Opportunities

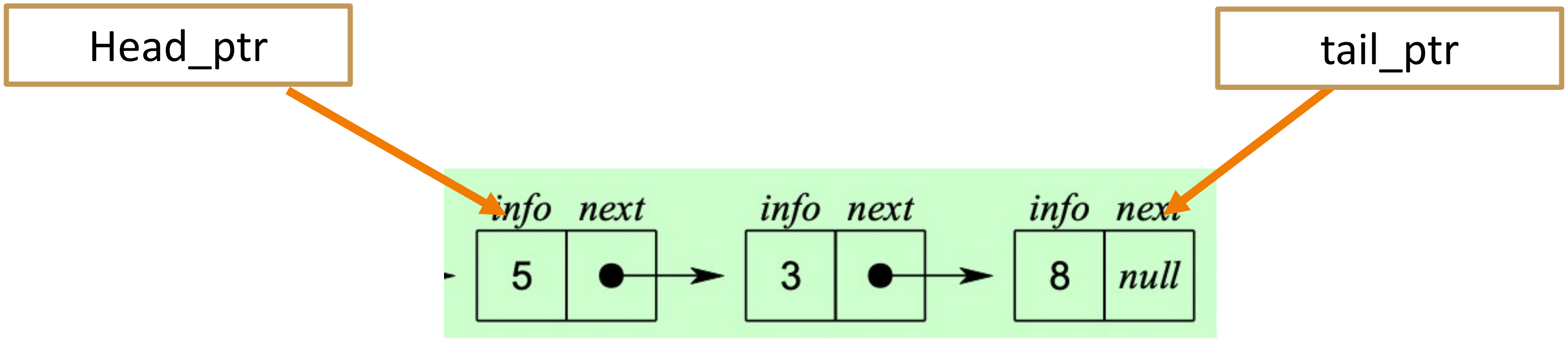
1. Mining dataset for evictions in LV ($20 per hour X 15 hours per week) for this semester.

- Working with me and social science.

Up-to-date information

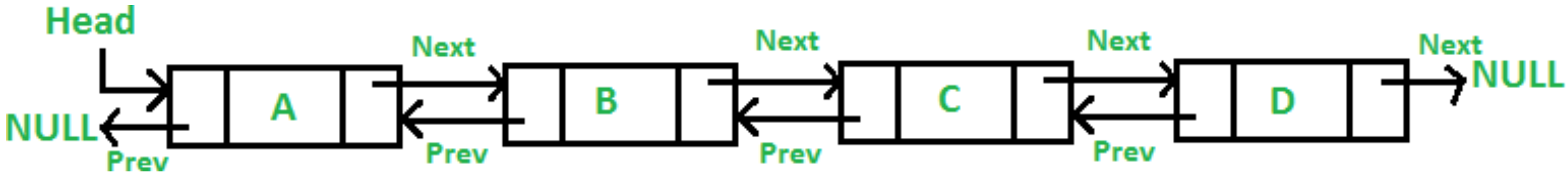
1. Google CEO Sundar: Google an AI first company
2. Microsoft is aggressively investing in healthcare AI

Linked List



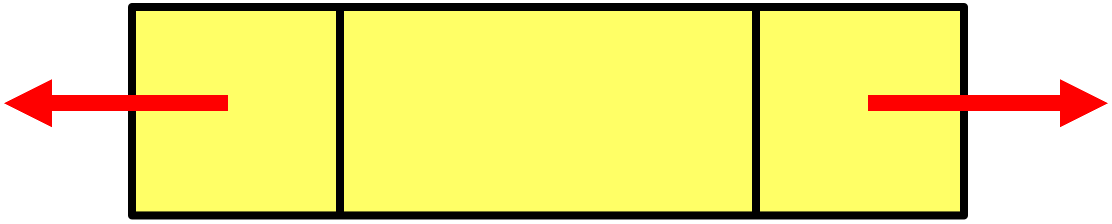

```
Struct Node
{
    typedef double Item;
    Item data;
    Node *link;
};

Node *head_ptr;
Node *tail_ptr;
```

