Коллекции, Closures

Что рассмотрим?

- Коллекции в Swift
- Их виды
- Практика
- O - нотация
- Циклы и базовые алгоритмы
- Closure
- Виды Closure

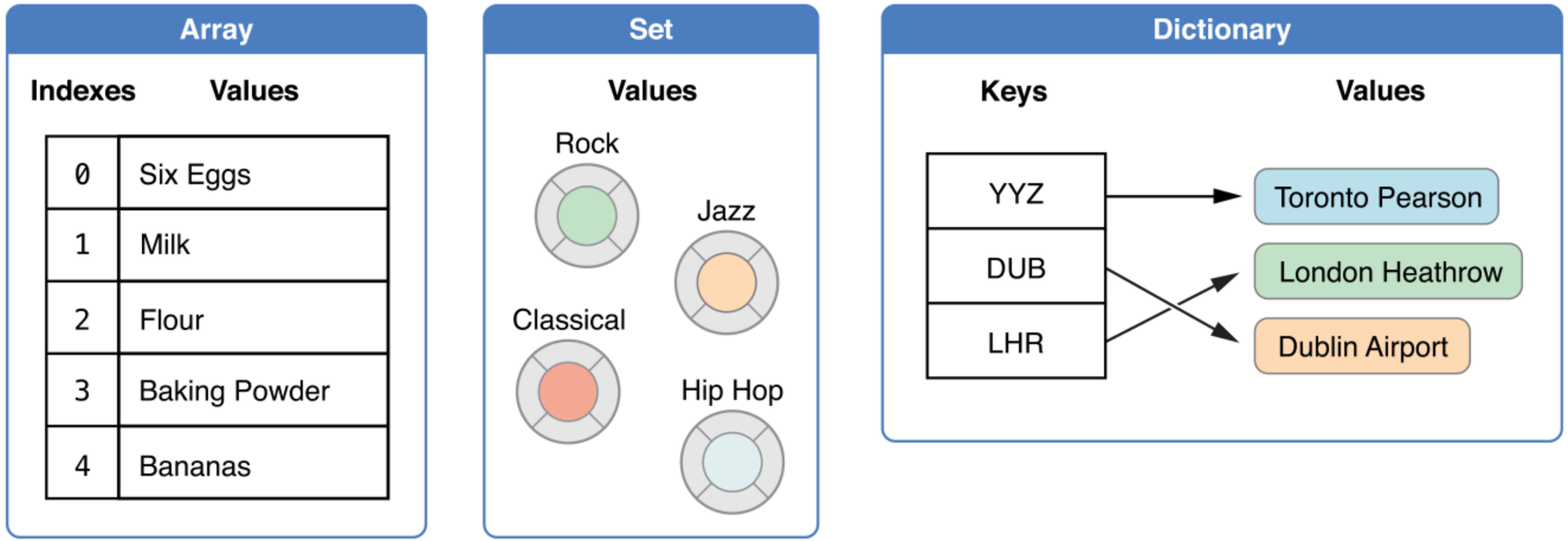
Коллекция -

Объект, содержащий в себе набор значений одного или разных типов.

Коллекции

- Array
- Dictionary
- Set

Коллекции


Array
Indexes
Values
0
Six Eggs
1
Milk
2
Flour
3
Baking Powder
4
Bananas
Set
Values
Rock
Jazz
Classical
Hip Hop
Dictionary
Keys
Values
YYZ
DUB
LHR
Toronto Pearson
London Heathrow
Dublin Airport

Array

```
@frozen public struct Array<Element> { }
```

- Array<Element>
- [Element]

Array: Basics

Creating an Empty Array

```
var someInts = [Int]()
print("someInts is of type [Int] with \(someInts.count) items.")
// Prints "someInts is of type [Int] with 0 items."
```

```
someInts.append(3)
// someInts now contains 1 value of type Int
someInts = []
// someInts is now an empty array, but is still of type [Int]
```

```
let array: [Int] = []
```

