Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент	ИУ7-54Б		С. Д. Параскун
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Л. Л. Волкова
		Подпись, дата	И. О. Фамилия

Москва, 2021 г.

Содержание

Введение

В настоящее время компьютерные системы оперируют большими объемами данных. Все они могут иметь разное представление, однако порядок обработки нередко зависит от размещения данных относительно друг друга. Для оптимизации этого процесса используется предварительная сортировка.

Сортировка - алгоритм упорядочивания элементов в порядке возрастания или убывания. Она применяется в различных областях, будь то группировка данных, поиск общих элементов в нескольких последовательностях или же поиск какой-либо информации.

Алгоритмы сортировки характеризуются временем выполнения и объемом затрачиваемой памяти. Они определяются как зависимость от длины сортируемой последовательности данных. Также выделяют устойчивую и неустойчивую сортировки. Устойчивая не меняет положение данных, с одинаковым значением.

Целью данной лабораторной работы является анализ различных видов сортировки. Для достижения поставленной цели необходимо выполнить следующие задачи:

- Изучить три выбранных алгоритма сортировки;
- Составить схемы данных алгоритмов;
- Реализовать разработанные алгоритмы сортировок;
- Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- Описать и обосновать полученные результаты.

1. Аналитический раздел

В данном разделе будут представлены описания алгоритмов выбранных сортировок: пузырьком, вставками и выбором.

1.1 Сортировка пузырьком

Сортировка пузырьком или простыми обменами [1] (англ. *bubble sort*) - простой алгоритм сортировки, работающий по следующему приниципу: за каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

1.2 Сортировка вставками

Сортировка вставками [2] (англ. *insertion sort*) - алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

На вход алгоритма подается массив A из n элементов. Сортируемые элементы также называются ключами. Каждый ключ последовательно сравнивается с уже отсортированной частью массива. Если для конкретного ключа выполняется следующее условие

$$K_{i-1} \leq K_{ind} \leq K_i \tag{1.1}$$

где K_{i-1}, K_i - ключи уже отсортированной последовательности, а K_{ind} -

ключ, для которого осуществляется поиск позиции. Когда условие выполняется элементы больше текущего сдвигаются вправо на одну позицию, а текущий помещается на нужную позицию. Сортировка заканчивается когда неотсортированная часть становится пустой.

1.3 Сортировка выбором

Сортировка выбором [3] (англ. *selection sort*) - алгоритм сортировки, основанный на поиске минимального элемента на каждом шаге. Когда такой будет найден, он помещается в конец отсортированной части массива и в слудющих итерациях учитываться уже не будет. Изнчально отсортированная часть пуста. Сортировка заканчивается, когда все элементы рассмотренны.

Данный алгоритм похож на сортировку пузырьком, за тем исключением, что в пузырьке все элементы сравниваются попарно и после соответствующих обменов элемент оказывается в нужной части массива; в этом же алгоритме сначала происходит поиск минимума, после чего перестановка элемента в нужное место. Это позволяет сократить количество обменов значениями.

1.4 Вывод

В данном разделе были рассмотренны принципы работы трех алгоритмов сортировки. Полученных знаний достаточно для разработки выбранных алгоритмов. На вход алгоритмам будут подаваться массив элементов и его размер. Реализуемое ПО будет работать в пользовательском режиме (вывод отсортированного массива тремя методами), а также в экспериментальном (проведение замеров времени выполнения алгоритмов).

2. Конструкторский раздел

В данном разделе будут спроектированы схемы алгоритмов, произведена оценка трудоемкости алгоритмов, описаны используемые типы данных, а также произведена оценка памяти и описана структура ПО.

2.1 Схемы алгоритмов

В следующих схемах массив будет представлен как A, N - размер массива, а A_i - i-ый элемент массива.

На рисунках 2.1 - 2.3 представлены схемы рассматриваемых алгоритмов.