Doubly Linked List



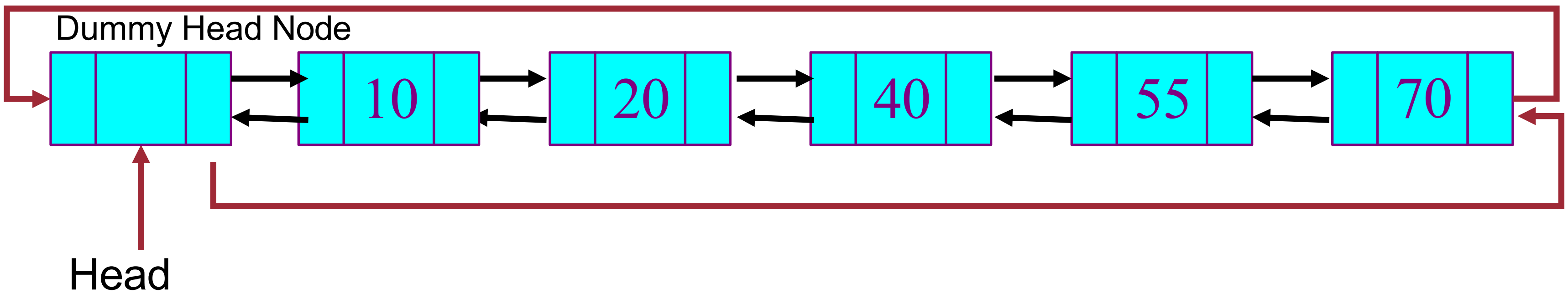
Node data

- info: the user's data
- next, back pointer: the address of the next and previous node in the list

