

# CS2010 – Data Structures and Algorithms II

## Lecture 02 – Census Problem

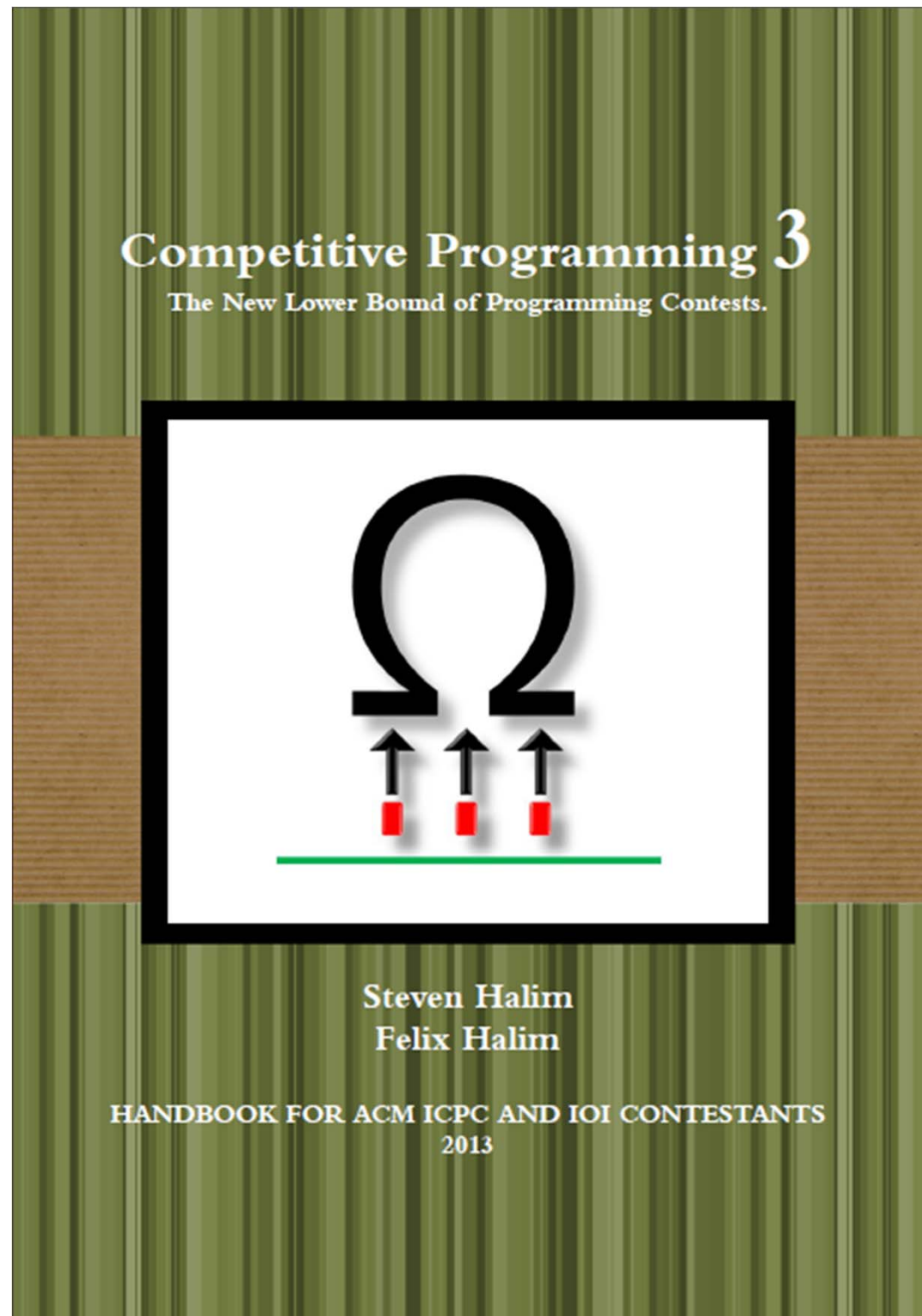
[stevenhalim@gmail.com](mailto:stevenhalim@gmail.com)



# CP3 Book

Out of stock now...

I will reprint more  
copies soon



# Tutorial, Lab, Coursemology Status

See [IVLE](#) for Tutorial and Lab Status

See [Coursemology](#) for Enrollment Status

- You cannot download PS1 (opened soon)  
if you do not enroll

# Outline

## Motivation: Census Problem

- Abstract Data Type (ADT) Table
- Solving Census Problem with CS1020 Knowledge
- The “performance issue”

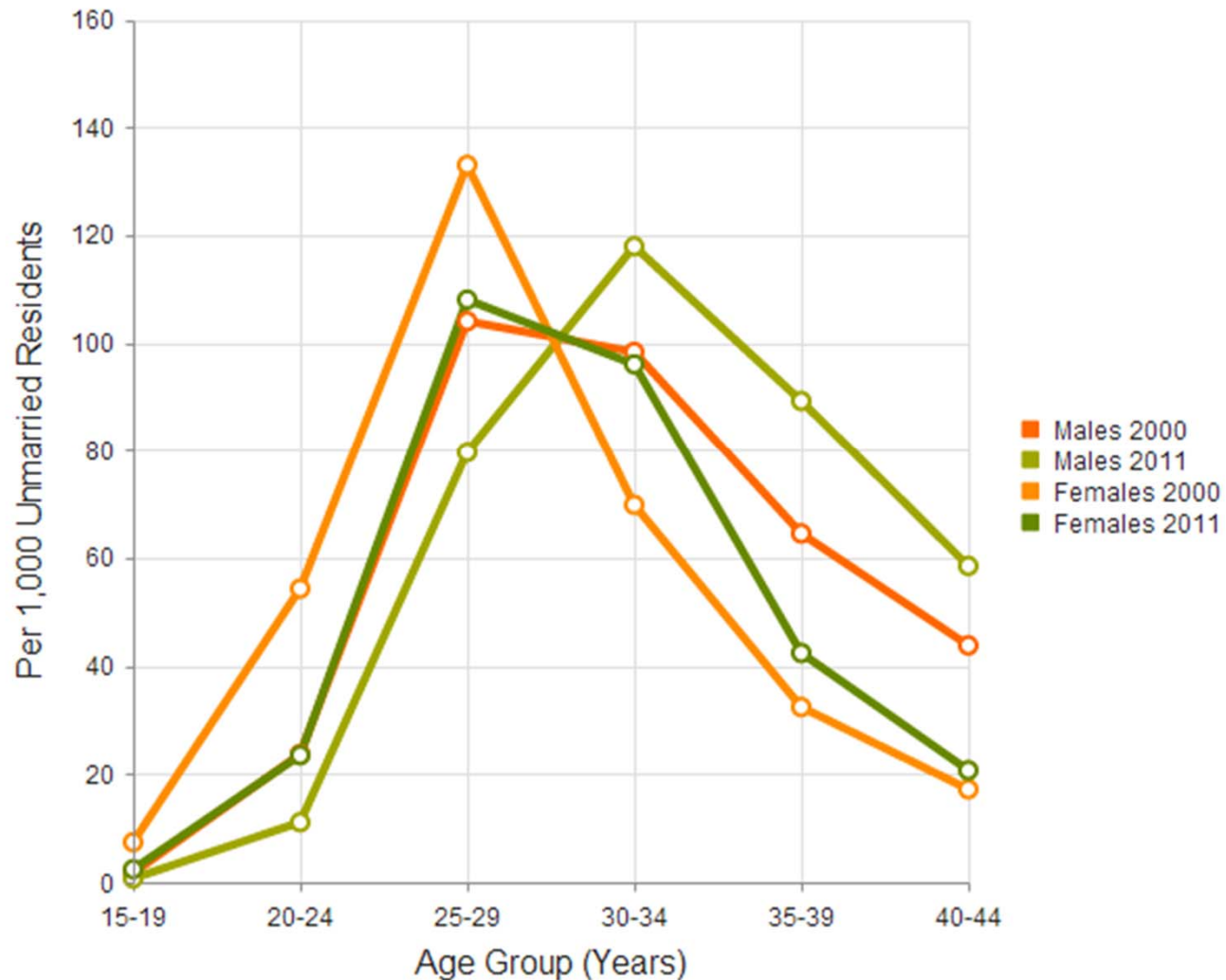
## Binary Search Tree (BST)

