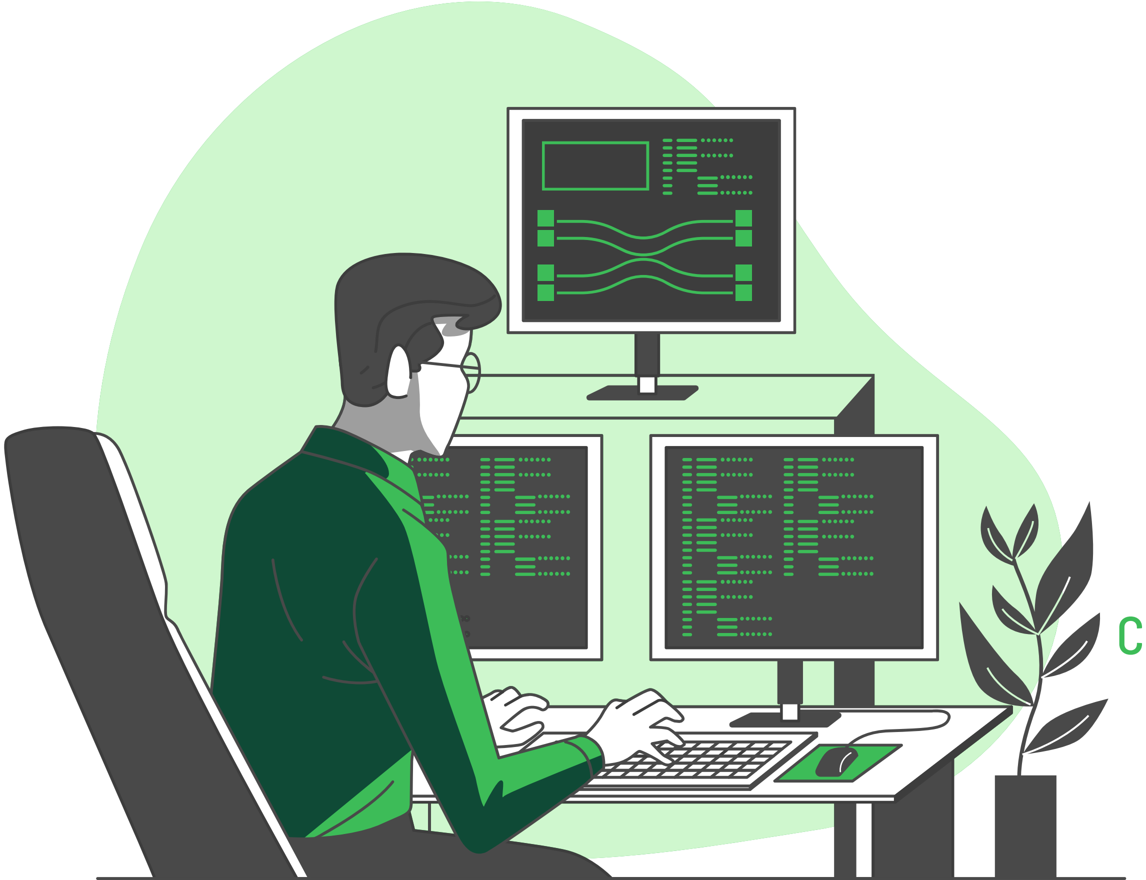


Stack and Queue

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents

01

Abstract Data Types

Abstraction of a data structure

02

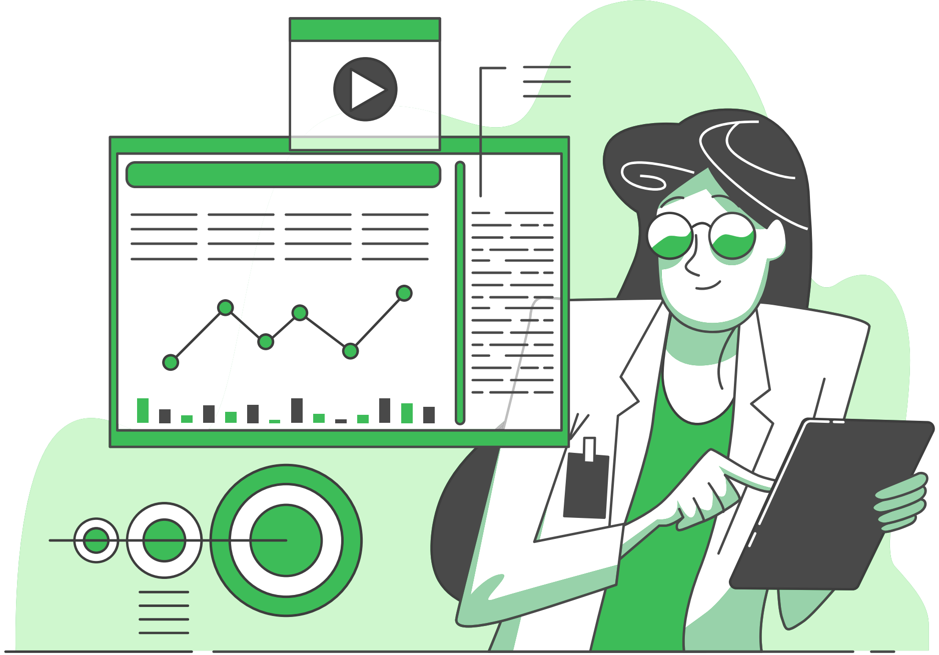
Stack

Last In, First Out (LIFO)

03

Queue

First In, First Out (FIFO)



01

Abstract Data Types

Abstraction of a data structure

Abstract Data Type (ADT)

Specifies the **data**, **operations**, and **error conditions** of a data structure.

Implementation details not that important for an abstract data type.

The more we narrow our definition, the more important the implementation details matter.

