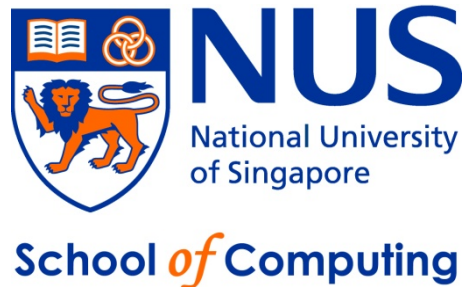


CS2040 – Data Structures and Algorithms

Lecture 10 – Census Problem

chongket@comp.nus.edu.sg



Outline

Motivation: Census Problem

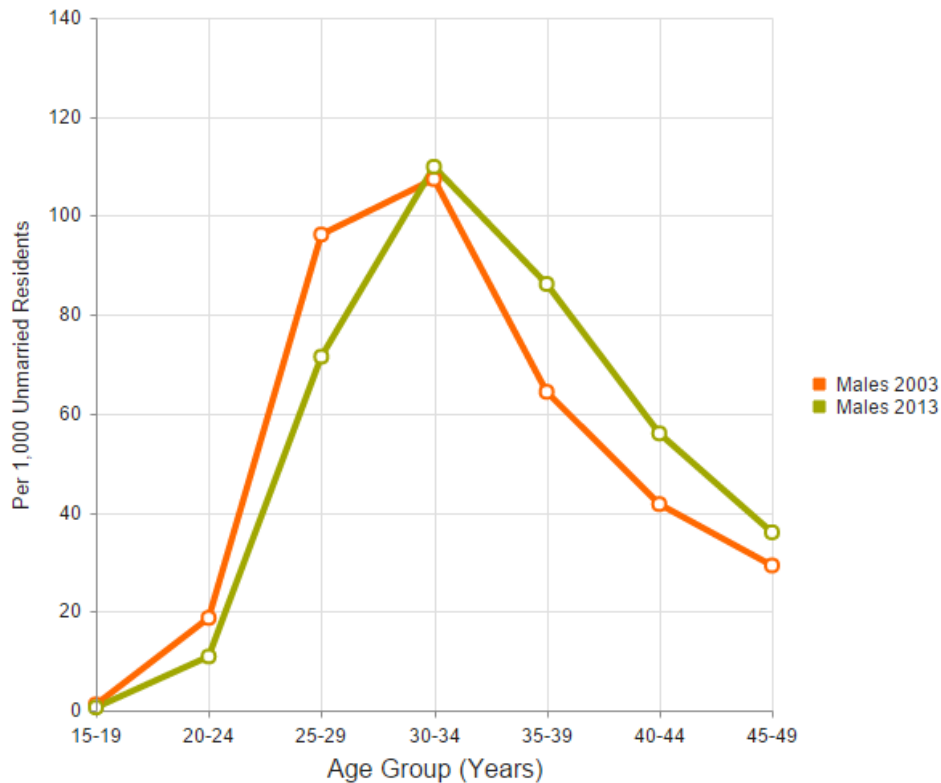
- Abstract Data Type (ADT) Table
- Solving Census Problem with CS2040 1st Half Knowledge
- The “performance issue”

Binary Search Tree (BST)

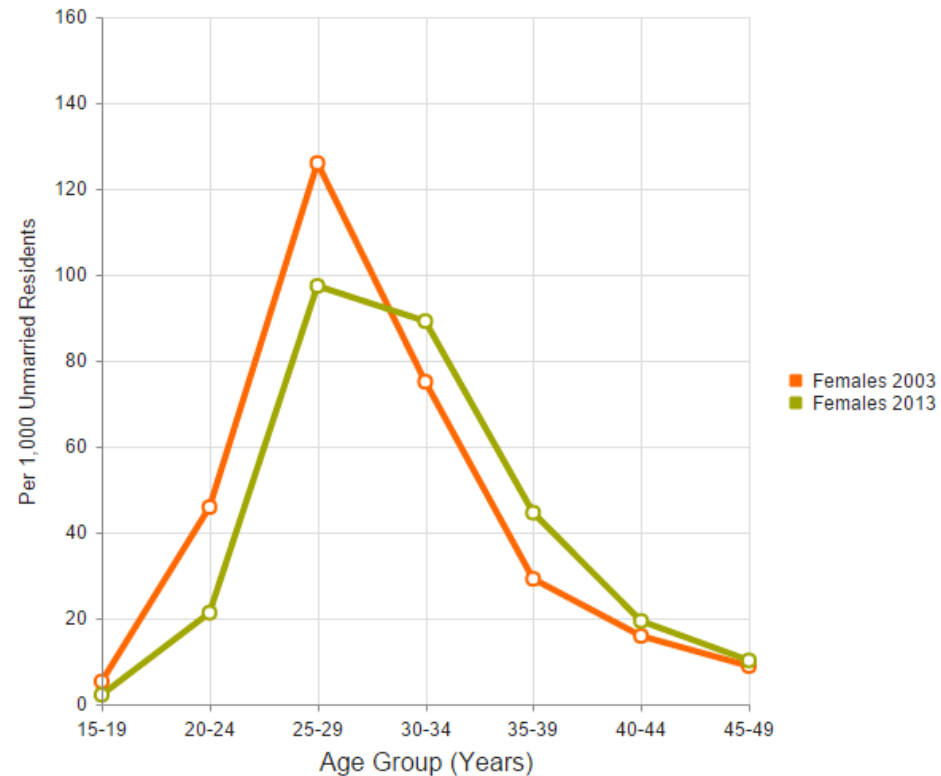
- Heavy usage of [VisuAlgo Binary Search Tree Visualization](#)
- Simple analysis of BST operations
- Java Implementation

Census is Important!

