Extended Introduction to Computer Science
CS1001.py

Module C More on Complexity

Instructors: Amir Rubinstein, Michal Kleinbort

Teaching Assistants: Noam Parzanchevsky,
Hussen Abu Hamad, Asaf Cassel, Shaked Dovrat

School of Computer Science
Tel-Aviv University
Fall Semester 2020-21
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

Defining Complexity - Reminder

- We will be interested in how the number of operations changes with input size.
- In most cases, we will not care about the exact function, but in its "order", or growth rate (e.g., logarithmic, linear, quadratic, etc.)
- Sometimes we will only be interested/able to give an upper bound for this growth rate. We will, however, strive to make this upper bound as tight (=low) as we can.
  - In this course, we will almost always be able to give tight upper bounds.
- We need some formal definition for "upper bound for the number of operations growth rate, as a function of input size".

Big O Notation - Reminder

- We say that a function $f(n)$ is $O(g(n))$ if there is a constant c such that for large enough n,

$$|f(n)| \leq c \cdot |g(n)|$$

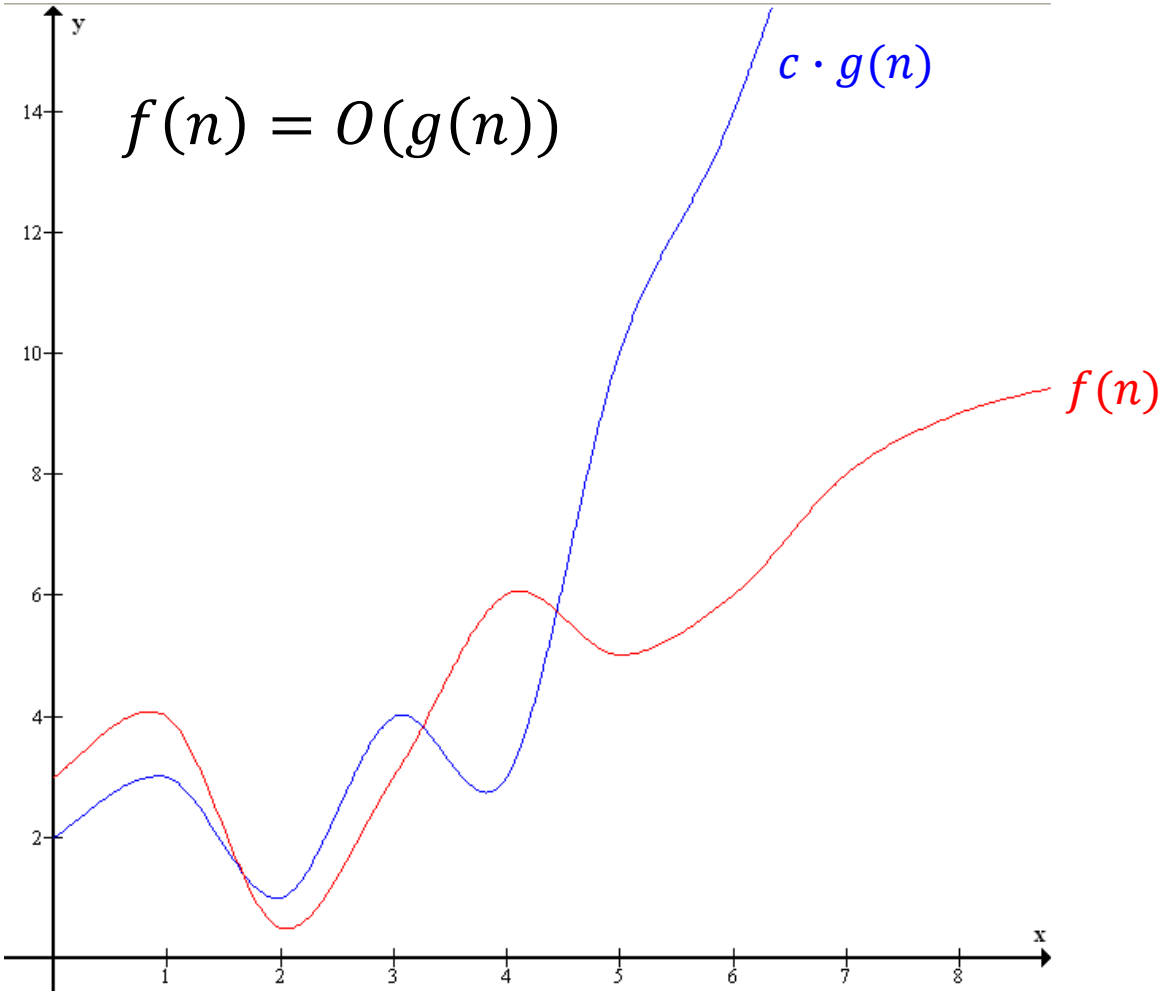
- We denote this as $f(n) = O(g(n))$

- In our context, $f(n)$ will usually denote the number of operations an algorithm performs on an input of size n
  - a number with n bits
  - a collection with n elements (list, string, etc.)

- sometimes $f(n)$ will denote the number of memory cells required by the algorithm on an input of size n

- So in our context f and g are positive functions for every n, and so we will omit the absolute value notation.

Big O Notation – Visualized



Previous Results - Reminder

- All these results refer to worst case scenarios.

- Algorithms on sequences:
  - Binary search on a sorted list of length n takes $O(logn)$ iterations
  - Selection Sort on a list of length n takes $O(n^2)$ iterations
  - Merging of 2 sorted lists of sizes n and m takes $O(n+m)$ iterations
  - Palindrome checking on a string of length n takes $O(n)$ iterations

- Algorithms on integers:
  - Addition of two n-bit integers takes $O(n)$ iterations
  - Multiplication of two n-bit integers takes $O(n^2)$ iterations

Tractability - Basic Distinction

How would execution time for a very fast, modern processor (10^{10} ops per second, say) vary for a task with the following time complexities and n = input sizes?

	10	20	30	40	50	60
n	1.0E-09 seconds	2.0E-09 seconds	3.0E-09 seconds	4.0E-09 seconds	5.0E-09 seconds	6.0E-09 seconds
n^2	1.0E-08 seconds	4.0E-08 seconds	9.0E-08 seconds	1.6E-07 seconds	2.5E-07 seconds	3.6E-07 seconds
n^3	1.0E-07 seconds	8.0E-07 seconds	2.7E-06 seconds	6.4E-06 seconds	1.3E-05 seconds	2.2E-05 seconds
n^5	1.0E-05 seconds	0.00032 seconds	0.00243 seconds	0.01024 seconds	0.03125 seconds	0.07776 seconds
2^n	1.02E-07 seconds	1.05E-04 seconds	0.107 seconds	1.833 minutes	1.303 days	0.64 years
3^n	5.9E-06 seconds	0.35 seconds	5.72 hours	38.55 years	22764 centuries	1.34E+09 centuries

Modified from Garey and Johnson's classical book

Polynomial time = tractable. Exponential time = intractable.

Time Complexity - What is tractable in Practice?

- A polynomial-time algorithm is good.
  - n^{100} is polynomial, hence good.
- An exponential-time algorithm is bad.
  - $2^{n/100}$ is exponential, hence bad.

Yet for input of size n = 4000, the n^{100} time algorithm takes more than 10^{35} centuries on the above mentioned machine, while the

$2^{n/100}$ algorithm runs in just under two minutes.

Time Complexity - Advice

- Trust, but check! Don't just mumble "polynomial-time algorithms are good", "exponential-time algorithms are bad" because the lecturer told you so.
- Asymptotic run time and the O notation are important, and in most cases help clarify and simplify the analysis.
- But when faced with a concrete task on a specific problem size, you may be far away from "the asymptotic".
- In addition, constants hidden in the O notation may have unexpected impact on actual running time.

Input Size - Clarifications

- We measure running time (or computational complexity) as a function of the input size.