- Web-based lecture
- Simple analysis of BST operations

Relation with CS2010 PS1: “The Baby Names Problem”

# Census is Important!

## Age-Specific Marriage Rates



Source: <http://www.singstat.gov.sg>



# Sun Tzu's Art of War

## Chapter 1 "The Calculations"

知彼知己百戰不殆

zhī bǐ zhī jǐ bǎi zhàn bù dài

(If you know your enemies and know yourself,  
you will not be imperiled in a hundred battles)

# Your Age (2013 data)

'[' (or '[') means that  
endpoint is included  
(closed)

1. [24 ...  $\infty$ )

2. [23 ... 24)

3. [22 ... 23)

4. [21 ... 22)

5. [20 ... 21)

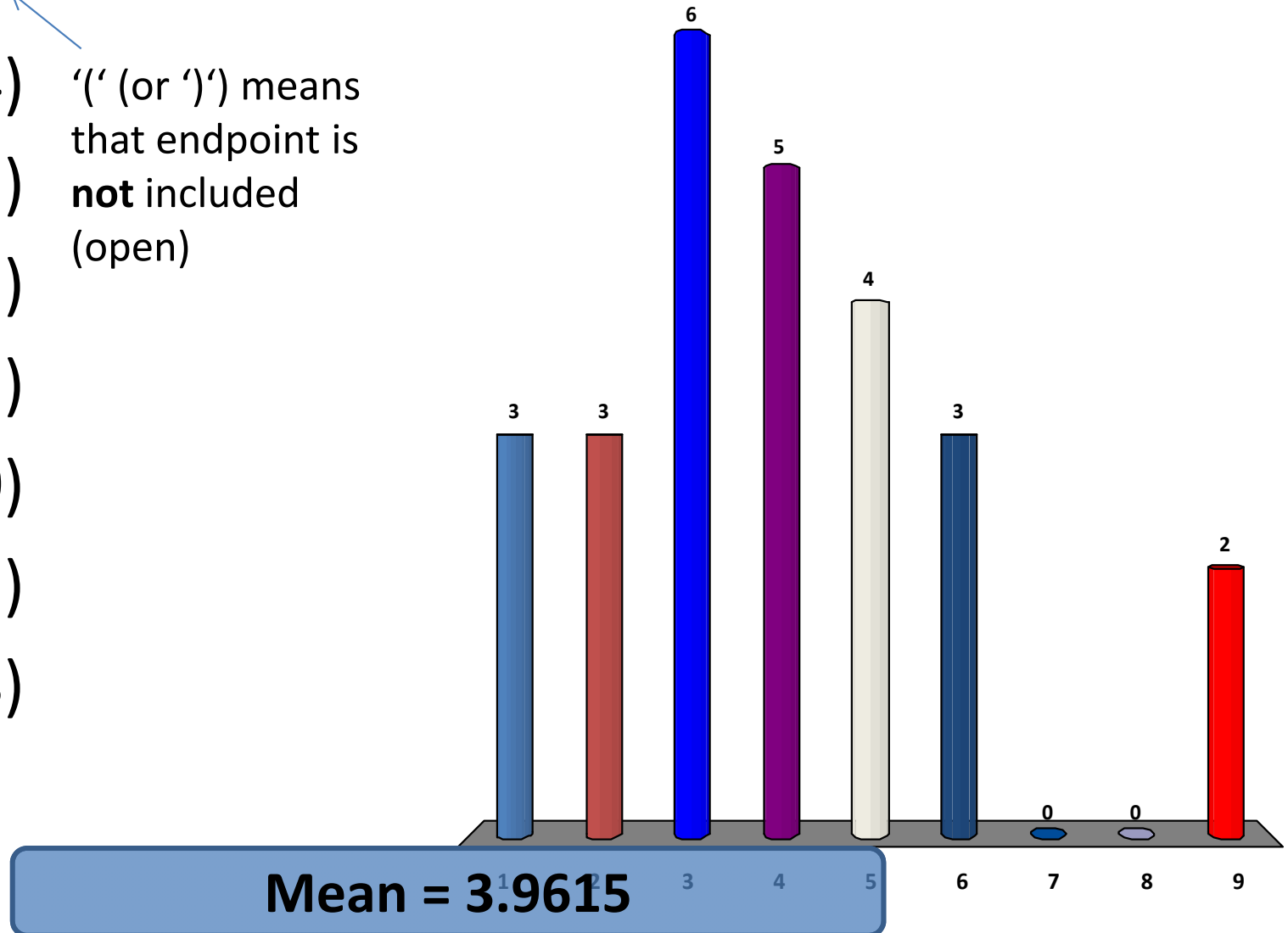
6. [19 ... 20)

7. [18 ... 19)

8. [17 ... 18)

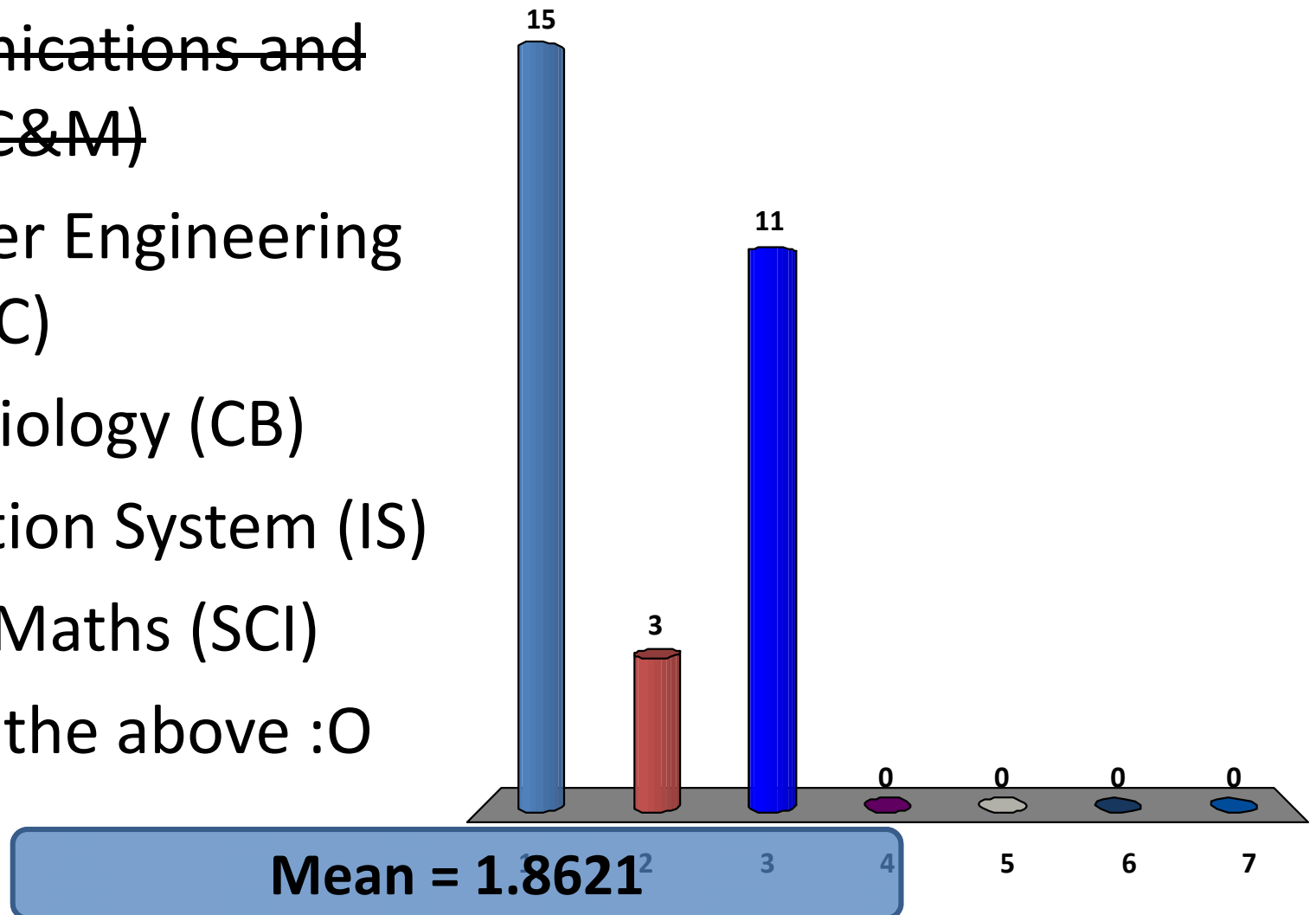
9. [0 ... 17)