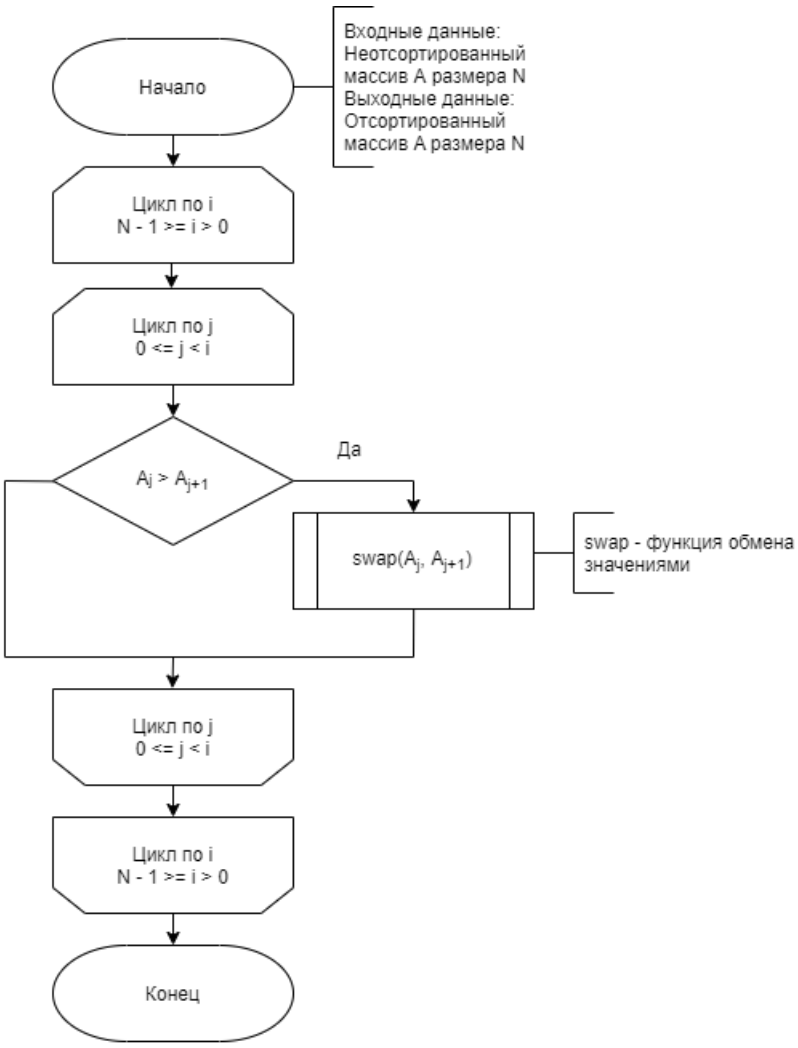

Рисунок 2.1 – Схема алгоритма сортировки пузырьком

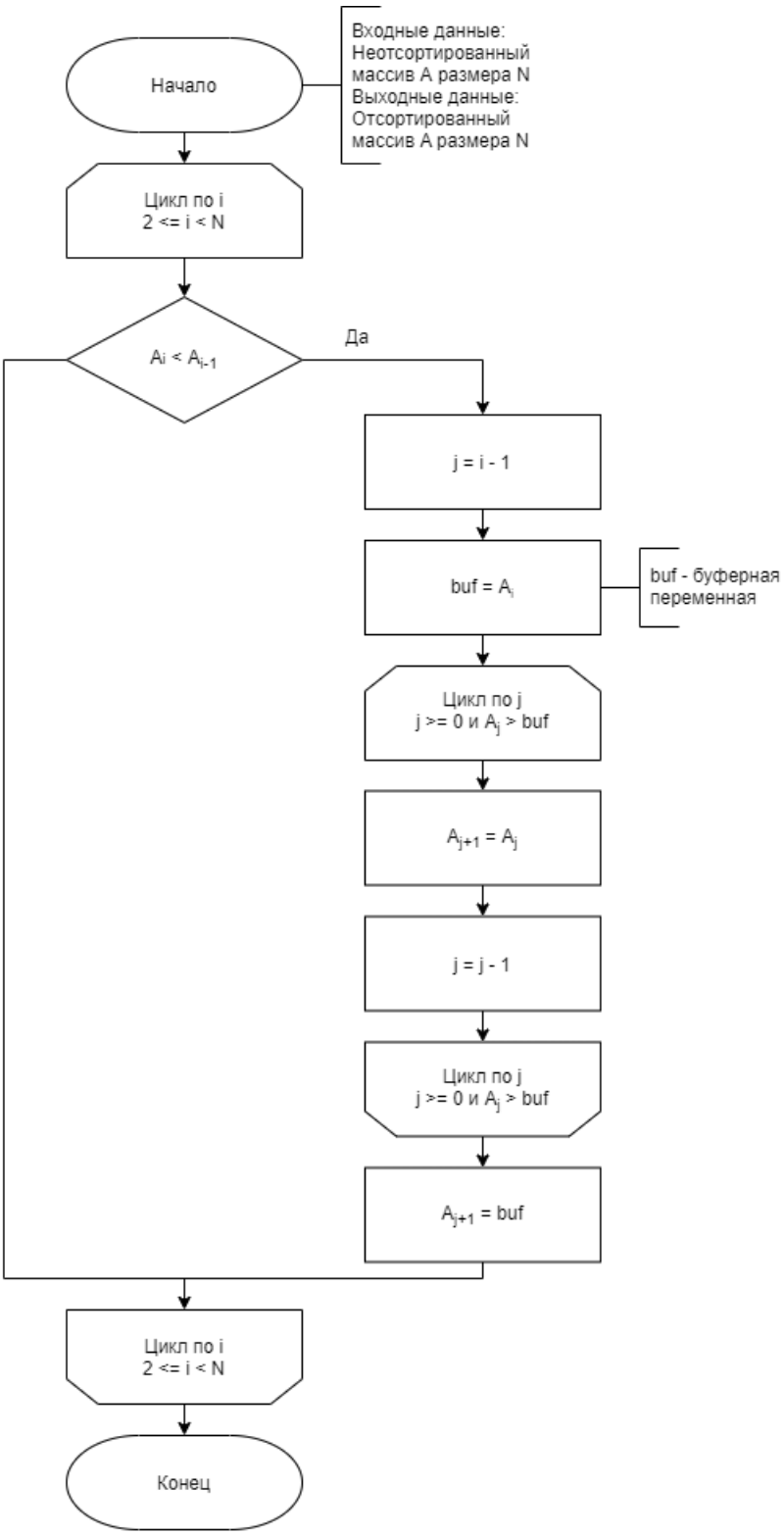

Рисунок 2.2 – Схема алгоритма сортировки вставками

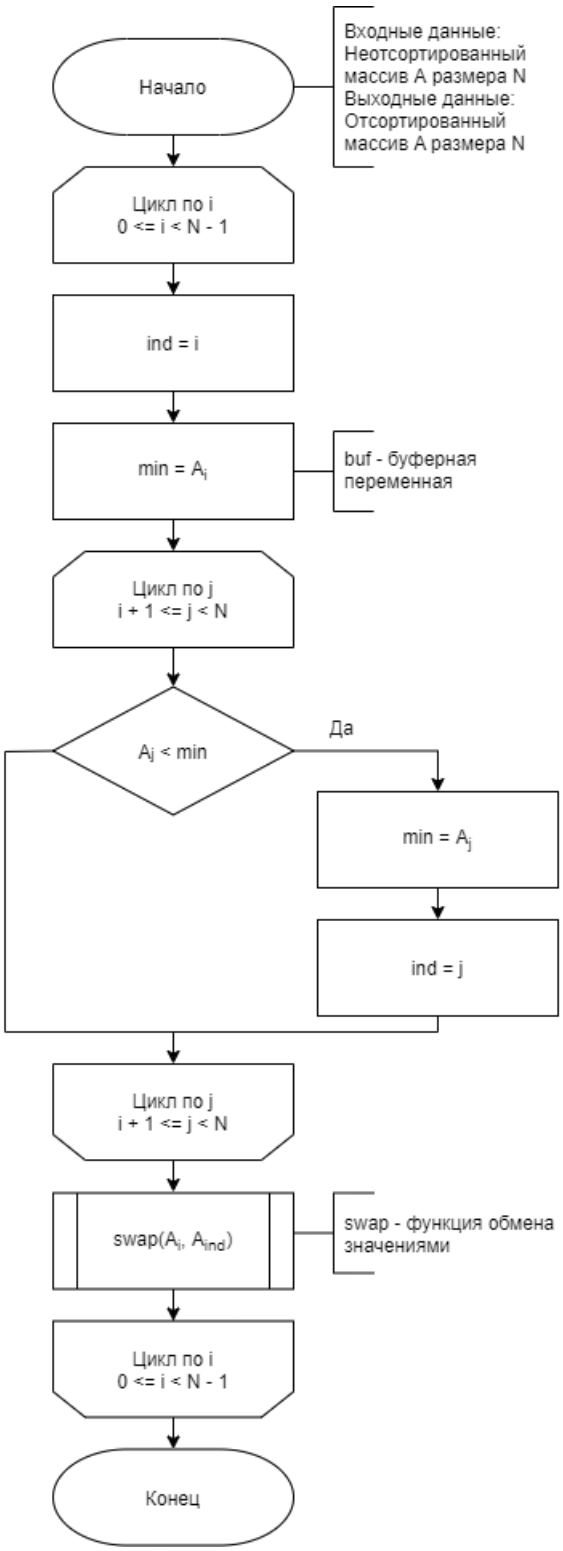

Рисунок 2.3 – Схема алгоритма сортировки выбором

2.2 Модель оценки трудоемкости алгоритмов

Трудоемкость алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций
   - Примем трудоемкость следующих операций равной 1:

$$=, +, -, + =, - =, ==, ! =, <, >, \leq, \geq, !, \&\&, ||, [] \tag{2.1}$$

   - Примем трудоемкость следующих операций равной 2:

$$*, /, \%, * =, / =, \tag{2.2}$$

2. Трудоемкость циклов

$$f = f_{init} + f_{compare} + N_{iter} * (f_{body} + f_{inc} + f_{compare}) \tag{2.3}$$

3. Трудоемкость условного оператора Пусть трудоемкость самого условного перехода равна 0. Тогда

$$f_{if} = f_{condition} + \begin{bmatrix} min(f_{true}, f_{false}), \text{в лучшем случае} \\ max(f_{true}, f_{false}), \text{в худшем случае} \end{bmatrix} \tag{2.4}$$

где

f_{init} - трудоемкость инициализации,

$f_{compare}$ - трудоемкость сравнения,

N_{iter} - количество итераций,

f_{body} - трудоемкость тела цикла,

f_{inc} - трудоемкость инкрементации,

$f_{condition}$ - трудоемкость условия,

f_{true}, f_{false} - трудоемкость веток условного оператора.

