Stack & Queue

Duc Vo & Spencer Meren

Stack & Queue

_ Stack and Queue are both data structure, with minor difference:

+ Stack is Last in First out (LIFO)
+ Queue is First in First out (FIFO)

_ They are both very easy to comprehend, but very powerful in solving problems, as well as building data

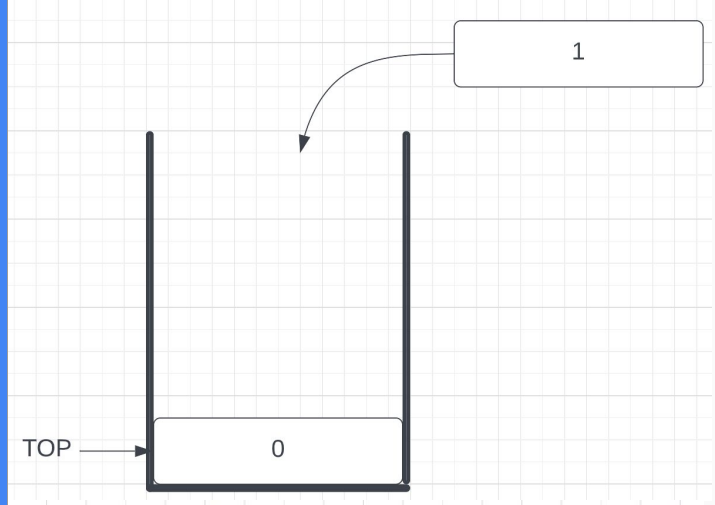
Stack

_ Stack is FILO, first in last out

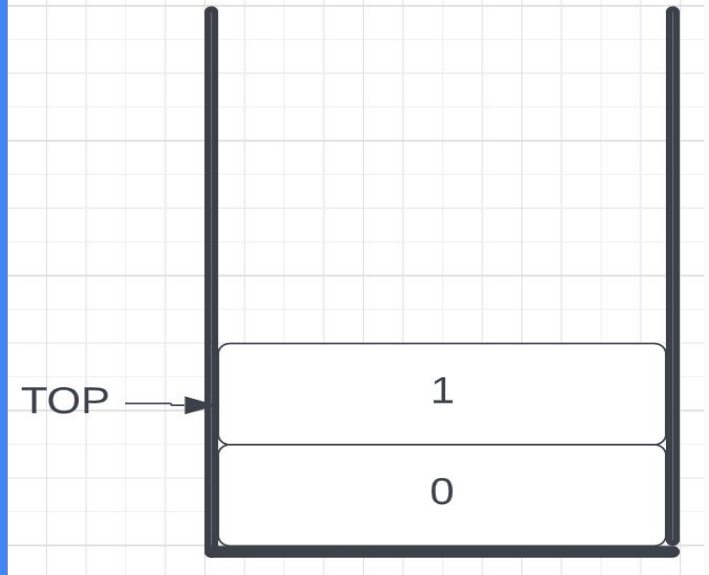
_ Can be interpreted as a bucket

_ Keep track of the top element

_ Entered element will push down the top element and become the top of the stack



_ The top pointer points to the most recently added item

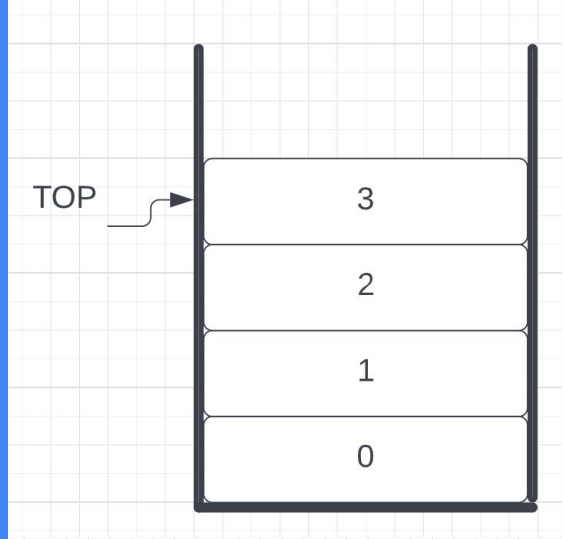


_ When another item join, it is pushed onto the stack, and the top pointer would point to the newly pushed item

