Binary Search Trees

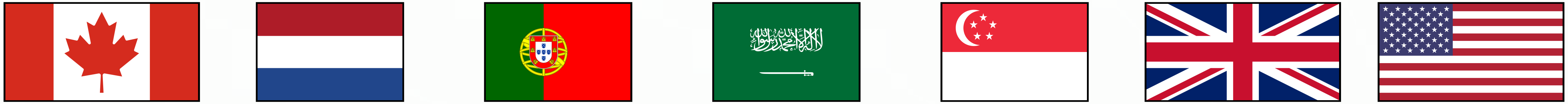
Benoît Corsini

Table of Contents

Table of Contents

Where are you from?

Where are you from?



Where are you from?

Where are you from?

Q: How long to go through the list in order?

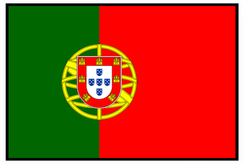
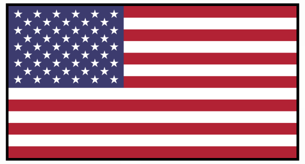
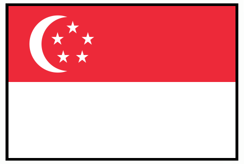
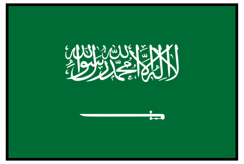
Where are you from?

Q: How long to go through the list in order?

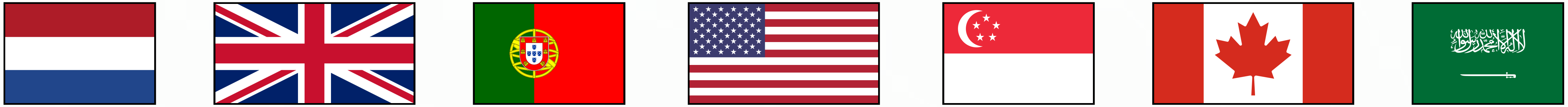
→ On average 4 queries to find the answer.

Where are you from?

Where are you from?

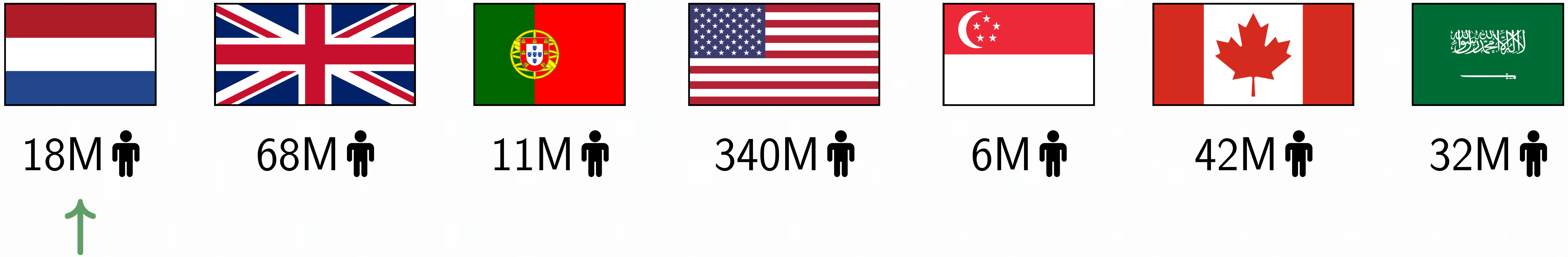
      

Where are you from?

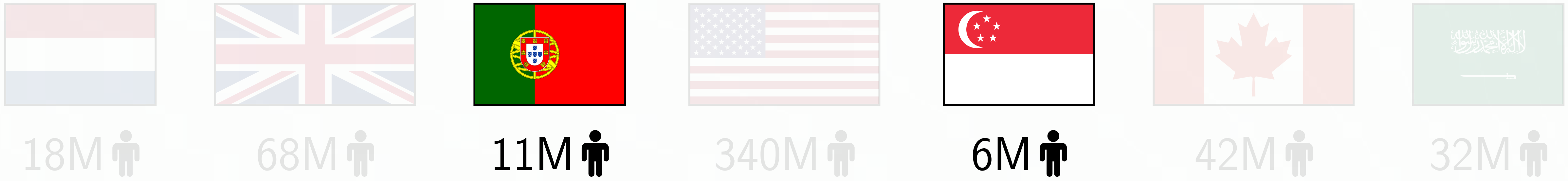


18M 68M 11M 340M 6M 42M 32M

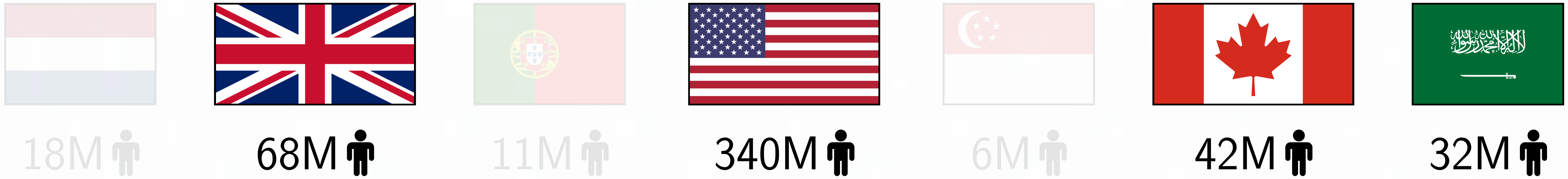
Where are you from?



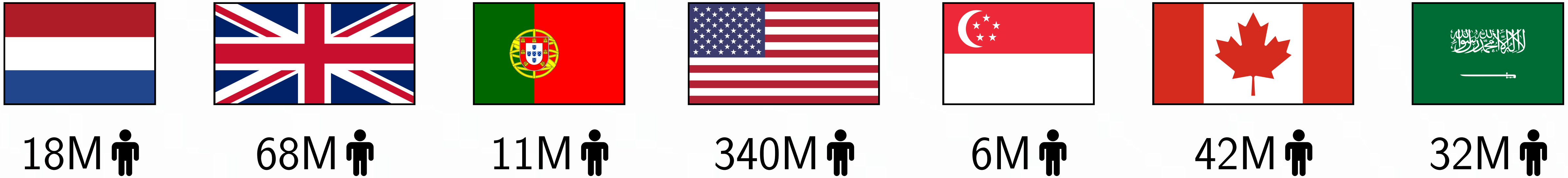
Where are you from?



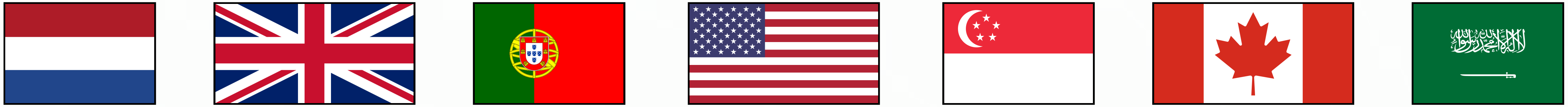
Where are you from?



Where are you from?



Q: How long to go through the list in order, if you can use the population?

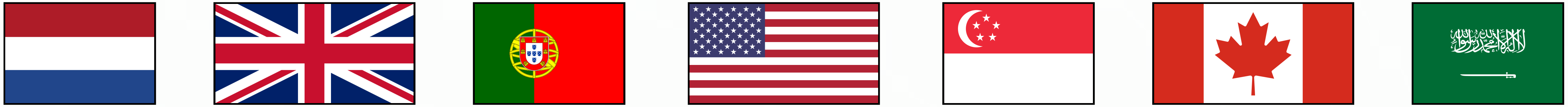


18M | 68M | 11M | 340M | 6M | 42M | 32M

Where are you from?

Q: How long to go through the list in order, if you can use the population?

→ The answer is the rest of this talk!

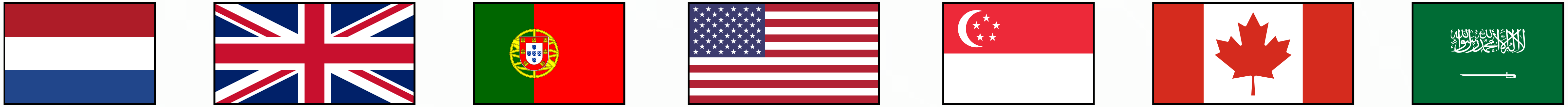


18M 68M 11M 340M 6M 42M 32M

Where are you from?

Q: How long to go through the list in order, if you can use the population?

→ The answer is the rest of this talk!

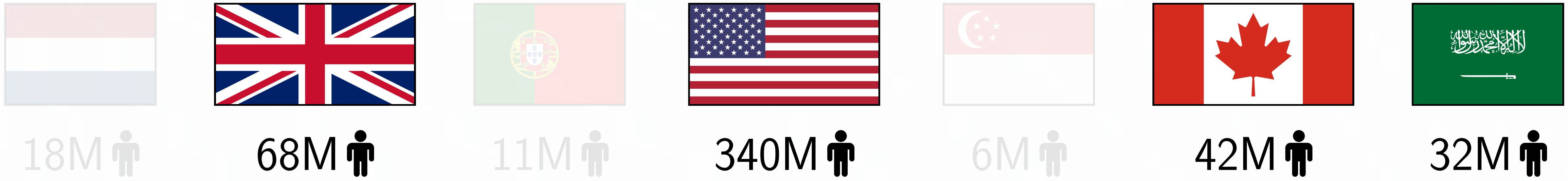


18M | 68M | 11M | 340M | 6M | 42M | 32M

Where are you from?

Q: How long to go through the list in order, if you can use the population?

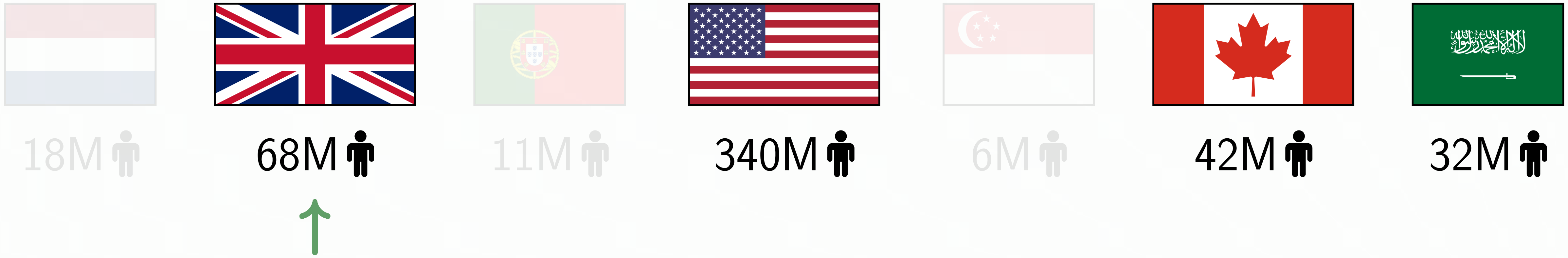
→ The answer is the rest of this talk!



Where are you from?

Q: How long to go through the list in order, if you can use the population?

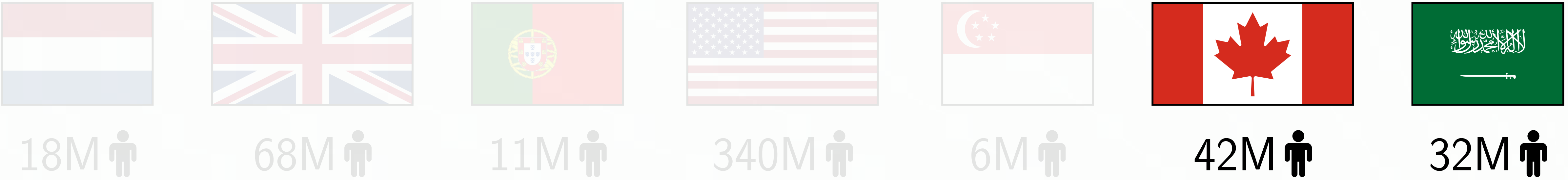
→ The answer is the rest of this talk!



Where are you from?

Q: How long to go through the list in order, if you can use the population?

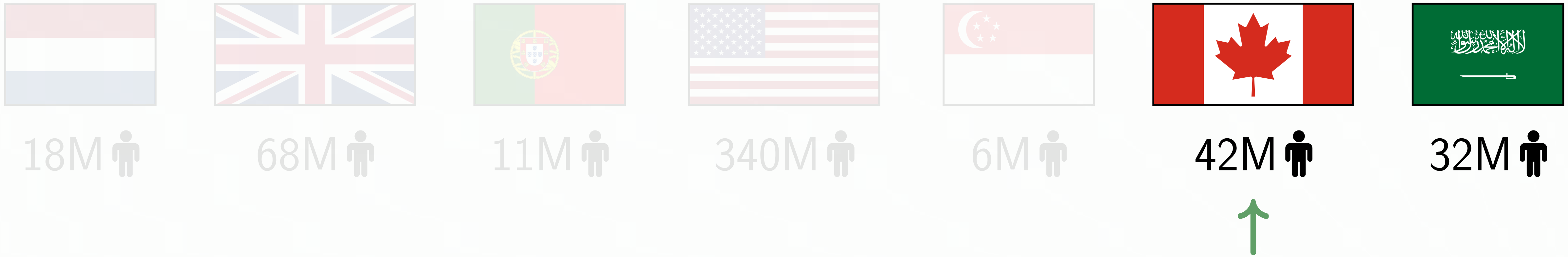
→ The answer is the rest of this talk!



Where are you from?

Q: How long to go through the list in order, if you can use the population?

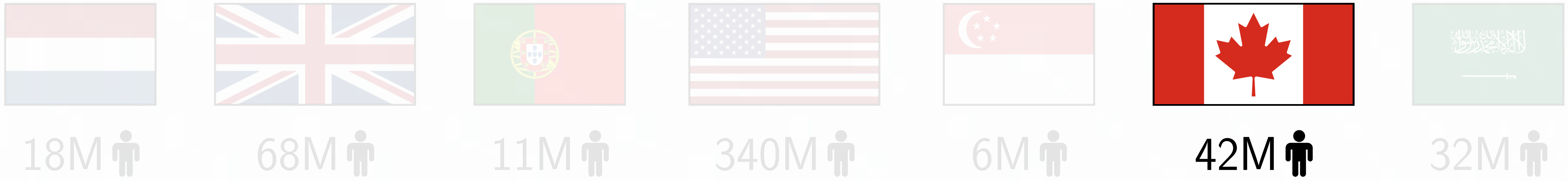
→ The answer is the rest of this talk!



Where are you from?

Q: How long to go through the list in order, if you can use the population?

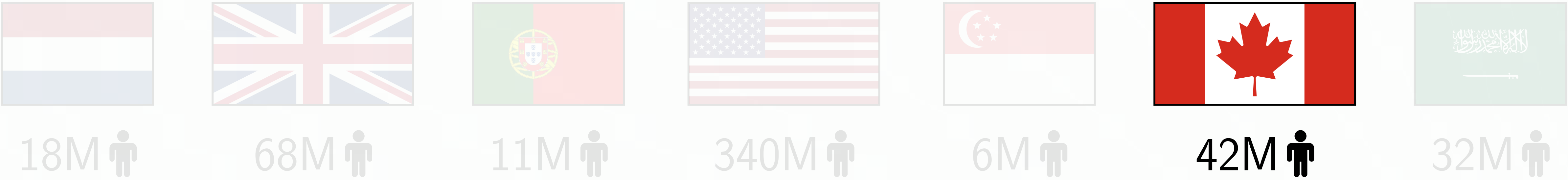
→ The answer is the rest of this talk!



Where are you from?

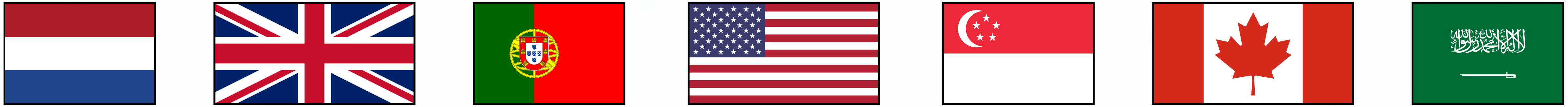
Q: How long to go through the list in order, if you can use the population?

→ The answer is the rest of this talk!



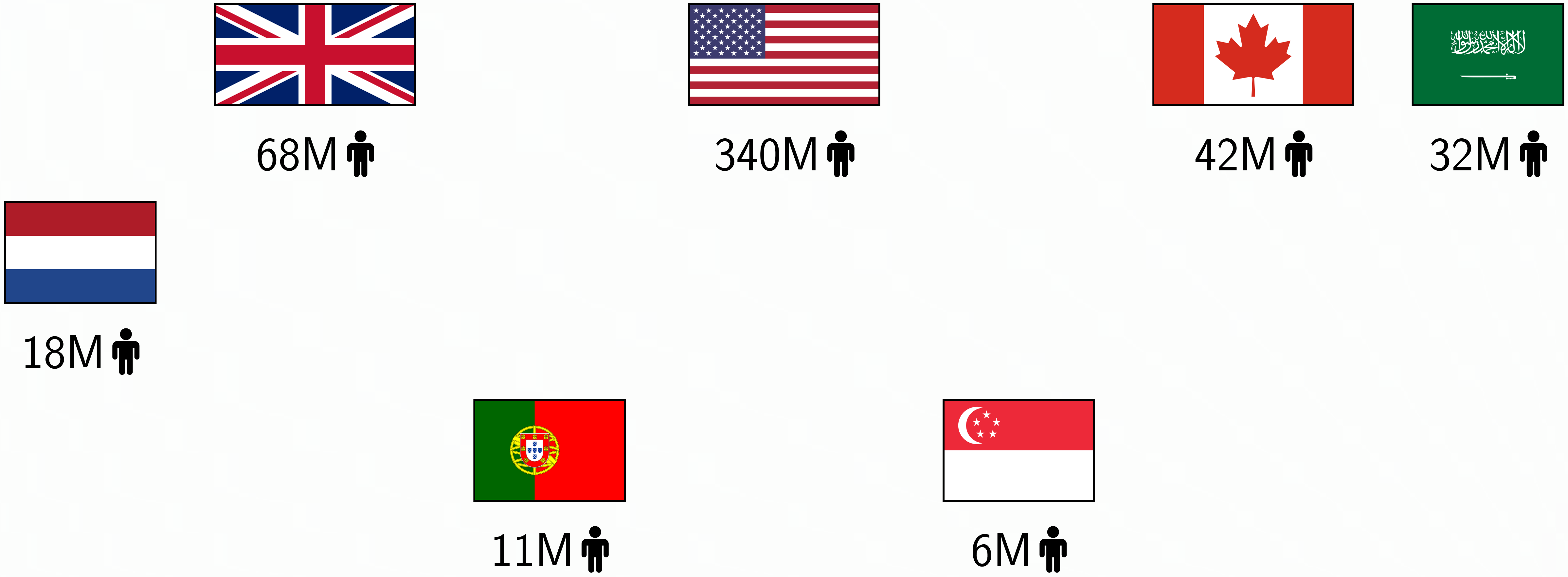
→ In this case, it took 3 queries using the population, against 6 not using it.

Where are you from?

