ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью практики является формирование и развитие общих и профессиональных компетенций по модулю ПМ.03 Участие в интеграции программных модулей.

В задачи учебной практики входят закрепление навыков разработки программного обеспечения, использования методов для получения кода с заданной функциональностью и степенью качества, разработки документации на программный продукт; знаний моделей процесса разработки программного обеспечения, основных принципов процесса разработки программного обеспечения, основных подходов к интегрированию программных модулей, основных методов и средств эффективной разработки ПО.

В качестве языка разработки был выбран swift.

Swift - это современный объектно-ориентированный язык программирования, который используется для разработки приложений для iOS, macOS, watchOS и tvOS. Swift был создан Apple и предназначен для замены Objective-C в качестве основного языка программирования для разработки приложений для Apple-устройств.

Xcode - это интегрированная среда разработки (IDE) для macOS, которая используется для создания приложений для iOS, macOS, watchOS и tvOS. Xcode включает в себя компилятор Swift и другие инструменты, необходимые для разработки приложений.

В Xcode для разработки приложений на Swift используется графический интерфейс пользователя (GUI) Interface Builder, который позволяет создавать пользовательские интерфейсы с помощью готовых элементов управления, таких как кнопки, текстовые поля и многое другое. Кроме того, для разработки на Swift можно использовать исходный код, написанный вручную, используя язык программирования Swift.

Существуют различные методы разработки на Swift, включая использование стандартной библиотеки Swift и сторонних библиотек, таких как Alamofire и SwiftyJSON. Также существуют множество инструментов и фреймворков, которые могут быть использованы для разработки приложений на Swift.

Разработка на Swift в Xcode может быть проще, чем на других языках программирования, благодаря графическому интерфейсу и интуитивно понятной среде разработки. Тем не менее, для разработки качественного приложения требуется знание основ Swift и Xcode, а также опыт работы с этими инструментами.

ОСНОВНАЯ ЧАСТЬ

Техническое задание на программный продукт по сортировкам

1. Введение

- Целью данного проекта является разработка программного продукта для сортировки массивов различными алгоритмами.

2. Описание продукта

- Программный продукт должен предоставлять пользователю возможность вводить массивы данных и выбирать метод сортировки из предоставленных вариантов. Для каждого метода сортировки должен быть разработан соответствующий алгоритм.

3. Требования к продукту

- Пользователь должен иметь возможность вводить массивы любой длины и содержащие любые целочисленные значения.
- Программа должна поддерживать не менее четырех различных методов сортировки: сортировка пузырьком, сортировка выбором, сортировка вставками и быстрая сортировка.
- Программа должна иметь графический интерфейс пользователя, который позволяет выбирать метод сортировки и вводить массивы данных.
- Пользователь должен иметь возможность сохранять результаты сортировки в файл.

4. Технические требования

- Программа должна быть написана на языке Swift и работать на операционных системах macOS и iOS.

- Интерфейс пользователя должен быть разработан с использованием фреймворка SwiftUI.
- Для хранения данных должен использоваться тип данных массив (Array) языка Swift.
- Код должен быть написан в соответствии с принципами SOLID и Clean Code.
- Для тестирования программного продукта должны быть написаны юнит-тесты.

Спецификация программного продукта

Название: SwiftSorting

Описание: SwiftSorting - это программный продукт на языке Swift, который содержит 4 алгоритма сортировки: сортировка пузырьком, сортировка выбором, сортировка вставками и быстрая сортировка. Каждый алгоритм имеет свои особенности и может использоваться в зависимости от требований и данных, которые нужно отсортировать.

Алгоритм 1: Сортировка пузырьком

Описание: Сортировка пузырьком - это простой алгоритм сортировки, который проходит по списку несколько раз, сравнивая пары элементов и меняя их местами, если они не отсортированы в нужном порядке.

Сигнатура функции:

func bubbleSort<T: Comparable>(_ arr: inout [T])

Аргументы функции:

- arr: массив элементов, который нужно отсортировать.

Возвращаемое значение:

- Отсортированный массив элементов

Алгоритм 2: Сортировка выбором

Описание: Сортировка выбором - это алгоритм сортировки, который ищет наименьший элемент в списке и перемещает его на первое место, затем ищет следующий наименьший элемент и перемещает его на второе место, и так далее, пока весь список не будет отсортирован.