Creating an Array with a Default Value

```
var threeDoubles = Array(repeating: 0.0, count: 3)
// threeDoubles is of type [Double], and equals [0.0, 0.0, 0.0]
```

Creating an Array by Adding Two Arrays Together

```
var anotherThreeDoubles = Array(repeating: 2.5, count: 3)
// anotherThreeDoubles is of type [Double], and equals [2.5, 2.5, 2.5]

var sixDoubles = threeDoubles + anotherThreeDoubles
// sixDoubles is inferred as [Double], and equals [0.0, 0.0, 0.0, 2.5, 2.5,
  2.5]
```

Array: Basics

Creating an Array with an Array Literal

```
var shoppingList: [String] = ["Eggs", "Milk"]
// shoppingList has been initialized with two initial items
```

==

Accessing and Modifying an Array

```
print("The shopping list contains \(shoppingList.count) items.")
// Prints "The shopping list contains 2 items."
```

```
if shoppingList.isEmpty {
    print("The shopping list is empty.")
} else {
    print("The shopping list isn't empty.")
}
// Prints "The shopping list isn't empty."
```

```
shoppingList.append("Flour")
// shoppingList now contains 3 items, and someone is making pancakes
```

Array: Basics

```swift
var firstItem = shoppingList[0]
// firstItem is equal to "Eggs"
```

subscript(index: Int) -> Element { get set }

```swift
shoppingList[0] = "Six eggs"
// the first item in the list is now equal to "Six eggs" rather than "Eggs"
```

```swift
shoppingList[4...6] = ["Bananas", "Apples"]
// shoppingList now contains 6 items
```

```swift
shoppingList.insert("Maple Syrup", at: 0)
// shoppingList now contains 7 items
// "Maple Syrup" is now the first item in the list
```

```swift
let mapleSyrup = shoppingList.remove(at: 0)
// the item that was at index 0 has just been removed
// shoppingList now contains 6 items, and no Maple Syrup
// the mapleSyrup constant is now equal to the removed "Maple Syrup" string
```

mutating func popLast() -> Element?

Array: Iterating

```swift
for item in shoppingList {
    print(item)
}
// Six eggs
// Milk
// Flour
// Baking Powder
// Bananas
```

```swift
for (index, value) in shoppingList.enumerated() {
    print("Item \(index + 1): \(value)")
}
// Item 1: Six eggs
// Item 2: Milk
// Item 3: Flour
// Item 4: Baking Powder
// Item 5: Bananas
```

NOTE

If you try to access or modify a value for an index that's outside of an array's existing bounds, you will trigger a runtime error. You can check that an index is valid before using it by comparing it to the array's `count` property. The largest valid index in an array is `count - 1` because arrays are indexed from zero—however, when `count` is `0` (meaning the array is empty), there are no valid indexes.

Set

```
@frozen struct Set<Element> where Element : Hashable
```

- Set<Element>
- Element должен быть Hashable

```
func hash(into hasher: inout Hasher)
```

Set: Basics

Creating and initializing an Empty Set

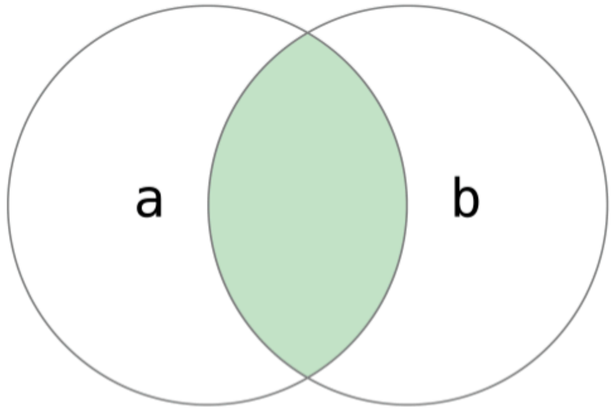
```
var letters = Set<Character>()
print("letters is of type Set<Character> with \(letters.count) items.")
// Prints "letters is of type Set<Character> with 0 items."
```

