Asymptotic Notation

- The need for asymptotic notation
- Definition of asymptotic notations O, Ω, Θ
- Asymptotic relations between common functions
- Analyzing running time and other applications

Motivations

Consider this piece of code. What it's running time?

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n ; j++) {
     x = y + i*j;
        A[i][j] = 7*i*j + 3*x;
        cout << A[i][j];
  }
}
```

n^2 iterations

Running time = $n^2 \times$ (time to execute these instructions)

The time to execute these instructions is a constant, independent of n, but dependent on the computing environment (processor, compiler, system load, …)

So we can only say that running time = $c{\cdot}n^2$, for some unknown constant c

Motivations

How about this piece of code?

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n ; j++) {
      x = y + i*j;
        A[i][j] = 7*i*j + 3*x;
        cout << A[i][j];
  }
}
for (int i = 0; i < n; i++) x = x+ i*i;
```

time c

time d

Running time = $c{\cdot}n^2 + d{\cdot}n$

not informative and gets messy quickly

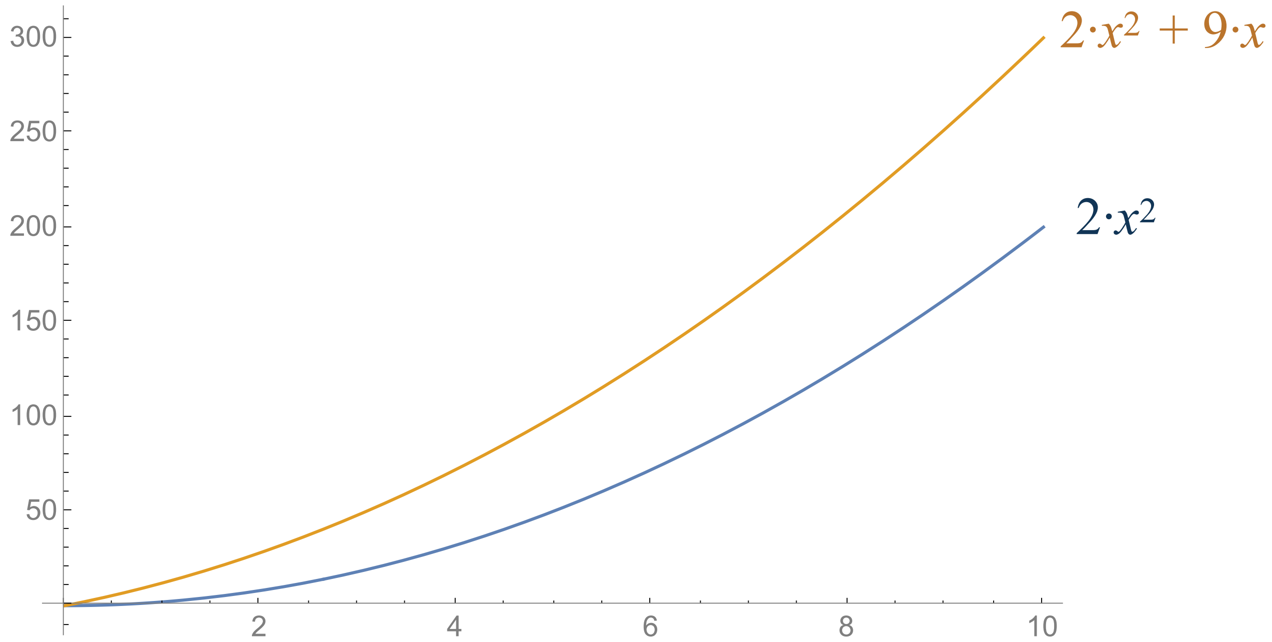
We need some concept of “running time” that would be

- independent of the computing environment
- independent of time units
- informative — provide useful information about performance

