Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 3

ТЕМА

Студент группы ИУ7-54,
Лозовский Алексей

2019 г.

Содержание

Введение

Порядок - вещь, о которой мечтают многие люди, программисты не исключение, если, конечно, считать их людьми, именно поэтому при виде неотсортированных данных так и чешутся руки что-нибудь отсортировать, как по убыванию, так и по возрастанию. Причем, поскольку данные могут быть самые разные, сортировок создано буквально на любой фетиш, каждому придется какая-нибудь по душе - для больших данных, для маленьких, кто-то увлекается упорядочиванием символов, у всех свои причуды. Вот и мы в данной лабораторной работе решили заняться сортировкой данных.

Ну так для чего же используются сортировки? Для программирования, конечно, в основном. Хотя, множество психологов уверены, что порядок - неотъемлемая составляющая жизни человека, хорошим примером является книга [1]. По словам автора, человек не станет состоятельным, если у него в шкафу не отсортированы вещи по дням недели, что уж говорить о программистах... какой ты программист, если у тебя даже массив не отсортирован. Кроме того, жизнь в университете без сортировок это вообще не жизнь, семестр без этих алгоритмов - время потраченное впустую, таким образом, очевидно, что сортировки применяются повсеместно и нужно быть глупцом, чтобы их не любить.

1 Аналитическая часть

1.1 Описание алгоритмов

Как уже было сказано выше, сотрировки - важная составляющая жизни программиста. Разберем, что они из себя представляют.

Сортировка пузырьком.

Согласно описанию в [2] сортировка пузырьком - базовый алгоритм сортировки. Рассмотрим его работу. Шаг состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию...

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

Сортировка вставками.

Исходя из написанного в [3] в сортировке пузырьком можно было четко заявить, что на i-м шаге элементы a[0]...a[i] стоят на правильных местах и никуда более не переместятся. Здесь же подобное утверждение будет более слабым: последовательность a[0]...a[i] упорядочена. При этом по ходу алгоритма в нее будут вставляться все новые элементы.

Будем разбирать алгоритм, рассматривая его действия на i-м шаге. Как говорилось выше, последовательность к этому моменту разделена на две части: готовую a[0]...a[i] и неупорядоченную a[i+1]...a[n].

На следующем, (i+1)-м каждом шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.

В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена), либо они меняются местами и процесс повторяется. Таким образом, в процессе вставки мы "просеиваем"элемент x к началу массива, останавливаясь в случае, когда

- Найден элемент, меньший x;
- Достигнуто начало последовательности;

Сортировка выбором.

Основываясь на информации в [4] идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке.

Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от нулевого и заканчивая (n-1)-м.

На i-м шаге выбираем наименьший из элементов a[i] ... a[n] и меняем его местами с a[i]. Вне зависимости от номера текущего шага i, последовательность a[0]...a[i] является упоря-

доченной. Таким образом, на (n-1)-м шаге вся последовательность, кроме a[n] оказывается отсортированной, а a[n] стоит на последнем месте по праву: все меньшие элементы уже ушли влево.

1.2 Задание на выполнение лабораторной работы

- Реализовать три алгоритма сортировки;
- Оценить их трудоемкость;
- Исследовать время работы и определить лучший, средний и худший случаи

2 Конструкторская часть

2.1 Структура кода и представление данных

Программа будет написана в структурном стиле. Будут функция для тестирования, функции для сортировок. Работа только с целыми числами.