Age-Specific Marriage Rates (Males)



Age-Specific Marriage Rates (Females)



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 (2016 data)

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

1. [24 ... ∞)

2. [23 ... 24)

3. [22 ... 23)

4. [21 ... 22)

5. [20 ... 21)

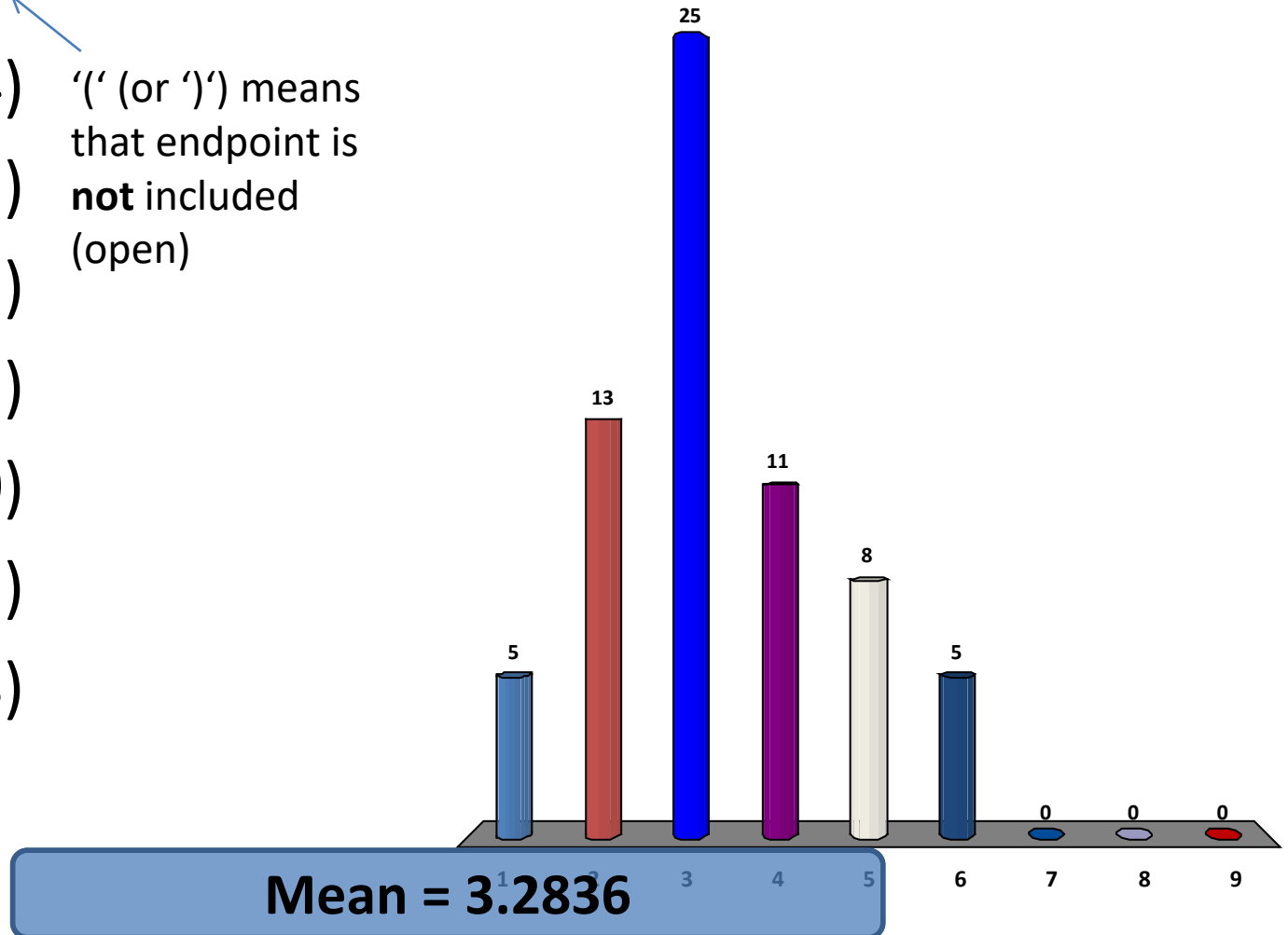
6. [19 ... 20)

7. [18 ... 19)

8. [17 ... 18)

