Complexity Analysis

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

Lecture 02

sunho.lim@ttu.edu

Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

Algorithm vs. Data Structure

- Suppose we have two algorithms, how can we tell which one is better?
  - could implement both algorithms, run them both
    - expensive and error prone…
  - preferably, analyze them mathematically
    - ***Algorithm analysis***
- ***Algorithms*** ←→ **Data Structures**
  - data structures are implemented using algorithms
  - some algorithms are more efficient than others
  - how to measure efficiency?

Computational and Asymptotic Complexity

- ***Algorithm's complexity***
  - the efficiency of the algorithm in terms of the **amount of data** the algorithm must process
  - need metrics to compare
- Main complexity measures of efficiency:
  - **Time complexity**, the amount of **time** an algorithm takes in terms of the amount of input
  - **Space complexity**, the amount of **memory (space)** an algorithm takes in terms of the amount of input
- Algorithm's ***asymptotic* complexity**
  - when ***n* (number of input items)** goes to infinity, what happens to the algorithm's performance?

Computational and Asymptotic Complexity (cont.)

- e.g., $f(n) = n^2 + 100n + \log_{10}n + 1000$
- As the value of n increases, only the **n^2 term** is significant

n	f(n)	n^2		100n		$\log_{10}n$		1,000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1,000	90.83
10	2,101	100	4.76	1,000	47.6	1	0.05	1,000	47.60
100	21,002	10,000	47.6	10,000	47.6	2	0.001	1,000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1,000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1,000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1,000	0.00

Asymptotic Analysis

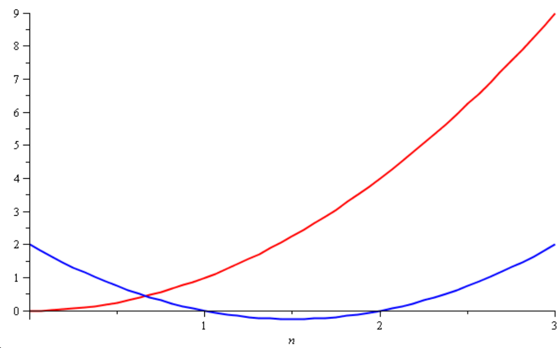
- Given an algorithm:
  - need to be able to describe these values mathematically
  - need **a systematic means** of using the description of the algorithm together with the properties of an associated data structure
  - need to do this in a **machine-independent way**
    - Supercomputer vs. PC ?
- Need to run the algorithm to measure running time (execution time)? But…
  - want to estimate running time without running the algorithm
  - **asymptotic notation**

Big-O Notation

- The most commonly used notation for asymptotic complexity
  - estimate the rate of function growth
  - e.g., $n^2 + 100n + \log_{10}n + 1000 = O(n^2)$
- **Definition**: Let $f(n)$ and $g(n)$ be functions, where n is a positive integer. We write $f(n) = O(g(n))$ if and only if there exists a real number c and positive integer N satisfying $0 \le f(n) \le cg(n)$ for all $n \ge N$.
- Examples:
  - $f(n) = 3n + 2$
  - $f(n) = 6 \times 2^n + n^2$

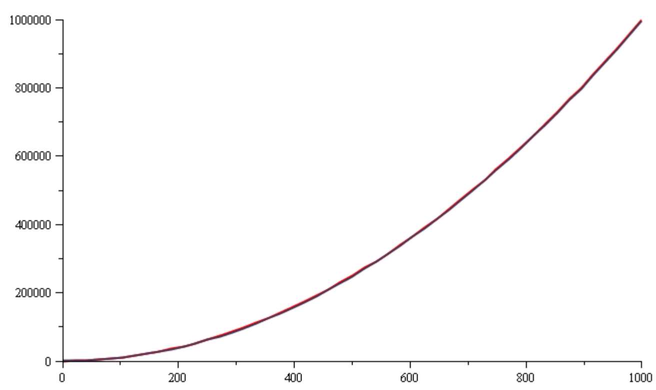
Quadratic Growth

- Consider the two functions
  - $f(n) = n^2$ and $g(n) = n^2 - 3n + 2$
  - Around $n = 0$, they look very different



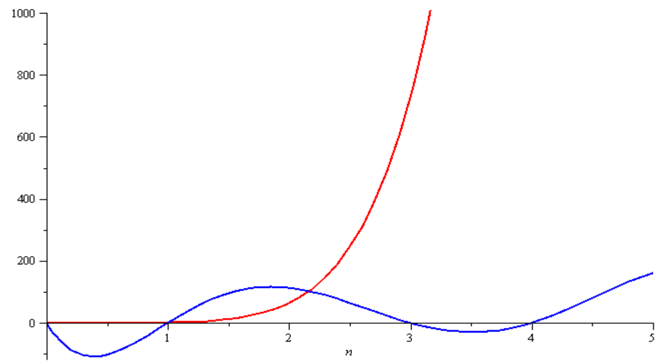
Quadratic Growth (cont.)

