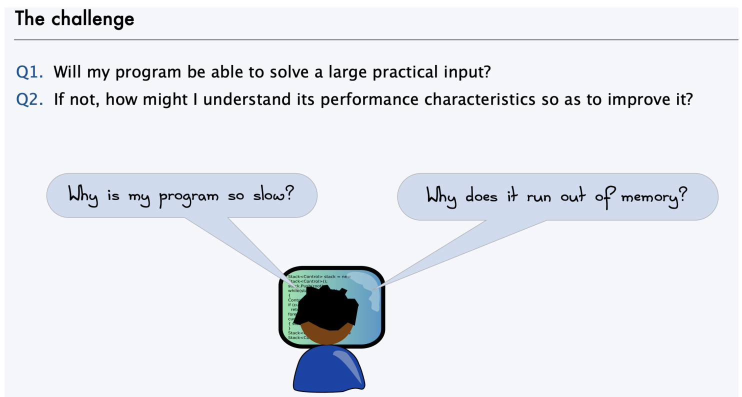
Algorithms analysis

CS Osvita

January 9, 2025



An algorithm's execution time can be expressed as the number of steps required to solve the problem. This abstraction is exactly what we need: it characterizes an algorithm's efficiency in execution time while remaining independent of any particular program or computer.

Understanding this is like getting brand new glasses - you start seeing the world in a way you didn't even know existed before - it's a completely new perspective.

Let's take a closer look at some examples. The most expensive computation unit will be an *apicall()*. So, our goal is to find the number of times *apicall()* is called as a function of the input size n.

For loop

```
for (i = 4; i < n + 42; i += 2)
    apicall()
```

apicall() is called exactly

$$(n + 42 - 4)/2$$

The time complexity is linear; it means that if we increase the input size *10, our program will make 10 times more *apicall()*

```
for (i = 10; i < n + 5; i *= 3)
    apicall()
```

The loop will multiply i by 3 until

$$10 * 3^i >= n + 5$$

$$3^i >= (n + 5)/10$$

$$i = log_3(n + 5)/10$$

Therefore, *apicall()* is called roughly logn times

Double loop

```
for (i = 0; i < n; ++i)
    for (j = 0; j < n; j += 2)
            apicall()
```

```
i=0 => n/2
i=1 => n/2
i=2 => n/2
...
i=n => 0
```

The total number of calls is

$$\frac{n}{2} + \frac{n}{2} + ... + \frac{n}{2} = \frac{n^2}{2}$$

.

Triple loop

```
for (i = 1; i <= n; ++i)
    for (j = 1; j <= i; ++j)
        for (k = 1; k <= i; ++k)
            apicall()
```

```
i=1 => 1*1
i=2 => 2*2
i=3 => 3*3
...
i=n => n*n
```

The total number of times *apicall()* is performed is

$$1^2 + 2^2 + ... + n^2 = n^3$$

To understand why, consider the following observations:

1. It's not geometric or arithmetic progression
2. Upped bound:

$$1^2 + 2^2 + ... + n^2 \leq n^2 + n^2 + ... + n^2 \leq n * n^2 = n^3$$

   Therefore, the order of growth is not more than n^3
3. Lower bound:

$$1^2+2^2+...+n^2 \geq (\frac{n}{2})^2+(\frac{n}{2}+1)^2+...+n^2 \geq (\frac{n}{2})^2+(\frac{n}{2})^2+...+(\frac{n}{2})^2 \geq n/2*(n/2)^2 = \frac{n^3}{8}$$

   Therefore, the order of growth is not less than $\frac{n^3}{8}$

Since we drop constant coefficients, the upper and lower bounds imply that the order of growth is n^3

Recursion

```
def f(n: int) -> None:
    if n == 0: return
    for (i = 0; i < n; ++i)
        apicall()
    f(n-1)
```

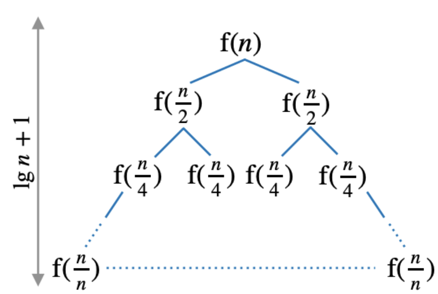
apicall() is executed n times for $f(n)$ and $n-1$ times for $f(n-1)$, etc. This means that *apicall()* is executed

$$n + (n-1) + (n-2) + ... + 1 = n * (n+1)/2$$

The amount of space required to execute our code is linear.

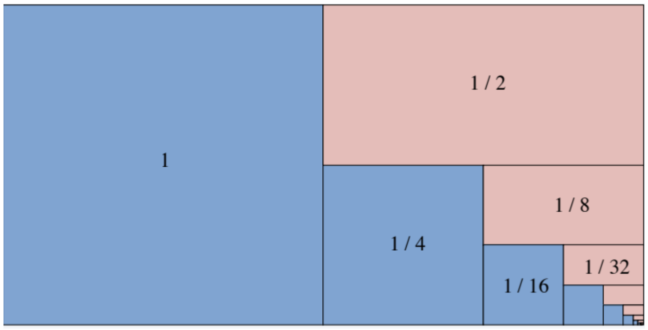
```
def f(n: int) -> None {
    if n == 0: return
    apicall()
    f(n/2)
    f(n/2)
}
```

Consider the following visualization:



apicall() is performed once in each recursive call:

$$1 + 2 + 4 + ... + n/4 + n/2 + n = 2n - 1$$




```python
def fib(n: int) -> int:
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)
```

Here is the animation of the code execution (zoom it if it is small).
The number of additions is:

$$2^0 + 2^1 + 2^2 + ... + 2^n$$

Using the formula for the sum of the first $n+1$ terms of a geometric series with the first term $a = 1$ and the common ratio $r = 2$ we got $2^{n+1} - 1$.