МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Трудоемкость сортировок

Работу выполнила: Овчинникова Анастасия, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Целью данной работы является изучение алгоритмов сортировки массивов, сравнительный анализ времени работы алгоритмов и анализ их трудоемкости.

Задачи лабораторной работы:

1. реализовать алгоритм сортировки вставками;
2. реализовать алгоритм быстрой сортировки;
3. реализовать алгоритм сортировки пузырьком с флагом;
4. провести оценку трудоемкости алгоритмов сортировки вставками, быстрой сортировки и сортировки пузырьком с флагом;
5. провести анализ времени работы алгоритмов размера от 100 до 2000 элементов.

Аналитическая часть

Под сортировкой в программировании понимается такая перестановка предметов, при которой они располагаются в порядке возрастания или убывания. Д. Кнут в [4] выделяет слудующие наиболее важные области применения сортировки.

1. Решение задачи группирования, когда нужно собрать вместе все элементы с одинаковыми значениями некоторого признака.

2. Поиск общих элементов в двух или более массивах.

3. Поиск информации по значениям ключей.

4. Провести оценку трудоемкости алгоритмов сортировки вставками, быстрой сортировки и пузырьковой сортировки.

5. Провести анализ времени работы алгоритмов размера от 100 до 1000 элементов.

Исторически сложилось так, что на сортировку уходит больше компьютерного времени, чем на остальные задачи. Поэтому сортировка - самая изученная задача в теории вычислительных систем. Известны десятки алгоритмов сортировки, большинство из которых имеют определенное преимущество над другими алгоритмами в определенных ситуациях. Далее будут рассмотрены алгоритм сортировки вставкам, алгоритм быстрой сортировки и алгоритм сортировки пузырьком.

Описание алгоритмов

Алгоритм сортировки вставками

Основная идея сортировки вставками состоит в том, что при добавлении нового элемента в уже отсортированный список его стоит сразу вставлять в нужное место. Сортировка вставками считает первый элемент любого списка отсортированным списком длины 1. Двухэлементный отсортированный список создается добавлением второго элемента исходного списка в нужное место одноэлементного списка, содержащего первый элемент. Теперь можно вставить третий элемент исходного списка в отсортированный двухэлементный список.

Этот процесс повторяется до тех пор, пока всеэлементы исходного списка не окажутся в расширяющейся отсортированной части списка.

Алгоритм быстрой сортировки

Быстрая сортировка является рекурсивным алгоритмом сортировки. Быстрая сортировка выбирает элемент списка, называемый осевым, а затем переупорядочивает список таким образом, что все элементы, меньшие осевого, оказываются перед ним, а большие элементы - за ним. В каждой из частей списка элементы не упорядочиваются. Если s — окончательное положение осевого элемента, то нам известно лишь, что все значения в позициях с первой по s — 1 меньше осевого, а значения с номерами от s + 1 до N больше осевого. Затем алгоритм быстрой сортировки вызывается рекурсивно на каждой из двух частей. При вызове процедуры Quicksort на списке, состоящем из одного элемента, он ничего не делает, поскольку одноэлементный список уже отсортирован.

Функция для расщепления списка PivotList берет в качестве осевого элемента первый элемент списка и устанавливает указатель pivot в начало списка. Затем она проходит по списку, сравнивая осевой элемент с остальными элементами списка. Обнаружив элемент, меньший осевого, она увеличивает указатель PivotPoint, а затем переставляет элемент с новым номером PivotPoint и вновь найденный элемент. После того, как сравнение части списка с осевым элементом уже произведено, список оказывается разбит на четыре части. Первая часть состоит из первого осевого — элемента списка. Вторая часть начинается с положения first + 1,кончается в положении PivotPoint и состоит из всех просмотренныэлементов, оказавшихся меньше осевого. Третья часть начинается вположении PivotPoint+1 и заканчивается указателем параметра цикла index. Оставшаяся часть списка состоит из еще не просмотренных значений.

Алгоритм сортировки пузырьком с флагом

Основной принцип сортировки пузырьком состоит в выталкивании маленьких значений на вершину списка в то время, как большие значения опускаются вниз. Алгоритм пузырьковой сортировки совершает несколько проходов по списку. При каждом проходе происходит сравнение соседних элементов. Если порядок соседних элементов неправильный, то они меняются местами. Каждый проход начинается с начала списка. Сперва сравниваются первый и второй элементы, затем второй и третий, потом третий и четвертый и так далее; элементы с неправильным порядком в паре переставляются. При обнаружении на первом проходе наибольшего элемента списка он будет переставляться со всеми последующими элементами пока не дойдет до конца списка. Поэтому при втором проходе нет необходимости производить сравнение с последним элементом. При втором проходе второй по величине элемент списка опустится во вторую позицию с конца. При продолжении процесса на каждом проходе по крайней мере одно из следующих по величине значений встает на свое место. При этом

меньшие значения тоже собираются наверху. Если при каком-то проходе не произошло ни одной перестановки элементов, то все они стоят в нужном порядке, и исполнение алгоритмаможно прекратить. Стоит заметить, что при каждом проходе ближе к своему месту продвигается сразу несколько элементов, хотя гарантированно занимает окончательное положение лишь один.