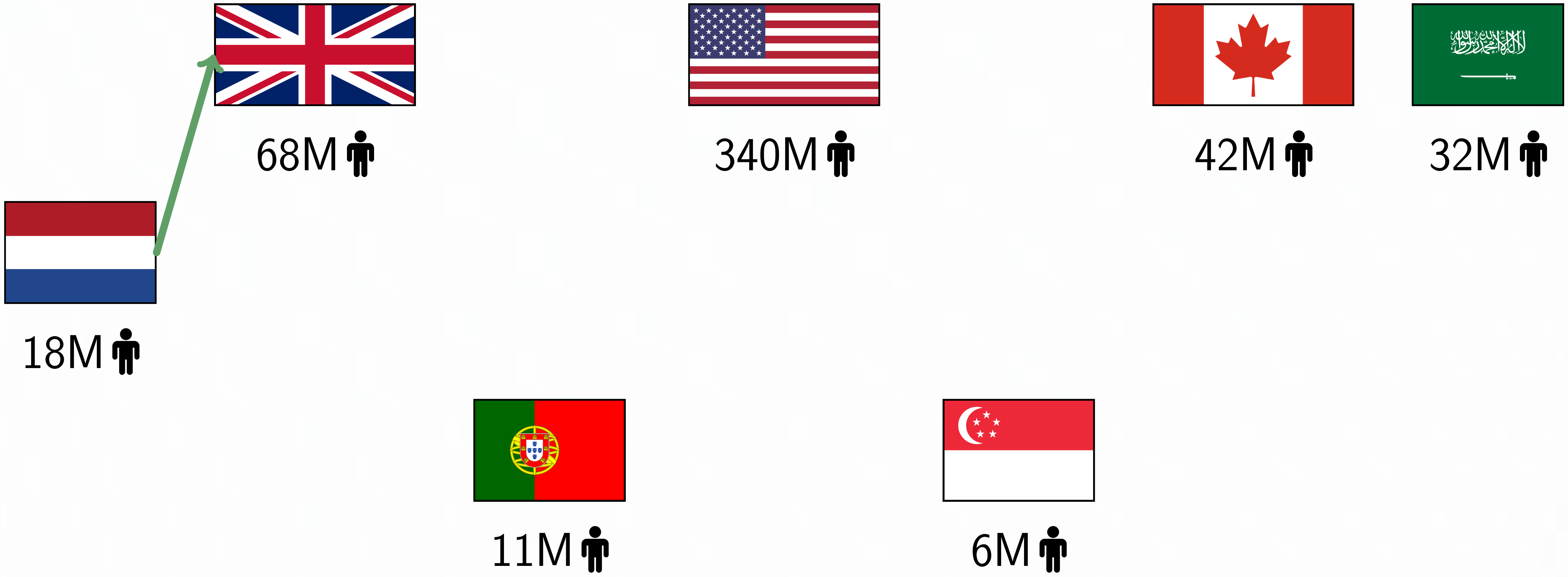


18M 68M 11M 340M 6M 42M 32M

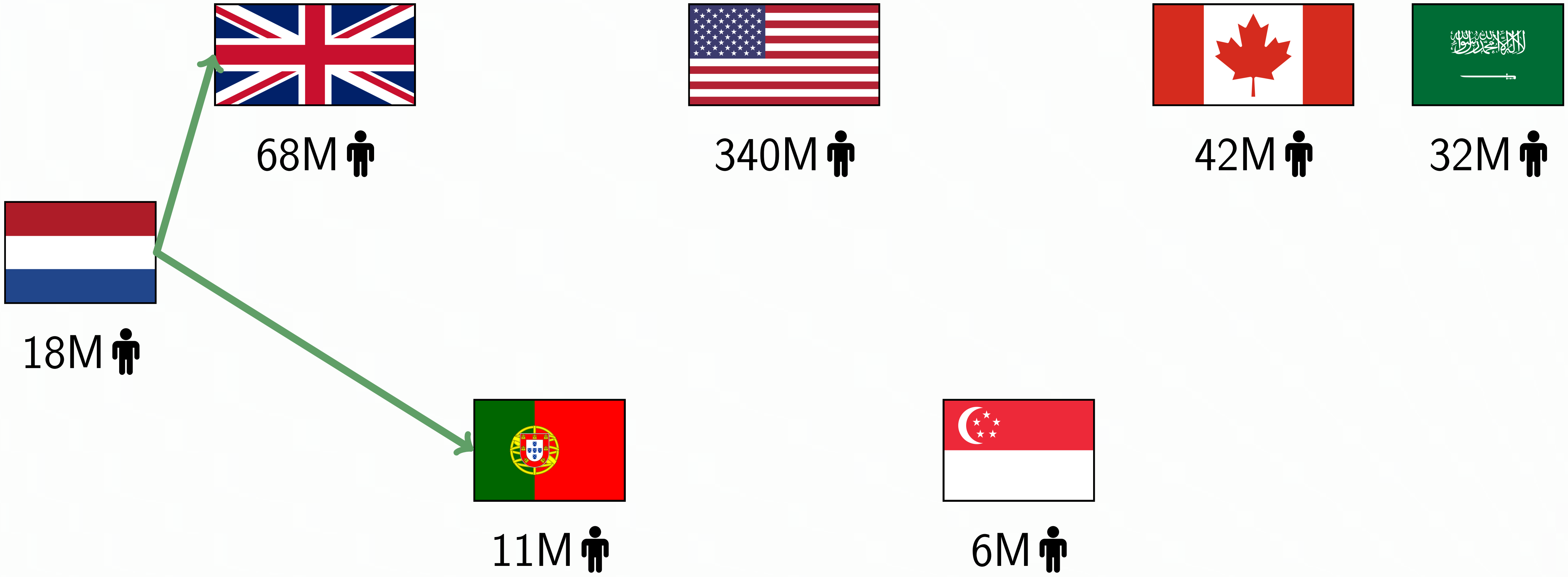
Where are you from?



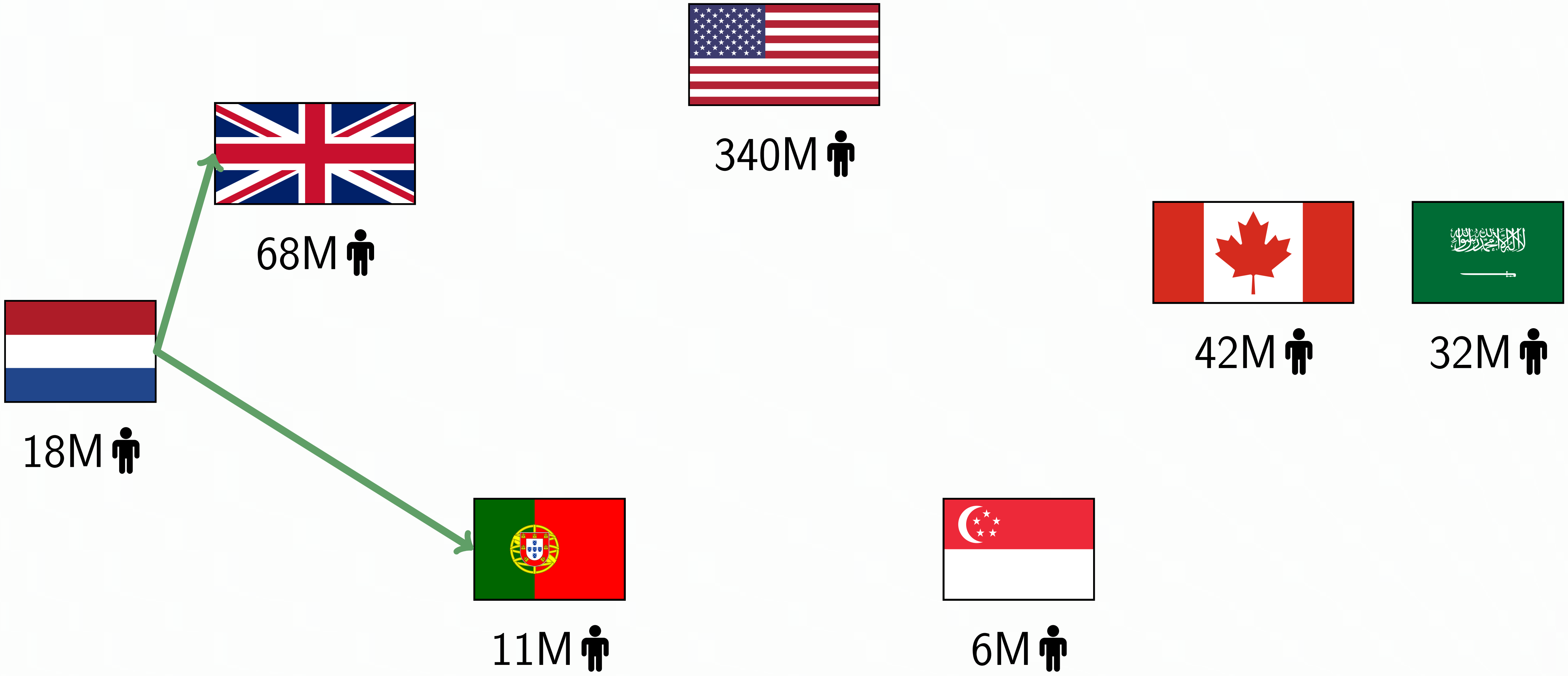
Where are you from?



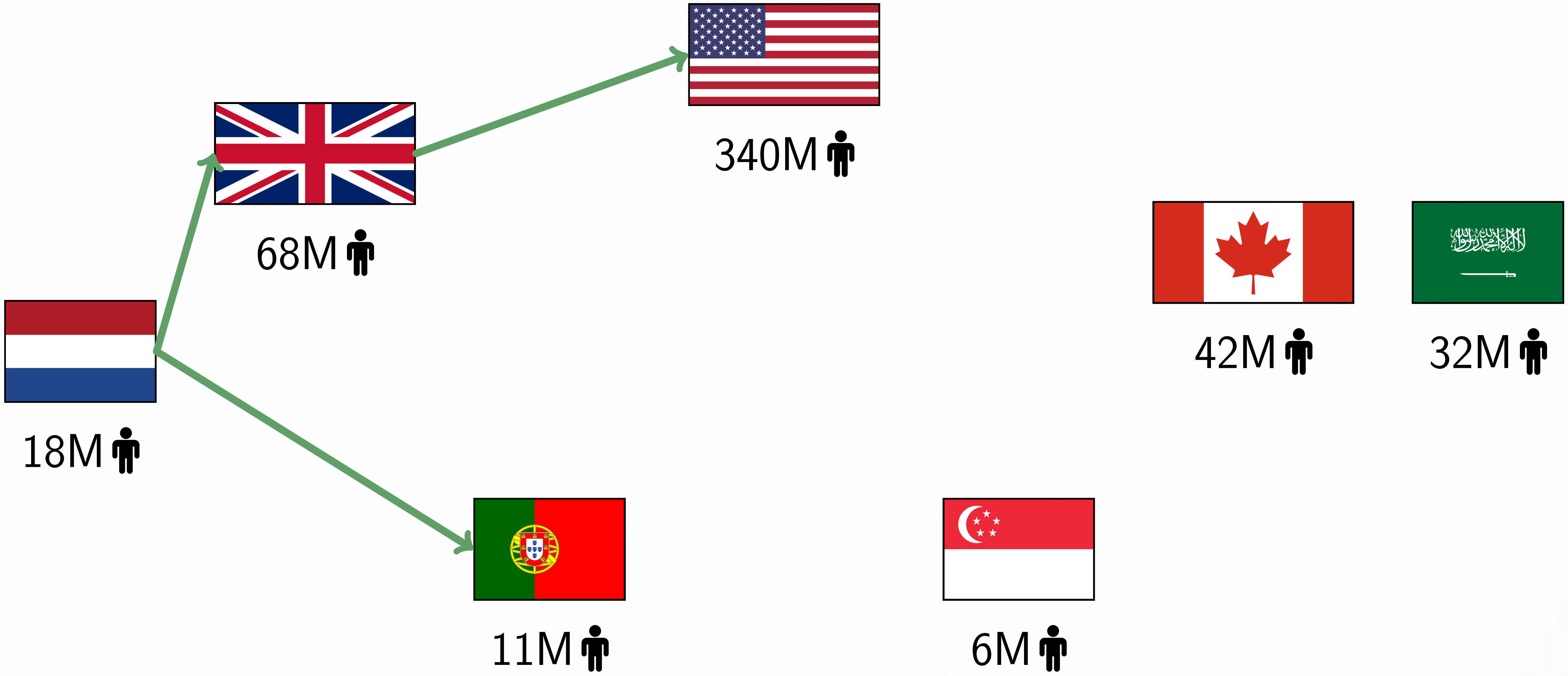
Where are you from?



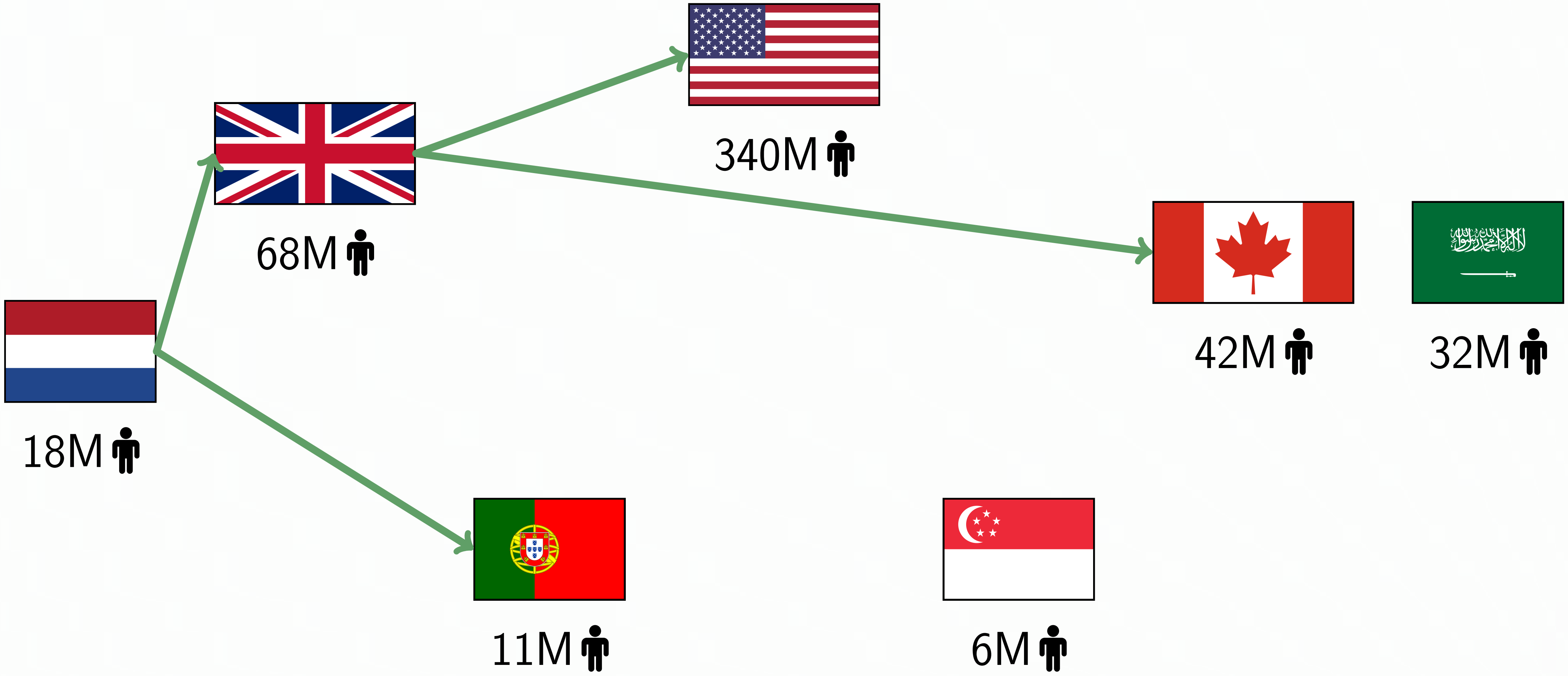
Where are you from?



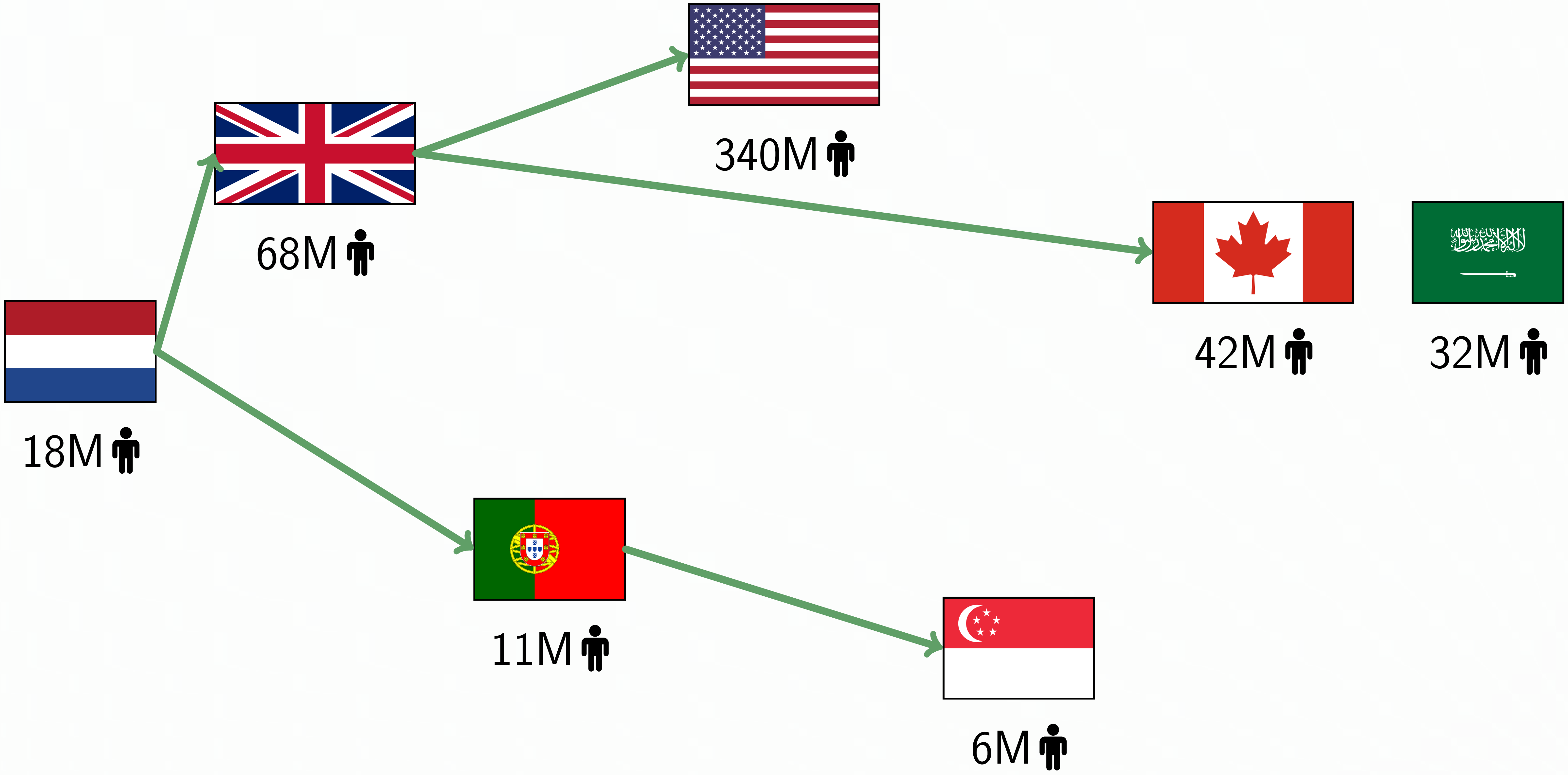
Where are you from?