2.2 Разработка алгоритмов

2.2.1 Сортировка пузырьком

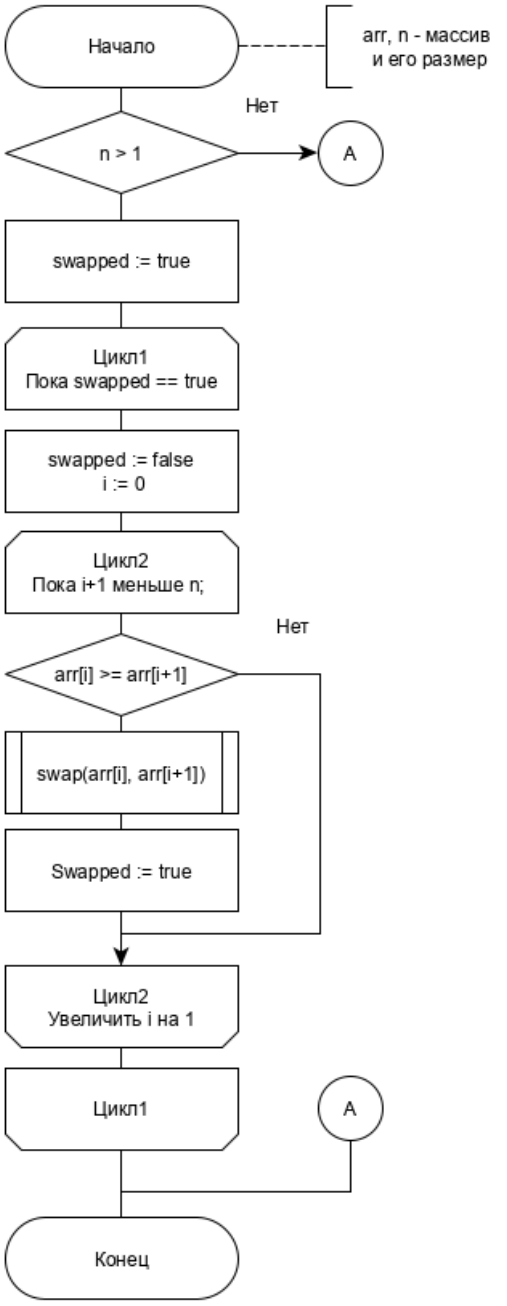

Рис. 1: Сортировка пузырьком

$$F_{cond} + \begin{cases} F_{cbody}, & n > 1 \\ 0, & \text{иначе} \end{cases}$$

$$F_{cbody} = 1 + F_{outer}$$

$$F_{outer} = 1 + N(1+$$

$$1 + 1 + F_{inner}$$

$$F_{inner} = (N-1)(2+$$

$$+2 + 1 + 1+$$

$$+2+$$

$$+1+$$

$$+1$$

Тип оператора	Стоимость
Арифметические операторы	1
Операторы сравнения	1
Логические операторы	1
Побитовые операторы	1
Составное присваивание	1
Операторы работы с указателями	1
Другие операторы	1
функция swap(x,y)	2

Анализ трудоемкости в общем виде:

$$F_{all} = F_{cond} + F_{body} \tag{1}$$

$$F_{body} = 1 + F_{outer} \tag{2}$$

$$F_{outer} = N(1 + 2 + 1) + F_{inner} \tag{3}$$

$$F = \frac{N^2}{2} \tag{4}$$

Анализ лучшего случая

При обработке отсортированного массива алгоритм сортировки пузырьком показывает лучший результат.

Опишем количество итераций для внешнего(F_{outer}), внутреннего (F_{inner}) циклов.

При обработке отсортированного массива внешний цикл будет работать 1 раз, так как зависит от флага *swapped*. Тогда внутренний цикл пройдет по N-1 элементу массива. Посчитаем количество операций для всего алгоритма.

$$F_{inner} = N - 1(2 + 1) = 3N - 3;$$

$$F_{outer} = 1 * (2 + 2) + F_{inner} = 3N - 3 + 4 = 3N + 1;$$

$$F_{body} = 1 + F_{outer} = 3N + 1 + 1 = 3N + 2;$$

$$F_{all} = 1 + F_{body} = 3N + 3;$$

Общая сложность алгоритма: $O(N)$

Анализ худшего случая

При обработке обратносортированного массива алгоритм сортировки пузырьком показываем худший результат.

Опишем количество итераций для внешнего(F_{outer}), внутреннего (F_{inner}) циклов.