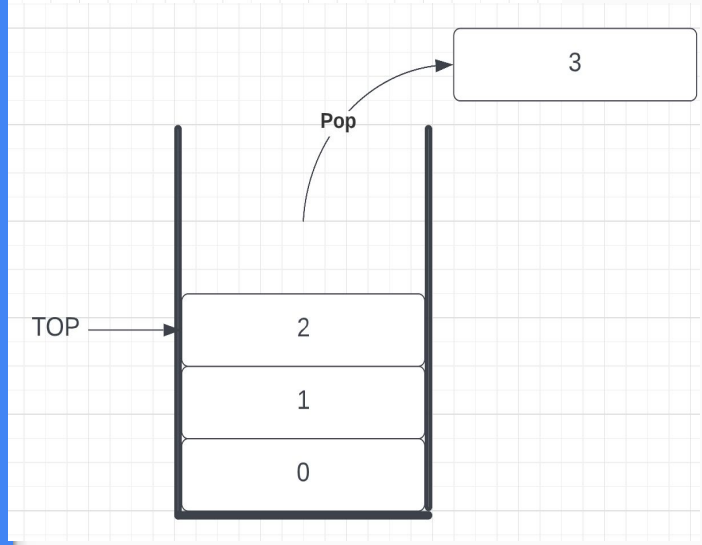
Stack (cont.)

_ Stack is usually used for history log, or the ability to undo the most recent activity

_ It is also implemented inside almost every computer, since a stack for memory is one of the most basic necessities for a processor to operate



_ Stack only has the access to the first item on top of the stack



_ When the stack is popped, the most recently added item will be popped out of the stack, and the top pointer pointed to the second recently added item

Queue

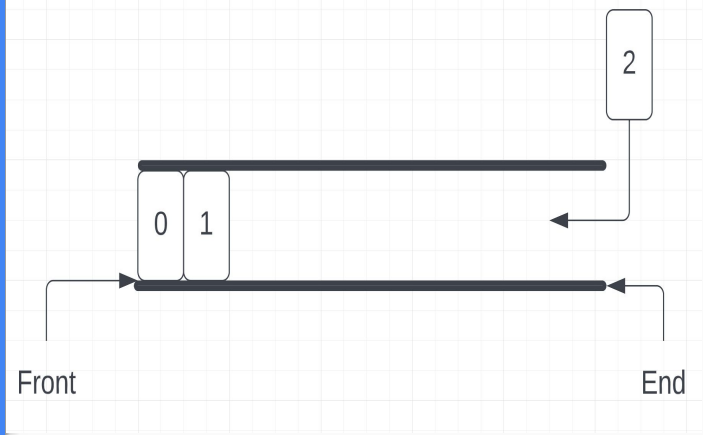
_ Queue is FIFO, first in first out

_ Can be interpreted as a line

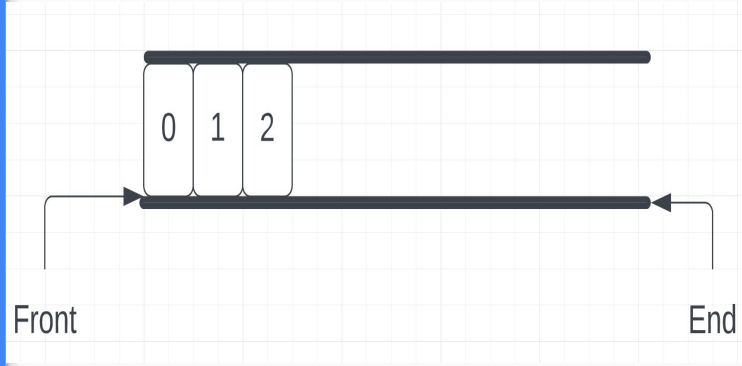
_ Keep track of the front and the end of the queue

