Embracing Algorithms

Dave Abrahams
Meet Crusty
Mind your own business
extension Canvas {
    mutating func deleteSelection() {
        for i in 0..<shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in 0..<shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in 0..<shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in 0..<shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in 0..<shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
        i += 1
    }
}
?!@##?
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
                shapes.remove(at: i)
            }
            else {
                i += 1
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        var i = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
            else {
                i += 1
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
```
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
```
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
Summary
Removes and returns the element at the specified position.

Declaration

```swift
mutating func remove(at index: Int) -> Shape
```

Discussion
All the elements following the specified position are moved up to close the gap.

```swift
var measurements: [Double] = [1.1, 1.5, 2.9, 1.2, 1.5, 1.3, 1.2]
let removed = measurements.remove(at: 2)
print(measurements)
// Prints "[1.1, 1.5, 1.2, 1.5, 1.3, 1.2]"
```

**Complexity**

$O(n)$, where $n$ is the length of the array.

**Parameters**

- `index` The position of the element to remove. `index` must be a valid index of the array.

**Returns**

The element at the specified index.
Summary
Removes and returns the element at the specified position.

Declaration
```
mutating func remove(at index: Int) -> Shape
```

Discussion
All the elements following the specified position are moved up to close the gap.

```
var measurements: [Double] = [1.1, 1.5, 2.9, 1.2, 1.5, 1.3, 1.2]
let removed = measurements.remove(at: 2)
print(measurements)
// Prints "[1.1, 1.5, 1.2, 1.5, 1.3, 1.2]"
```

Complexity
```
O(n), where n is the length of the array.
```

Parameters
```
index    The position of the element to remove. index must be a valid index of the array.
```

Returns
The element at the specified index.
Summary
Removes and returns the element at the specified position.

Declaration

```swift
mutating func remove(at index: Int) -> Shape
```

Discussion
All the elements following the specified position are moved up to close the gap.

```swift
var measurements: [Double] = [1.1, 1.5, 2.9, 1.2, 1.5, 1.3, 1.2]
let removed = measurements.remove(at: 2)
print(measurements)
// Prints "[1.1, 1.5, 1.2, 1.5, 1.3, 1.2]"
```

Complexity

$O(n)$, where $n$ is the length of the array.

Parameters

- `index` : The position of the element to remove. `index` must be a valid index of the array.

Returns

The element at the specified index.
Summary
Removes and returns the element at the specified position.

Declaration
mutating func remove(at index: Int) -> Shape

Discussion
All the elements following the specified position are moved up to close the gap.

```
var measurements: [Double] = [1.1, 1.5, 2.9, 1.2, 1.5, 1.3, 1.2]
let removed = measurements.remove(at: 2)
print(measurements)
// Prints "[1.1, 1.5, 1.2, 1.5, 1.3, 1.2]"
```

Complexity
\( O(n) \), where \( n \) is the length of the array.

Returns
The element at the specified index.
Delete the Green Ones
Delete the Green Ones
Delete the Green Ones
Scalability Matters

\[ N^2 = 225 \]
Scalability Matters
Scalability Matters

N^2 = 10,000
Scalability Matters

$N^2 = 10,000$

$N^2 = 225$
The Application Architect Speaks
Algorithm: Look it Up
Algorithm: Look it Up
Algorithm: Look it Up
Algorithm: Look it Up
Algorithm: Look it Up
Algorithm: Look it Up

**algorithm**  ['algəˌrithəm]  noun

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer: a basic algorithm for division.
Algorithm: Look it Up

algorithm
noun
a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer: a basic algorithm for division.
Algorithm: Look it Up
Algorithm: Look it Up
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        for i in (0..<shapes.count).reversed() {
            if shapes[i].isSelected {
                shapes.remove(at: i)
            }
        }
    }
}
extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: { $0.isSelected })
    }
}
extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: { $0.isSelected })
    }
}

extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: { $0.isSelected })
    }
}
extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: { $0.isSelected })
    }
}
extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: {$0.isSelected})
    }
}
Summary
Removes from the collection all elements that satisfy the given predicate.