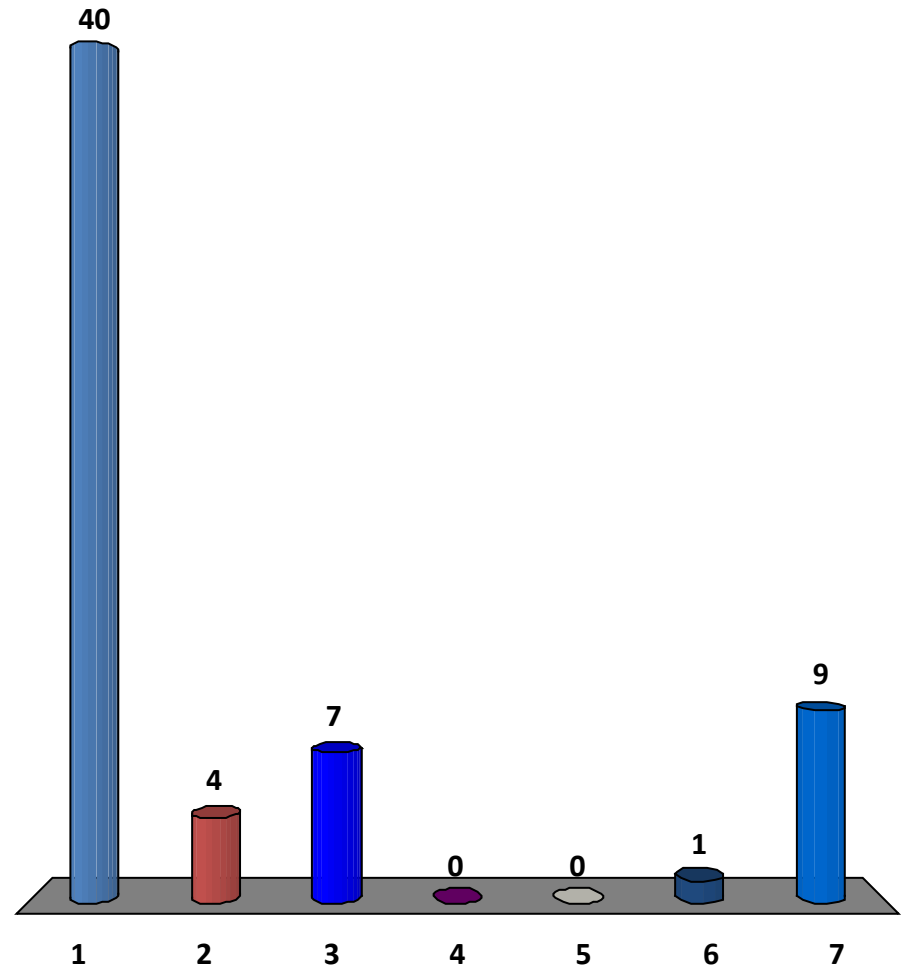
9. [0 ... 17)

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



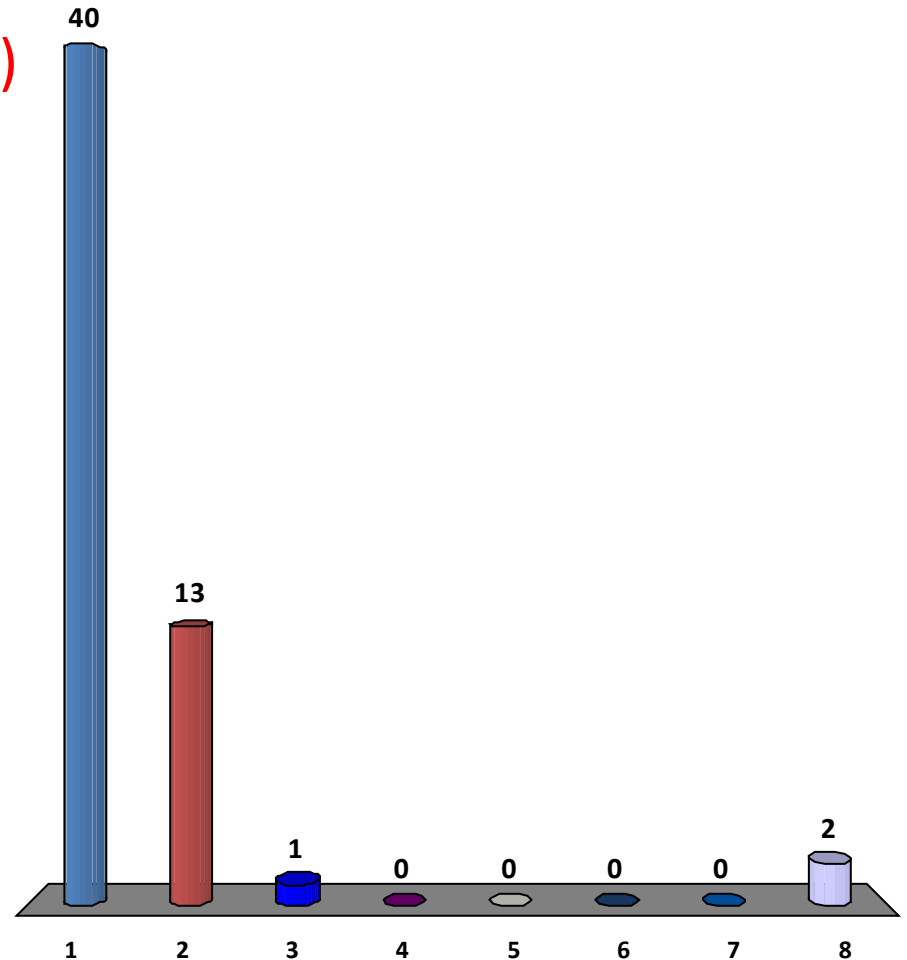
Your Major (2016 data)

1. Computer Science (CS)
2. Business Analytics (BZA)
3. Computer Engineering (CEG/CEC)
4. Comp. Biology (CB)
5. Information System (IS)
6. Science Maths (SCI)
7. None of the above :O