_ Entered element will entry from the end and exit at the front

_ Element can only be pushed at the end and popped at the front



_ Items would make the way from the back to the front as the queue process

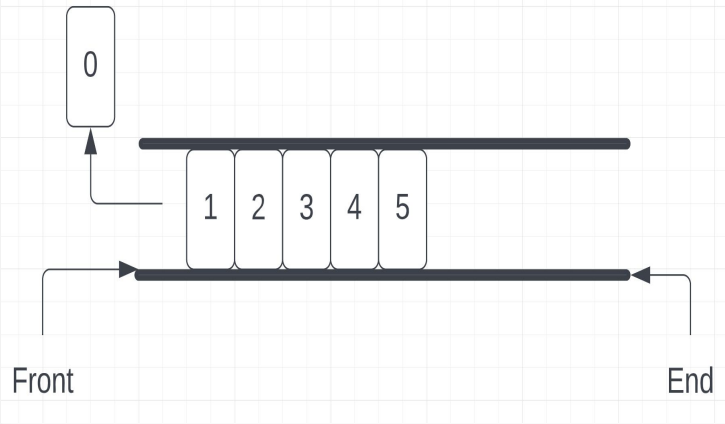


_ When another item join, it is pushed onto the end of the queue, making no changes to the front.

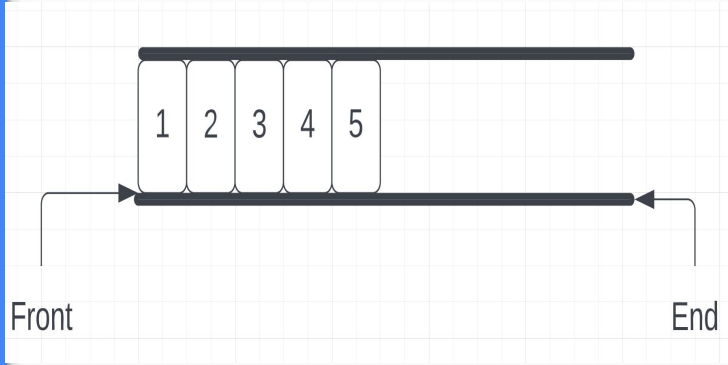
Queue (cont.)

_ Queue is used for planning tasks, scheduling operations that needs to carry out

_ Another version of queue is high-priority queue where item can be added from the front, and will be prioritized to be popped before the lower ones



_The item in the front will be popped first.



_ This will move the queue forward, to the second entered item, so on and so forth