Real life example: Application Programming Interface (API).

Example: Bags

Bags are data structures that store any kind of data.

Application programming interface (API):

- **add(x:item):** Inserts an item into the bag.
- **isempty( ):** Checks whether the bag is empty or not.
- **size( ):** Returns the number of items in the bag.
- **(iterate):** A mechanism to iterate over the items in the bag.

Do we care about item ordering? **No**.
De we care about removing an item? **No**.

How do we implement this data structure?

02

Stack

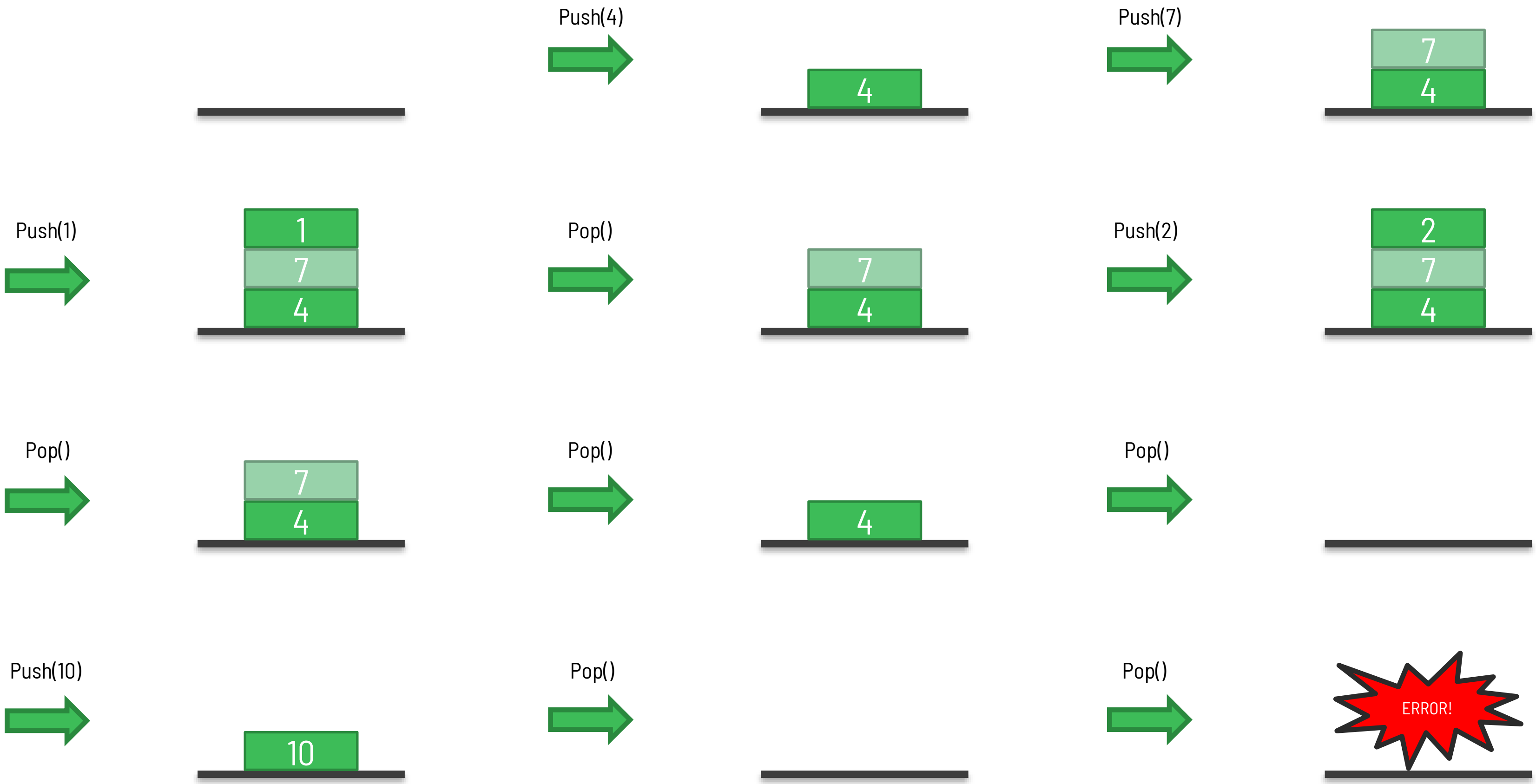
Last In, First Out (LIFO)

