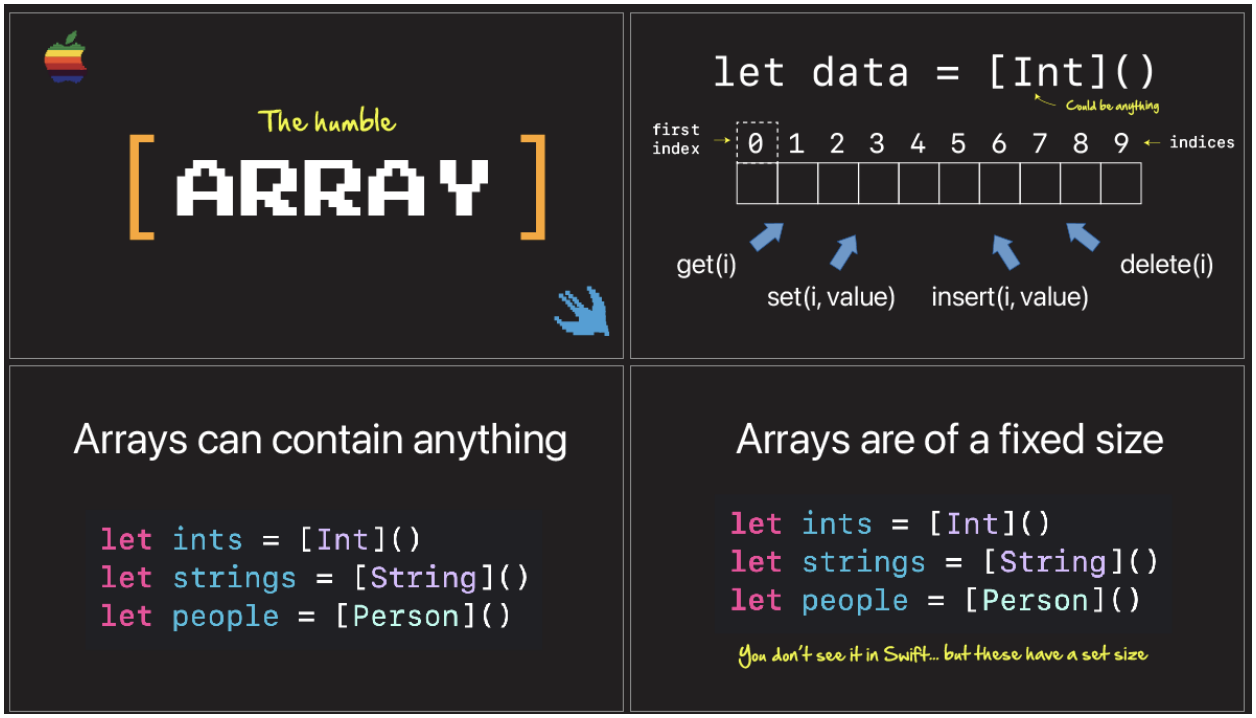


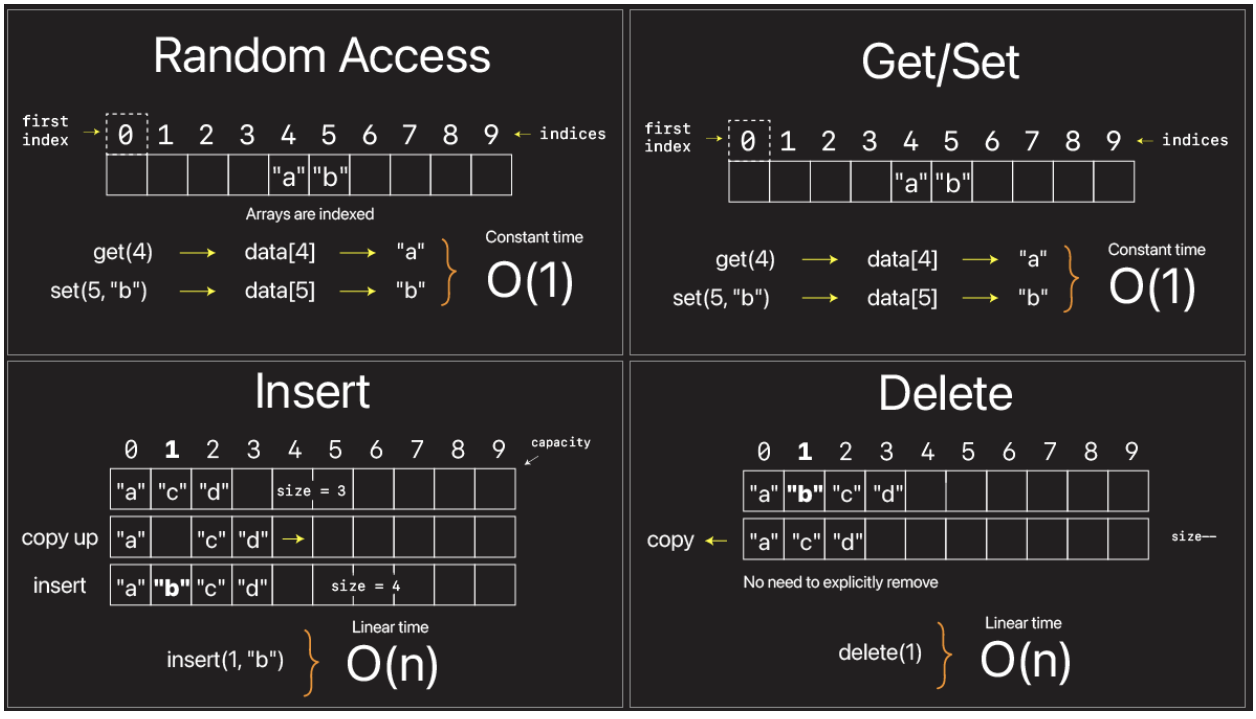
Arrays can contain anything

```
let ints = [Int]()
let strings = [String]()
let people = [Person]()
```

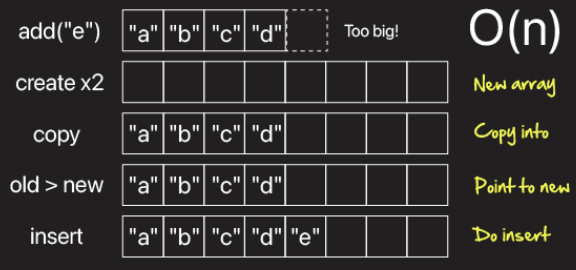
Arrays are of a fixed size

```
let ints = [Int]()
let strings = [String]()
let people = [Person]()
```

You don't see it in Swift... but these have a set size



Growing the Size of an Array



O(n)

New array

Copy into

Point to new

Do insert

https://developer.apple.com/documentation/swift/array

What's different about Swift?

```
var array = ["a", "c", "d"]
array.insert("b", at: 1)
array.remove(at: 1)
```

All this heavy lifting is done for you...