'(' (or '(') means  
that endpoint is  
**not** included  
(open)



# Your Major (2013 data)

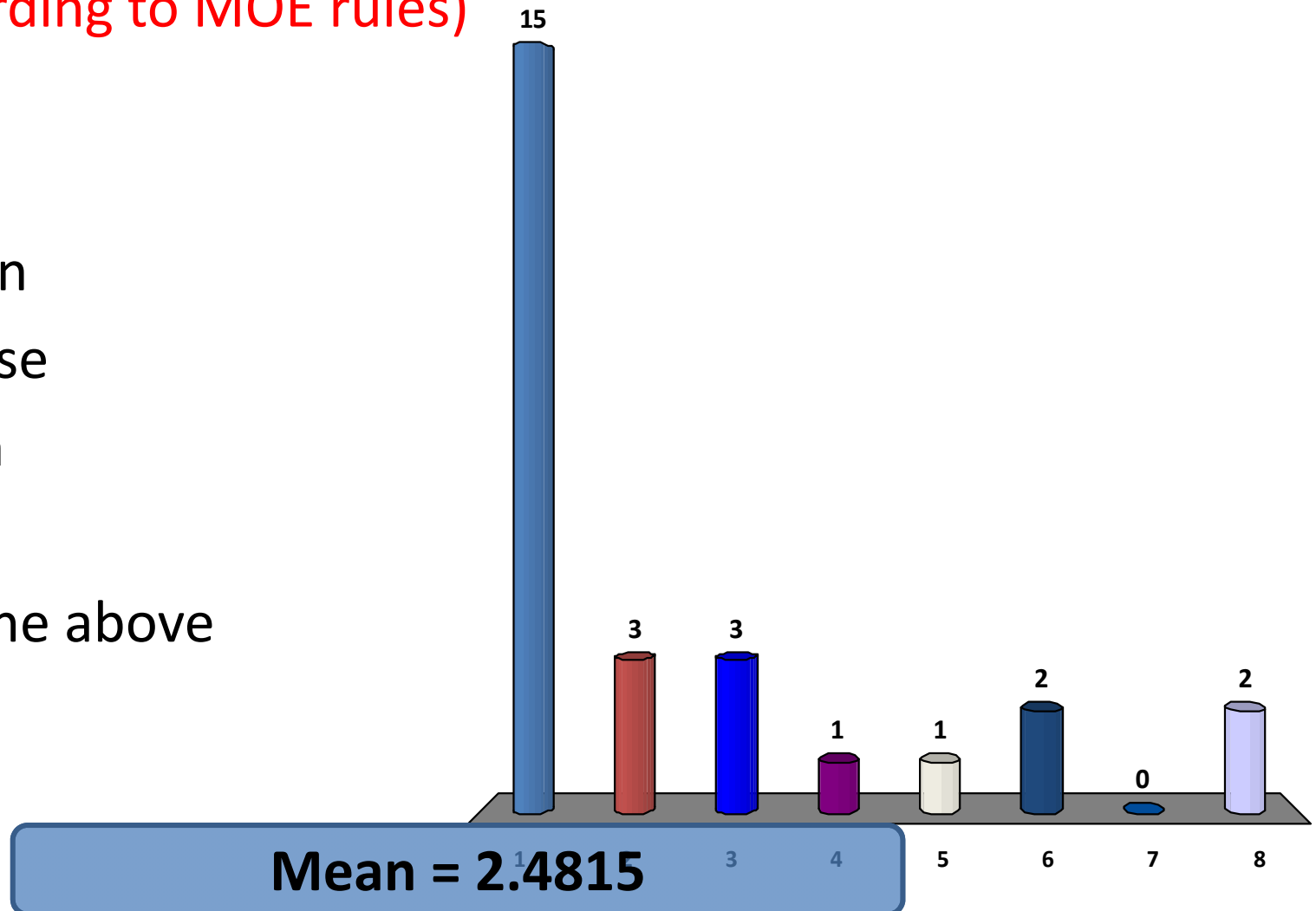
1. Computer Science (CS)
- ~~2. Communications and Media (C&M)~~
3. Computer Engineering (CEG/CEC)
4. Comp. Biology (CB)
5. Information System (IS)
6. Science Maths (SCI)
7. None of the above :O





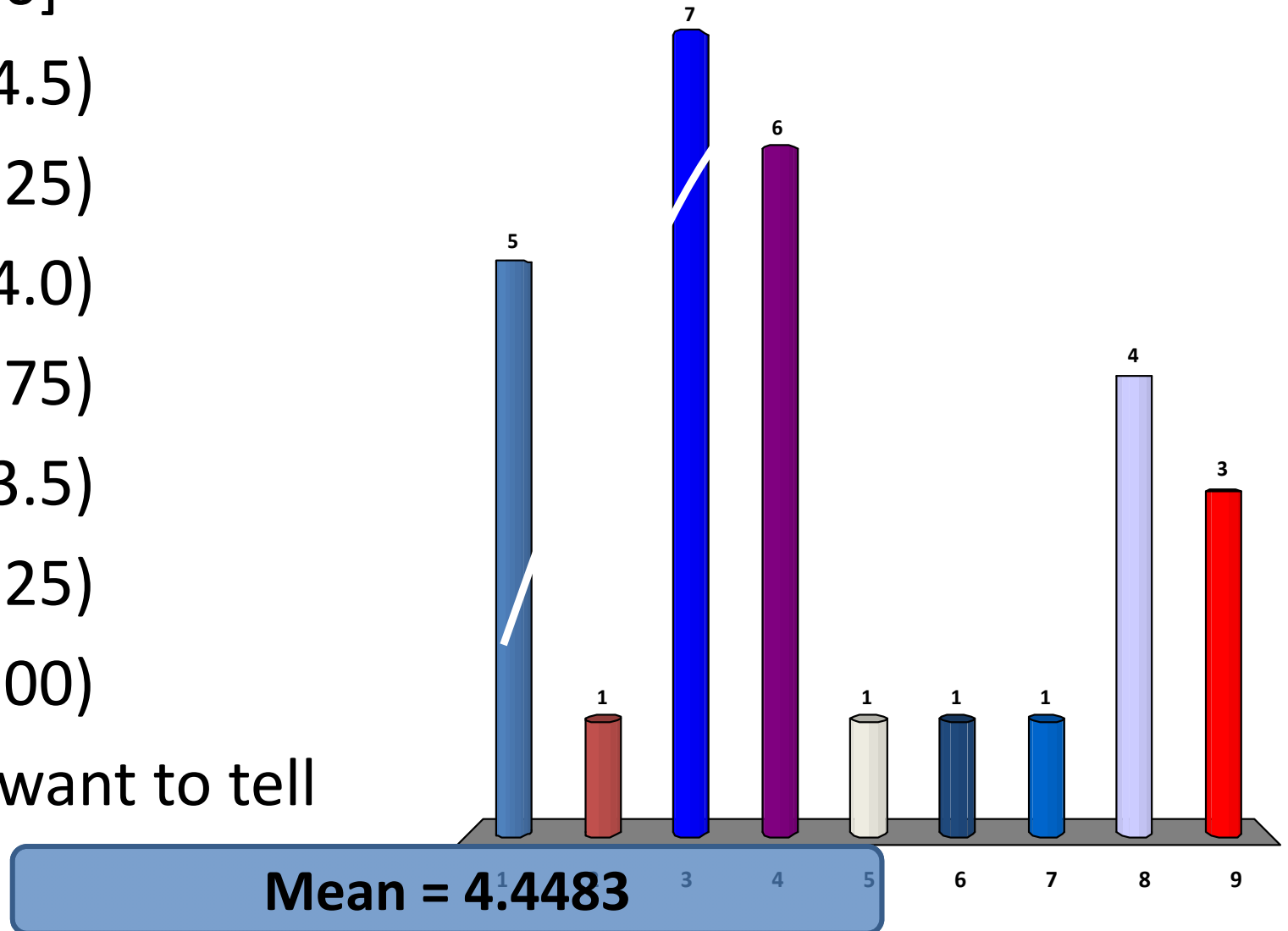
# Your Nationality (2013 data)

