Singly

Linked Lists

Objectives

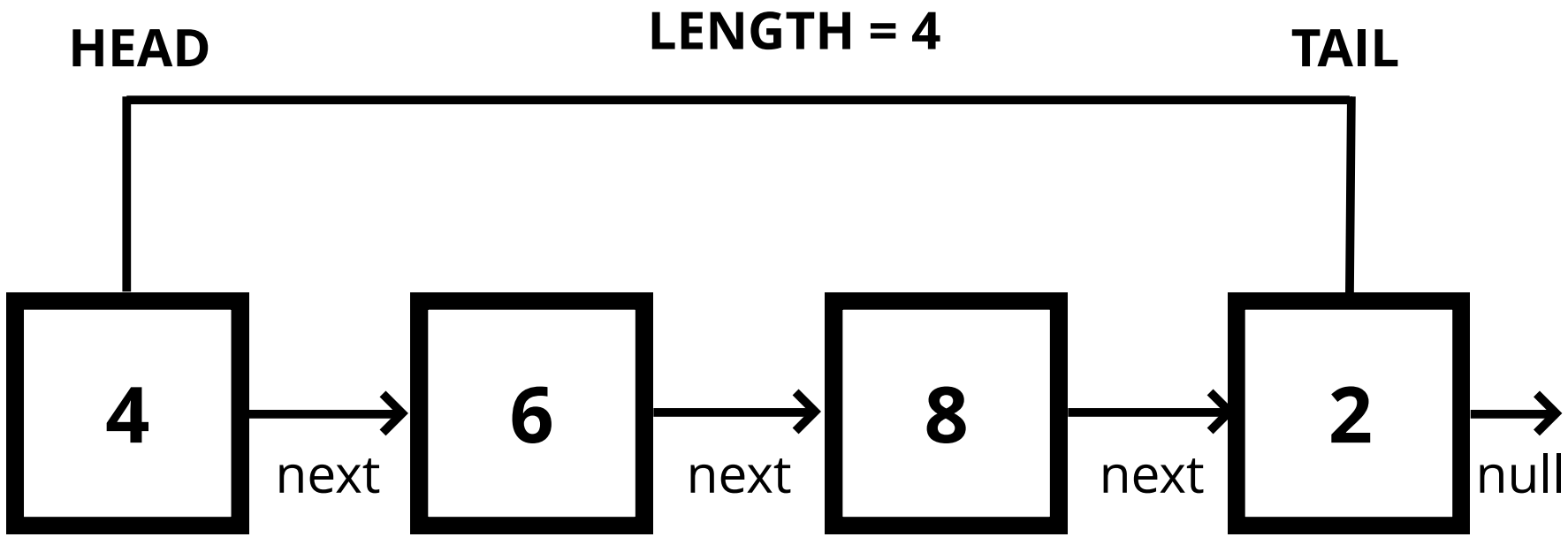
- Define what a Singly Linked List is
- Compare and contrast Linked Lists with Arrays
- Implement insertion, removal and traversal methods on Singly Linked Lists

What is a linked list?

A data structure that contains a **head**, **tail** and **length** property.

Linked Lists consist of nodes, and each **node** has a **value** and a **pointer** to another node or null

Singly Linked Lists



Comparisons with Arrays

Lists

- Do not have indexes!
- Connected via nodes with a **next** pointer
- Random access is not allowed

Arrays

- Indexed in order!
- Insertion and deletion can be expensive
- Can quickly be accessed at a specific index

Pushing

Adding a new **node** to the end of the Linked List!

Pushing pseudocode

- This function should accept a value
- Create a new node using the value passed to the function
- If there is no head property on the list, set the head and tail to be the newly created node
- Otherwise set the next property on the tail to be the new node and set the tail property on the list to be the newly created node
- Increment the length by one
- Return the linked list

Let's visualize this!

YOUR TURN

Popping

Removing a **node** from the end
of the Linked List!

Popping pseudocode

- If there are no nodes in the list, return undefined
- Loop through the list until you reach the tail
- Set the next property of the 2nd to last node to be null
- Set the tail to be the 2nd to last node
- Decrement the length of the list by 1
- Return the value of the node removed

Let's visualize this!

YOUR TURN

Shifting

Removing a new **node** from the beginning of the Linked List!

Shifting pseudocode

- If there are no nodes, return undefined
- Store the current head property in a variable
- Set the head property to be the current head's next property
- Decrement the length by 1
- Return the value of the node removed

Let's visualize this!

YOUR TURN

Unshifting

Adding a new **node** to the beginning of the Linked List!

Unshifting pseudocode

- This function should accept a value
- Create a new node using the value passed to the function
- If there is no head property on the list, set the head and tail to be the newly created node
- Otherwise set the newly created node's next property to be the current head property on the list
- Set the head property on the list to be that newly created node
- Increment the length of the list by 1
- Return the linked list

Let's visualize this!

YOUR TURN

Get

Retrieving a **node** by it's position
in the Linked List!

Get pseudocode

- This function should accept an index
- If the index is less than zero or greater than or equal to the length of the list, return null
- Loop through the list until you reach the index and return the node at that specific index

Let's visualize this!

YOUR TURN

Set

Changing the **value** of a node based on it's position in the Linked List

Set pseudocode

- This function should accept a value and an index
- Use your **get** function to find the specific node.
- If the node is not found, return false
- If the node is found, set the value of that node to be the value passed to the function and return true

Let's visualize this!