Модель вычислений

Трудоемкость алгоритма измеряется в количестве операций, которые необходимо выполнить.

Введем модель вычислений, используемую при оценке трудоемкости алгоритмов.

1. Базовые операции трудоемкости 1: +, -, *, /, =, ==, <=, >=, !=, >, <, +=, -=, |=, *=, проход по адресу.

2. Трудоемкость цикла вычисляется по формуле $f = 2 + N(2 + f_{body})$, где N - число повтрорений цикла, f_{body} - трудоемкость тела цикла.

3. Трудоемкость условного перехода равна 0, стоимость вычисления условия остается.

4. Вызов метода объекта класса имеет трудоемкость 1.

5. Объявление переменной/массива/структуры без определения имеет трудоемкость 0.

6. Условный оператор без условий внутри имеет трудоемкость 0.

7. Логические операции имеют трудоемкость 1.

Конструкторская часть

Требования к программе

Программа представляет собой консольное приложение с меню для выбора пользователем необходимого действия. Меню содержит следующие пункты:

- "1 - сортировка случайного масива с помощью алгоритма сортировки вставками";
- "2 - сортировка случайного масива с помощью алгоритма сортировки пузырьком";
- "3 - сортировка случайного масива с помощью алгоритма быстрой сортировки";
- "4 - временной анализ";
- "5 - тесты".

Для выбора соответствующих пунктов меню пользователь вводит соответствующую цифру (1-5). При вводе любого другого символа программа должна корректно завершаться. Для пунктов 1 - 3 программа просит пользователя ввести желаемую длину случайного массива. Затем она генерирует случайный массив указанной длины с помощью втроенной функции rand().

Далее в данном разделе будут рассмотрены схемы выбранных алгоритмов сортировки.

Схемы алгоритмов

Схема алгоритма сортировки вставками представлена на рисунке 1. Схема алгоритма пузырьковой сортировки представлена на рисунке 2. Схема алгоритма быстрой сортировки представлена на рисунке 3, а подпрограмма pivotList для разбиение массива, используемая в быстрой сортировке, представлена на рисунке 4. В алгоритмах используется функция std::swap(T& a, T& b) из C++ библиотеки <utility>, которая меняет местами значения a и b.

