Time Complexity in DSA

```
1.  for i in range(n): # c time
              i = i*2  # n*c

# Time Complexity(TC) = n*c = O(n)
```

So, the time complexity of the above code is O(n).

```
2.  def fact_iter(n): # c1 time
  "assumes n an int >= 0"
  answer = 1          # c2 time
  while n > 1:        # c3 time
    answer *= n       # n*c4 time
    n -= 1            # n*c5 time
  return answer       # c6 times

#Time Complexity(TC) = c1 + c2 + c3 + n*c4 + n*c5 + c6 = O(n)
```

So, the worst time complexity of the above code is O(n).

```
3.  A = [1, 2, 3, 4]  # c1
    B = [2, 3, 4, 5, 6] # c2
    for i in A: # c3
      for j in B: # c4
        if i<j:
          print('{}, {}'.format(i, j)) # n*m*c5

#Time Complexity (TC) = c1 + c2 + c3 + c4 + n*m*c5 = O(n*m)
```

So, the time complexity of the above code is the O(n*m)

```
4. L = [1, 2, 3, 4, 5, 6, 7, 8] # c1
   for i in range(len(L)//2): # c2
      other = len(L) - i - 1
      temp = L[i]
      L[i] = L[other]
      L[other] = temp        # len(n)/2*c3
```

Here, the code is for the swapping, so the above code will run for n/2 times the length of the list.

Time Complexity = O(n)

```
5. def fib(n):
  if n==1 or n==0:
    return 1
  else:
    return fib(n-1) + fib(n-2)
```

This is the recursive function to calculate the Fibonacci of the given number n. So, the time complexity

Time complexity (TC) = 2^n.