and the unsorted array is [3, 3, 12].
6. Take the next element from the unsorted array, which is 3, and insert it into the sorted array. The sorted array is now [3, 6, 8, 8, 9] and the unsorted array is [3, 12].
7. Take the next element from the unsorted array, which is 3, and insert it into the sorted array. The sorted array is now [3, 3, 6, 8, 8, 9] and the unsorted array is [12].
8. Take the last element from the unsorted array, which is 12, and insert it into the sorted array. The sorted array is now [3, 3, 6, 8, 8, 9, 12] and the unsorted array is empty.

The final sorted array is [3, 3, 6, 8, 8, 9, 12].

Insertion sort algorithm is stable.
In the example above, there were two elements with the value of 8 in the original array. After sorting, these two elements remained in the same relative order in the sorted array. This shows that the algorithm is stable.

Algorithm of bubble sort:

1. Start by comparing the first two elements of the list. If the first element is larger than the second, swap them.
2. Continue this process for the next pair of elements (2nd and 3rd elements) and so on, until you reach the end of the list.
3. Now, go back to the beginning of the list and repeat the process until no more swaps are needed.

Apply this algorithm to the array [6, 8, 9, 8, 3, 3, 12]:

1. Compare the first two elements (6 and 8) and swap them if they are in the wrong order. In this case, the elements are already in the correct order, so we don't need to swap them. The array now looks like this: [6, 8, 9, 8, 3, 3, 12]
2. Compare the next two elements (8 and 9) and swap them if necessary. Again, these elements are already in the correct order, so we don't need to do anything. The array now looks like this: [6, 8, 9, 8, 3, 3, 12]
3. Continue this process for the next pair of elements (9 and 8). Since 9 is greater than 8, we need to swap these elements. The array now looks like this: [6, 8, 8, 9, 3, 3, 12]
4. Compare the next pair of elements (8 and 9), and again, these are already in the correct order, so we don't need to do anything. The array now looks like this: [6, 8, 8, 9, 3, 3, 12]
5. Continue the process for the next pair of elements (9 and 3), and since 9 is greater than 3, we need to swap them. The array now looks like this: [6, 8, 8, 3, 9, 3, 12]
6. Compare the next pair of elements (3 and 9), and since 3 is smaller than 9, we need to swap them. The array now looks like this: [6, 8, 8, 3, 3, 9, 12]
7. Go back to the beginning of the list and repeat the process until no more swaps are needed.
   → [6, 8, 3, 3, 8, 9, 12]
   → [6, 3, 3, 8, 8, 9, 12]
   → [3, 3, 6, 8, 8, 9, 12]
8. The array is now fully sorted in ascending order: [3, 3, 6, 8, 8, 9, 12] (Some steps were passed for simplicity)

Bubble sort is also a stable sorting algorithm, In the array [6, 8, 8, 9, 3, 3, 12], the two 8s are already in the correct order relative to each other, so they are never swapped during the sorting process. This means that the final sorted array will have the two 8s in the same relative position as they were in the original array.

Algorithm of quicksort:
is a sorting algorithm that uses a divide-and-conquer approach to sort a list of elements. It works by selecting a pivot element from the list and partitioning the other elements into two groups: those that are less than the pivot, and those that are greater than the pivot. The quicksort algorithm then recursively sorts the sublists of elements that are less than and greater than the pivot.

To apply the quicksort algorithm to the array [6, 8, 9, 8, 3, 3, 12], we can follow these steps:

1. Select a pivot element from the array. For this example, let's choose the first element, 6.
2. Partition the other elements of the array into two groups: those that are less than 6, and those that are greater than 6. In this case, the array becomes [3, 3, 8, 8, 9, 12] with the pivot element 6 in the middle.
3. Recursively sort the sublists of elements that are less than 6 and greater than 6. The array becomes [3, 3, 6, 8, 8, 9, 12].

To show the recursive steps in detail, let's use the example array [6, 8, 9, 8, 3, 3, 12] and the pivot element 6.

