WORKSHEET WEEK 01: ASYMPTOTIC BOUNDS

1.1 Introduction

Goal: The focus of this course is efficiency – creating algorithms that can work on large input data and handle complex structures sufficiently fast.

Why use Big-O Notation? It is convenient to measure speed of algorithms – for example, to find the best algorithms for a given problem. Or to find out which problems are easy (have fast algorithms) and which ones are hard (have only slow or unfeasible algorithms).

- Measuring the speed should not depend on the speed of the computing hardware – do not care about constant factors.
- Measuring the speed should not depend on how fast it works on very short inputs. (One can "cheat" for short inputs – just remember a large lookup table containing values for inputs of length $n < n_0$ with precomputed correct answers. Clearly, this does not tell us anything about the performance of this algorithm for arbitrary inputs.)
- Measuring the speed should be conceptually easy, it should not take into account insignificant optimizations or count too many extra factors.

Example: Energy needed to lift a stone of mass m to the height h is mgh. (Is this the best-case estimate? The worst-case estimate? The exact value?)

1.1.1 Running Time as a Complexity Measure

The Worst Running Time function: Given an algorithm, denote by $T(n)$ the number of elementary steps that are needed to complete the algorithm for any input of length n. (It can be called the *upper bound* of the running time.)

Discussions on the Worst-Case Running Time:

- Is $T(n)$ the upper bound also for inputs shorter than n?
- Is $T(n)$ a non-decreasing function (i.e. do longer inputs always imply a longer running time)?
- What counts as an elementary step? (Any CPU instruction? One line in a pseudocode? One comparison in a sorting algorithm?)
- Let $T(n)$ be a numeric algorithm receiving single natural numbers as input. Does the worst running time function change, if the input numbers are provided in binary (instead of decimal) notation?
- What is the worst running time to multiply two square matrices of size $n \times n$? What is the size of input in this case?

Sometimes it is common to have n as some important parameter of the input data (not necessarily the exact size of its encoding). For matrix tasks – the size of the matrices n. For graph problems – the number of vertices n and the number of edges m.

1.1.2 Definitions

Definition of Big-O: Let $g\colon \mathbb{N} \to \mathbb{R}_{0+}$ be a function from natural numbers (non-negative integers) to non-negative real numbers. Then $O(g)$ is the set of all functions $f\colon \mathbb{N} \to \mathbb{R}^{0+}$ such that there exist real constants $c > 0$ and $n_0 \in \mathbb{N}$ satisfying

$$0 \leq f(n) \leq c \cdot g(n) \;\text{ for all }\; n \geq n_0.$$

Definition of Big-Omega: Let $g\colon \mathbb{N} \to \mathbb{R}_{0+}$ be a function from natural numbers to non-negative real numbers. Then $\Omega(g(n))$ is the set of all functions $f\colon \mathbb{N} \to \mathbb{R}$ such that there exist real constants $c > 0$ and $n_0 \in \mathbb{N}$ satisfying $\forall n \in \mathbb{N}\,\big(n \geq n_0 \to f(n) \geq c \cdot g(n)\big)$.

Definition of Big Theta: Let $g\colon \mathbb{N} \to \mathbb{R}_{0+}$ be a function from natural numbers to non-negative real numbers. Then $\Theta(g)$ is the set of all functions $f : \mathbb{N} \to \mathbb{R}$ such that there exist positive constants $c_1, c_2 > 0$ and $n_0 \in \mathbb{N}$ satisfying

$$\forall n \in \mathbb{N}\,\big(n \geq n_0 \to 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\big).$$

Informally, the following terms are also usable:

- If $f(n) \in O(g(n))$, then $g(n)$ is called *asymptotic upper bound* of $f(n)$.
- If $f(n) \in \Omega(g(n))$, then $g(n)$ is called *asymptotic lower bound* of $f(n)$.
- If $f(n) \in \Theta(g(n))$, then $g(n)$ is called *asymptotic growth order* of $f(n)$.

All these concepts (Big-O, Big-Omega, Big-Theta) are related to calculus (real analysis); it is functional behavior as $n \to \infty$. Predicting the speed of an algorithm for short input lengths n, the dependence on n is typically quite complex (and we cannot ignore "lower order" terms). As n becomes very large, only the "dominant parts" in the expression $f(n)$ matter.

1.2 Properties of Big-O, Big-Omega, Big-Theta

Big-O and Limit of the Ratio: If the following limit exists and is finite:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C < +\infty,$$

then $f(n)$ is in $O(g(n))$.

Big-O is transitive: If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$.

Sum of two functions: If $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$, then $f(n) + g(n) = O(h(n))$.

All polynomials: Any k-th degree polynomial $P(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ is in $O(n^k)$.

Logarithms of any base: If $a, b > 1$ are any real numbers, then $\log_a n = O(log_b\, n)$. Typically use just one base (usually, it is base 2 or base e of the natural logarithm, if you prefer that), and write just $O(\log\ n)$ without specifying base at all.

The last result directly follows from the formula to change the base of a logarithm: $\forall a, b, m > 1 \left(\log_a b = \dfrac{\log_m b}{\log_m a}\right)$.

1.3 Problems

Problem 1: Show using the above definition of $O(g)$ the following facts:

(A) $f(n) = 13n + 7$ is in $O(n)$. (Formally, $f \in O(g)$, where $g(n) = n$.)