Motivations

Consider running time function $2 \cdot x^2 + 9 \cdot x$

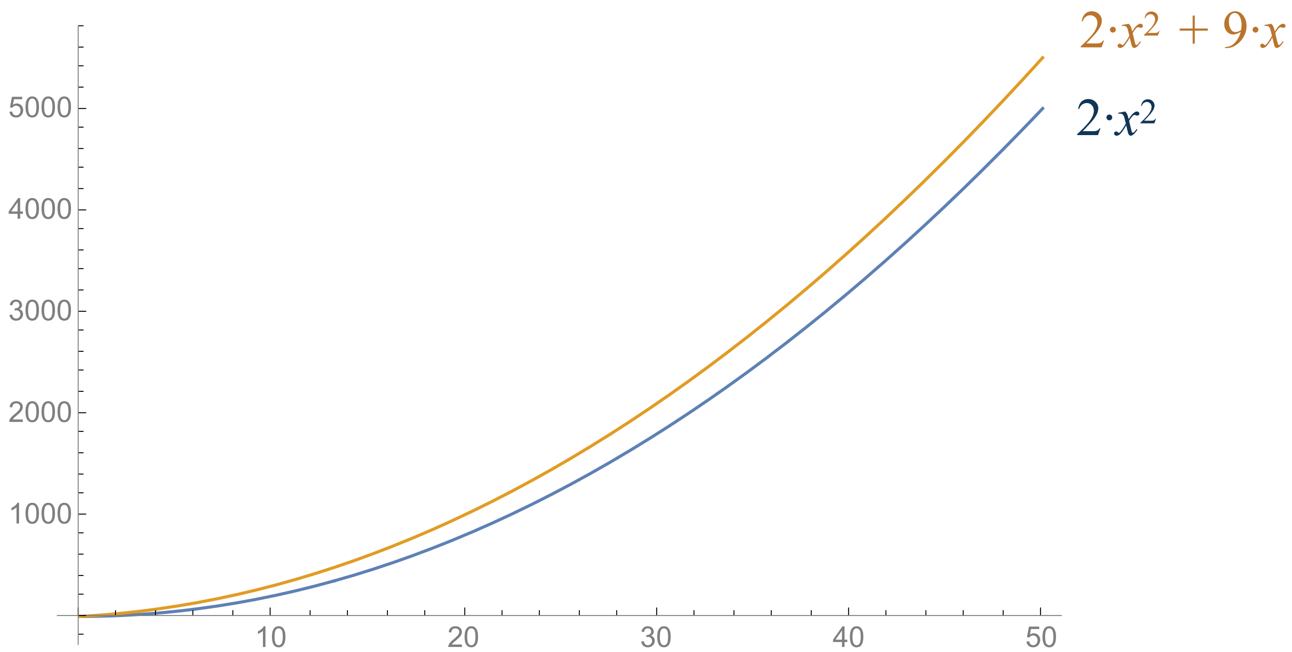
For $x \leq 10$



Motivations

Consider running time function $2 \cdot x^2 + 9 \cdot x$

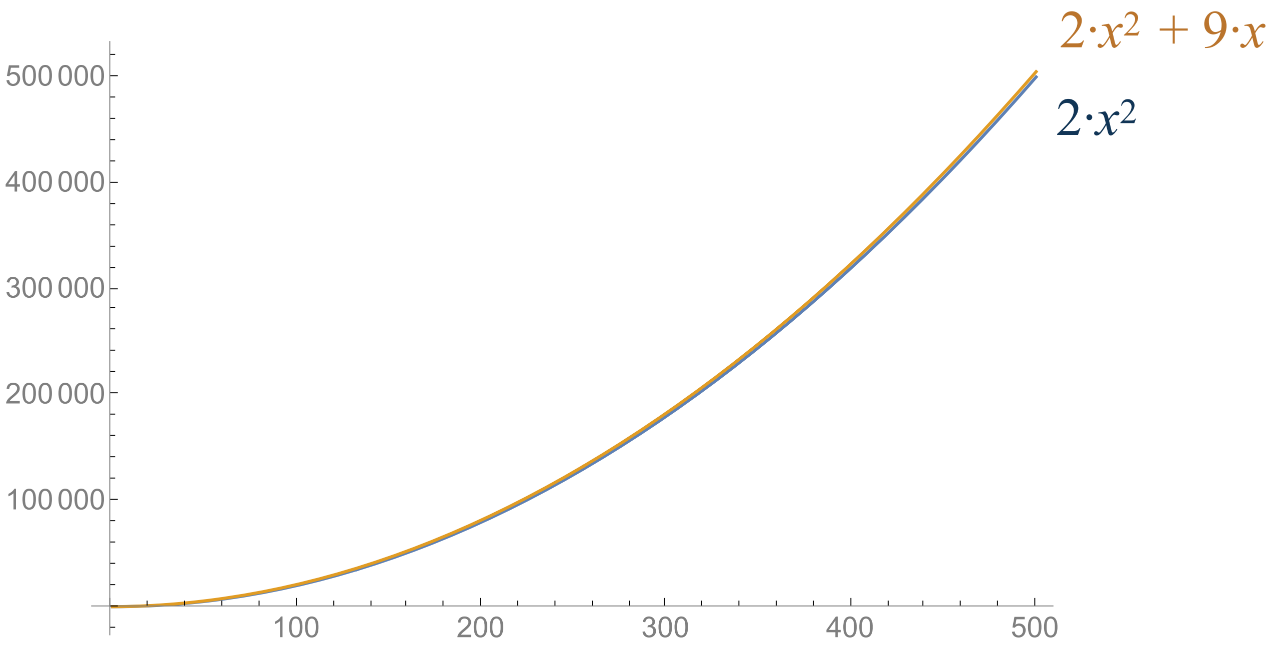
Now zoom out : for $x \leq 50$



Motivations

Consider running time function $2 \cdot x^2 + 9 \cdot x$

And zoom out even more: for $x \leq 500$



As x grows, the term $9 \cdot x$ becomes negligible *compared to the value of the function*

Motivations

We need some concept of "running time" that would be

- independent of the computing environment
- independent of time units
- informative — provide useful information about performance

Key word: *scaling*. Instead of capturing the absolute performance, we want to know *how does the performance scale as the input size n increases?*

To capture this, we use asymptotic notations:

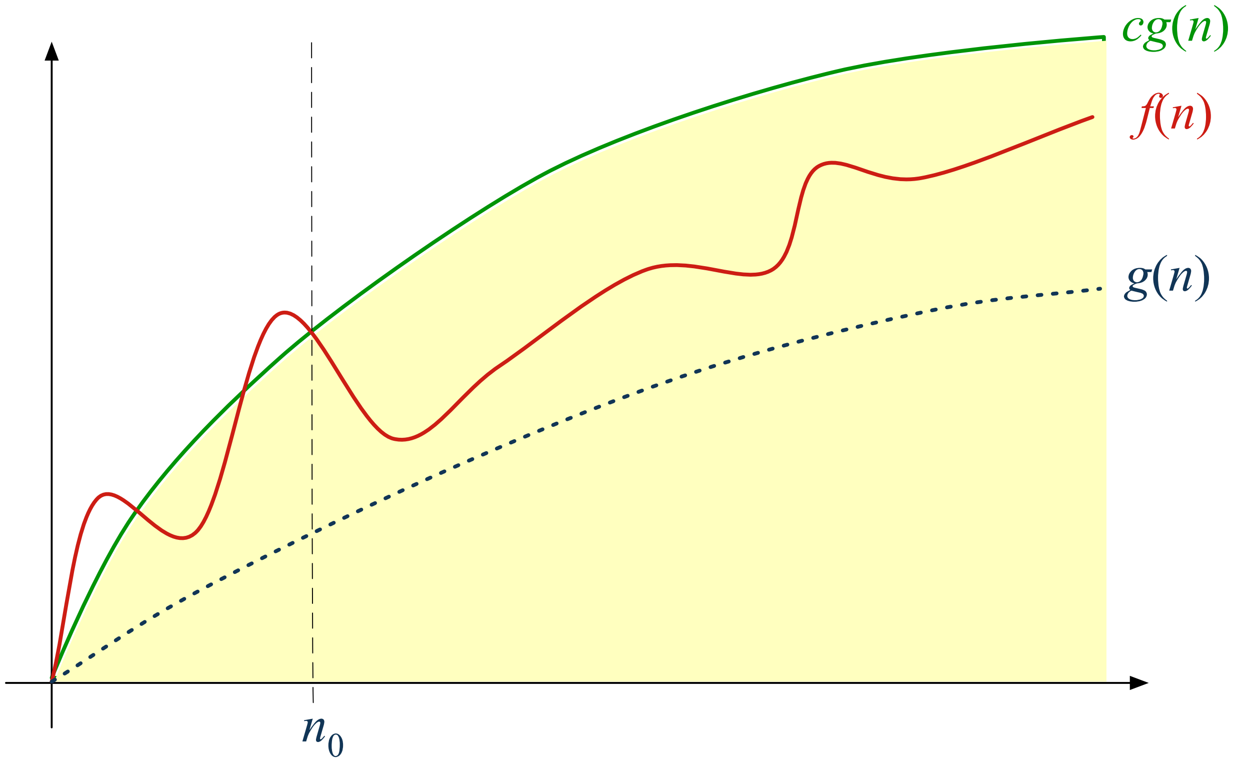
$T(n) = O(g(n))$ $T(n)$ grows not faster than proportionally with $g(n)$

$T(n) = \Omega(g(n))$ $T(n)$ grows not slower than proportionally with $g(n)$

$T(n) = \Theta(g(n))$ $T(n)$ grows proportionally with $g(n)$

Big-Oh Notation — Definition

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order (at most) $g(n)$, denoted $f(n) = O(g(n))$, iff there are constants c and n_0 such that $|f(n)| \le c \cdot g(n)$ for all $n \ge n_0$.



Big-Oh Notation — Definition

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order (at most) $g(n)$, denoted $f(n) = O(g(n))$, iff there are constants c and n_0 such that $|f(n)| \leq c \cdot g(n)$ for all $n \geq n_0$.

Example: Prove, directly from the definition, that $10n+5 = O(n)$.

To prove it, all we need to do is to observe that $10n+5 \leq 11n$ for $n \geq 5$.

$c = 11$ $\quad n_0 = 5$

Example: Prove, directly from the definition, that $2n^3+6n^2+2 = O(n^3)$.

We can estimate $2n^3+6n^2+2 \leq 2n^3+6n^3+2n^3 = 10n^3$ for $n \geq 1$.

$c = 10$ $\quad n_0 = 1$

Big-Oh Notation — Definition

- Comments:
  - Definition also applies to functions $\mathbb{R} \to \mathbb{R}$.
  - In this class we mostly care about functions $\mathbb{N} \to \mathbb{N}$ (running time cannot be negative). In this case the absolute value in the definition is not needed.
  - The choice of c and n_0 is not unique. For example, to show that $10n+5 = O(n)$ we can estimate

$$10n+5 \le 11n \quad \text{for} \quad n \ge 5$$

$$10n+5 \le 15n \quad \text{for} \quad n \ge 1$$

$$\dots$$

  - In particular, if $g(n)$ is strictly positive, then we can always take $n_0 = 0$, by taking c large enough.

Big-Oh Notation — Definition

- Comments:
  - The goal is to express a possibly complex $f(n)$ in terms of a simple function $g(n)$. So while it is true that

    $$n^3 = O(2n^3+6n^2+2)$$

    this estimate is not useful.

  - We can write $2n^3+6n^2+2 = O(n^3)$, but it makes no sense to write

    $$O(n^3) = 2n^3+6n^2+2.$$

    Why?

    This equation symbol does *not* represent equality. It represents $\in$ relation. Some people write it as $2n^3+6n^2+2 \in O(n^3)$.

  - Important: the big-Oh notation is only *an upper bound*. So

    $2n^3+6n^2+2 = O(n^3)$, but it is also true that

    $2n^3+6n^2+2 = O(n^4)$, or

    $2n^3+6n^2+2 = O(n^5)$, etc.

    But typically we look for the best possible upper bound, which is $O(n^3)$. This will be later captured using the Θ notation.

Big-Oh Notation — Definition

$$H_1 = 1$$
$$H_2 = 1 + \tfrac{1}{2} = \tfrac{3}{2}$$
$$H_3 = 1 + \tfrac{1}{2} + \tfrac{1}{3} = \tfrac{11}{6}$$

Example: Let's derive a big-Oh estimate for harmonic numbers:

$$H_n = \sum_{j=1}^{n} \frac{1}{j} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

Theorem: For $n \geq 1$ we have $\frac{1}{2}(\log n - 1) \leq H_n \leq \log n + 1.$

We'll prove this theorem later if time suffices

From this theorem, for $n \geq 2$ we get :

$n_0 = 2$

$$\begin{aligned} H_n &\leq \log n + 1 \\ &\leq \log n + \log n \\ &= 2 \log n \end{aligned}$$

$c = 2$

So we can conclude that $H_n = O(\log n)$

Big-Oh Notation — Definition

Example: Let's derive some big-Oh estimate for the sequence defined recursively:

$$a_0 = 3\ ,\ \ a_1 = 8, \text{ and } \ a_n = 2 \cdot a_{n-1} + a_{n-2} \ \text{ for } n \geq 2\ .$$

$$\begin{aligned} a_2 &= 19 \\ a_3 &= 46 \\ a_4 &= 111 \end{aligned}$$

Claim: $a_n \leq 3(2.75)^n$ for $n \geq 0$.