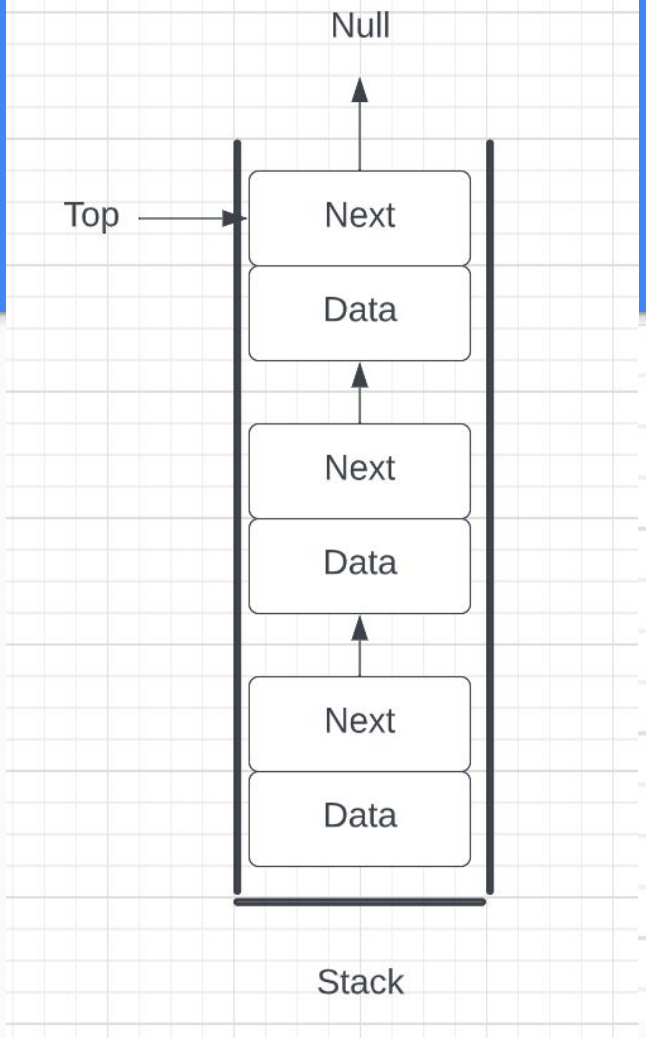
Complexity

bigocheetsheet.com

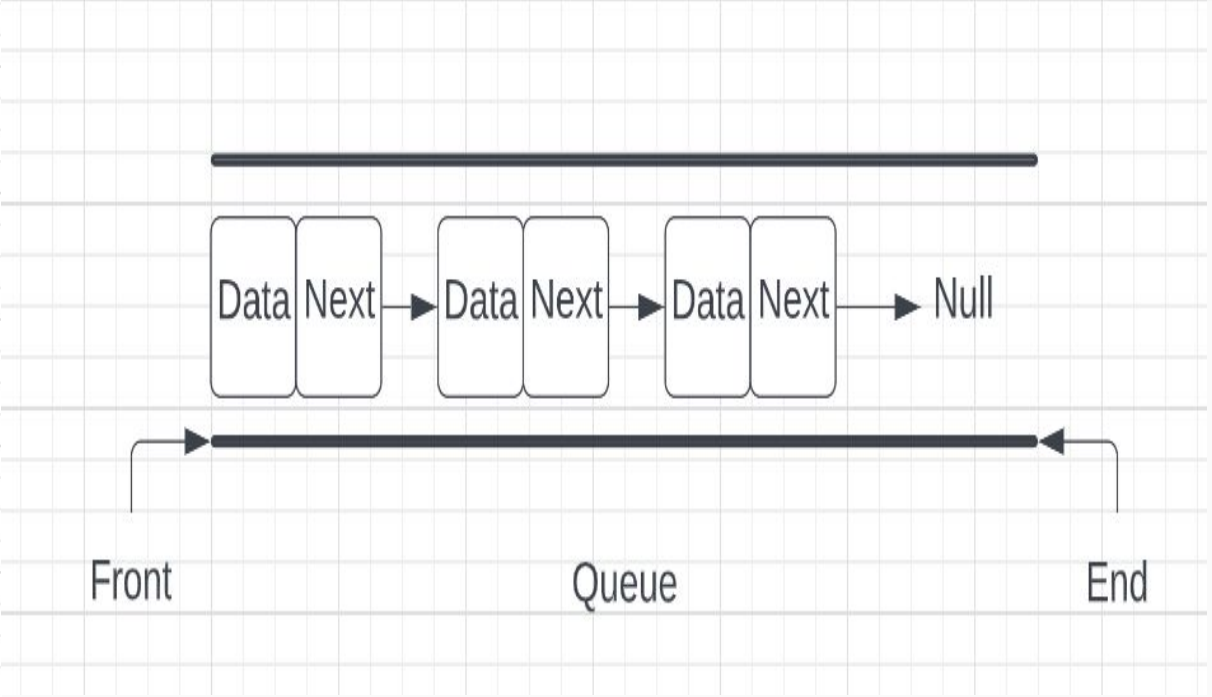
Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

_ Since the stack can only access the element at the top, or the queue can only access the element at the front, their access and search complexity is not the best, they are both O(n)

_ But what makes them powerful is their insertion and deletion, because of the same reasons, the insertion and deletion (or push and pop) is very fast, at O(1)



Singly-linked list can easily be used to implement both the stack and queue.