Stack: Last-In, First-Out

- **push(x:item):** Inserts an item at the top of the stack.
- **pop():** Removes the item at the top of the stack and returns it.
- **isempty():** Checks whether the stack is empty or not.
- **size():** Returns the number of items in the stack.
- **peek():** Returns the item at the top of the stack without removing it.

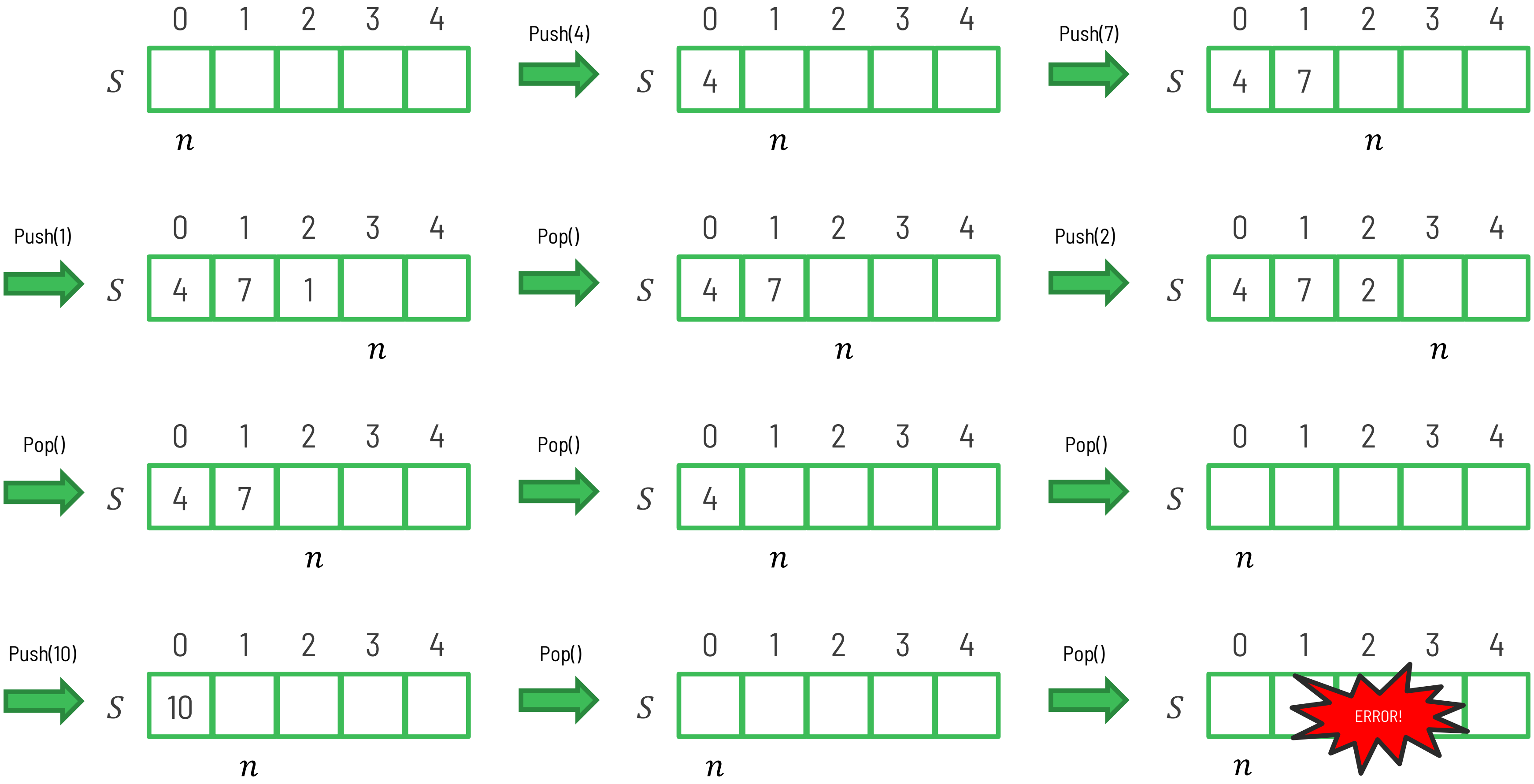
Consider the following operations:

Push(4), Push(7), Push(1), Pop(), Push(2), Pop(), Pop(), Pop(), Push(10), Pop(), Pop()



How do we implement a stack?