(B) $f(n) = 3n^2 - 100n + 6$ is in $O(n^2)$.

(C) $f(n) = 3n^2 - 100n + 6$ is in $O(n^3)$.

(D) $f(n) = 3n^2 - 100n + 6$ is **not** in $O(n)$.

Problem 2: Let us have a zero-based dictionary D with n items from $D[0]$ to $D[n-1]$.

```
LinearSearch(D, w)
    for i in range(0, n):
        if w == D[i]:
            return FOUND w at location i
    return NOT FOUND
```

Problem 3: What is the worst running time to find, if the given input m is a prime number? (Primality testing is done by dividing the input with all numbers $2, 3, \ldots, \lfloor \sqrt{m} \rfloor$ until m is found to be divisible by some number.)

Problem 4: Answer the following Yes/No questions:

(A) For any $g(n)$, is the set of functions $\Theta(g(n))$ the intersection of $O(g(n))$ and $\Omega(g(n))$?

(B) Does every function $f(n)$ belong to the set $Omega(1)$?

(C) Let $f(n), g(n)$ be two functions from natural numbers to non-negative real numbers. Is it true that we have either $f(n)$ in $O(g(n))$ or $g(n)$ in $f(n)$ (or both)?

(D) Does the definition of $f(n)$ in $O(g(n))$ make sense, if $f(n)$ and $g(n)$ can take negative values?

(E)

Let $f(n)$ be a function from natural numbers to non-negative real numbers. Do we always have that $f(n)$ is in $O(f(n))$, and $f(n)$ is in $\Omega(f(n))$ and $f(n)$ is in $\Theta(f(n))$? (In other words, is being in Big-O, in Big-Omega and in Big-Theta a reflexive relation?)

(F)

Let $f(n), g(n), h(n)$ be functions from natural numbers to non-negative real numbers. It is known that $f(n)$ is in $O(g(n))$ and also $g(n)$ is in $h(n)$. Can we always imply that $f(n)$ is in $O(h(n))$. (In other words, is being in Big-O, in Big-Omega and in Big-Theta a transitive relation?)

(H)

Let $f(n), g(n)$ be functions from natural numbers to non-negative real numbers. It is known that $f(n)$ is in $\Theta(g(n))$. Can we always imply that $g(n)$ is in $\Theta(f(n))$? (In other words, is being in Big-Theta an equivalence relation?)

(I)

A function $f(n)$ is defined for natural arguments and takes natural values. It is known that $f(n)$ is in $O(1)$. Is it true that $f(n)$ is a constant function: $f(n) = C$ for all $n \in \mathbf{N}$.

Problem 5: Given a sequence a_i ($i = 0, \ldots, n-1$) we call its element a_i a *peak* iff it is a local maximum (at least as big as any of its neighbors):

$$a_i \geq a_{i-1} \ \textbf{and} \ a_i \geq a_{i+1}$$

(In case if $i = 0$ or $i = n-1$, one of these neighbors does not exist; and in such cases we only compare a_i with neighbors that do exist.)

(A) Suggest an algorithm to find some peak in the given array $A[0], \ldots, A[n-1]$ and find its worst-case running time.

(B) Suggest an algorithm that is faster than linear time to find peaks in an array. Namely, its worst-case running time should satisfy the limit:

$$\lim_{n \to \infty} \frac{T(n)}{n} = 0.$$

Problem 6: Order these functions in increasing order regarding Big-O complexity (f_i is considered "not larger" than f_j iff $f_i \in O(f_j)$.

- $f_1(n) = n^{0.9999} \log_2 n$
- $f_2(n) = 10000n$
- $f_3(n) = 1.0001^n$
- $f_4(n) = n^2$

Problem 7: Order these functions in increasing order regarding Big-O complexity:

- $f_1(n) = 2^{2^{10000}}$
- $f_2(n) = 2^{10000n}$
- $f_3(n) = \binom{n}{2} = C_n^2$
- $f_4(n) = \binom{n}{\lfloor n/2 \rfloor}$
- $f_5(n) = \binom{n}{n-2}$
- $f_6(n) = n!$
- $f_7(n) = n\sqrt{n}$

Problem 8: Order these functions in increasing order regarding Big-O complexity:

- $f_1(n) = n^{\sqrt{n}}$
- $f_2(n) = 2^n$
- $f_3(n) = n^{10} \cdot 2^{n/2}$
- $\displaystyle\sum_{i=1}^{n} (i+1).$

Problem 9: A black box $\mathcal{B}$ receives two numbers $k_1, k_2 \in \{1, \ldots, n\}$ as inputs and returns a value $v = \mathcal{B}(k_1, k_2)$. What is the worst-case time complexity to find the maximum possible value $v = \mathcal{B}(k_1, k_2)$ for any two inputs.

What if the black box receives permutations of n elements as its inputs?

Problem 10: Prove or disprove the following statement: If $f(n)$ is in $O(g(n))$ and also $g(n)$ is in $O(f(n))$, then $f(n)$ is also in $\Theta(g(n))$ (and $g(n)$ is in $\Theta(f(n))$. (You can assume that $f(n)$ and $g(n)$ always take positive values.)