CS311 Data Structures
Lecture 02 — BigO

Jyh-Ming Lien

June 4, 2018

Logistics

At Home

- Read Weiss Ch 1-4: Java Review
- Read Weiss Ch 5: Big-O
- Get your java environment set up

Goals

- Max Subarray problem (Code provided on Course webpage.)
- Review Big O and other asymptotic notations

How Fast/Big?

Review

Algorithmic time/space complexity depend on problem size

- Problem size: Often have some input parameter like n or N or (M, N)
- Describe both time and space complexity as *functions* of those parameters
- Example: For an input array of size N, the maximum element can be found in $5 * N + 3$ operations while the array can be sorted in $2N^2 + 11N + 7$ operations.

Big O

Big-O notation: upper bounding how fast functions grow based on input $T(n)$ is $O(F(n))$ if there are positive constants c and n_0 such that

- When $n \geq n_0$
- $T(n) \leq cF(n)$

Bottom line:

- If $T(n)$ is $O(F(n))$
- Then $F(n)$ grows as fast or faster than $T(n)$

Show It

Show

$$f(n) = 2n^2 + 3n + 2 \text{ is } O(n^3)$$

- Pick $c = 0.5$ and $n_0 = 6$

n	$f(n)$	$0.5n^3$
0	2	0
1	7	0.5
2		
3		
4		
5		
6		
7		

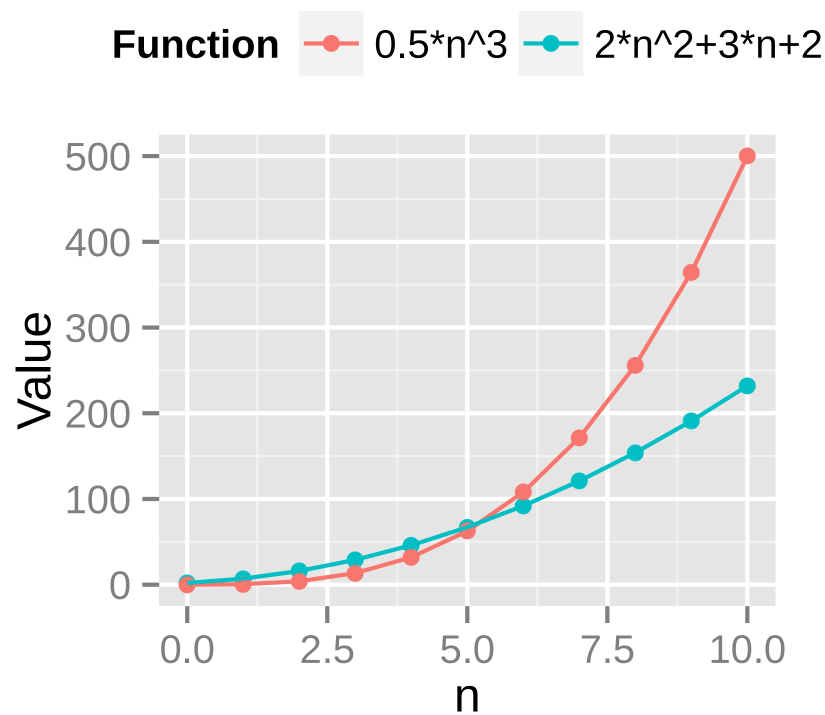
Show It

Show

$$f(n) = 2n^2 + 3n + 2 \text{ is } O(n^3)$$

- Pick $c = 0.5$ and $n_0 = 6$

n	$f(n)$	$0.5n^3$
0	2	0
1	7	0.5
2	16	4
3	29	13
4	46	32
5	67	62
6	92	108
7	121	171



