CS216: Program and Data Representation
University of Virginia Computer Science
Spring 2006 — David Evans

Lecture 3: Levels of Abstraction

http://www.cs.virginia.edu/cs216

Menu

- Orders of Growth: O, Ω, Θ, o
- Levels of Abstraction
- List Datatype

Recap

- Big-O: the set $O(f)$ is the set of functions that *grow* **no faster than** f
  - There exist positive integers c, $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Omega (Ω): the set $\Omega(f)$ is the set of functions that *grow* **no slower than** f
  - There exist positive integers c, $n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.

Question from last class

Given $f \in O(h)$ and $g \notin O(h)$ which of these are true:

a. For **all** positive integers m, $f(m) < g(m)$.

Proved **false** by counterexample

Statement a is **false**, so opposite of a must be **true**. What is the opposite of statement a?

a $\equiv \forall f \in O(h)\ \forall g \notin O(h)\ \forall m \in Z+, f(m) < g(m)$.

Question from last class

Given $f \in O(h)$ and $g \notin O(h)$ which of these are true:

a. For **all** positive integers m, $f(m) < g(m)$.

a $\equiv \forall f \in O(h)\ \forall g \notin O(h)\ \forall m \in Z+, f(m) < g(m)$.

a is false $\Rightarrow$ **not a** $\equiv$

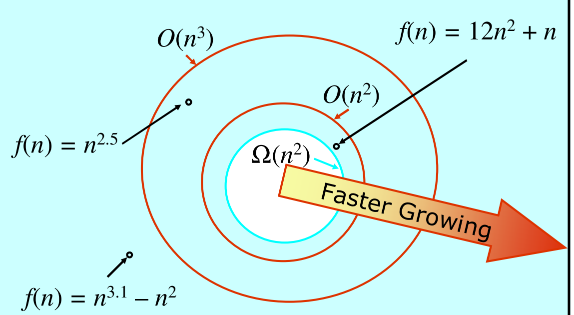
$\exists f \in O(h)\ \exists g \notin O(h)\ \exists m \in Z+, f(m) \geq g(m)$.

(this is exactly our counterexample)

Note this is very different from a claim like,

$\forall f \in O(h)\ \forall g \notin O(h)\ \exists m \in Z+, f(m) \geq g(m)$.

What else might be useful?

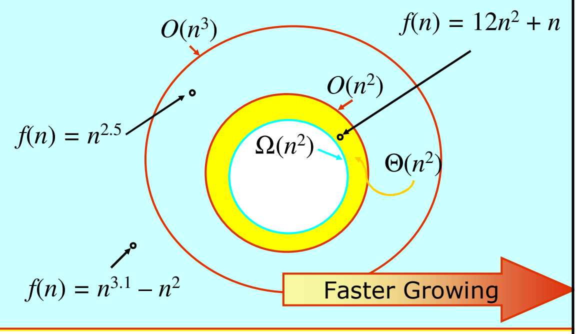


Theta ("Order of")

- Intuition: the set $\Theta(f)$ is the set of functions that *grow* **as fast as** f
- Definition: $f(n) \in \Theta(g(n))$ if and only if both:
  1. $f(n) \in O(g(n))$

  and 2. $f(n) \in \Omega(g(n))$
  – Note: we do not have to pick the same c and n_0 values for 1 and 2
- When we say, "f is order g" that means $f(n) \in \Theta(g(n))$

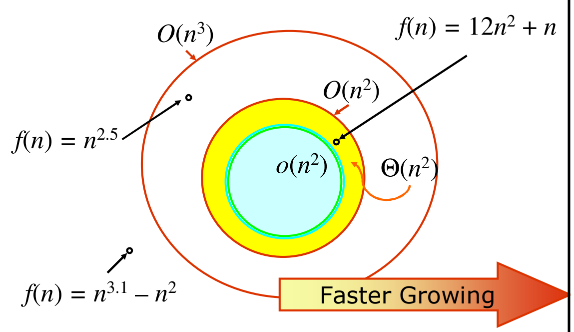
Tight Bound Theta (Θ)



Little-oh (o)

- Definition: $f \in o(g)$: **for all** positive constants c there is a value n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Compare: $f \in O(g)$: there are positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Little-Oh (o)



Summary

- Big-O: there exist c, $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Omega (Ω): there exist c, $n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.
- Theta (Θ): both O and Ω are true
- Litte-o: there exists $n_0 > 0$ such that **for all** $c > 0$, $f(n) \leq cg(n)$ for all $n \geq n_0$.