Where are you from?



Where are you from?



Where are you from?

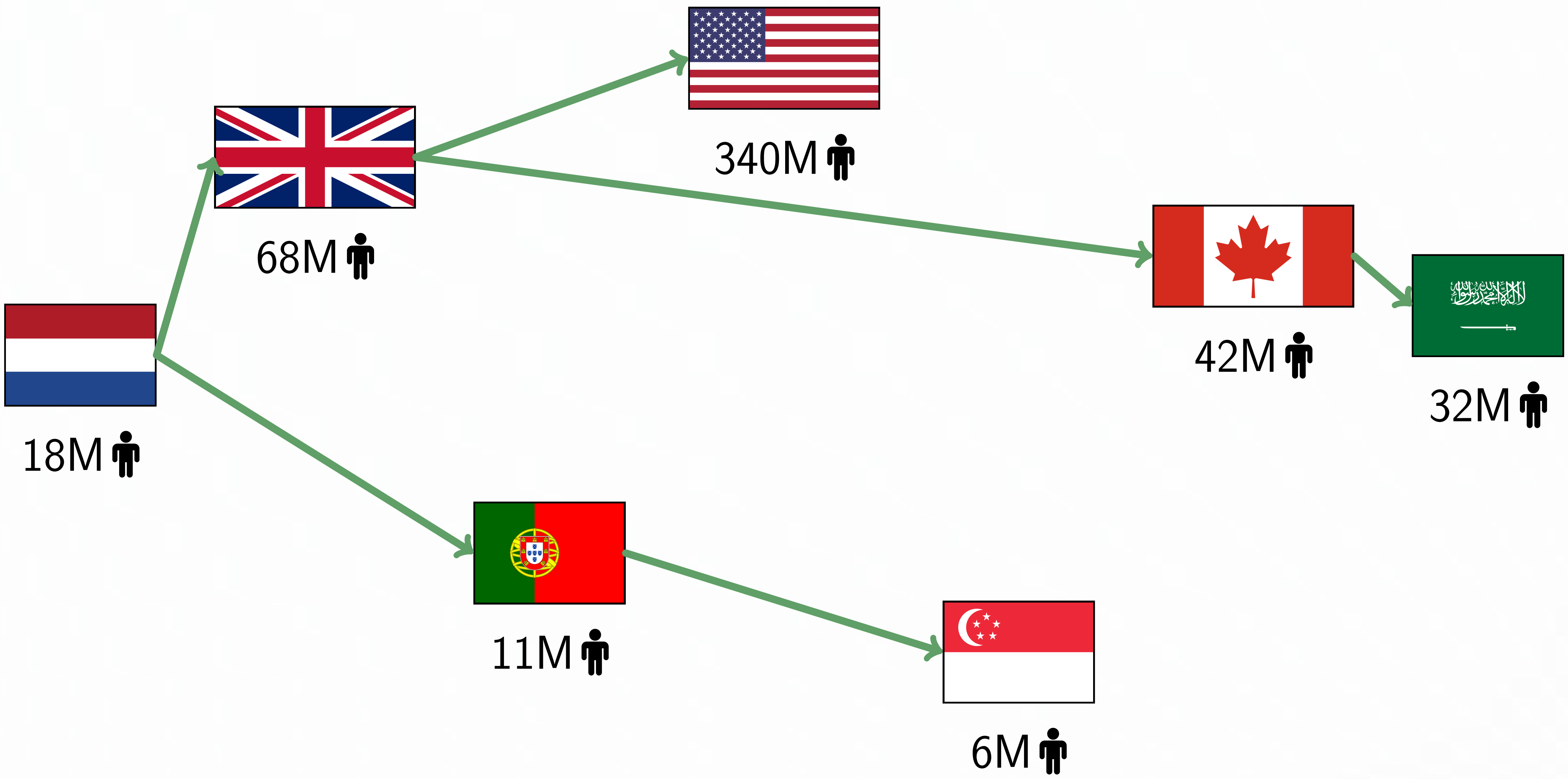


Table of Contents

Binary Search Trees

Binary Search Trees

Definition (Binary Search Trees)

Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

Binary Search Trees

Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

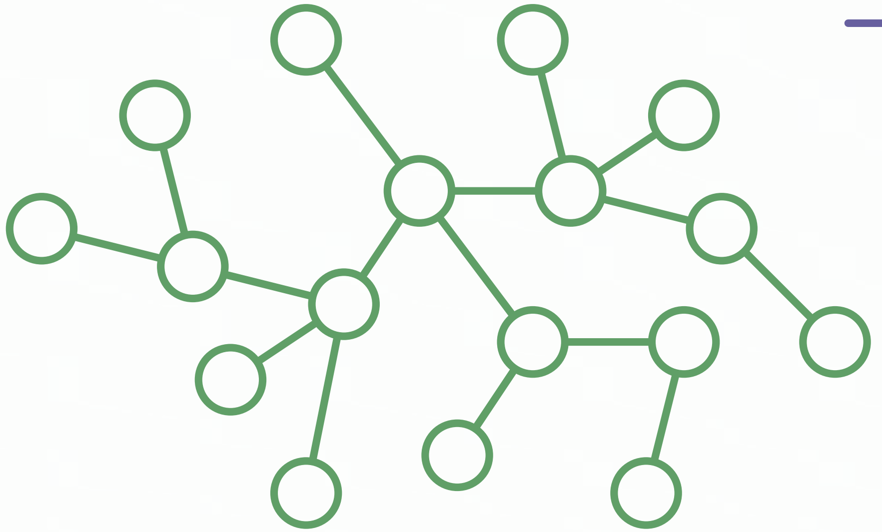
Tree:

Binary Search Trees

Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

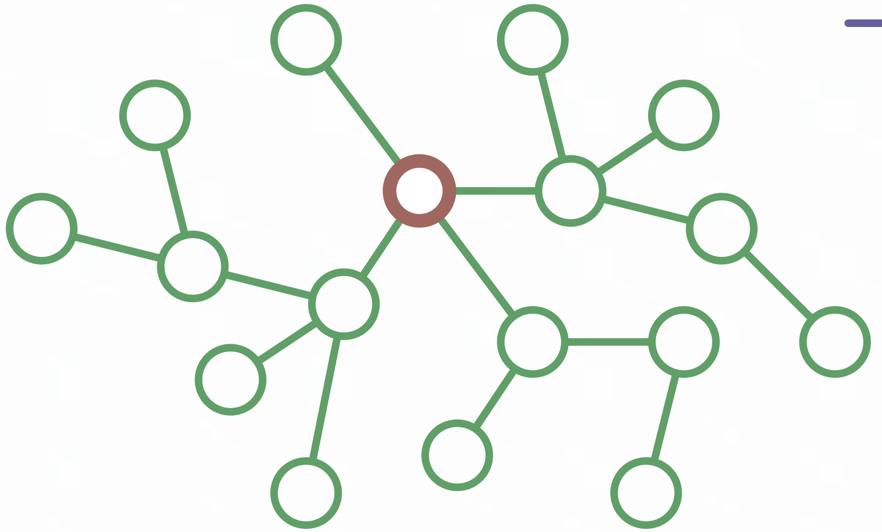
Tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

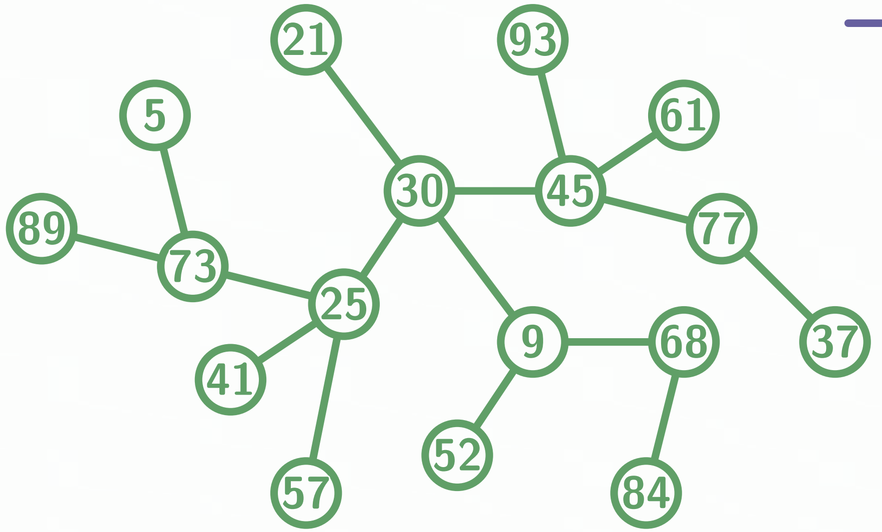
Rooted tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

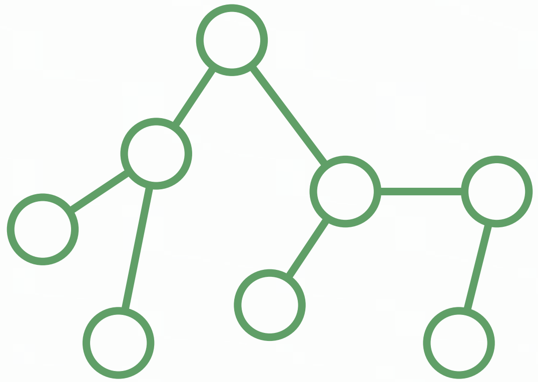
Labelled tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

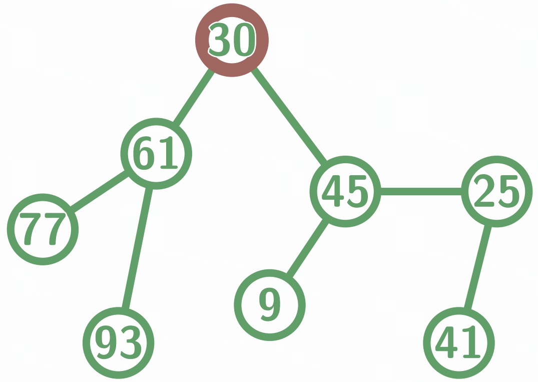
Binary tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

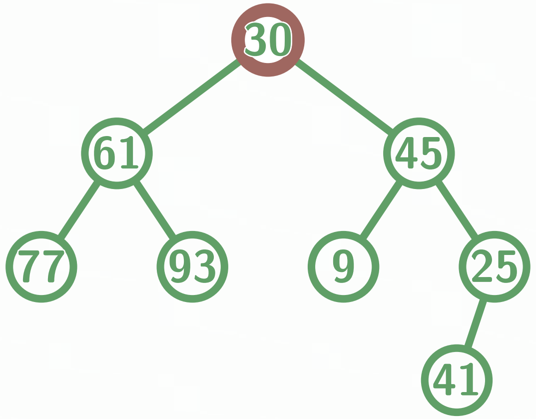
Rooted labelled binary tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

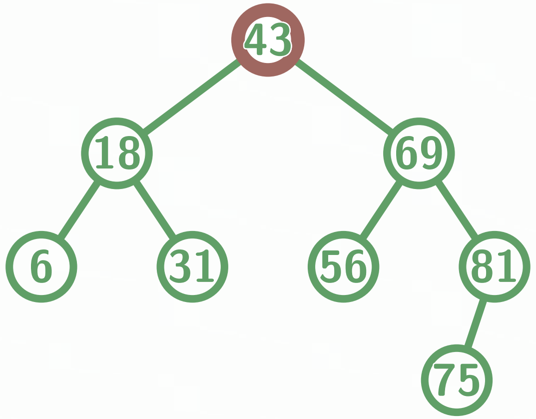
Rooted labelled binary tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

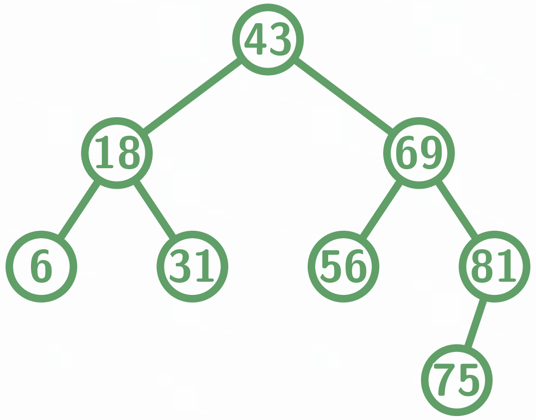
Binary search tree:



Definition (Binary Search Trees)

A binary search tree is a rooted labelled binary tree, where the label of each node is larger than all the labels within its left subtree and smaller than all the labels within its right subtree.

Binary search tree:



Binary Search Trees

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

→ $(43, 18, 69, 6, 31, 56, 81, 75)$

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

→ $(43, 18, 69, 6, 31, 56, 81, 75)$

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$

Binary Search Trees

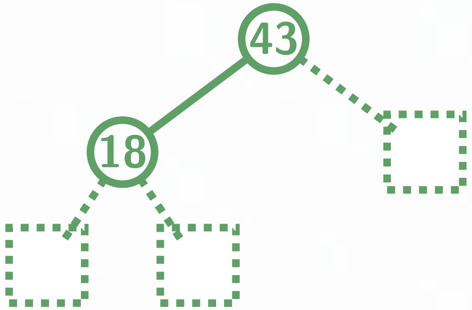
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$

Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

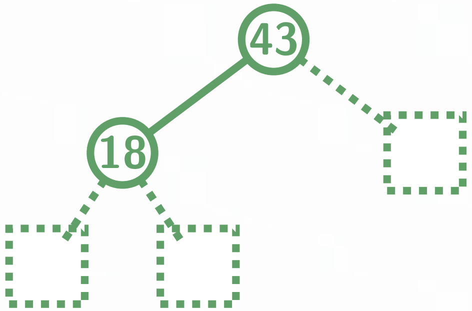
$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

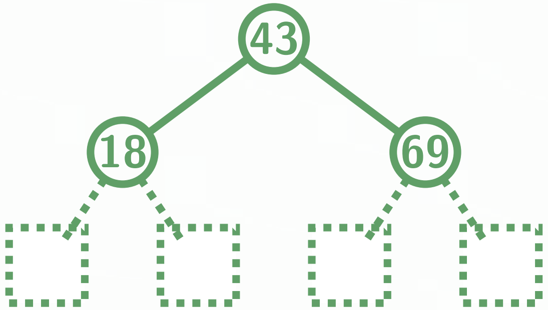
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$



For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

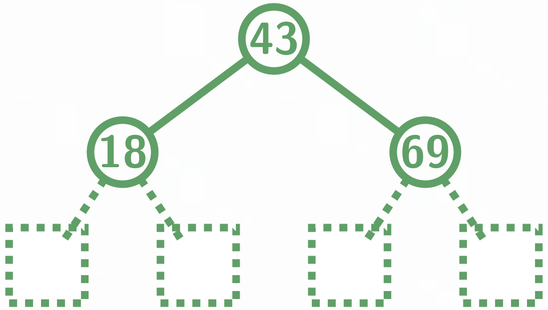
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

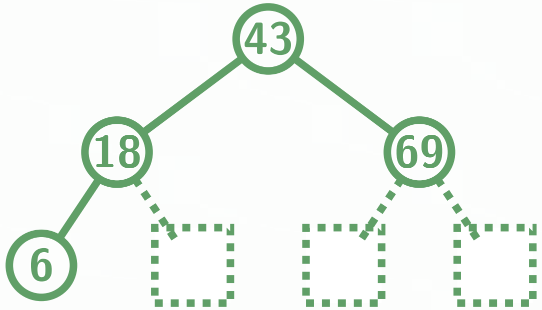
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

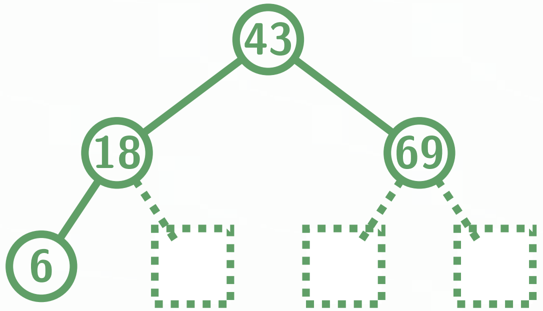
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

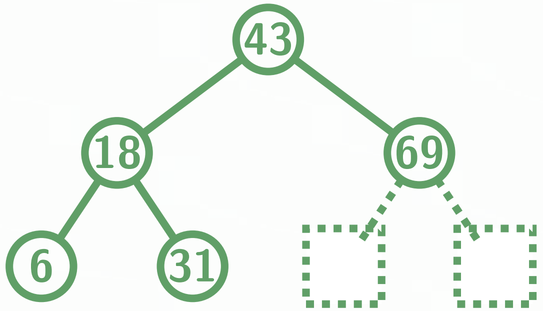
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



For any sequence of distinct numbers $(x_1, \dots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

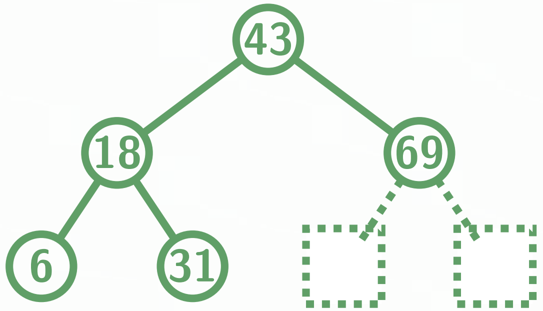
$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

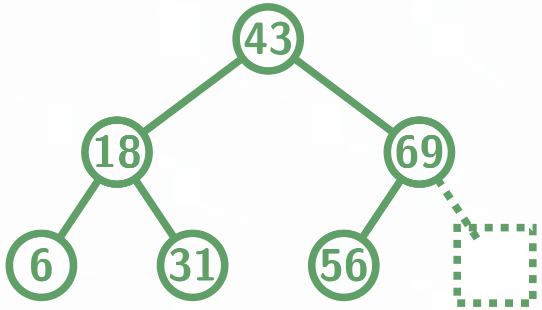
→ $(43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

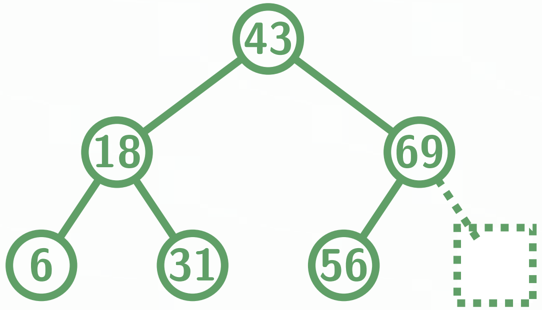
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

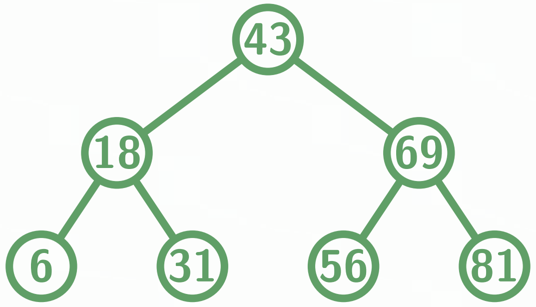
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



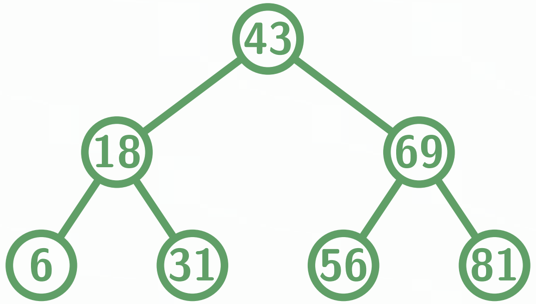
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