Declaration
`mutating func removeAll(where predicate: (Self.Element) throws -> Bool) rethrows`

Discussion

**Complexity**

$O(n)$, where $n$ is the length of the collection.

**Parameters**

- **predicate**  
  A closure that takes an element of the sequence as its argument and returns a Boolean value indicating whether the element should be removed from the collection.
Linear vs Quadratic
Linear vs Quadratic
Linear vs Quadratic
Linear vs Quadratic
Linear vs Quadratic

Problem Size vs Time
Starting here, $O(n)$ wins forever
extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll(where: { $0.isSelected })
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {

    /// Removes all elements satisfying `shouldRemove`.
    ///                        ...
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {

        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)

    }

}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///                        ...
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)  
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///                        ...
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension `MutableCollection` where `Self : RangeReplaceableCollection` {
/// Removes all elements satisfying `shouldRemove`.
///                        ...
/// - Complexity: O(n) where n is the number of elements.
mutating func removeAll(where shouldRemove: (Element)->Bool) {
    let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
    removeSubrange(suffixStart...)
}
}
extension MutableCollection where Self : RangeReplaceableCollection {  
/// Removes all elements satisfying `shouldRemove`.  
/// ...  
/// - Complexity: O(n) where n is the number of elements.  
mutating func removeAll(where shouldRemove: (Element)->Bool) {  
    let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)  
    removeSubrange(suffixStart...)  
}  
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///                        ...
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {
  /// Removes all elements satisfying `shouldRemove`.
  ///
  /// - Complexity: O(n) where n is the number of elements.
  mutating func removeAll(where shouldRemove: (Element)->Bool) {
    let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
    removeSubrange(suffixStart...)
  }
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///
    /// – Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element)->Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension MutableCollection where Self : RangeReplaceableCollection {
    /// Removes all elements satisfying `shouldRemove`.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func removeAll(where shouldRemove: (Element) -> Bool) {
        let suffixStart = halfStablePartition(isSuffixElement: shouldRemove)
        removeSubrange(suffixStart...)
    }
}
extension MutableListableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }

        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) { swapAt(i, j); formIndex(after: &i) }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableCollection {

    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// returning the start position of the resulting suffix. 
    /// 
    /// - Complexity: $O(n)$ where $n$ is the number of elements.

    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else {
            return endIndex
        }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            j = index(after: j)
        }
        return i
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableListCollection {
/// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
/// returning the start position of the resulting suffix.
///
/// - Complexity: O(n) where n is the number of elements.
mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
    guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
    var j = index(after: i)
    while j != endIndex {
        if !isSuffixElement(self[j]) {
            swapAt(i, j); formIndex(after: &i)
        }
        formIndex(after: &j)
    }
    return i
}
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.

    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j); formIndex(after: &i)
            }
            formIndex(after: &j)
        }
        return i
    }
}
extension MutableCollection {

    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// returning the start position of the resulting suffix.
    ///
    /// - Complexity: O(n) where n is the number of elements.
    mutating func halfStablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        guard var i = firstIndex(where: isSuffixElement) else { return endIndex }
        var j = index(after: i)
        while j != endIndex {
            if !isSuffixElement(self[j]) {
                swapAt(i, j);
                formIndex(after: &i)
            } else {
                formIndex(after: &j)
            }
        }
        return i
    }
}
Get to know your standard library
Swift Standard Library

Solve complex problems and write high-performance, readable code.