Creating a Set with an Array Literal

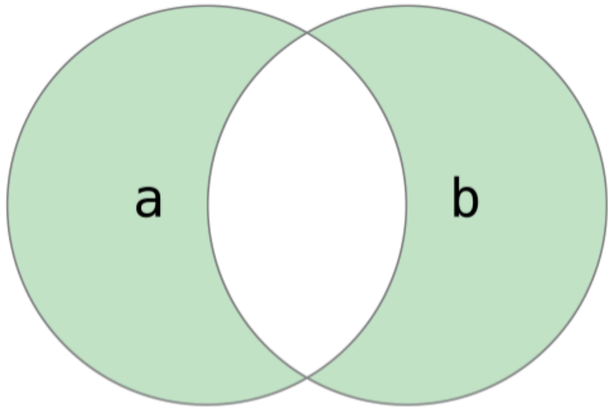
```
var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]
// favoriteGenres has been initialized with three initial items
```

Set: Operations

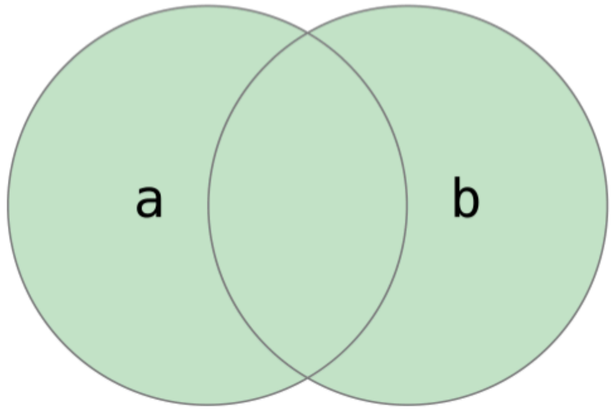
a.intersection(b)



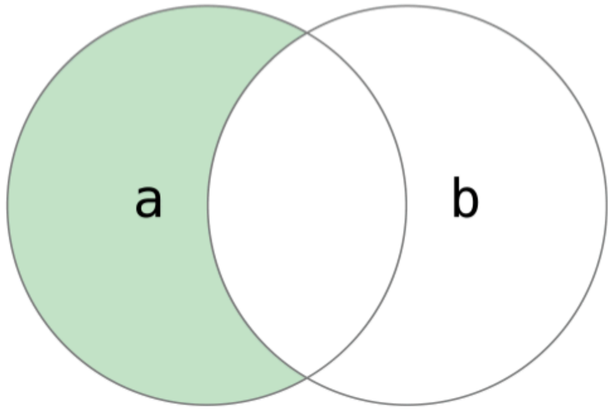
a.symmetricDifference(b)



a.union(b)



a.subtracting(b)



Dictionary

```
@frozen struct Dictionary<Key, Value> where Key : Hashable
```

- Dictionary<Key, Value>
- Key должен быть Hashable
- [Key: Value]

Playground

Работаем с коллекциями в playground

Loops

- for in
- while
- forEach

For In

```swift
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    print("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

```swift
for index in 1...5 {
    print("\(index) times 5 is \(index * 5)")
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

```swift
let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
for (animalName, legCount) in numberOfLegs {
    print("\(animalName)s have \(legCount) legs")
}
// cats have 4 legs
// ants have 6 legs
// spiders have 8 legs
```

```swift
let base = 3
let power = 10
var answer = 1
for _ in 1...power {
    answer *= base
}
print("\(base) to the power of \(power) is \(answer)")
// Prints "3 to the power of 10 is 59049"
```

```swift
let minuteInterval = 5
for tickMark in stride(from: 0, to: minutes, by: minuteInterval) {
    // render the tick mark every 5 minutes (0, 5, 10, 15 ... 45, 50, 55)
}
```