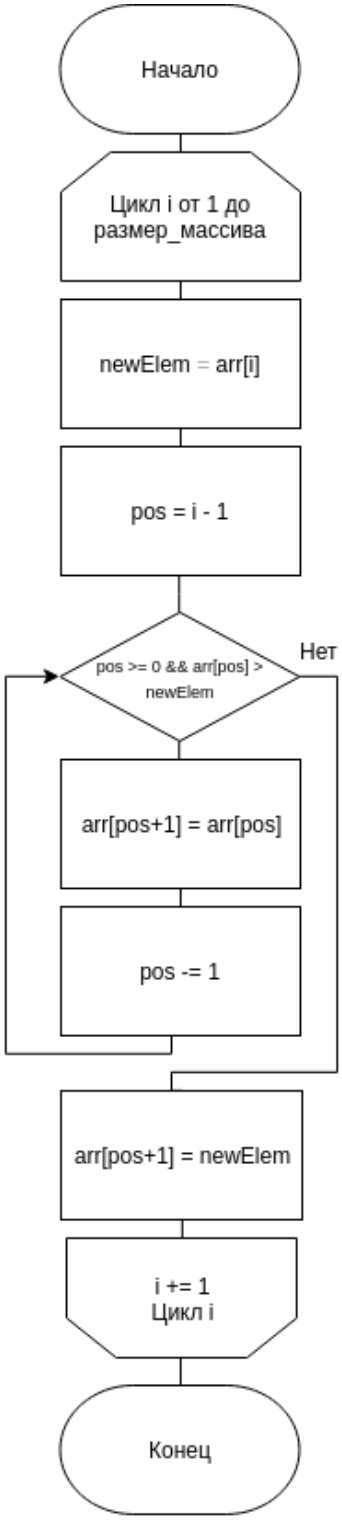


Рис. 1: Схема алгоритма сортировки вставками

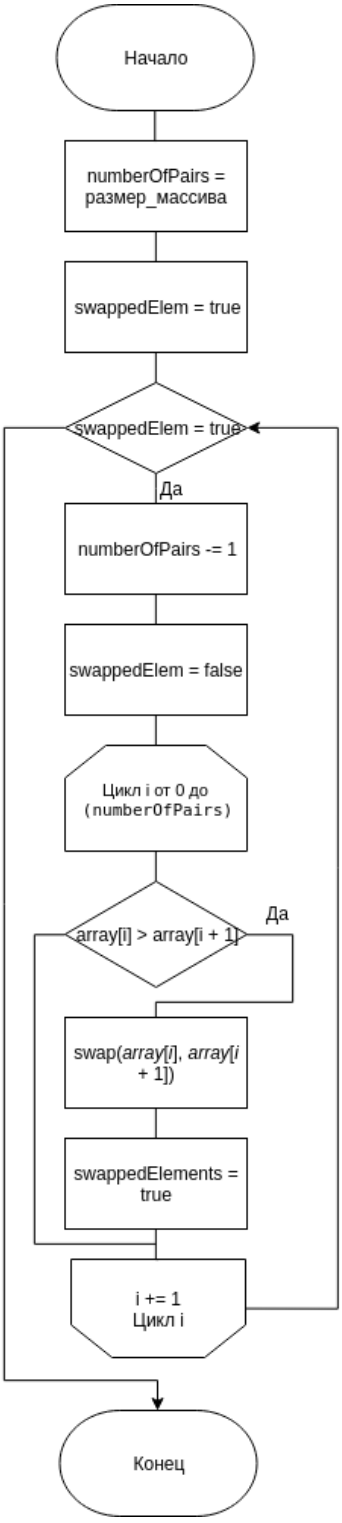


Рис. 2: Схема алгоритма пузырьковой сортировки

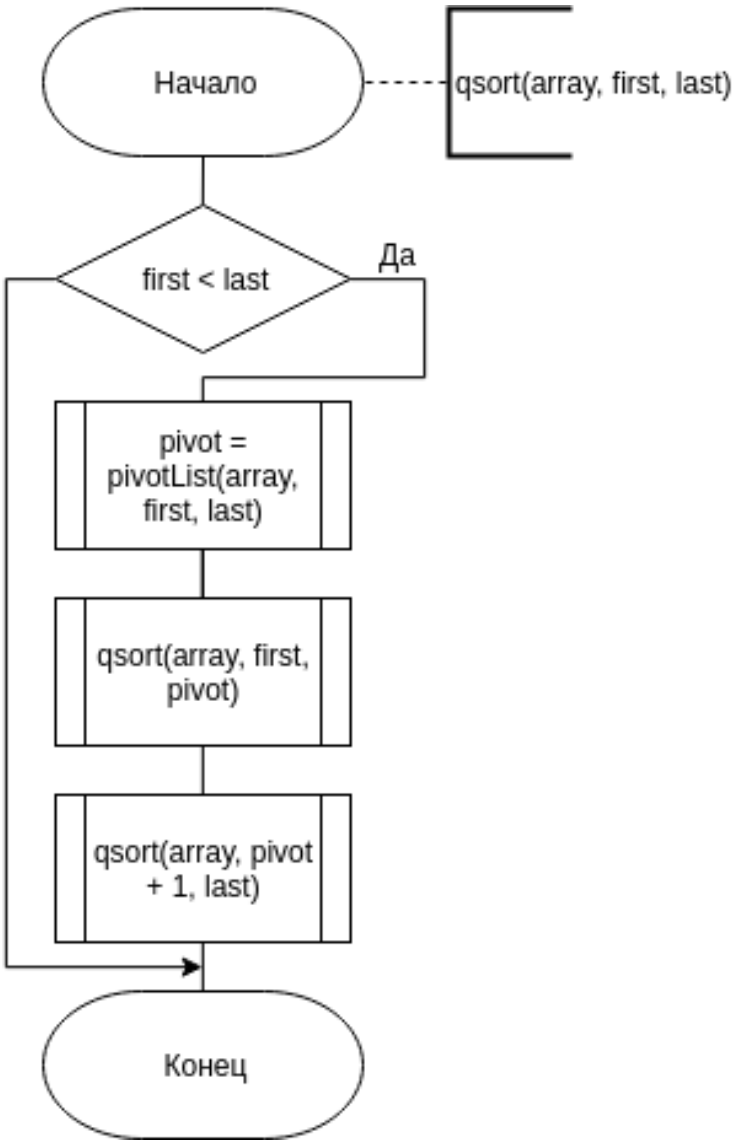


Рис. 3: Схема алгоритма быстрой сортировки

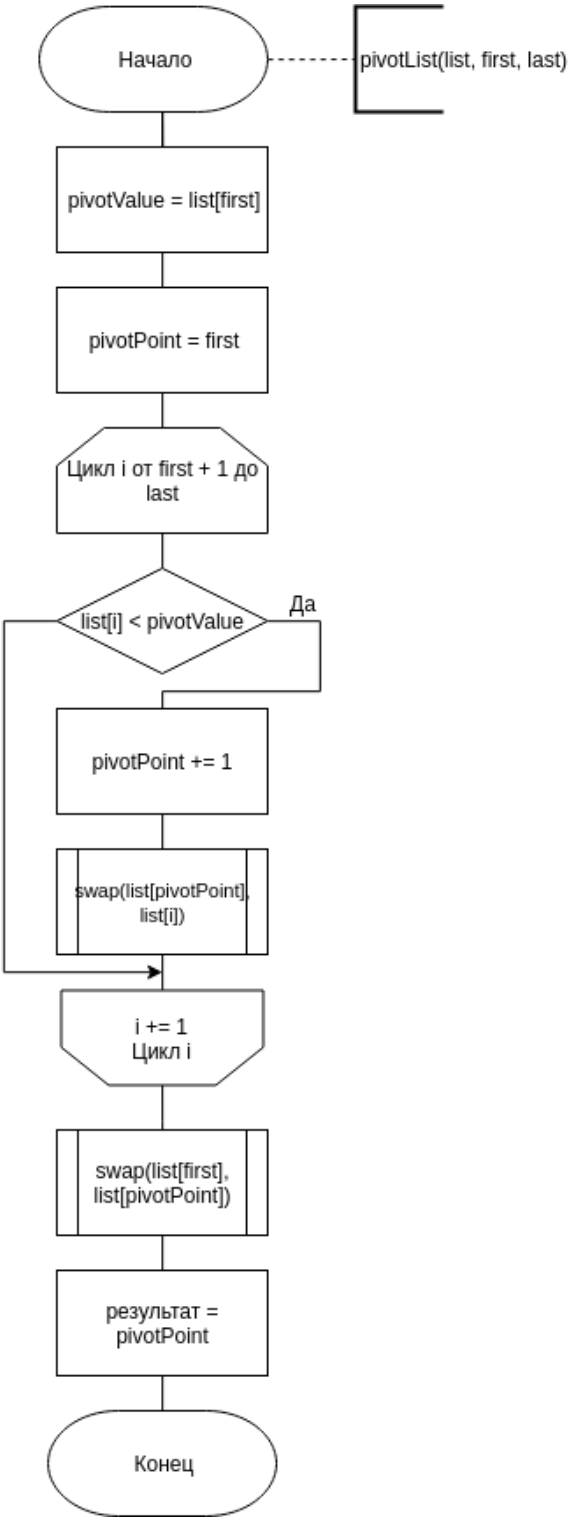


Рис. 4: Схема подпрограммы pivotList

Технологическая часть

Выбор языка программирования

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

- язык C++ имеет высокую вычислительную производительность;
- язык C++ поддерживает различные стили программирования;
- в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Для замеров времени использовалась функция clock() модуля ctime. Эта функция возвращает количество временных тактов, прошедших с начала запуска программы. С помощью макроса CLOCKS_PER_SEC можно узнать количество пройденных тактов за 1 секунду.

Сведения о модулях программы

Программа состоит из следующих файлов:

- myarray.h, myarray.cpp - заголовочный файл и файл, в котором расположена реализация алгоритмов сортировки;
- main.cpp - главный файл программы, в котором расположена реализация меню;
- testsorting.h, testsorting.cpp - файл и заголовочный файл, в котором расположена реализация тестов.

Листинги кода алгоритмов

Далее представлены листинги кода алгоритмов сортировки вставками (листинг 1), пузырьковой сортировки с флагом (листинг 2) и быстрой сортировки

(листинг 3). Листинг кода алгоритма расщипления списка, используемого в алгоритме быстрой сортировки, представлен в листинге 4.

Листинг 1: Алгоритм сортировки вставками

```
void MyArray::insertionSort()
{
    for (int i = 2; i < size; ++i)
    {
        int newElement = array[i];
        int location = i - 1;