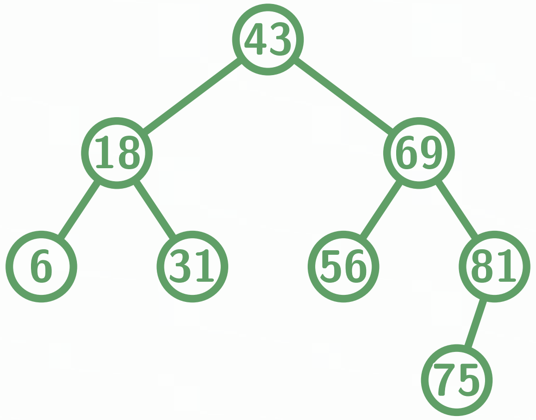
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



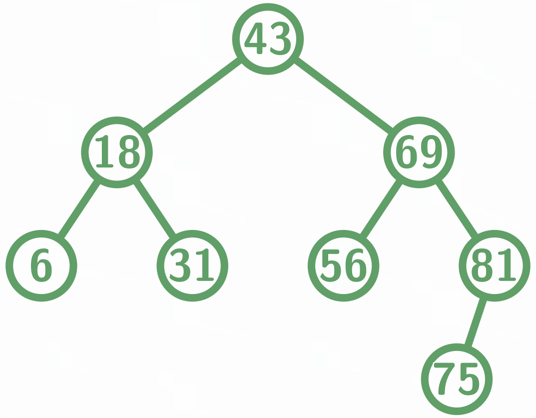
For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

→ $(43, 18, 69, 6, 31, 56, 81, 75)$



For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

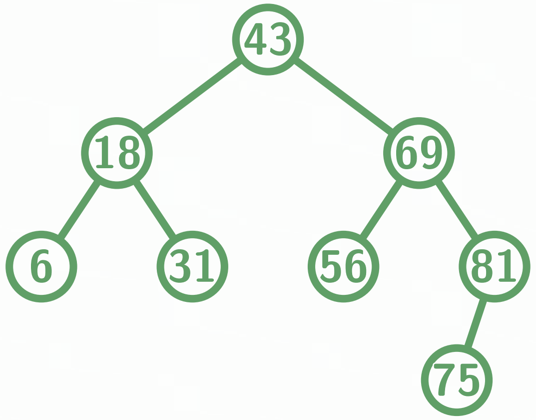
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

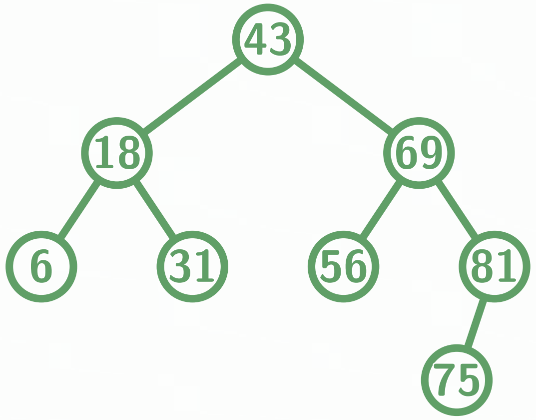
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$
$\rightarrow (43, 18, 6, 31, 69, 56, 81, 75)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

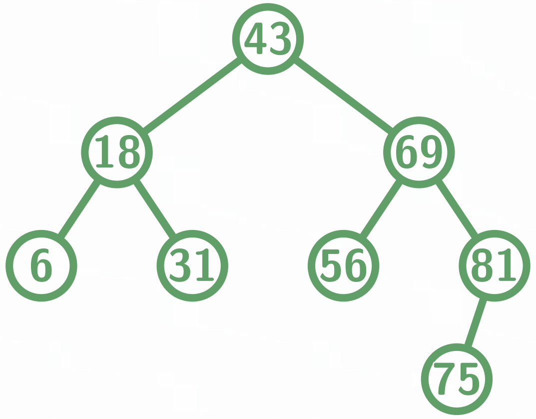
$\rightarrow (43, 18, 69, 6, 31, 56, 81, 75)$
$\rightarrow (43, 18, 6, 31, 69, 56, 81, 75)$
$\rightarrow (43, 69, 81, 75, 56, 18, 31, 6)$



Binary Search Trees

For any sequence of distinct numbers $(x_1, \ldots, x_n)$, there exists a unique binary search tree obtained by recursively inserting the values of the sequence in order.

$\rightarrow$ $(43, 18, 69, 6, 31, 56, 81, 75)$
$\rightarrow$ $(43, 18, 6, 31, 69, 56, 81, 75)$
$\rightarrow$ $(43, 69, 81, 75, 56, 18, 31, 6)$
$\rightarrow$ $(43, 69, 18, 6, 31, 81, 75, 56)$



Examples

Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$.

Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \dots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \dots, n\} \longrightarrow \{1, \dots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \dots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\dots,7\}}$ for the identity on $\{1, \dots, 7\}$, what is the tree corresponding to:

Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \dots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \dots, n\} \longrightarrow \{1, \dots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \dots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\dots,7\}}$ for the identity on $\{1, \dots, 7\}$, what is the tree corresponding to:

- $\sigma = \mathrm{id} = (1, 2, 3, 4, 5, 6, 7)$?

Examples

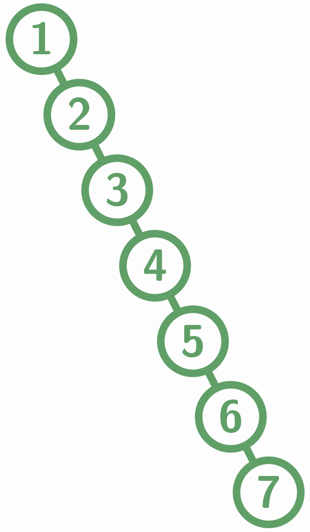
To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\ldots,7\}}$ for the identity on $\{1, \ldots, 7\}$, what is the tree corresponding to:

- $\sigma = \mathrm{id} = (1, 2, 3, 4, 5, 6, 7)$?



Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\ldots,7\}}$ for the identity on $\{1, \ldots, 7\}$, what is the tree corresponding to:

- $\sigma = 8 - \mathrm{id} = (7, 6, 5, 4, 3, 2, 1)$?

Examples

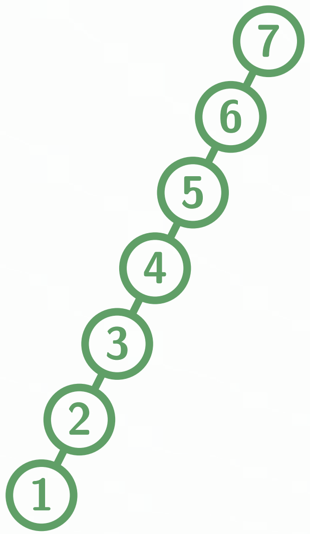
To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\ldots,7\}}$ for the identity on $\{1, \ldots, 7\}$, what is the tree corresponding to:

- $\sigma = 8 - \mathrm{id} = (7, 6, 5, 4, 3, 2, 1)$?



Examples

To simplify the study of binary search trees, we forget about generic sequences $(x_1, \dots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \dots, n\} \longrightarrow \{1, \dots, n\}.$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \dots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\dots,7\}}$ for the identity on $\{1, \dots, 7\}$, what is the tree corresponding to:

- $\sigma = 1 + \big(2 + \mathrm{id} \equiv 7\big) = (4, 5, 6, 7, 1, 2, 3)$?

Examples

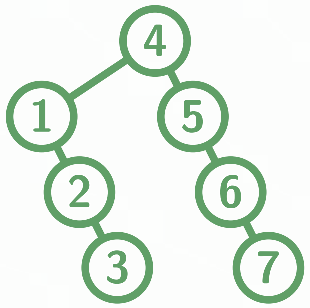
To simplify the study of binary search trees, we forget about generic sequences $(x_1, \ldots, x_n)$ and instead focus on permutations

$$\sigma : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\} .$$

We insert the values of σ in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$.

→ Writing $\mathrm{id} = \mathrm{id}_{\{1,\ldots,7\}}$ for the identity on $\{1, \ldots, 7\}$, what is the tree corresponding to:

- $\sigma = 1 + \left(2 + \mathrm{id} \equiv 7\right) = (4, 5, 6, 7, 1, 2, 3)$?



Research and Applications

Research and Applications

Q: One might wonder why study binary search trees on permutations?

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.
- Permutations provide a more contained and yet diverse framework for sequences.

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.
- Permutations provide a more contained and yet diverse framework for sequences.
- Binary search trees are common structures in computer science:

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.
- Permutations provide a more contained and yet diverse framework for sequences.
- Binary search trees are common structures in computer science:
  - related to the time to access entries in a list;

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.
- Permutations provide a more contained and yet diverse framework for sequences.
- Binary search trees are common structures in computer science:
  - related to the time to access entries in a list; and
  - linked to quick-sort, the fastest and most common sorting algorithm.

Research and Applications

Q: One might wonder why study binary search trees on permutations?

- Studying generic sequences $(x_1, \ldots, x_n)$ or permutations σ is equivalent.
- Permutations provide a more contained and yet diverse framework for sequences.
- Binary search trees are common structures in computer science:
  - related to the time to access entries in a list; and
  - linked to quick-sort, the fastest and most common sorting algorithm.

→ There are also a relatively "recent" object (early results from the end of the 70[4]) with a lot of interesting research left to pursue!

Table of Contents

Random Binary Search Trees

Random Binary Search Trees

Q: If you had to create a random model of binary search trees, how would you do it?

Random Binary Search Trees

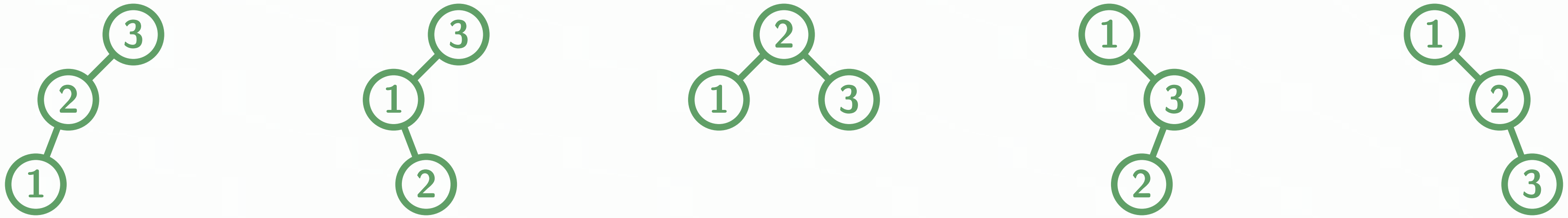
Q: If you had to create a random model of binary search trees, how would you do it?

→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).

Random Binary Search Trees

Q: If you had to create a random model of binary search trees, how would you do it?

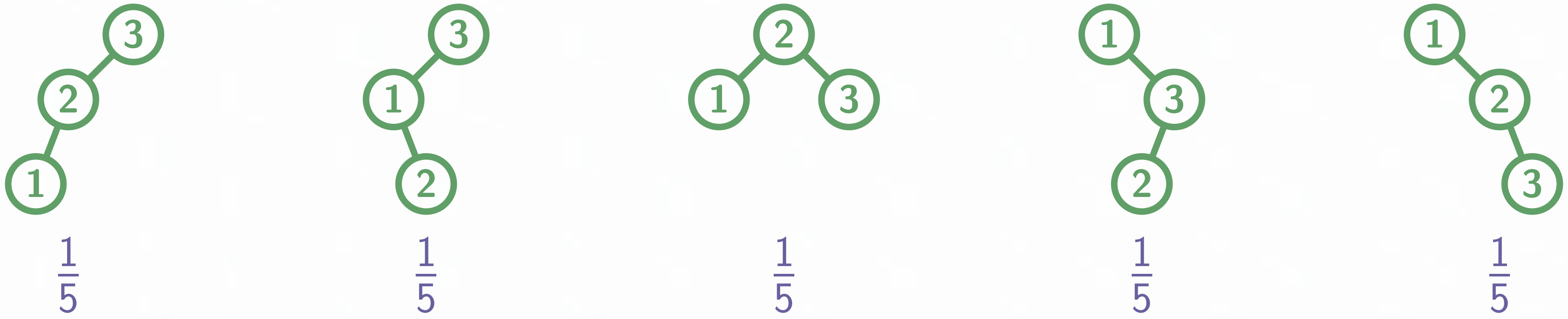
→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).



Random Binary Search Trees

Q: If you had to create a random model of binary search trees, how would you do it?

→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).

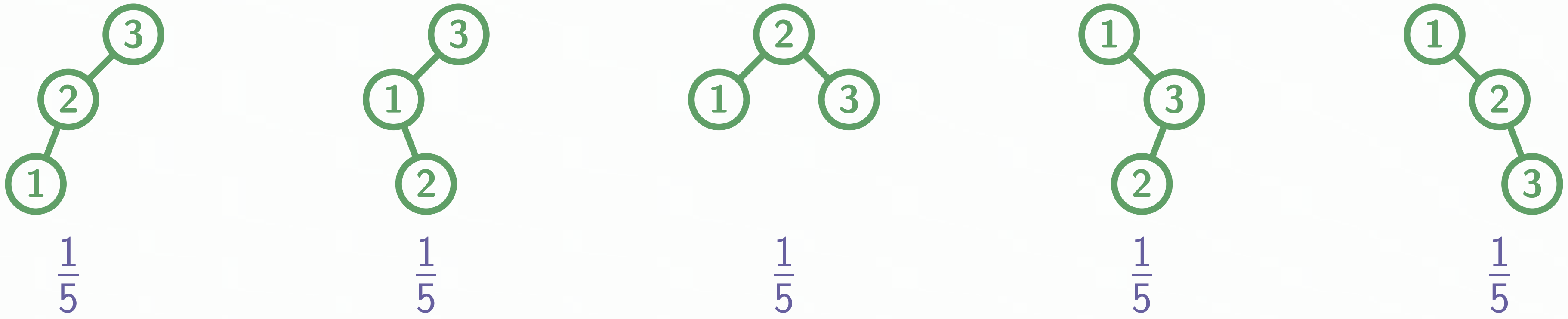


Random Binary Search Trees

Q: If you had to create a random model of binary search trees, how would you do it?

→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).

Note: Here, it is actually easier to use uniform permutations instead of uniform trees.

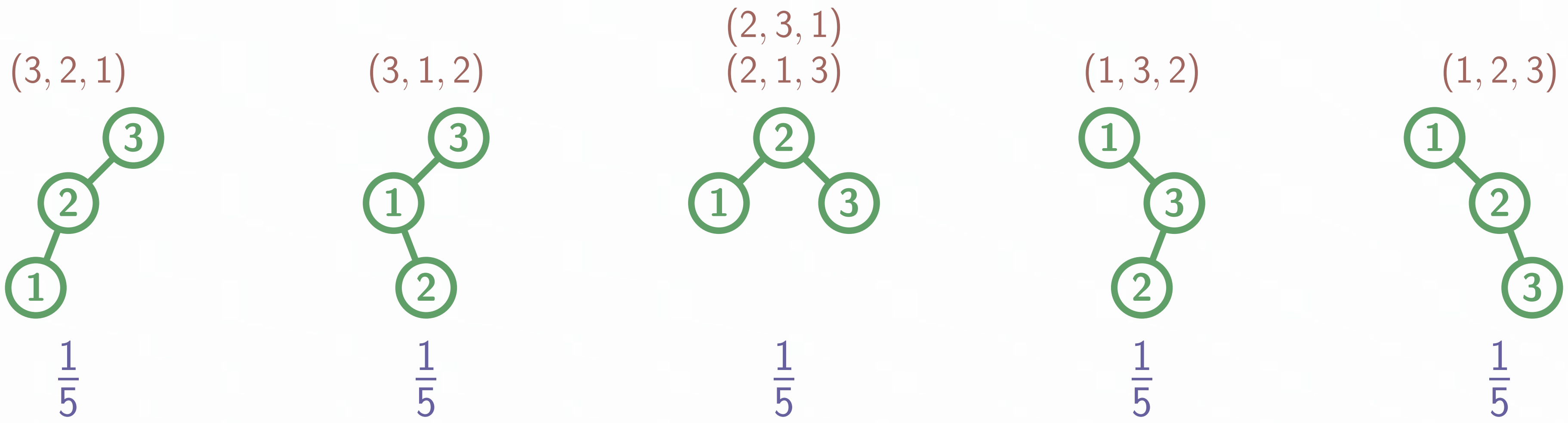


Random Binary Search Trees

Q: If you had to create a random model of binary search trees, how would you do it?

→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).

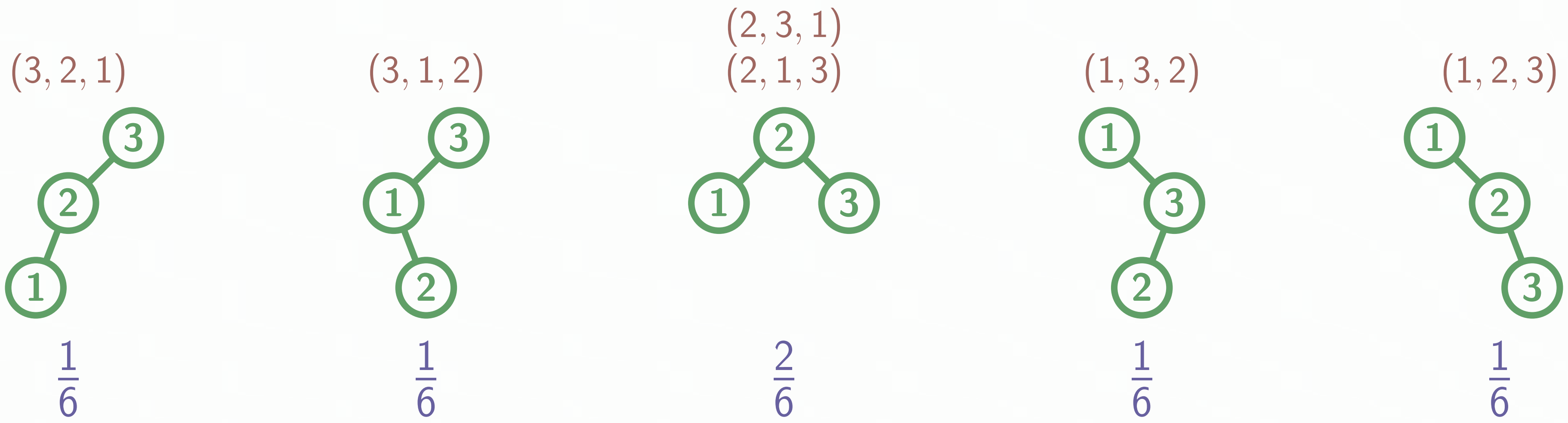
Note: Here, it is actually easier to use uniform permutations instead of uniform trees.



Q: If you had to create a random model of binary search trees, how would you do it?

→ The usual "easy" way to generate random objects is to pick one uniformly at random from a given set (for example all trees of size n here).

Note: Here, it is actually easier to use uniform permutations instead of uniform trees.