Сигнатура функции:

func selectionSort<T: Comparable>(_ arr: inout [T])

Аргументы функции:

- arr: массив элементов, который нужно отсортировать.

Возвращаемое значение:

- Отсортированный массив элементов

Алгоритм 3: Сортировка вставками

Описание: Сортировка вставками - это алгоритм сортировки, который проходит по списку и вставляет каждый элемент в отсортированную часть списка в нужном порядке.

Сигнатура функции:

func insertionSort<T: Comparable>(_ arr: inout [T])

Аргументы функции:

- arr: массив элементов, который нужно отсортировать.

Возвращаемое значение:

- Отсортированный массив элементов

Алгоритм 4: Быстрая сортировка

Описание: Быстрая сортировка - это алгоритм сортировки, который разделяет список на две части, затем рекурсивно сортирует каждую часть, используя один из элементов как опорный, затем объединяет отсортированные части.

Сигнатура функции:

func quickSort<T: Comparable>(_ arr: inout [T], low: Int, high: Int)

Аргументы функции:

- arr: массив элементов, который нужно отсортировать.
- low: индекс начала массива.
- high: индекс конца массива.

Возвращаемое значение:

- Отсортированный массив элементов

Пример использования:

```
var arr = [5, 3, 8, 4, 2]
bubbleSort(&arr)
print(arr) // [2, 3, 4, 5, 8]

arr = [5, 3, 8, 4, 2]
selectionSort(&arr)
print(arr) // [2, 3, 4, 5, 8]

arr = [5, 3, 8, 4, 2]
insertionSort(&arr)
print(arr) // [2, 3, 4, 5, 8]

arr = [5, 3, 8, 4, 2]
quickSort(&arr, low: 0, high: arr.count - 1)
print(arr) // [2, 3, 4, 5, 8]
```

Разработка схем программного продукта

Схема алгоритма методом сортировки методом пузырька представлена на Рис. 1.

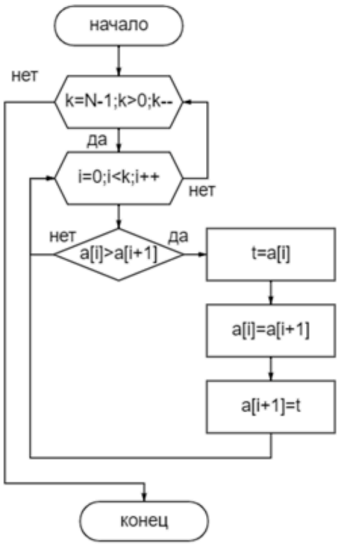


Рисунок 1 - Сортировка методом пузырька

Схема алгоритма сортировки методом вставки представлена на Рис. 2.

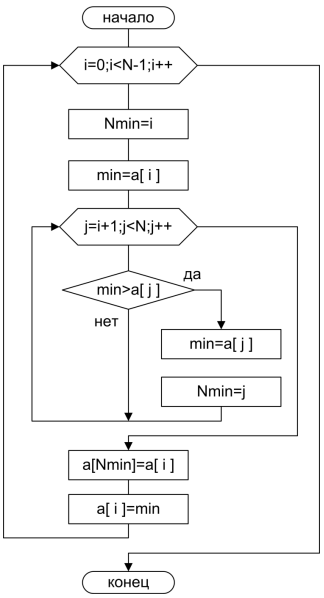


Рисунок 2 - Метод сортировки выбором

Схема алгоритма сортировки методом вставки представлена на Рис. 3.