For the interview

Fixed size

Random access - O(1)

Insert / Delete - O(n)

Arrays can shrink and grow - O(n)

Swift arrays handle heavy lifting for you

x3 Things to Know

Arrays can contain anything

Arrays are of a fixed size

Arrays support random access

Tips for solving

Start Brute Force

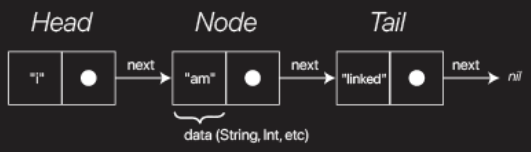
Use Paper

Handle edge cases

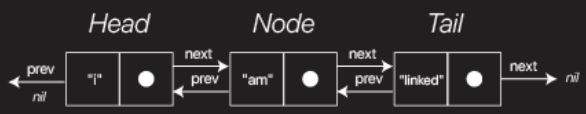
Optimize after

LINKED LIST

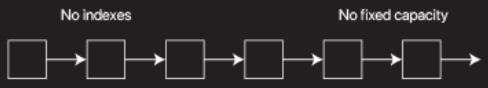
What is a linked list?



Double linked list?



How different than an array?



Disadvantage

No random access

Get/set is linear O(n)

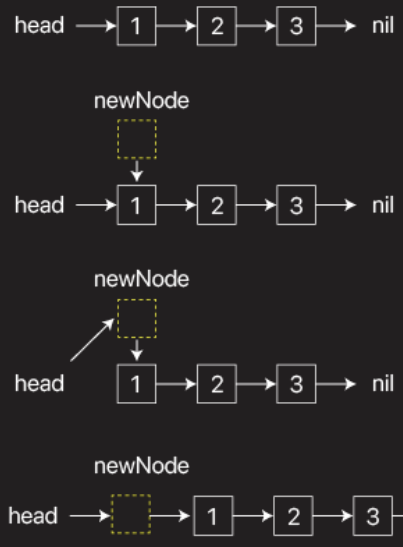
Advantage

Super fast Insert/Delete at front O(1)

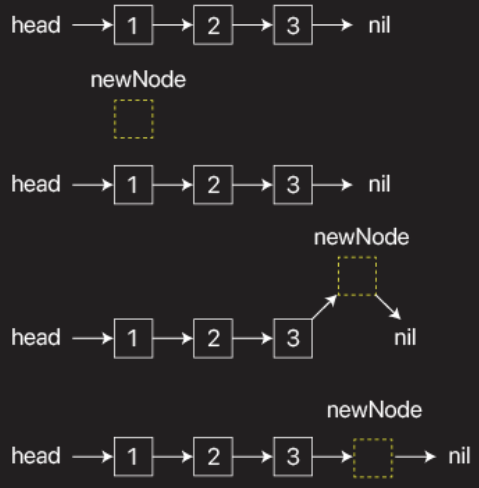
Can grow incrementally

Can be used in Stacks and Queues

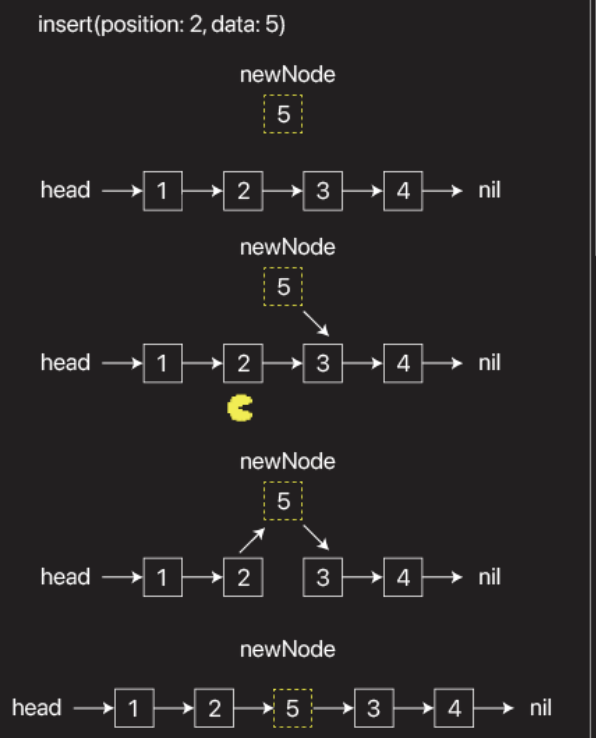
Add front - O(1)



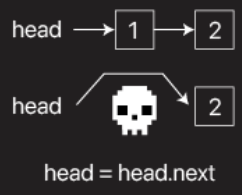
Add back- O(n)



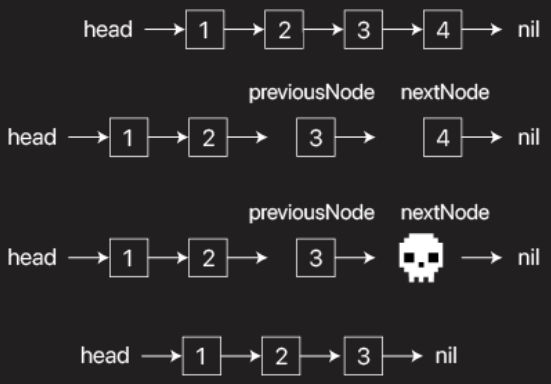
Insert Position - O(n)



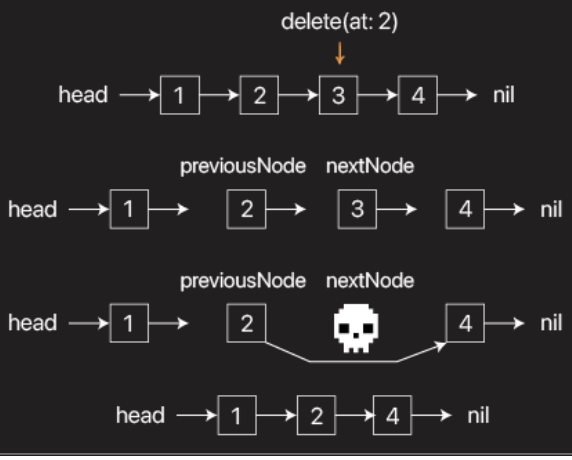
Delete first - O(1)



