Cheatsheet - Time Complexity of Recursive Algorithms

Fabio Lama – fabio.lama@pm.me

1. About

This cheatsheet covers the time complexity of recursive algorithms, including how to apply the master theorem.

2. Recurrence Relation

Consider:

```
function fact(N)
    if (N==1)
        return 1
    return N × fact(N − 1)
```

If we analyze this function one by one:

```
    if (N==1)
```

Constant time C_0 (grouped).

```
    return N × fact(N − 1)
```

Only one of the two return statements is executed (we assume the second statement, worst case).

```
        return 1
    return N × fact(N − 1)
```

Constant time C_1 (return), C_2 (read N), C_3 (multiplication), and the fact function has a time of $T(N-1)$. If we group all the constants together, we have:

$$C_1 + C_2 + C_3 = C_4$$

So we say that the return statement has a time complexity of:

$$C_4 + T(N-1)$$

Finally, we determine that the function fact has a time complexity of:

$$T(N) = C_0 + C_4 + T(N-1)$$

Or grouped together:

$$T(N) = C_5 + T(N-1)$$

This expression is known as a **recurrence relation**.

3. Solving a Recurrence Relation

We can demonstrate the solution of a recurrence relation by expanding the calculation:

$$T(N) = C_5 + T(N-1)$$
$$T(N) = C_5 + T(C_5 + T(N-2))$$
$$T(N) = C_5 + T(C_5 + T(C_5 + T(N-3)))$$
$$\ldots$$
$$T(N) = k \times C_5 + T(N-k)$$

If k is $N-1$, then:

$$T(N) = (N-1) \times C_5 + T(N-(N-1))$$
$$T(N) = (N-1) \times C_5 + C$$

That's because:

$$T(N-(N-1))$$
$$= T(N-N+1)$$
$$= T(1)$$
$$= C$$

TODO: we need more explanation and clarification here.

We have hence solved the recurrence equation. In summary:

- Find its recurrence equation.
- Solve the recurrence equation.
- Asymptotic analysis.

4. The Master Theorem

The master theorem makes the asymptotic analysis of recursive functions much easier. However, it can only be applied if the recurrence equation has this specific structure:

$$T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$.

For example, this is solvable:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + n \\ &= 1 \times T\left(\frac{n}{2}\right) + n \end{aligned}$$

Meanwhile, this is not:

$$\begin{aligned} T(n) &= 2 \times T(n) + n \\ &= 2 \times T\left(\frac{n}{1}\right) + n \end{aligned}$$

The master theorem classifies the recurrence equiation in one of three cases:

4.1. Case 1

$$f(n) < n^{\log_b a}$$

where the running time is:

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

4.2. Case 2

$$f(n) = n^{\log_b a}$$

where the running time is:

$$T(n) = \Theta\left(n^{\log_b a} \times \log N\right)$$

4.3. Case 3

$$f(n) > n^{\log_b a}$$

and:

$$a \times f\left(\frac{n}{b}\right) \leq c \times f(n) \quad \text{where} \quad c < 1 \quad \text{and} \quad n \quad \text{large}$$

where the running time is:

$$T(n) = \Theta(f(n))$$

4.4. Example

Consider:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

we deterime that the master theorem can be applied here, given that $a = 2 \geq 1$ and $b = 2 > 1$.

To determine the case, we first calculate:

$$\log_b a = \log_2 2 = 1$$

For case 1, we have:

$$\begin{aligned} n &< n^1 \\ n &< n \end{aligned}$$

which is **false**.

For case 2, we have:

$$n = n^1$$

$$n = n$$

which is **true**. Hence, we classify the formula as case 2 and the running time is:

$$T(n) = \Theta(n \times \log N)$$