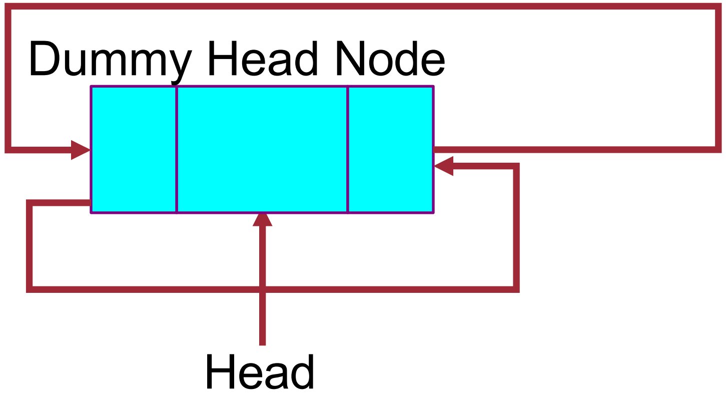


Doubly Linked Lists with Dummy Head

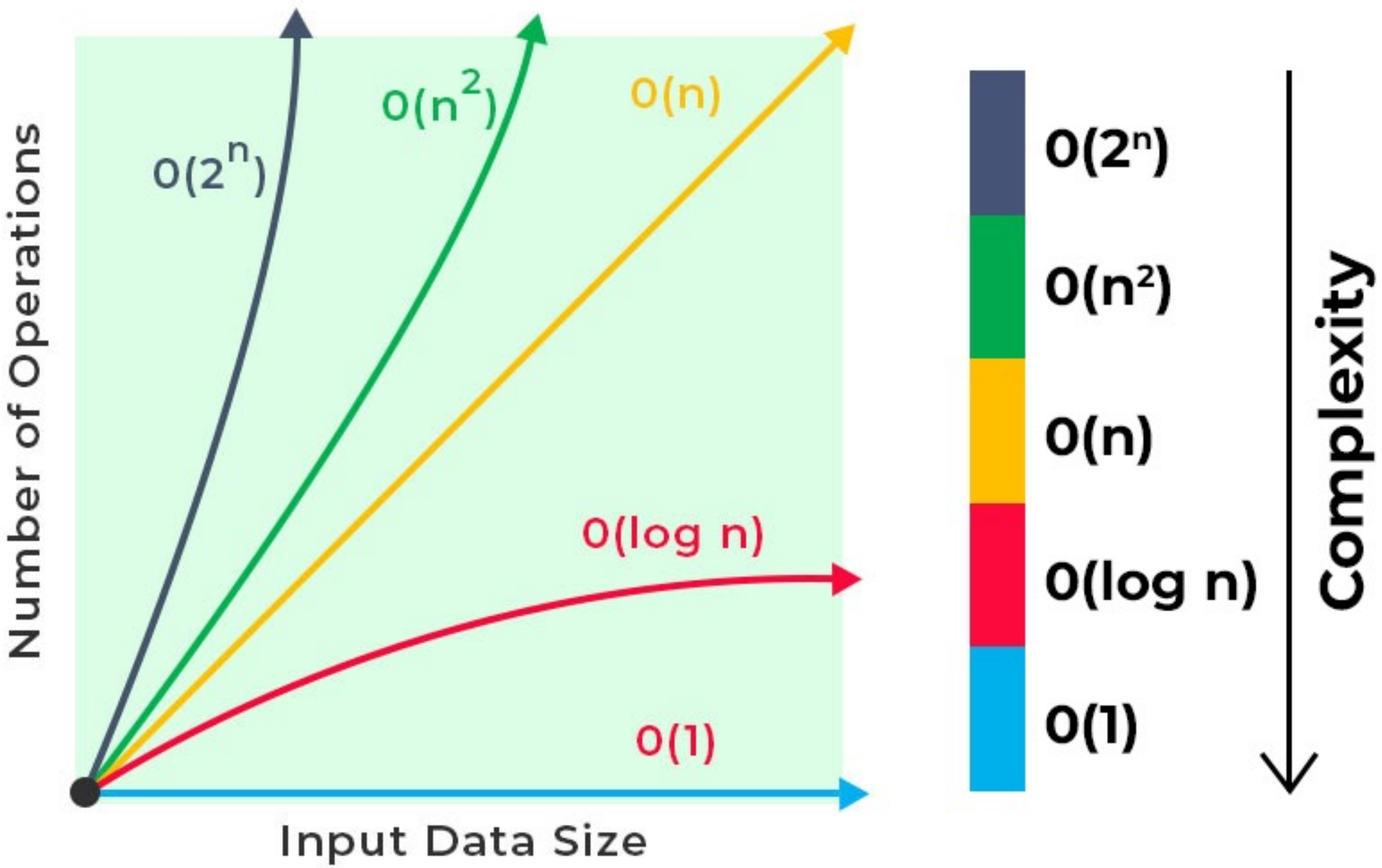
– Non-Empty List



– Empty List



Computational Complexity



Constant Time Operations

- Assigning the value to a variable: x = y
- Mathematical operations: x = y + 2 * z
- Comparisons: if (x > max) max = x
- Accessing a value in an array: x = y[3]

Linear Operations

```
for (let i = 0; i < n; i ++){
        cout << i;
}
```

The number of additions depends on the length of the array. Hence the run time is $O(n)$

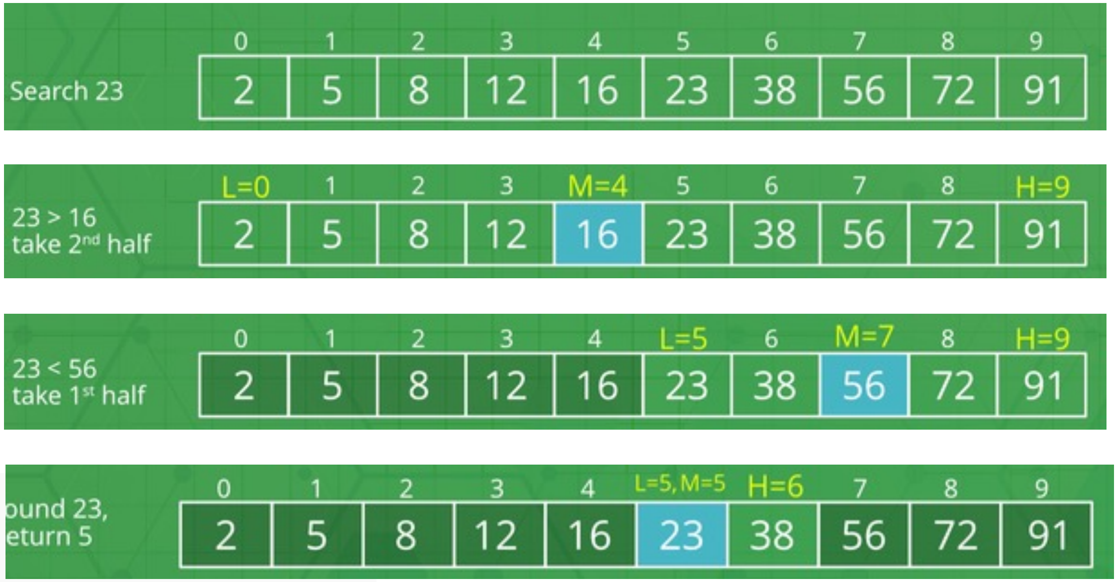
Quadratic Operations

```
for (let i = 0; i < n; i ++){
        for (let j = 0; j < m; j++){
                cout << i << " " << j;
}
}
```

