CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

[1]based on notes by Anna Bretscher and Albert Lai

how long do things take

Remember this algorithm?

```
i = 1
while i < len(A):
  v = A[i]
  j = i
  while j > 0 and A[j-1] > v:
    A[j] = A[j-1]
    j = j - 1
  A[j] = v
  i = i + 1
```

What do we count? Does it matter?

how long do things take

Let's try counting this way:

- get/set variables: 1 step
- function call: 1 + steps to evaluate each argument + steps to execute function
- return statement: 1 + steps to evaluate return value
- if/while condition: 1 + steps to evaluate the boolean expression
- assignment statement: 1 + steps to evaluate each side
- arithmetic/comparison/boolean operators: 1 + steps to evaluate each operand
- array access: 1 + steps to evaluate array index
- constants: free!

how long do things take

```
def InsertionSort (A):                        STEPS
    i = 1                                       2
    while i < len(A):                           5
      v = A[i]                                  5
      j = i                                     3
      while j > 0 and A[j-1] > v:               10 or 3
        A[j] = A[j-1]                           8
        j = j - 1                               4
      A[j] = v                                  5
      i = i + 1                                 4
```

What assumptions did we make? Are they realistic?

So, what's the total number of steps?

how long do things take

In the **worst case**:

how long do things take

In the **best case**:

how long do things take

What if we write the same algorithm differently?

```
0 def InsertionSort (A):
1     n = len(A)
2     for (i = 1; i < n; i++):
3         for (j = i; j > 0 and A[j] < A[j-1]; j--):
4             swap A[j], A[j-1]
```

- line 1: once, 4 steps

For $n \geq 1$:

- line 2: 1 step (once) + 2 steps (once) + 3 steps (n times) + 2 steps ($n - 1$ times)
- line 3: for each i: 1 step (once) + 3 steps (once) + 11 steps (i times) + 2 steps (once) + 2 steps (i times)
- line 4: for each i: 9 steps (i times)

how long do things take

- line 1: once, 4 steps

For $n \geq 1$:

- line 2: 1 step (once) + 2 steps (once) + 3 steps (n times) + 2 steps ($n - 1$ times)
- line 3: for each i: 1 step (once) + 3 steps (once) + 11 steps (i times) + 2 steps (once) + 2 steps (i times)
- line 4: for each i: 9 steps (i times)

$= 11n^2 - 1$

Is this the same running time? In what sense?

how long do things take

Q. What if I now run this algorithm on a machine that is slower to perform variable look up and write?

Q. Should the complexity change?

Q. How important are those constants as the input size n gets large?

Q. How are our two results $11n^2 + 14n - 18$ and $11n^2 - 1$ similar?

Q. They are both quadratic polynomials.

how long do things take

We say...

- that a quadratic polynomial is of order n^2,
- that a cubic polynomial is of order n^3,
- that $4n\lg(n) + 2n + 10$ is of order $n\lg(n)$.

Why can we say this? With a little mathemagic:

$$11n^2 + 14n - 18 \leq$$

Another example:

$$11n^2 - 21n + 19 \leq$$

how long do things take — formally

For all natural $n \geq 19$:

$$11n^2 - 21n + 19 \leq 12n^2$$

There exists an $n_0 \in \mathbb{N}$ such that, for all natural $n \geq n_0$,

$$11n^2 - 21n + 19 \leq 12n^2$$

We can take this even further and say, there exists real $c > 0$ and natural n_0 such that, for all natural $n \geq n_0$,

$$11n^2 - 21n + 19 \leq c \cdot n^2$$

which is exactly the definition of "Big-Oh"!

Big-Oh — Asymptotic Upper Bound

We denote:

- $\mathbb{N}$: the set of natural numbers
- $\mathbb{R}^+$: the set of positive real numbers
- $\mathcal{F}$: the set of functions $f : \mathbb{N} \to \mathbb{R}^+$

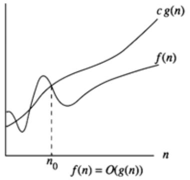
Let $g \in \mathcal{F}$. Define $\mathcal{O}(g)$ to be the set of functions $f \in \mathcal{F}$ such that

$$\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n)$$

Big-Oh — Asymptotic Upper Bound

Let $g \in \mathcal{F}$. Define $\mathcal{O}(g)$ to be the set of functions $f \in \mathcal{F}$ such that

$$\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n)$$



Let's practice proving a function belongs to big-Oh of another function.

