Analysis of Algorithms

Saikrishna Arcot
(edits by M. Hudachek-Buswell)

January 17, 2017

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.
- The runtime of an algorithm is the period during which the computer is executing, and typically grows with the input size.

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.
- The runtime of an algorithm is the period during which the computer is executing, and typically grows with the input size.
- The average case runtime may be difficult to determine, and best case runtime is not really considered .

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.
- The runtime of an algorithm is the period during which the computer is executing, and typically grows with the input size.
- The average case runtime may be difficult to determine, and best case runtime is not really considered .
- Our main focus in this course is the worst case runtime (Big-O) of algorithms.

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.
- The runtime of an algorithm is the period during which the computer is executing, and typically grows with the input size.
- The average case runtime may be difficult to determine, and best case runtime is not really considered .
- Our main focus in this course is the worst case runtime (Big-O) of algorithms.
    - Easier to analyze and approximate.

Running Time

- The algorithms considered here will transform input objects into output objects for the most part.
- The runtime of an algorithm is the period during which the computer is executing, and typically grows with the input size.
- The average case runtime may be difficult to determine, and best case runtime is not really considered .
- Our main focus in this course is the worst case runtime (Big-O) of algorithms.
  - Easier to analyze and approximate.
  - Crucial to applications such as games, finance, and robotics.

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$
  - N-Log-N $\approx n \log n$

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$
  - N-Log-N $\approx n \log n$
  - Quadratic $\approx n^2$

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$
  - N-Log-N $\approx n \log n$
  - Quadratic $\approx n^2$
  - Cubic $\approx n^3$

Growth of Functions

- Seven functions that describe growth of functions that often appear in algorithm analysis:
    - Constant ≈ 1
    - Logarithmic $\approx \log n$
    - Linear $\approx n$
    - N-Log-N $\approx n \log n$
    - Quadratic $\approx n^2$
    - Cubic $\approx n^3$
    - Exponential $\approx 2^n$

Growth of Functions

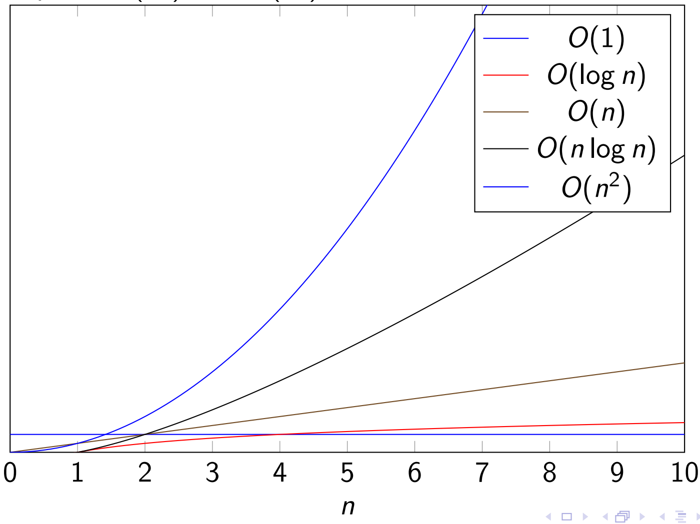
- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$
  - N-Log-N $\approx n \log n$
  - Quadratic $\approx n^2$
  - Cubic $\approx n^3$
  - Exponential $\approx 2^n$
- In a log-log chart, the slope of the line corresponds to the growth rate.

