МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА САПР

Отчет по лабораторной работе № 2
по дисциплине «Алгоритмы и структуры данных»
Тема: «Алгоритмы сортировки»
Вариант 1

Студент гр. 9302 Чугуй В. В.

Преподаватель Тутуева А. В.

Санкт-Петербург
2019

Оглавление

Цель работы

Получить практические навыки в разработке алгоритма и написании программы на языке C++ для знакомства с синтаксисом, а также правилами написания кода на языке C++.

Задание

Реализовать алгоритмы из списка.

Алгоритмы поиска и сортировки реализуются для целочисленного типа данных int, если не указанно иное

Список алгоритмов:

1. Двоичный поиск (BinarySearch)
2. Быстрая сортировка (QuickSort)
3. Сортировка вставками (InsertionSort)
4. Глупая сортировка (BogoSort)
5. Сортировка подсчётом (CountingSort) для типа char

Описание реализуемых алгоритмов

Рассмотрим алгоритм двоичного поиска. Данный алгоритм на вход получает на вход массив, его размер и ключ поиска (те то число, которое необходимо найти). Также этот массив должен быть отсортирован по возрастанию. И возвращает либо позицию данного числа в массиве, либо -1. Суть его заключается в том, что в цикле while(true) выбирается средняя позиция, путём нахождения среднего арифметического между левой и правой границей массива. Далее в структуре if else проверяется, если значение массива средней позиции меньше значения ключа, то левая граница изменяется на значение средней +1, если же значение массива больше ключа, то правая граница изменяется на значение среднего значения -1, иначе (те значение ключа и массива совпадают) то возвращается значение средней позиции. Если по итогу всех этих махинаций окажется так, что левая граница станет больше правой,

то возвращается значение -1, что информирует нас о том, что данный элемент в массиве отсутствует. И этот алгоритм продолжает работу в бесконечном цикле, пока не вернётся одно из значений.

Теперь рассмотрим алгоритм быстрой сортировки. На вход этот алгоритм получает массив и его размер. Суть его состоит в том, что мы разделяем массив на подмассивы и рекурсивно вызываем этот алгоритм, пока эти подмассивы не окажутся элементарными для сортировки. Для начала выбираются левая и правая граница, а также опорный элемент (он может быть любым эл-ом массива). Далее все элементы, которые будут меньше опорного элемента необходимо поместить слева от него, а большие – справа. После этого рекурсивно вызывается алгоритм для левой или правой части массива, ока каждый получаемый подмассив не будет содержать всего 1 эл-т, который является отсортированным.

Теперь рассмотрим сортировку вставками. Входные данные для данного алгоритма те же, что и у быстрой сортировки. Алгоритм помещает несортированный элемент в подходящее место на каждой итерации. Сортировка вставками работает аналогично сортировке игральных карт в руке: предполагаем, что первая карта уже отсортирована, тогда выбираем неотсортированную карту. Если неотсортированная карта больше, чем карта в руке, она помещается справа, в противном случае - слева. Таким же образом берутся и другие неотсортированные карты и перемещаются на свои места.

Ну а сейчас любимец публики и всех девятиклассников: глупая сортировка. Входные данные не изменились. Суть очень проста. Сначала проверяется отсортированность, если она нарушена, то эти элементы, которые нарушают правило меняются местами и проверка начинается сначала. Данный алгоритм имеет огромную временную сложность и в жизни не используется.

Последний алгоритм который рассматривается сегодня: сортировка подсчётом. Алгоритм сортирует эл-ты массива путем подсчета количества появлений каждого уникального эл-та в массиве. Счетчик хранится во

вспомогательном массиве, а сортировка выполняется путем сопоставления счетчика с индексом вспомогательного массива.

Оценка временной сложности каждого алгоритма

Алгоритм двоичного поиска:

Средний и худший случай: O(log[2]n)

Алгоритм быстрой сортировки:

Худший случай: O(n^2). Такое наблюдается когда опорный элемент является самым большим или самым маленьким элементом, те границей.

Лучший случай: O(n*log[2]n). Выбранный элемент является средним значением массива.

Средний случай: O(n*log[2]n)

Алгоритм сортировки вставками:

Худший случай: O(n^2). Массив отсортирован в обратном порядке.

Лучший случай: O(n). Массив уже отсортирован.

Средний случай: O(n^2)

Алгоритм глупой сортировки:

O(n*n!)

Алгоритм сортировки подсчётом:

O(max+n)

Сравнение временной сложности алгоритмов 2 и 3

Был произведен замер времени выполнения сортировки при случайных данных на размерности данных 10, 100, 1000, 10 000, 100000. Замер был средним для 10 запусков. Данные приведены в таблице. Значения времени представлены в миллисекундах.