Overview

The Swift standard library defines a base layer of functionality for writing Swift programs, including:

- Fundamental data types such as `Int`, `Double`, and `String`
- Common data structures such as `Array`, `Dictionary`, and `Set`
- Global functions such as `print(_:,separators:,terminator:)` and `abs(_:)`
- Protocols, such as `Collection` and `Equatable`, that describe common abstractions.
- Protocols, such as `CustomDebugStringConvertible` and `CustomReflectable`, that you use to customize operations that are available to all types.
- Protocols, such as `OptionSet`, that you use to provide implementations that would otherwise require boilerplate code.
Readability and Maintainability

Before

```swift
for i in (0..<shapes.count).reversed() {
    if shapes[i].isSelected {
        shapes.remove(at: i)
    }
}
```

After

```swift
shapes.removeAll(where: { $0 isSelected })
```
// Remove each selected item.
for i in (0..<shapes.count).reversed() {
    if shapes[i].isSelected {
        shapes.remove(at: i)
    }
}

shapes.removeAll(where: { $0.isSelected })
Readability and Maintainability

Before

// Remove each selected item.
// Go backward to avoid shifting
// as-yet-unprocessed elements.
for i in (0..<shapes.count).reversed() {
    if shapes[i].isSelected {
        shapes.remove(at: i)
    }
}

After

shapes.removeAll(where: { $0.isSelected })
Readability and Maintainability

Before

```swift
// Remove each selected item.
// Go backward to avoid shifting
// as-yet-unprocessed elements.
for i in (0..<shapes.count).reversed() {
    if shapes[i].isSelected {
        shapes.remove(at: i)
    }
}
```

After

```swift
shapes.removeAll { $0.isSelected }
```
“No Raw Loops”

Sean Parent (a.k.a. “Crusty's cousin”), C++ Seasoning
// Done, and done!

extension Canvas {
    mutating func deleteSelection() {
        shapes.removeAll { $0.isSelected }
    }
}

// Layering Operations
// bringToFront()

extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}

extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
// Layering Operations
// bringToFront()

extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
// Layering Operations
// bringToFront()

extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}

extension Canvas {
    mutating func sendToBack() {
        var i = 0, j = shapes.count
        while i < j {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.append(selected)
                j -= 1
            } else {
                i += 1
            }
        }
    }
}

extension Canvas {
    mutating func sendBackward() {
        for i in shapes.indices.reversed() where shapes[i].isSelected {
            var insertionPoint = i + 1
            if insertionPoint == shapes.count { return }
            for j in (0...i).reversed() where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint -= 1
            }
        }
    }
}

extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var shapesToInsert: [Shape] = []
        var insertionPoint = target
        var i = 0
        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
                insertionPoint -= 1
            } else {
                i += 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
            } else {
                i += 1
            }
        }
        shapes.insert(contentsOf: shapesToInsert, at: insertionPoint)
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var shapesToInsert: [Shape] = []
        var insertionPoint = target
        var i = 0
        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
                insertionPoint -= 1
            } else {
                i += 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
            } else {
                i += 1
            }
        }
        shapes.insert(contentsOf: shapesToInsert, at: insertionPoint)
    }
}
extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].is_selected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
    mutating func sendToBack() {
        var i = 0, j = shapes.count
        while i < j {
            if shapes[i].is_selected {
                let selected = shapes.remove(at: i)
                shapes.append(selected)
                j -= 1
            } else {
                i += 1
            }
        }
    }
}

extension Canvas {
    mutating func sendBackward() {
        for i in shapes.indices.reversed() where shapes[i].is_selected {
            var insertionPoint = i + 1
            if insertionPoint == shapes.count { return }
            for j in (0...i).reversed() where shapes[j].is_selected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint -= 1
            }
            return
        }
    }
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].is_selected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].is_selected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var shapesToInsert: [Shape] = []
        var insertionPoint = target
        var i = 0
        while i < insertionPoint {
            if shapes[i].is_selected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
                insertionPoint -= 1
            } else {
                i += 1
            }
        }
        while i < shapes.count {
            if shapes[i].is_selected {
                let x = shapes.remove(at: i)
                shapesToInsert.append(x)
            } else {
                i += 1
            }
        }
        shapes.insert(contentsOf: shapesToInsert, at: insertionPoint)
    }
}