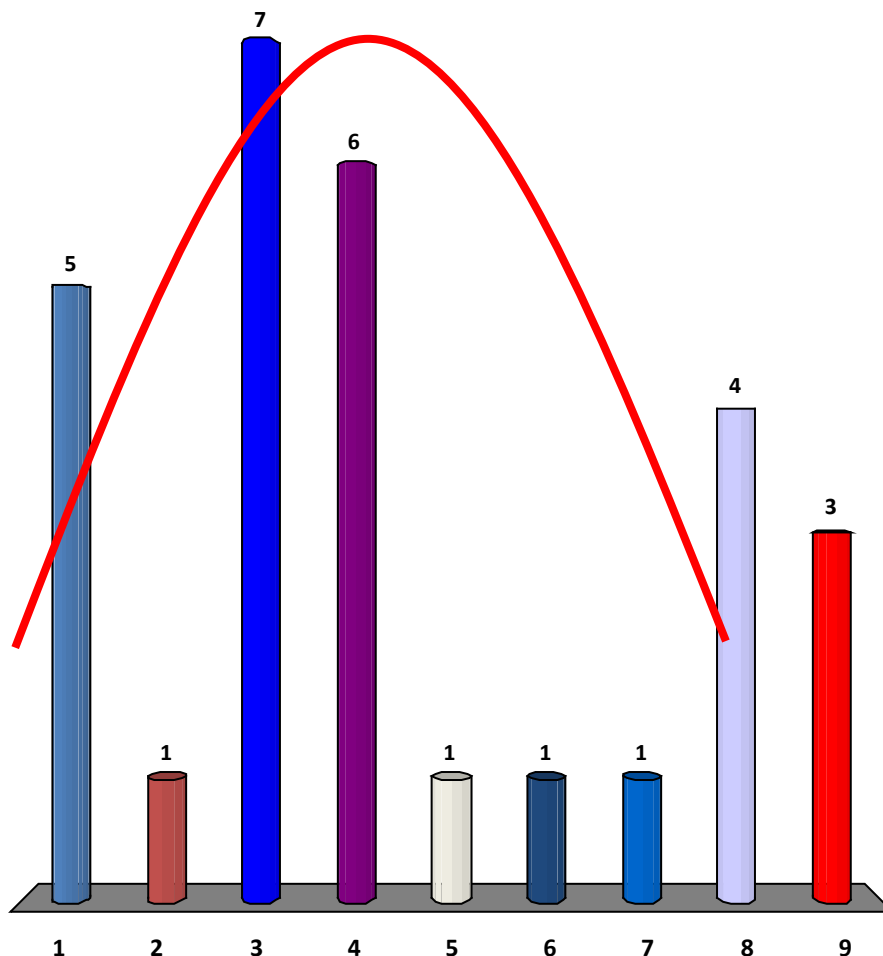
Your Nationality (2016 data)

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) <- very old 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 as the key: **Age**


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

Some required operations:

1. Search whether there is a student with a certain age?
2. Insert a new student (insert using 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 (remove using his/her age)
7. Determine the median age of students
8. How many students are younger than a certain age?

Using 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()	$O(N \log N)/O(N)$
8	 NumYounger(age)	$O(N \log N)/O(N)$

QuickSelect (Recursion Lecture)
 GetMedian() = Select($N/2$)
 Expected $O(N)$



Using Sorted Array

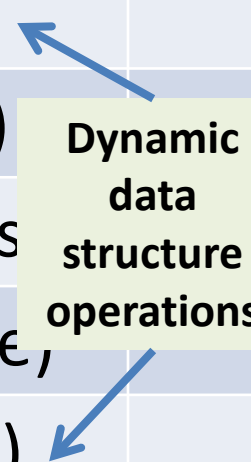
Index	0	1	2	3	4	5	6	7	
A	4	5	6	7	15	23	50	71	

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()	$O(1)$
8	NumYounger(age)	$O(\log N)$

With Just 1st Half 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	$(N \log N)$	$O(N)$
5	NextOlder(age,)	$O(N)$	$O(\log N)$
6	Remove(age)	$O(N)$	$O(N)$
7	GetMedian()	$O(N \log N)/O(N)$	$O(1)$
8	NumYounger(age)	$O(N \log N)/O(N)$	$O(\log N)$

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. $N = 1\,000\,000\dots$

A Versatile, Non-Linear Data Structure

BINARY SEARCH TREE (BST)

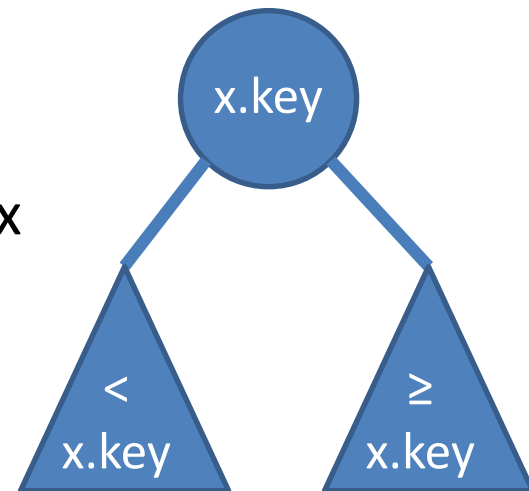
Binary Search Tree (BST) Vertex

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:

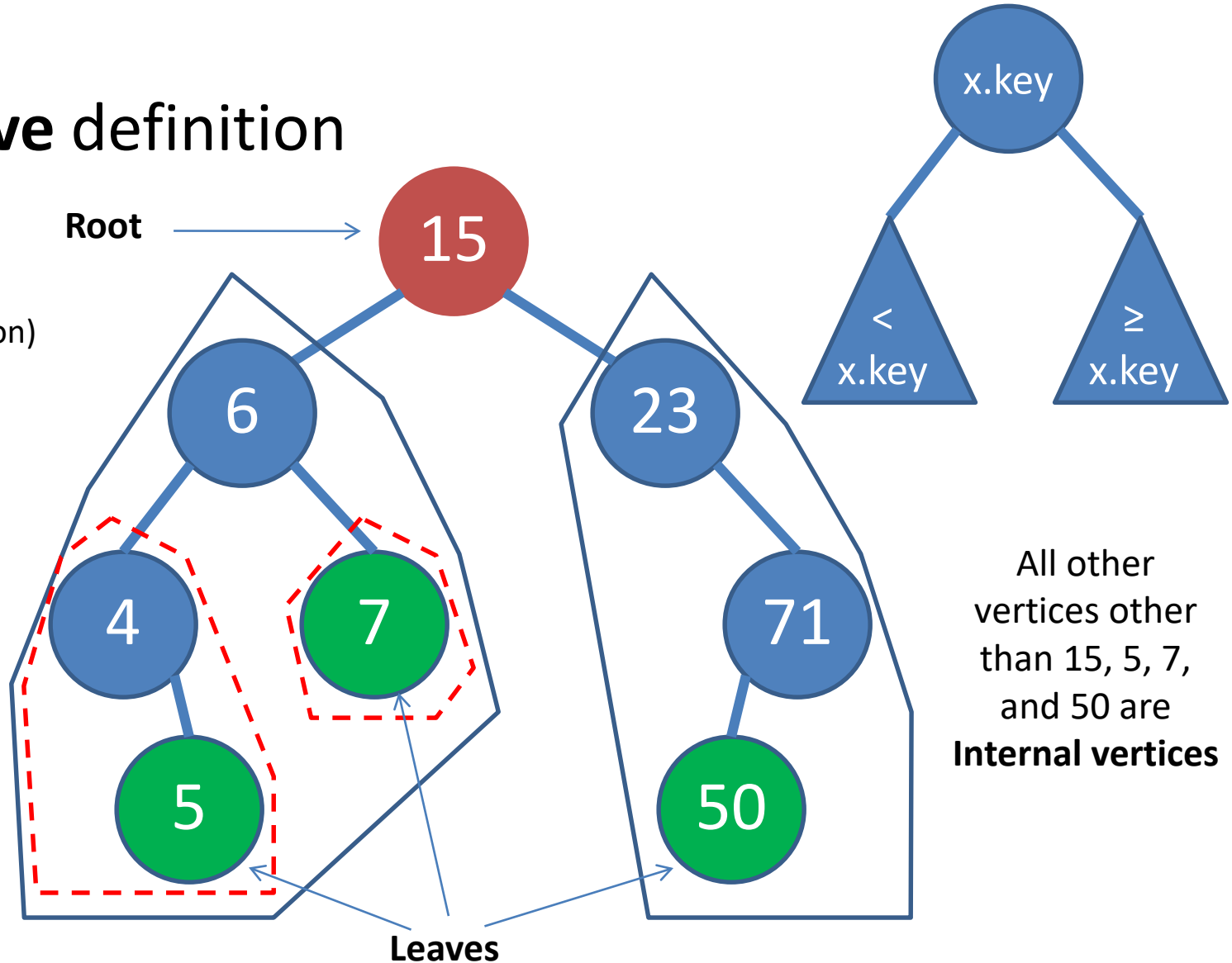
- For every vertex x and y
 $y.\text{key} < x.\text{key}$ if y is in left subtree of x
 $y.\text{key} \geq x.\text{key}$ if y is in right subtree of x
- For simplicity, we assume that the keys are unique so that we can change \geq to $>$

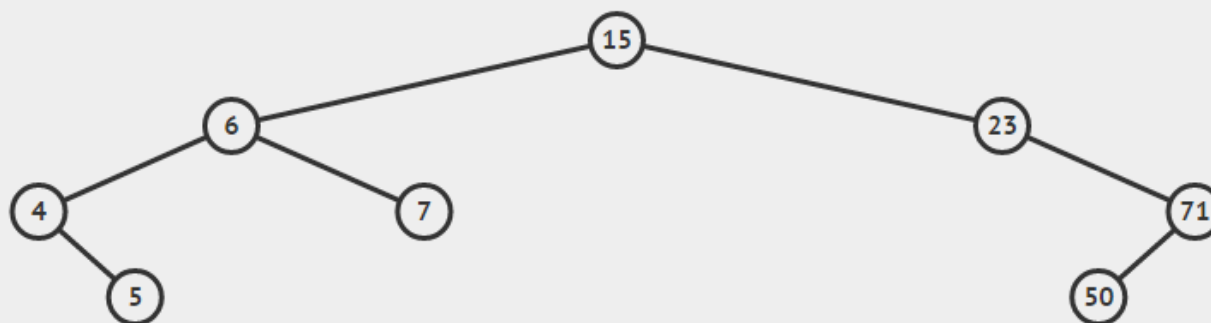


BST: An Example, Keys = Ages

Recursive definition

⚡ Root is not an internal node
(VisuAlgo definition)





View the BST visualisation here! Root vertex does not have a parent. There can only be one root vertex in a BST. Leaf vertex does not have any child. There can be more than one leaf vertex in a BST. All other vertices that are not root nor leaf are called the internal vertices. All vertices have at least 4 attributes: parent, left, right, key/value/data although not all attributes will be used for all vertices.

As we do not allow duplicate integer in this visualization (the full implementation must consider duplicate integers too), notice that for every vertex X, all vertices on the left subtree of X are smaller than X and all vertices on the right subtree of X are greater than X. This is called the 'BST property'.

All available operations on the BST/AVL Tree will be visualized/animated here.