Random Binary Search Trees

Random Binary Search Trees

Definition (RBST)

Random Binary Search Trees

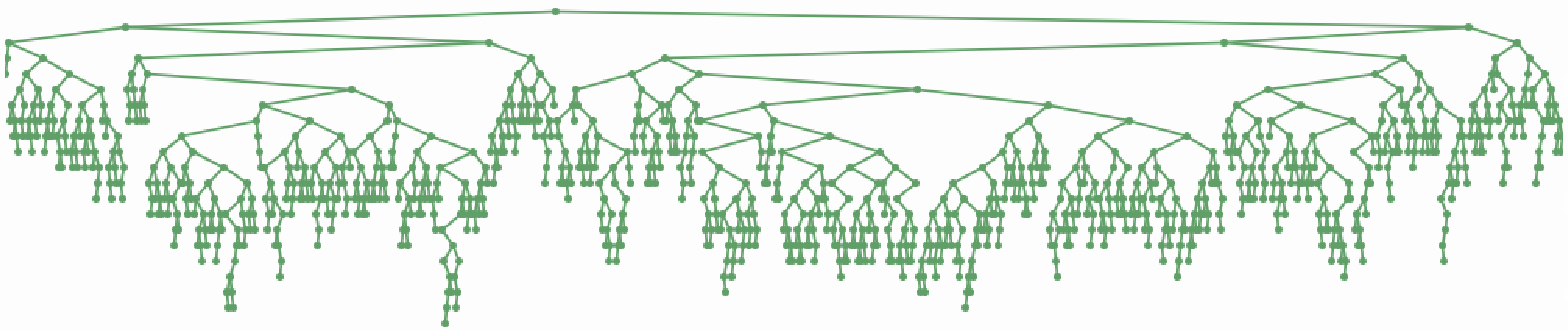
Definition (RBST)

A *Random Binary Search Tree* (RBST) of size n is the binary search tree obtained by inserting a uniform permutation of size n into a binary search tree.

Random Binary Search Trees

Definition (RBST)

A *Random Binary Search Tree* (RBST) of size n is the binary search tree obtained by inserting a uniform permutation of size n into a binary search tree.



Properties of RBSTs

Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

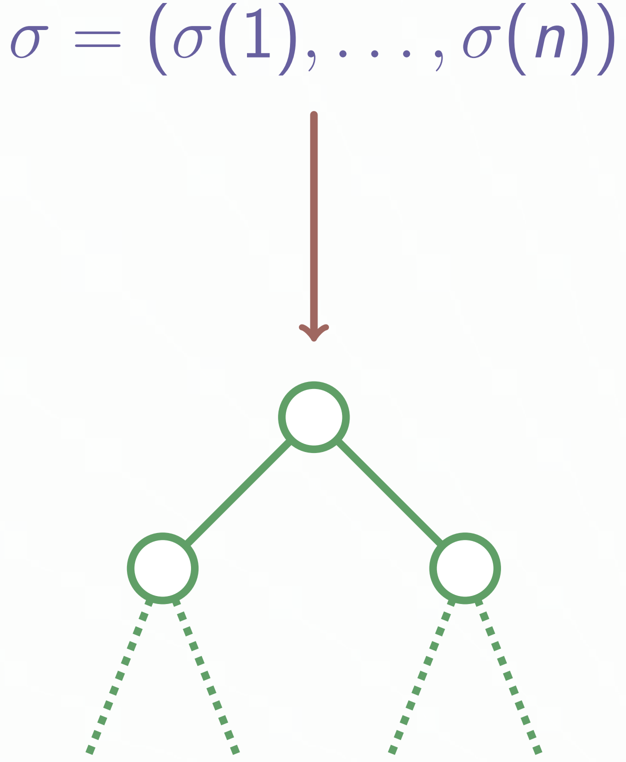
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ into a binary search tree, we can make the following observations.

$$\sigma = (\sigma(1), \dots, \sigma(n))$$

Properties of RBSTs

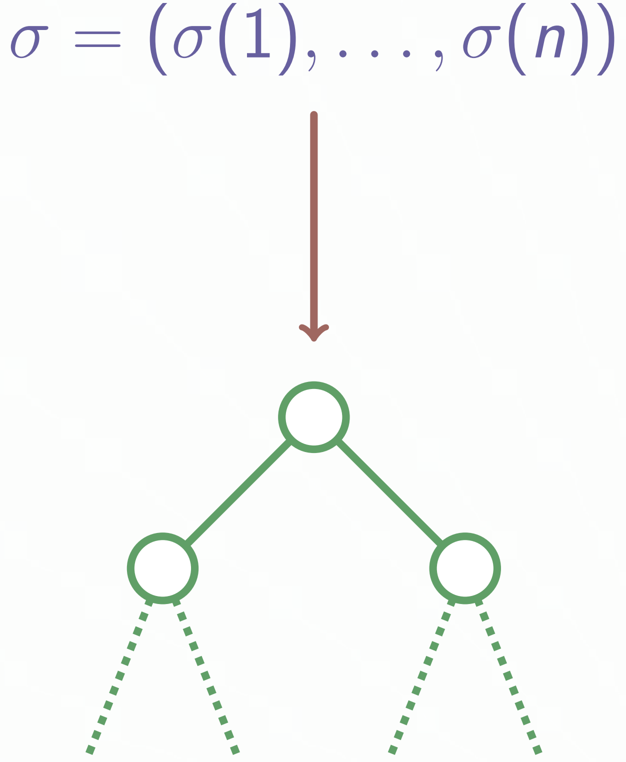
When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

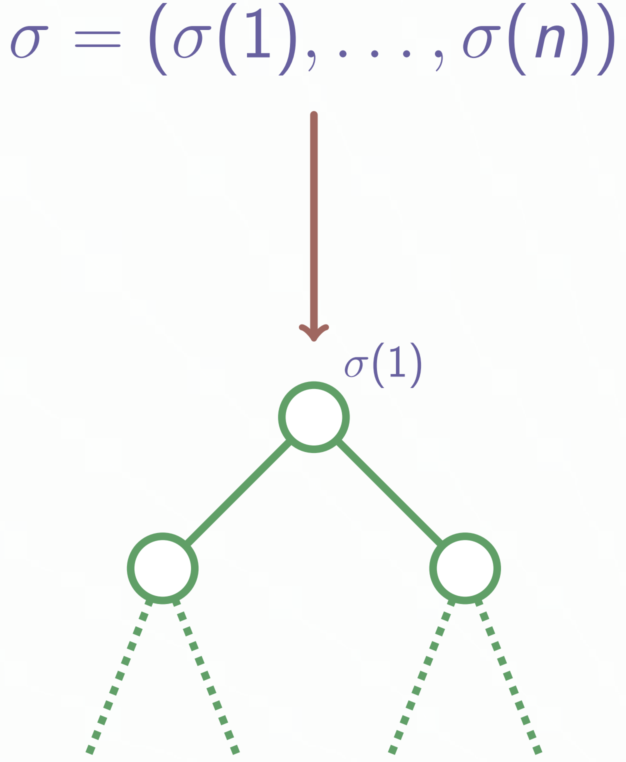
- The value $\sigma(1)$ is placed at the root.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

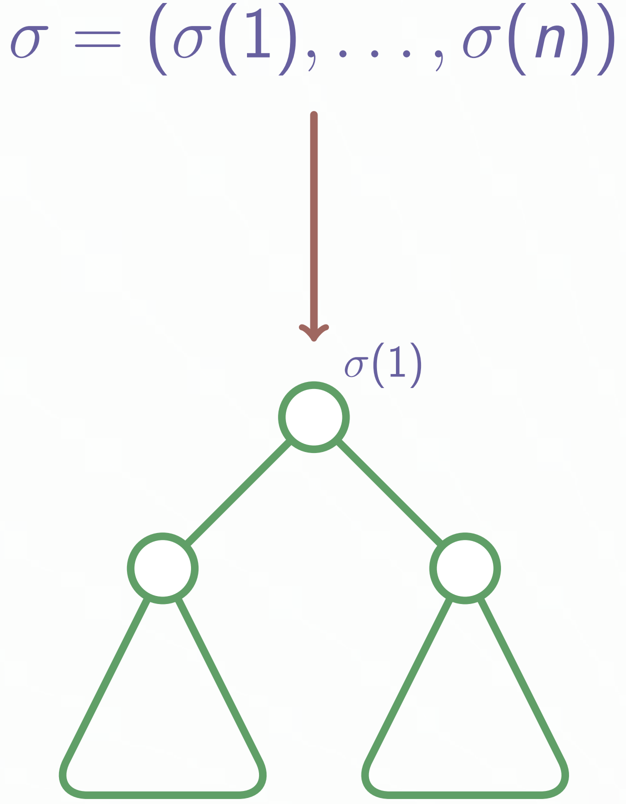
- The value $\sigma(1)$ is placed at the root.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

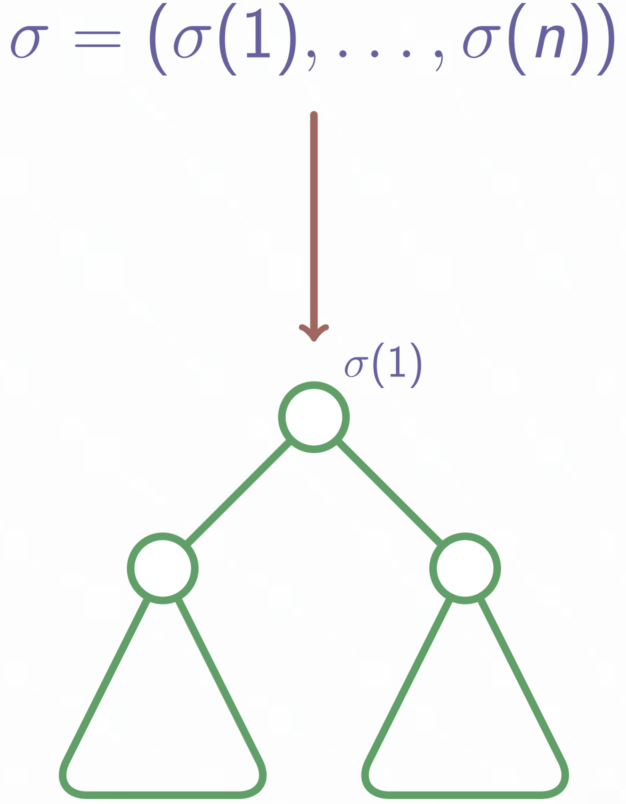
- The value $\sigma(1)$ is placed at the root.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

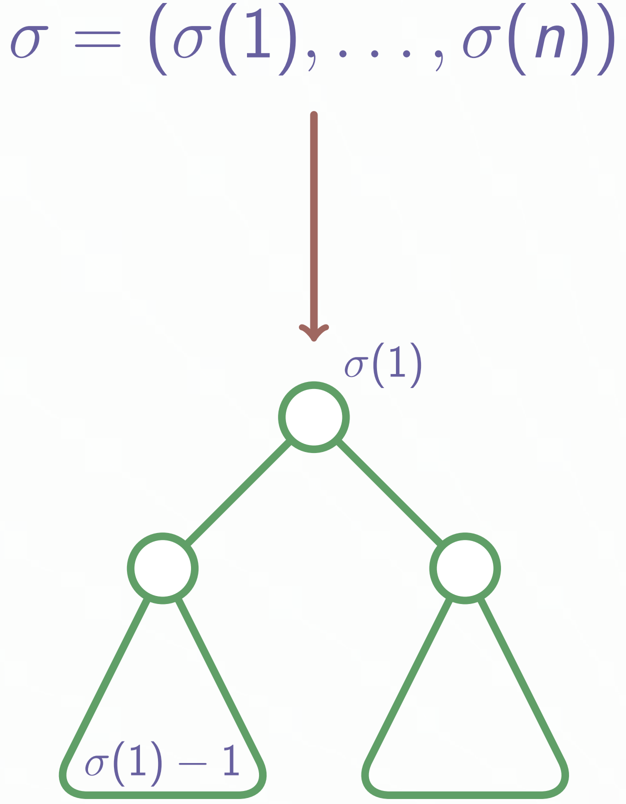
- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

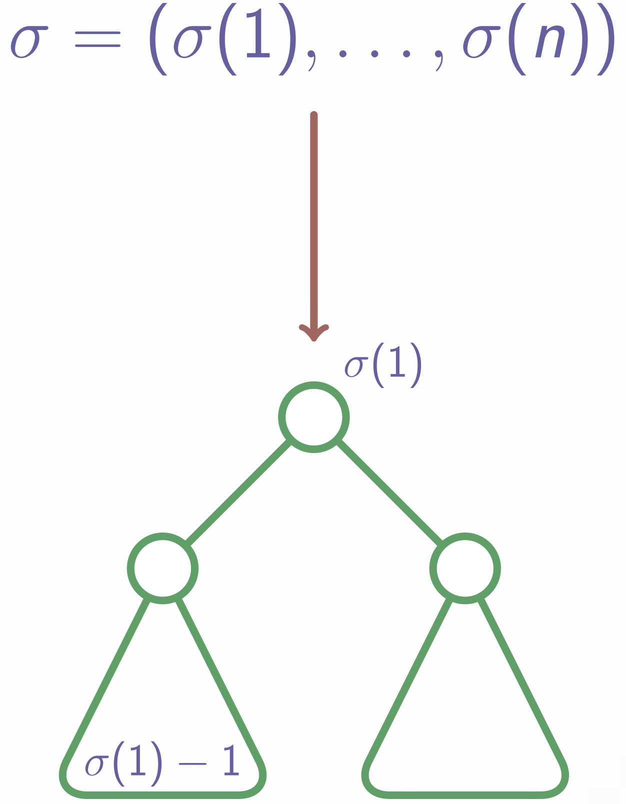
- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ into a binary search tree, we can make the following observations.

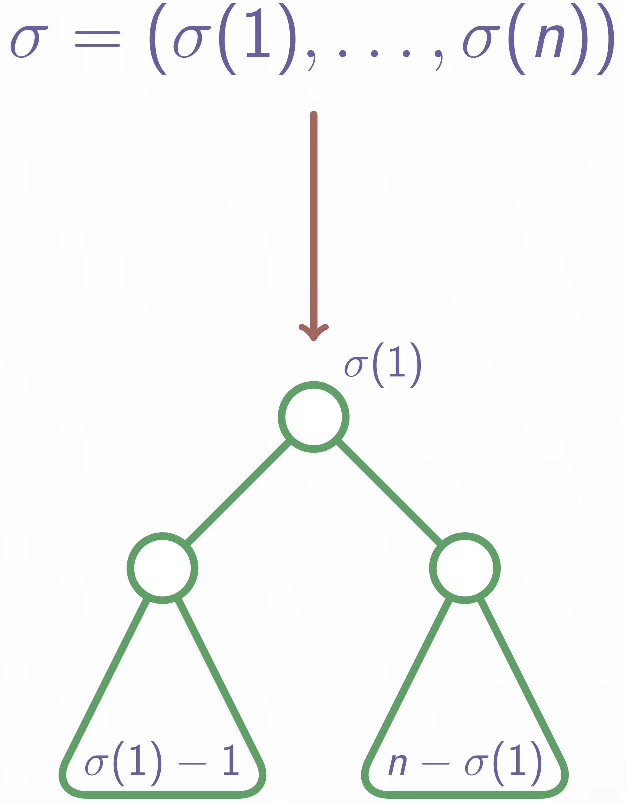
- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

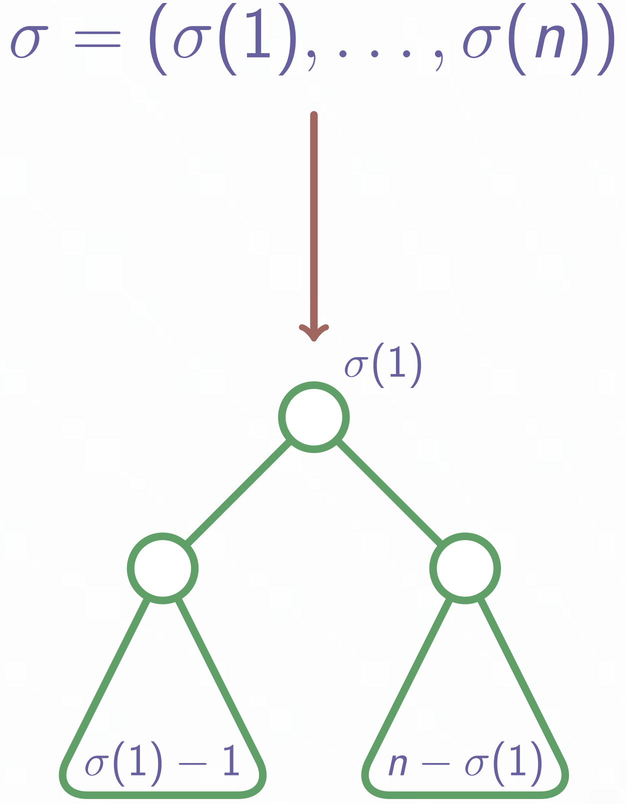
- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.



Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.



When σ is uniform, we also note the following.

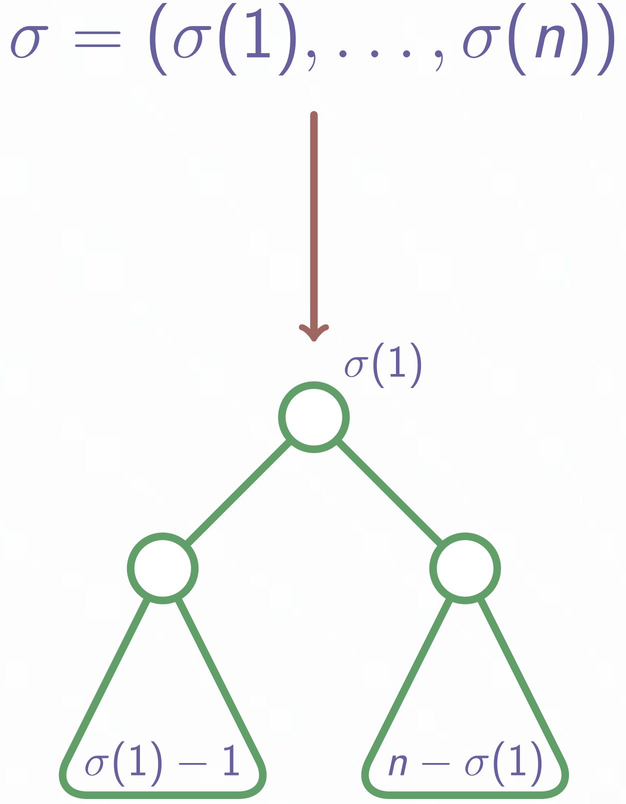
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.



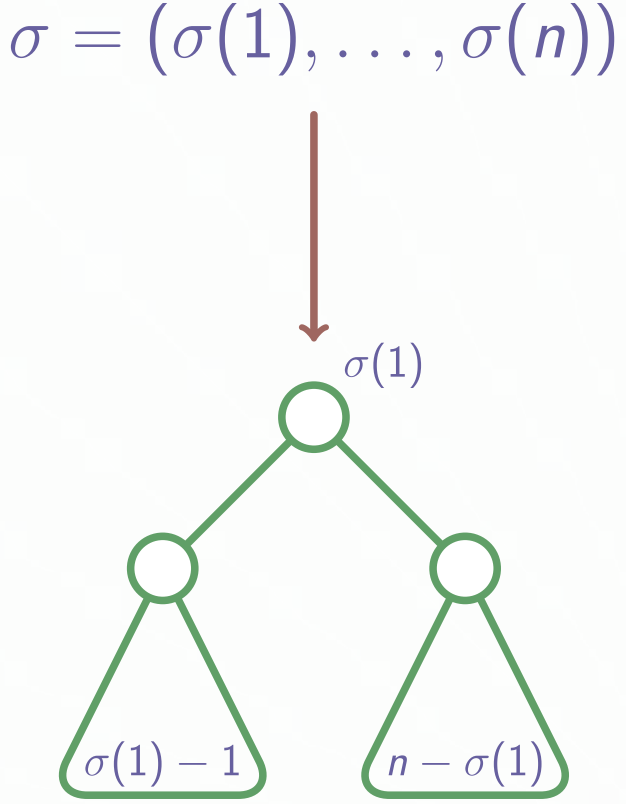
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.