$O(n^2)$
extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {

    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[0].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: 0)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
https://github.com/apple/swift/blob/master/test/Prototypes/Algorithms.swift
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        var i = 0, j = 0
        while i < shapes.count {
            if shapes[i].isSelected {
                let selected = shapes.remove(at: i)
                shapes.insert(selected, at: j)
                j += 1
            }
            i += 1
        }
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: (Shape) -> Bool)
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: (Shape) -> Bool)
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: (Shape) -> Bool)
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }

    /// Moves the selected shapes to the back, maintaining their relative order.
    mutating func sendToBack() {
        shapes.stablePartition(isSuffixElement: { $0.isSelected })
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }

    /// Moves the selected shapes to the back, maintaining their relative order.
    mutating func sendToBack() {
        shapes.stablePartition(isSuffixElement: { $0.isSelected })
    }
}
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }

    /// Moves the selected shapes to the back, maintaining their relative order.
    mutating func sendToBack() {
        shapes.stablePartition(isSuffixElement: { $0.isSelected })
    }
}
extension Canvas {

    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }

    /// Moves the selected shapes to the back, maintaining their relative order.
    mutating func sendToBack() {
        shapes.stablePartition(isSuffixElement: { $0.isSelected })
    }
}

Summary
Moves all elements satisfying isSuffixElement into a suffix of the collection, preserving their relative order, and returns the start of the resulting suffix.

Declaration
mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index

Discussion

Complexity
O(n log n) where n is the number of elements.

Declared In
Example.playground
extension Canvas {
    /// Moves the selected shapes to the front, maintaining their relative order.
    mutating func bringToFront() {
        shapes.stablePartition(isSuffixElement: { !$0.isSelected })
    }

    /// Moves the selected shapes to the back, maintaining their relative order.
    mutating func sendToBack() {
        shapes.stablePartition(isSuffixElement: { $0.isSelected })
    }
}

**Summary**

Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, and returns the start of the resulting suffix.

**Declaration**

```swift
mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index
```

**Discussion**

**Complexity**

\(O(n \log n)\) where \(n\) is the number of elements.

**Declared In**

Example.playground
/// Moves the selected shapes to the back, maintaining their relative order.

mutating func sendToBack() {
    shapes.stablePartition(isSuffixElement: { $0.isSelected })
}
Scalability Redux
Scalability Redux
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
            return
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
        for i in shapes.indices where shapes[i].isSelected {
            if i == 0 { return }
            var insertionPoint = i - 1
            for j in i..<shapes.count where shapes[j].isSelected {
                let x = shapes.remove(at: j)
                shapes.insert(x, at: insertionPoint)
                insertionPoint += 1
            }
        }
        return
    }
}
extension Canvas {
    mutating func bringForward() {
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            ...
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
        }
    }
}

extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...] // Means the same as shapes[predecessor..<endIndex]
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
Discover Algorithms
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0 isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Canvas {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward() {
        if let i = shapes.firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            shapes[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward() {
        if let i = firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward() {
        if let i = firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward() {
        if let i = firstIndex(where: { $0.isSelected }) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !$0.isSelected })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension Array where Element == Shape {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension Array {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension Array {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension Array {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableListCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == 0 { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableListCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection where Index == Int {
    mutating func bringForward(\(elementsSatisfying\ predicate: (Element) -> Bool) {\n        if let i = firstIndex(where: predicate) {\n            if i == startIndex { return }\n            let predecessor = i - 1\n            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })\n        }\n    }\n}
extension MutableCollection where Index == Int {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableListCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            if i == startIndex { return }
            let predecessor = i - 1
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let i = firstIndex(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...]!.stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...]!.stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

extension Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

definition Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

extension Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

extension Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

different extension Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
Building Towers Of Abstraction
Building Towers Of Abstraction