- Yet on the range $n = [0, 1000]$, they are (relatively) indistinguishable:



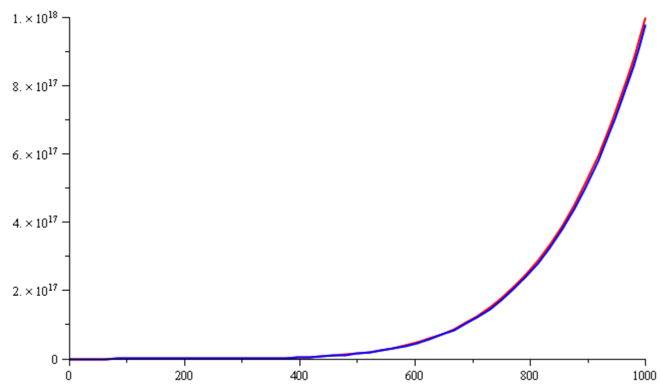
Polynomial Growth

- To demonstrate with another example,
  - $f(n) = n^6$ and $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$
  - Around $n = 0$, they are very different



Polynomial Growth (cont.)

- Still, around $n = 1000$, the relative difference is less than 3%



Big-O Notation (cont.)

- While c and N exist,
  - how to calculate them? or what to do?, if multiple candidates exist
- e.g., the function f:

$$f(n) = 2n^2 + 3n + 1$$

and g:

$$g(n) = n^2$$

- Clearly $f(n)$ is $O(n^2)$; **possible candidates** for c and N

c	≥ 6	$\geq 3\frac{3}{4}$	$\geq 3\frac{1}{9}$	$\geq 2\frac{13}{16}$	$\geq 2\frac{16}{25}$	...	$\rightarrow$	2
N	1	2	3	4	5	...	$\rightarrow$	∞

Big-O Notation (cont.)

- Solving the inequality from the definition of big-O:

$$f(n) \leq cg(n)$$

- Substituting for $f(n)$ and $g(n)$,

$$2n^2 + 3n + 1 \leq cn^2 \text{ or } 2 + 3/n + 1/n^2 \leq c$$

- Since $n \geq N$, and N is a positive integer, start with $N = 1$ and substitute in either expression to obtain c

c	≥ 6	$\geq 3\frac{3}{4}$	$\geq 3\frac{1}{9}$	$\geq 2\frac{13}{16}$	$\geq 2\frac{16}{25}$	...	$\rightarrow$	2
N	1	2	3	4	5	...	$\rightarrow$	∞

Big-O Notation (cont.)

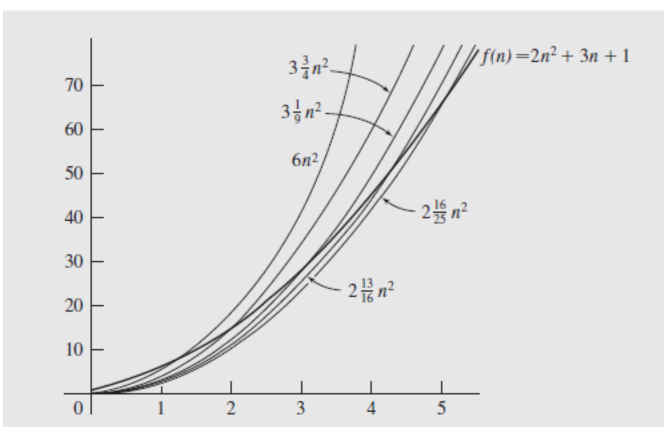
$f(n) = 2n^2 + 3n + 1$

- Generally, choose an N that allows **one term of f** to **dominate** the expression
  - only two terms to consider: **$2n^2$ and $3n$**, since the last term is a constant
  - as long as n is greater than 1.5, **$2n^2$** dominates the expression
  - N must be 2 or more, and c is greater than 3.75
- The choice of c depends on the choice of N and vice-versa

c	≥ 6	$\geq 3\frac{3}{4}$	$\geq 3\frac{1}{9}$	$\geq 2\frac{13}{16}$	$\geq 2\frac{16}{25}$	...	$\rightarrow$	2
N	1	2	3	4	5	...	$\rightarrow$	∞

Big-O Notation (cont.)

