Computational Complexity

ICT Officers

September 2017

1 Introduction

Computational Complexity is a measure of algorithmic efficiency. It is defined as the amount of time *(time complexity)* or space *(space complexity)* an algorithm takes as a function of input size.

2 Big-O Asymptotic Complexity

We will use Big-O notation describe the efficiency of an algorithm. It is an upper-bound for the algorithm's computational complexity. Given that the function $f(N)$ is the exact complexity of a particular algorithm of input size N, there are two main rules for using Big-O notation:

1. Consider only the term with the highest degree
2. If $f(N)$ is a product of several factors, any constants are omitted

Example: the complexity function of $f(N) = \frac{1}{8}N^3 + 2N^2 + 230N$ can be represented in Big-O notation as O(N^3)

3 Common Complexities

complexity	**order of growth**	**description**	**example**
constant	1	statement	add two numbers
logarithmic	$\log N$	divide in half	binary search
linear	N	loop through all numbers	find maximum
linearithmatic	$N \log N$	divide and conquer	mergesort
quadratic	N^2	double loop	form all pairs
cubic	N^3	triple loop	form all triplets
exponential	2^N	exhaustive search	recursive fibonacci

4 Contest Cheat Sheet

In USACO, for each test case, you are given 2 second for C++ and 4 seconds for Java and Python. Your programs are run on machines that do approximately 10^6 extensive or 10^7 trivial operations per second. Based on the input size bounds given to you, here are around the complexities your programs should be:

- $N \leq 10$: O($N!$)
- $N \leq 25$: O(2^N)
- $N \leq 500$: O(N^3)
- $N \leq 5000$: O(N^2)
- $N \leq 100000$: O($N \log N$)
- $N \leq 1000000$: O(N)