$$F_{inner} = \frac{N^2}{2}(4 + 2 + 1 + 1 + 2) = 5N^2;$$

$$F_{outer} = N(1 + 2 + 1 + 2) + F_{inner} = 5N^2 + 6N;$$

$$F_{body} = 1 + F_{outer} = 5N^2 + 6N + 1;$$

$$F_{all} = 1 + F_{body} = 5N^2 + 6N + 2;$$

Общая сложность алгоритма: $O(N^2)$

2.2.2 Сортировка вставками

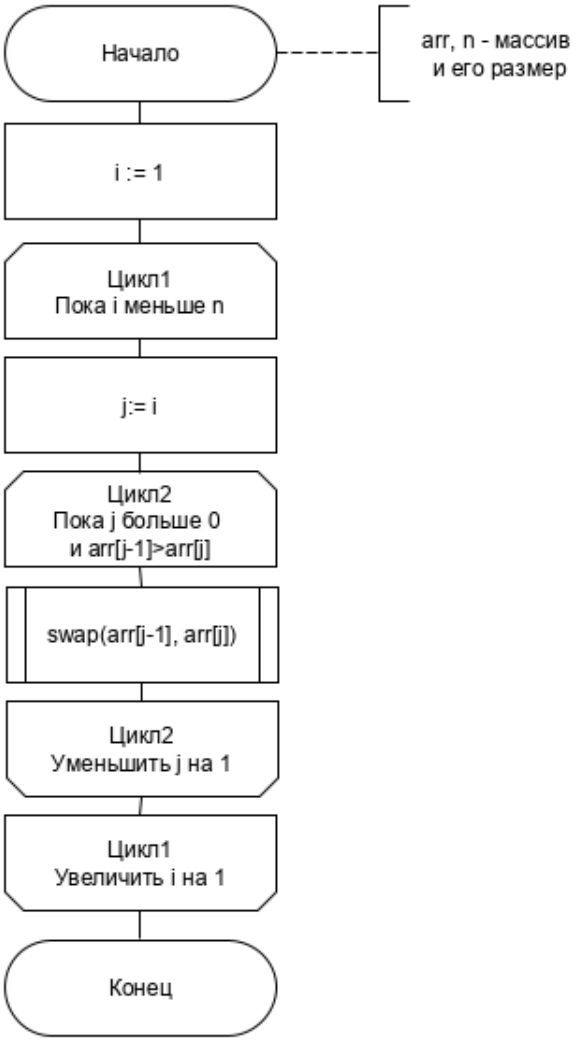

Рис. 2: Сортировка вставками

Анализ трудоемкости.
Согласно [2] сложность сортировки вставками для **лучшего случая** оценивается как $O(N)$.
Худший случай: $O(N^2)$.

2.2.3 Сортировка выбором

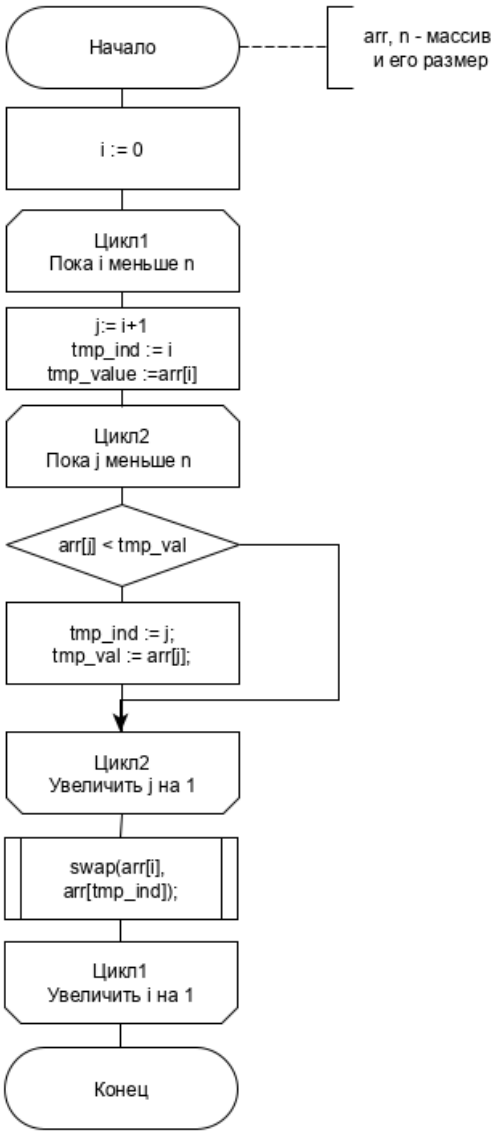

Рис. 3: Сортировка выбором

Анализ трудоемкости.
Согласно информации, приведенной в [3]:
Общая сложность алгоритма $O(N^2)$
Согласно [2] сложность сортировки вставками для **лучшего случая** оценивается как $O(N^2)$.
Худший случай: $O(N^2)$.