YOUR TURN

Insert

Adding a node to the Linked List
at a **specific** position

Insert pseudocode

- If the index is less than zero or greater than the length, return false
- If the index is the same as the length, push a new node to the end of the list
- If the index is 0, unshift a new node to the start of the list
- Otherwise, using the **get** method, access the node at the index - 1
- Set the next property on that node to be the new node
- Set the next property on the new node to be the previous next
- Increment the length
- Return true

Let's visualize this!

YOUR TURN

Remove

Removing a node from the
Linked List at a **specific** position

Remove pseudocode

- If the index is less than zero or greater than the length, return undefined
- If the index is the same as the length-1, pop
- If the index is 0, shift
- Otherwise, using the **get** method, access the node at the index - 1
- Set the next property on that node to be the next of the next node
- Decrement the length
- Return the value of the node removed

Let's visualize this!

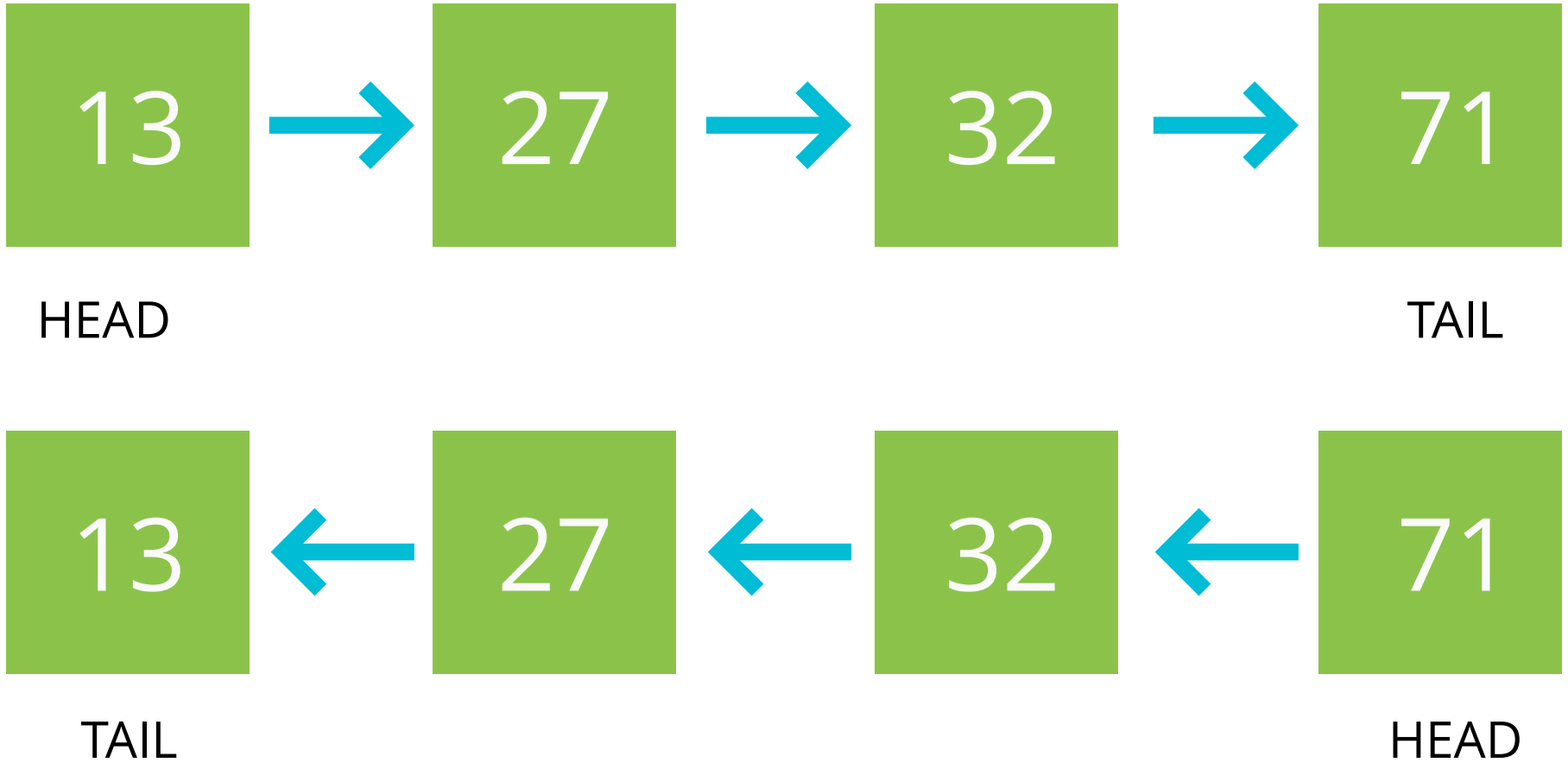
YOUR TURN

REVERSE

Reversing the Linked List

in place!

REVERSING A SINGLY LINKED LIST



Reverse pseudocode

- Swap the head and tail
- Create a variable called next
- Create a variable called prev
- Create a variable called node and initialize it to the head property
- Loop through the list
- Set next to be the next property on whatever node is
- Set the next property on the node to be whatever prev is
- Set prev to be the value of the node variable
- Set the node variable to be the value of the next variable
- Once you have finished looping, return the list

Let's visualize this!

YOUR TURN

Big O
of Singly Linked Lists

Insertion - **O(1)**

Removal - **It depends.... O(1) or O(N)**

Searching - **O(N)**

Access - **O(N)**

RECAP

- Singly Linked Lists are an excellent alternative to arrays when insertion and deletion at the beginning are frequently required
- Arrays contain a built in index whereas Linked Lists do not
- The idea of a list data structure that consists of nodes is the foundation for other data structures like Stacks and Queues