Proof: The base case involves values $n = 0,1$. For $n = 0$ we have $a_0 = 3 \leq 3(2.75)^0$, and for $n = 1$ we have $a_1 = 8 \leq 3(2.75)^1$.

Inductive step: assume that the claim holds for all values smaller than some n, where $n \geq 2$. Then

$$\begin{aligned} a_n &= 2a_{n-1} + a_{n-2} \\ &\leq 2 \cdot 3(2.75)^{n-1} + 3(2.75)^{n-2} \\ &= 3(2.72)^{n-2}(2 \cdot 2.75 + 1) \\ &\leq 3(2.75)^{n-2}(2.75)^2 \\ &\leq 3(2.75)^n \end{aligned}$$

applying inductive assumption

This completes the inductive step, and the proof of the claim. ■

Big-Oh Notation — Definition

$$a_2 = 19$$
$$a_3 = 46$$
$$a_4 = 111$$

Example: Let's derive some big-Oh estimate for the sequence defined recursively:

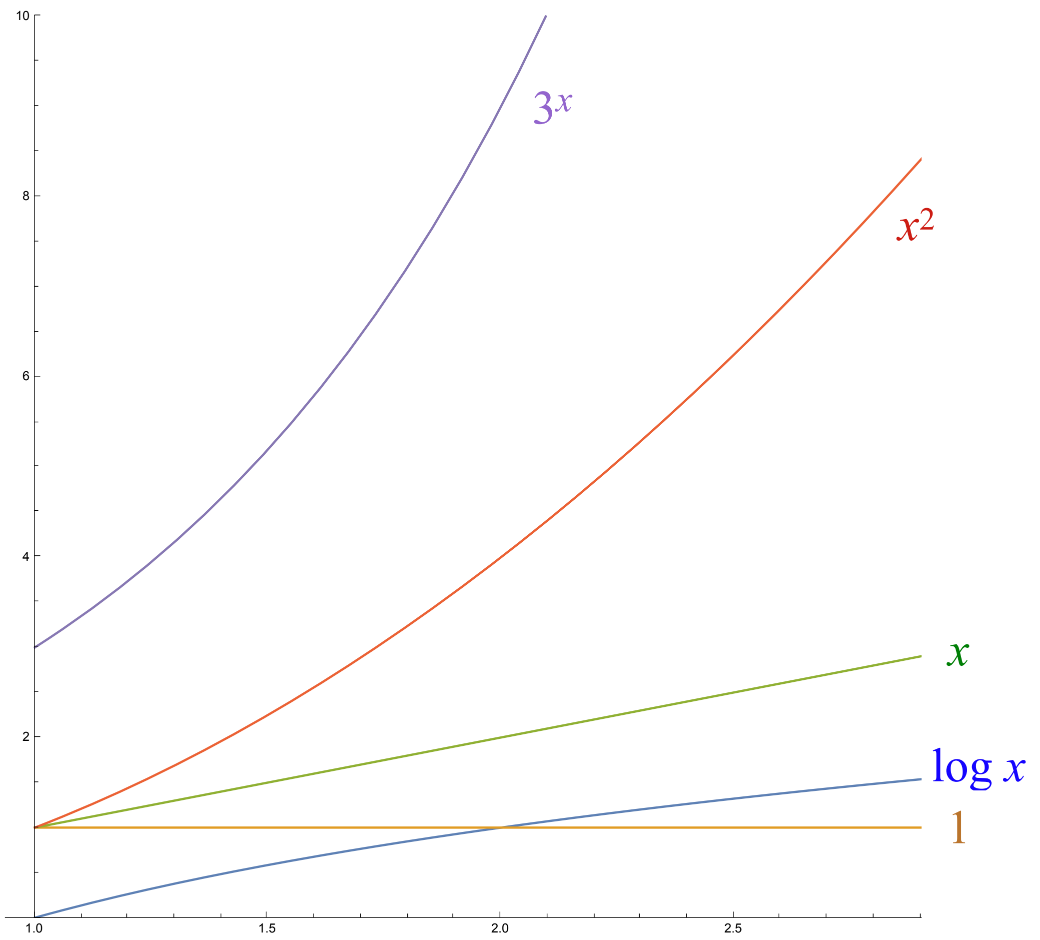
$$a_0 = 3\ ,\ \ a_1 = 8, \text{ and } \ a_n = 2\cdot a_{n-1} + a_{n-2} \ \text{ for } n \geq 2\ .$$

Claim: $a_n \leq 3(2.75)^n$ for $n \geq 0.$

From this claim, we obtain that $a_n = O((2.75)^n)$.

Big-Oh Notation — Common functions

- Common functions used in asymptotic bounds:
  - constant 1
  - logarithmic $\log n$
  - polynomial n^b where $b > 0$
  - exponential c^n where $c > 1$