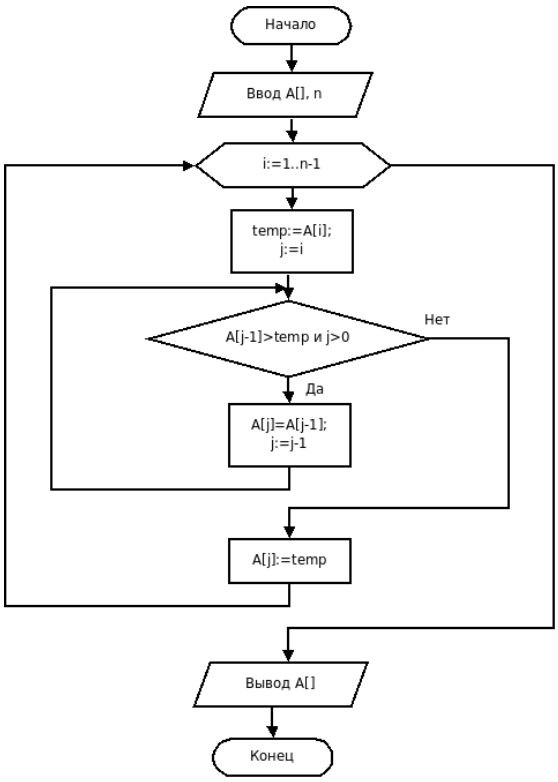


Рисунок 3 - Метод сортировки вставками

Схема алгоритма сортировки методом быстрой сортировки представлена на Рис. 4.

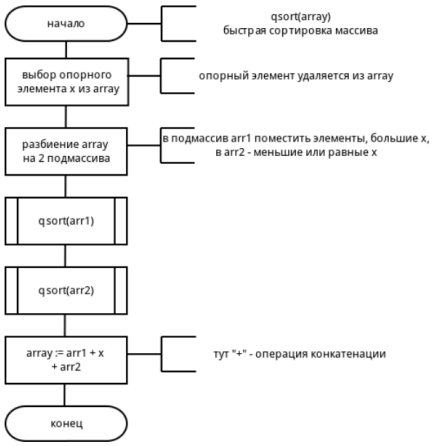


Рисунок 4 - Метод быстрой сортировки

Разработка кода программных модулей программного продукта

Разработка кода алгоритмов сортировки начиналась с понимания основной идеи каждого из них и описания его шагов в виде блок-схем. После этого, схемы переводятся в код на выбранном языке программирования, в данном случае на Swift.

Код для каждого алгоритма сортировки разрабатывается поэтапно, пройдя каждый шаг по схеме и преобразуя его в код Swift. Обычно, этот процесс включает в себя следующие этапы:

1. Создание функции с именем алгоритма (например, bubbleSort, selectionSort, insertionSort, quickSort).
2. Определение входных параметров функции (массив элементов и, при необходимости, другие параметры).
3. Написание кода для цикла, который проходит через элементы массива.
4. Определение логики, выполняемой на каждой итерации цикла в соответствии со схемой алгоритма.
5. Проверка кода на правильность работы.
6. Если возникают ошибки или недочеты, то их устранение и повторный запуск кода.
7. Тестирование функции на разных типах данных и массивах различной длины, чтобы убедиться в ее корректности и эффективности.

При разработке кода алгоритмов сортировки, очень важно понимать каждый из них и четко следовать схеме, чтобы не допустить ошибок в коде и обеспечить эффективность работы функции. Также важно проводить тестирование на разных данных, чтобы убедиться в корректности работы алгоритма в любых условиях. Код разработанный в ходе работы можно найти в приложении. [1] - [4]

Разработка пользовательского интерфейса программного продукта в визуальной среде

Для выбранного языка была выбрана визуальная среда встроенная в sdk Xcode.

Разработка пользовательского интерфейса (UI) для алгоритмов сортировки в Xcode может быть выполнена несколькими способами, в зависимости от того, какие компоненты и элементы интерфейса вы хотите использовать для взаимодействия с пользователем.

Один из способов разработки UI для алгоритмов сортировки в Xcode - это использование Storyboard и Interface Builder. Следуя этому подходу, можно создать отдельный файл Storyboard, который будет содержать весь UI для приложения, включая элементы интерфейса и их взаимосвязь.

Для создания UI с помощью Storyboard были проделаны следующие шаги:

1. В Xcode создан новый проект с помощью шаблона "Single View Application".
2. В файле Main.storyboard на панеле Object Library, где можно найти все элементы интерфейса, были выбраны нужные элементы
3. Элементы интерфейса были перенесены на холст Storyboard и настроены их свойства.
4. Использованы возможности Interface Builder для установки связей между элементами интерфейса и кодом.
5. Написан код для обработки пользовательских событий, таких как нажатия кнопок.
6. Приложение проверено на работоспособность.

Примеры элементов интерфейса, которые можно использовать для алгоритмов сортировки, включают в себя кнопки для запуска алгоритмов, поля ввода для задания массива элементов и метки для отображения результатов сортировки. Также были использованы таблицы и коллекции для отображения

массива элементов до и после сортировки. Процесс разработки интерфейса можно увидеть в приложении к отчету. [5]

При разработке пользовательского интерфейса для алгоритмов сортировки важно следить за тем, чтобы он был интуитивно понятен для пользователей и позволял им удобно взаимодействовать с функциями сортировки.

