You are encouraged to discuss amongst yourselves, but please write your answers yourselves, based on your own understanding.

Questions 4 and 5 are optional (need not be solved either), but are still recommended.

1. Choose one of the sorting algorithms (other than Insertion Sort) and draw a decision tree for an input of size 3. (If you choose QuickSort, please also specify how the pivot is chosen.)

2. Write a formal proof to show that

   - the decision tree has exactly $n!$ leaves.
   - the decision tree must have a height of $\Omega(n \log n)$.

   (Please note that, it was evident from the doubt clearing sessions that many people had doubts with this and so, contrary to what was said in the video, the second part is not optional.)

3. Does the following algorithm correctly check whether a number is prime or not? Is it a polynomial time algorithm for the problem? Clearly justify your answers.

```
ISPRIME($n$)
Input. Integer $n \geq 4$
Output. YES if $n$ is a prime number, NO otherwise.
1: $j = 1, k = 1$
2: for $i = 2$ to $\lfloor \sqrt{n} \rfloor + 1$ do
3:     if $n \mod i == 0$ then
4:         return NO
5:     end if
6: end for
7: return YES
```

4. Using the recurrence tree method, prove that there exist constants $c, d > 1$ such that the solution of the following recurrence relation lies between c^n and d^n.

   $$T(n) = 2 \cdot T(n-1) + T(n-2)$$

   (In other words, show that the solution of the recurrence is exponential in n.)

5. Solve the recurrence given in the previous question using the characteristic roots method.

6. Clearly explain why the DP algorithm for computing the n^{th} Fibonacci number is *not* quadratic in the size of the input.

7. Recall the repeated squaring approach we had for computing the n^{th} Fibonacci number. We observed that, as the computation involved irrational numbers, the complexity of the approach was undefined, unless we first determined the number of decimal digits needed to ensure that the correct output will be obtained.

   As an alternative to this, present a way to run this same algorithm without dealing with any decimal numbers. Also, analyze the bit complexity of your algorithm.