Delete last - O(n)



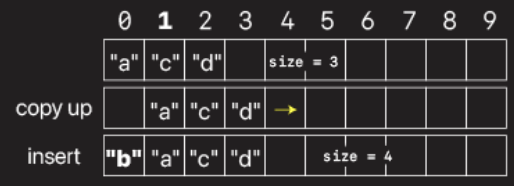
Delete at position - O(n)



What you need to know

- Anything to do with the front is O(1)
  - addFront/getFirst/deleteFirst
- Anytime you need to walk O(n)
  - addBack/getBack/deleteLast
- No random access
- Always the right size

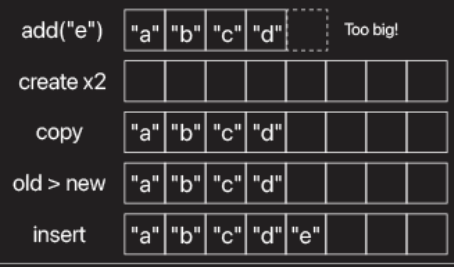
Arrays are slow inserting front

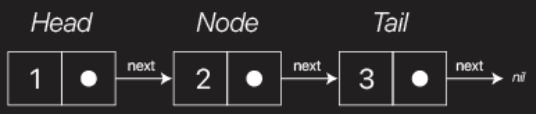


insert(0, "b") } Linear time O(n)

You must specify size at creation

And double to grow





BIG O REDUCED

Rules for reducing

1. Drop the non-dominant terms.
2. Drop the constants.
3. Add dominant.
4. Multiply nested.

Drop the non-dominant terms

```
func someFunc(_ n: Int) {
    var a = 0
    a = 5                                   O(1)
    a += 1

    for _ in 0..<n {
        // simple statements                O(n)
    }

    for _ in 0..<n {
        // simple statements                O(n^2)
        for _ in 0..<n {
            // simple statements
        }
    }
}
```

O(1) + O(n) + O(n^2)

↓

O(n^2)

If conditionals - take worst case

```
func someConditional(_ n: Int) {
    if n == 2 {
        for _ in 0..<n {
            // simple statements
        }
    } else {
        for _ in 0..<n {
            // simple statements
            for _ in 0..<n {
                // simple statements
            }
        }
    }
}
```

O(n^2)

Drop any constants

```
func dropConstants(_ n: Int) {
    for _ in 0..<n { // O(n)
        // simple statements
    }

    for _ in 0..<n { // O(n)
        // simple statements
    }

    for _ in 0..<n { // O(n)
        // simple statements
    }
}
```

O(n) + O(n) + O(n)

↓

O(3n)

↓

O(n)

Add dominant

```
func addDominant(_ n: Int, _ m: Int) {
    for _ in 0..<n { // O(n)
        // simple statements
    }

    for _ in 0..<m { // O(m)
        // simple statements
    }
}
```

O(n)
+
O(m)

O(n+m)

Multiply nested

```
func nested(_ n: Int, _ m: Int) {
    for _ in 0..<n { // O(n)
        // simple statements
        for _ in 0..<m { // O(m)
            // simple statements
        }
    }
}
```

O(n)
x
O(m)

O(n*m)

How would you reduce these?

O(n + log n) → ?

O(7*2^n + 5000*n^99) → ?

How would you reduce these?

O(n + log n) → O(n)

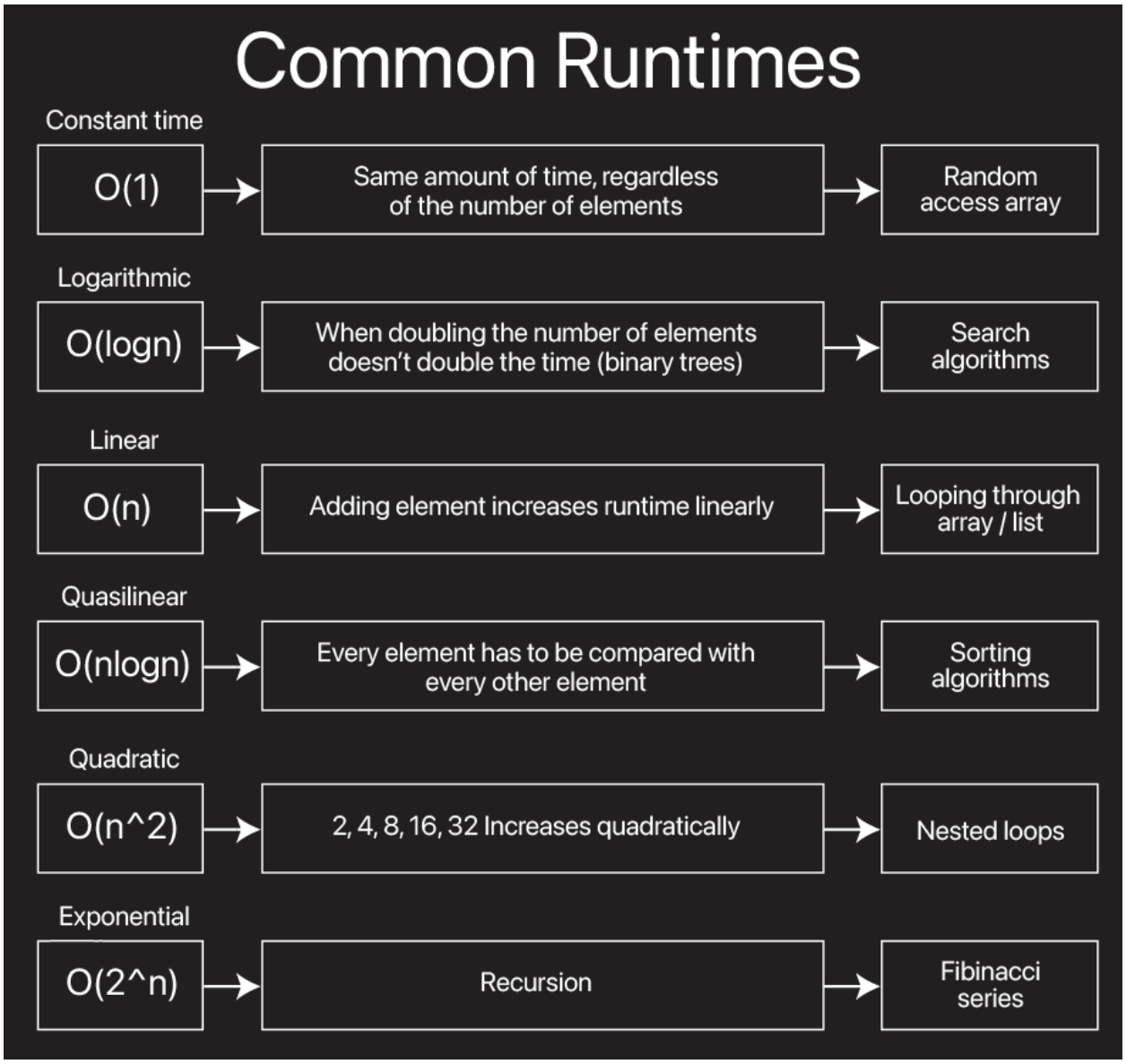
O(7*2^n + 5000*n^99) → O(2^n)

