Analysis of Algorithms - 2 Big O, Big Omega, Big Theta

CS313E - Elements of Software Design

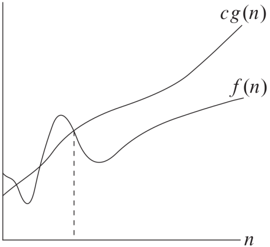
Kia Teymourian

05/11/2022

Agenda

1. Big O Notation
2. Big Ω-Notation
3. Θ-Notation
4. Asymptotic Costs of Programs

Big O-Notation, Asymptotic Upper Bound



$f(n) = O(g(n))$

$$O(g(n)) = \{f(n) \text{ there exists constants } c, n_0 > 0 \text{ such that } 0 \le f(n) \le c \times g(n) \text{ for all } n \ge n_0\}$$

Note: As you can see it does not matter what the function does before n_0, we are interested to know about the growth of the function for sufficiently large n.

Example - A Polynomial Function (1)

Let us claim that the $f(n) = a_k n^k + a_{k-1} + \cdots + a_1 n + a_0$ is upper bounded by a function $g(n) = n^k$,
$f(n) = O(n^k)$
Note: In a polynomial $a_k, \ldots, a_1, a_0$ are named coefficients.
Proof: Let us consider the following n_0 and c
$n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \cdots + |a_1| + |a_0|$ the sum of the absolute values of all coefficients.
We have to illustrate that $\forall n \geq 1, f(n) \leq c \times n^k$

Example - A Polynomial Function (2)

We have to illustrate that $\forall n \geq 1, f(n) \leq c \times n^k$
In this case, we have for all $n \geq 1$, we can consider the coefficients as absolute values and use n bigger than one, then we can get the following:

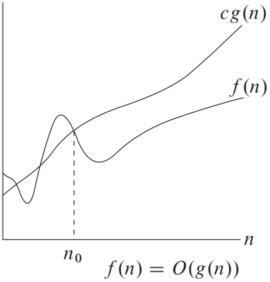
$$f(n) \leq |a_k|n^k + |a_{k-1}|n^{(k-1)} + \cdots + |a_1|n + |a_0|$$

- ▷ We get the above because if we use the absolute values we turn some of the negative values into positive values so that the actual $f(n)$ value will always be smaller than the multiplication of all coefficients to the n values when n is bigger than one.
- ▷ We can make the above bigger when we multiply it to the a larger number n^k which makes it bigger.

$$f(n) \leq |a_k|n^k + |a_{k-1}|n^k + \cdots + |a_1|n^k + |a_0|n^k$$
$$f(n) \leq c \times (n^k)$$

And this is the multiplication of the above $c = |a_k| + |a_{k-1}| + \cdots + |a_1| + |a_0|$ to the n^k. So we have proved that the $f(n)$ is always smaller equal to a constant c multiply to n^k

Big O-Notation, Asymptotic Upper Bound



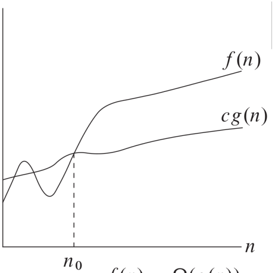
$$O(g(n)) = \{f(n) \text{ there exists constants } c, n_0 > 0 \text{ such that } 0 \leq f(n) \leq c \times g(n) \text{ for all } n \geq n_0\}$$

Note: As you can see it does not matter what the function does before n_0 or how the flow if the function is, we are interested to know about the growth of the function for sufficiently large n.

Ω-Notation, Asymptotic Lower Bound

- Big O Notation provides an asymptotic upper bound of a function.
- An *asymptotic lower bound* is provided by Ω notation.

We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or above $c \times g(n)$

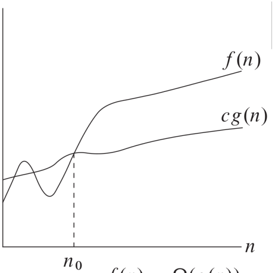


$f(n) = \Omega(g(n))$

Ω-Notation, Asymptotic Lower Bound

- Big O Notation provides an asymptotic upper bound of a function.
- An *asymptotic lower bound* is provided by Ω notation.

We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or above $c \times g(n)$



$f(n) = \Omega(g(n))$

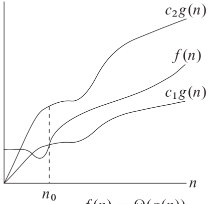
We can formally define:

$$\boldsymbol{\Omega(g(n)) = \{f(n) : \text{if there exist constants } c, n_0 > 0 \text{ such that } 0 \le c \times g(n) \le f(n) \text{ for all } n \ge n_0\}}$$

Θ-Notation, Asymptotic Tight Bound

- The Θ-notation asymptotically bounds a function from above and below.
- Θ-notation bounds a function to within constant factors.