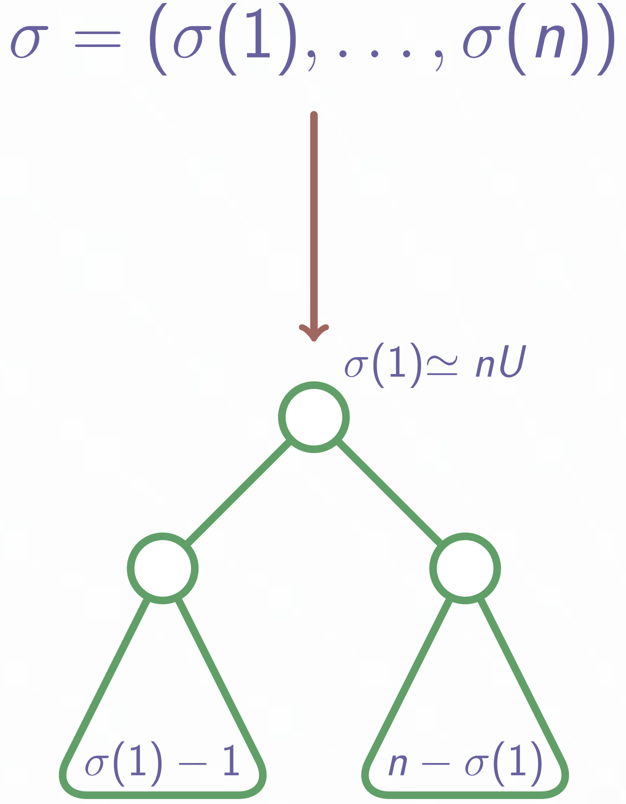
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.



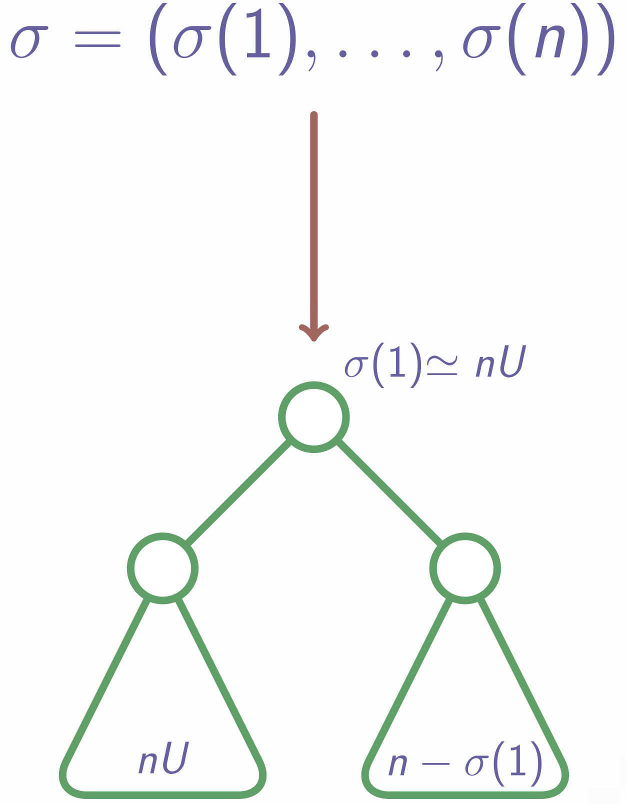
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.



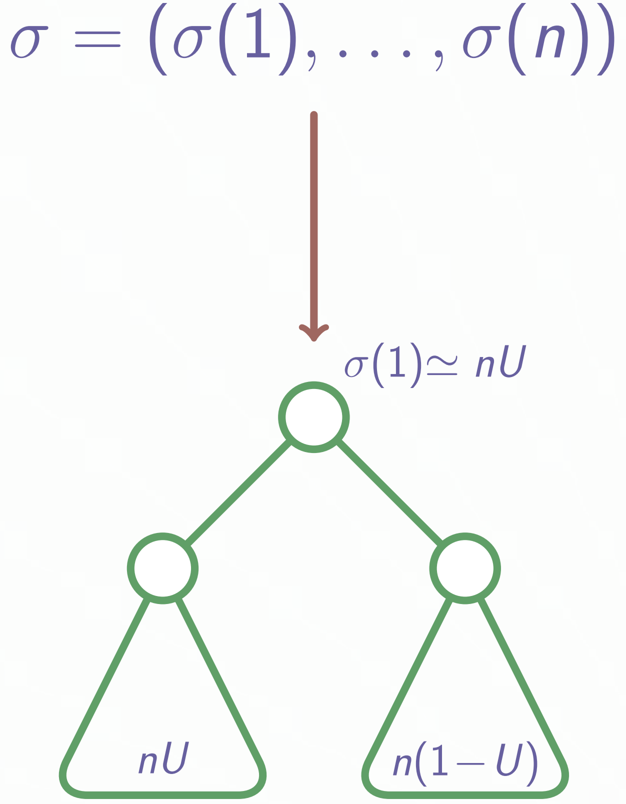
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \dots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.



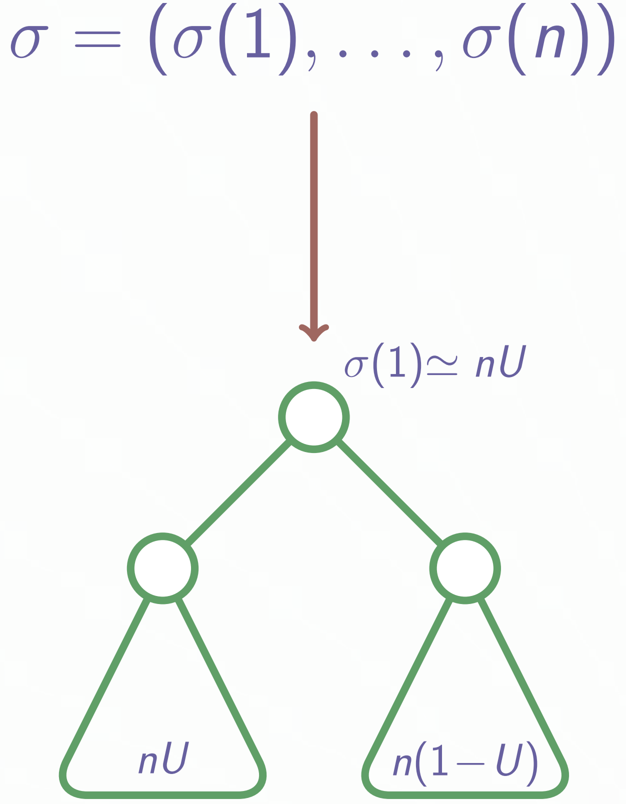
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.
- The ordering of $(\sigma(i) : \sigma(i) < \sigma(1))$ is uniform.



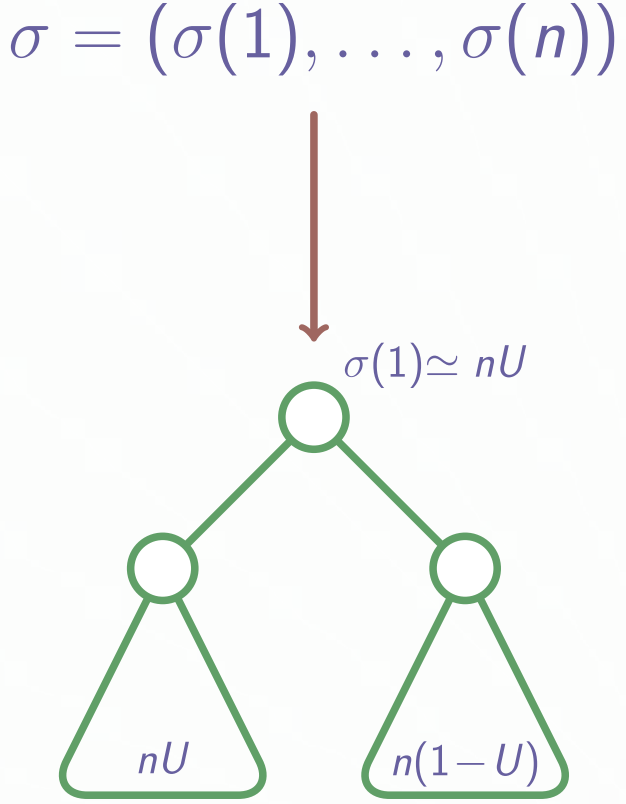
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.
- The ordering of $(\sigma(i) : \sigma(i) < \sigma(1))$ is uniform.
- The ordering of $(\sigma(i) : \sigma(i) > \sigma(1))$ is uniform.



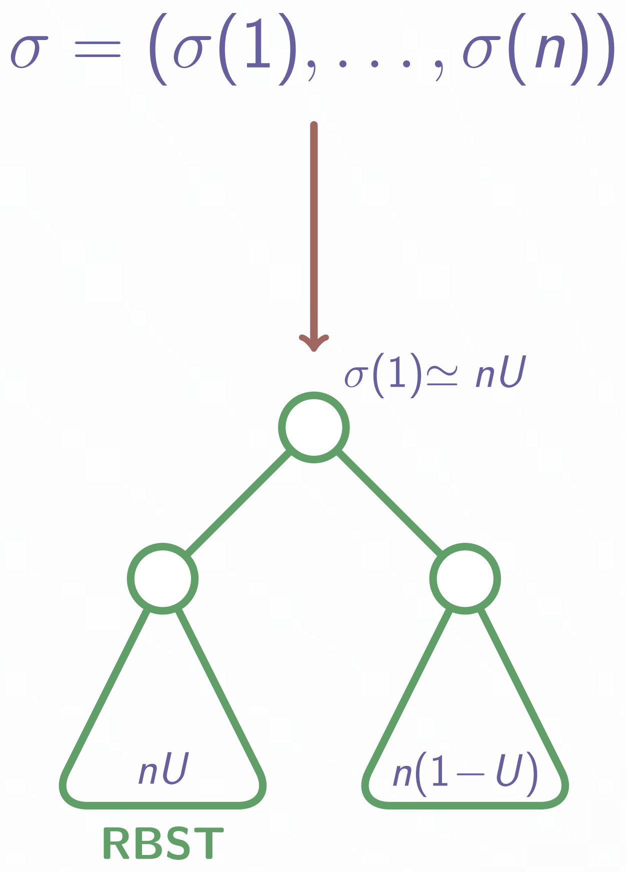
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.
- The ordering of $(\sigma(i) : \sigma(i) < \sigma(1))$ is uniform.
- The ordering of $(\sigma(i) : \sigma(i) > \sigma(1))$ is uniform.



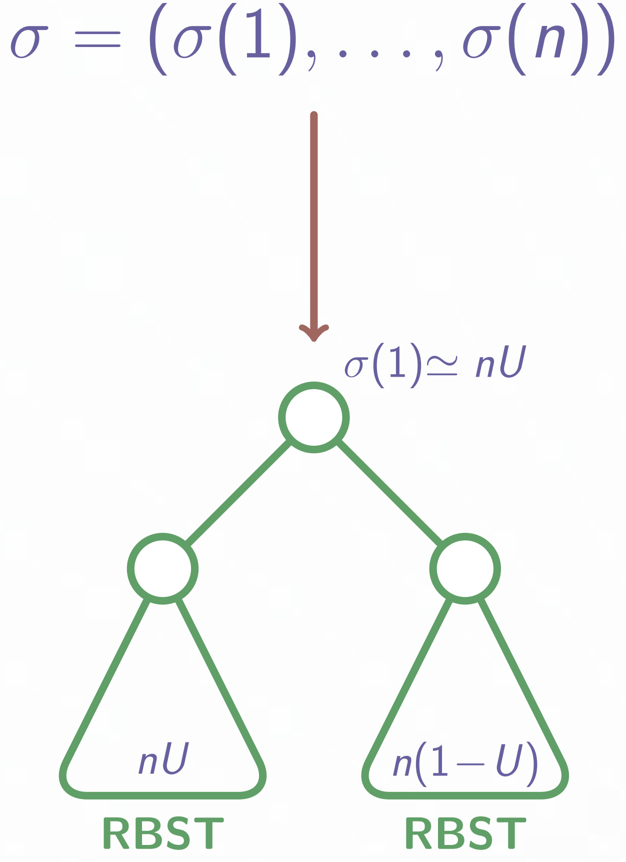
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \dots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.
- The ordering of $(\sigma(i) : \sigma(i) < \sigma(1))$ is uniform.
- The ordering of $(\sigma(i) : \sigma(i) > \sigma(1))$ is uniform.



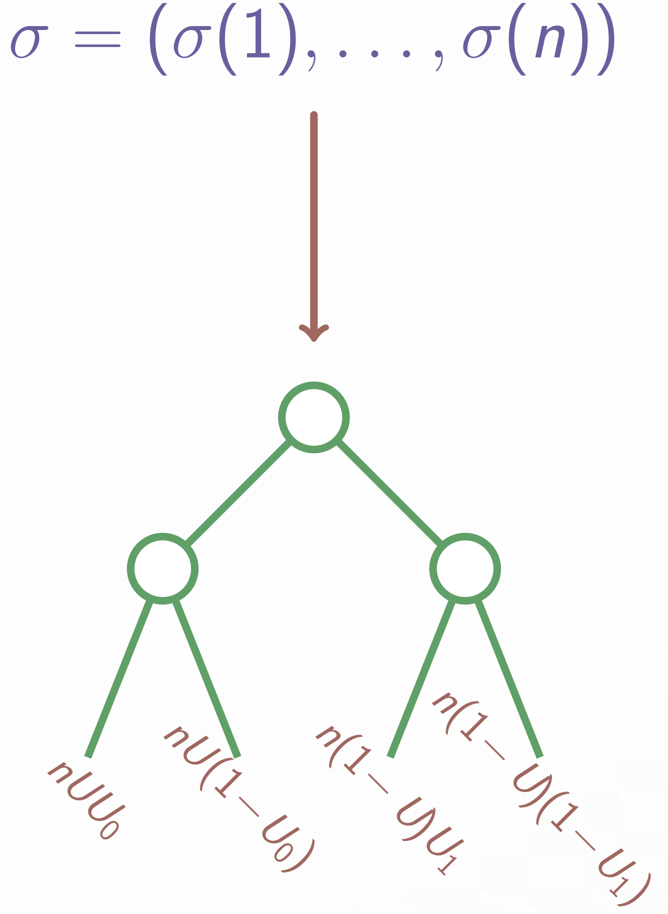
Properties of RBSTs

When placing a permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ into a binary search tree, we can make the following observations.

- The value $\sigma(1)$ is placed at the root.
- The number of values on the left is $\sigma(1) - 1$.
- The number of values on the right is $n - \sigma(1)$.

When σ is uniform, we also note the following.

- The value $\sigma(1)$ is uniform on $\{1, \ldots, n\}$.
- We can use the approximation $\sigma(1) \simeq nU$.
- The ordering of $(\sigma(i) : \sigma(i) < \sigma(1))$ is uniform.
- The ordering of $(\sigma(i) : \sigma(i) > \sigma(1))$ is uniform.



Back To Basics

Back To Basics

How many questions do you need to ask to find the correct country?

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

- countries are in a random order;

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

- countries are in a random order; and
- the order of the "correct" country is also random.

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

- countries are in a random order; and
- the order of the "correct" country is also random.

This is equivalent to:

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

- countries are in a random order; and
- the order of the "correct" country is also random.

This is equivalent to:

- considering a uniform permutation;

Back To Basics

How many questions do you need to ask to find the correct country?

Recall that:

- countries are in a random order; and
- the order of the "correct" country is also random.

This is equivalent to:

- considering a uniform permutation; and
- asking how far down the tree a random number is.

Solving The Problem

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables.

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] =$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\,T_n \mid U\,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}] \,.$$

- Taking the expected value, the previous formula re-writes as

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U\,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] =$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] = 1 + 2\mathbb{E}[UT_{nU}]$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U\,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] = 1 + 2\mathbb{E}[UT_{nU}] = 1 + 2\int_0^1 u\mathbb{E}[T_{nu}]du$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] = 1 + 2\mathbb{E}[UT_{nU}] = 1 + 2\int_0^1 u\mathbb{E}[T_{nu}]du = 1 + \frac{2}{n^2}\int_0^n u\mathbb{E}[T_u]du\,.$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right.
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] = 1 + 2\mathbb{E}[UT_{nU}] = 1 + 2\int_0^1 u\mathbb{E}[T_{nu}]du = 1 + \frac{2}{n^2}\int_0^n u\mathbb{E}[T_u]du\,.$$

→ Let us solve this equation.

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2} \int_0^n u f(u) du \, .$$

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2} \int_0^n u f(u) du \,.$$

- Changing from discrete n to continuous x, we can take the derivative:

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2} \int_0^n uf(u)du \, .$$

- Changing from discrete n to continuous x, we can take the derivative:

$$f'(x)$$

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2} \int_0^n uf(u)du \,.$$

- Changing from discrete n to continuous x, we can take the derivative:

$$f'(x) = -\frac{4}{x^3} \int_0^x uf(u)du + \frac{2}{x^2} xf(x)$$

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2}\int_0^n uf(u)du\,.$$

- Changing from discrete n to continuous x, we can take the derivative:

$$f'(x) = -\frac{4}{x^3}\int_0^x uf(u)du + \frac{2}{x^2}xf(x) = -\frac{2}{x}\Big(f(x) - 1\Big) + \frac{2f(x)}{x}$$

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2}\int_0^n uf(u)du\,.$$

- Changing from discrete n to continuous x, we can take the derivative:

$$f'(x) = -\frac{4}{x^3}\int_0^x uf(u)du + \frac{2}{x^2}xf(x) = -\frac{2}{x}\Big(f(x) - 1\Big) + \frac{2f(x)}{x} = \frac{2}{x}\,.$$

Solving The Problem

We observed that $f(n) = \mathbb{E}[T_n]$ satisfies $f(1) = 1$ and

$$f(n) = 1 + \frac{2}{n^2}\int_0^n uf(u)du\,.$$

- Changing from discrete n to continuous x, we can take the derivative:

$$f'(x) = -\frac{4}{x^3}\int_0^x uf(u)du + \frac{2}{x^2}xf(x) = -\frac{2}{x}\Big(f(x) - 1\Big) + \frac{2f(x)}{x} = \frac{2}{x}\,.$$

- The solution is exactly $f(x) = 1 + 2\log x$, implying that $\mathbb{E}[T_n] = 1 + 2\log n$.

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U \,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula re-writes as

$$\mathbb{E}[T_n] = 1 + 2\mathbb{E}[UT_{nU}] = 1 + 2\int_0^1 u\mathbb{E}[T_{nu}]du = 1 + \frac{2}{n^2}\int_0^n u\mathbb{E}[T_u]du\,.$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U\,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula implies that

$$\mathbb{E}[T_n] = 1 + 2\log n\,.$$

Solving The Problem

We recall that a RBST approximately correspond to splitting the tree recursively using random variables. We let T_n be the (random) number of steps to find the correct value.