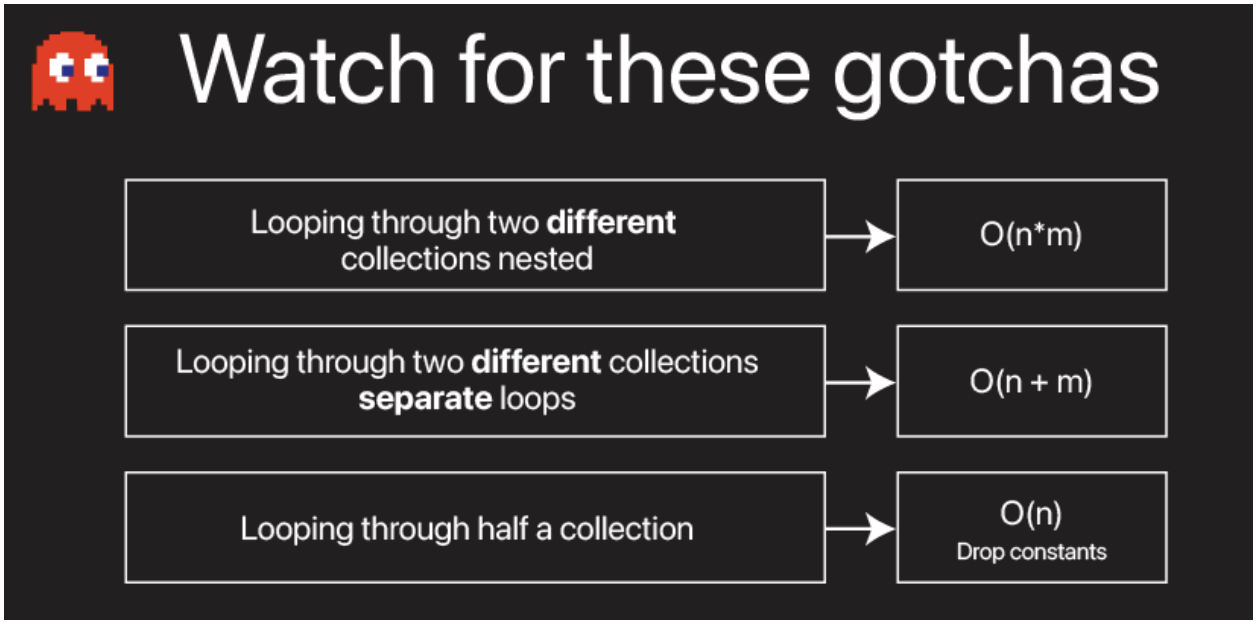
Which is equivalent to O(n)?

- O(n + m) where m < n/2
- O(2n)
- O(n + logn)
- O(n + m)

What you need to know

- Common runtimes
- How to determine runtime of an algorithm
- Rules for reducing


Common Runtimes
Constant time
O(1)
Same amount of time, regardless of the number of elements
Random access array
Logarithmic
O(logn)
When doubling the number of elements doesn't double the time (binary trees)
Search algorithms
Linear
O(n)
Adding element increases runtime linearly
Looping through array / list
Quasilinear
O(nlogn)
Every element has to be compared with every other element
Sorting algorithms
Quadratic
O(n^2)
2, 4, 8, 16, 32 Increases quadratically
Nested loops
Exponential
O(2^n)
Recursion
Fibinacci series


Watch for these gotchas
Looping through two **different** collections nested
O(n*m)
Looping through two **different** collections **separate** loops
O(n + m)
Looping through half a collection
O(n)
Drop constants

STACKS

QUEUES

You already know what stacks and queues are...

What is a stack?

stack of books....

What makes a stack so great?

Easy access to whatever is on top

What is a queue?

Get in a queue every day
- standing in line for bus
- ordering food at mcdonalds
- surfing
- waiting to play your favorite video game

Computer science users these concepts in data structures

Only they use slightly different names when describing them

Computer Stack (LIFO)

Last In First Out

push → ← pop

Used in...
- Forward/back in browser
- Undo/Redo functionality

Computer Queue (FIFO)

First In First Out

enqueue →

→ dequeue

Used in...
- Printer queue
- Input streams

Can be built using Arrays or Linked Lists...

Stack

Array

push →

0	1	2	3

→ pop

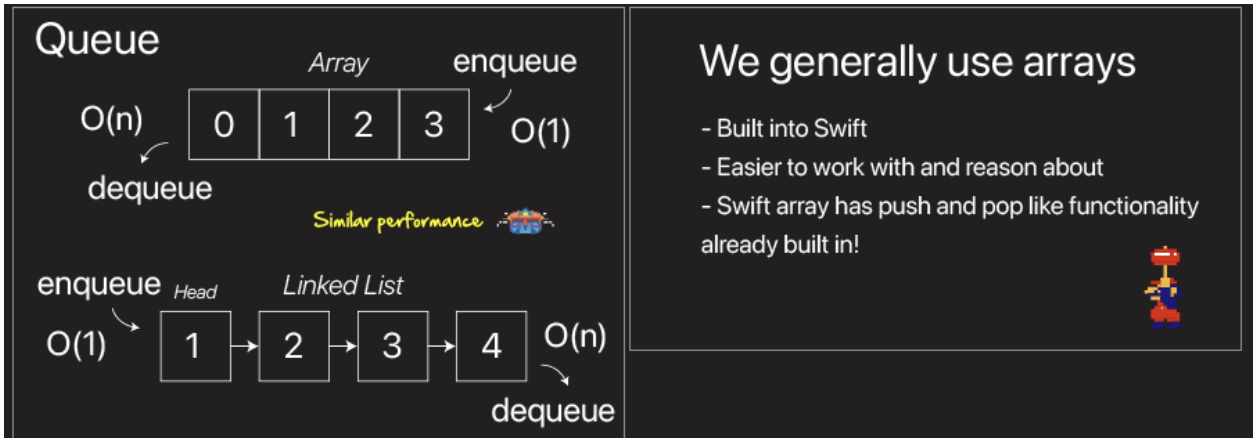
O(1)

push → O(1) → pop

Linked List

Head

1 → 2 → 3 → 4



We generally use arrays

- Built into Swift
- Easier to work with and reason about
- Swift array has push and pop like functionality already built in!