- removeAll(\text{where:})
- removeSubrange
- replaceSubrange
- halfStablePartition
- firstIndex(\text{where:})
- bringForward
- bringToFront
- sendToBack
- indexBeforeFirst(\text{where:})
- stablePartition
- rotate
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

extension Collection {
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}

extension Collection {
    /// Returns the `index` before the first one whose element satisfies `predicate`.
    /// - Complexity: O(n) where n is the length of the collection.
    func indexBeforeFirst(where predicate: (Element) -> Bool) -> Index? {
        return indices.first {
            let successor = index(after: $0)
            return successor != endIndex && predicate(self[successor])
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {

    /// Gathers the elements satisfying `predicate` at the position preceding
    /// the first element satisfying `predicate`.
    ///
    /// - Complexity: O(n log n) where n is the length of the collection.
    mutating func bringForward(elementsSatisfying predicate: (Element) -> Bool) {
        if let predecessor = indexBeforeFirst(where: predicate) {
            self[predecessor...].stablePartition(isSuffixElement: { !predicate($0) })
        }
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    /// - Complexity: $O(n \log n)$ where $n$ is the number of elements.
    /// - Precondition: `$n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension MutableCollection {

    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`.

    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}

/// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 {
            return startIndex
        }
        if n == 1 {
            return isSuffixElement(self[startIndex]) ? startIndex : endIndex
        }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension MutableCollection {

/// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
/// preserving their relative order, returning the start of the resulting suffix.
///
/// - Complexity: O(n log n) where n is the number of elements.
/// - Precondition: `n == self.count`

mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
    if n == 0 { return startIndex }
    if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
    let h = n / 2, i = index(startIndex, offsetBy: h)
    let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
    let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
    return self[j..<k].rotate(shiftingToStart: i)
}
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    
    /// - Complexity: $O(n \log n)$ where $n$ is the number of elements.
    /// - Precondition: `n == self.count`

    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElementElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }

    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    
    /// - Complexity: $O(n \log n)$ where $n$ is the number of elements.
    public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        // Implementation
    }
}
extension MutableCollection {
  /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
  /// preserving their relative order, returning the start of the resulting suffix.
  ///
  /// - Complexity: O(n log n) where n is the number of elements.
  /// - Precondition: `n == self.count`

  mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
    if n == 0 { return startIndex }
    if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
    let h = n / 2, i = index(startIndex, offsetBy: h)
    let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
    let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
    return self[j..<k].rotate(shiftingToStart: i)
  }
  /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
  /// preserving their relative order, returning the start of the resulting suffix.
  ///
  /// - Complexity: O(n log n) where n is the number of elements.
  ///
  public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
    
  }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElementElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`

    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
    
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    ///
    public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        return stablePartition(count: self.count, isSuffixElement: isSuffixElement)
    }
}
extension MutableListCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// preserving their relative order, returning the start of the resulting suffix. 
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, 
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        return stablePartition(count: self.count, isSuffixElement: isSuffixElement)
    }
}
extension MutableCollection {

/// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
/// preserving their relative order, returning the start of the resulting suffix.
///
/// - Complexity: O(n log n) where n is the number of elements.
/// - Precondition: `n == self.count`

mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
    if n == 0 { return startIndex }
    if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
    let h = n / 2, i = index(startIndex, offsetBy: h)
    let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
    let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElementElement)
    return self[j..<k].rotate(shiftingToStart: i)
}

/// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
/// preserving their relative order, returning the start of the resulting suffix.
///
/// - Complexity: O(n log n) where n is the number of elements.
public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`

    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElementElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElementElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    /// - Complexity: O(n log n) where n is the number of elements.
    public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
    }
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    public mutating func stablePartition(isSuffixElement: (Element) -> Bool) -> Index {
        // Implementation...
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection, preserving their relative order, returning the start of the resulting suffix.
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`
    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension MutableCollection {
    /// Moves all elements satisfying `isSuffixElement` into a suffix of the collection,
    /// preserving their relative order, returning the start of the resulting suffix.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    /// - Precondition: `n == self.count`