- There are nU nodes on the left and $n(1-U)$ nodes on the right
- Given U, the expected number number of steps is

$$\mathbb{E}[\, T_n \mid U\,] = 1 + U\mathbb{E}[T_{nU}] + (1-U)\mathbb{E}[T_{n(1-U)}]\,.$$

- Taking the expected value, the previous formula implies that

$$\mathbb{E}[T_n] = 1 + 2\log n\,.$$

$\rightarrow$ On average it takes us around $2\log n$ steps to find the correct value!

Remarks

Remarks

A few remarks on the previous proof.

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.
  - They rely on the structure of the permutation.

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.
  - They rely on the structure of the permutation.
- This method can be generalized to other various methods.

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.
  - They rely on the structure of the permutation.
- This method can be generalized to other various methods.
  - What if we split in three instead of two?

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.
    - They rely on the structure of the permutation.
- This method can be generalized to other various methods.
    - What if we split in three instead of two?
    - What if the split between left and right is not uniform?

Remarks

A few remarks on the previous proof.

- The previous solution only provides an approximation for $\mathbb{E}[T_n]$.
- Some (even simpler) methods can provide a tighter formula for $\mathbb{E}[T_n]$.
  - They rely on the structure of the permutation.
- This method can be generalized to other various methods.
  - What if we split in three instead of two?
  - What if the split between left and right is not uniform?
- This "splitting tree" approach is quite robust to show other properties of the tree, such as its height[2], corresponding to the worst case scenario.

Table of Contents

Infinite Trees

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

$\rightarrow$ Is it possible to extend this definition to infinite sequences?

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

$\rightarrow$ Is it possible to extend this definition to infinite sequences?

There are three types of infinite sequences:

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

$\rightarrow$ Is it possible to extend this definition to infinite sequences?

There are three types of infinite sequences:

- Right-sided: $(x_1, x_2, \ldots, x_n, \ldots)$.

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

$\rightarrow$ Is it possible to extend this definition to infinite sequences?

There are three types of infinite sequences:

- Right-sided: $(x_1, x_2, \ldots, x_n, \ldots)$.
- Left-sided: $(\ldots, x_{-n}, \ldots, x_{-2}, x_{-1})$.

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \ldots, x_n)$ of distinct numbers.

$\rightarrow$ Is it possible to extend this definition to infinite sequences?

There are three types of infinite sequences:

- Right-sided: $(x_1, x_2, \ldots, x_n, \ldots)$.
- Left-sided: $(\ldots, x_{-n}, \ldots, x_{-2}, x_{-1})$.
- Two-sided: $(\ldots, x_{-n}, \ldots, x_{-1}, x_0, x_1, \ldots, x_n, \ldots)$.

Infinite Trees

So far we have only considered finite binary search trees, built from finite sequences $(x_1, \dots, x_n)$ of distinct numbers.

$\longrightarrow$ Is it possible to extend this definition to infinite sequences?

There are three types of infinite sequences:

- Right-sided: $(x_1, x_2, \dots, x_n, \dots)$.
- Left-sided: $(\dots, x_{-n}, \dots, x_{-2}, x_{-1})$.
- Two-sided: $(\dots, x_{-n}, \dots, x_{-1}, x_0, x_1, \dots, x_n, \dots)$.

$\longrightarrow$ For each such type can we define corresponding binary search trees?

Right-Sided Sequences

Right-Sided Sequences

Consider the following infinite sequences:

Right-Sided Sequences

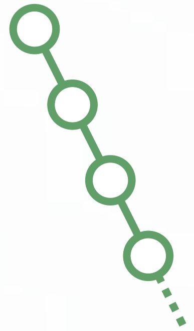
Consider the following infinite sequences:

$$(1, 2, 3, 4, \dots)$$

Right-Sided Sequences

Consider the following infinite sequences:

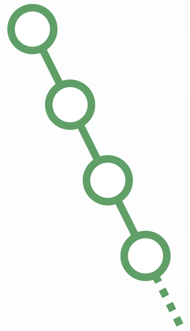
$$(1, 2, 3, 4, \ldots)$$



Right-Sided Sequences

Consider the following infinite sequences:

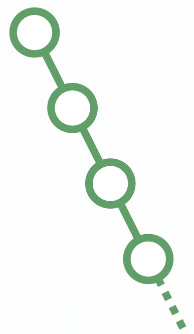
$$(1, 2, 3, 4, \dots) \qquad (2, 1, 4, 3, 6, \dots)$$



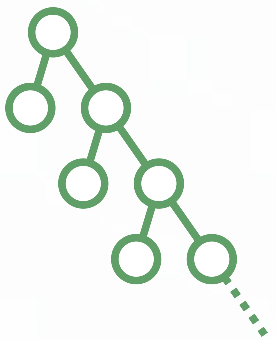
Right-Sided Sequences

Consider the following infinite sequences:

$$(1, 2, 3, 4, \dots)$$



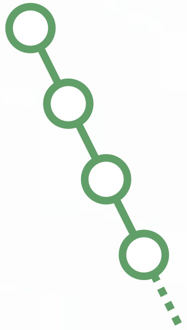
$$(2, 1, 4, 3, 6, \dots)$$



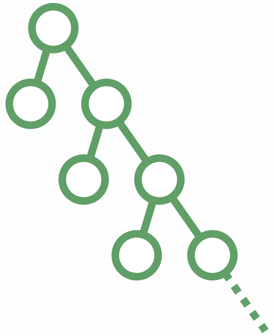
Right-Sided Sequences

Consider the following infinite sequences:

$(1, 2, 3, 4, \ldots)$



$(2, 1, 4, 3, 6, \ldots)$

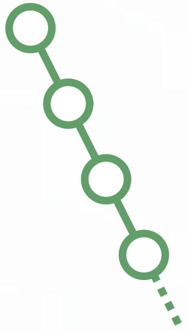


$(U_1, U_2, U_3, \ldots)$

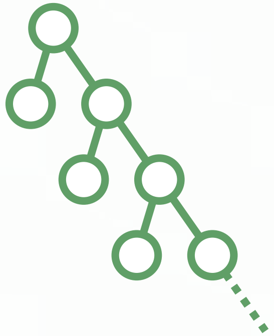
Right-Sided Sequences

Consider the following infinite sequences:

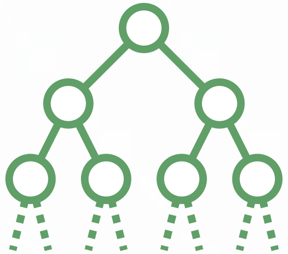
$(1, 2, 3, 4, \ldots)$



$(2, 1, 4, 3, 6, \ldots)$



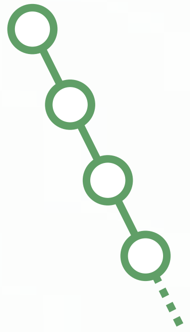
$(U_1, U_2, U_3, \ldots)$



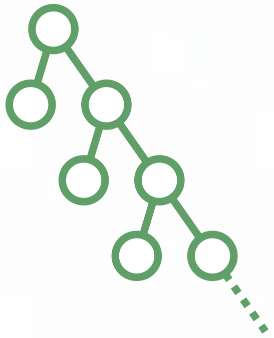
Right-Sided Sequences

Consider the following infinite sequences:

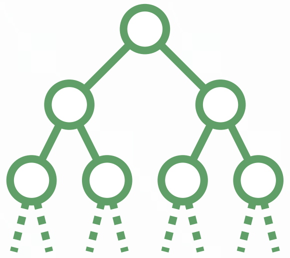
$(1, 2, 3, 4, \dots)$



$(2, 1, 4, 3, 6, \dots)$



$(U_1, U_2, U_3, \dots)$

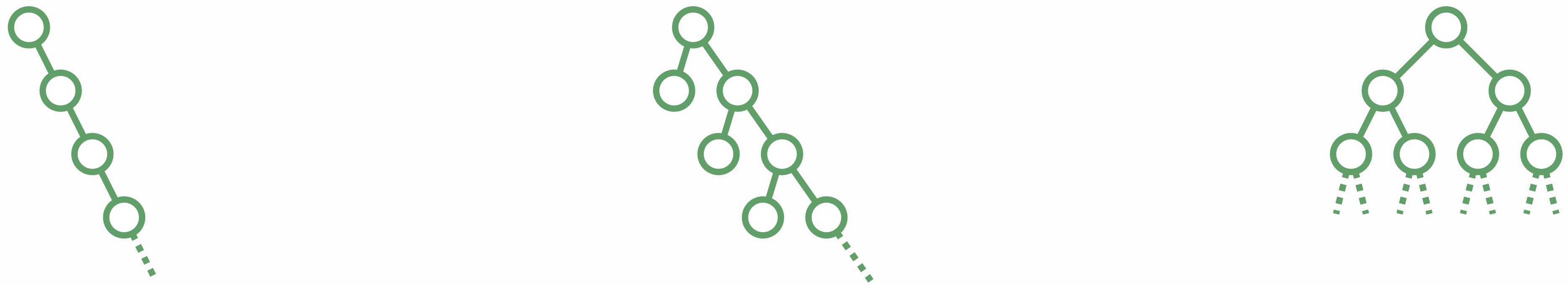


$\rightarrow$ Any right-sided sequence can be placed into an infinite binary search tree.

Right-Sided Sequences

Consider the following infinite sequences:

$(1, 2, 3, 4, \ldots)$ $(2, 1, 4, 3, 6, \ldots)$ $(U_1, U_2, U_3, \ldots)$



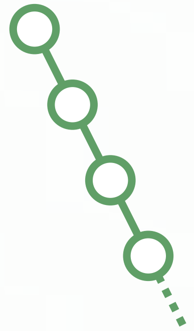
$\rightarrow$ Any right-sided sequence can be placed into an infinite binary search tree.

$\rightarrow$ They create infinite "downward" binary trees, which can be defined by words on $\{0, 1\}$

Right-Sided Sequences

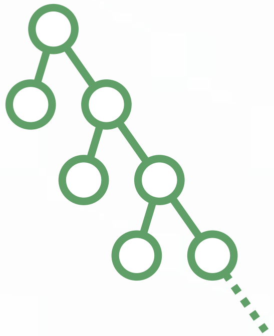
Consider the following infinite sequences:

$(1, 2, 3, 4, \dots)$

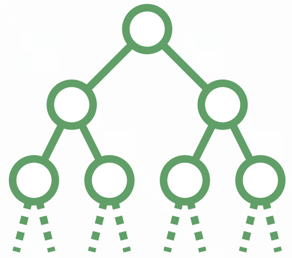


$\{\varnothing, 1, 11, 111, \dots\}$

$(2, 1, 4, 3, 6, \dots)$



$(U_1, U_2, U_3, \dots)$



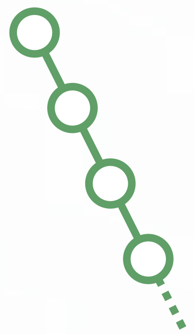
$\longrightarrow$ Any right-sided sequence can be placed into an infinite binary search tree.

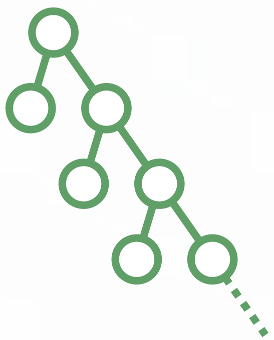
$\longrightarrow$ They create infinite "downward" binary trees, which can be defined by words on $\{0, 1\}$

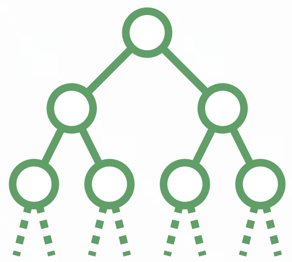
Right-Sided Sequences

Consider the following infinite sequences:

$(1, 2, 3, 4, \dots)$ $\quad$ $(2, 1, 4, 3, 6, \dots)$ $\quad$ $(U_1, U_2, U_3, \dots)$







$\{\varnothing, 1, 11, 111, \dots\}$ $\quad$ $\{\varnothing, 0, 1, 10, 11, 110, \dots\}$

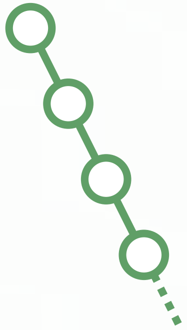
$\rightarrow$ Any right-sided sequence can be placed into an infinite binary search tree.

$\rightarrow$ They create infinite "downward" binary trees, which can be defined by words on $\{0, 1\}$

Right-Sided Sequences

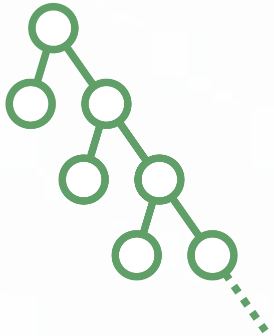
Consider the following infinite sequences:

$(1, 2, 3, 4, \dots)$



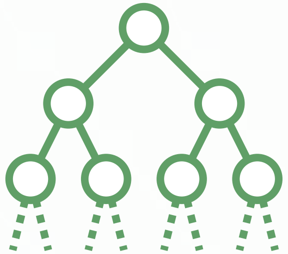
$\{\varnothing, 1, 11, 111, \dots\}$

$(2, 1, 4, 3, 6, \dots)$



$\{\varnothing, 0, 1, 10, 11, 110, \dots\}$

$(U_1, U_2, U_3, \dots)$



$\{\varnothing, 0, 1, 00, 01, 10, 11, 000, \dots\}$

$\rightarrow$ Any right-sided sequence can be placed into an infinite binary search tree.

$\rightarrow$ They create infinite "downward" binary trees, which can be defined by words on $\{0, 1\}$

Left-Sided Sequences

Left-Sided Sequences

Let's skip this one for now...

Two-Sided Sequences

Two-Sided Sequences

Consider the following infinite sequences:

Two-Sided Sequences

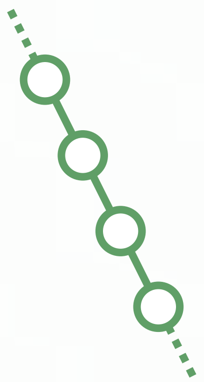
Consider the following infinite sequences:

$$(\ldots, -2, -1, 0, 1, 2, \ldots)$$

Two-Sided Sequences

Consider the following infinite sequences:

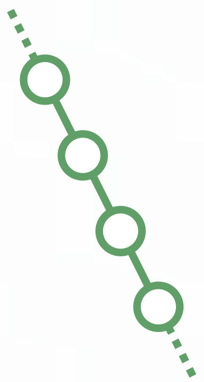
$$(\ldots, -2, -1, 0, 1, 2, \ldots)$$



Two-Sided Sequences

Consider the following infinite sequences:

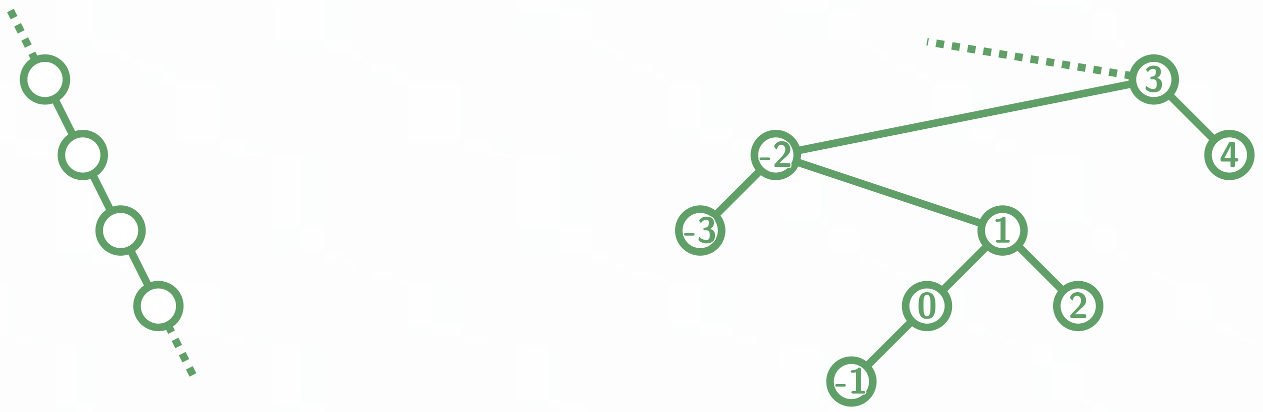
$$(\ldots, -2, -1, 0, 1, 2, \ldots) \qquad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$$



Two-Sided Sequences

Consider the following infinite sequences:

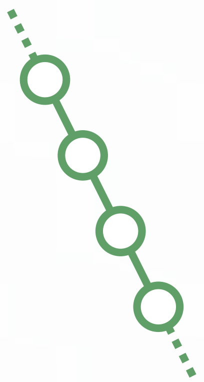
$$(\dots, -2, -1, 0, 1, 2, \dots) \qquad (\dots, 3, -2, 1, 0, -1, 2, -3, \dots)$$

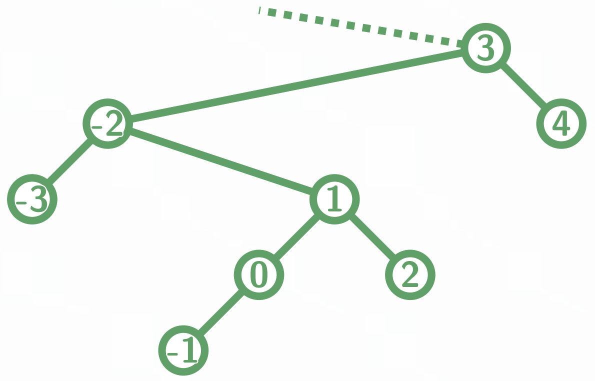


Two-Sided Sequences

Consider the following infinite sequences:

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \qquad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots) \qquad (\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$$

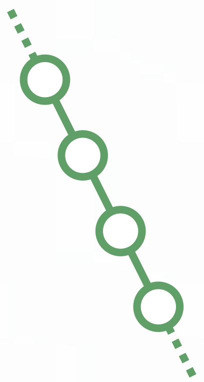


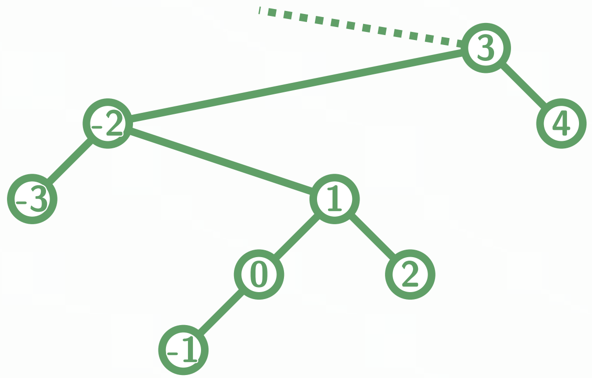


Two-Sided Sequences

Consider the following infinite sequences:

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \quad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots) \quad (\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$$



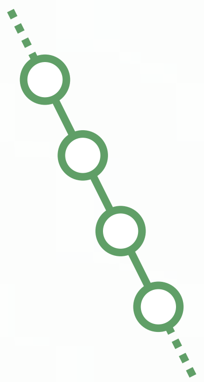


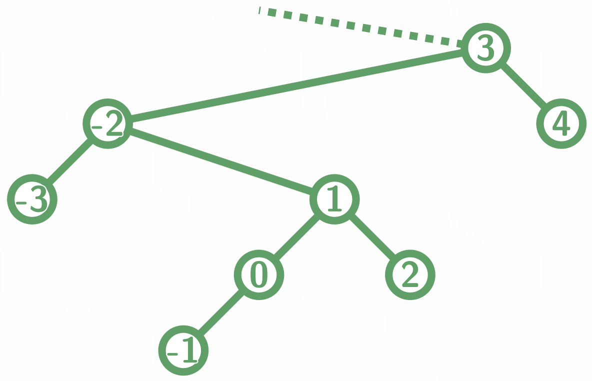
✗

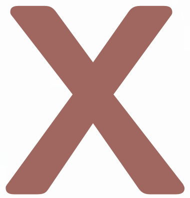
Two-Sided Sequences

Consider the following infinite sequences:

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \quad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots) \quad (\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$$





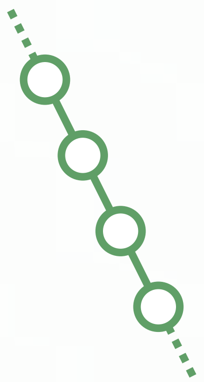


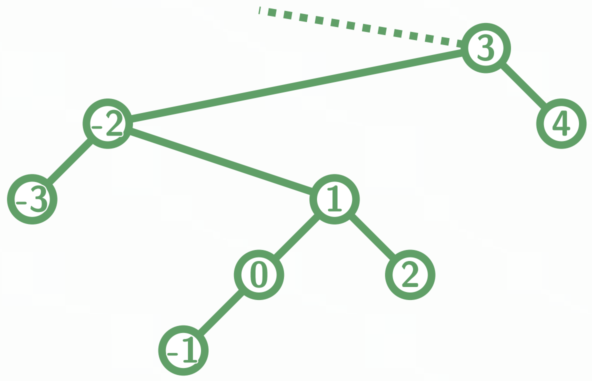
→ Not all two-sided sequence can be placed into an infinite binary search tree.