        while (location >= 1 && array[location] > newElement)
        {
            array[location + 1] = array[location];
            location -= 1;
        }
        array[location + 1] = newElement;
    }
}
```

Листинг 2: Алгоритм пузырьковой сортировки с флагом

```
int numberOfPairs = this->size;
bool swappedElements = true;
while (swappedElements)
{
        numberOfPairs -= 1;
        swappedElements = false;
        for (int i = 0; i < numberOfPairs; ++i)
        {
                if (this->array[i] > this->array[i + 1])
                {
                        std::swap(array[i], array[i + 1]);
                        swappedElements = true;
                }
        }
}
```

Листинг 3: Алгоритм быстрой сортировки

```
void MyArray::quickSort(int first, int last)
{
    int pivot;
    if (first < last)
    {
        pivot = pivotList(array, first, last);
        quickSort(first, pivot - 1);
        quickSort(pivot + 1, last);
    }
}
```

Листинг 4: Подпрограмма разбиения массива

```
int MyArray::pivotList(int* list, int first, int last)
{
```

```
3       int p = last;
4       int firstHigh = first;
5       for (int i = first; i < last; ++i)
6       {
7           if (list[i] < list[p])
8           {
9               std::swap(list[i], list[firstHigh]);
10              firstHigh++;
11          }
12      }
13      std::swap(list[p], list[firstHigh]);
14      return firstHigh;
15  }
```

Расчет трудоемкости

Сортировка вставками

Обозначим за N длину массива, поданного на вход алгоритму сортировки. Рассчитаем трудоемкость алгоритма сортировки вставками. Для этого необходимо рассмотреть лучший, худший и произвольный (массив заполнен случайным образом) случаи.