Growth of Functions

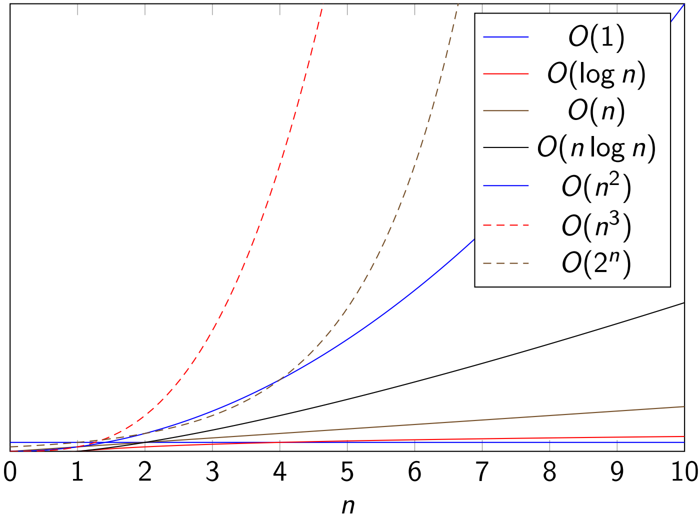
- Seven functions that describe growth of functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic $\approx \log n$
  - Linear $\approx n$
  - N-Log-N $\approx n \log n$
  - Quadratic $\approx n^2$
  - Cubic $\approx n^3$
  - Exponential $\approx 2^n$
- In a log-log chart, the slope of the line corresponds to the growth rate.
- The Big-O of a function is usually described using one of the above functions.

Functions Graphed

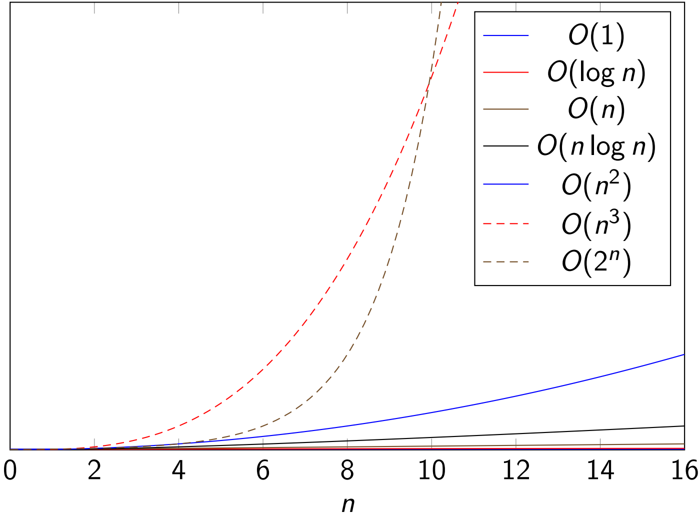
Graph of $O(n^3)$ and $O(2^n)$ not included here.



Functions Graphed (zoomed out)



Functions Graphed (zoomed out even more)



Primitive Operations

- Basic computations performed by an algorithm

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression
  - Assigning a value to a variable

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
  - Calling a method

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
  - Calling a method
  - Returning from a method

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Examples
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
  - Calling a method
  - Returning from a method
- These functions are considered to run in $O(1)$ time.

Estimating Running Time

Given the algorithm

```
1  public static double arrayMax(double[] data) {
2      int n = data.length;
3      double currentMax = data[0];
4      for (int j = 1; j < n; j++) {
5          if (data[j] > currentMax) {
6              currentMax = data[j];
7          }
8      }
9      return currentMax;
10 }
11
```

Lines 6 to 8 may be executed from 0 to n times, while all of the other lines will get executed exactly once. Therefore, the Big-O of this function is $O(n)$.

Why Big-O Matters

For example, given a processor that can do 100 operations per second, the time to do a certain number of operations (in seconds) with a function of a certain Big-O might be as follows (assume that only one operation is done for each input that is used):

Big-O	1000 inputs	1001 inputs	2000 inputs
$O(1)$	constant	constant	constant
$O(\log n)$	0.09966	0.09967	0.10967
$O(n)$	10	10.01	20
$O(n \log n)$	99.66	99.77	219
$O(n^2)$	10000	10020	40000
$O(n^3)$	10000000	100300300	80000000
$O(2^n)$	1.07×10^{299}	2.14×10^{299}	1.14×10^{598}

Notes on Big-O

- The Big-O of a function only describes how the runtime of a function changes with respect to input size; it does **not** say anything about the absolute runtime (a function with a Big-O of $O(n^2)$ may take less time than a function with a Big-O of $O(n)$ for some given input size).

Notes on Big-O

- The Big-O of a function only describes how the runtime of a function changes with respect to input size; it does **not** say anything about the absolute runtime (a function with a Big-O of $O(n^2)$ may take less time than a function with a Big-O of $O(n)$ for some given input size).
- Given the Big-O of a function, you **cannot** simply calculate the time it will take for the function to run given an input of certain size; there are other factors involved.