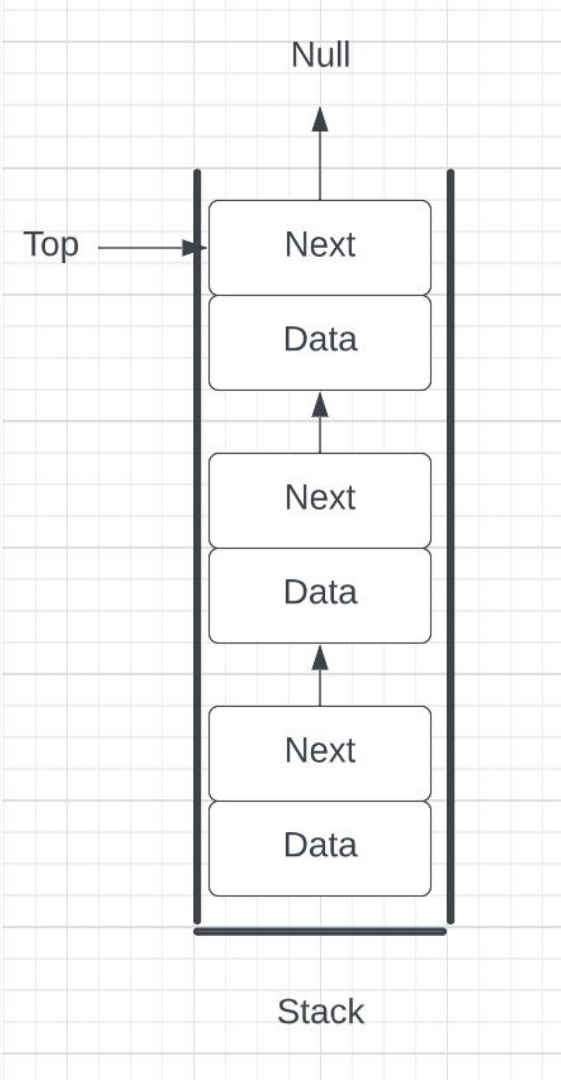


Stack Implementation

_ Keep track of the top pointer.

_ When push, create a node to hold the data and let it point to the current top and let the current top point the the newly created node (this would also work when the stack is empty since the top should point to null)

_ When pop, push the top pointer to the next node pointed by the top.

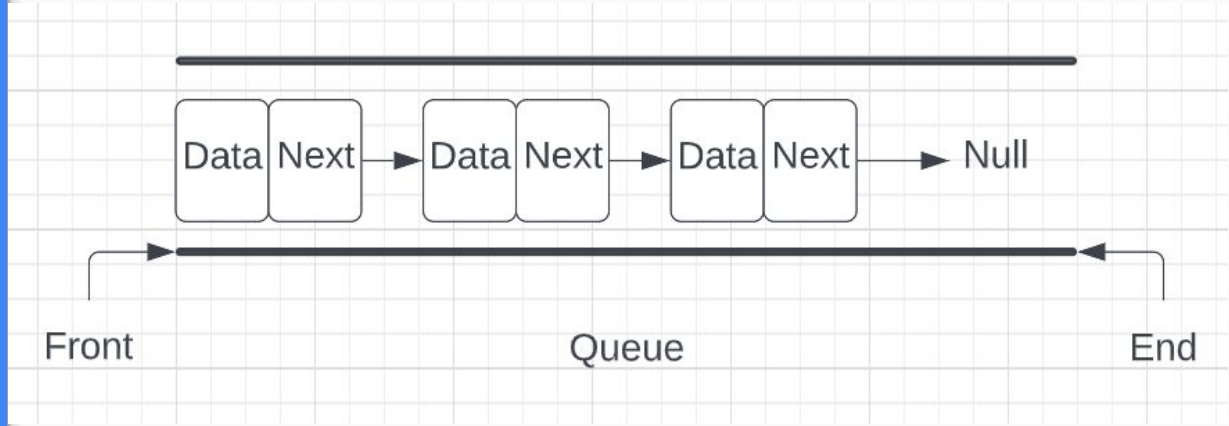


Queue Implementation

_ Keep a pointer to the front and the end

_ When push, create a node to hold the data and let the current node at the end point to it. Then update the end pointer to point to the added node (if this is the first element added to the node, point both the front and the end to it)

_ When pop, push the top pointer to the next node pointed by the top. This is the same as the stack



Work Cited

“Know Thy Complexities!” Big, https://www.bigocheatsheet.com/.