1. Singaporean (should be  $\geq$  70% according to MOE rules)
2. Chinese
3. Indian
4. Indonesian
5. Vietnamese
6. Malaysian
7. European
8. None of the above



# Your CAP (2013 data)

1. [4.5 ... 5.0]
2. [4.25 ... 4.5)
3. [4.0 ... 4.25)
4. [3.75 ... 4.0)
5. [3.5 ... 3.75)
6. [3.25 ... 3.5)
7. [3.0 ... 3.25)
8. [0.0 ... 3.00)
9. I do not want to tell



# What Happen After Census?

Data  
Mining



Statistical  
Analysis

# Abstract Data Type (ADT) Table

Let's deal with one aspect of our census: **Age**

To simplify this lecture, we assume that students' age ranges from  $[0 \dots 100)$ , all integers, and distinct

Required operations:

1. Search whether there is a student with a certain age?
2. Insert a new student (that is, insert his/her age)
3. Determine the youngest and oldest student
4. List down the ages of students in sorted order
5. Find a student slightly older than a certain age!
6. Delete existing student (that is, remove his/her age)
7. Determine the median age of students

# CS1020: Unsorted Array

Index	0	1	2	3	4	5	6	7	
A	5	7	71	50	23	4	6	15	

No	Operation	Time Complexity
1	Search(age)	$O(n)$
2	Insert(age)	$O(1)$
3	FindOldest()	$O(n)$
4	ListSortedAges()	$O(n \log n)$
5	NextOlder(age)	$O(n)$
6	Remove(age)	$O(n)$
7	GetMedian()	$O(n \log n)/O(n)$

# CS1020: Sorted Array

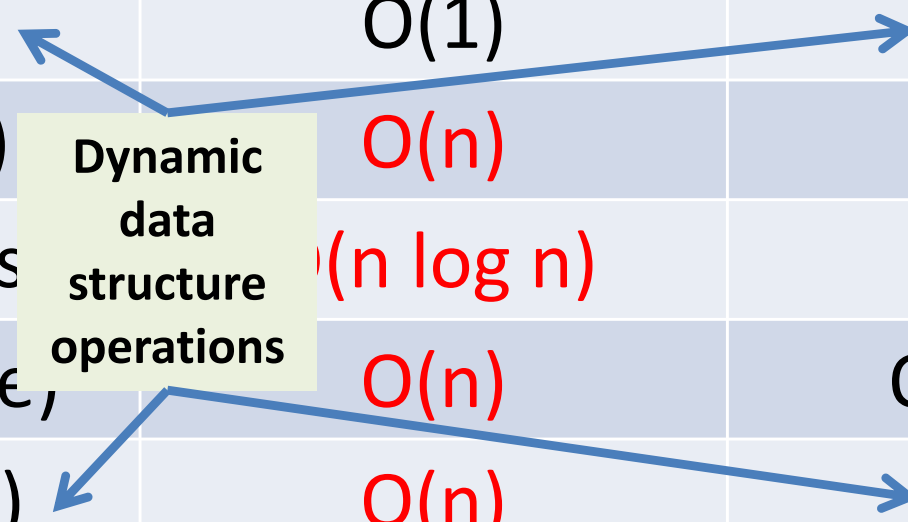
Index	0	1	2	3	4	5	6	7	
A	4	5	6	7	15	23	50	71	

No	Operation	Time Complexity
1	Search(age)	$O(\log n)$
2	Insert(age)	$O(n)$
3	FindOldest()	$O(1)$
4	ListSortedAges()	$O(n)$
5	NextOlder(age)	$O(\log n)$
6	Remove(age)	$O(n)$
7	GetMedian()	$O(1)$

# With Just CS1020 Knowledge

No	Operation	Unsorted Array	Sorted Array
1	Search(age)	$O(n)$	$O(\log n)$
2	Insert(age)	$O(1)$	$O(n)$
3	FindOldest()	$O(n)$	$O(1)$
4	ListSortedAges	$O(n \log n)$	$O(n)$
5	NextOlder(age, ...)	$O(n)$	$O(\log n)$
6	Remove(age)	$O(n)$	$O(n)$
7	GetMedian()	$O(n \log n) / O(n)$	$O(1)$

Dynamic data structure operations



If  $n$  is large, our queries are slow...



# $O(n)$ versus $O(\log n)$ : A Perspective



$$n = 8$$



$$\log_2 n = 3$$



$$n = 16$$



$$\log_2 n = 4$$



$$n = 32$$



$$\log_2 n = 5$$

Try larger  $n$ , e.g.  $n = 1000000...$



A Versatile, Non-Linear Data Structure

# **BINARY SEARCH TREE (BST)**

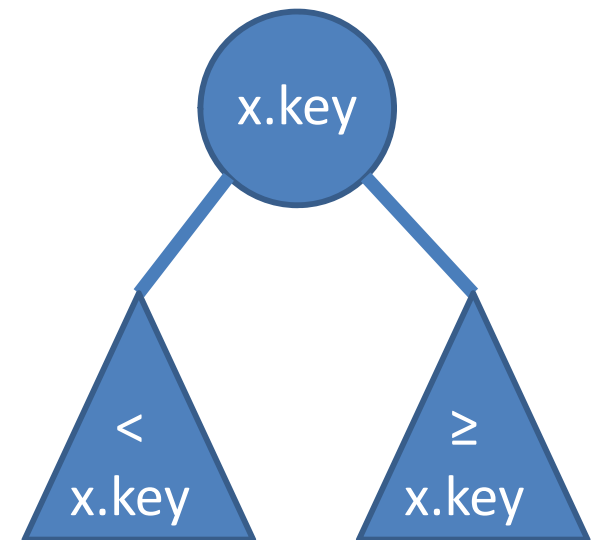
# Binary Search Tree (BST) Vertex

For every vertex  $x$ , we define:

- $x.\text{left}$  = the left child of  $x$
- $x.\text{right}$  = the right child of  $x$
- $x.\text{parent}$  = the parent of  $x$
- $x.\text{key}$  (or  $x.\text{value}$ ,  $x.\text{data}$ ) = the value stored at  $x$