Трудоемкость сортировки выбором

Трудоемкость для сортировки выбором состоит из:

- Трудоемкость внешнего цикла:

$$f_{outer} = 3 + 13(N - 1) \tag{2.5}$$

- Суммарная трудоемкость внутренних циклов при повторах [1..N - 1]:

$$f_{inner} = 2(N - 1) + \frac{N(N - 1)}{2} \cdot f_{if} \tag{2.6}$$

- Трудоемкость условия внутреннего цикла:

$$f_{if} = 2 + \begin{bmatrix} 0, & \text{л.с.} \\ 3, & \text{х.с.} \end{bmatrix} \tag{2.7}$$

Тогда в лучшем случае трудоемкость будет равна

$$f = -12 + 14N + N^2 \approx N^2 = O(N^2) \tag{2.8}$$

В худшем случае

$$f = -12 + 12.5N + \frac{5}{2}N^2 \approx N^2 = O(N^2) \tag{2.9}$$

Трудоемкость сортировки вставками

Трудоемкость для сортировки вставками состоит из:

- Трудоемкость внешнего цикла:

$$f_{outer} = 2 + (N - 1)(2 + f_{if}) \tag{2.10}$$

- Трудоемкость условия:

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 4 + f_{inner}, & \text{х.с.} \end{cases} \tag{2.11}$$

- Трудоемкость внутреннего цикла:

$$f_{inner} = 4 + N(6 + 4) \tag{2.12}$$

В лучшем случае массив является отсортированным и N = 0, в худшем случае массив обратно отсортирован и количество проходов равно N - 1, тогда трудоемкость внутреннего цикла:

$$f_{inner} = 4 + \begin{cases} 0, & \text{л.с.} \\ 10(N-1), & \text{х.с.} \end{cases} \tag{2.13}$$

Тогда в лучшем случае трудоемкость будет равна

$$f = 6N \approx N = O(N) \tag{2.14}$$

В худшем случае

$$f = 6 - 14N + 10N^2 \approx N^2 = O(N^2) \tag{2.15}$$

Трудоемкость сортировки пузырьком

Трудоемкость данной сортировки может быть рассчитана таким же образом. Сложность алгоритма составляет $O(N^2)$ как в лучшем, так и в худшем случае [1].

2.3 Используемые типы данных

При реализации алгоритмов будут использованы следующие структуры данных:

- Массив типа int заданного размера;
- Размер массива - целое число int.

2.4 Оценка памяти

Так как алгоритмы выбранных сортировок отличаются по объему используемой памяти только количеством вспомогательных переменных, достаточно будет рассмотреть лишь один из них.

Пусть array - массив целых чисел, n - его размерность. Тогда для алгоритма сортировки выбором будет затрачена память на:

- Размер массива - sizeof(int);
- Массив array - sizeof(int) * n;
- Вспомогательные переменные - 2 * sizeof(int).

Алгоритм сортировки пузырьком обходится без использования вспомогательных переменных, что дает ему выигрыш, хоть и незначительный.

2.5 Структура ПО

ПО будет состоять из следующих модулей:

- Главный модуль - из него будет осуществляться запуск программы и выбор соответствующего режима работы;
- Модуль интерфейса - в нем будет описана реализация режимов работы программы;

- Модуль, содержащий функции работы с данными (такие как генерация данных, вывод, обмен значениями);
- Модуль, содержащий реализации алгоритмов.

2.6 Вывод

На основе полученных в аналитическом разделе знаний об алгоритмах были спроектированы схемы алгоритмов, произведена оценка их трудоемкости, выбраны используемые типы данных, проведена оценка затрачиваемого объема памяти, а также описана структура ПО.

3. Технологический раздел

В данном разделе будут приведены требования к ПО, средства его реализации и листинга кода алгоритмов, а также рассмотрены тестовые случаи.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- ПО принимает целочисленный массив размером не больше 1000;
- ПО возвращает отсортированный массив.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый язык C. В качестве среды разработки - QtCreator. Оба средства были выбраны из тех соображений, что навыки работы с ними были получены в более ранних курсах.