- Different values of c and N:



Examples of Complexities

- Classes of algorithms and their execution times
  - Use a computer executing 1 million operations per second

Class	Complexity	Number of Operations and Execution Time (1 instr/μsec)					
n		10		10^2		10^3	
constant	$O(1)$	1	1 μsec	1	1 μsec	1	1 μsec
logarithmic	$O(\lg n)$	3.32	3 μsec	6.64	7 μsec	9.97	10 μsec
linear	$O(n)$	10	10 μsec	10^2	100 μsec	10^3	1 msec
$O(n \lg n)$	$O(n \lg n)$	33.2	33 μsec	664	664 μsec	9970	10 msec
quadratic	$O(n^2)$	10^2	100 μsec	10^4	10 msec	10^6	1 sec
cubic	$O(n^3)$	10^3	1 msec	10^6	1 sec	10^9	16.7 min
exponential	$O(2^n)$	1024	10 msec	10^{30}	$3.17 * 10^{17}$ yrs	10^{301}	

15

Examples of Complexities (cont.)

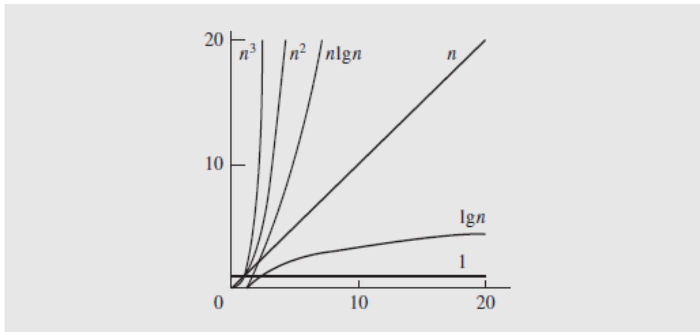
- The ***class*** of an algorithm based on big-O notation
  - a convenient way to describe its behavior
- e.g., a **linear function** is $O(n)$;
  - its time increases in direct proportion to the amount of data processed

n		10^4		10^5		10^6	
constant	$O(1)$	1	1 μsec	1	1 μsec	1	1 μsec
logarithmic	$O(\lg n)$	13.3	13 μsec	16.6	7 μsec	19.93	20 μsec
linear	$O(n)$	10^4	10 msec	10^5	0.1 sec	10^6	1 sec
$O(n \lg n)$	$O(n \lg n)$	$133 * 10^3$	133 msec	$166 * 10^4$	1.6 sec	$199.3 * 10^5$	20 sec
quadratic	$O(n^2)$	10^8	1.7 min	10^{10}	16.7 min	10^{12}	11.6 days
cubic	$O(n^3)$	10^{12}	11.6 days	10^{15}	31.7 yr	10^{18}	31,709 yr
exponential	$O(2^n)$	10^{3010}		10^{30103}		10^{301030}	

16

Examples of Complexities (cont.)

- Relationships expressed graphically:



- With today's supercomputers..
  - cubic order algorithms or higher are impractical for large numbers of elements

Finding Asymptotic Complexity

- Asymptotic bounds
  - used to determine the time and space efficiency of algorithms
  - generally, we are interested in **time complexity**!!
- Consider a simple loop:

```
for (i = sum = 0; i < n; i++)
    sum = sum + a[i]
```

  - In initialization, execute two assignments once
    - sum = 0 and i = sum
  - In the loop, n times
    - update sum (sum = sum + a[i]) and increment i (e.g., i++)
  - 2 + 2n assignments → O(n)   /* asymptotic complexity */

Finding Asymptotic Complexity (cont.)

- A nested loop case,
  - the complexity grows by a factor of *n*, although this isn't always the case
- Consider,

```
for (i = 0; i < n; i++) {
   for (j = 1, sum = a[0]; j <= i; j++)
      sum += a[j];
   cout << ”sum for subarray 0 through “ << i
        <<” is “<<sum<<endl;
}
```


Finding Asymptotic Complexity (cont.)

```
for (i = 0; i < n; i++) {
       for (j = 1, sum = a[0]; j <= i; j++)
          sum += a[j];
       cout << ”sum for subarray 0 through “ << i
              << ” is “ << sum << endl;
}
```

- In the outer loop, initialize i and execute n times
  - Increment i, and execute the inner loop and cout statement
- In the inner loop, initialize j and sum each time,
  - the number of assignments so far, 1 + 3n
  - execute i times, where i ranges from 1 to n – 1
  - each time the inner loop executes, increment j and update sum
  - the inner loop executes $\sum_{i=1}^{n-1} 2i = 2(1 + 2 + \ldots + n - 1) = n(n-1)$ assignments
- The total number of assignments, 1 + 3n + n(n – 1) → O(n^2)