2.3 Выводы по конструкторскому разделу

Таким образом, был определен стиль кода для реализации алгоритмов, а также разработаны схемы каждого из исследуемых алгоритмов.Был проанализирован алгоритм сортировки пузырьком, исследованы оценки сложности алгоритмов сортировки вставками и выбором. Для каждого алгоритма были рассмотрены лучший и худший случаи.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна поддерживать режим тестирования. При запуске программы автоматически запускаются тесты, которые считают время. Программа должна записывать результаты тестирования в файл в формате (array_size, time).

3.2 Средства реализации

Для реализации программы был выбран язык C++, стандарт C++11 [7], так как на языке имеется наибольший опыт работы. Для замера времени используется библиотека ctime.

3.3 Листинг кода

Листинг 1: Сортировка пузырьком

```
void buble_sort(int arr[], int n)
{
    if (n > 1)
    {
        bool swapped = true;
        while (swapped)
        {
             swapped = false;

             for (int i=0; i+1<n; i++)
             {
                  if (arr[i] >= arr[i+1])
                  {
                      swap(arr[i], arr[i+1]);
                      swapped = true;
                  }
              }
              n--;
        }
    }
}
```

Листинг 2: Сортировка вставками

```
void insertion_sort(int arr[], int n)
{
    for(int i=1; i<n; i++)
        for(int j=i; j>0 && arr[j-1]>arr[j]; j--)
            swap(arr[j-1], arr[j]);
}
```

Листинг 3: Сортировка выбором

```
void selection_sort(int arr[], int n)
{
    for (int i=0; i<n; i++)
    {
        int tmp_ind = i;
        int tmp_val = arr[i];

        for (int j = i+1; j<n; j++)
            if (arr[j] < tmp_val)
            {
                tmp_ind = j;
                tmp_val = arr[j];
            }

        swap(arr[i], arr[tmp_ind]);
    }
}
```

3.4 Выводы по технологическому разделу

Был выбран язык программамирования C++, стандарт языка(C++11) для разработки программы, была определена библиотека для замера времени(ctime), приведены листинги кода. Описаны требования к ПО.

4 Экспериментальная часть

4.1 Примеры работы

Сортировка пузырьком

Входные данные:

Array: 0 1 2 3 4 5 6 7 8 9
Array: 10 9 8 7 6 5 4 3 2 1
Array: 0 6 2 8 6 5 3 9 9 8

Выходные данные:

Result: 0 1 2 3 4 5 6 7 8 9
Result: 1 2 3 4 5 6 7 8 9 10
Result: 0 2 3 5 6 6 8 8 9 9

Сортировка вставками

Входные данные:

Array: 0 1 2 3 4 5 6 7 8 9
Array: 10 9 8 7 6 5 4 3 2 1
Array: 1 9 7 5 3 0 1 4 1 1

Выходные данные:

Result: 0 1 2 3 4 5 6 7 8 9
Result: 1 2 3 4 5 6 7 8 9 10
Result: 0 1 1 1 1 3 4 5 7 9

Сортировка выбором

Входные данные:

Array: 0 1 2 3 4 5 6 7 8 9
Array: 10 9 8 7 6 5 4 3 2 1
Array: 10 4 1 0 0 4 5 6 6 6

Выходные данные:

Result: 0 1 2 3 4 5 6 7 8 9
Result: 1 2 3 4 5 6 7 8 9 10
Result: 0 0 1 4 4 5 6 6 6 10

4.2 Постановка эксперимента

Будем замерять время работа алгортимов сортировки на отсортированном, обратносортированном и случайно сгенерированном массивах размерностями $100, 200, \ldots, 1000$ элементах. Будем брать усредненное из 5 тестов время для получения большей точности. Для замера времени будем использовать библиотеку C++ ctime.