While

```
while condition {
    statements
}
```

```
repeat {
    statements
} while condition
```

```
var square = 0
var diceRoll = 0
while square < finalSquare {
    // roll the dice
    diceRoll += 1
    if diceRoll == 7 { diceRoll = 1 }
    // move by the rolled amount
    square += diceRoll
    if square < board.count {
        // if we're still on the board, move up or down for a snake or a
  ladder
        square += board[square]
    }
}
print("Game over!")
```

```
repeat {
    // move up or down for a snake or ladder
    square += board[square]
    // roll the dice
    diceRoll += 1
    if diceRoll == 7 { diceRoll = 1 }
    // move by the rolled amount
    square += diceRoll
} while square < finalSquare
print("Game over!")
```

forEach

```
label name: while condition {
    statements
}
```

```
let puzzleInput = "great minds think alike"
var puzzleOutput = ""
let charactersToRemove: [Character] = ["a", "e", "i", "o", "u", " "]
for character in puzzleInput {
    if charactersToRemove.contains(character) {
        continue
    }
    puzzleOutput.append(character)
}
print(puzzleOutput)
// Prints "grtmndsthnklk"
```

```
let numberWords = ["one", "two", "three"]
for word in numberWords {
    print(word)
}
// Prints "one"
// Prints "two"
// Prints "three"
```

```
numberWords.forEach { word in
    print(word)
}
// Same as above
```

