Analyzing the complexity of a recursive algorithm

Week 13

The basic idea

- Usually, counting the running time of a recursive algorithm is straigthforward, because the formula for $T(n)$ can be seen directly from the form of the algorithm.

```
def factorial(n):
    if n == 0:                    # base case
        return 1
    else:                         # recursive case
        return n * factorial(n-1)
```

Analysis of `factorial(n)`

- **Base Case**: If `n` is 0, the function returns 1. Suppose that this takes some constant time c.
- **Recursive Case:** If `n` is greater than 1, the function calls itself with `n-1` and multiplies the result by `n`. Let us assume that this takes time d.
- We can write the following equation

$$T(n) = T(n-1) + d$$

Analysis of `factorial(n)`

- First iteration: $T(n) = T(n-1) + d$
- Second iteration: $T(n-1) = T(n-2) + d$

  => $T(n) = T(n-2) + d + d$ = $T(n-2) + 2d$
- Third iteration: $T(n-2) = T(n-3) + d$

  => $T(n) = T(n-3) + d + 2d$ = $T(n-3) + 3d$

General Form: After k iterations, the general form is:

$$T(n) = T(n-k) + kd$$

Analysis of `factorial(n)`

- The base case: When $n - k = 0$, we have reached the bottom. Then $k = n$. We obtain

$$T(n) = T(n-k) + kd = T(n-n) + nd$$

- Thus,

$$T(n) = T(0) + dn = dn + c$$

- The running time is **linear**.