$O(n^2)$

Computational Complexity

- O(logN)
- Binary search:




```
binarySearch(arr, x, low, high)
       if low > high
           return False

       else
           mid = (low + high) / 2
               if x == arr[mid]
               return mid

           else if x > arr[mid]        // x is on the right side
               return binarySearch(arr, x, mid + 1, high)

           else                        // x is on the left side
               return binarySearch(arr, x, low, mid - 1)
```

Asymptotic Complexity

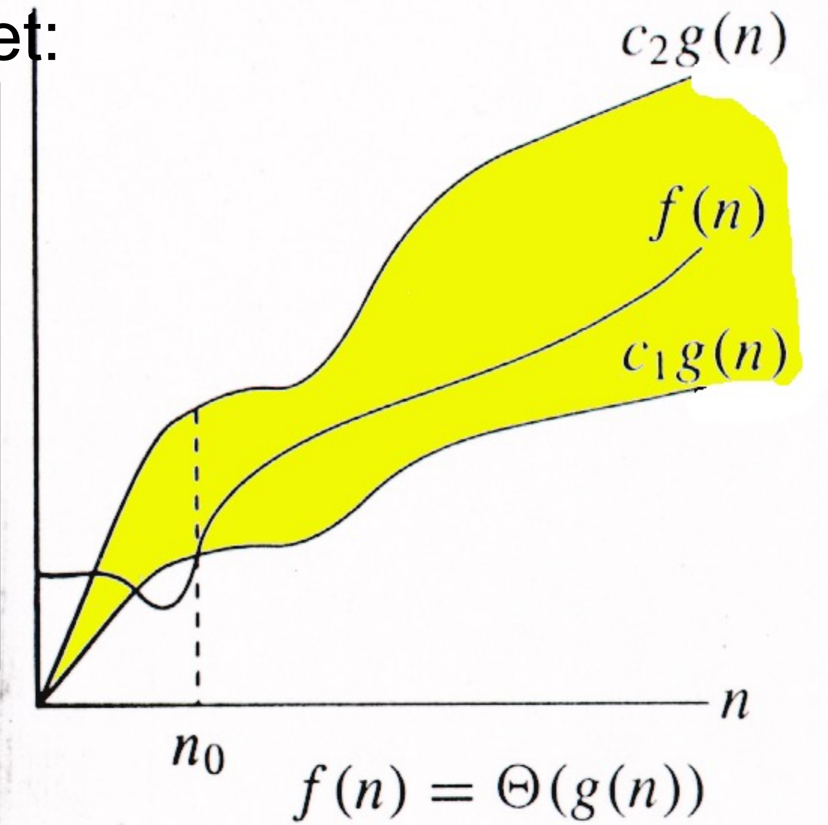
- Running time of an algorithm as a function of input size *n* **for large *n***.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notation***.

Asymptotic Notation

- **Θ, *O*, Ω, *o*, ω**
- Defined for functions over the natural numbers.
  - **Ex:** $f(n) = \Theta(n^2)$.
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:



$\Theta(g(n)) = \{f(n) :$
$\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$