(Trick) Question

If $\text{wealth}(n)$ is your net worth n days after today, would you prefer:

a. $\text{wealth}(n) \in O(n)$
b. $\text{wealth}(n) \in O(n^2)$
c. $\text{wealth}(n) \in o(n)$
d. $\text{wealth}(n) \in \Omega(n)$

Which of these are satisfied by $\text{wealth}(n) = 0.0001n$?
Which is better:
$\text{wealth}(n) = 100000000$
$\text{wealth}(n) = 0.0001n$

Levels of Abstraction

Course Goal 3

Understand how a program executes at levels of abstraction ranging from a high-level programming language to machine memory.

From Lecture 1…

Levels of Abstraction: Program

Real World Problem

High-Level Program

Physical World

Virtual World

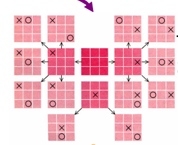
Machine Instructions

Physical Processor

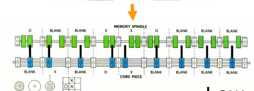
http://www.rci.rutgers.edu/~cfs/472_html/Intro/TinkertoyComputer/TinkerToy.html

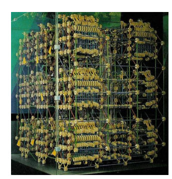
Tic-Tac-Toe

Play Tic-Tac-Toe



Tic-Tac-Toe Strategy





Low-level description

Tinker Toy Computer

Sequence Alignment: Program

Genome Similarity

High-Level Program

Physical World

Virtual World

Python Interpreter

Align.py

Low-Level Program

Electrons, etc.

Levels of Abstraction: Data

Real World Thing(s)

Data Abstraction

Physical World

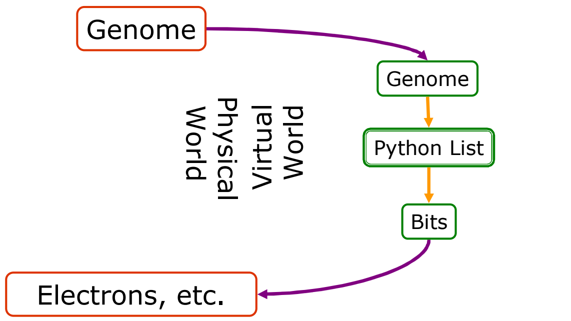
Virtual World

Low-Level Data Structure

Bits

Electrons, etc.

Levels of Abstraction: PS1



List Abstract Datatype

- Ordered collection datatype:
  $<x_0, x_1, \ldots, x_{n-1}>$
- Operations for manipulating and observing elements of list

List Operations (Ch 3)

- *Access* (L, i): returns the i^{th} element of L
- *Length* (L): returns the number of elements in L
- *Concat* (L, M): returns the result of concatenating L with M. (Elements $<l_0, l_1, \ldots, l_{|L|-1}, m_0, m_1 \ldots, m_{|M|-1}, >$)
- *MakeEmptyList*(): returns $<>$
- *IsEmptyList*(L): returns true iff $|L| = 0$.

Is this a sufficient list of List operations?

Constructing Lists

- The book's list operations have no way of constructing any list other than the empty list!
- We need at least:
  - *Append* (L, e): returns the result of appending e to L: $<l_0, l_1, \ldots, l_{|L|-1}, e>$

Revised List Operations

Can define using *Append*, *Access*, *Length*

- *Access* (L, i): returns $L[i]$
- *Length* (L): returns $|L|$
- *Concat* (L, M): returns the result of concatenating L with M.
- *MakeEmptyList*(): returns $<>$
- *IsEmptyList*(L): returns true iff $|L| = 0$.
- *Append* (L, e): returns the result of appending e to L: $<l_0, l_1, \ldots, l_{|L|-1}, e>$

Easy to define using Length

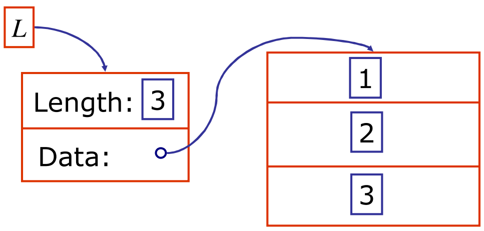
Are all of these operations necessary?