3.3 Листинги кода

Листинги 1 - 3 демонстрируют реализацию алгоритмов сортировки.

Листинг 3.1 – Алгоритм сортировки пузырьком

```
int *bubbleSort(int *array, int n)
{
    for (int i = n - 1; i > 0; i--)
        for (int j = 0; j < i; j++)
        {
            if (array[j] > array[j + 1])
               swap(&array[j], &array[j + 1]);
        }
    return array;
}
```

Листинг 3.2 – Алгоритм сортировки вставками

```
int *insertionSort(int *array, int n)
{
    int j, buf;
    for (int i = 2; i < n; i++)
    {
        if (array[i] < array[i - 1])
        {
            buf = array[i];
            j = i - 1;
            while (j >= 0 && array[j] > buf)
            {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = buf;
        }
    }
    return array;
}
```

Листинг 3.3 – Алгоритм сортировки выбором

```
int *selectionSort(int *array, int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int min = array[i], ind = i;
        for (int j = i + 1; j < n; j++)
            if (array[j] < min)
            {
                min = array[j];
                ind = j;
            }
        swap(array + i, array + ind);
    }
    return array;
}
```

3.4 Тестирование ПО

В таблице 3.1 приведены тестовые случаи для алгоритмов сортировки. Случай 1 описывает пустой массив, случай 2 - неотсортированный массив,

случай 3 - отсортированный массив, случай 4 - обратно отсортированный массив, случай 5 - массив из равных между собой элементов.

Таблица 3.1 – Тестовые случаи

№	Входной массив	Ожидаемый результат	Результат
1			
2	-8 5 1 25 7 -44	-44 -8 1 5 7 25	-44 -8 1 5 7 25
3	-116 3 11 28 47 89	-116 3 11 28 47 89	-116 3 11 28 47 89
4	283 115 97 42 13 0	0 13 42 97 115 283	0 13 42 97 115 283
5	1 1 1 1 1	1 1 1 1 1	1 1 1 1 1

3.5 Вывод

На основе схем из конструкторского раздела были написаны реализации требуемых алгоритмов, а также проведено их тестирвание.

4. Исследовательский раздел

В данном разделе будет проведен сравнительный анализ алгоритмов по времени и затрачиваемой памяти.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими характеристиками:

- Операционная система Windows 10 [4]
- Память 8 Гб
- Процессор Intel Core i3 7020U, 2.3 ГГц [5]

Во время проведения эксперимента устройство не было нагружено сторонними задачами, а также было подключено к блоку питания.

Замеры процессорного времени проводились с помощью ассемблерной вставки, вычисляющей затраченное процессорное время в тиках.

Листинг 4.1 – Ассемблерная вставка замера процессорного времени в тиках

```
unsigned long long tick(void)
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc": "=A" (d));
    return d;
}
```

При проведении эксперимента была отключена оптимизация командой компилятору -o0 [6].

4.2 Оценка времени работы алгоритмов

В таблицах 4.1 - 4.3 представлены замеры процессорного времени работы алгоритмов на массивах размером от 50 до 650 элементов. Такая раз-

мерность выбрана, потому что на больших размерах массива происходит переполнение типа счетчика тиков. Каждое значение было получено усреднением по 1000 замерам.

Таблица 4.1 – Временные замеры работы алгоритмов на случайных значениях

N	Bubble	Insert	Selection
50	30705	8844	17135
100	109026	27299	63087
150	233112	63269	124151
200	378871	102549	209498
250	568232	156701	315817
300	828940	231688	451743
350	1065136	296117	589815
400	1367519	392281	774546
450	1755752	497132	952358
500	2131493	626607	1182272
550	2577364	735687	1407664
600	2966913	862352	1703104
650	3972662	1131503	1988339

Таблица 4.2 – Временные замеры работы алгоритмов на отсортированных данных

N	Bubble	Insert	Selection
50	11582	625	12805
100	45791	1312	51898
150	102348	1737	104398
200	177895	3238	180238
250	288828	2874	279981
300	404472	3624	401549
350	548377	4775	536613
400	718761	4758	709811
450	899666	5272	890204
500	1107400	6437	1092161
550	1324354	6395	1323357
600	1577523	9432	1563866
650	1964788	8014	1871389

Таблица 4.3 – Временные замеры работы алгоритмов на обратно отсортированных данных

N	Bubble	Insert	Selection
50	25466	12921	14846
100	101150	49445	55077
150	224551	111362	125920
200	393320	192321	213532
250	609787	304684	320888
300	874043	426119	455669
350	1189252	588355	620395
400	1552689	762852	813592
450	1948077	970662	1015567
500	2410845	1173846	1265244
550	2935642	1466136	1523410
600	3589197	1761747	1826844
650	4225705	2065891	2126514

На рисунке 4.1 представлен график сравнения времени работы алгоритмов на случайных данных. В результате эксперимента было получено, что на размерах массивов до 650 элементов сортировка пузырьком работает в 2 раза медленнее сортировки выбором и в 4 раза медленнее сортировки вставками.