    mutating func stablePartition(count n: Int, isSuffixElement: (Element) -> Bool) -> Index {
        if n == 0 { return startIndex }
        if n == 1 { return isSuffixElement(self[startIndex]) ? startIndex : endIndex }
        let h = n / 2, i = index(startIndex, offsetBy: h)
        let j = self[..<i].stablePartition(count: h, isSuffixElement: isSuffixElement)
        let k = self[i...].stablePartition(count: n - h, isSuffixElement: isSuffixElement)
        return self[j..<k].rotate(shiftingToStart: i)
    }
}
extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0
        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            } else {
                i += 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
            } else {
                i += 1
            }
        }
        shapes.insert(contentsOf: buffer, at: insertionPoint)
    }
}
extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0

        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
            }
            else {
                i += 1
            }
        }
    }
}
extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0

        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
            }
            else {
                i += 1
            }
        }
    }
}
extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0

        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
            }
            else {
                i += 1
            }
        }
    }
}
mutating func gatherSelected(at target: Int) {
    var buffer: [Shape] = []
    var insertionPoint = target
    var i = 0

    while i < insertionPoint {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
            insertionPoint -= 1
        }
        else {
            i += 1
        }
    }

    while i < shapes.count {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
        }
        else {
            i += 1
        }
    }
}
mutating func gatherSelected(at target: Int) {
    var buffer: [Shape] = []
    var insertionPoint = target
    var i = 0

    while i < insertionPoint {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
            insertionPoint -= 1
        }
    }
    else {
        i += 1
    }
}
```swift
mutating func gatherSelected(at target: Int) {
    var buffer: [Shape] = []
    var insertionPoint = target
    var i = 0

    while i < insertionPoint {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
            insertionPoint -= 1
        }
        else {
            i += 1
        }
    }
    while i < shapes.count {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
        }
        else {
            i += 1
        }
    }
}
```
mutating func gatherSelected(at target: Int) {
    var buffer: [Shape] = []
    var insertionPoint = target
    var i = 0

    while i < insertionPoint {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
            insertionPoint -= 1
        }
        else {
            i += 1
        }
    }

    while i < shapes.count {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
        }
        else {
            i += 1
        }
    }
}
mutating func gatherSelected(at target: Int) {
    var buffer: [Shape] = []
    var insertionPoint = target
    var i = 0

    while i < insertionPoint {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
            insertionPoint -= 1
        }
        else {
            i += 1
        }
    }
    while i < shapes.count {
        if shapes[i].isSelected {
            let x = shapes.remove(at: i)
            buffer.append(x)
        }
        i += 1
    }
}
else {
    i += 1
}

}

while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    } else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
else {
    i += 1
}

while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    } else {
        i += 1
    }
}
else {
    i += 1
}

while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    } else {
        i += 1
    }
    i += 1
}
else {
    i += 1
}
}

while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}
{   if (shapes[i].isSelected) {     let x = shapes.remove(at: i)     buffer.append(x)   } else {     i += 1   } }
}
}
while i < shapes.count {   if shapes[i].isSelected {     let x = shapes.remove(at: i)     buffer.append(x)   } else {     i += 1   } }

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    } else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
if shapes[i].isSelected {
    let x = shapes.remove(at: i)
    buffer.append(x)
}
else {
    i += 1
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
if shapes[i].isSelected {
    let x = shapes.remove(at: i)
    buffer.append(x)
}
else {
    i += 1
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
  if shapes[i].isSelected {
    let x = shapes.remove(at: i)
    buffer.append(x)
  }
  else {
    i += 1
  }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
}
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
}
if shapes[i].isSelected {
    let x = shapes.remove(at: i)
    buffer.append(x)
}
else {
    i += 1
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
}
while i < shapes.count {
  if shapes[i].isSelected {
    let x = shapes.remove(at: i)
    buffer.append(x)
  }
  else {
    i += 1
  }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
while i < shapes.count {
    if shapes[i].isSelected {
        let x = shapes.remove(at: i)
        buffer.append(x)
    }
    else {
        i += 1
    }
}