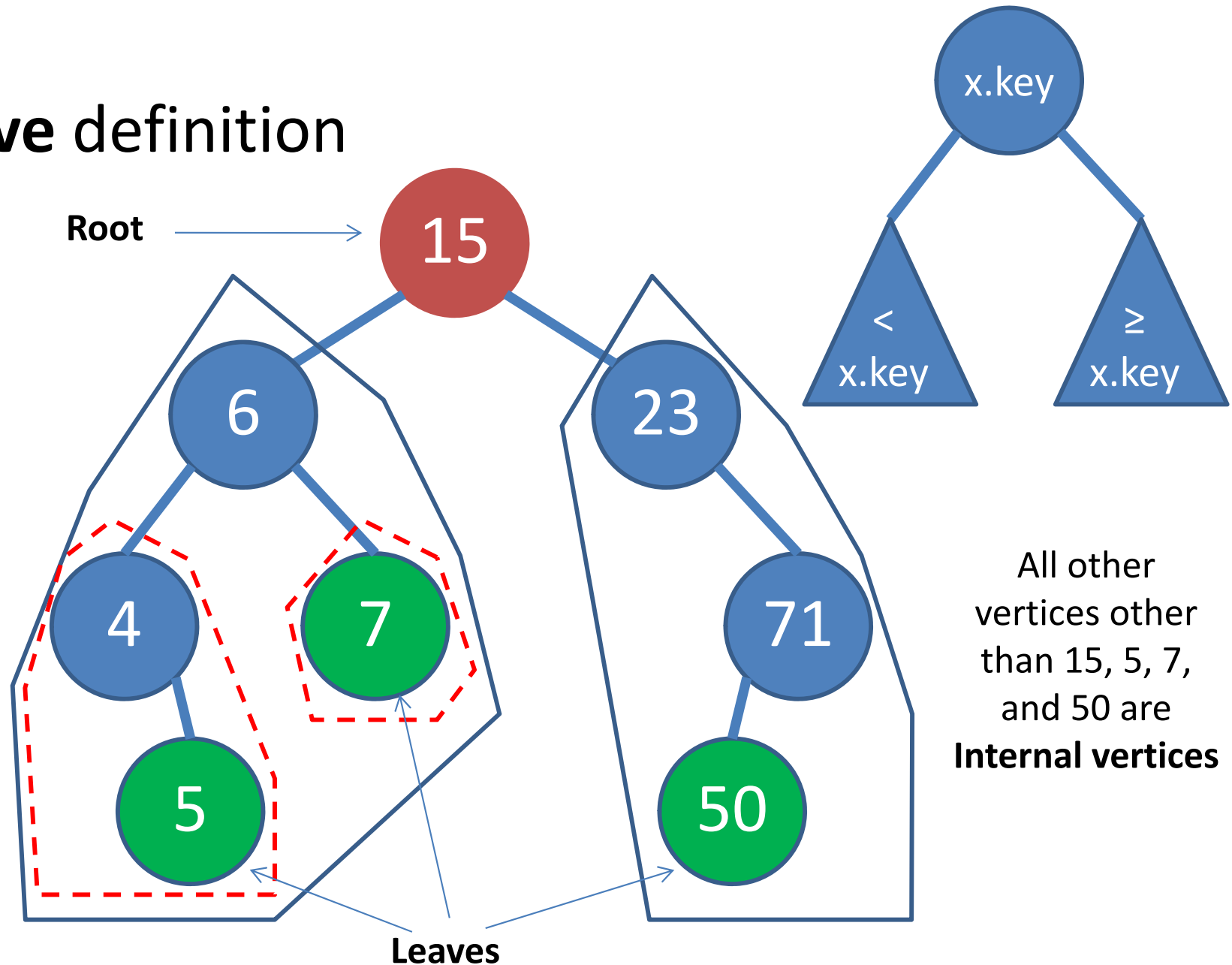
BST Property:

- $x.\text{left}.\text{key} < x.\text{key} \leq x.\text{right}.\text{key}$
- For simplicity, we assume that the keys are unique so that we can change  $\geq$  to  $>$



# BST: An Example, Keys = Ages

## Recursive definition



# NEW For this semester

We will use:

<http://www.comp.nus.edu.sg/~stevenha/visualization/bst.html>

during most part of this lecture

Note that since VisuAlgo is a 24/7 virtual copy of myself, we can try a semi flipped classroom learning style: Play with the BST visualization first before coming to CS2010 lecture 02

We will jump to [analysis slides](#) after web-based demo

# BST: Search/Min/Max Operations

Ask VisuAlgo to perform various search operations on the sample BST, including find min and find max

In the screen shot below, we show **search(5)**

The screenshot displays the VisuAlgo interface for a Binary Search Tree (BST). The tree structure is as follows:

- Root node: 15 (orange)
- Left child of 15: 6 (orange)
- Right child of 15: 23 (grey)
- Left child of 6: 4 (orange)
- Right child of 6: 7 (grey)
- Left child of 4: 5 (orange)
- Left child of 23: 50 (grey)
- Right child of 23: 71 (grey)

The search operation for the value 5 is shown in the bottom right panel. The search path is highlighted in orange, starting from the root 15, moving to 6, then to 4, and finally to 5.

Search for 5

```
Found node 5
```

```
if this == null
    return null
else if this key == search value
    return this
else if this key < search value
    search right
else search left
```

At the bottom of the interface, there is a control bar with buttons for "Create", "Search", "Insert", "Remove", "Successor", "Predecessor", and "In-order Traversal". The "Search" button is active, and the input field shows "5". The "GO" button is also visible.

# BST: Insert Operation

Ask VisuAlgo to perform various insert operations on the sample BST

In the screen shot below, we show **insert(20)**

The screenshot displays the VisuAlgo interface for performing a Binary Search Tree (BST) insert operation. The main area shows a BST with the following structure:

- Root: 15 (orange)
- Left child of 15: 6 (black)
- Right child of 15: 23 (orange)
- Left child of 6: 4 (black)
- Right child of 6: 7 (black)
- Left child of 4: 5 (black)
- Left child of 23: 20 (orange)
- Right child of 23: 71 (black)
- Left child of 71: 50 (black)

The node 20 is highlighted in orange, indicating it is the current node being inserted. The interface includes a menu on the left with options: Create, Search, Insert, Remove, Successor, Predecessor, and In-order Traversal. A search bar contains the value 20, and a GO button is next to it. On the right, a code block shows the insertion logic:

```
Insert 20
20 has been inserted!
if found insertion point
  create new node
if value to be inserted < this key
  go left
else go right
```

At the bottom, there is a progress bar and navigation controls, including a play button and a speed slider (slow to fast). The footer contains links for About, Team, and Terms of use.

# BST: Inorder Traversal Operation

Ask VisuAlgo to perform inorder traversal operation on the sample BST

In the screen shot below, we *partial* inorder traversal

In-order Traversal

```
Visit node 15.  
if this is null  
    return  
inOrder(left)  
visit this, then inOrder(right)
```

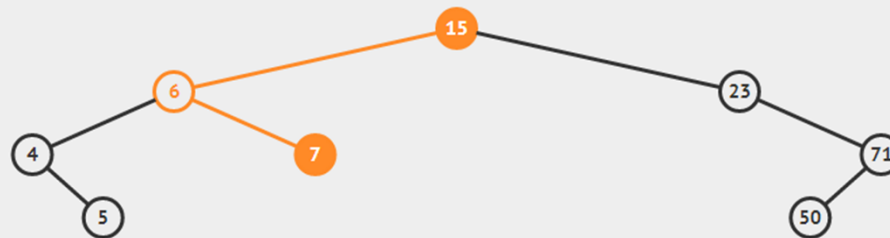
slow fast

About Team Terms of use

# BST: Succ/Pred-essor Operations

Ask VisuAlgo to perform Succ/Pred operations  
on the sample BST

In the screen shot below, we show **pred(15)**



Predecessor(15)

Predecessor found!

```
if this.left != null return findMax(this.left)
else
    p = this.parent, T = this
    while(p != null && T == p.left)
        T = p, p = T.parent
    if p is null return -1
    else return p
```

Create

Search

Insert

Remove

Successor

Predecessor

In-order Traversal

15 GO

Enter an integer

slow fast



About Team Terms of use



# BST: Delete/Remove Operation (1)

Ask VisuAlgo to perform various delete operations on the sample BST (3 cases, this is **delete leaf**)

In the screen shot below, we show **remove(5)** before deletion

The screenshot displays the VisuAlgo interface for performing a delete operation on a Binary Search Tree (BST). The tree structure is as follows:

- Root: 15 (orange)
- Left child of 15: 6 (orange)
- Right child of 15: 23 (grey)
- Left child of 6: 4 (orange)
- Right child of 6: 7 (grey)
- Left child of 4: 5 (orange)
- Left child of 23: 50 (grey)
- Right child of 23: 71 (grey)

The interface includes a sidebar on the left with the following options: Create, Search, Insert, Remove, Successor, Predecessor, and In-order Traversal. The 'Remove' option is selected. A search bar at the bottom left contains the number 5 and a 'GO' button. A text input field below the search bar says 'Enter an integer or comma-separated array of integers'. On the right side, a code editor shows the logic for removing a leaf node:

```
search for v
if v is a leaf
    delete leaf v
else if v has 1 child
    bypass v
else replace v with successor
```

At the bottom of the interface, there is a progress bar and navigation controls. The bottom right corner contains links for 'About', 'Team', and 'Terms of use'.