< Prev

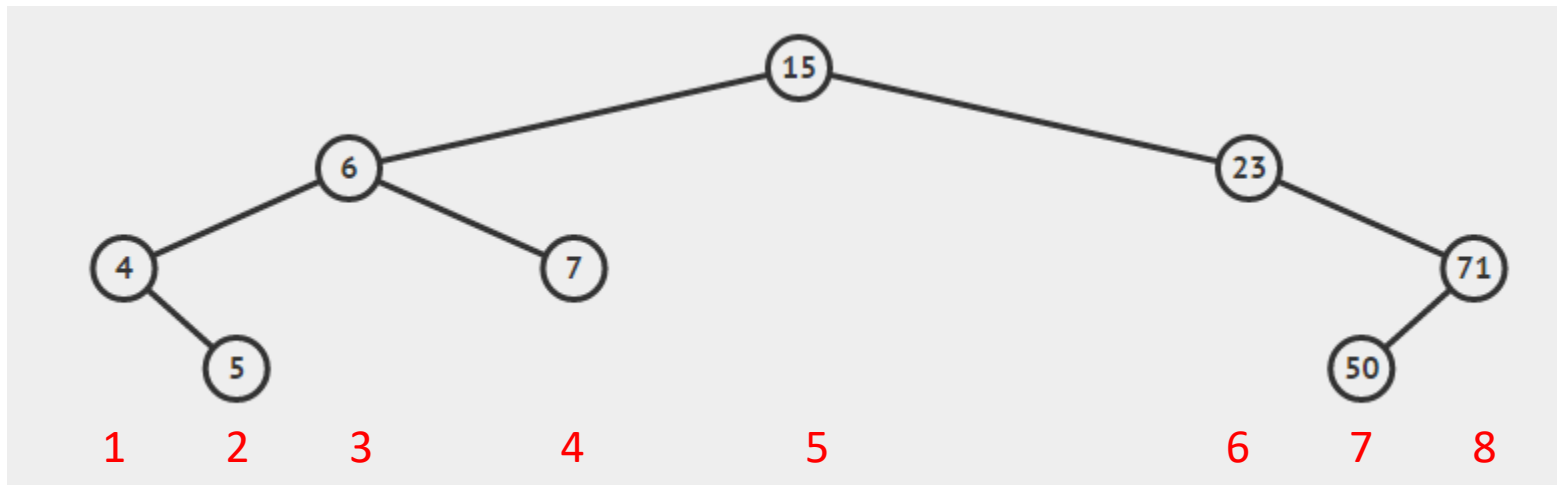
Next >

Create
Search
Insert
Remove
Successor
Predecessor
Inorder Traversal

BST: NEW Select/Rank Operations

These 2 operations are not yet in VisuAlgo BST visualization; for now, here are the concepts:

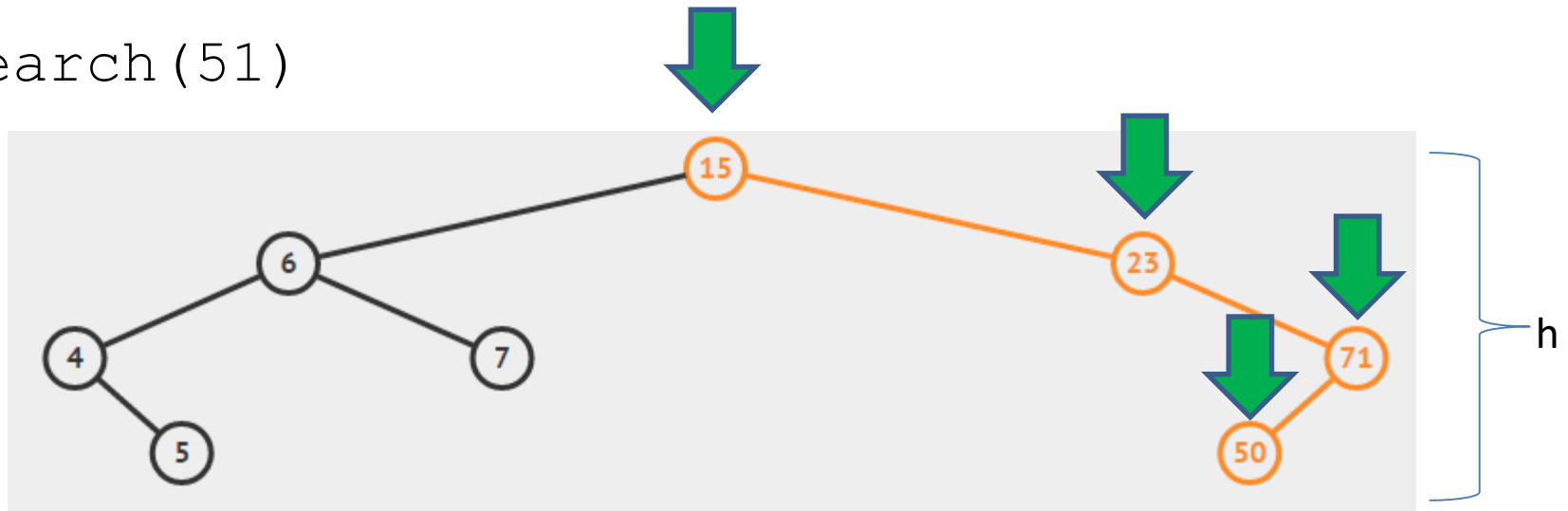
- $\text{Select}(k)$ – Return the value v of k -th smallest* element
 - Examples: $\text{Select}(1) = 4$, $\text{Select}(3) = 6$, $\text{Select}(8) = 71$, etc (1-based index)
- $\text{Rank}(v)$ – Return the rank* k of element with value v
 - Examples: $\text{Rank}(4) = 1$, $\text{Rank}(6) = 3$, $\text{Rank}(71) = 8$, etc
- Details will be discussed in topic for AVL trees



ANALYSIS OF BST OPERATIONS

BST: Search Analysis

search(51)