How about the opposite? Show

$$g(n) = n^3 \text{ is } \Omega(2n^2 + 3n + 2)$$

Basic Rules

- Constant additions disappear
  - $N + 5$ is $O(N)$
- Constant multiples disappear
  - $0.5N + 2N + 7$ is $O(N)$
- Non-constant multiples multiply:
  - Doing a constant operation $2N$ times is $O(N)$
  - Doing a $O(N)$ operation $N/2$ times is $O(N^2)$
  - Need space for half an array with N elements is $O(N)$ space overhead
- Function calls are not free (including library calls)
  - Call a function which performs 10 operations is $O(1)$
  - Call a function which performs $N/3$ operations is $O(N)$
  - Call a function which copies object of size N takes $O(N)$ time and uses $O(N)$ space

Growth Ordering of Some Functions

Name	Leading Term	Big-Oh	Example
Constant	$1, 5, c$	$O(1)$	$2.5, 85, 2c$
Log-Log	$\log(\log(n))$	$O(\log\log n)$	$10 + (\log\log n + 5)$
Log	$\log(n)$	$O(\log(n))$	$5\log n + 2$
			$\log(n^2)$
Linear	n	$O(n)$	$2.4n + 10$
			$10n + \log(n)$
N-log-N	$n\log n$	$O(n\log n)$	$3.5n\log n + 10n + 8$
Super-linear	$n^{1.x}$	$O(n^{1.x})$	$2n^{1.2} + 3n\log n - n + 2$
Quadratic	n^2	$O(n^2)$	$0.5n^2 + 7n + 4$
			$n^2 + n\log n$
Cubic	n^3	$O(n^3)$	$0.1n^3 + 8n^{1.5} + \log(n)$
Exponential	a^n	$O(2^n)$	$8(2^n) - n + 2$
		$O(10^n)$	$100n^{500} + 2 + 10^n$
Factorial	$n!$	$O(n!)$	$0.25n! + 10n^{100} + 2n^2$

Common Patterns

- Adjacent Loops Additive: $2 \times n$ is $O(n)$

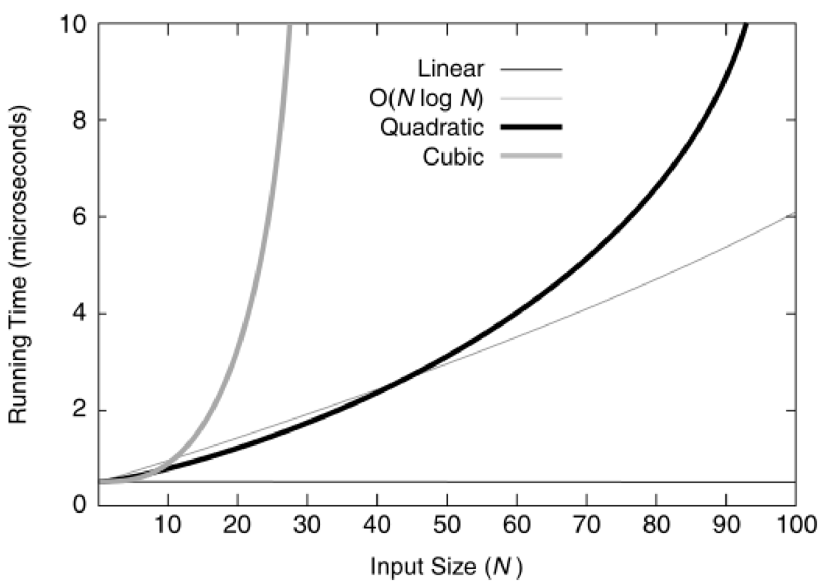
```
for(int i=0; i<N; i++){
  blah blah blah; //constant time operation
}
for(int j=0; j<N; j++){
  yakkety yack; //another constant time operation
}
```

- Nested Loops Multiplicative usually polynomial
  - 1 loop, $O(n)$
  - 2 loops, $O(n^2)$
  - 3 loops, $O(n^3)$
- Repeated halving usually involves a logarithm
  - Binary search is $O(\log n)$
  - Fastest sorting algorithms are $O(n \log n)$
  - Proofs are harder, require solving recurrence relations

Lots of special cases so be careful

FIGS/Idealized Functions

Smallish Inputs



Larger Inputs