Growth Functions

Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu
Growth Functions

Remember, algorithms are general recipes for solving problems not specific of any language or platform. We characterize their performance in time/speed/effort/complexity with growth functions.

Examples:

- $O(n)$ “Order $n$” or “Big-oh of $n$”
- $O(n^2)$ “Order $n$ squared” or “Big-oh of $n$ squared”
- $O(\log_2 n)$ “Order log to the base two of $n$” or . . .

We only want the largest (or dominant) function of $n$ and we ignore constant factors.

- $\frac{1}{2} n^2 + 2112$ is $O(n^2)$
- $42 n^{1.5} - 8,675,309$ is $O(n^{1.5})$
- $11 n \log_2 n + 1$ is $O(n \log_2 n)$
- $42 n^{1.5} + \sqrt{n}$ is $O(n^{1.5})$ because $\sqrt{n} = n^{0.5}$ so $n^{1.5}$ dominates
Growth Functions

Growth functions let us characterize how the *time/effort/space* required to execute the algorithm grows as the size of the input grows.

Think of this as “complexity”.

We’re concerned with the measures of effort/complexity needed to correctly solve a problem.

We’re also concerned with how those measures change proportionally with the size of the input. I.e., how does the effort scale or grow with the input? What is its “order of growth”?
Common Growth Functions
A Quick log Refresher

Assume log means \( \log_2 \).

Definition: \( \log(n) \) is the number so that \( 2^{\log(n)} = n \).

In other words, \( \log(n) \) is the number of times you need to divide \( n \) by 2 to get down to 1.

\[
\log_2(32) = 5 \quad \text{because} \quad 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \quad \text{halve 5 times}
\]

\[
\log_2(64) = 6 \quad \text{because} \quad 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \quad \text{halve 6 times}
\]

Thanks to Mary Wootters.
A Quick log Refresher

Assume log means $\log_2 n$.

Definition: $\log(n)$ is the number so that $2^{\log(n)} = n$.

In other words, $\log(n)$ is the number of times you need to divide $n$ by 2 to get down to 1.

$log_2(32) = 5$ because $32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$log_2(64) = 6$ because $64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$log_2(128) = 7$ and $2^7 = 128$

$log_2(256) = 8$ and $2^8 = 256$

$log_2(512) = 9$ and $2^9 = 512$

$log_2(1024) = 10$ and $2^{10} = 1024$

$log_2(\text{number of particles in the universe}) < 280$ so $\log(n)$ grows very slowly.
Worst Case Analysis

The “running time” (time/speed/effort/complexity) of an algorithm is determined by its worst possible input.

In other words, if we say some recipe is an O($n^2$) algorithm, that means that the worst possible running time is proportional to $n^2$ and never worse than that. It could — under lucky circumstances — be better (faster) than O($n^2$), but never worse no matter what.

A common example of where we need to apply this thinking is in sorting lists.
Worst Case Analysis

The “running time” \((time/speed/effort/complexity)\) of an algorithm is determined by its worst possible input.

In other words, if we say some recipe is an \(O(n^2)\) algorithm, that means that the worst possible running time is proportional to \(n^2\) and never worse than that. It could — under lucky circumstances — be better (faster) than \(O(n^2)\), but never worse no matter what.

A common example of where we need to apply this thinking is in sorting lists.

Q: Which input to a sort algorithm is worse?
   • the elements of the list are arranged randomly
   • the elements of the list are already sorted in ascending order
   • the elements of the list are already sorted in descending order

A: It depends on the specifics of the sorting algorithm.

But when we characterize the sorting algorithm as \(O(something)\), that must represent the worst-case input.
Asymptotic Analysis

From the CLRS text, section 3.1
Asymptotic Analysis

From the CLRS text, section 3.1

- $f(n) = \Theta(g(n))$: upper-bound worst case
- $f(n) = O(g(n))$: "Big Oh" upper-bound
- $f(n) = \Omega(g(n))$: "Big Omega" lower-bound best case
Asymptotic Analysis

From the CLRS text, section 3.1

“Big Theta”
tight-bound
worst and best range

“Big Oh”
upper-bound
worst case

“Big Omega”
lower-bound
best case
Asymptotic Analysis :: Big Oh

From the CLRS text, section 3.1

“Big Oh”
upper-bound
worst case

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Oh

From the CLRS text, section 3.1

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Oh

From the CLRS text, section 3.1

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Oh

From the CLRS text, section 3.1

"Big Oh" upper-bound worst case

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Oh

From the CLRS text, section 3.1

"Big Oh"
upper-bound
worst case

\[ f(n) = O(g(n)) \]

\[
O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0 \}.
\]
Asymptotic Analysis :: Big Omega

From the CLRS text, section 3.1

\[ f(n) = \Omega(g(n)) \]

\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \} \]
Asymptotic Analysis :: Big Omega

From the CLRS text, section 3.1

"Big Omega"
lower-bound
best case

\[ f(n) = \Omega(g(n)) \]

\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Omega

From the CLRS text, section 3.1

"Big Omega"
lower-bound
best case

\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Theta

From the CLRS text, section 3.1

\[ f(n) = \Theta(g(n)) \]

\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \]

\[ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \} . \]
Asymptotic Analysis :: Big Theta

From the CLRS text, section 3.1

\[ f(n) = \Theta(g(n)) \]

\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \} \]
Asymptotic Analysis and Growth Functions

Let’s do some examples.