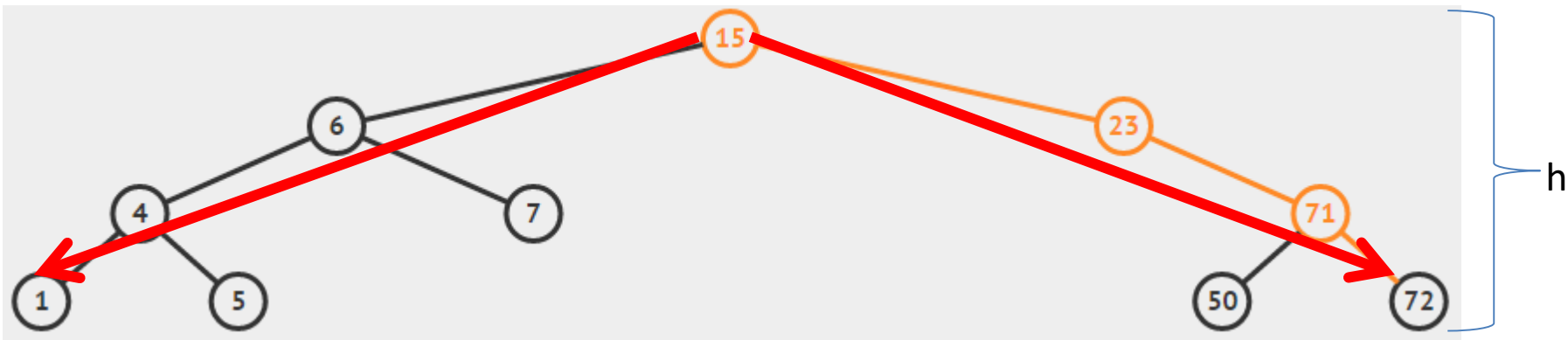
Quick analysis:

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

51 is not
found 😞

BST: FindMin/FindMax Analysis

`findMin()` / `findMax()`

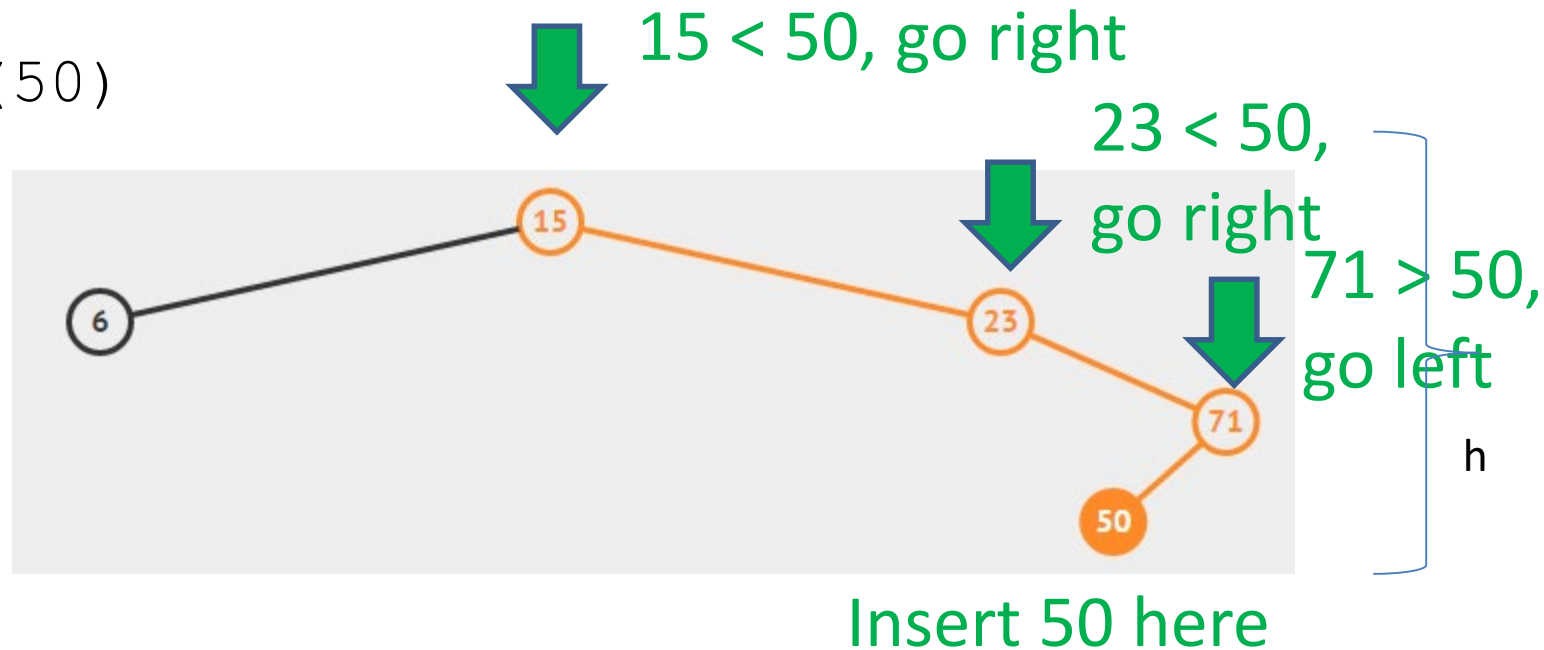


Quick analysis:

`findMin()` / `findMax` also runs in **$O(h)$**

BST: Insertion Analysis

`insert(50)`



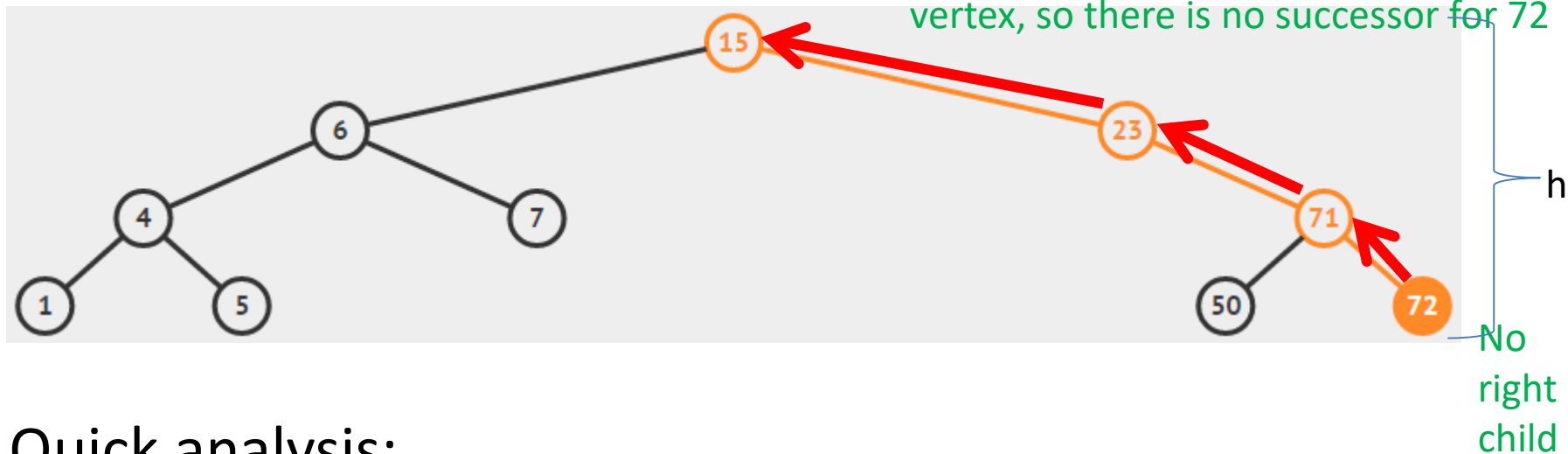
Quick analysis:

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

BST: Successor/Predecessor Analysis

Assumption, we already done an $O(h)$ search(72) before

`successor(72)`



Quick analysis:

$O(h)$ again, similarly for predecessor

BST: Inorder Traversal Analysis

Using a *new* analysis technique

Ask this question:

- How many times is a vertex *visited* 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)
- $O(3 * N) = O(N)$

Why is successor of x 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: Deletion Analysis

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

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

Running time: $O(\mathbf{h})$

BST: Select/Rank Analysis

We have not explored the operations in detail yet

This will be discussed in more details in the next lecture

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()	$O(N \log N)/O(N)$	$O(1)$?
8	NumYounger(age)	$O(N \log N)/O(N)$	$O(\log N)$?

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

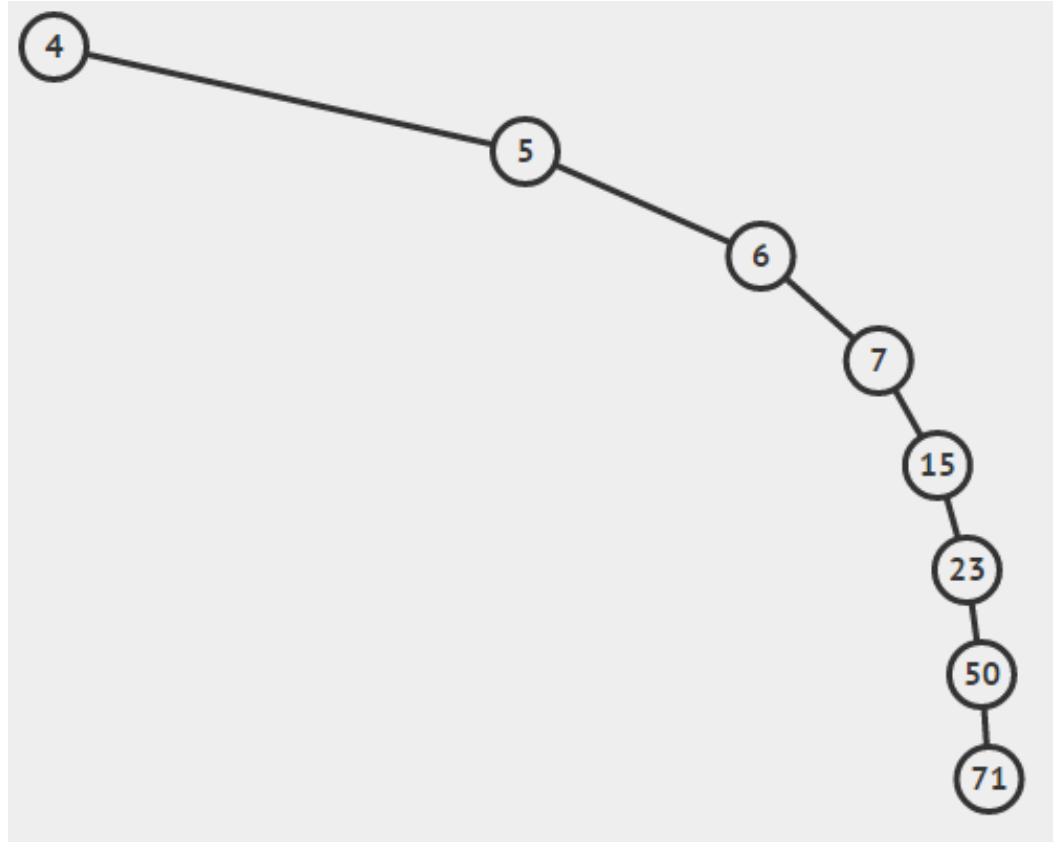
Worst case height of a BST

$$h = O(N) \dots \text{😞}$$

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

Can we do better?

YES, $h = O(\log N) \rightarrow$ next topic 😊



Testing/Training BST knowledge on Visualgo 😊

- Go to <https://visualgo.net/training>
- Select Binary Search Tree and unselect the rest
- Set the question difficulty (go from easy to hard)
- Set the number of questions (try 5 to 10 questions)
- Set a suitable time limit (20 to 60 mins)