# Day 81/180 Time and Space Complexity in Recursion

# Calculate the time and space complexity of each program

1:

```
int fact(int n)
{
   if(n<=1)
   return 1;

   return n*fact(n-1);

}
```

**Recursive calls** : fact(n) -> fact(n-1) -> … …… … -> fact(1).

**Time Complexity** : O(n)
**Space Complexity** : O(n)

2:

```
int power(int base, int exponent) {
   if (exponent == 0)
       return 1;
   return base * power(base, exponent - 1);
}
```

**Recursive calls** : power(base, exponent ) -> power(base, exponent -1 )-> … …… … ->power(base, 0 ).

So, no of recursive calls are dependent on exponent variable.

**Time Complexity** : O(exponent)
**Space Complexity** : O(exponent)

3:

```
bool isPalindrome(string str, int start, int end) {
   if (start >= end)
       return true;
   return (str[start] == str[end]) && isPalindrome(str, start + 1, end - 1);
}
```

**Assume** : size of str = n
**Recursive calls** : isPalindrome( str, 0,  n -1) -> isPalindrome( str, 1 , n-2) -> isPalindrome( str, start+2,  n-3) -> …… …… ……. -> isPalindrome( str, mid,  mid)

So, no of recursive call will be ther (start to mid) or (mid to end) that is n/2 i.e size of str.
But Here **str is passed by value**, so for each of the recursive call new string created for that call. So, copying str will take **O(n)** time complexity for each calls and there are nearly **n/2** calls.

Here **str is passed by value**, so for each of the recursive call new string created for that call. So, each calls space complexity is **O(n),** and there are nearly **n/2** calls.

**Time Complexity** : O( n ^2)
**Space Complexity** : O( n^2 )

4:

```
void reverseString(string& str, int start, int end) {
   if (start < end) {
```

```
        swap(str[start], str[end]);
        reverseString(str, start + 1, end - 1);
    }
}
```

**Assume** : size of str = n

**Recursive calls** : reverseString( str, 0,  n -1) -> reverseString( str, 1 , n-2) -> reverseString( str, start+2,  n-3) -> ……  ……  ……. -> reverseString( str, mid,  mid)

So, no of recursive call will be ther (start to mid) or (mid to end) that is n/2.

Here **str is passed by reference**, so it the same str in every call, so, no extra space.

**Time Complexity** : O( n  )
**Space Complexity** : O( n )

5:

```
bool isEven(int n) {
    if (n == 0)
        return true;
    return !isEven(n - 1);
}
```

**Recursive calls** : isEven(n) -> isEven(n-1) -> … …..-> isEven(0)

**Time Complexity** : O(n)
**Space Complexity** : O(n)