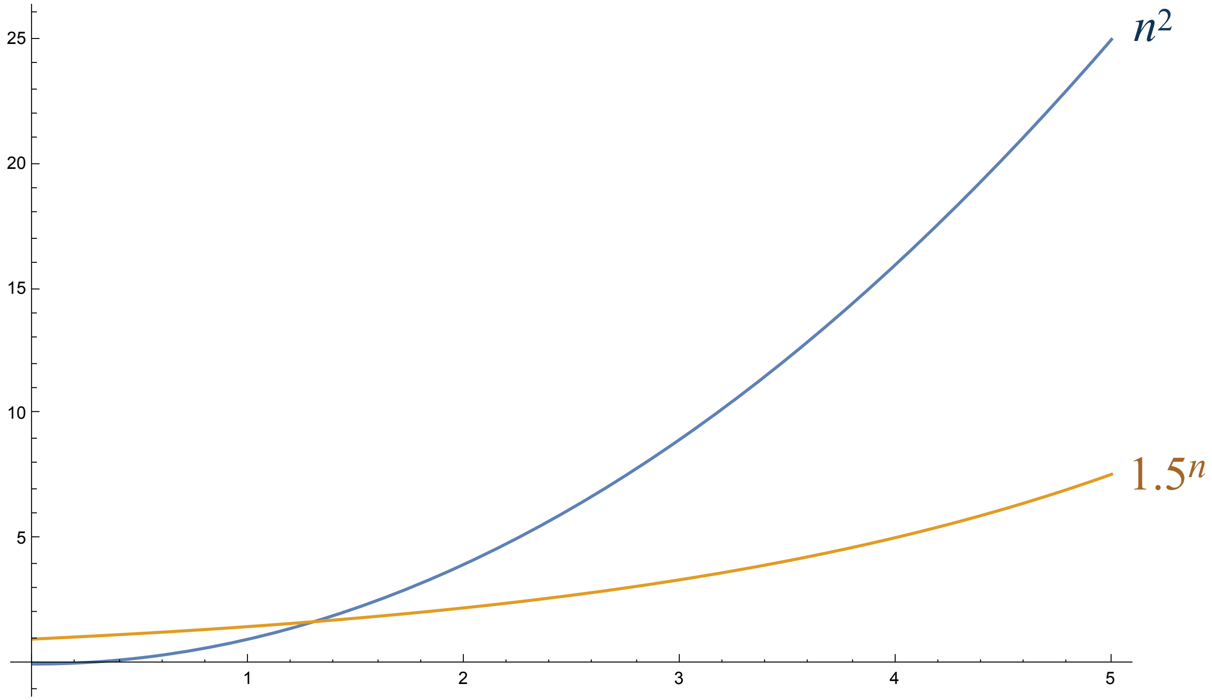
Most of the asymptotic bounds used in the analysis of algorithms can be expressed as combinations of these “reference functions”

We focus on properties of these functions…

Big-Oh Notation — Common functions

Question: Which function grows faster as $n \to \infty$?

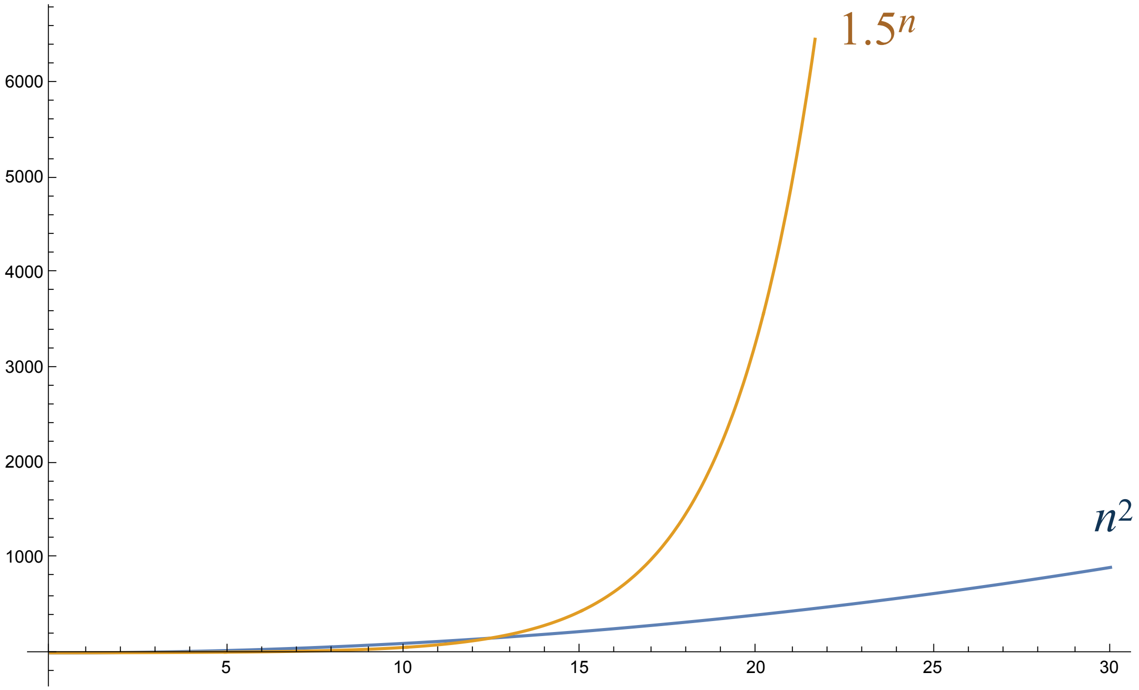
- n^2
- 1.5^n



Big-Oh Notation — Common functions

Question: Which function grows faster as $n \to \infty$?

- n^2
- 1.5^n



Answer: 1.5^n

Big-Oh Notation — Combining Asymptotic Bounds

First, we show some general rules for combining asymptotic bounds:

Theorem: Suppose that $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$.
Then:

(a) $f_1(n) + f_2(n) = O(\, g_1(n) + g_2(n)\,)$

(b) $f_1(n) + f_2(n) = O(\, \max\, (\, g_1(n)\, , g_2(n)\,)\,)$

(c) $f_1(n) \cdot f_2(n) = O(\, g_1(n) \cdot g_2(n)\,)$

Proof:

Big-Oh Notation — Properties of Common Functions

- Logarithmic functions.

In this class we use notations

$$\log x = \log_2 x$$
$$\ln x = \log_e x \quad \longleftarrow \text{natural logarithm}$$

Fact: Let $r, p > 1$, $x > 0$. Then

this is a constant (independent of x)

$$\log_r x = \frac{\log_p x}{\log_p r} = \frac{1}{\log_p r} \cdot \log_p x$$

So all logarithmic functions have the same asymptotic behavior: for all bases $r, p > 1$ we have

$$\log_r x = O(\log_p x)$$

Big-Oh Notation — Properties of Common Functions

- Polynomial functions.