shapes.insert(contentsOf: buffer, at: insertionPoint)
extension MutableCollection {
    /// Gathers elements satisfying `predicate` at `target`, preserving their relative order.
    /// - Complexity: \(O(n \log n)\) where \(n\) is the number of elements.
    mutating func gather(at target: Index, allSatisfying predicate: (Element)\(\rightarrow\)Bool) {
        let start = self[..<target].stablePartition(isSuffixElement: predicate)
        let end = self[target...].stablePartition(isSuffixElement: { !predicate($0) })
    }
}
extension MutableCollection {
    /// Gathers elements satisfying `predicate` at `target`, preserving their relative order.
    ///
    /// - Complexity: O(n log n) where n is the number of elements.
    mutating func gather(at target: Index, allSatisfying predicate: (Element)->Bool) {
        let start = self[..<target].stablePartition(isSuffixElement: predicate)
        let end = self[target...].stablePartition(isSuffixElement: { !predicate($0) })
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        shapes.gather(at: target) { $0.isSelected } // No contextually correct exponentiation method available
    }
}
extension MutableCollection {
    /// Gathers elements satisfying `predicate` at `target`, preserving their relative order.
    /// - Complexity: O(n log n) where n is the number of elements.
    mutating func gather(at target: Index, allSatisfying predicate: (Element)->Bool) {
        let start = self[..<target].stablePartition(isSuffixElement: predicate)
        let end = self[target...].stablePartition(isSuffixElement: { !predicate($0) })
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        shapes.gather(at: target) { $0.isSelected }
    }
}

extension MutableCollection {
    /// Gathers elements satisfying `predicate` at `target`, preserving their relative order.
    /// - Complexity: O(n log n) where n is the number of elements.
    mutating func gather(at target: Index, allSatisfying predicate: (Element)->Bool) {
        let start = self[..<target].stablePartition(isSuffixElement: predicate)
        let end = self[target...].stablePartition(isSuffixElement: { !predicate($0) })
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0
        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            }
            else {
                i += 1
            }
        }
        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
            } else {
                i += 1
            }
        }
        shapes.insert(contentsOf: buffer, at: insertionPoint)
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        var buffer: [Shape] = []
        var insertionPoint = target
        var i = 0

        while i < insertionPoint {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
                insertionPoint -= 1
            }
            else {
                i += 1
            }
        }

        while i < shapes.count {
            if shapes[i].isSelected {
                let x = shapes.remove(at: i)
                buffer.append(x)
            }
            else {
                i += 1
            }
        }

        shapes.insert(contentsOf: buffer, at: insertionPoint)
    }
}

extension MutableCollection {
    /// Gathers elements satisfying `predicate` at `target`, preserving their relative order.
    /// - Complexity: O(n log n) where n is the number of elements.
    mutating func gather(at target: Index, allSatisfying predicate: Element -> Bool) {
        let start = self[..<target].stablePartition(isSuffixElement: predicate)
        let end = self[target...].stablePartition(isSuffixElement: { !predicate($0) })
    }
}

extension Canvas {
    mutating func gatherSelected(at target: Int) {
        shapes.gather(at: target) { $0.isSelected }
    }
}
Discover Generic Algorithms
“If you want to improve the code quality in your organization, replace all of your coding guidelines with one goal:

No Raw Loops”

Sean Parent, *C++ Seasoning*
More Information

Sean Parent, C++ Seasoning
https://channel9.msdn.com/Events/GoingNative/2013/Cpp-Seasoning

Alexander Stepanov and Paul McJones, Elements of Programming
https://www.youtube.com/watch?v=Ih9gpJga4Vc

Swift Algorithms Prototype
https://github.com/apple/swift/blob/master/test/Prototypes/Algorithms.swift

Swift Standard Library Documentation
https://developer.apple.com/documentation/swift/swift_standard_library
More Information

https://developer.apple.com/wwdc18/223

Swift Generics
Hall 2
Wednesday 3:00PM