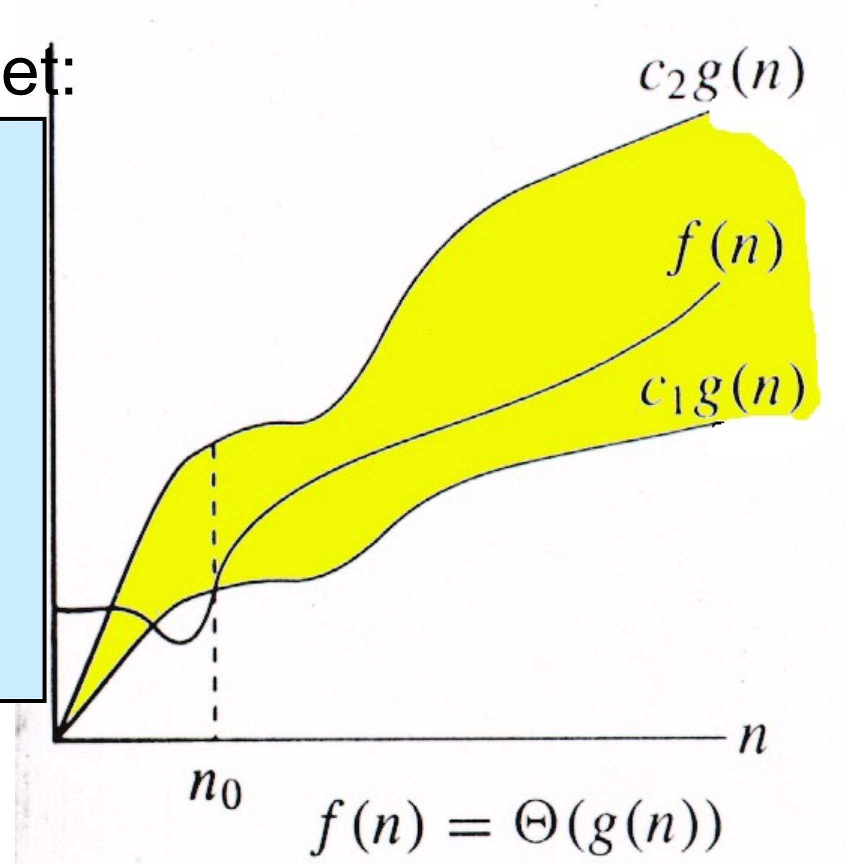
Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.

***f*(*n*) and *g*(*n*) are nonnegative, for large *n*.**

Example

$\Theta(g(n)) = \{f(n)$: $\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

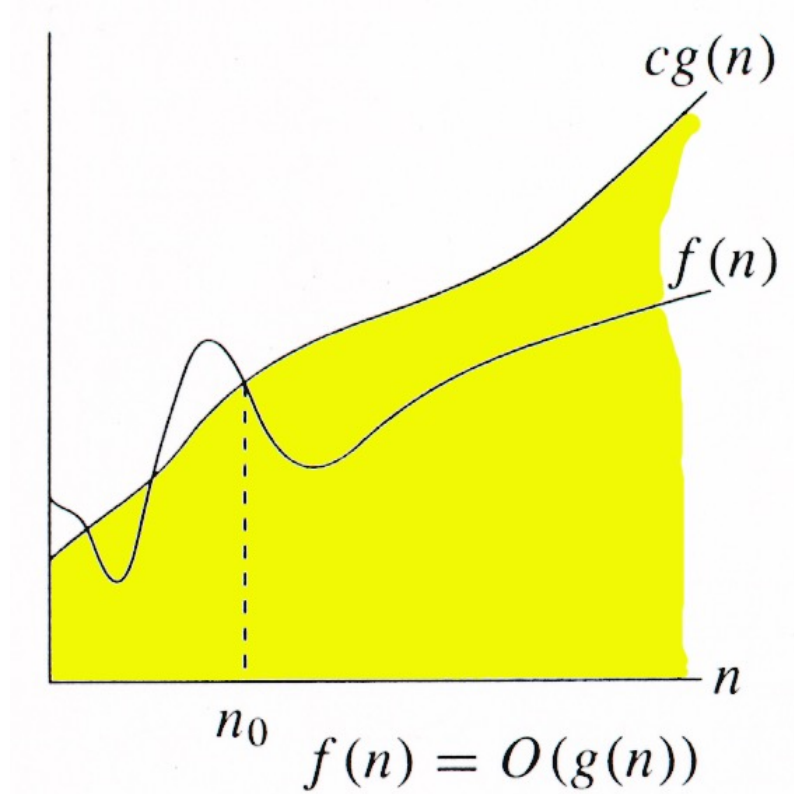
- $10n^2 - 3n = \Theta(n^2)$
- What constants for n_0, c_1, and c_2 will work?
- Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.
- ***To compare orders of growth, look at the leading term.***
- Exercise: Prove that $n^2/2 - 3n = \Theta(n^2)$

O-notation

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n)$:
$\exists$ **positive constants c and n_0,**
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)$ $\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.



$g(n)$ is an *asymptotic upper bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.
$\Theta(g(n)) \subset O(g(n))$.