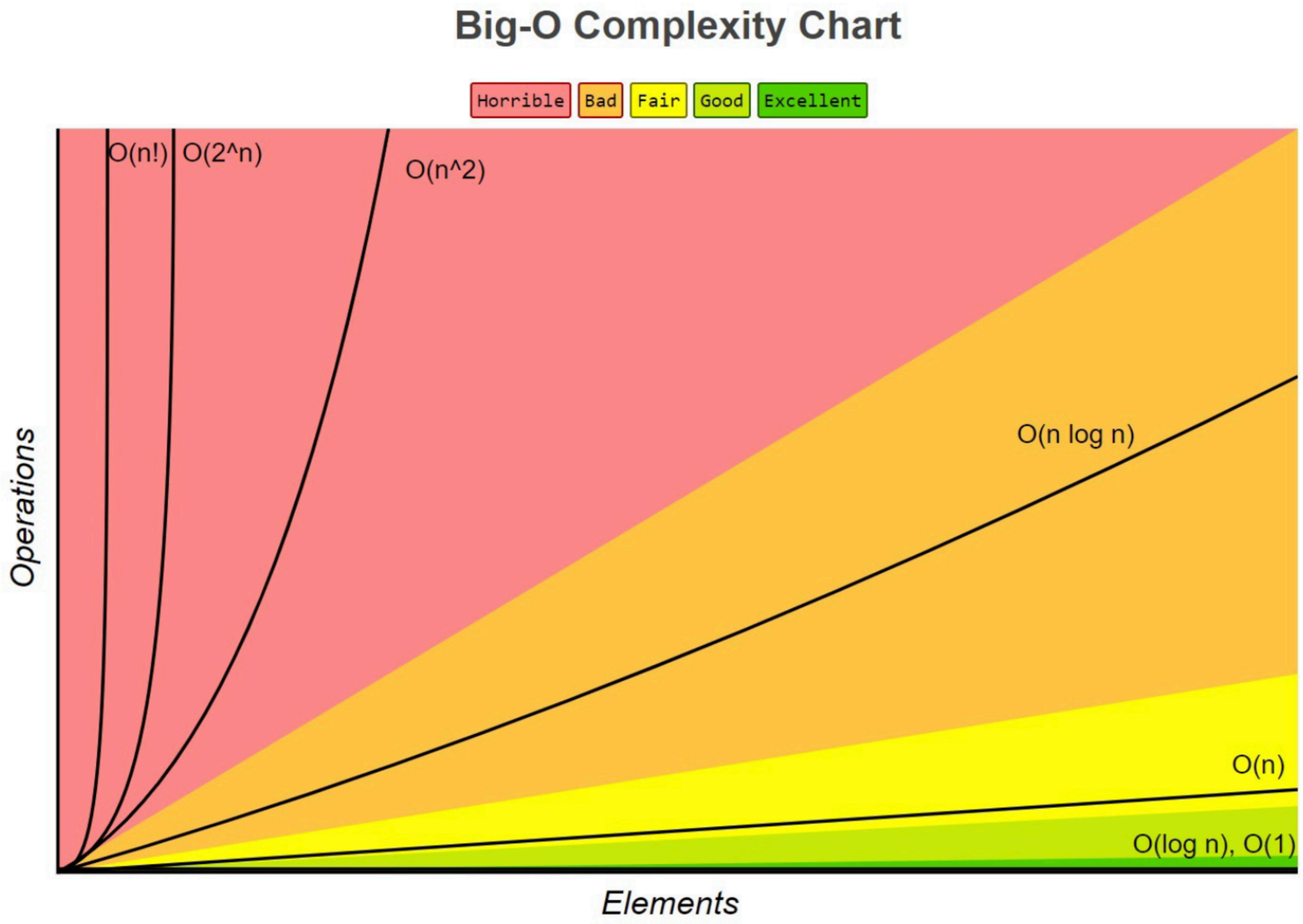
Loops

Работаем в Playground:

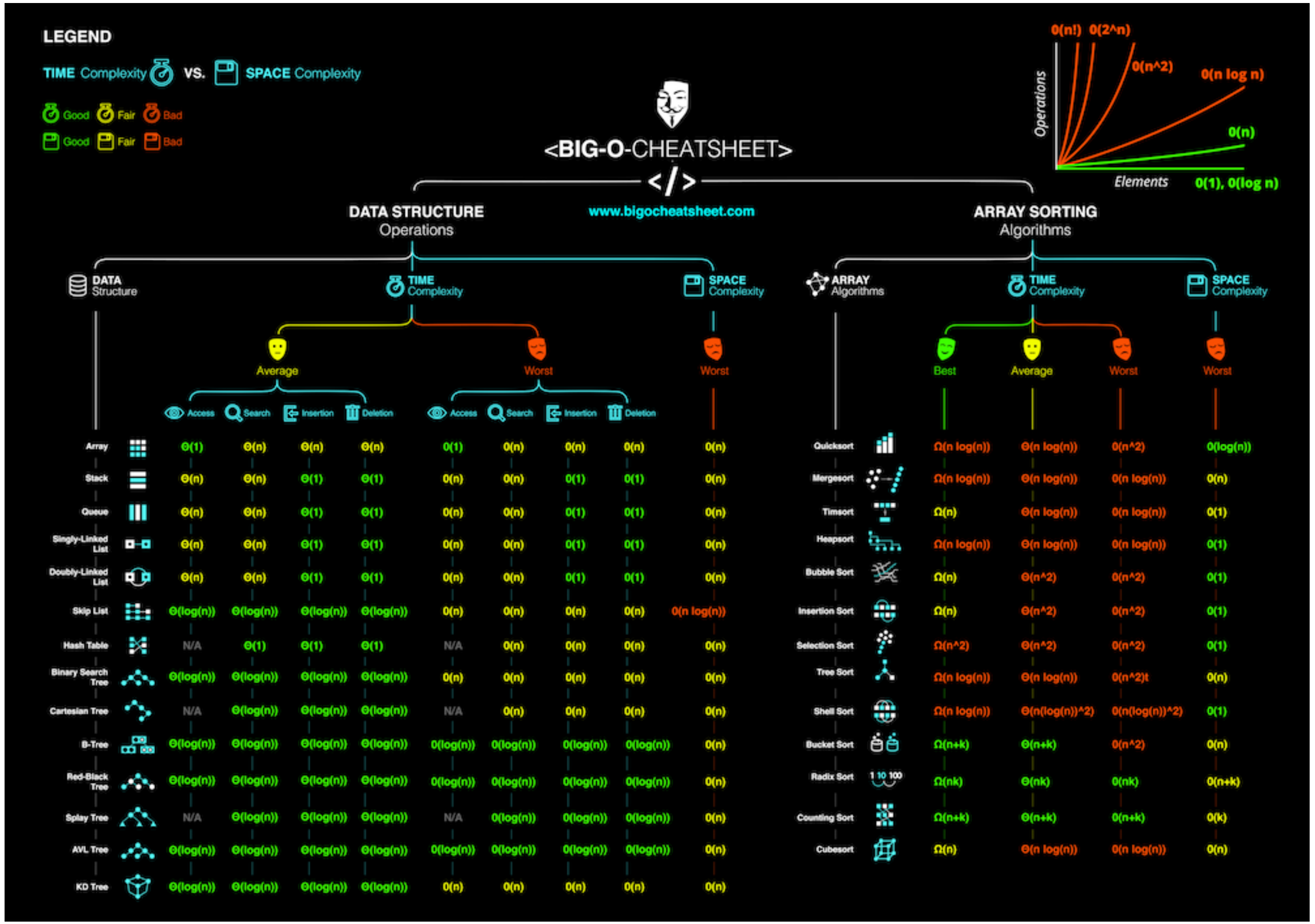
Big-O notation

- математическая нотация, которая описывает ограничивающее поведение функции, когда аргумент стремится к определенному значению или бесконечности. Он является членом семейства нотаций, изобретенных Полом Бахманом, Эдмундом Ландау и другими, которые в совокупности называются нотациями Бахмана-Ландау или асимптотическими нотациями
- https://habr.com/ru/post/444594/
- https://webdevblog.ru/bolshoe-o-chto-eto-takoe-pochemu-eto-vazhno-i-pochemu-eto-ne-vazhno/
- Грокаем алгоритмы. Бхаргава Адитья

Big-O notation



Big-O notation



Closures

- @nonescaping
- @escaping
- @autoclosure
- @convention(block) -> Objective-C compatible block

@escaping @nonescaping

- @escaping - не дожидается выполнения, функция возвращает сразу
- @nonescaping - по-умолчанию, дожидается выполнения
- опциональные closure по-умолчанию @escaping

Capture list

```
var closure = { [value] in
  value = 100
  withUnsafePointer(to: value) { print($0) }
}
```

- Захватывают значения (и у value и у reference типов)
- weak, unowned

@autoclosure

```
// customersInLine is ["Alex", "Ewa", "Barry", "Daniella"]
func serve(customer customerProvider: () -> String) {
    print("Now serving \(customerProvider())!")
}
serve(customer: { customersInLine.remove(at: 0) } )
// Prints "Now serving Alex!"
```

```
// customersInLine is ["Ewa", "Barry", "Daniella"]
func serve(customer customerProvider: @autoclosure () -> String) {
    print("Now serving \(customerProvider())!")
}
serve(customer: customersInLine.remove(at: 0))
// Prints "Now serving Ewa!"
```

Что почитать

- Коллекции и их поведение
- Array
- Dictionary
- Set
- Control Flow
- Big-O Cheatsheet

Домашнее задание

- Описать все отличия objective-c блоков от swift closure в виде чекпоинтов
- Реализовать коллекцию multiset
- Multiset - позволяет хранить элементы и их количество
- При добавлении - если элемент уже есть в multiset, то увеличивается счетчик этого элемента, в противном случае элемент добавляется
- Также должна быть возможность узнать общее количество всех элементов
- Время операций должно быть оптимальное
- Необходимые операции: add(element:), count(for element:), totalCount, remove(element:)
- Срок до 4 июля включительно