We write $f(n) = \Theta(g(n))$ if there exist positive constants $n_0, c_1,$ and c_2 such that at and to the right of n_0, the value of $f(n)$ always lies between $c_1 \times g(n)$ and $c_2 \times g(n)$ inclusive.

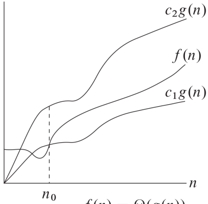


$f(n) = \Theta(g(n))$

Θ-Notation, Asymptotic Tight Bound

- The Θ-notation asymptotically bounds a function from above and below.
- Θ-notation bounds a function to within constant factors.

We write $\boldsymbol{f(n) = \Theta(g(n))}$ if there exist positive constants $\boldsymbol{n_0, c_1,}$ and $\boldsymbol{c_2}$ such that at and to the right of $\boldsymbol{n_0}$, the value of $\boldsymbol{f(n)}$ always lies between $\boldsymbol{c_1 \times g(n)}$ and $\boldsymbol{c_2 \times g(n)}$ inclusive.



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

We can formally define:

$$\Theta(g(n)) = \{f(n) \text{ if there exists constants } c_1, c_2 \text{ and } n_0 \text{ such that } \boldsymbol{0 \le c_1 \times g(n) \le f(n) \le c_2 \times g(n)} \text{ for all } n \ge n_0\}$$

Example

Claim is that $f(n) = \frac{1}{2}n^2 - 3n$ has $f(n) = \Theta(n^2)$

Example

Claim is that $f(n) = \frac{1}{2}n^2 - 3n$ has $f(n) = \Theta(n^2)$

Proof.

To proof, we must find positive constants of c_1, c_2 and n_0 such that:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ divide all sides by } n^2$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

- ▷ This holds for any value bigger equal to one $n \geq 1$ by choosing any constant $c_2 \geq \frac{1}{2}$
- ▷ Similarly, the left hand-side of the inequality holds for any values $n \geq 7$ by choosing any constant $c_1 \leq \frac{1}{14}$.

Example

Claim is that $f(n) = \frac{1}{2}n^2 - 3n$ has $f(n) = \Theta(n^2)$

Proof.

To proof, we must find positive constants of c_1, c_2 and n_0 such that:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ divide all sides by } n^2$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

- ▷ This holds for any value bigger equal to one $n \geq 1$ by choosing any constant $c_2 \geq \frac{1}{2}$
- ▷ Similarly, the left hand-side of the inequality holds for any values $n \geq 7$ by choosing any constant $c_1 \leq \frac{1}{14}$.

Thus, we found constants $c_1 = \frac{1}{14}$ and $c_2 = \frac{1}{2}$, and $n_0 = 7$ that the we can verify that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

- ▷ Other choices for the constants c_1, c_2 and n_0 are possible.
- ▷ An informal notation of Θ is to throw away lower-order terms and ignore the leading coefficient of the highest-order term, on the above example we could simply find the $\Theta(n^2)$. Here, we can simply find the $\Theta(n^2)$.

□

Example

Claim is that $f(n) = 6n^3$ is not $\Theta(n^2)$

Proof.

Here, we do here proof by contradiction.

- Suppose that there exists c_2 and n_0 exists so that we have the inequality :

$$6n^3 \leq c_2 n^2 \qquad \forall n \geq n_0$$

- But we can divide both sides of the above inequality by n^2 and we have

$$n \leq \frac{c_2}{6}$$

which is not true for any arbitrarily large n, since c_2 is a constant.

$\square$

Example from Book CLRS, page 46

Examples of Ω

Example

$f(n) = 2n^2 + 4n$

Is $f(n) = \Omega(n)$?

Examples of Ω

Example

$f(n) = 2n^2 + 4n$

Is $f(n) = \Omega(n)$?

To proof this, $g(n) = n$ and we have to show
$\boldsymbol{0 \leq c \times g(n) \leq f(n)}$ **for** $\boldsymbol{n > n_0}$

$n_0 = 1$ and $c = 2$
$0 \leq 2 \times 1 \leq 2 + 4$

Examples of Θ

Example

$f(n) = 2n^2 + 4n$

Is $f(n) = \Theta(n^2)$?

Examples of Θ

Example

$f(n) = 2n^2 + 4n$

Is $f(n) = \Theta(n^2)$?

