Big $\mathcal{O}$ notation

Ulises Tirado Zatarain

May, 2016

A little problem: What kind of algorithm is that?

```
/* uli is an abbreviation for the type:
   unsigned long int */
function do_something_weird(uli $x$, uli y): uli
    uli result = 0;
    while x != 0 do:
        if x & 1 == 1 then:
            result = result + y;
        end
        x = x >> 1;
        y = y << 1;
    end
    return result;
end
```

Definitions

- Big $\mathcal{O}$ time/space is the lenguage and metric we use to describe the efficiency of algorithms.
- Imagine the following scenario:
  - You got a file on a hard drive and you need to send it to a friend who lives across the country. You need to get the file to your friend as fast as possible. How should you send it?
  - E-mail
  - FTP, HTTP, SCP, ...
  - Dropbox, Google Drive, Sky Drive, ...
  - Airplane
  - What if the file was really large?

Definitions

- The ammount of elemental operations or stored information to solve a problem we know as “Computational Complexity”
- The are two kinds of complexity:
- Runtime (temporal complexity)
- Memory (spatial complexity)
- Big $\mathcal{O}$ notation allow us classify our algorithms in categories based-on input size.

Definitions

- The ammount of elemental operations or stored information to solve a problem we know as "Computational Complexity"
- The are two kinds of complexity:
- Runtime (temporal complexity)
- Memory (spatial complexity)
- Big $\mathcal{O}$ notation allow us classify our algorithms in categories based-on input size.

Definitions

- The ammount of elemental operations or stored information to solve a problem we know as "Computational Complexity"
- The are two kinds of complexity:
- Runtime (temporal complexity)
- Memory (spatial complexity)
- Big $\mathcal{O}$ notation allow us classify our algorithms in categories based-on input size.

Definitions

- The ammount of elemental operations or stored information to solve a problem we know as "Computational Complexity"
- The are two kinds of complexity:
- Runtime (temporal complexity)
- Memory (spatial complexity)
- Big $\mathcal{O}$ notation allow us classify our algorithms in categories based-on input size.

Definitions

- The ammount of elemental operations or stored information to solve a problem we know as “Computational Complexity”
- The are two kinds of complexity:
- Runtime (temporal complexity)
- Memory (spatial complexity)
- Big $\mathcal{O}$ notation allow us classify our algorithms in categories based-on input size.

Notation $\mathcal{O}(\cdot)$: categories

- Those categories are:
  - $\mathcal{O}(1)$, $\mathcal{O}(n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$, $\mathcal{O}(n^4)$, ...
  - $\mathcal{O}(\log n)$, $\mathcal{O}(n \log n)$, $\mathcal{O}(n^2 \log n)$, ...
  - $\mathcal{O}(n!)$, $\mathcal{O}(2^n)$, ...
  - What the hell does this mean? $\mathcal{O}$.o o.$\mathcal{O}$ @-@

Definitions and notation

- For example, to send a file to a friend the run time is:
  - By e-mail, FTP, Dropbox is $\mathcal{O}\left(s\right)$ where s is the file size.
  - By airplane is $\mathcal{O}\left(1\right)$.
  - But, maybe is $O\left(2^{n}\right)$ in money, where n ...? WTF! Why?
- Rule: Drop the constants
  - What?
  - Why?
  - Example: seems that to send by e-mail is indeed at least $2s$ in time where s is the file size.

Best, Worst and Average/Expected cases

- Think about search an element in an unsorted simply linked list:
  - Best case: the element is at the start of the list. (Maybe you think this is $\mathcal{O}(1)$).
  - Worst case: the element is at the end of the list. (Maybe you think this is $\mathcal{O}(n)$).
  - Average/Expected case: the element is in any position at the list. (What do you think about this?)

Examples: Print elements in arrays A and B

$\mathcal{O}(a+b)$: where a and b are the size of array A and B respectively.

```
for each x in A:
    print(x);
end
for each y in B:
    print(y);
end
```

Examples: Print elements in arrays A and B

$\mathcal{O}(ab)$: where a and b are the size of array A and B respectively.

```
for each x in A:
    for each y in B:
        print(x,y);
    end
end
```

Examples: Print elements in arrays A

$\mathcal{O}\left(n^2\right)$: where n is the size of array A.

```
for i = 0 to n − 1 do:
    for j = i to n − 1 do:
        print(A[j]);
    end
end
```

```
A[0], A[1], A[2],  ...  , A[n−1]
A[1], A[2],  ...  , A[n−1]
A[2],  ...  , A[n−1]
.
.
.
A[n−1]
```

What does following function? What is the run time?

```
function something(n: integer): integer
    x = 0;
    while x * x < n do:
        x = x + 1;
    end
    return x;
end
```

References

- Gayle Laakmann - Cracking the Coding Interview
- Robert Sedgewick - Algorithms C++
- Thomas H. Cormen - Introduction to Algorithms
- Donald E. Knuth - The Art of Computer Programming
- Wikipedia
- Quora