Самым благоприятным случаем является отсортированный массив. При этом тело внустреннего цикла while не выполняется ни разу, на каждой итерации цикла for проверяется лишь условие цикла while. Всего внешний цикл будет выполняться $N-1$ раз. Тогда получим, что трудоемкость сортировки вставками f_{best} в лучшем случае равна:

$f_{best} = 2 + (N-1)(2+4+3+4) = 13N - 11 \approx O(N)$

Наихудшим случаем является массив, отсортированный в порядке, обратном нужному. При этом каждый новый элемент сравнивается со всеми в отсортированной последовательности. Это означает, что все внутренний цикл while состоит из i итераций. Тогда получим, что трудоемкость сортировки вставками f_{worst} в худшем случае равна:

$$f_{worst} = 1 + 2N + (N-1)2 + (N-1)2 + (N-1)3 + 4\sum_{i=1}^{n-1} i + 6\sum_{i=1}^{n-1}(i-1) + 6\sum_{i=1}^{n-1}(i-1) = 9N - 6 + 4\frac{N(N-1)}{2} + 12(\frac{N(N-1)}{2} - 1) \approx \frac{N^2}{2} \approx O(N^2)$$

Быстрая сортировка

Трудоемкость быстрой сортировки в худшем сулучае:
$f_{worst} = \frac{N^2-N}{2} \approx O(N^2)$ [1]
Трудоемкость быстрой сортировки в лучшем сулучае:
$f_{best} = \approx O(N \log_2 N)$ [3]

Сортировка пузырьком с флагом

Трудоемкость сортировки пузырьком с флагом в худшем сулучае:
$f_{worst} = \frac{1}{2}N^2 \approx O(N^2)$ [1]
Трудоемкость сортировки пузырьком с флагом в лучшем сулучае:
$f_{best}(N-1) \approx O(N)$ [1]

Тесты

Тестирование проводилось с помощью модуля QtTest. Сначала каждый алгоритм сортировки тестировался по оддельности на заранее заготовленном наборе тестовых данных. После этого генерировались случайные массивы целых чисел в диапазоне от -100 до 100 размером от 0 до 1000 с шагом 100 (для каждой длины массива генерировалось 10 случайных массивов). Проводилась сортировка этих массивов с помощью алгоритмов сортировки вставками, пузырьковой и быстрой сортировки и сравнивались результаты работы этих трех алгоритмов.

Использованный набор тестовых данных приведен в таблице 1.

Все проведенные тесты были пройдены.

Таблица 1: Набор тестовых данных

Исходный результат	Ожидаемый результат
1, 1	1, 1
1	1
1, 1, 1	1, 1, 1
-1, -1, -1	-1, -1, -1
1, 2, 3	1, 2, 3
3, 2, 1	1, 2, 3
1, 2, 1	1, 1, 2
2, 2, 1	1, 2, 2
1, 1, 2	1, 1, 2
1, 1, 2, 2	1, 1, 2, 2
1, 2, 1, 2	1, 1, 2, 2
2, 2, 1, 1	1, 1, 2, 2
2, 1, 2, 1	1, 1, 2, 2
-1, -2, -3	-3, -2, -1
-3, -2, -1	-3, -2, -1
-1, -2, -1	-2, -1, -1
-2, -2, -1	-2, -2, -1
-1, -1, -2	-2, -1, -1
-1, -1, -2, -2	-2, -2, -1, -1
-1, -2, -1, -2	-2, -2, -1, -1
-2, -2, -1, -1	-2, -2, -1, -1
-2, -1, -2, -1	-2, -2, -1, -1
-1, 2, 3	-1, 2, 3
-1, -2, 3	-2, -1, 3
-1, 2, -1	-1, -1, 2
-2, -2, 1	-2, -2, 1
-1, 1, -2	-2, -1, 1
-1, 0, 2	-1, 0, 2
2, 0, -1	-1, 0, 2
0, -1, 2	-1, 0, 2
2, -1, 0	-1, 0, 2
-1, 1, -2, 2	-2, -1, 1, 2
-1, 2, -1, 2	-1, -1, 2, 2
-2, -2, 1, 1	-2, -2, 1, 1
2, -1, -2, -1	-2, -1, -1, 2

Исследовательская часть

Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Сравнение времени работы алгоритмов сортировок в лучших, средних и худших случаях на массивах размерности от 0 до 1000 с шагом 50 для.

Использовались три типа массивов

1. Отсортированный по возрастанию.
2. Отсортированный по убыванию.
3. Заполненный случайным образом.

Измерения проводились на компьютере HP Pavilion Notebook на базе Intel Core i5-7200U, 2.50 Гц с 6 Гб оперативной памяти под управлением операционной системы Linux Mint.

Сравнительный анализ на материале экспериментальных данных

Условные обозначения, используемые в таблицах и на графиках:

1. insBest - время работы алгоритма сотрировки вставками в лучшем случае в секундах;
2. insWorst - время работы алгоритма сотрировки вставками в худшем случае в секундах;
3. insAvg - время работы алгоритма сотрировки вставками в произвольном случае в секундах;
4. bubbleBest - время работы алгоритма сотрировки пузырьком с флагом в лучшем случае в секундах;
5. bubbleWorst - время работы алгоритма сотрировки пузырьком с флагом в худшем случае в секундах;

6. bubbleAvg - время работы алгоритма сотрировки пузырьком с флагом в произвольном случае в секундах;

7. qsortBest - время работы алгоритма быстрой сотрировки в лучшем случае в секундах;

8. qsortWorst - время работы алгоритма быстрой сотрировки в худшем случае в секундах;