Two-Sided Sequences

Consider the following infinite sequences:

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \quad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots) \quad (\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$$





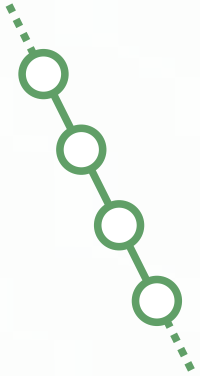
→ Not all two-sided sequence can be placed into an infinite binary search tree.

→ Even when they exist, their definition might be ambiguous...

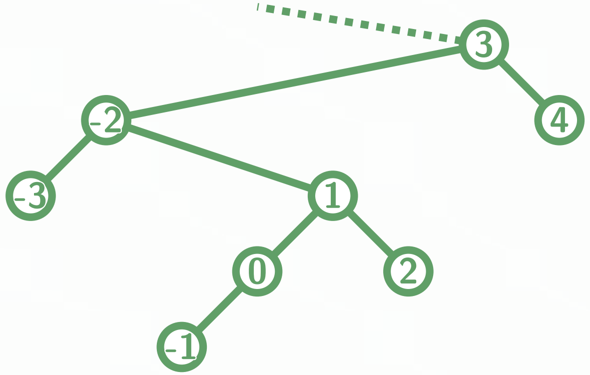
Two-Sided Sequences

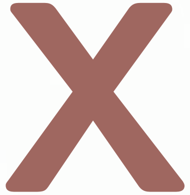
Consider the following infinite sequences:

$$(\dots, -2, -1, 0, 1, 2, \dots) \qquad (\dots, 3, -2, 1, 0, -1, 2, -3, \dots) \qquad (\dots, U_{-1}, U_0, U_1, U_2, \dots)$$



$\{\dots, (11)^{-1}, 1^{-1}, \varnothing, 1, 11, \dots\}$





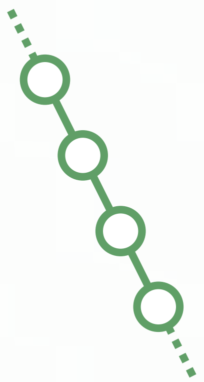
→ Not all two-sided sequence can be placed into an infinite binary search tree.

→ Even when they exist, their definition might be ambiguous...

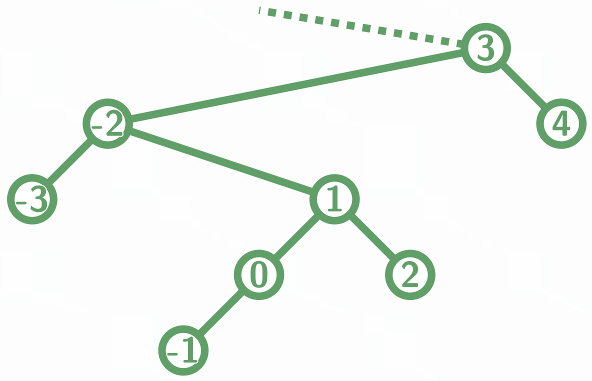
Two-Sided Sequences

Consider the following infinite sequences:

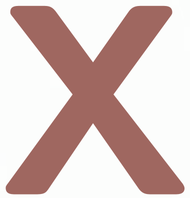
$$(\ldots, -2, -1, 0, 1, 2, \ldots) \quad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots) \quad (\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$$



$\{\ldots, (11)^{-1}, 1^{-1}, \varnothing, 1, 11, \ldots\}$



???



→ Not all two-sided sequence can be placed into an infinite binary search tree.

→ Even when they exist, their definition might be ambiguous...

Two-Sided Sequences

Two-Sided Sequences

Open Questions

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?
- What kind of two-sided sequences admit a binary search tree representation?

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?
- What kind of two-sided sequences admit a binary search tree representation?
- Do these objects have unique and interesting properties?

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?
- What kind of two-sided sequences admit a binary search tree representation?
- Do these objects have unique and interesting properties?

→ The first question has several answers, which might depend on the application.

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?
- What kind of two-sided sequences admit a binary search tree representation?
- Do these objects have unique and interesting properties?

→ The first question has several answers, which might depend on the application.

→ Solving the second question for two-sided or left-sided sequences is equivalent.

Two-Sided Sequences

Open Questions

- What is a unified standard representation for infinite binary search tree?
- What kind of two-sided sequences admit a binary search tree representation?
- Do these objects have unique and interesting properties?

→ The first question has several answers, which might depend on the application.
→ Solving the second question for two-sided or left-sided sequences is equivalent.
→ The last question is the most important and tends to ignore left-sided sequences.

Two-Sided Sequences: A Special Case

Two-Sided Sequences: A Special Case

These extensions of binary search trees appeared in my own research[1].

Two-Sided Sequences: A Special Case

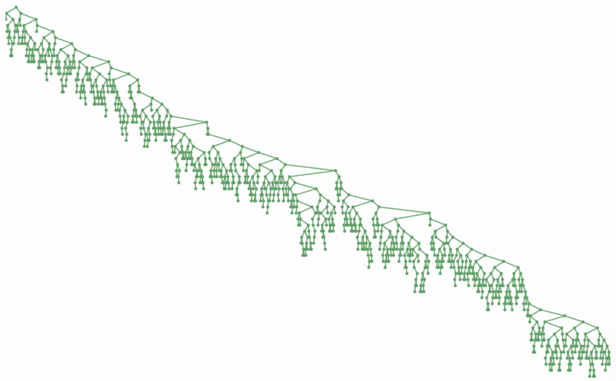
These extensions of binary search trees appeared in my own research[1].

- I was studying a specific model of trees, called *Mallows trees*.

Two-Sided Sequences: A Special Case

These extensions of binary search trees appeared in my own research[1].

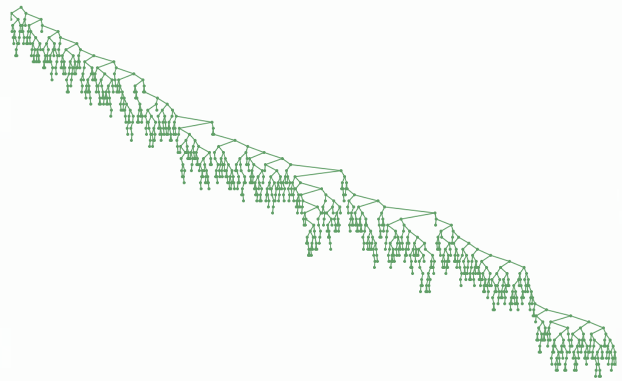
- I was studying a specific model of trees, called *Mallows trees*.



Two-Sided Sequences: A Special Case

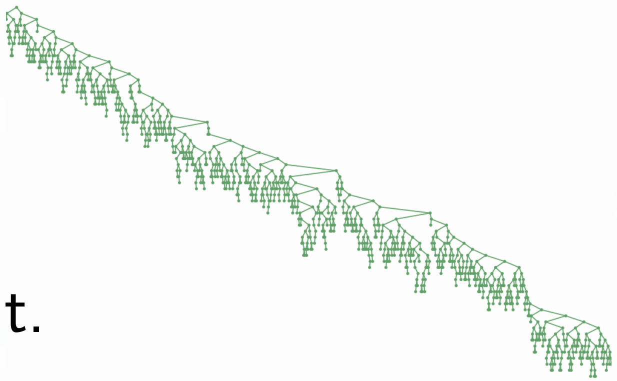
These extensions of binary search trees appeared in my own research[1].

- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.



Two-Sided Sequences: A Special Case

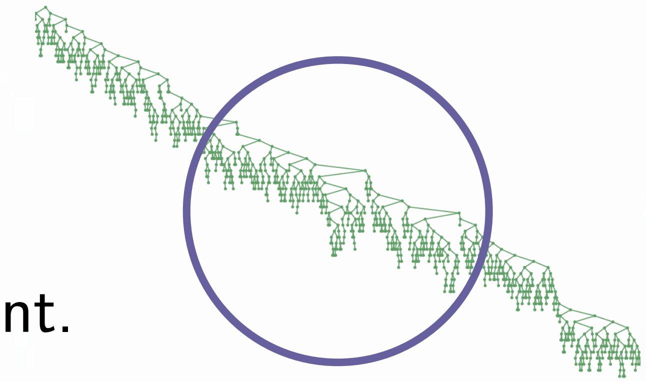
These extensions of binary search trees appeared in my own research[1].



- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.

Two-Sided Sequences: A Special Case

These extensions of binary search trees appeared in my own research[1].

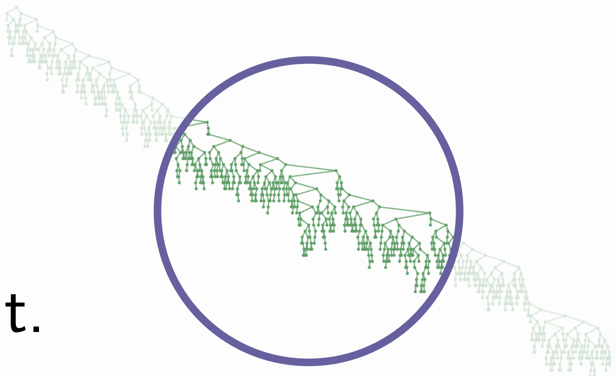


- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.

Two-Sided Sequences: A Special Case

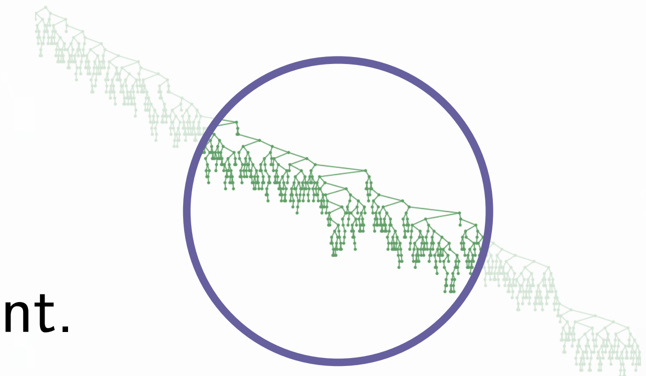
These extensions of binary search trees appeared in my own research[1].

- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.



Two-Sided Sequences: A Special Case

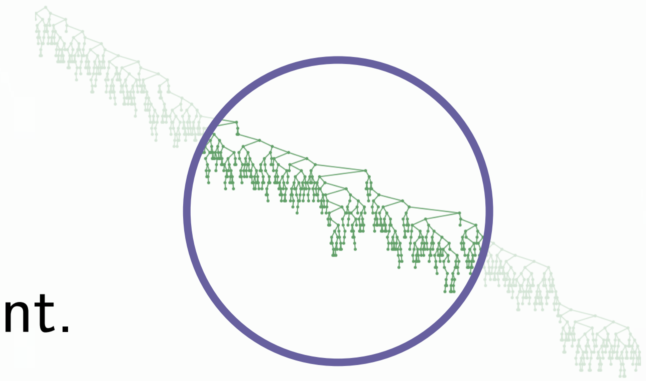
These extensions of binary search trees appeared in my own research[1].



- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree “looked like” from a random point.
- These trees could apparently be constructed as follows.

Two-Sided Sequences: A Special Case

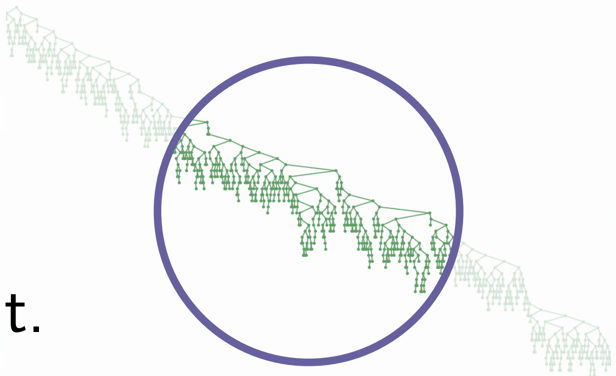
These extensions of binary search trees appeared in my own research[1].



- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree “looked like” from a random point.
- These trees could apparently be constructed as follows.
    - Extend Mallows permutations to bijections of $\mathbb{Z}$ (two-sided Mallows permutation)[3].

Two-Sided Sequences: A Special Case

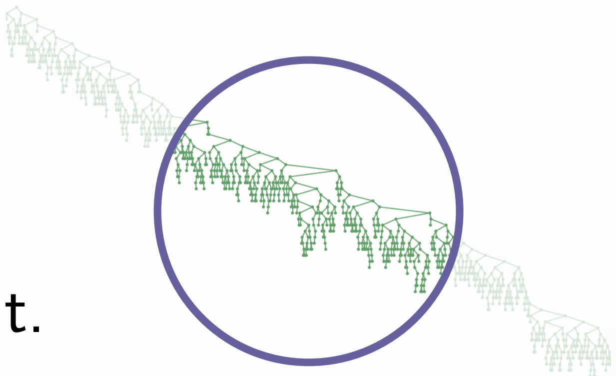
These extensions of binary search trees appeared in my own research[1].



- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.
- These trees could apparently be constructed as follows.
  - Extend Mallows permutations to bijections of $\mathbb{Z}$ (two-sided Mallows permutation)[3].
  - See this as a two-sided sequence.

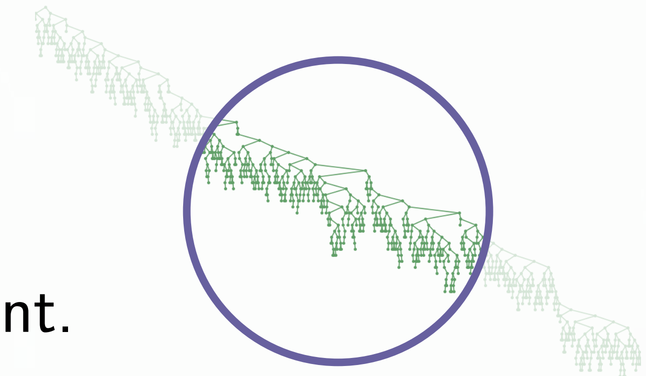
Two-Sided Sequences: A Special Case

These extensions of binary search trees appeared in my own research[1].



- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.
- These trees could apparently be constructed as follows.
  - Extend Mallows permutations to bijections of $\mathbb{Z}$ (two-sided Mallows permutation)[3].
  - See this as a two-sided sequence.
  - Insert this sequence into a binary search tree.

Two-Sided Sequences: A Special Case



These extensions of binary search trees appeared in my own research[1].

- I was studying a specific model of trees, called *Mallows trees*.
- These are the binary search trees of (finite) Mallows permutations.
- I was curious about what the tree "looked like" from a random point.
- These trees could apparently be constructed as follows.
  - Extend Mallows permutations to bijections of $\mathbb{Z}$ (two-sided Mallows permutation)[3].
  - See this as a two-sided sequence.
  - Insert this sequence into a binary search tree.

→ Why does it work here?

Two-Sided Sequences: A Special Case

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

$(\ldots, -2, -1, 0, 1, 2, \ldots)$ $(\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$ $(\ldots, U_{-1}, U_0, U_1, U_2, \ldots)$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.

$$(\dots, -2, -1, 0, 1, 2, \dots) \quad (\dots, 3, -2, 1, 0, -1, 2, -3, \dots) \quad (\dots, U_{-1}, U_0, U_1, U_2, \dots)$$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \qquad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.
  - I believe it is possible to define the binary search tree of any two-sided permutation.

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \qquad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.
  - I believe it is possible to define the binary search tree of any two-sided permutation.
- They are "well-ordered"

$$(\ldots, -2, -1, 0, 1, 2, \ldots) \qquad (\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.
  - I believe it is possible to define the binary search tree of any two-sided permutation.
- They are "well-ordered": for any $n \in \mathbb{Z}$, there is k_-, k_+ such that

$$\sup\left\{\sigma(k) : k \leq k_-\right\} \leq n \leq \inf\left\{\sigma(k) : k \geq k_+\right\}.$$

$(\ldots, -2, -1, 0, 1, 2, \ldots)$ $(\ldots, 3, -2, 1, 0, -1, 2, -3, \ldots)$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.
  - I believe it is possible to define the binary search tree of any two-sided permutation.
- They are "well-ordered": for any $n \in \mathbb{Z}$, there is k_-, k_+ such that

$$\sup \left\{ \sigma(k) : k \leq k_- \right\} \leq n \leq \inf \left\{ \sigma(k) : k \geq k_+ \right\}.$$

  - This removes the case of "fluctuating" permutations.

$(\ldots, -2, -1, 0, 1, 2, \ldots)$

Two-Sided Sequences: A Special Case

Two-sided Mallows permutations happen to have a few properties simplifying the definition of their corresponding binary search trees.

- They are permutations σ from $\mathbb{Z}$ to $\mathbb{Z}$.
  - This removes the case of uniform random variables.
  - I believe it is possible to define the binary search tree of any two-sided permutation.
- They are "well-ordered": for any $n \in \mathbb{Z}$, there is k_-, k_+ such that

$$\sup\left\{\sigma(k) : k \leq k_-\right\} \leq n \leq \inf\left\{\sigma(k) : k \geq k_+\right\}.$$

  - This removes the case of "fluctuating" permutations.
  - The tree grows up leftward, thus we only need to define 1^{-1}.

$(\ldots, -2, -1, 0, 1, 2, \ldots)$

References

1. Corsini, B. (2024). **Limits of Mallows trees**. *Electronic Journal of Probability, 29, 1-44.*.
2. Devroye, L. (1986). **A note on the height of binary search trees**. *Journal of the ACM (JACM), 33(3), 489-498*.
3. Gnedin, A., & Olshanski, G. (2012). **The two-sided infinite extension of the Mallows model for random permutations**. *Advances in Applied Mathematics, 48(5), 615-639*.
4. Robson, J. M. (1979). **The height of binary search trees**. *Australian Computer Journal, 11(4), 151-153*.

Thank you!

Thank you!

Thank you!

Thank you!

Thank you!

Thank you!