Fact: Let $f(x) = \sum_{i=0}^{k} a_i x^i$. Then $f(x) = O(x^k)$.

Example:

$$f(x) = 2x^5 + 3x^2 + 1$$
$$f(x) = x + 7$$
$$f(x) = 5x^{121} + x^{37}$$

Proof: Let $A = \max |a_i|$. For $x \geq 1$ we can then estimate $f(x)$ as follows:

$$\begin{aligned} f(x) &= a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0 \\ &\leq A(\, x^k + x^{k-1} + \ldots + x + 1\,) \\ &\leq A(\, x^k + x^k + \ldots + x^k + x^k\,) \\ &= A(k+1)x^k \end{aligned}$$

This gives us that $f(x) \leq c \cdot x^k$ for $c = A(k+1)$ and $x \geq 1$. ■

Big-Oh Notation — Properties of Common Functions

Theorem: For all $a,b > 0$, $c > 1$, we have

(a) $1 = O(\log^a n)$

(b) $\log^a n = O(n^b)$

(c) $n^b = O(c^n)$

Proof: We prove (c). Take $d = c^{1/b}$ and $A = 1/(d-1)^b$. Since $c > 1$ and $b > 0$, we have $d > 1$. Then, for $n \geq 1$ we can estimate n^b as follows

$$
\begin{aligned}
n^b &= (\overbrace{1 + 1 + \ldots + 1}^{n})^b \\
&\leq (1 + d + d^2 + \ldots + d^{n-1})^b \\
&= \left(\tfrac{d^n - 1}{d-1}\right)^b \\
&\leq \left(\tfrac{1}{d-1}\right)^b \cdot (d^n)^b \;=\; A \cdot (d^b)^n \;=\; A \cdot c^n
\end{aligned}
$$

This gives us that $n^b \leq A \cdot c^n$ for $n \geq 1$. ■

Big-Oh Notation — Examples

Example: Determine the best big-Oh estimate for $f(n) = n^2 \log^2 n + n^3$.

suspect for the dominating term

We can estimate it as follows:

$$
\begin{aligned}
f(n) &= n^2 \log^2 n + n^3 \\
&= n^2 O(n) + n^3 \\
&= O(n^3) + n^3 \\
&= O(n^3)
\end{aligned}
$$

this is actually the $\in$ relation

because $log^2 n = O(n)$, by previous slide

this is actually the $\subseteq$ relation

So $f(n) = O(n^3)$.

Big-Oh Notation — Examples

Example: Determine the best big-Oh estimate for $f(n) = 13n^{2.3}\log_5^2 n + 11\sqrt{n}\log^7 n + n^3$.

suspect for the dominating term

We can estimate it as follows:

$$
\begin{aligned}
f(n) &= 13n^{2.3}\log_5^2 n + 11\sqrt{n}\log^7 n + n^3 \\
&= 13n^{2.3}O(n^{0.7}) + 11n^{0.5}O(n^{2.5}) + n^3 \\
&= O(n^3) + O(n^3) + n^3 \\
&= O(n^3)
\end{aligned}
$$

So $f(n) = O(n^3)$.

Big-Oh Notation — Examples

Example: Determine the best big-Oh estimate for $f(n) = 7n^5 2^n + 3^n$. suspect for the dominating term

We can estimate $f(n)$ as follows:

$$\begin{aligned} f(n) &= 7n^5 2^n + 3^n \\ &= 2^n \cdot (\, 7n^5 + 1.5^n \,) \\ &= 2^n \cdot (\, O(1.5^n) + 1.5^n \,) \qquad \text{because } n^5 = O(1.5^n) \\ &= 2^n \cdot O(1.5^n) \\ &= O(3^n) \end{aligned}$$

So $f(n)=O(3^n)$.

Big-Oh Notation — Examples

Example: Determine the best big-Oh estimate for the running time of this algorithm:

```
Algorithm WhatsMyRuntime(n: integer)
    for i ← 1 to 6n do z ← 2z − 1
    for i ← 1 to 2n² do
        for j ← 1 to n+1 do z ← z² − z
```

Number of iterations of the first "for" loop = $6n$

Number of iterations of the second (double) "for" loop = $2n^2(n+1)$

Each iterations takes $O(1)$ time, so the total running time is

$$6n + 2n^2(n+1) \;=\; 2n^3 + 2n^2 + 6n \;=\; O(n^3)$$

Big-Oh Notation — Examples

Example: Determine the best big-Oh estimate for the number of “hello”s printed by this algorithm:

```
Algorithm HowManyHellos(n: integer)
    for i ←1 to 6n do print(“hello”)
    for i ←1 to 2n+1 do
        for j ←1 to i+2 do print(“hello”)
```

$6n$ “hello”s

Analysis of double “for” loop:

- For each i, the internal loop makes $i+2$ iterations
- So the total number of iterations of the double “for” loop is