What you need to know

- Stack push/pop - O(1)
- Queue
  - enqueue - O(1)
  - dequeue - O(n)
- Can be built with linked lists or arrays

Stack

push

O(1)

pop

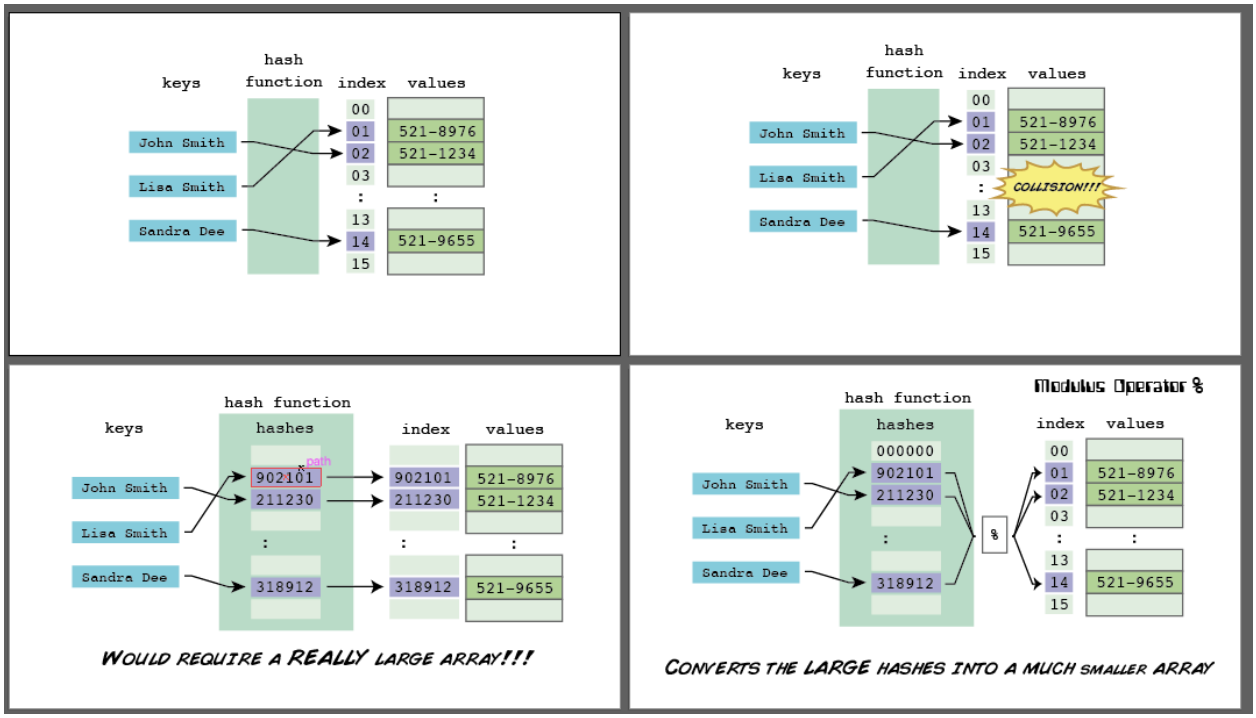
Queue

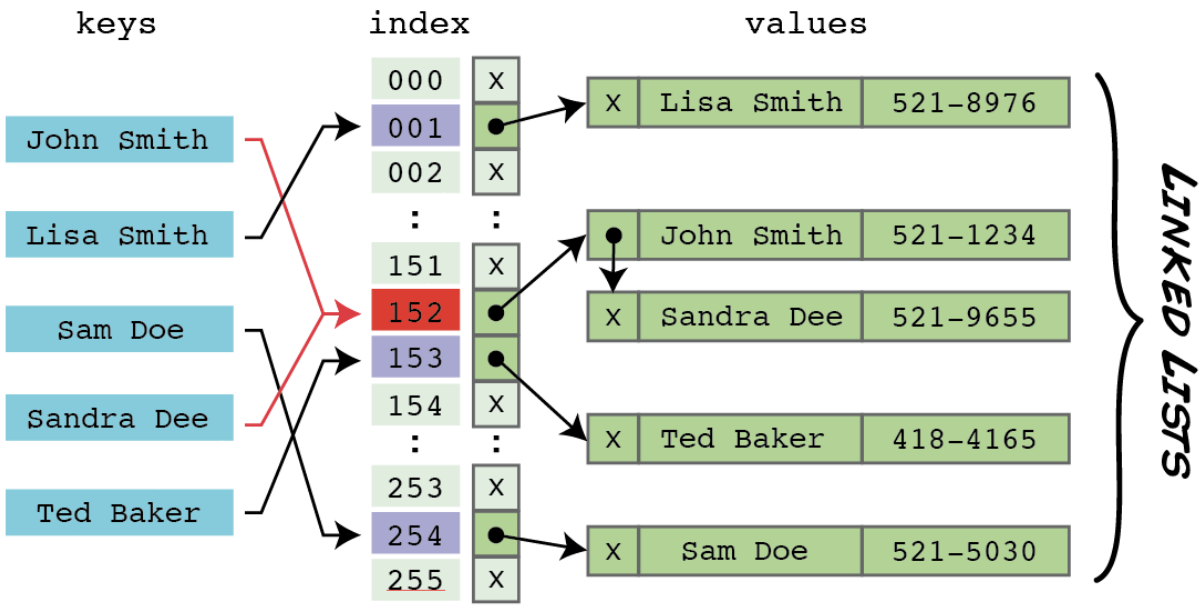
enqueue

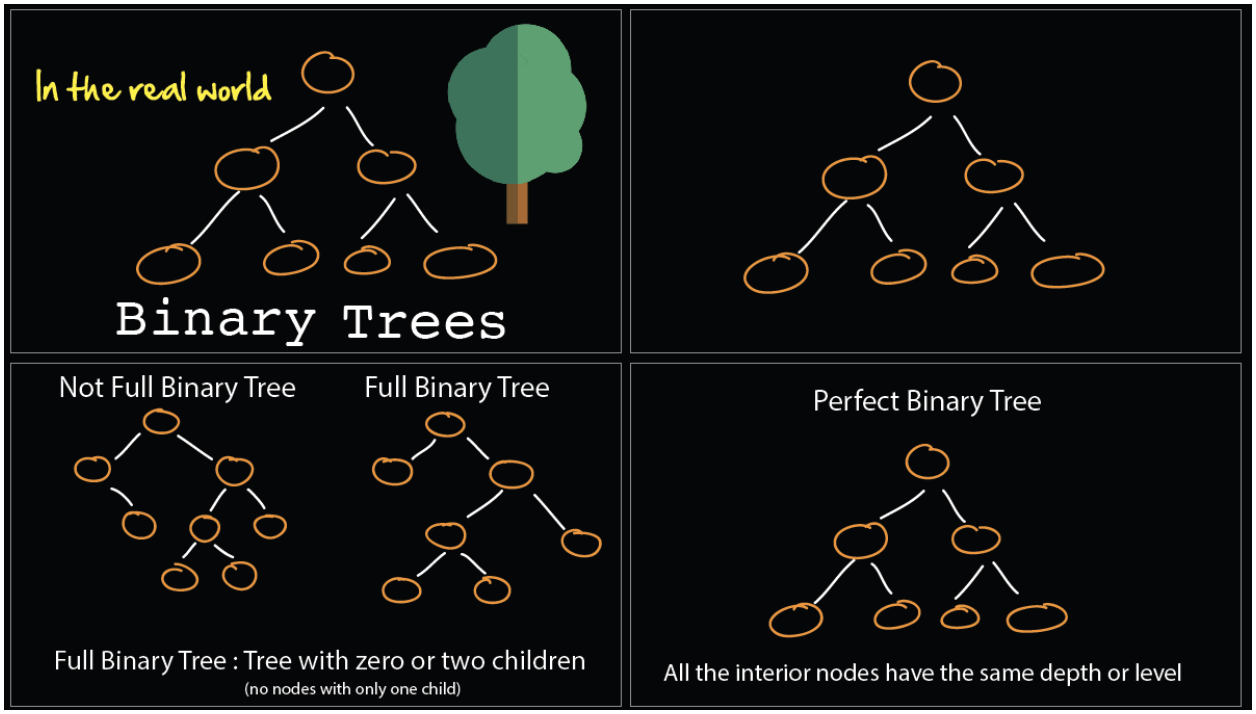
O(1)

O(n)

dequeue


keys
hash function
index
values
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655
COLLISION!!!
hashes
000000
902101
211230
318912
path
Modulus Operator %
%
WOULD REQUIRE A REALLY LARGE ARRAY!!!
CONVERTS THE LARGE HASHES INTO A MUCH SMALLER ARRAY

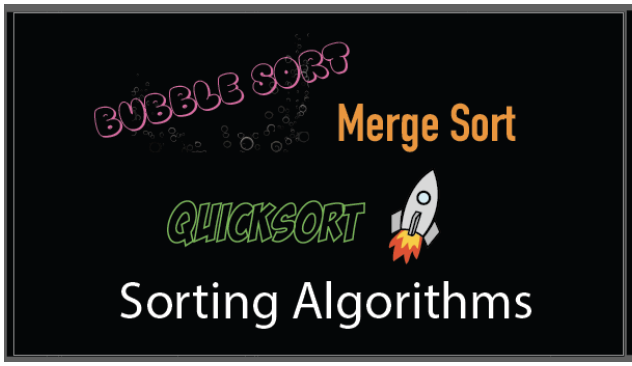

keys
index
values
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
X
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030
LINKED LISTS

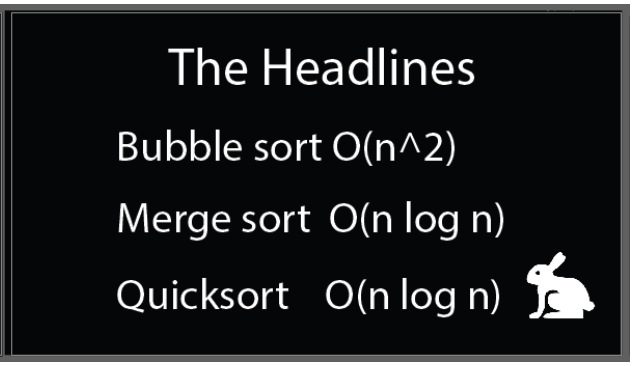

In the real world
Binary Trees
Not Full Binary Tree
Full Binary Tree