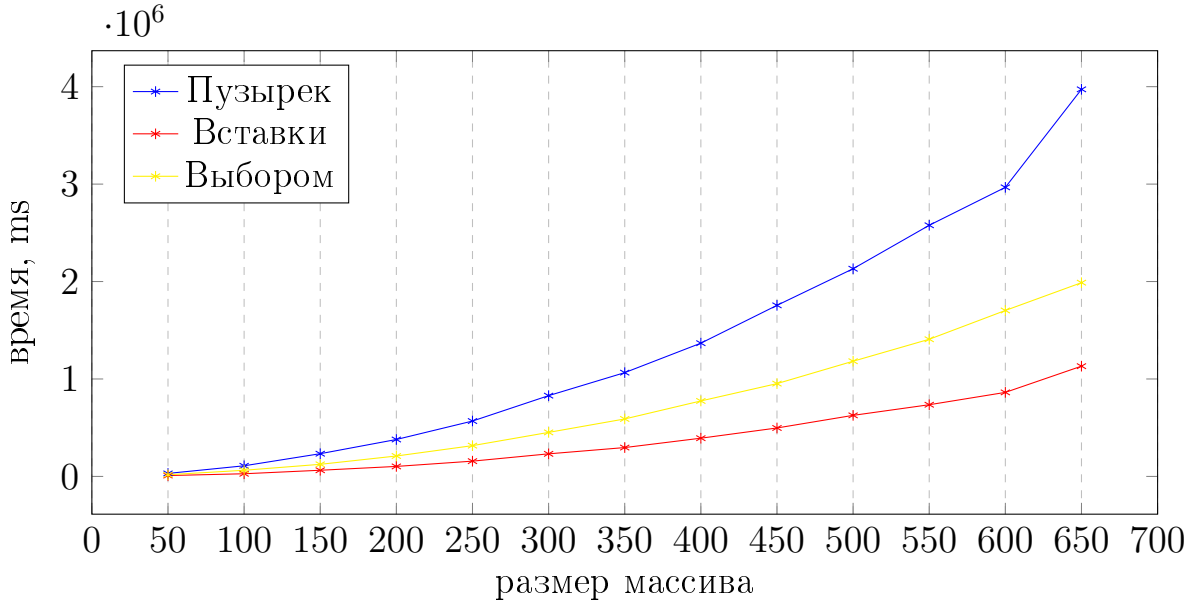

Рисунок 4.1 – Сравнение времени работы алгоритмов на случайных данных

На рисунке 4.2 представлен график сравнения времени работы алгорит-

мов на отсортированных данных. В результате эксперимента было получено, что на размерах массивов до 650 элементов сортировка пузырьком работает так же как сортировка выбором, имея квадратичную зависимость, в то время как сортировка вставками линейно зависит от размера массива. Ее поведение представленно на рисунке 4.3.