# BST: Delete/Remove Operation (2)

Ask VisuAlgo to perform various delete operations on the sample BST (this is **delete vertex with one child**)

In the screen shot below, we show **remove(23)** before relayout

Remove 23

Delete node 23 and connect its parent to its right child

```
search for v
if v is a leaf
    delete leaf v
else if v has 1 child
    bypass v
else replace v with successor
```

slow fast

About Team Terms of use

# BST: Delete/Remove Operation (3)

Ask VisuAlgo to perform various delete operations on the sample BST (delete **vertex with two children**)

In the screen shot below, we show **remove(6)** before relayout

The screenshot displays the VisuAlgo interface for a Binary Search Tree (BST). The tree structure is as follows:

- Root node: 15 (orange)
- Left child of 15: 7 (orange)
- Right child of 15: 23
- Left child of 7: 4
- Right child of 7: 15 (orange)
- Left child of 4: 5
- Right child of 23: 71
- Left child of 71: 50

The interface includes a menu on the left with options: Create, Search, Insert, Remove, Successor, Predecessor, and In-order Traversal. A search bar at the bottom left contains the number 6 and a GO button. A code editor on the right shows the logic for removing a node with two children:

```
Replace node 6 with its successor
search for v
if v is a leaf
    delete leaf v
else if v has 1 child
    bypass v
else replace v with successor
```

At the bottom of the interface, there are controls for slow/fast animation, a progress bar, and links for About, Team, and Terms of use.

# **ANALYSIS OF BST OPERATIONS**

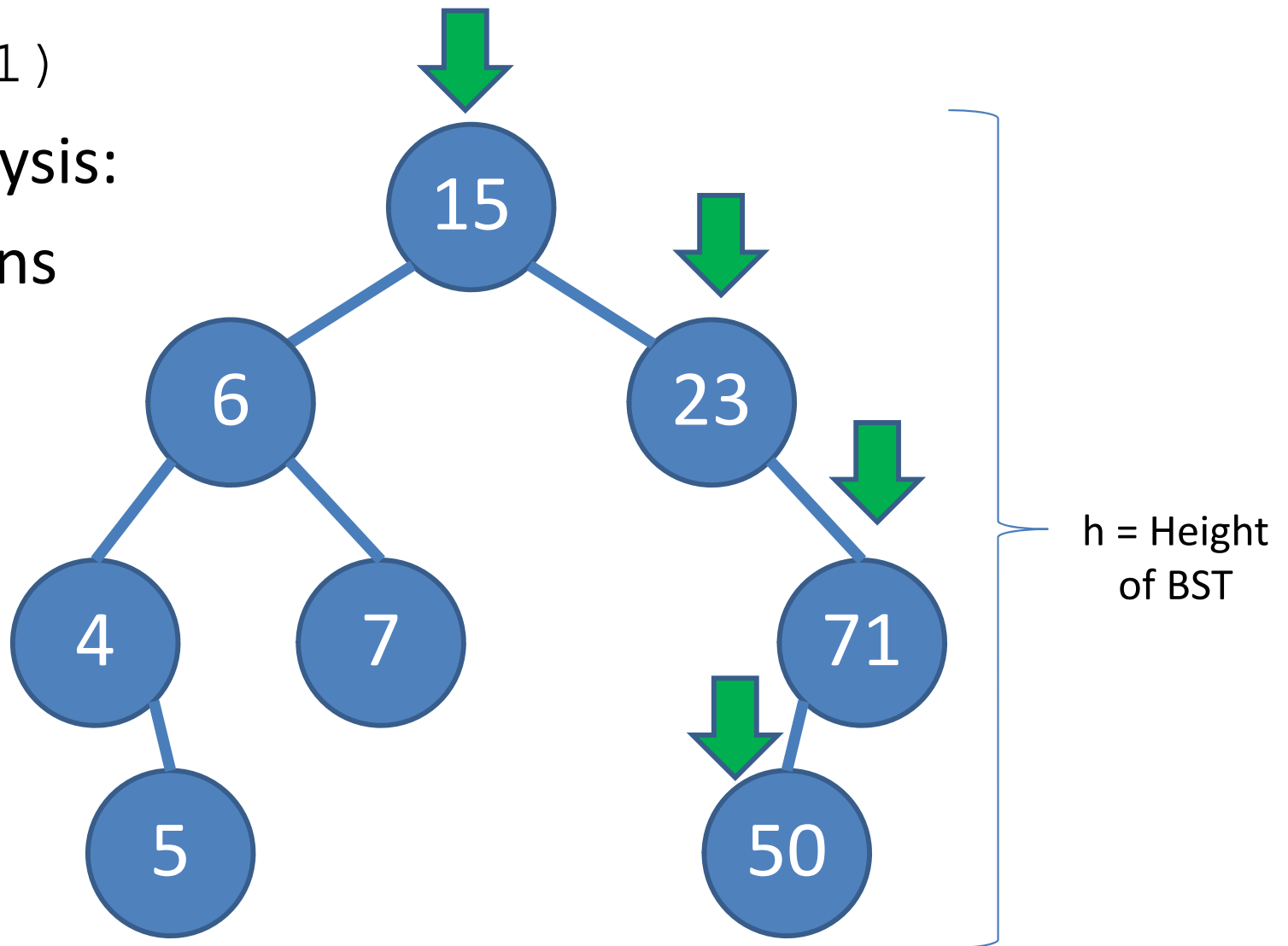
# BST: Search Analysis

`search(51)`

Quick analysis:

search runs

in  **$O(h)$**



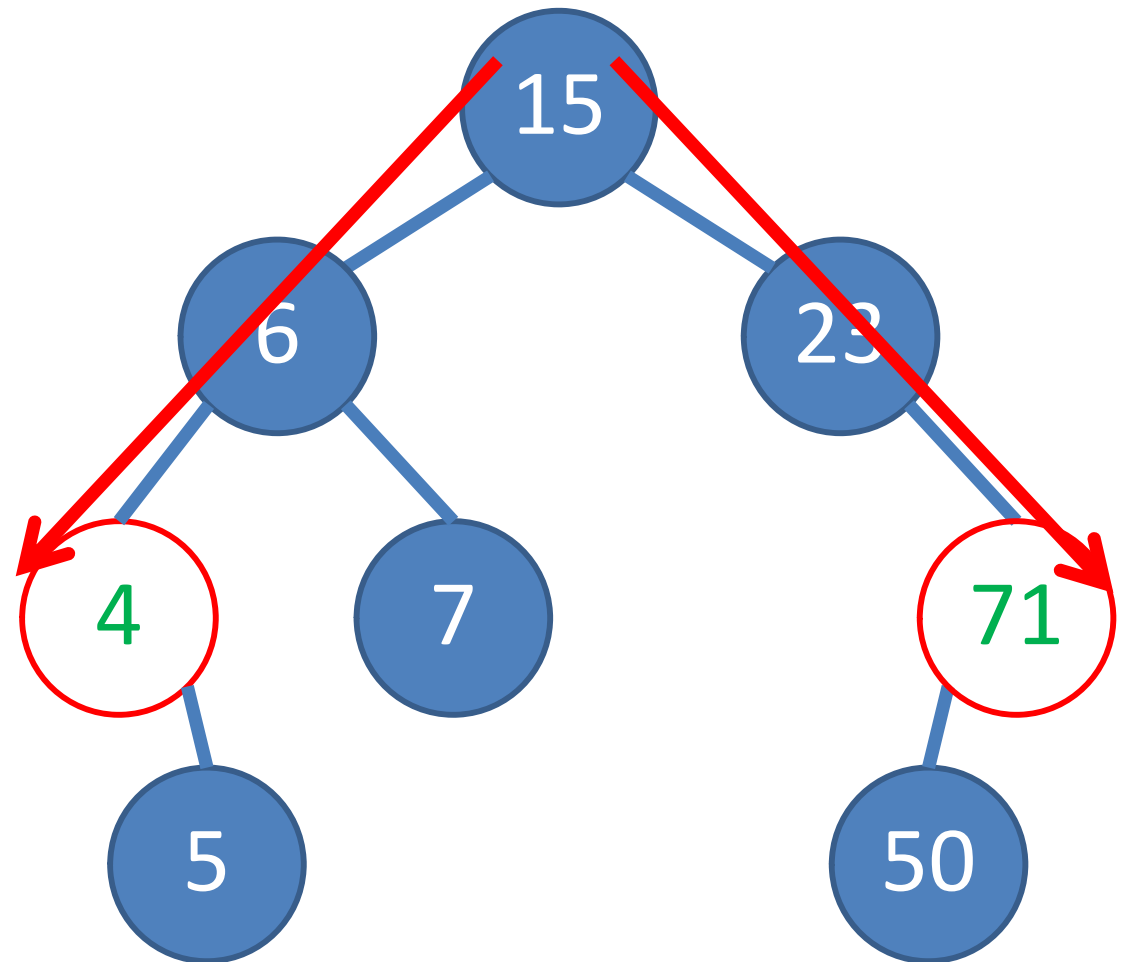
51 is not found 😞

# BST: Find Min/Max Analysis

Quick analysis:

`findMin/findMax`

also runs in  **$O(h)$**



# Inorder Traversal Analysis

Using a *new* analysis technique

Ask this question:

- How many times a vertex is *touched* during inorder traversal from the start until the end?