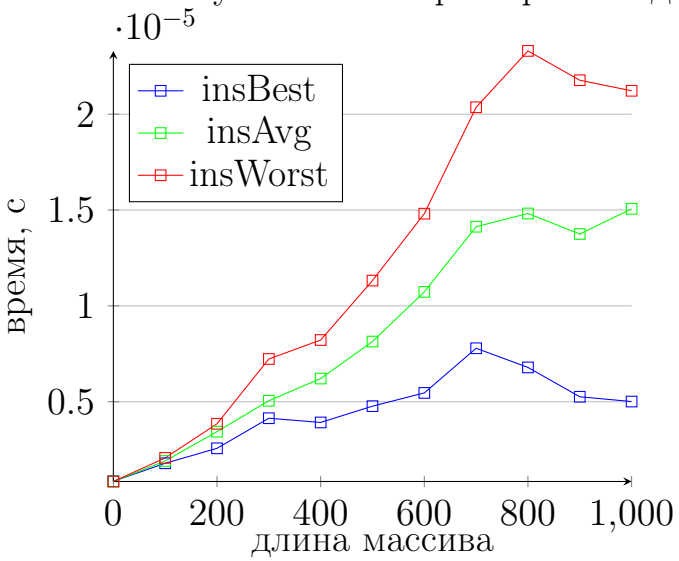
9. qsortAvg - время работы алгоритма быстрой сотрировкив произвольном случае в секундах.

Результаты замеров времени для алгоритма сортировки вставками для лучшего случая (Лучший), для худщего случай (Худший) и для произвольного случая (Произвольный) приведены в таблице 2. Здесь и далее все указанное время измеряется в секундах. На графиках изображена зависимость времени работы алгоритма сортировок от длины входного массива. Данные из таблицы изображены на рисунке 5.

Таблица 2: Результаты замеров времени для алгоритма сортировки вставками

Размер массива	Лучший, с	Проивольный, с	Худший, с
0	0.0000008500	0.0000008400	0.0000008400
100	0.0000017900	0.0000019100	0.0000020700
200	0.0000025700	0.0000034400	0.0000038500
300	0.0000041400	0.0000050500	0.0000072300
400	0.0000039200	0.0000062100	0.0000082200
500	0.0000047700	0.0000081400	0.0000113200
600	0.0000054600	0.0000107300	0.0000148100
700	0.0000077900	0.0000141300	0.0000203600
800	0.0000067900	0.0000148200	0.0000233000
900	0.0000052600	0.0000137500	0.0000217700
1000	0.0000050100	0.0000150600	0.0000212200
1100	0.0000048600	0.0000134500	0.0000210100
1200	0.0000054700	0.0000152500	0.0000245900
1300	0.0000057900	0.0000173200	0.0000281000
1400	0.0000061300	0.0000197300	0.0000325900
1500	0.0000065600	0.0000214900	0.0000363700
1600	0.0000070000	0.0000238900	0.0000427100
1700	0.0000073900	0.0000269700	0.0000456200
1800	0.0000078700	0.0000294500	0.0000507100
1900	0.0000081300	0.0000325000	0.0000558400
2000	0.0000087400	0.0000371100	0.0000628500

Рис. 5: Результаты замеров времени для алгоритма сортировки вставками



Результаты замеров времени для алгоритма сортировки пузырьком с флагом для лучшего случая (Лучший), для худщего случай (Худший) и для произвольного случая (Произвольный) приведены в таблице 3. Данные из таблицы изображены на рисунке 6.

Рис. 6: Результаты замеров времени для алгоритма пузырьковой сортировки с флагом

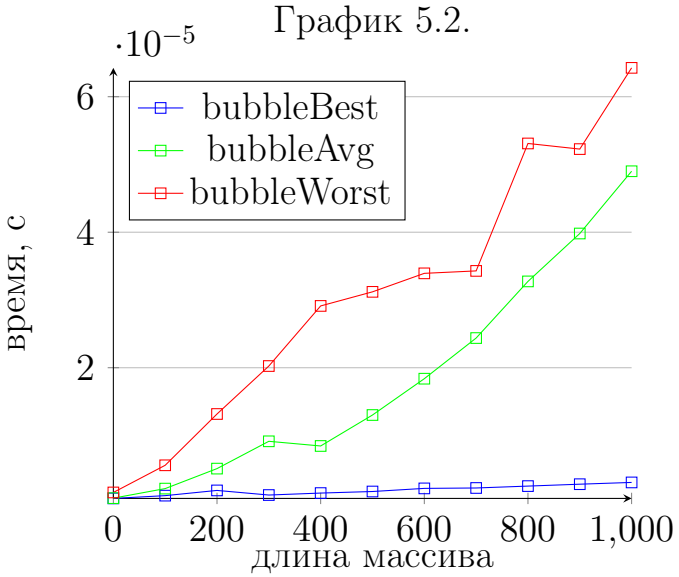


Таблица 3: Результаты замеров времени для алгоритма пузырьковой сортировки с флагом