Array implementation: Let n be the size of the stack. Consider the following operations:
Push(4), Push(7), Push(1), Pop(), Push(2), Pop(), Pop(), Pop(), Push(10), Pop(), Pop()



	0	1	2	3	4
S					
	n				

Push(4)

	0	1	2	3	4
S	4				
		n			

Push(7)

	0	1	2	3	4
S	4	7			
			n		

Push(1)

	0	1	2	3	4
S	4	7	1		
				n	

Pop()

	0	1	2	3	4
S	4	7			
			n		

Push(2)

	0	1	2	3	4
S	4	7	2		
				n	

Pop()

	0	1	2	3	4
S	4	7			
			n		

Pop()

	0	1	2	3	4
S	4				
		n			

Pop()

	0	1	2	3	4
S					
	n				

Push(10)

	0	1	2	3	4
S	10				
		n			

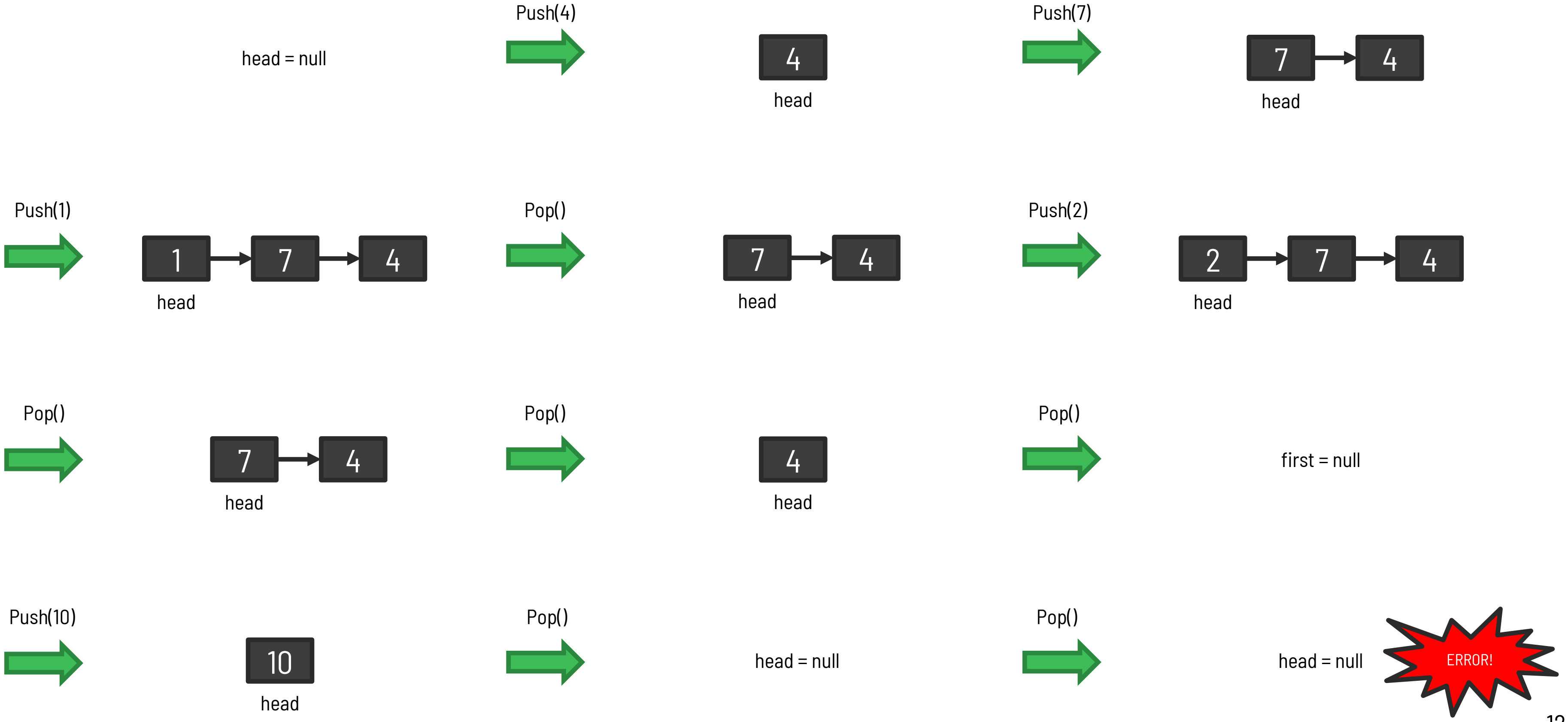
Pop()

