

Computer Science 2

Data Structures and Algorithms

CS 102

Intro to “big o”

Lists

Professor: Evan Korth

New York University

Road Map for Today

- A cursory introduction to Big Oh
- Abstract Data Types
- Lists
 - Array implementation
 - Linked Lists
- Reading: Asymptotic Analysis: Chapter 4

Abstract Data Type

- An abstract data type (ADT) is a set of objects together with a set of operations. For example:
 - Stacks
 - Queues
 - Dictionary
 - Trees
 - Priority queue

Analyzing Data Structures

- In order to discuss what makes a good data structure, we need a way to analyze the execution behavior of a data structure.
- Two of the most important considerations we need to keep in mind are:
 - Time complexity
 - Space complexity
- There are other important considerations such as code complexity but we will concentrate on time and space for now.

Time Complexity

- How long will the program take:
 - For each of the data structure's operations we want to know how many instructions must be executed.
 - We don't need to know the exact number of instructions because that depends on the compiler and instruction set of our environment.
 - In other words, we want this system to be machine and compiler independent.
 - For small sets of data, it does not matter much.
 - We need a theoretical system to quantify how the execution time grows as our data size grows.

Time Complexity

- For each of the data structure's operations we want to know how many instructions must be executed.
- Perhaps we can determine a program will execute a maximum of $4n^3 + 4n^2 + 87n + 15$ instructions on a pc for a dataset of size n.
- Do we need to be so precise if we are only concerned with the growth rate of our program's execution time?

Asymptotic analysis

- Asymptotic analysis is based on the idea that as the problem size grows, the complexity can be described as a simple proportionality to some known function. This idea is incorporated in the "Big Oh" notation for asymptotic performance.
- **Definition:** $T(n) = O(f(n))$ if and only if there are constants c_0 and n_0 such that $T(n) \leq c_0 f(n)$ for all $n \geq n_0$.
- The expression " $T(n) = O(f(n))$ " is read as "T of n is in Big Oh of f of n." Big Oh is sometimes said to describe an "upper-bound" on the complexity. Other forms of asymptotic analysis ("Big Omega", "Little Oh", "Theta") are similar in spirit to Big Oh, but will not be discussed in this class.

Big O: What does it mean in English?

- Big O defines the upper bound of the running time of an operation or algorithm without regard to constants or machine / compiler factors.
- We just throw away leading constants and low order terms.

Some common functions which describe growth rates (in order)

- $O(1)$: constant
- $O(\log n)$ logarithmic
- $O(n)$ linear
- $O(n \log n)$ linearithmic
- $O(n^2)$ quadratic
- $O(n^3)$ cubic
- $O(2^n)$ exponential

One quick example

- Linear search versus binary search:
- Linear search:
 - <https://yongdanielliang.github.io/animation/web/LinearSearchNew.html>
- Binary search
 - <https://yongdanielliang.github.io/animation/web/BinarySearchNew.html>
- What is the big oh for linear search?
- What is the big oh for binary search?
- We will discuss the time complexity for the operations of every ADT and algorithm we look at in this class.

List

- **List** (data structure)
- **Definition:** A collection of items accessible one after another beginning at the head and ending at the tail.
- Head: The first item in a list
- Tail: The last item in a list

List operations (one possible example)

- printList
- makeEmpty
- find
- findKth
- Insert (at head of list)
- RemoveKth
- isEmpty
- first
- findPrevious

First idea: Use an array

- Need to estimate size for the array from the beginning. If we need to expand the array it is expensive (in Java) or impossible (in some other languages). This wastes considerable space.
- Also insertions and deletions can be expensive because we may need to move elements in order to execute the operation (on average half are moved – worse case all must be moved).

Better idea: linked list

- Basic idea is to create a node for each element. The node will contain the element as well as a pointer to the next node.
 - We need to keep track of the head and the tail somehow.
- This change means when we insert and delete nodes, all we need to do is move the pointers around.