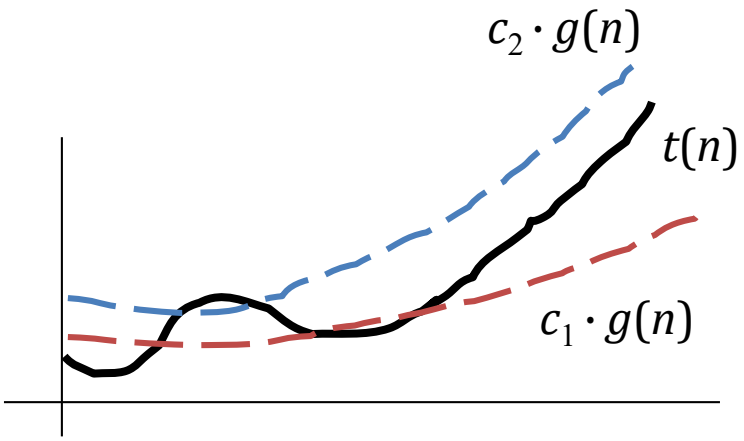
- For integers, input size is the number of bits in the representation of the number in the computer.
  - we normally count the number of "simple" bit operations (such as adding or multiplying two bits).

- For lists/strings/dictionaries/other collections, the input size is typically the number of elements in the collection.
  - We normally count the number of "simple" list element operations (such as comparisons, assignments), assuming that the size of each element is bound by some constant (therefore no need to consider operations on them, such as comparison, addition, etc.)
  - There are exceptions to this, however.
    For example, a list of n string each of size m.

Tight Bound - Θ

- We say that a function $f(n)$ is $\Theta(g(n))$ if there are two constant c_1,c_2 such that for large enough n,

$$c_1 \cdot |g(n)| \leq |f(n)| \leq c_2 \cdot |g(n)|$$



- $f(n) = \Theta(g(n))$ IFF $f(n) = O(g(n))$ and $g(n) = O(f(n))$
- It is very common to use O instead of Θ, but formally O is merely an upper bound

כמה טיפים חשובים

מעקב אחר לולאות:

שני מבני לולאות:

בטור (או, סדרתי):

for i in range($f_1(n)$):

    $O(1)$

for i in range($f_2(n)$):

    $O(1)$

מתקיים:

$$f_1(n) + f_2(n) = O\big(g_1(n) + g_2(n)\big)$$

וגם:

$$f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$$

מעקב אחר לולאות:

שני מבני לולאות:

בטור (או, סדרתי):

$$f_1(n) + f_2(n) = O\big(g_1(n) + g_2(n)\big) = O(\max\{g_1(n), g_2(n)\})$$

מקונן:

ללא תלות פנימית:

for i in range($f_1(n)$):
    for j in range($f_2(n)$):
        $O(1)$

מתקיים:

$$f_1(n) \cdot f_2(n) = O\big(g_1(n) \cdot g_2(n)\big)$$

הוכיחו!

מעקב אחר לולאות:

שני מבני לולאות:

בטור:

$$f_1(n) + f_2(n) = O\big(g_1(n) + g_2(n)\big) = O(\max\{g_1(n), g_2(n)\})$$

מקונן:

ללא תלות פנימית: $f_1(n) \cdot f_2(n) = O\big(g_1(n) \cdot g_2(n)\big)$

עם תלות פנימית:

for i in range($f_1(n)$):

for j in range($\boldsymbol{f_2(i)}$):

$O(1)$

בשונה מלולאות מקוננות ללא תלות פנימית, במקרה זה נצטרך לחשב את הסכום:

$$O\left(\sum_{i=0}^{f_1(n)} f_2(i)\right)$$

לולאות מקוננות עם תלות פנימית

נסתכל על הקוד הבא שמדגים מבנה של לולאות מקוננות תלויות:

for i in range($log(n)$):
    for j in range(2^i):
        $O(1)$

מכפלה של החסמים מלמעלה של שתי הלולאות לא תהווה חסם מדוייק (הדוק) מספיק.

נדגים:

הלולאה החיצונית רצה $\log n$ איטרציות. הלולאה הפנימית רצה לכל היותר n איטרציות (המקסימום מתקבל כאשר $i = \log n$). מכפלה היתה נותנת לנו $2^{\log n}$ חסם של $O(nlog\ n)$ על זמן הריצה. זהו חסם עליון נכון, אך לא הדוק מספיק.

עשוי להתקבל חסם הדוק (נמוך) יותר אם נחשב את הסכום ישירות:

$$\Sigma_{i=0}^{\log n}(2^i) = \frac{1(2^{\log n} - 1)}{2 - 1} = O(n)$$