1. First, the array is partitioned into two sublists: those elements that are less than 6, and those that are greater than 6. This results in the array [3, 3, 8, 8, 9, 12] with the pivot element 6 in the middle.
2. The quicksort algorithm then recursively sorts the sublist of elements that are less than 6. This sublist is [3, 3]. To apply the quicksort algorithm to this sublist, we need to select a pivot element. Let's choose the first element, 3.
3. The sublist [3, 3] is partitioned into two sublists: those elements that are less than 3, and those that are greater than 3. Since there are no elements in the sublist that are less than 3, the array becomes [3, 3] with the pivot element 3 in the middle.
4. Since there are no further elements to sort in the sublist of elements that are less than 3, the quicksort algorithm moves on to the sublist of elements that are greater than 3. This sublist is [8, 8, 9, 12]. To apply the quicksort algorithm to this sublist, we need to select a pivot element. Let's choose the first element, 8.
5. The sublist [8, 8, 9, 12] is partitioned into two sublists: those elements that are less than 8, and those that are greater than 8. This results in the array [8, 8, 9, 12] with the pivot element 8 in the middle.
6. The quicksort algorithm then recursively sorts the sublist of elements that are less than 8. This sublist is empty, so there are no further steps to take.
7. The quicksort algorithm then recursively sorts the sublist of elements that are greater than 8. This sublist is [9, 12]. To apply the quicksort algorithm to this sublist, we need to select a pivot element. Let's choose the first element, 9.
8. The sublist [9, 12] is partitioned into two sublists: those elements that are less than 9, and those that are greater than 9. Since there are no elements in the sublist that are less than 9, the array becomes [9, 12] with the pivot element 9 in the middle.
9. Since there are no further elements to sort in the sublist of elements that are less than 9, the quicksort algorithm moves on to the sublist of elements that are greater than 9. This sublist is empty, so there are no further steps to take.
10. At this point, all of the sublists have been recursively sorted, so the array [6, 8, 9, 8, 3, 3, 12] is now fully sorted in ascending order.

5. (a)
Exhaustive search is a method of problem solving or search that evaluates all possible solutions to a problem in order to find the optimal solution. It is also known as brute-force search or generate and test. Exhaustive search is often used when no more efficient algorithms are available.

Exhaustive search and brute-force are often used interchangeably to refer to the same problem-solving strategy. Brute-force is a subset of exhaustive search, meaning that all possible solutions are tested, but brute-force typically refers to computationally intensive techniques that are used to solve problems with a large search space.

An example of exhaustive search is a computer program that is trying to find the shortest route between two cities. The program will generate all possible routes and then evaluate each one to determine which is the shortest.

5. (b)
Caesar's Cipher is a type of substitution cipher where each letter in the plaintext is shifted a certain number of places to the right, creating a ciphertext.

AES (Advanced Encryption Standard) is a symmetric key encryption algorithm used by governments, businesses, and individuals to protect sensitive data. It is a block cipher, meaning that it takes a fixed-length string of plaintext and encrypts it into a fixed-length string of ciphertext. AES is considered one of the most secure encryption algorithms available and is widely used in both hardware and software.

Caesar's Cipher is vulnerable to a brute force attack, as it only uses a single key for encryption and decryption. A brute force attack involves trying every possible key combination until the correct key is found and the ciphertext is decrypted.

On the other hand, AES is not vulnerable to brute force attacks. AES is considered to be very secure, and it would take an extremely powerful computer and a very long time to break the encryption. AES uses much longer keys than Caesar's Cipher, and it also uses a complex algorithm that is designed to make it very difficult to break.

5. (c)
The naive solution to primality testing is a method of determining whether a given number is prime or not by checking if any number between 2 and n-1 (where n is the input) divides n. This method grows exponentially because the time it takes to check all the possible values increases exponentially with the size of the input. For example, if n is 10, the naive solution would check 2, 3, 4, 5, 6, 7, 8, and 9, which is 8 tests. If n is 1000, then it would take 999 tests to find out if it is prime or not. As the size of the input increases, the number of tests required to find out if the input is prime grows exponentially.