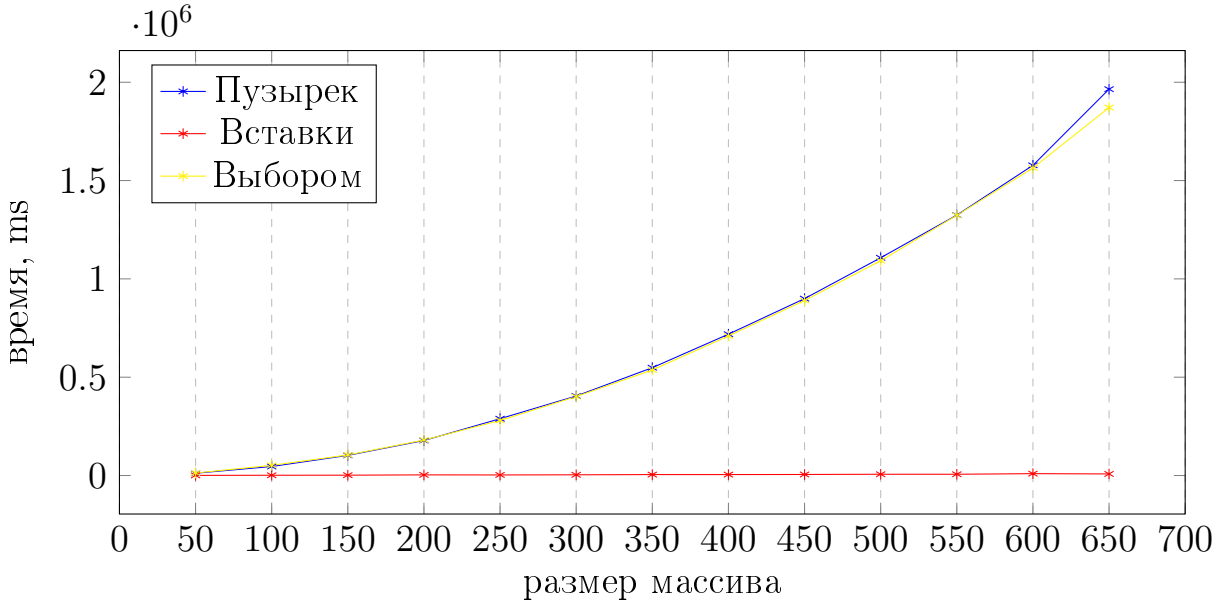

Рисунок 4.2 – Сравнение времени работы алгоритмов на отсортированных данных

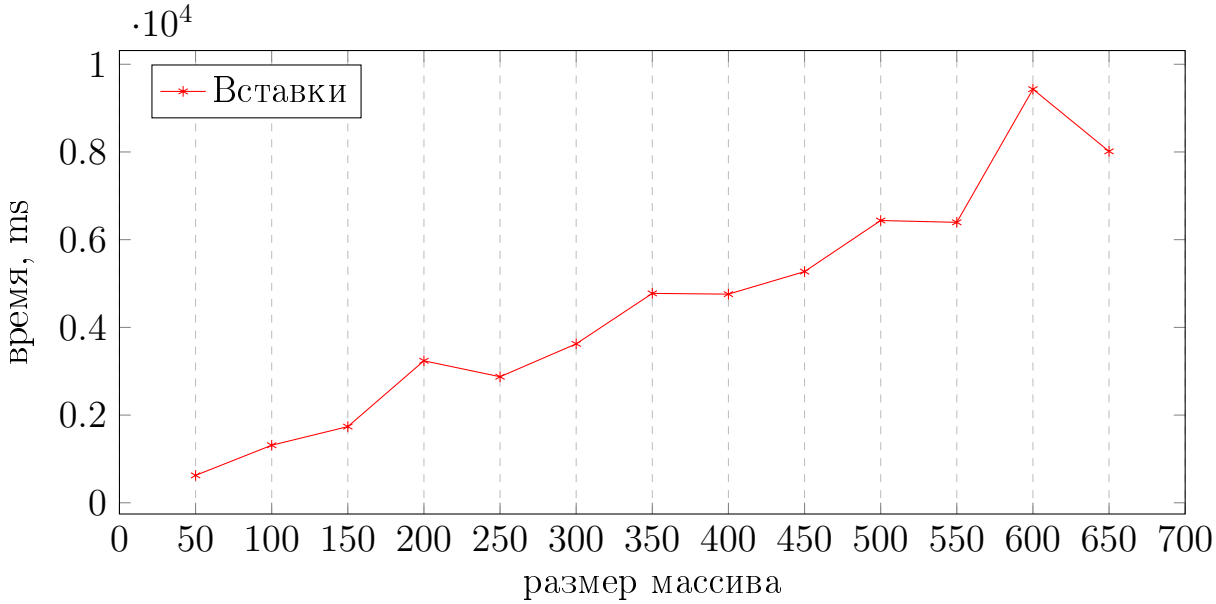

Рисунок 4.3 – Время работы алгоритма сортировки вставками на отсортированных данных

На рисунке 4.4 представлен график сравнения времени работы алгоритмов на обратно отсортированных данных. В результате эксперимента было

получено, что на размерах массивов до 650 элементов сортировка пузырьком работает в 2 раза медленнее сортировки выбором и вставками, в то время как между собой их графики почти идентичны.