Размерность	QuickSort	InsertionSort
10	0	0
100	0	0
1000	0,1	1,1
10000	1,3	95,2
100000	15	8706,2

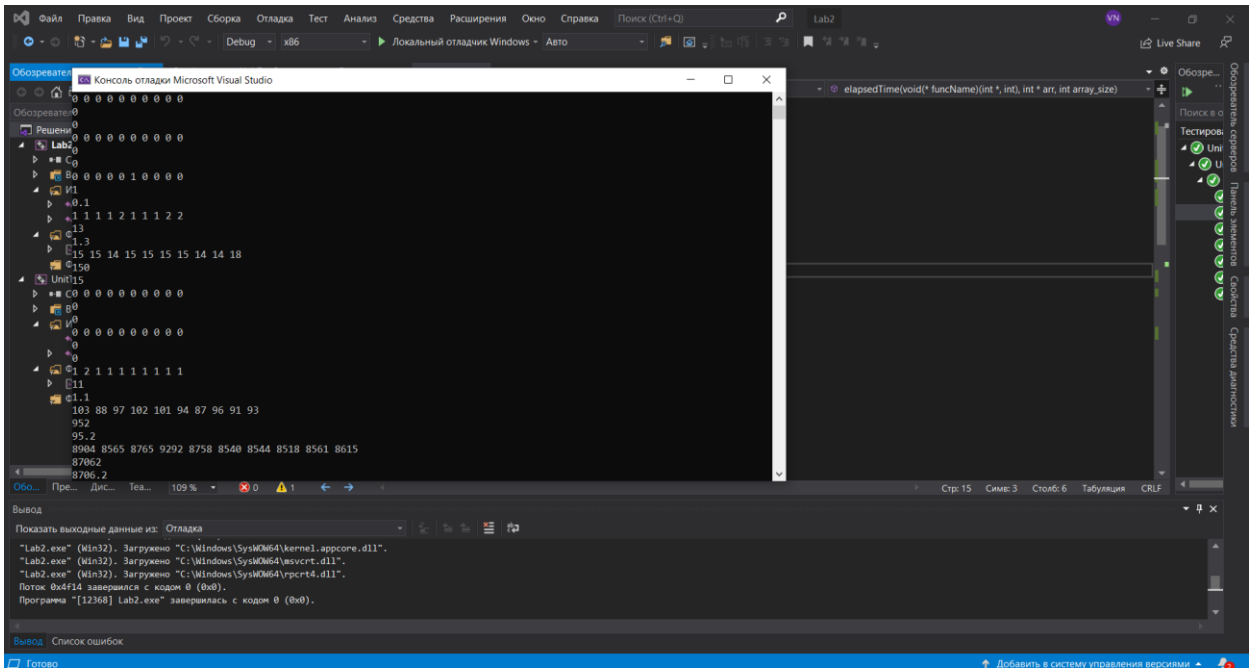
Из полученных данных видно, что на больших размерах массива алгоритм быстрой сортировки в разы быстрее сортировки вставками.

Описание реализованных unit-тестов

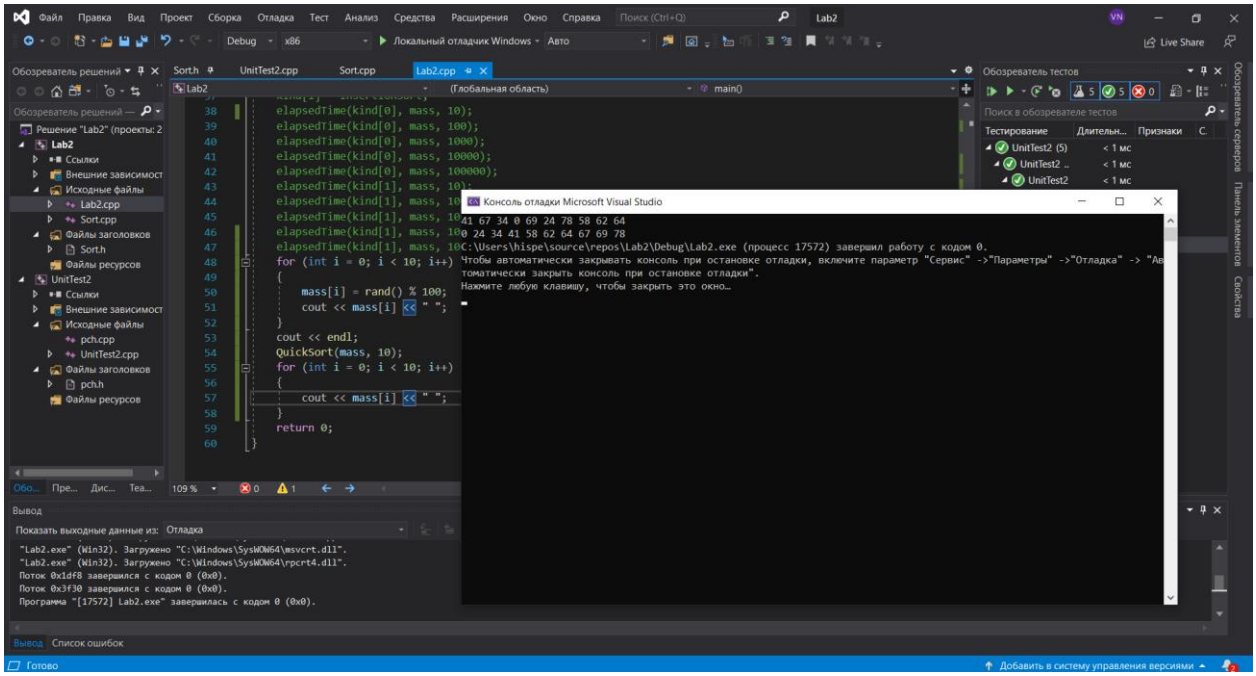
Для каждого алгоритма был написан свой unit тест. В юнит тесте для двоичного поиска находится массив и три проверки рабы этого алгоритма. Проверки нахождения элемента из середины списка, из конца и отсутствующего элемента. Для алгоритмов сортировки были созданы похожие друг на друга тесты. Разница была лишь в том, какой алгоритм сортировки используется. В каждом тесте создавался случайный массив и 10 итераций он сортировался определенным алгоритмом и в конце итерации проверялся на упорядоченность.