Examples

$O(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n)$ $\}$

- Any linear *function* $an + b$ is in $O(n^2)$.

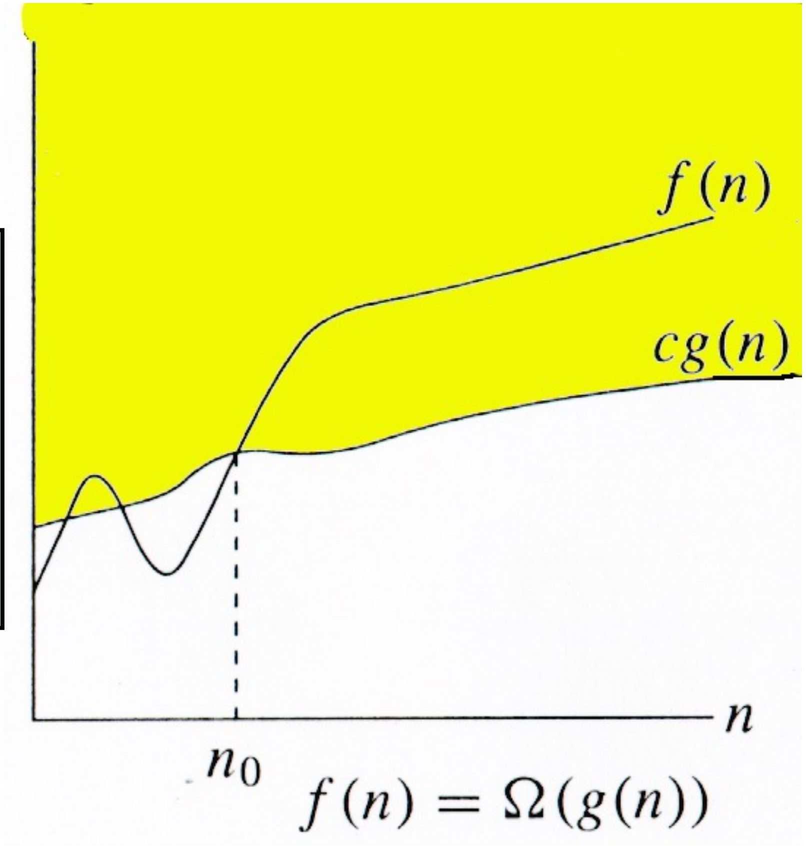
Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,

we have $0 \leq cg(n) \leq f(n)\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.



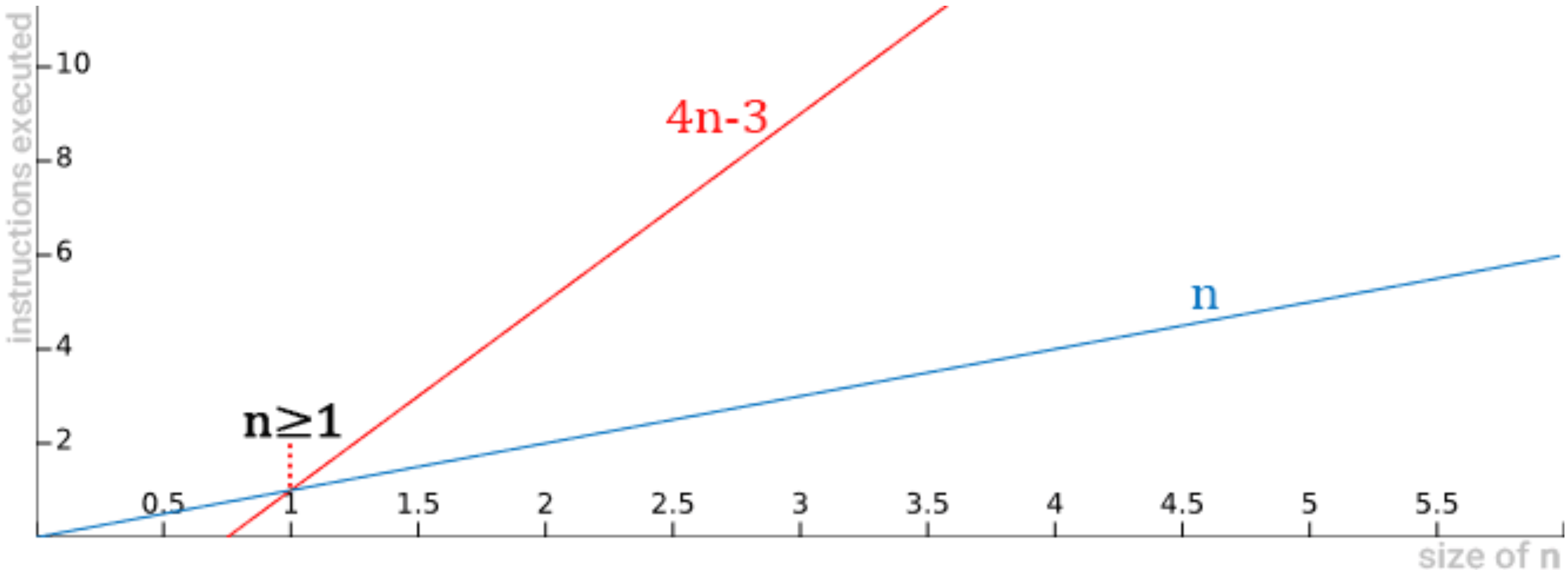
$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