Full Binary Tree : Tree with zero or two children
(no nodes with only one child)
Perfect Binary Tree
All the interior nodes have the same depth or level

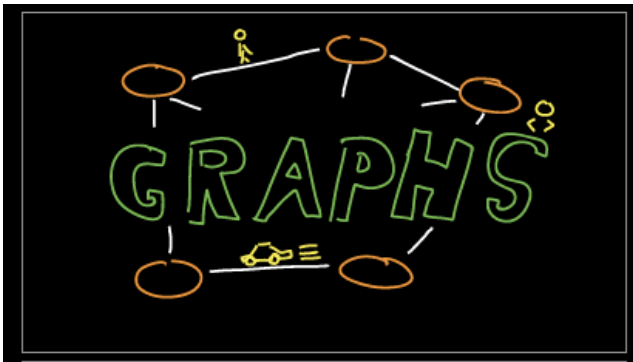
Memoization Defn

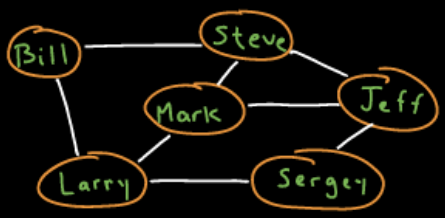
An optimization technique that
stores expensive calculated results
and returns them when asked for again.

It's like caching expensive results


BUBBLE SORT
Merge Sort
QUICKSORT
Sorting Algorithms

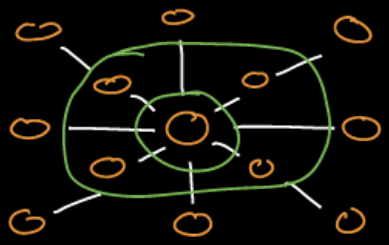

The Headlines
Bubble sort O(n^2)
Merge sort O(n log n)
Quicksort O(n log n)



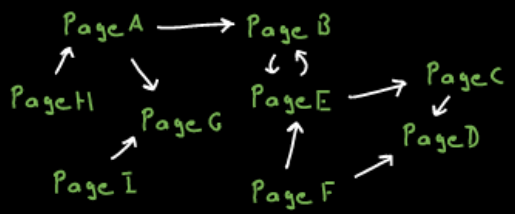


Social Networks

Pathfinding



Nearest Neighbours



Mapping World Wide Web

Binary Tree

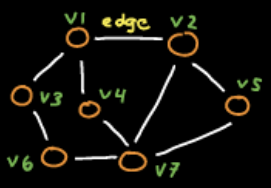
Graph Defn:

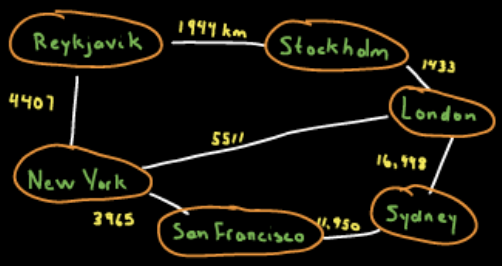
A graph G is an ordered pair of a set of vertices V and a set of edges E.