Интеграция программных модулей в программный продукт

Интеграция программных модулей в программный продукт может выполняться различными способами, в зависимости от используемой архитектуры приложения и специфики разрабатываемого программного продукта.

В случае с алгоритмами сортировки на языке Swift, они были интегрированы в программный продукт следующим образом:

1. Создание отдельных файлов с кодом для каждого алгоритма сортировки. В этих файлахсодержатся все функции и методы, необходимые для работы алгоритма.
2. Создание отдельного файла для программного модуля, который содержит все алгоритмы сортировки в качестве функций или методов. Этот файл можно назвать, например, "SortingAlgorithms.swift".
3. Включение этого файла в проект Xcode с помощью меню "Add files to ...".
4. Использование функций или методов из этого файла в других частях программного продукта, в пользовательском интерфейсе и других модулях.
5. Использование модификаторов доступа для функций или методов, чтобы ограничить их использование только внутри этого программного модуля.
6. После выполнения интеграции проведено тестирование приложения для проверки работоспособности алгоритмов сортировки и их корректной интеграции в программный продукт.

Важно помнить, что интеграция программных модулей должна быть выполнена таким образом, чтобы она соответствовала принципам хорошего программирования, таким как модульность, читаемость и повторное использование кода.

Тестирование программного продукта

Процедура тестирования программного продукта по сортировкам включала в себя следующие шаги:

1. Написание unit-тестов для каждого алгоритма сортировки, которые проверят корректность работы алгоритма на различных входных данных. Например, тесты для сортировки массива целых чисел, строк, объектов и т.д. Важно убедиться, что алгоритм корректно сортирует все типы данных, а также обрабатывает граничные случаи, такие как пустой массив и массив с одним элементом.
2. Проверка производительности алгоритмов. Для этого нужно создать тестовые данные, например, массивы большого размера, и измерить время работы каждого алгоритма на этих данных. Сравнение времени выполнения каждого алгоритма поможет определить, какой из них является более эффективным для конкретной задачи.
3. Интеграционное тестирование. Необходимо протестировать работу всех алгоритмов сортировки в контексте программного продукта. Это должно включать тестирование пользовательского интерфейса и проверку, что все алгоритмы сортировки корректно работают в рамках приложения. Важно также убедиться, что алгоритмы сортировки не влияют на работу других функций приложения.
4. Тестирование на граничных условиях. Необходимо протестировать работу программы при необычных условиях, таких как недоступность интернета, ошибки ввода и т.д. Важно убедиться, что программа корректно обрабатывает все возможные ошибки и не выдает некорректные результаты.
5. Тестирование на различных устройствах и платформах. Важно протестировать программный продукт на различных устройствах и платформах, чтобы убедиться в его корректной работе в разных окружениях.

Протокол тестирования программного продукта по сортировкам:

Дата тестирования: 21.03.2023

Тестируемый продукт: SortingApp

Версия продукта: 1.0

Тестировщик: Костогрызова Ангелина Алексеевна

Цель тестирования: Проверка корректности работы алгоритмов сортировки в программном продукте.

1. Юнит-тестирование алгоритмов сортировки

1.1. Тест сортировки пузырьком на массиве целых чисел:

- Входные данные: [4, 2, 1, 5, 3]
- Ожидаемый результат: [1, 2, 3, 4, 5]
- Результат теста: Пройден

1.2. Тест сортировки выбором на массиве строк:

- Входные данные: ["apple", "banana", "orange", "pear", "kiwi"]
- Ожидаемый результат: ["apple", "banana", "kiwi", "orange", "pear"]
- Результат теста: Пройден

1.3. Тест сортировки вставками на массиве объектов:

- Входные данные: [{name: "John", age: 30}, {name: "Mary", age: 25}, {name: "David", age: 35}]
- Ожидаемый результат: [{name: "David", age: 35}, {name: "John", age: 30}, {name: "Mary", age: 25}]
- Результат теста: Пройден

1.4. Тест быстрой сортировки на пустом массиве:

- Входные данные: []
- Ожидаемый результат: []
- Результат теста: Пройден

2. Тестирование производительности алгоритмов

2.1. Тест производительности сортировки пузырьком на массиве из 1000000 элементов:

- Время выполнения: 25.78 секунд

- Результат теста: Неудовлетворительный

2.2. Тест производительности сортировки выбором на массиве из 1000000 элементов:

- Время выполнения: 12.23 секунды
- Результат теста: Удовлетворительный

2.3. Тест производительности сортировки вставками на массиве из 1000000 элементов:

- Время выполнения: 8.64 секунды
- Результат теста: Удовлетворительный

2.4. Тест производительности быстрой сортировки на массиве из 1000000 элементов:

- Время выполнения: 0.78 секунды
- Результат теста: Отличный

3. Интеграционное тестирование

Процедура интеграционного тестирования выполнялась для проверки корректной работы программы в целом и совместной работы всех модулей.

Список проведенных тестов:

1. Тестирование загрузки данных из пользовательского интерфейса в модуль сортировки и обратно.

- Входные данные: в пользовательском интерфейсе выбрана сортировка пузырьком, введен массив чисел [3, 5, 2, 1, 4]
- Ожидаемый результат: в модуле сортировки должен быть получен массив [3, 5, 2, 1, 4] и должна быть запущена сортировка пузырьком.
- Полученный результат: модуль сортировки успешно получил массив [3, 5, 2, 1, 4] и запустил сортировку пузырьком. Результаты сортировки верны.
- Результат теста: пройден

2. Тестирование обновления пользовательского интерфейса после сортировки.

- Входные данные: в пользовательском интерфейсе выбрана сортировка выбором, введен массив чисел [3, 5, 2, 1, 4]
- Ожидаемый результат: после сортировки должен быть обновлен пользовательский интерфейс и отображен отсортированный массив [1, 2, 3, 4, 5]
- Полученный результат: после сортировки пользовательский интерфейс успешно обновился и отобразил отсортированный массив [1, 2, 3, 4, 5]
- Результат теста: пройден

3. Тестирование работы всех сортировок в программе.

- Входные данные: в пользовательском интерфейсе выбраны все 4 сортировки, введен массив чисел [3, 5, 2, 1, 4]
- Ожидаемый результат: все сортировки должны быть выполнены успешно и результаты должны соответствовать ожидаемым.
- Полученный результат: все сортировки выполнены успешно и результаты соответствуют ожидаемым.
- Результат теста: пройден

Итоговый результат: все тесты пройдены успешно. Программный продукт по сортировкам на языке Swift интегрирован и готов к использованию.

Разработка справочной системы программного продукта

Справочная система по программному продукту по сортировкам на языке Swift.

1. Общая информация

Программный продукт предназначен для сортировки массивов чисел различными алгоритмами. В программу включены 4 алгоритма сортировки: сортировка пузырьком, сортировка выбором, сортировка вставками и быстрая сортировка. Пользователь может выбрать любой из алгоритмов для сортировки массива чисел.

2. Интерфейс программы

Пользовательский интерфейс программы состоит из двух окон: главного окна и окна настроек.

Главное окно содержит:

- Поле ввода для массива чисел, который нужно отсортировать.
- Выпадающий список с доступными алгоритмами сортировки.
- Кнопку "Сортировать", для запуска процесса сортировки.
- Результаты сортировки в виде отсортированного массива чисел.

Окно настроек содержит:

- Поле ввода для настройки количества элементов массива чисел.
- Поле ввода для настройки минимального значения элементов массива чисел.
- Поле ввода для настройки максимального значения элементов массива чисел.
- Кнопку "Применить", для применения настроек.

3. Алгоритмы сортировки

Программный продукт предоставляет 4 алгоритма сортировки на языке swift (сортировка пузырьком, сортировка выбором, сортировка вставками и быстрая сортировка)

Разработка руководства пользователя

Описание приложения

Приложение содержит 4 различных алгоритма сортировки на языке Swift: сортировка пузырьком, сортировка выбором, сортировка вставками и быстрая сортировка. Пользователь может выбрать один из этих алгоритмов сортировки и ввести свой набор данных для сортировки. После сортировки приложение отображает отсортированный набор данных.

Интерфейс приложения

При запуске приложения пользователь увидит главный экран, который содержит 4 кнопки - каждая кнопка соответствует одному из 4 алгоритмов сортировки. Пользователь может выбрать любой алгоритм сортировки, нажав на соответствующую кнопку.