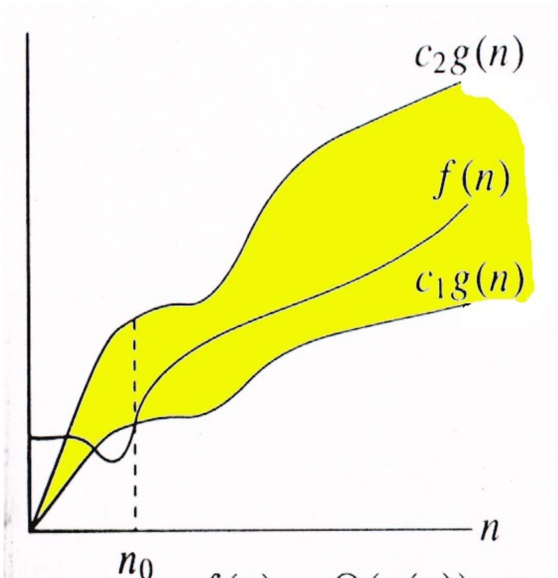
Example

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$

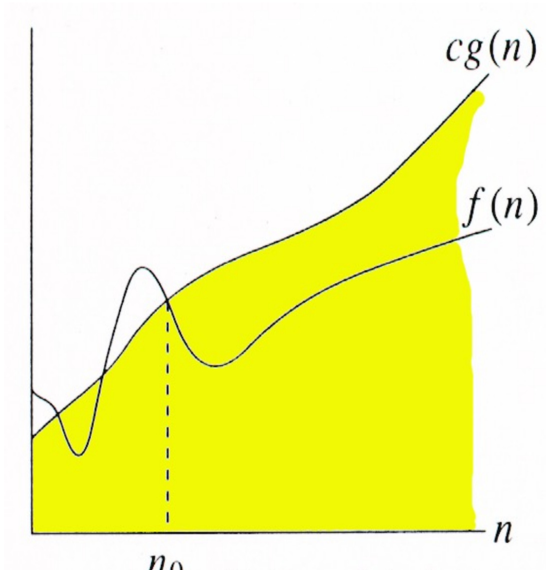
When we say that the function **4n-3** is Ω(**n**),