$G = (V, E)$

Edges:

directed undirected



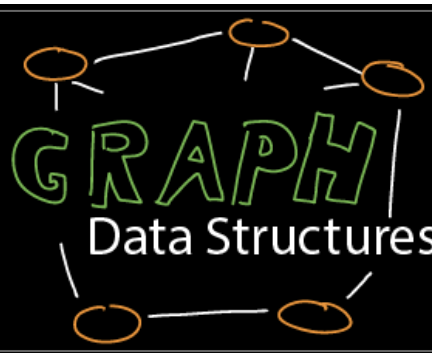


Weighted Graph

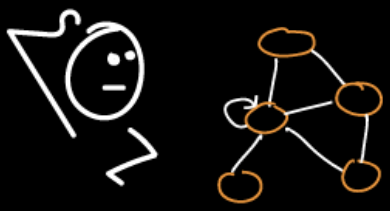
How to map our edges and vertices

Data Structure Algorithms

Intersting things we can do with them



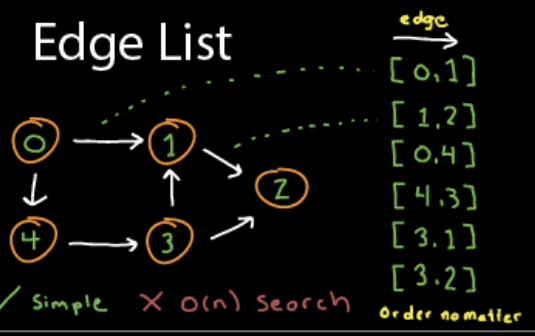
Graph Data Structures



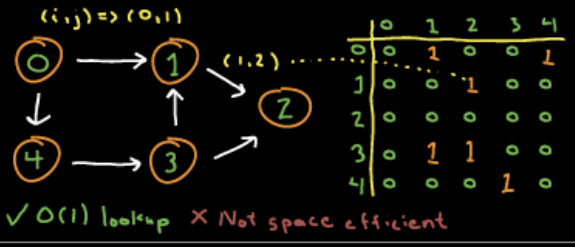
Graph Data Structures

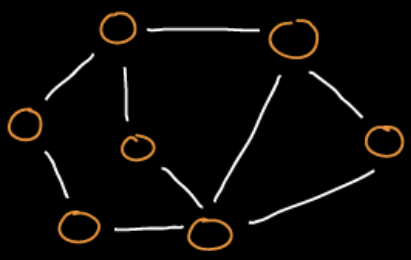
Edge Lists
Adjacency Matrices
Adjacency Lists

Edge List

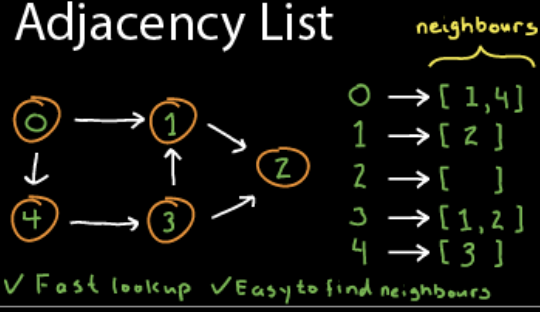


Adjacency Matrix

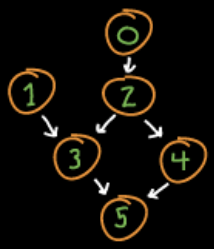




Adjacency List



Challenge



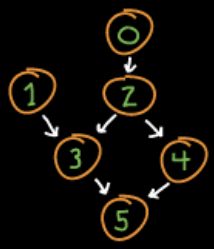
Come up with ...

=> Edge List

=> Adjacency Matrix

=> Adjacency List

Challenge



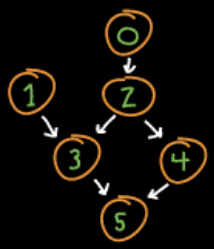
Come up with ...

=> Edge List

=> Adjacency Matrix

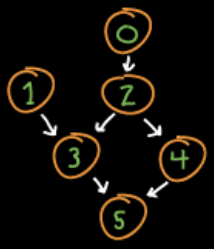
=> Adjacency List

Challenge Edge List



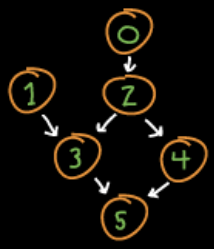
[0,2]
[2,3]
[2,4]
[1,3]
[3,5]
[4,5]

Challenge Adjacency Matrix



	0	1	2	3	4	5
0		1				
1				1		
2				1	1	
3						1
4						1
5						

Challenge Adjacency List



0 → [2]
1 → [3]
2 → [3,4]
3 → [5]
4 → [5]
5 → [ ]

- film yourself doing the above
- explain what you do
  - how you check (count edges)
- start with a blank array and fill in

Bonus Round!

Undirected

Takeaway -> if undirected
include both edges

Graphing Algorithms

Nearest neighbour

Breadth First Search

Depth First Search – Path Finding

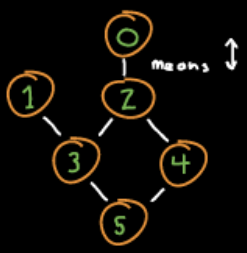
Dijstra's Algorithm

Shortest path

Challenge

← UNDIRECTED →

Edge List

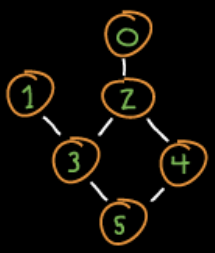


[0,2] [2,0]
[2,3] [3,2]
[2,4] [4,2]
[1,3] [3,1]
[3,5] [5,3]
[4,5] [5,4]