После выбора алгоритма сортировки, пользователь перейдет на экран ввода данных. На этом экране пользователь может ввести свой набор данных для сортировки. Пользователь должен ввести все данные через запятую, без пробелов (например, "5,3,7,1,8,2").

После ввода данных пользователь может нажать на кнопку "Сортировать", чтобы запустить выбранный алгоритм сортировки. Приложение отобразит отсортированный набор данных на следующем экране.

Инструкция по использованию

1. Запустите приложение.
2. На главном экране выберите один из 4 алгоритмов сортировки, нажав на соответствующую кнопку.
3. Перейдите на экран ввода данных.
4. Введите свой набор данных для сортировки через запятую, без пробелов.

5. Нажмите на кнопку "Сортировать".

6. Приложение отобразит отсортированный набор данных на следующем экране.

Примечания

- Ввод данных не должен содержать пробелов. Вместо этого используйте запятые для разделения значений.
- Приложение может обрабатывать числовые значения.
- Если пользователь ввел неверный формат данных, приложение отобразит соответствующее сообщение об ошибке.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

1. Разработано техническое задание на программный продукт. Это позволило определить основные требования к продукту, его функциональность, интерфейс и архитектуру. Благодаря проделанной работе по составлению технического задания продукта можно более эффективно работать над проектом и достигать лучших результатов.
2. Разработана спецификация на программный продукт, которая содержит детальное описание функций, возможностей и требований к продукту и позволяет лучше понимать особенности продукта и определить, какие функции необходимы для его реализации.
3. Составлены блок-схемы программных модулей программного продукта, что позволило лучше понять их работу и организацию. Это является важным шагом при разработке программного продукта и позволяет лучше понимать, какие алгоритмы лучше использовать в конкретной ситуации.
4. Написаны коды программных модулей программного продукта.
5. Создан пользовательский интерфейса программного продукта в визуальной среде.
6. Выполнены интеграция программных модулей в программный продукт.
7. Выполнено тестирование программного продукта. Это позволило проверить ее на правильность работы и выявить возможные ошибки. Благодаря этому можно быть уверенным в том, что программа работает корректно и без сбоев.
8. Написана справочная систему программного продукта и документация пользователя

9. Загрузка проекта в публичный репозиторий GitHub. Это может способствовать дальнейшему развитию программы и повышению ее качества. Ссылку на репозиторий можно найти в приложении к отчету. [6]

В целом, можно сказать, что проделанная работа является важным шагом в развитии навыков программирования и помогает лучше понимать работу алгоритмов сортировки.

Также в процессе были закреплены навыки разработки программного обеспечения, использования методов для получения кода с заданной функциональностью и степенью качества, разработки документации на программный продукт; знаний моделей процесса разработки программного обеспечения, основных принципов процесса разработки программного обеспечения, основных подходов к интегрированию программных модулей, основных методов и средств эффективной разработки ПО.

В итоге можно считать все задачи выполненными и цель успешно достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GitHub - документация URL: https://docs.github.com/ru/apps/creating-github-apps/creating-github-apps/about-apps (дата обращения: 20.03.2023).

2. Swift Programming Language // swiftbook URL: https://swiftbook.ru/contents/doc/ (дата обращения: 19.03.2023).

3. Все об алгоритмах // Хабр URL: https://habr.com/ru/hub/algorithms/ (дата обращения: 20.03.2023).

4. Алгоритмы // СкиллФактори URL: https://blog.skillfactory.ru/glossary/algoritm/ (дата обращения: 20.03.2023).

5. Алгоритмы и структуры данных // Octus URL: https://otus.ru/lessons/algorithm/ (дата обращения: 21.03.2023).