Answer:

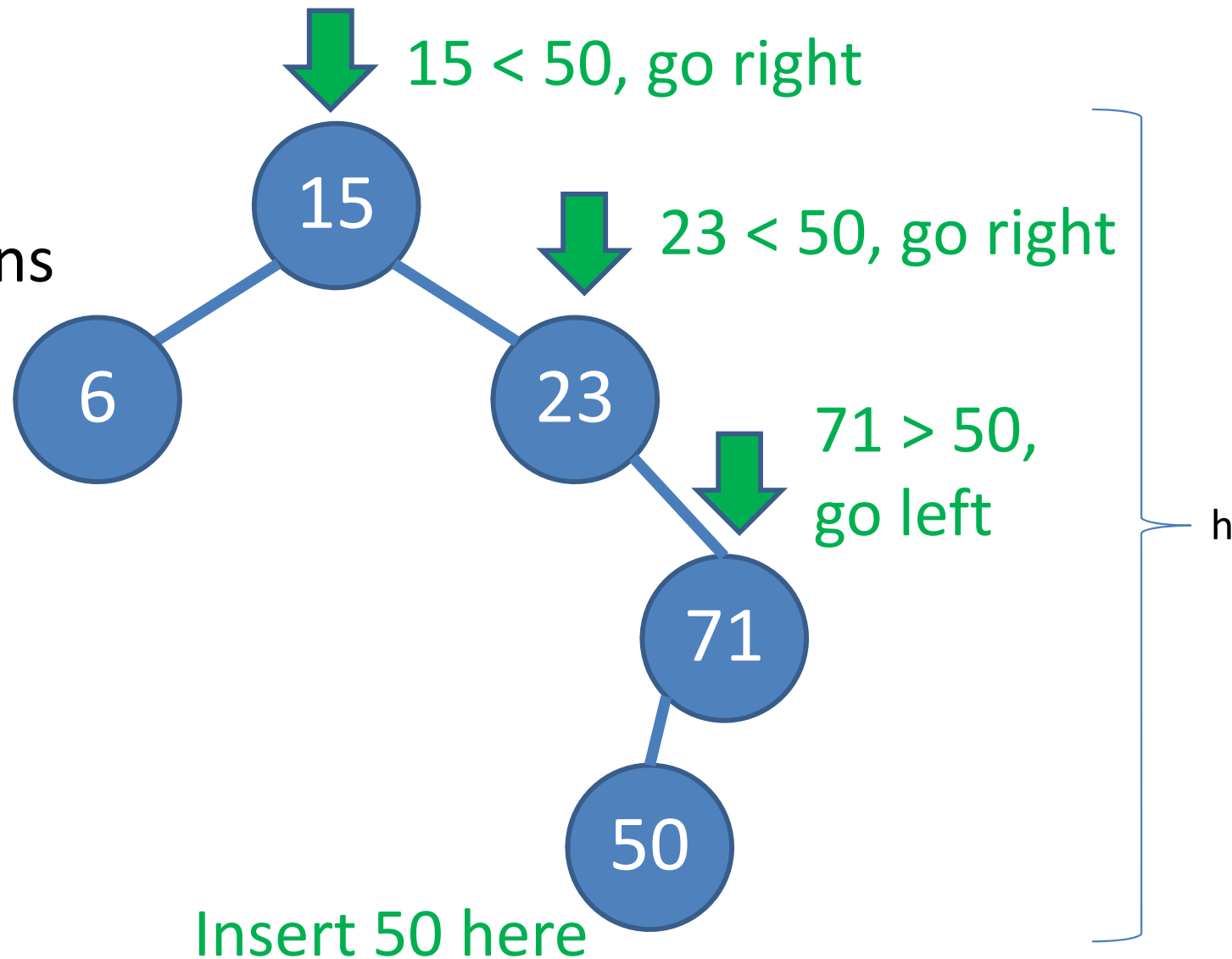
- Three times: from parent and from left + right children (even if one or both of them is/are empty/NULL)
- $O(3n) = O(n)$

# BST: Insertion Analysis

`insert(50)`

Quick analysis:

`insert` also runs  
in  **$O(h)$**



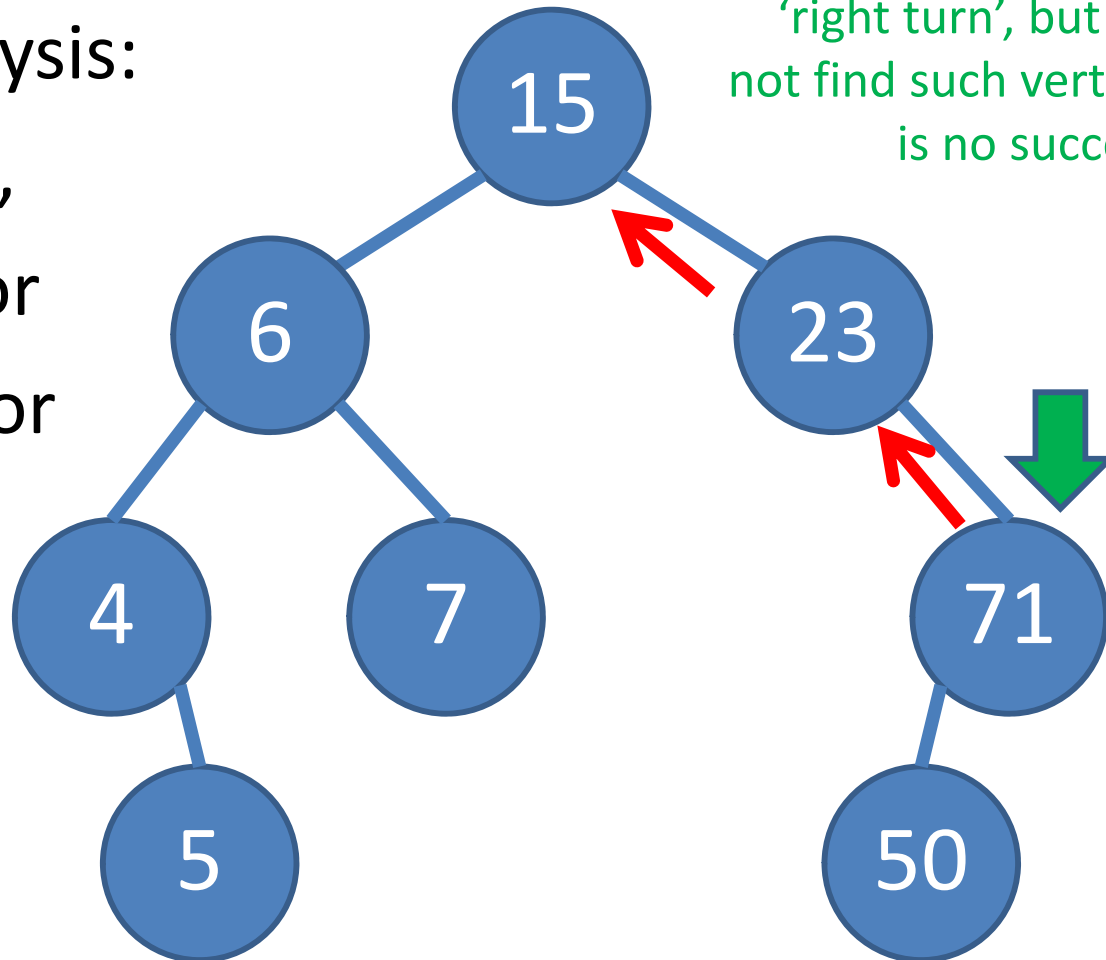


# BST: Successor/Predecessor Analysis

`successor(71)`

Quick analysis:

**$O(h)$**  again,  
similarly for  
predecessor



# Why successor of $x$ can be used for deletion of a BST vertex $x$ with 2 children?

Claim: Successor of  $x$  has at most 1 child!