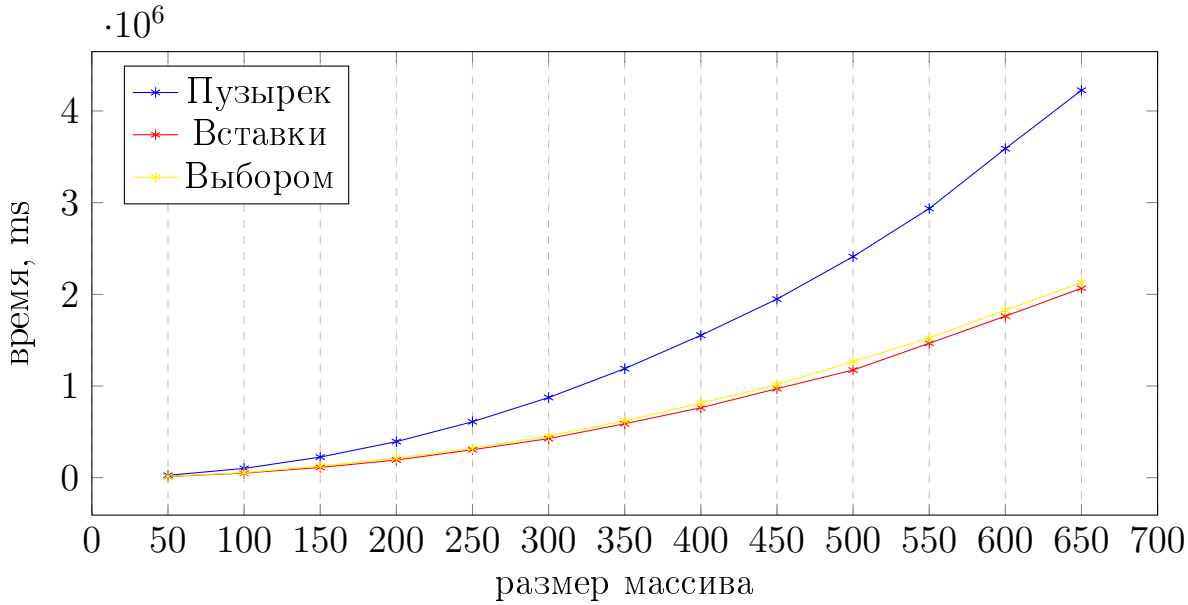

Рисунок 4.4 – Сравнение времени работы алгоритмов на обратно отсортированных данных

4.3 Вывод

Реализиция алгоритма сортировки пузырьком по сравнению с другими алгоритмами:

- На случайных данных - в 2 раза медленнее сортировки выбором, в 4 раза медленнее сортировки вставками;
- На отсортированных данных - на 5% медленнее сортировки выбором, в 2.5 $\cdot 10^2$ медленнее сортировки вставками;
- На обратно отсортированных данных - в 2 раза медленнее обеих сортировок.

Это связано с большим количеством проверок соседних элементов и простых обменов.

Реализиция алгоритма сортировки вставками по сравнению с другими алгоритмами:

- На случайных данных - в 2 раза быстрее сортировки выбором, в 4 раза быстрее сортировки пузырьком;
- На отсортированных данных - в $4 \cdot 10^3$ быстрее сортировки пузырьком и выбором;
- На обратно отсортированных данных - в 2 раза быстрее сортировки пузырьком, на 5% быстрее сортировки выбором.

Реализиция алгоритма сортировки выбором по сравнению с другими алгоритмами:

- На случайных данных - в 2 раза медленнее сортировки вставками, в 2 раза быстрее сортировки пузырьком;
- На отсортированных данных - на 5% быстрее сортировки пузырьком, в $2.5 \cdot 10^2$ медленнее сортировки вставками;
- На обратно отсортированных данных - в 2 раза быстрее сортировки пузырьком, на 5% медленнее сортировки вставками.

Заключение

В ходе выполнения данной лабораторной работы были выполнены следующие задачи:

- Изучены выбранные алгоритмы сортировок;
- Составлены схемы данных алгоритмов;
- Реализованны разработанные алгоритмы сортировок;
- Проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- Описаны и обоснованы полученные результаты.

В результате исследований можно прийти к выводу, что алгоритм сортировки пузырьком является самым медленным из рассмотренных. Его сложность, как и сложность сортировки выбором, в лучшем и худшем случае равняется $O(N^2)$. Сортировка вставками быстрее других сортировок в 4000 раз на отсортированных данных, а также быстрее от 2 до 4 раз на других данных. По используемой памяти алгоритмы практически не отличаются, за исключением использования разного количества вспомогательных переменных.

Список литературы

[1] В. Левитин А. Алгоритмы. Введение в разработку и анализ. – М.: Вильямс, 2006. Т. 3 из *Глава 3. Метод грубой силы: Сортировка выбором.* С. 143–143.

[2] В. Ахо А. Data structures and algorithms. – М.: Вильямс, 2000. с. 231.

[3] В. Левитин А. Алгоритмы. Введение в разработку и анализ. – М.: Вильямс, 2006. Т. 3 из *Глава 3. Метод грубой силы: Пузырьковая сортировка.* С. 144–146.

[4] Windows 10 [Электронный ресурс]. Режим доступа: `https://www.microsoft.com/ru-ru/windows/windows-10-specifications` (дата обращения: 18.10.2021).

[5] Процессор Intel Core i3-7020u [Электронный ресурс]. Режим доступа: `https://www.intel.ru/content/www/ru/ru/products/sku/122590/intel-core-i37020u-processor-3m-cache-2-30-ghz/specifications.html` (дата обращения: 18.10.2021).

[6] ISO/IEC 9899:1999 [Электронный ресурс]. Режим доступа: `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf` (дата обращения: 22.10.2021).