Time Complexity

1.

```
    for i in range(n):
       i = i*2
```

Since, the loop iterates for n number of times so the time complexity is O(n).

2.

```
    def fact_iter(n):
        "assumes n an int >= 0"
        answer = 1
        while n > 1:
          answer *= n
          n -= 1
        return answer
```

The loop iterates n times because it starts with n and decrements n by 1 in each iteration until it reaches 1. Hence, the time complexity is O(n).

3.

```
A = [1, 2, 3, 4]
B = [2, 3, 4, 5, 6]
for i in A:
  for j in B:
    if i<j:
      print('{}, {}'.format(i, j))
```

Here, the outer loop is iterated till the length of A and the inner loop is iterated till the length of B.

If we consider the length of A and B as m and n respectively, then the time complexity will be O(m*n).

4.

```
L = [1, 2, 3, 4, 5, 6, 7, 8]
for i in range(len(L)//2):
  other = len(L) - i - 1
  temp = L[i]
  L[i] = L[other]
  L[other] = temp
```

The loop iterates len(L)//2 times where len(L) denotes the length of the list. The approximated time complexity of the above code is O(n/2) which ultimately simplifies to O(n) where n is the length of the list L.

5.

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

Here, the number of recursive function calls doubles for each increment of n. In the worst case, when n is large, the recursive calls will create a binary tree-like structure, where each node has two child nodes. Therefore, the number of nodes (recursive calls) in the tree will be approximately 2^n, leading to an exponential number of function calls. As a result, the time complexity for this recursive function is O(2^n).