Challenge

← UNDIRECTED →

Adjacency Matrix

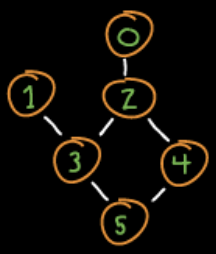


	0	1	2	3	4	5
0			1			
1				1		
2	1				1	1
3		1	1			1
4			1			1
5				1	1	

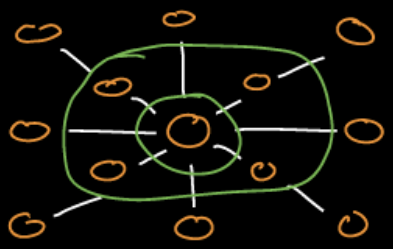
Challenge

← UNDIRECTED →

Adjacency List



0 → [2]
1 → [3]
2 → [0, 3, 4]
3 → [1, 2, 5]
4 → [2, 5]
5 → [3, 4]



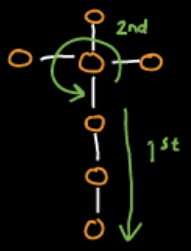
Breadth First Search

Nearest neighbour

Breadth First Search

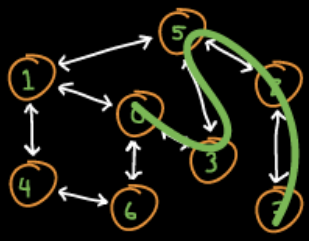
Nearest neighbour

Depth First Search



Breadth First Search

← UNDIRECTED →

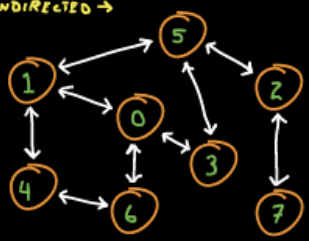


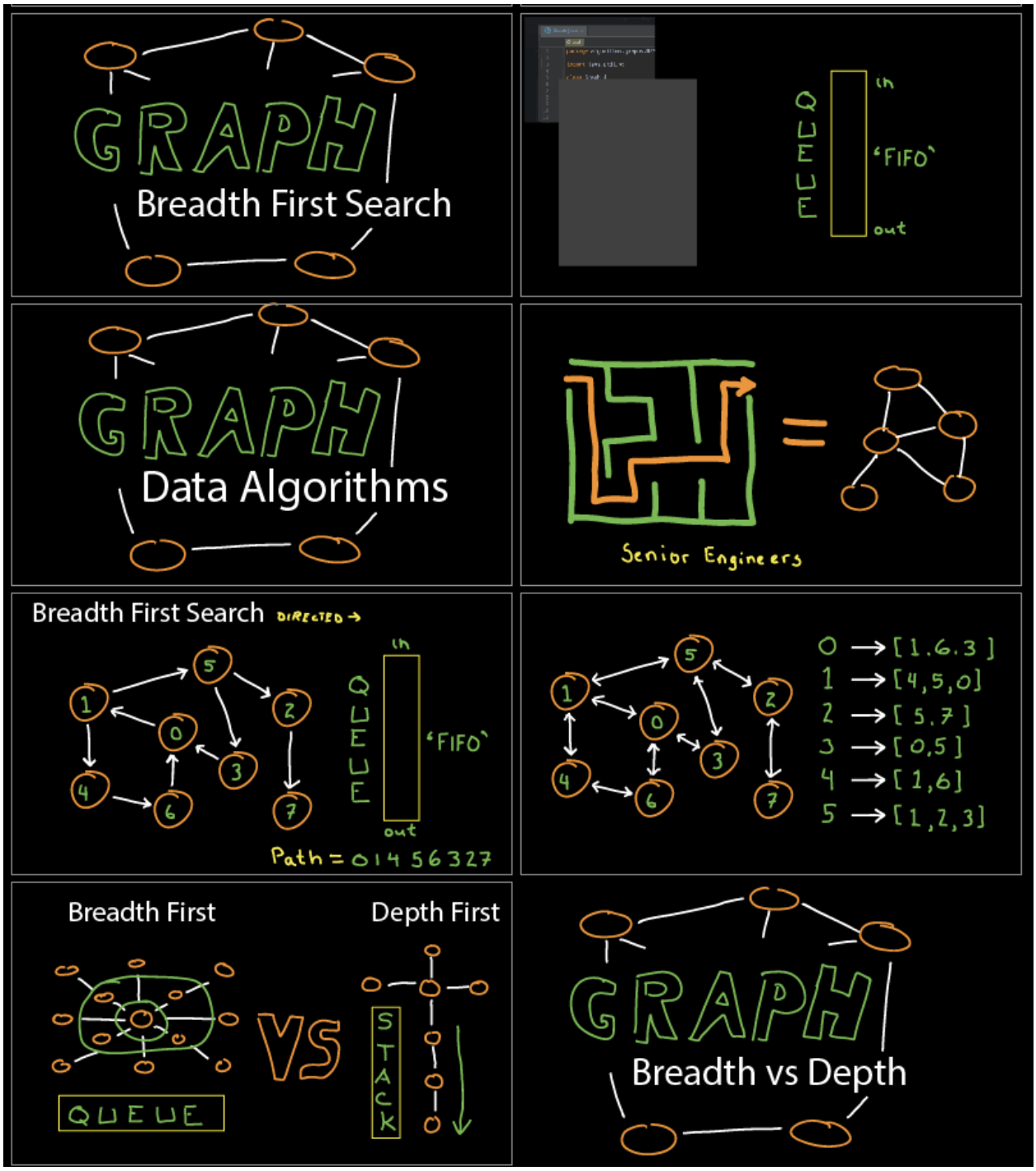
Path = 0 1 6 3 4 5 2 7

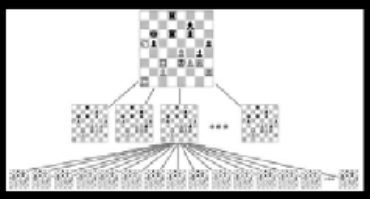
Depth First Search

Path = 0 3 5 2 7 6 4 1

← UNDIRECTED →




GRAPH
Breadth First Search
QUEUE
in
'FIFO'
out
GRAPH
Data Algorithms
Senior Engineers
Breadth First Search DIRECTED →
QUEUE
in
'FIFO'
out
Path = 0 1 4 5 6 3 2 7
0 → [1,6,3]
1 → [4,5,0]
2 → [5,7]
3 → [0,5]
4 → [1,6]
5 → [1,2,3]
Breadth First
Depth First
QUEUE
VS
STACK
GRAPH
Breadth vs Depth

Breadth First	Depth First
Better near the top 	Better faraway 
Social networks (FB, LinkedIn)	Game simulations
Nearby peers in games	
 Nearest neighbour	 Faraway