	0	1	2	3	4
S					
	n				

Pop()

	0	1	2	3	4
S					
	n				

Singly Linked List implementation: Consider the following operations:
Push(4), Push(7), Push(1), Pop(), Push(2), Pop(), Pop(), Pop(), Push(10), Pop(), Pop()



Stack Runtime Complexities

Implementation using an array:

- Push: Insertion at the next available location. Then, Push $\in O(1)$ amortized
- Pop: Remove the item at index $n - 1$, where n is the size of the stack. Then, Pop $\in \theta(1)$
- Peek: Return the item at index $n - 1$, where n is the size of the stack. Then, Peek $\in \theta(1)$

Implementation using a singly linked list:

- Push: Insertion at the front of the list. Then, Push $\in \theta(1)$
- Pop: Deletion from the front of the list. Then, Pop $\in \theta(1)$
- Peek: Return the front of the list. Then, Peek $\in \theta(1)$

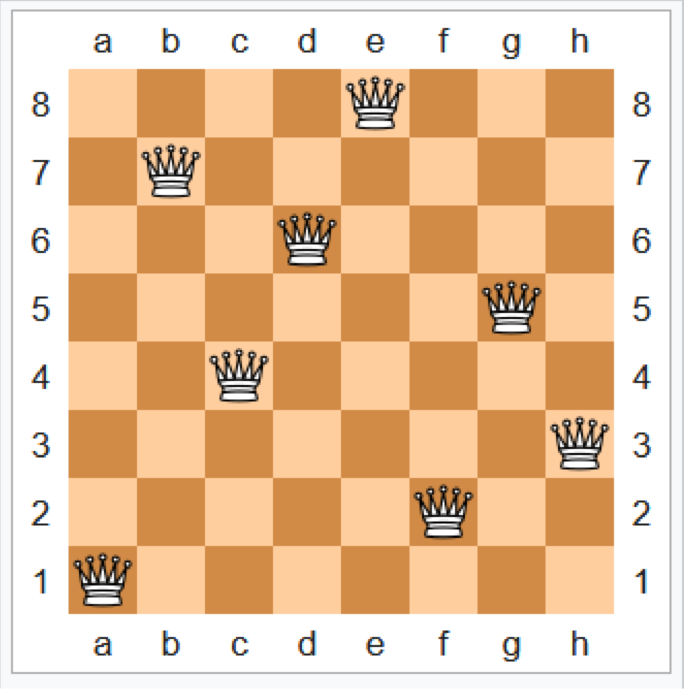
Dijkstra’s Expression Evaluation Algorithm

How do we evaluate `(8*((7+3)-((4+2)*(3-1))))`?

```
Input: A mathematical expression E
Output: The evaluation value of the expression

let S1 and S2 be empty stacks

for each character c in E do
   if c is an operand then
      push c into S1
   else if c is an operator then
      push c into S2
   else if c is a right parenthesis then
      pop an operator op from S2
      pop the requisite number of operands from S1
      calculate r by applying op to the operands
      push r into S1
   end if
end for

return the last value in S1
```

Edsger W. Dijkstra may be most famous for his Shortest Path algorithm, but his contributions to math and CS go beyond such an algorithm.

Example: Eight Queen Puzzle



Place N queens in a $N \times N$ chessboard so that no two queens threaten each other.

Solutions exist for all natural numbers N except for $N = 2$ and $N = 3$.