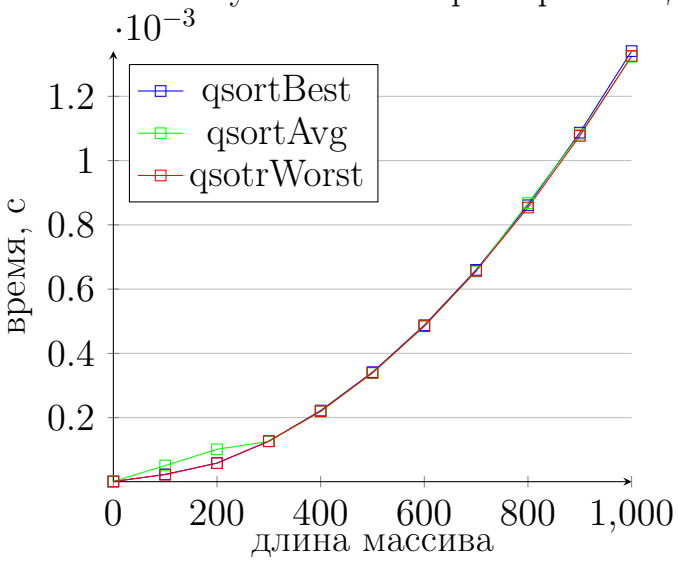
Размер массива	Лучший, с	Произвольный, с	Худший, с
0	0.0000007100	0.0000007700	0.0000007300
100	0.0000011400	0.0000020400	0.0000023100
200	0.0000017900	0.0000053300	0.0000064000
300	0.0000023600	0.0000100000	0.0000128400
400	0.0000027700	0.0000168100	0.0000212400
500	0.0000032700	0.0000254400	0.0000323000
600	0.0000037700	0.0000358200	0.0000477400
700	0.0000029000	0.0000445000	0.0000452700
800	0.0000024900	0.0000379400	0.0000437300
900	0.0000028000	0.0000420900	0.0000510500
1000	0.0000030400	0.0000524300	0.0000635800
1100	0.0000032600	0.0000645900	0.0000753400
1200	0.0000035900	0.0000756000	0.0000890400
1300	0.0000038500	0.0000890400	0.0001042100
1400	0.0000041100	0.0001022200	0.0001215900
1500	0.0000043400	0.0001169500	0.0001385000
1600	0.0000044700	0.0001389200	0.0001572500
1700	0.0000048900	0.0001512100	0.0001767500
1800	0.0000052000	0.0001717200	0.0001978200
1900	0.0000054200	0.0001893500	0.0002204700
2000	0.0000057600	0.0002104600	0.0002503500

Результаты замеров времени для алгоритма быстрой сортировки для лучшего случая (Лучший), для худщего случай (Худший) и для произвольного случая (Произвольный) приведены в таблице 4. Данные из таблицы изображены на рисунке 7.

Таблица 4: Результаты замеров времени для алгоритма быстрой сортировки

Размер массива	Лучший, с	Произвольный, с	Худший, с
0	0.0000008700	0.0000008300	0.0000008700
100	0.0000250300	0.0000247900	0.0000253200
200	0.0000583800	0.0000581400	0.0000587400
300	0.0001268200	0.0001257600	0.0001269300
400	0.0002210300	0.0002190900	0.0002201700
500	0.0003414300	0.0003385800	0.0003394500
600	0.0004874400	0.0004838100	0.0004850700
700	0.0006600400	0.0006550900	0.0006561900
800	0.0008590600	0.0008583700	0.0008536500
900	0.0010836400	0.0010747500	0.0010767500
1000	0.0013424000	0.0013239100	0.0013358500
1100	0.0016167300	0.0016061200	0.0016179100
1200	0.0019166800	0.0018998800	0.0019019000
1300	0.0022445600	0.0022251900	0.0022278300
1400	0.0026313700	0.0025909200	0.0025871600
1500	0.0029849100	0.0029554200	0.0029589000
1600	0.0034023600	0.0033690600	0.0033839600
1700	0.0038231100	0.0038000500	0.0037939800
1800	0.0042883000	0.0042634700	0.0042723000
1900	0.0047766200	0.0047310700	0.0047571000
2000	0.0052843800	0.0052326100	0.0052383000

Рис. 7: Результаты замеров времени для алгоритма быстрой сортировки



Далее проведем сравнение трех разных сортировок между собой.

На рисунке 8 представлены зависимости времени работы алгоритма сортировок от длины массива для лучших случаев.

На рисунке 9 представлены зависимости времени работы алгоритма сортировок от длины массива для случайных случаев.