Necessary List Operations

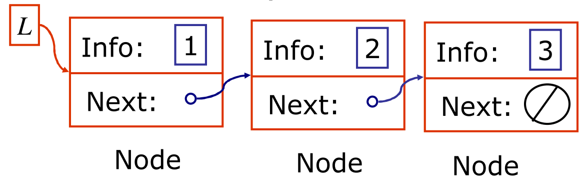
- *Access* (L, i): returns $L[i]$
- *Length* (L): returns $|L|$
- *MakeEmptyList*(): returns $<>$
- *Append* (L, e): returns the result of appending e to L: $<l_0, l_1, \ldots, l_{|L|-1}, e>$

Note that we have defined an *immutable* list. There are no operations for changing the value of a list, only making new lists.

Continuous Representation



Linked Representation



We need a special value for Next when there is no Next node:

Book: Λ C: 0

Python: None Scheme, Java: null

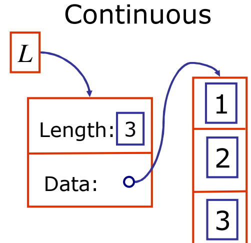
Necessary List Operations

- *Access* (L, i): returns $L[i]$
- *Length* (L): returns $|L|$
- *MakeEmptyList*(): returns <>
- *Append* (L, e): returns the result of appending e to L: $<l_0, l_1, \ldots, l_{|L|-1}, e>$

Can we implement all of these with both representation choices?

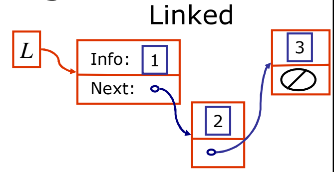
Length

Continuous




```
def Length(L):
   return L.Length
```

Linked




```
def Length(L):
   if L == None:
      return 0
   return 1 + \
      Length(L.Next)
```


Which Representation is Better?

- Time of Length: (n is number of elements)
  - Continuous: $O(1)$
  - Linked: $O(n)$
  - Are these bounds tight? (Θ)
- What about other operations?
- Other factors to consider?

Will explore this more next week

Python Lists

- Provide necessary operations:
  - *Access* (L, i): L[i]
  - *Length* (L): len(L)
  - *MakeEmptyList*(): L = []
  - *Append* (L, e): L.append (e)

Python List Operations

- insert: L.insert (i, e)
  Returns <l_0, l_1, …, l_{i-1}, e, l_i, …, $l_{|L|-1}$>
- concatenation: L **+** M
  Returns <l_0, l_1, …, $l_{|L|-1}$, m_0, m_1, …, $m_{|M|-1}$ >
- slicing: L[from:to]
  Returns <l_{from}, l_{from+1}, …, l_{to-1} >
- Lots more

How are they implemented?

- PS1
  - Try to "guess" by measuring performance of different operations
  - Unless you can do exhaustive experiments (hint: you can't) you can't be assured of a correct guess
- Around PS4:
  - Look an lower abstraction level: C code for the Python List implementation

Charge

- Problem Set 1 due Monday
- Point of PSs is to learn:
  - You can (and should) discuss your approaches and ideas with anyone
  - You should discuss and compare your answers to 1-6 with your assigned partner and produce a consensus best answer that you both understand and agree on
- Take advantage of Small Hall On-Call Hours:
  - Wednesday 7-8:30pm
  - Thursday 4-5:30pm, 6:30-8:30pm
  - Friday 11am-12:30, 3:30-5pm
  - Saturdays 3-6pm
  - Sunday 3:30-9:30pm
- Monday: Dynamic Programming