- Easier to delete and will not violate BST property

Proof:

- Vertex  $x$  has two children
- Therefore, vertex  $x$  must have **a right child**
- Successor of  $x$  must then be the minimum of the right subtree
- A minimum element of a BST has no left child!!
- *So, successor of  $x$  has at most 1 child! ☺*

# BST: Delete Analysis

Delete a BST vertex  $v$ , find  $v$  in  $O(h)$ , then three cases:

- Vertex  $v$  has no children:
  - Just remove the corresponding BST vertex  $v \rightarrow O(1)$
- Vertex  $v$  has 1 child (either left or right):
  - Connect  $v.\text{left}$  (or  $v.\text{right}$ ) to  $v.\text{parent}$  and vice versa  $\rightarrow O(1)$
  - Then remove  $v \rightarrow O(1)$
- Vertex  $v$  has 2 children:
  - Find  $x = \text{successor}(v) \rightarrow O(h)$
  - Replace  $v.\text{key}$  with  $x.\text{key} \rightarrow O(1)$
  - Then delete  $x$  in  $v.\text{right} \rightarrow O(h)$

Running time:  $O(h)$

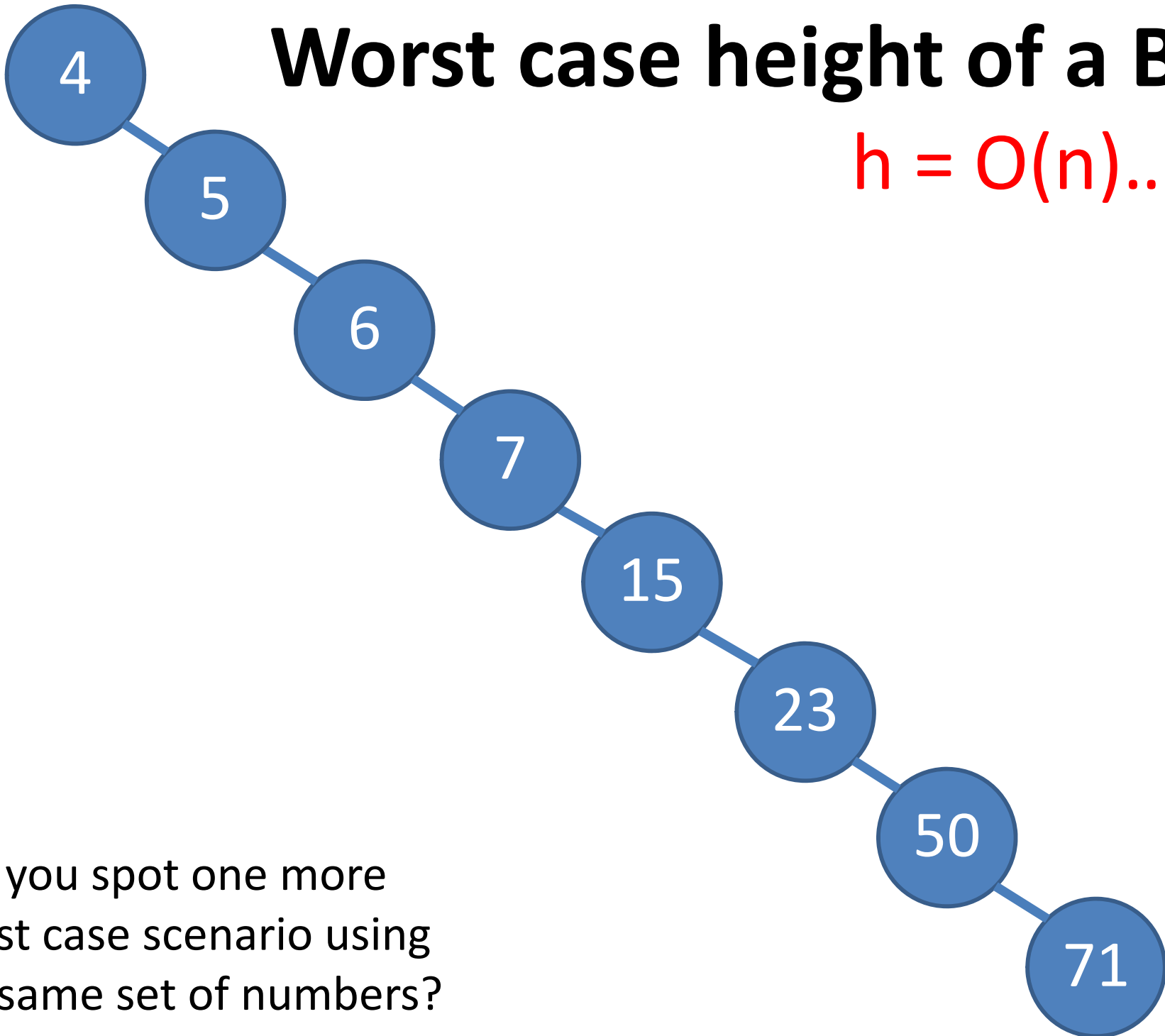
# Now, after we learn BST...

No	Operation	Unsorted Array	Sorted Array	BST
1	Search(age)	$O(n)$	$O(\log n)$	$O(h)$
2	Insert(age)	$O(1)$	$O(n)$	$O(h)$
3	FindOldest()	$O(n)$	$O(1)$	$O(h)$
4	ListSortedAges()	$O(n \log n)$	$O(n)$	$O(n)$
5	NextOlder(age)	$O(n)$	$O(\log n)$	$O(h)$
6	Remove(age)	$O(n)$	$O(n)$	$O(h)$
7	GetMedian()	$O(n \log n)$	$O(1)$	$O(h)$

It is all now depends on 'h'... → next lecture 😊

# Worst case height of a BST

$$h = O(n) \dots \text{😞}$$



Can you spot one more worst case scenario using the same set of numbers?

# Java Implementation

See BSTVertex.java, BST.java, and BSTDemo.java

Concepts covered:

1. Java Object Oriented Programming (OOP)  
implementation of BST data structure
2. Java Error Handling: Throw & Catch Exception

# The Baby Names Problem (PS1)

Given a list of male and female baby names suggestions (*from your parents, in-laws, friends, yourself, Internet, **etc***), your task is to answer some queries (see the next slide)

*This problem is always* encountered by every parents with new baby

(Including the search for baby Joshua name, born on 16 July 2014)



# PS1 Queries

(Note: Unlike this lecture, the keys are strings)

**Easy:** How many names start with a certain letter?

**Medium:** How many names start with a certain prefix?

*Definition: A prefix of a string  $T = T_0T_1...T_{n-1}$  with length  $n$  is string  $P = T_0T_1...T_m$  where  $m < n$ .*

**Hard:** Can you do it without Java API library code?

**CS2010R:** How many names have a certain substring?

*Definition: A substring of a string  $T = T_0T_1...T_{n-1}$  with length  $n$  is string  $S = T_iT_{i+1}...T_{j-1}T_j$  where  $0 \leq i \leq j < n$ .*

**You need efficient DS(es) to answer those queries**



# End of Lecture Quiz 😊

VisuAlgo online quiz demo

Go to:

<http://www.comp.nus.edu.sg/~stevenha/visualization/test.html>

Use your CS2010 account to try the sample  
7 BST questions listed there

I can track your progress

Go ahead and do it now