To proof this, $g(n) = n^2$ and we have to show
$\mathbf{0 \le c_1 \times g(n) \le f(n) \le c_2 \times g(n)}$ **for** $\mathbf{n > n_0}$

$n_0 = 1$ and $c_1 = 2$ and $c_2 = 8$
$0 \le 2 \le 6 \le 8$

Examples of O

Claim is that $f(n) = 2^{3+n} = O(2^n)$

Example

To proof this, $g(n) = n^2$ and we have to show
$\boldsymbol{f(n) = 2^{3+n} \leq c_{\times} 2^n}$ **for** $\boldsymbol{\forall n > n_0}$

Examples of O

Claim is that $f(n) = 2^{3+n} = O(2^n)$

Example

To proof this, $g(n) = n^2$ and we have to show

$\boldsymbol{f(n) = 2^{3+n} \leq c_{\times} 2^n}$ **for** $\boldsymbol{\forall n > n_0}$

$f(n) = 2^{3+n} = 2^3 \times 2^n = 8 \times 2^n$

$n_0 = 1$ and $c = 8$

$0 \leq 8 \leq 8$

Example

Claim is that $f(n) = an^2 + bn + c$ where a, b and c are constants and $a > 0$ so that $f(n) = \Theta(n^2)$

Proof.

To proof the above, we can take the following constants.

$$c_1 = \frac{a}{4}$$
$$c_2 = \frac{7a}{4}$$
$$n_0 = 2 \times max\left(\frac{|b|}{a}, \sqrt{\frac{|c|}{a}}\right)$$

We can then verify that

$$0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2 \qquad \forall n \geq n_0$$

$\square$

In general for any polynomial $p(n) = \sum_{i=0}^{d} a_i n^i$, where we have set of constant coefficients a_i, and $a_d > 0$, we have $p(n) = \Theta(n^d)$

Little o notation.

We use little o-notation to denote an upper bound that is not asymptotically tight.

$$\boldsymbol{o(g(n)) = \{f(n)\text{: for any positive constant } c > 0 \text{ exist a constant } n_0 > 0 \text{ such that } 0 \le f(n) < c \times g(n) \text{ for all } n \ge n_0\}}$$

$$\forall c > 0, \exists n_0 > 0$$
$$o(g(n)) = \{f(n) : 0 \leq f(n) < cg(n)\}$$

Example

Is $2n = o(n^2)$ little o ?

The bound of $2n^2 = O(n^2)$ big O is asymptotically tight.

$2n = o(n^2)$ little o

But not $2n^2 \neq o(n^2)$

Little ω notation

$$\omega(g(n)) = \{f(n) : \text{for any positive constants } c > 0 \\ \text{there exists a constant } n_0 > 0 \text{ such that} \\ \boldsymbol{0 \leq c \times g(n) < f(n)} \text{ for all } n \geq n_0\}$$

Little ω notation - Examples

$$\omega(g(n)) = \{f(n) : \text{for any positive constants } c > 0 \\ \text{there exists a constant } n_0 > 0 \text{ such that} \\ \boldsymbol{0 \leq c \times g(n) < f(n)} \text{ for all } n \geq n_0\}$$

Example

$f(n) = \frac{n^2}{2}$ vs. $g(n) = n$

$\boldsymbol{\frac{n^2}{2} = \omega(n)}$

Example

$f(n) = \frac{n^2}{2}$ vs. $g(n) = n^2$

$\frac{n^2}{2} \neq \omega(n^2)$

Comparing functions

We can apply many of real number relations to asymptotically function comparisons, like we can apply:

- Transitivity,
- Reflexivity,
- Symmetry,
- Transpose symmetry

Comparing functions

We can apply many of real number relations to asymptotically function comparisons, like we can apply:

- Transitivity,
- Reflexivity,
- Symmetry,
- Transpose symmetry

Transitivity

$$f(n) = \Theta(g(n)) \land g(n) = \Theta(h(n)) \implies f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \land g(n) = O(h(n)) \implies f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \land g(n) = \Omega(h(n)) \implies f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \land g(n) = o(h(n)) \implies f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \land g(n) = \omega(h(n)) \implies f(n) = \omega(h(n))$$

Comparing functions

Reflexivity.

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Reflexivity.

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Symmetry.

$$f(n) = \Theta(g(n)) \equiv g(n) = \Theta(f(n))$$

Note: The sign $\equiv$ means "if and only if" or equivalent

Comparing functions

Reflexivity.