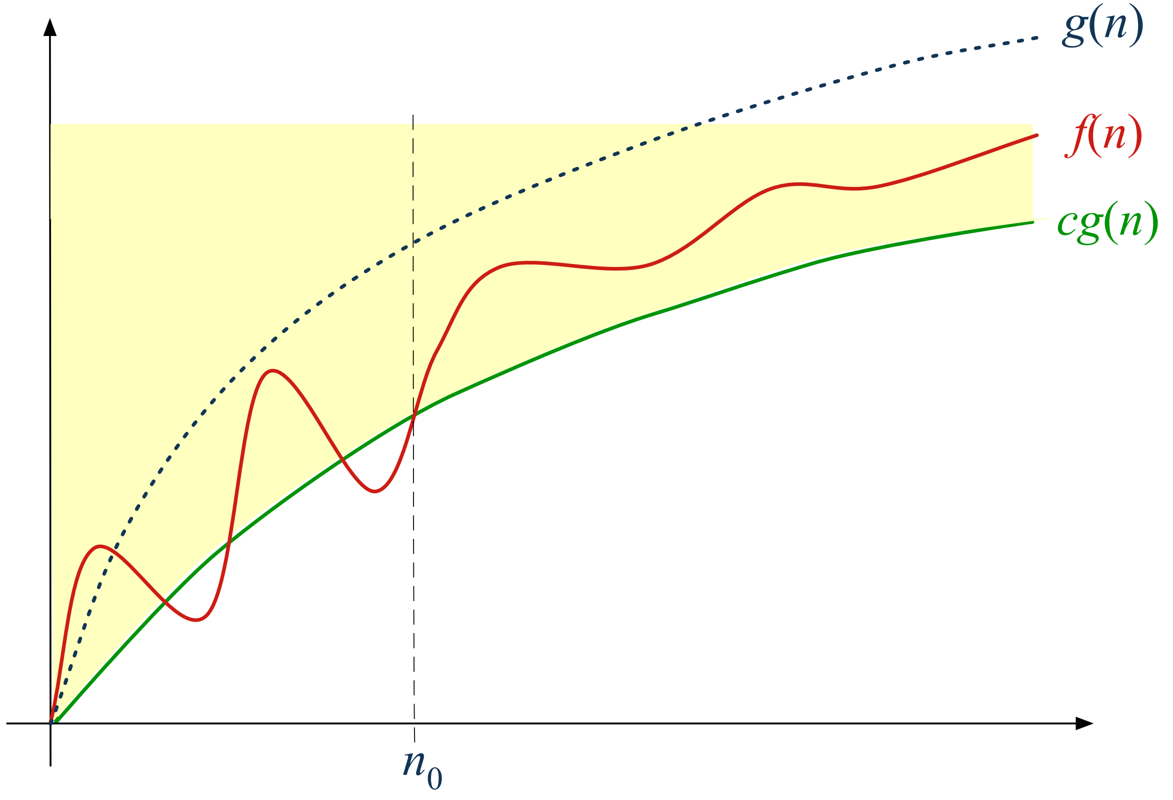
$$\begin{aligned} \sum_{i=1}^{2n+1}(i+2) &= \sum_{i=1}^{2n+1} i + \sum_{i=1}^{2n+1} 2 \\ &= \tfrac{1}{2}(2n+1)(2n+2) + 2(2n+1) \\ &= 2n^2 + 7n + 3 \end{aligned}$$

Therefore the total number of “hello”s is $2n^2 + 13n + 3 = O(n^2)$

Big-Ω Notation — Definition

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order at least $g(n)$, denoted $f(n) = \Omega(g(n))$, iff there are constants c and n_0 such that $|f(n)| \geq c \cdot g(n)$ for all $n \geq n_0$.

lower bound, as opposed to big-Oh, that is an upper bound



Big-Ω Notation — Definition

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order at least $g(n)$, denoted $f(n) = \Omega(g(n))$, iff there are constants c and n_0 such that $|f(n)| \geq c \cdot g(n)$ for all $n \geq n_0$.

Example: Prove, directly from the definition, that $10n+5 = \Omega(n)$. $c = 10$ $n_0 = 0$

This is straightforward when all terms are non-negative: $10n+5 \geq 10n$ for $n \geq 0$.

Example: Prove, directly from the definition, that $2n^3 - 6n^2 + 2 = \Omega(n^3)$.

We can estimate

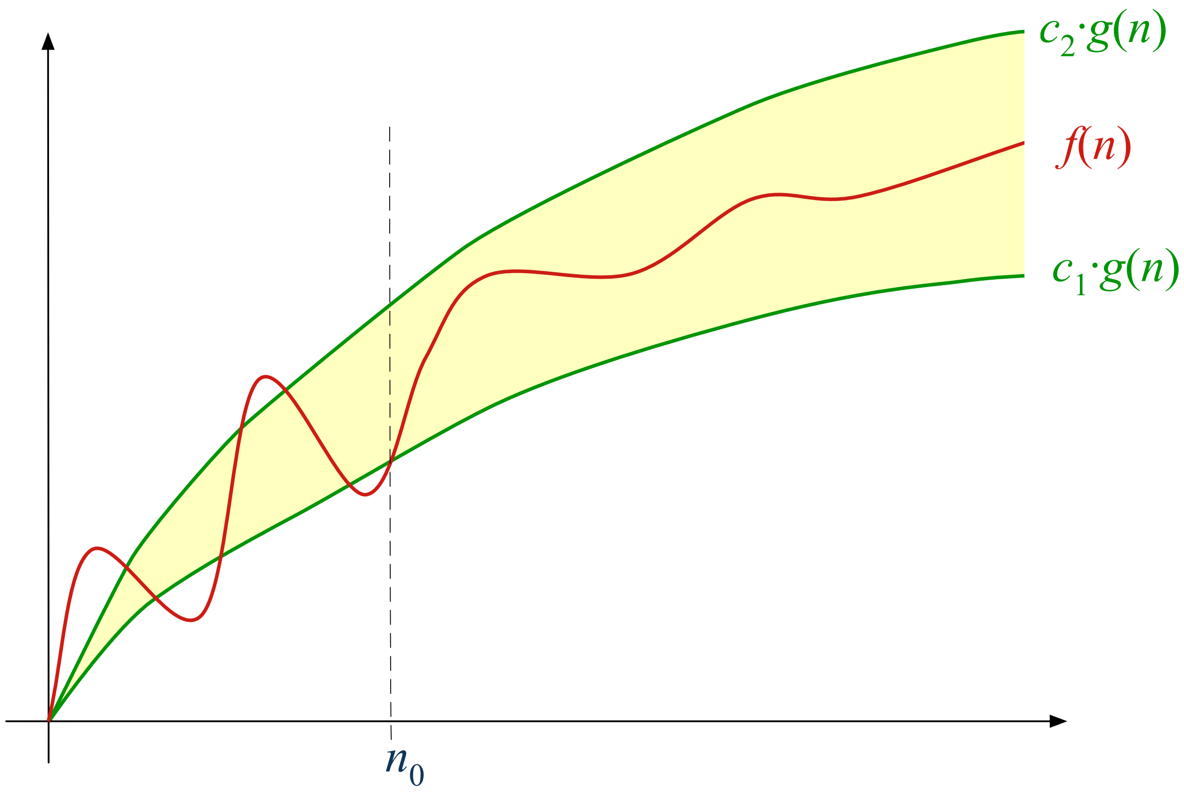
$$\begin{aligned} 2n^3 - 6n^2 + 2 &\geq 2n^3 - 6n^2 \\ &= n^3 + n^2(n-6) \\ &\geq n^3 \qquad \text{for } n \geq 6 \end{aligned}$$