Рис. 8: Сравнение алгоритмов сортировок в лучших случаях

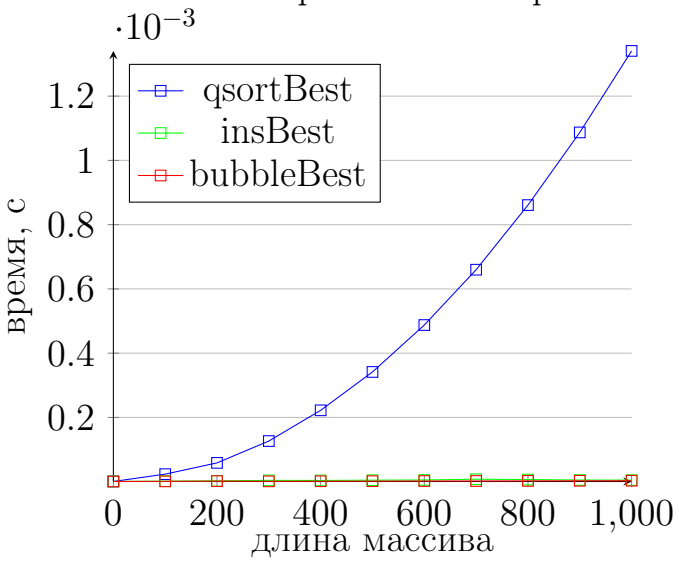
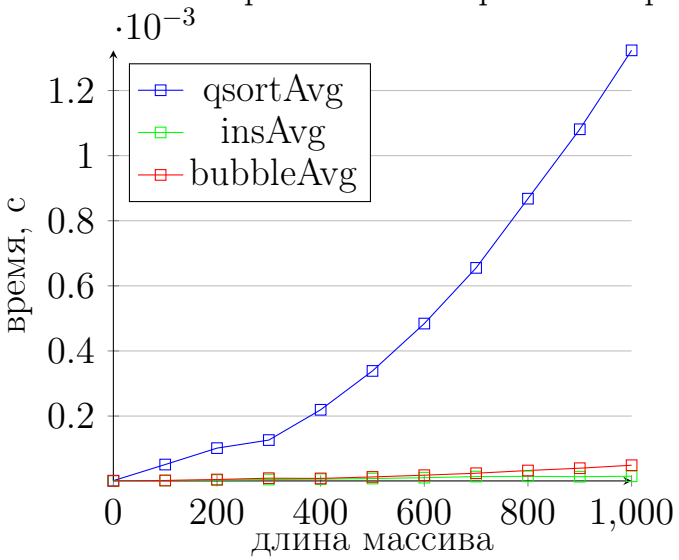
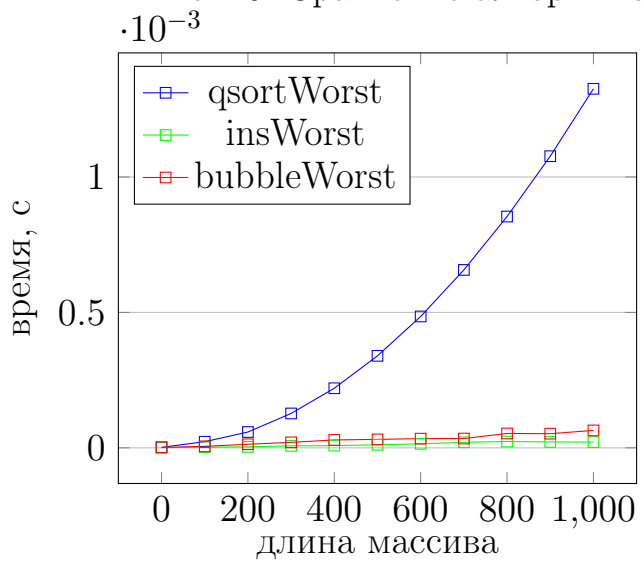


Рис. 9: Сравнение алгоритмов сортировок в проивзольных случаях



На рисунке 10 представлены зависимости времени работы алгоритма сортировок от длины массива для худших случаев.

Рис. 10: Сравнение алгоритмов сортировки в худших случаях



Выводы

В результате проведенных экспериментов было установлено, что

1. Быстрая сортировка является самой медленной во всех рассмотренных случаях.

2. Время работы алгоритма сортировки вставками для худших и лучших случаев сильно различается.

3. Время работы алгоритма сортировки пузырьком с флагом для худших и лучших случаев сильно различается.

4. Время работы алгоритма быстрой сортировки для худших и лучших случаев не различается.

Такой результат для быстрой сортировки можно объяснить несколькими причинами:

1. Быстрая сортировка является рекурсивной и это сильно замедляет ее работу.

2. Данная конкретная реализация алгоритма разбиения массива для быстрой сортировки использует в качестве опорного элемента первый элемент массива. Быстрая сортировка показывает лучшие результаты, когда опорный элемент выбирается случайным образом [1].

Заключение

В ходе данной лабораторной работы были изучены и реализованы три алгоритма сортировок: сортировка вставками, быстрая сортировка и пузырьковая сортировка с флагом.

Было проведено исследование времени работы трех алгоритмов сортировок (сортировка вставками, быстрая сортировка и пузырьковая сортировка с флагом).

Было проведено сравнение времени работы трех алгоритмов сортировок (сортировка вставками, быстрая сортировка и пузырьковая сортировка с флагом).

Была провеедена оценка трудоемкости трех алгоритмов сортировок (сортировка вставками, быстрая сортировка и пузырьковая сортировка с флагом).

Список литературы

1. Дж. Маконелл. Анализ алгоритмов. Активный обучающий подход. - М.:Техносфера 2009.

2. С. Скиена. Алгоритмы. Руководство по разработке. - 2-е изд.: Пер. с англ. - СПб.: БХВ-Петербург, 2011.

3. Hoare, C. a. R. Quicksort (англ.) // The Computer Journal: journal. — 1962.

4. Д. Кнут. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. : Пер. с англ. - М. : ООО "И. Д. Вильямс 2007.