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Symmetry.

$$f(n) = \Theta(g(n)) \equiv g(n) = \Theta(f(n))$$

Note: The sign $\equiv$ means "if and only if" or equivalent

Transpose Symmetry.

$$f(n) = O(g(n)) \equiv g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \equiv g(n) = \omega(f(n))$$

Common Functions

Function Name	**Big O**
Constant	$O(c)$
Linear	$O(n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$
Logarithmic	$O(log(n))$
Log Linear	$O(nlog(n))$

Table: Examples of Big-O Function names

Simple Loop

```
def hasIt(myList, x):

for i in myList:
    if(i == x):
        return True

return False

>>> print(hasIt([4,1,2,3,10], 12))
False
```

Listing 1: Example of Linear Time Python Function

Search an item in two lists

```
def isInOneOfThem(myList1, myList2, x):
'''
This function searches a value in both of the
    given lists
'''

for i in myList1:
    if(i == x):
        return True


for i in myList2:
    if(i == x):
        return True

return False

>>> print(isInOneOfThem([4,1,2,3,10],
    [5,1,0,13,110], 12))
False
```

Listing 2: Search an item in two lists

Two Nested Loops

```
def haveIntersection(a1, a2):
    '''
    Returns true if the intersection of the two list is not empty
    '''
    for i in a1:
        for j in a2:
            # equal or not
            if(a1[i] == a2[j]):
                return True

    return False

>>> print(haveIntersection([4,1,2,3], [5,1,0,13]))
True
>>> print(haveIntersection([4,1,2,3], [11,7,8,9]))
False
```

Listing 3: Is intersection of two lists empty?

3 Nested Loops

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            k = 2 + 2
```

Listing 4: What about this code?

Intersection of two Lists

```
def haveIntersection(myList1, myList2):
'''
Returns true if the intersection of the two list is not empty
'''
for i in myList1:
    for j in myList2:
        if(i==j):
            return True

return False

>>> print(haveIntersection([4,1,2,3], [5,1,0,13]))
True
>>> print(haveIntersection([4,1,2,3], [11,7,8,9]))
False
```

Listing 5: Is intersection of two lists empty?

```
def find_Intersection(a, b):
    for i in a:
        for j in b:
            if(i==j):
            yield i


a=[1,2,3,4,5,6]
b=[5,6,7,8,9,10]

>>> for value in find_Intersection(a, b):
>>>     print(value)
5
6
```

Listing 6: Find intersection of two lists?

$O(log(n))$ Example

How many times you can divide by 2?

```
def doIt(x):
    '''
    O(log(n)) Example
    '''

    while (x > 0.01):
        print(x)
        x = x/2

    return x

>>> print(doIt(10))
10
5.0
2.5
1.25
0.625
0.3125
0.15625
0.078125
0.0390625
0.01953125
0.009765625
```

Listing 7: Big O of this code?

How many times the divide by 2 happens?

$$log_2(n) = k$$
$$2^k = n$$

$O(log(n))$ Example - Binary Search

```python
import numpy as np
def bSearch(data, value):
    n = len(data)
    left = 0
    right = n - 1
    count = 0

    while left <= right:
        middle = (left + right) // 2
        print("Count: ", count)
        count +=1
        if value < data[middle]:
            right = middle - 1
        elif value > data[middle]:
            left = middle + 1
        else:
            return middle
    print("Not in the list")

# We have a list like the following data, find the index of it.
myList = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(bSearch(myList, 6))

myList2 = list(np.random.randint(1000000, size=1000))
print(bSearch(myList2, 2341))
```

Listing 8: $O(log(n))$ Example

log(2n) Example

```
def my_log2N(n):
count = 0
i = 1

while(i < n):
    print(i)
    count+=1
    i = i * 2

return count

>>> print("Result: ", my_log2N(100))
1
2
4
8
16
32
64
Result:  7
```

Listing 9: log(2n) example

What is the Big O of the following code?

```
for i in range(0, n):
    count = 0
    x = i * n
    print("i, x:", i, x)
    for j in range(0, x):
        count += 1
    print("count", count)
```

Check how many times the inner loop runs.

What is the Big O of the following code?

```
for i in range(0, n):
    count = 0
    x = i * n
    print("i, x:", i, x)
    for j in range(0, x):
        count += 1
    print("count", count)
```

Check how many times the inner loop runs.

$$f(n) = 0 + n + 2n + 3n + 4n + ... + n(n-1)$$
$$= n(0 + 1 + 2 + 3 + 4... + (n-1))$$

$\sum_{k=1}^{n} = \frac{1}{2}n(n+1)$, it is $\Theta(n^2)$
$f(n) = O(n^3)$

Code Examples on Colab

Code Examples are shared on Colab here:
`https://colab.research.google.com/drive/18sCo48wvDBe5lldUmG7Y2MGb3yB4NtnQ?usp=sharing`

Readings from CLRS Book (Introduction to Algorithms, 3rd Edition)

- Chapters 1, and 2
- Sec. 2.2 - Introduction
- Sec. 1.2 - Analysis of insertion sort
- Sec. 1.2 - Growth of Functions
- Sec. 3.1 - Asymptotic notation
- Sec. 3.1 and 3.2 Standard notations and common functions
- Algorithms Illuminated: Part 1: The Basics
- `http://www.algorithmsilluminated.org/`