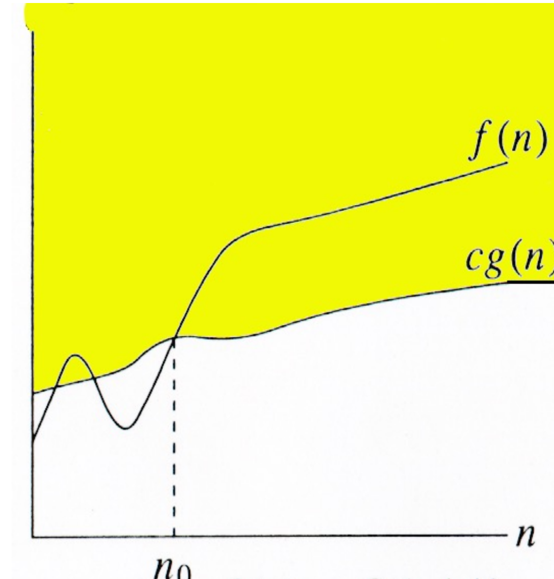
Relations Between Θ, *O*, Ω



$f(n) = \Theta(g(n))$



$f(n) = O(g(n))$



$f(n) = \Omega(g(n))$

Relations Between Θ, Ω, O

Theorem : For any two functions $g(n)$ and $f(n)$,

$$f(n) = \Theta(g(n)) \text{ iff}$$

$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

- I.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Running Times

- "Running time is $O(f(n))$" $\Rightarrow$ Worst case is $O(f(n))$
- $O(f(n))$ bound on the worst-case running time $\Rightarrow$ $O(f(n))$ bound on the running time of every input.
- $\Theta(f(n))$ bound on the worst-case running time $\not\Rightarrow$ $\Theta(f(n))$ bound on the running time of every input.
- "Running time is $\Omega(f(n))$" $\Rightarrow$ Best case is $\Omega(f(n))$
- Can still say "Worst-case running time is $\Omega(f(n))$"
  - Means worst-case running time is given by some unspecified function $g(n) \in \Omega(f(n))$.

Asymptotic Notation in Equations

- Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- For example,

  $4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$

  $= 4n^3 + \Theta(n^2) = \Theta(n^3)$. **How to interpret?**
- In equations, $\Theta(g(n))$ always stands for an ***anonymous function*** $f(n) \in \Theta(g(n))$
  - In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

$\Theta(g(n)) \subset O(g(n))$

$\Theta(g(n)) = O(g(n)) \cap W(g(n))$

Common Functions

Monotonicity

- $f(n)$ is
  - **monotonically increasing** if $m \leq n \Rightarrow f(m) \leq f(n)$.
  - **monotonically decreasing** if $m \geq n \Rightarrow f(m) \geq f(n)$.
  - **strictly increasing** if $m < n \Rightarrow f(m) < f(n)$.
  - **strictly decreasing** if $m > n \Rightarrow f(m) > f(n)$.

Exponentials

- **Useful Identities:**

$$a^{-1} = \frac{1}{a}$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

- **Exponentials and polynomials**

$$\lim_{n \to \infty} \frac{n^b}{a^n} = 0$$

$$\Rightarrow n^b = o(a^n)$$

Logarithms

x = $\log_b a$ is the exponent for $a = b^x$.

Natural log: $\ln a = \log_e a$

Binary log: $\lg a = \log_2 a$

$\lg^2 a = (\lg a)^2$

$\lg \lg a = \lg(\lg a)$

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

Logarithms and exponentials – Bases

- If the base of a logarithm is changed from one constant to another, the value is altered by a constant factor.
  - **Ex:** $\log_{10} n$ * $\mathbf{\log_2 10}$ = $\log_2 n$.
  - Base of logarithm is not an issue in asymptotic notation.
- Exponentials with different bases differ by a exponential factor (not a constant factor).
  - **Ex:** $2^n = \mathbf{(2/3)^n} * 3^n$.

Polylogarithms

- **For $a \geq 0$, $b > 0$,** $\lim_{n\to\infty} (\lg^a n \,/\, n^b) = 0$, so $\lg^a n = o(n^b)$, and $n^b = \omega(\lg^a n)$
  - Prove using L'Hopital's rule repeatedly

- $\lg(n!) = \Theta(n \lg n)$
  - Prove using Stirling's approximation (in the text) for $\lg(n!)$.

Exercise

```
for (int i = 0; i < n; i++){
        for (int j = 0; j< l; j++){
                cout << “hello”;
}
}
```

$O(n^2)$

Exercise

```
for (int i = 0; i < n; i=i*2){
        cout << “hello”;
}
```

O(log n)

Exercise

```
for (int i = 0; i < n; i++){
        for (int j = 0; j< n; j=j*2){
                cout << “hello”;
}
}
```

O(nlogn)

Exercise

```
for (int i = 0; i < n; i++){
        for (int j = 0; j< i; j=j*2){
                cout << “hello”;
}
}
```

O(nlogn)

Exercise

```
for (int i = n; i > 0; i=i/2){
        for (int j = 0; j< n; j++){
                cout << "hello";
}
}
```

O(nlogn)

Exercise

```
for (int i = n; i > 0; i=i/2){
        for (int j = 0; j< i; j++){
                cout << "hello";
}
}
```

O(n)