Algorithm paradigm: **Backtracking**

03

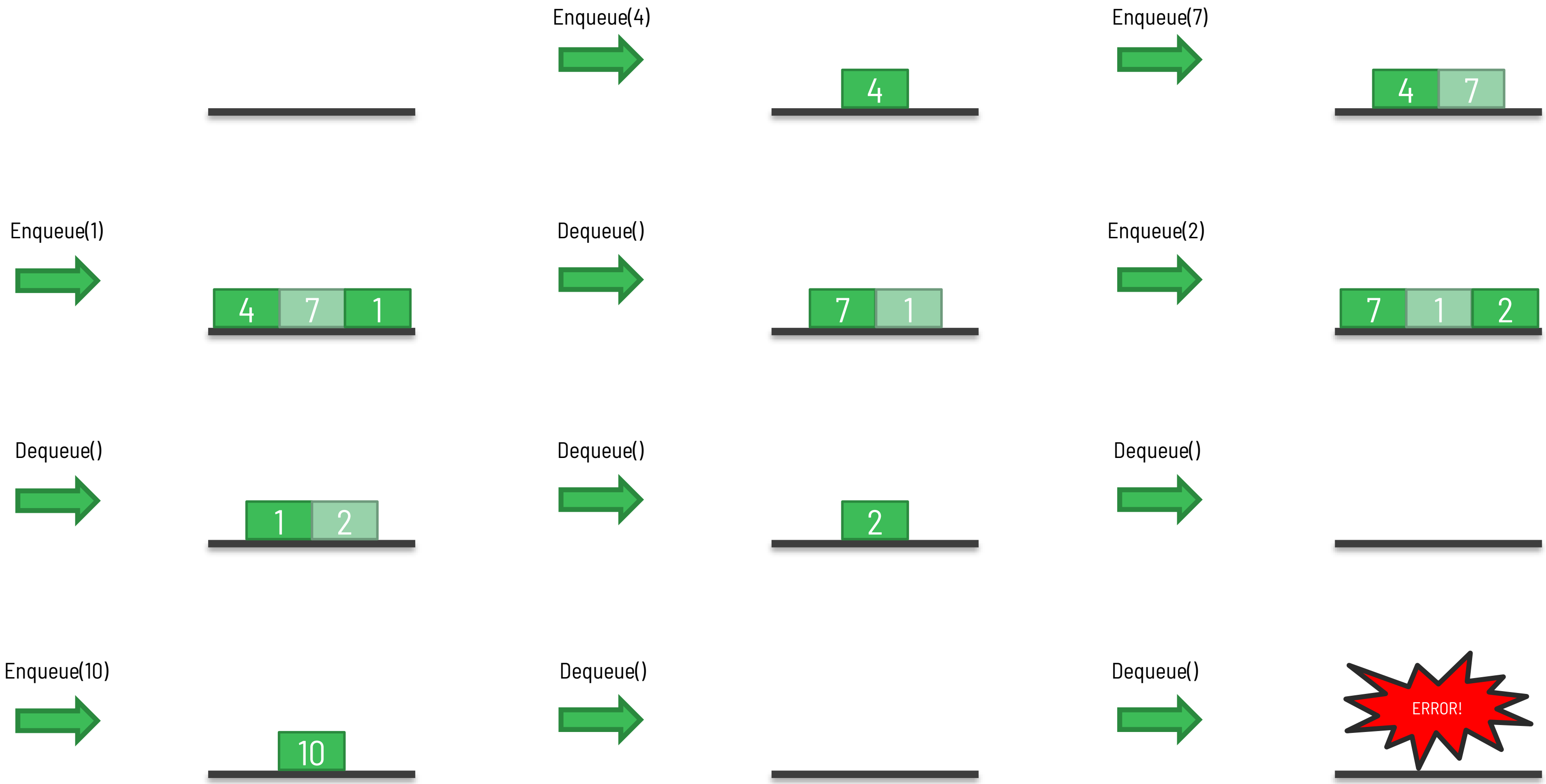
Queue

First In, First Out (FIFO)