$n_0 = 6$

$c = 1$

Θ-Notation

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order $g(n)$, denoted $f(n) = \Theta(g(n))$, iff there are constants c_1, c_2 and n_0 such that $c_1 \cdot g(n) \leq |f(n)| \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Θ-Notation

Definition: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. We say that $f(n)$ is of order $g(n)$, denoted $f(n) = \Theta(g(n))$, iff there are constants c_1, c_2 and n_0 such that $c_1 \cdot g(n) \leq |f(n)| \leq c_2 \cdot g(n)$ for all $n \geq n_0$.

In other words, $f(n) = \Theta(g(n))$ means that $g(n)$ is *a tight asymptotic* estimate for $f(n)$. This is capture by the following theorem:

Theorem: Let $f(n)$ and $g(n) : \mathbb{Z} \to \mathbb{Z}$ be two functions. Then $f(n) = \Theta(g(n))$ iff both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Proof: Since (⇒) is trivial, so we will only prove (⇐). Since $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, there are constants such that

$$c_1 \cdot g(n) \leq |f(n)| \leq c_2 \cdot g(n)$$

for $n \geq n_1$ for $n \geq n_2$

Taking $n_0 = \max(n_1, n_2)$, both inequalities will hold for $n \geq n_0$. ■

Θ-Notation — Examples

$$\begin{aligned} H_1 &= 1 \\ H_2 &= 1 + \tfrac{1}{2} = \tfrac{3}{2} \\ H_3 &= 1 + \tfrac{1}{2} + \tfrac{1}{3} = \tfrac{11}{6} \end{aligned}$$

Example: Determine the Θ-estimate for the harmonic sequence:

$$H_n = \sum_{j=1}^{n} \frac{1}{j} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

Theorem: For $n \geq 1$ we have $\frac{1}{2}(\log n - 1) \leq H_n \leq \log n + 1$.

We showed earlier that $H_n = O(\log n)$. So we now need to show that $H_n = \Omega(\log n)$.

From the theorem, for $n \geq 4$ we have:

$$\begin{aligned} H_n &\geq \tfrac{1}{2}(\log n - 1) \\ &\geq \tfrac{1}{4} \log n \end{aligned}$$

$n_0 = 4$

$c = 1/4$

So $H_n = \Omega(\log n)$.

Since $H_n = O(\log n)$ and $H_n = \Omega(\log n)$, we obtain that $H_n = \Theta(\log n)$.

Θ-Notation — Examples

Example 1: Give the Θ-estimate for the running time of this code, as a function of n (no proofs).

```
for i ← 1 to 2n+1 do
    x ← x^2
    for j ← 1 to n+2 do
        for k ← 1 to n+1 do
            x ← x/k
```

Answer: $\Theta(n^3)$.

Explanation: We have three nested independent loops, each of range $\Theta(n)$.
Operations $x \leftarrow x^2$ and $x \leftarrow x/k$ take time $\Theta(1)$.

Θ-Notation — Examples

Example 2: Give the Θ-estimate for the running time of this code, as a function of n (no proofs).

```
i ← 1
while i < n do
    x ← x^2
    i ← 2 · i
```

Question: How many iterations will this loop make for $n = 125$?

- 5
- 8
- 4
- 7
- 6
- none of the above

Θ-Notation — Examples

Example 2: Give the Θ-estimate for the running time of this code, as a function of n (no proofs).

$i \leftarrow 1$
while $i < n$ do
    $x \leftarrow x^2$
    $i \leftarrow 2 \cdot i$

Question: How many iterations will this loop make for $n = 125$?

- 5
- 8
- 4
- 7
- 6
- none of the above

Answer: 7. Values of i for which the loop will execute: $1 \quad 2 \quad 4 \quad 8 \quad 16 \quad 32 \quad 64$

Θ-Notation — Examples

Example 2: Give the Θ-estimate for the running time of this code, as a function of n (no proofs).

$i \leftarrow 1$
while $i < n$ do
    $x \leftarrow x^2$
    $i \leftarrow 2 \cdot i$

Answer: $\Theta(\log n)$.

Explanation: i will double exactly $\lceil \log n \rceil$ times, and $\lceil \log n \rceil = \Theta(\log n)$.

Θ-Notation — Examples

Challenge questions: Give the Θ-estimate, as a function of n, for the running time of these three pieces of code.

```
i ← 1
while i < n do
    for j = i to n do
        x ← x^2
        i ← 2 · i
```

```
i ← 1
while i < n do
    for j = 1 to i do
        x ← x^2
        i ← 2 · i
```

```
i ← 2
while i < n do
    x ← x^2
    i ← i^2
```