Big-Oh practice

Suppose we determine an algorithm has running time

$$T(n) = n^3 - n^2 + 5$$

Prove. $T(n) \in O(n^3)$

Is Big-Oh good enough?

Q. Is $12n^2 + 10n + 10 \in O(n^3)$?

Q. Is $12n^2 + 10n + 10 \in O(n^2 \lg n)$?

Q. Is $n \in O(n^2)$?

Q. Is $3 \in O(n^2)$?

$O(n^2)$ includes quadratic functions and “lesser” functions as well.

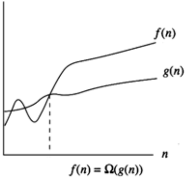
We need another definition to exclude “lesser” functions.

Big-Ω — Asymptotic Lower Bound

Idea. Want a function g such that for big enough n,

$$0 \leq b \cdot g(n) \leq f(n)$$

where b is a constant.



“Big Omega.” Let $g \in \mathcal{F}$. Define $\Omega(g)$ to be the set of functions $f \in \mathcal{F}$ such that

$$\exists b \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow f(n) \geq b \cdot g(n) \geq 0$$

Equivalently, $f \in \Omega(g)$ iff $g \in O(f)$.

Big-Θ — Asymptotic Tight Bound

What if it's both?
If $f \in O(g)$ and $f \in \Omega(g)$ then we say that $f \in \Theta(g)$.

"Big Theta". Let $g \in \mathcal{F}$. Define $\Theta(g)$ to be the set of functions $f \in \mathcal{F}$ such that $f \in O(g) \cap \Omega(g)$

or alternatively,

$$\exists b \in \mathbb{R}^+, \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N},$$
$$n \geq n_0 \Rightarrow 0 \leq b \cdot g(n) \leq f(n) \leq c \cdot g(n)$$

Big-Θ practice

Show: $11n^2 + 14n - 18 \in \Theta(n^2)$

Let $f(n) = n^3 - n^2 + 5$. Show: $f \in \Theta(n^3)$

Show: $n \notin \Theta(n^2)$

in summary

- concerned about the efficiency of an algorithm as the input size gets large
- not concerned about small constants as these are machine dependent
- therefore, use asymptotic notation: $\mathcal{O}, \Omega, \Theta$

using limits to prove Big-O

Assume. $\exists n_0 \in \mathbb{N} \colon \forall n \geq n_0 \colon f(n) \geq 0$ and $g(n) > 0$.

Theorem. If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then $f \in O(g)$.

Example. Prove $n(n+1)/2 \in O(n^2)$

Example. Prove $\ln(n) \in O(n)$

using limits to disprove Big-O

Assume. $\exists n_0 \in \mathbb{N}\colon \forall n \geq n_0\colon f(n) \geq 0$ and $g(n) > 0$.

Theorem. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f \notin O(g)$.

Example. Disprove $n^2 \in O(n)$

Example. Disprove $n \in O(\ln(n))$

when limits don't help

Theorem. If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then ...

Theorem. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then ...

Q. Which case is not covered?

A. If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ does not exist and is not ∞, then no conclusion. (Hopefully this happens rarely.)

Q. Can you think of a function *crazy* where limits do not help to show $crazy \notin O(1)$?

when limits don't help

Q. Can you think of a function *crazy* where limits do not help to show $crazy \notin O(1)$?

using limits for Θ

Theorem. $f \in \Theta(g)$ iff $f \in O(g)$ and $g \in O(f)$.

(Handy when you want to use limits!)

Example. $n^2 + n^{3/2} \in \Theta(n^2)$

Example. $\ln(n) \notin \Theta(n)$

Big-O, Big-Θ may miss something

Q. Can the Big-O definition be not at all useful?

A.

$$n + 10^{100} \in \Theta(n)$$

$$10^{100} n \in \Theta(n)$$

Can't say these are practical algorithm times, but O, Θ can't tell.

This is a price for ignoring constants (which we want to account for machine differences!)

Such pathological cases are rare. O and Θ are usually informative.

Myth Buster

Myth: O means worst-case time, Ω means best-case.

Truth: O, Ω, Θ classify functions, do not say what the functions stand for.

$9n^2 + 4n + 13$ may be best-case time, or worst-case time, or best-case space, or worst-case space, or just a polynomial from nowhere.

"Best case time is in $O(n^2)$" means:
Best case time is some function, that function is in $O(n^2)$.
Clearly a sensible statement and possible scenario.
O, Ω, Θ are good for any function from natural to non-negative real.