Queue: First-In, First-Out

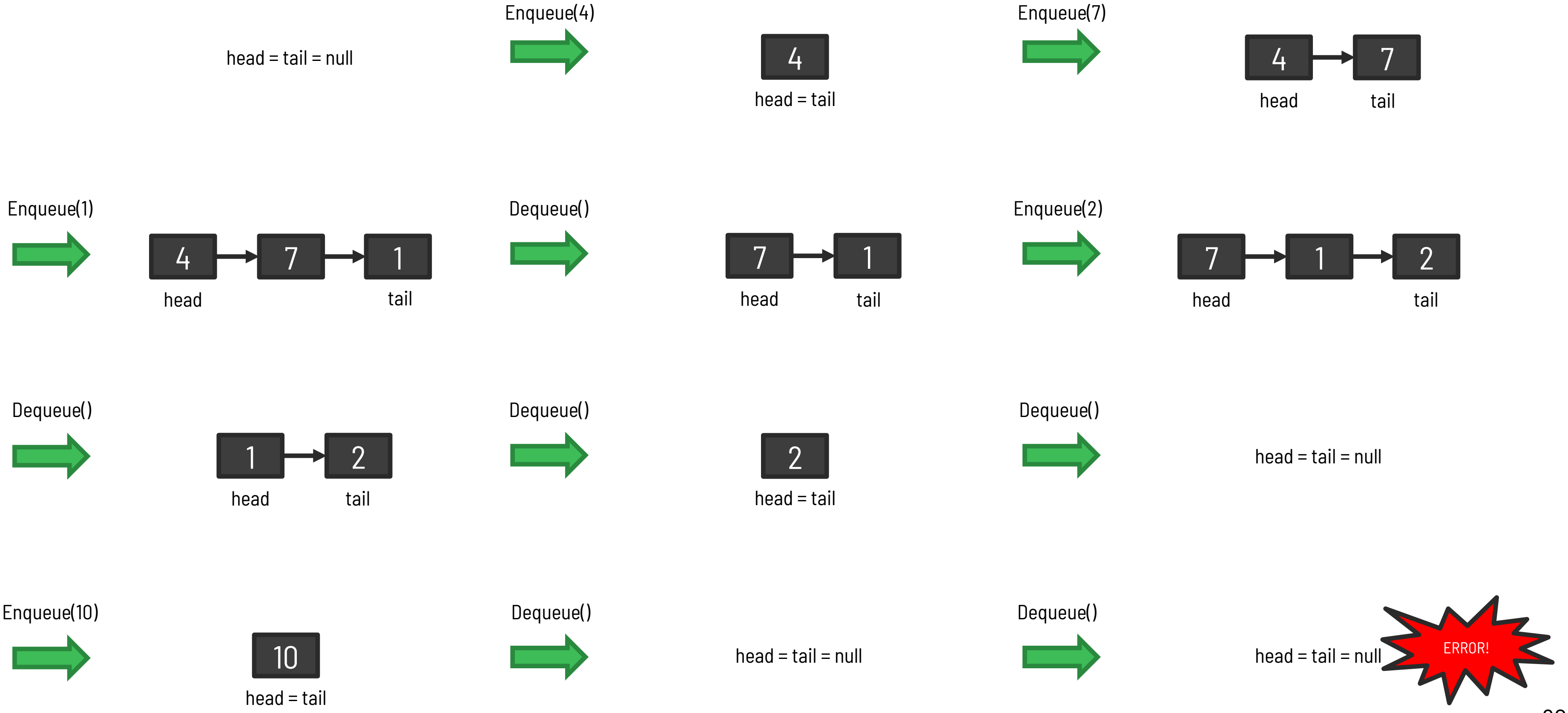
- **enqueue(x:item):** Inserts an item at the end of the queue.
- **dequeue( ):** Removes the item at the front of the queue and returns it.
- **isempty( ):** Checks whether the queue is empty or not.
- **size( ):** Returns the number of items in the queue.
- **peek( ):** Returns the item at the front of the queue without removing it.

Consider the following operations: Enqueue(4), Enqueue(7), Enqueue(1), Dequeue(), Enqueue(2), Dequeue(), Dequeue(), Dequeue(), Queue(10), Dequeue(), Dequeue()



How do we implement a queue?

Singly Linked List implementation: Consider the following operations: Enqueue(4), Enqueue(7), Enqueue(1), Dequeue(), Enqueue(2), Dequeue(), Dequeue(), Dequeue(), Queue(10), Dequeue(), Dequeue()



head = tail = null

Enqueue(4)

4
head = tail

Enqueue(7)

4 7
head tail

Enqueue(1)

4 7 1
head tail

Dequeue()

7 1
head tail

Enqueue(2)

7 1 2
head tail

Dequeue()

1 2
head tail

Dequeue()

2
head = tail

Dequeue()

head = tail = null

Enqueue(10)

10
head = tail

Dequeue()