Пример работы программы

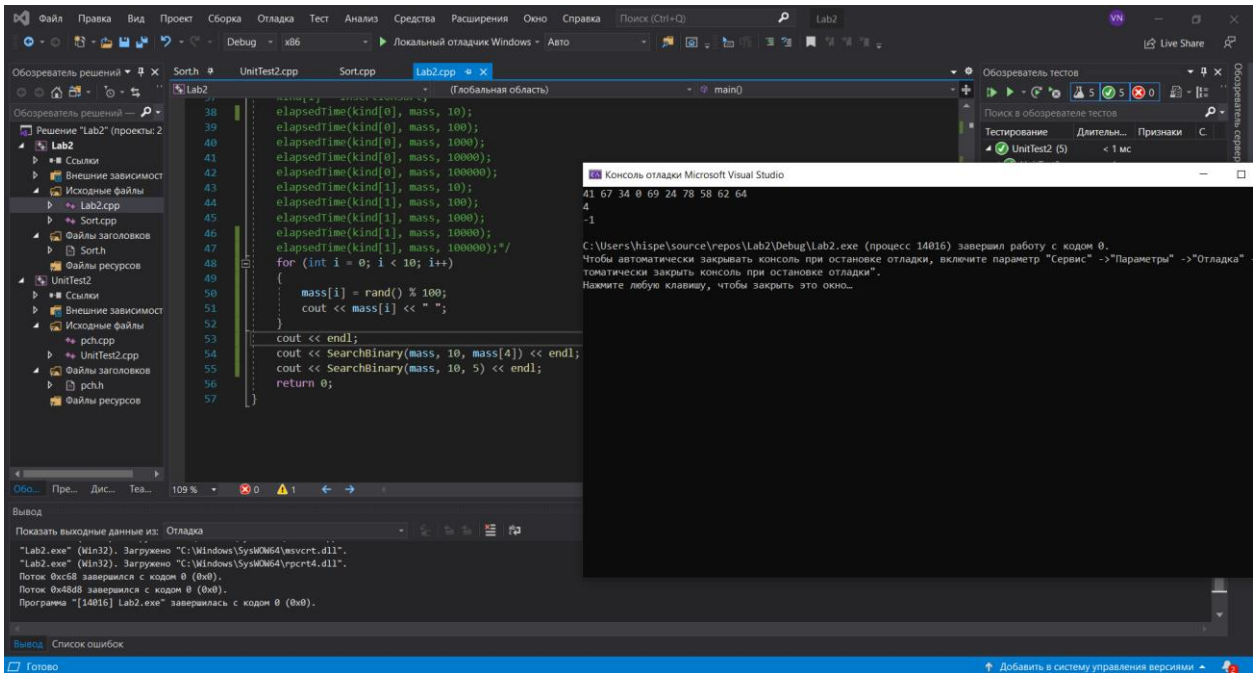
Пример замера времени:



Пример работы быстрой сортировки:



Пример работы двоичного поиска:



Заключение

Выводы:

При выполнении лабораторной работы были получены практические навыки в разработке алгоритмов решения задач, а также изучение синтаксиса языка C++.

Ссылка на программу

https://github.com/VolodyaZAVR/ALGSTR_Lab2