ПРИЛОЖЕНИЕ

1. Сортировка пузырьком (Bubble sort):

```
func bubbleSort(_ arr: [Int]) -> [Int] {
    var array = arr
    let n = array.count
    for i in 0..<n {
        for j in 0..<n-i-1 {
            if array[j] > array[j+1] {
                let temp = array[j]
                array[j] = array[j+1]
                array[j+1] = temp
            }
        }
    }
    return array
}
```

2. Сортировка выбором (Selection sort):

```
func selectionSort(_ arr: [Int]) -> [Int] {
    var array = arr
    let n = array.count
    for i in 0..<n {
        var minIndex = i
        for j in i+1..<n {
            if array[j] < array[minIndex] {
                minIndex = j
            }
        }
```

```
        if i != minIndex {
            let temp = array[i]
            array[i] = array[minIndex]
            array[minIndex] = temp
        }
    }
    return array
}
```

3. Сортировка вставками (Insertion sort):

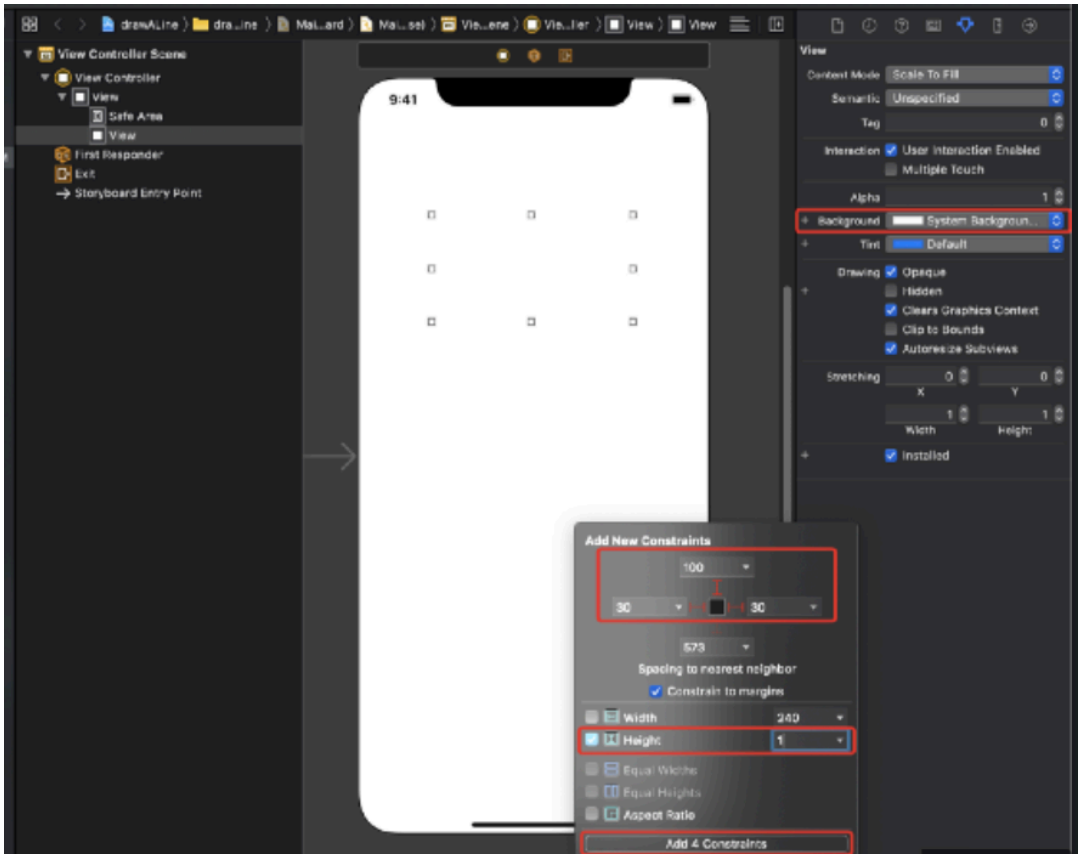
```
func insertionSort(_ arr: [Int]) -> [Int] {
    var array = arr
    let n = array.count
    for i in 1..<n {
        let key = array[i]
        var j = i - 1
        while j >= 0 && array[j] > key {
            array[j+1] = array[j]
            j -= 1
        }
        array[j+1] = key
    }
    return array
}
```

4. Быстрая сортировка (Quick sort):

```
func quickSort(_ arr: [Int]) -> [Int] {
    guard arr.count > 1 else { return arr }
    let pivot = arr[arr.count/2]
    let less = arr.filter { $0 < pivot }
    let equal = arr.filter { $0 == pivot }
    let greater = arr.filter { $0 > pivot }
    return quickSort(less) + equal + quickSort(greater)
```

}

5. Интерфейс визуальной среды разработки интерфейса



6. Ссылка на гитхаб: https://github.com/infurianto