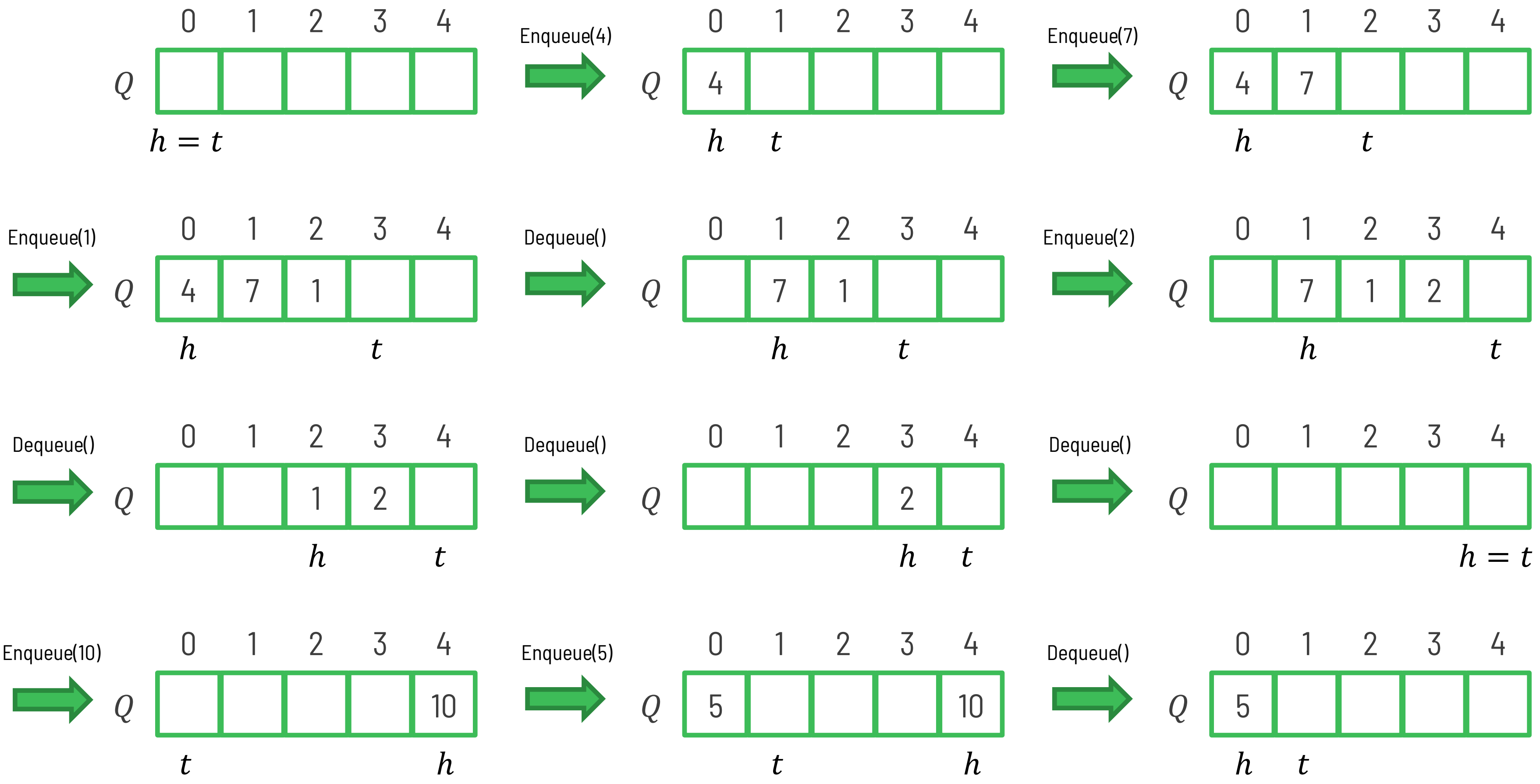
head = tail = null

Dequeue()

head = tail = null

ERROR!

Array implementation: Consider the following operations: Enqueue(4), Enqueue(7), Enqueue(1), Dequeue(), Enqueue(2), Dequeue(), Dequeue(), Dequeue(), Enqueue(10), Enqueue(5), Dequeue()



	0	1	2	3	4
Q					
	$h = t$				

Enqueue(4)

	0	1	2	3	4
Q	4				
	h	t			

Enqueue(7)

	0	1	2	3	4
Q	4	7			
	h		t		

Enqueue(1)

	0	1	2	3	4
Q	4	7	1		
	h			t	

Dequeue()

	0	1	2	3	4
Q		7	1		
		h		t	

Enqueue(2)

	0	1	2	3	4
Q		7	1	2	
		h			t

Dequeue()

	0	1	2	3	4
Q			1	2	
			h		t

Dequeue()

	0	1	2	3	4
Q				2	
				h	t

Dequeue()

	0	1	2	3	4
Q					
					$h = t$

Enqueue(10)

	0	1	2	3	4
Q					10
	t				h

Enqueue(5)

	0	1	2	3	4
Q	5				10
		t			h

Dequeue()

	0	1	2	3	4
Q	5				
	h	t			

Queue Runtime Complexities

Implementation using a circular array:

- Enqueue: Insertion at index t. Then, Enqueue $\in O(1)$ amortized
- Dequeue: Remove the item at index h. Then, Dequeue $\in \theta(1)$
- Peek: Return the item at index h. Then, Peek $\in \theta(1)$

Implementation using a singly linked list that tracks its tail:

- Enqueue: Insertion at the back of the list. Then, Enqueue $\in \theta(1)$
- Dequeue: Deletion from the front of the list. Then, Dequeue $\in \theta(1)$
- Peek: Return the front of the list. Then, Peek $\in \theta(1)$

Queueing Theory

Queueing theory is the mathematical study of waiting lines, or queues.

A queueing model is constructed so that **queue lengths** and **waiting time** can be predicted.

Queueing theory is generally considered a branch of **operations research** because the results are often used when making business decisions about the resources needed to provide a service.

- Wikipedia

SlideOverflowException

Do you have any questions?

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories