Question - 02

Code:

```
int find_a(char *str)
{
  int i;
  for(i=0; str[i]; i++)
  {
    if (str[i] == 'a')
      return i;
  }
  return -1;
}
```

Here, the worst case time complexity of the function is O(N), because the function iterates over each character in the string 'str' and checks if it is equal to the character 'a'. In the worst case, the character

'a' may not be present in the string, so the loop will iterate over all N characters in the string before reaching the end and returning -1. Therefore, the time complexity is O(N) in the worst case.

Here, the best case time complexity of the function is O(1), because the loop will only execute once and function will immediately return the index of the first character, which is 0. The best case scenario occurs when the first character of the input string 'str' is 'a'. Therefore, the time complexity of the function in this case is O(1).

Question-3

The Tower of Hanoi algorithm is recursive algorithm that solves the tower of Hanoi puzzle. The problem involves moving a stack of disks from one peg to another, with the constraint that larger disks cannot be placed on smaller disks.

Therefore,

Moving n-1 disks from source to aux means the first peg to the second peg, this can be done in T(n-1). step

Moving n disks from source to dest means a larger disk from peg to the third peg will require 1 step.

Moving n-1 disks from aux to dest means the second peg to the third peg will require again

T(n-1) step.

$\therefore T(n) = T(n-1) + 1 + T(n-1)$.

So, the equations,

$T(n) = 2T(n-1) + 1 \ldots\ldots (i)$

$T(n-1) = 2T(n-2) + 1 \ldots\ldots (ii)$

$T(n-2) = 2T(n-3) + 1 \ldots\ldots (iii)$.

From (ii) and (iii),

$T(n-1) = 2T(2T(n-3)+1) + 1 \ldots\ldots (iv)$.

From (i) and (iv),

$T(n) = 2(2(2T(n-3)+1)+1)+1$

$T(n) = 2^3 T(n-3) + 2^2 + 2^1 + 1$

$\therefore T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \ldots\ldots + 2^2 + 2^1 + 2^0$

From our base condition $T(1) = 1$

$n - k = 1 \Rightarrow k = n-1 \ldots\ldots (v)$

From (iv) and (v),

$T(n) = 2^{n-1} T(n-(n-1)) + 2^{k-1} + 2^{k-2} + \ldots\ldots + 2^2 + 2^1 + 2^0$

$T(n) = 2^k \cdot (1) + 2^{k-1} + 2^{k-2} + \ldots\ldots + 2^2 + 2^1 + 2^0$

$$\therefore T(n) = 1 \cdot \frac{1-2^{i+1}}{1-2}$$

$$\therefore T(n) = 2^{i+1} - 1$$

From the above equation;

$$T(n) = 2^{n-1+1} - 1$$

$$\therefore T(n) = 2^n - 1$$

So, the time complexity is $O(2^n)$ which is exponential.