4.2.1 Тестирование времени работы функций

Сортировка пузырьком

Массив	sorted, c	reversed, c	randomed, c
100	0	0	0
200	0	0.0002	0.0004
300	0	0.0004	0.0004
400	0	0.0008	0.0008
500	0	0.0012	0.001
600	0	0.0012	0.0014
700	0	0.0014	0.002
800	0	0.0024	0.0024
900	0	0.0030	0.0027
1000	0	0.0038	0.0003

Сортировка выбором

Массив	sorted	reversed	randomed
100	0	0	0
200	0	0.0002	0
300	0	0.0004	0.0002
400	0	0.0008	0.0002
500	0	0.0012	0.0002
600	0.0006	0.0012	0.0006
700	0.0006	0.0014	0.0006
800	0.0006	0.0024	0.0008
900	0.0010	0.0030	0.0010
1000	0.0012	0.0038	0.0017

Сортировка вставками

Массив	sorted	reversed	randomed
100	0	0.0002	0.0002
200	0	0.0002	0
300	0	0.0002	0.0002
400	0	0.0008	0.0002
500	0	0.0012	0.0002
600	0	0.0012	0.0006
700	0	0.0014	0.0006
800	0	0.0024	0.0008
900	0	0.0030	0.0010
1000	0	0.0038	0.0017

4.3 Сравнительный анализ на материале экспериментальных данных

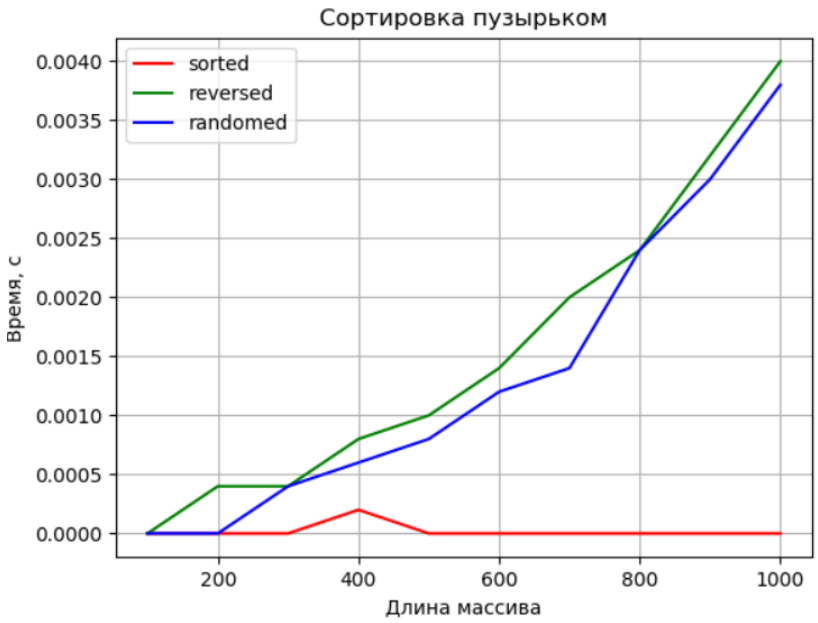

Рис. 4: Сортировка пузырьком

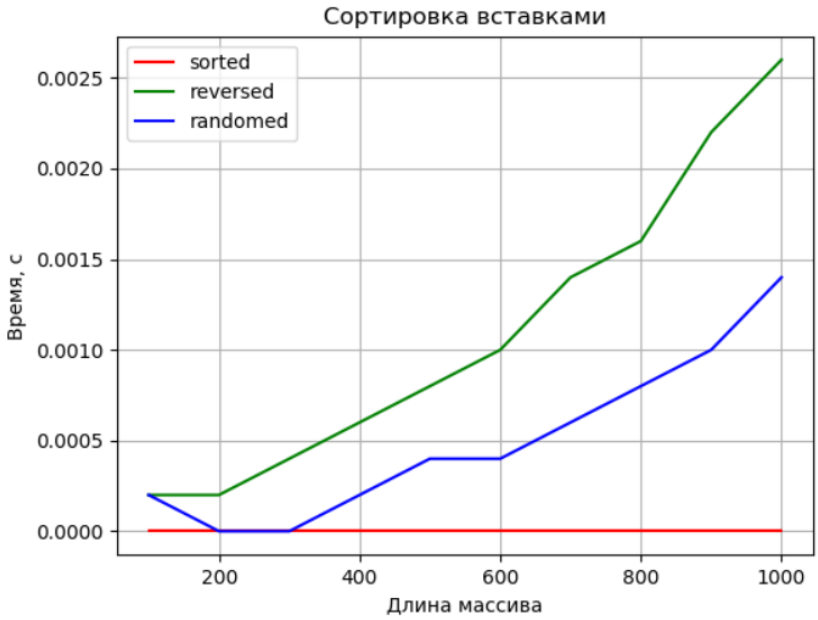

Рис. 5: Сортировка вставками

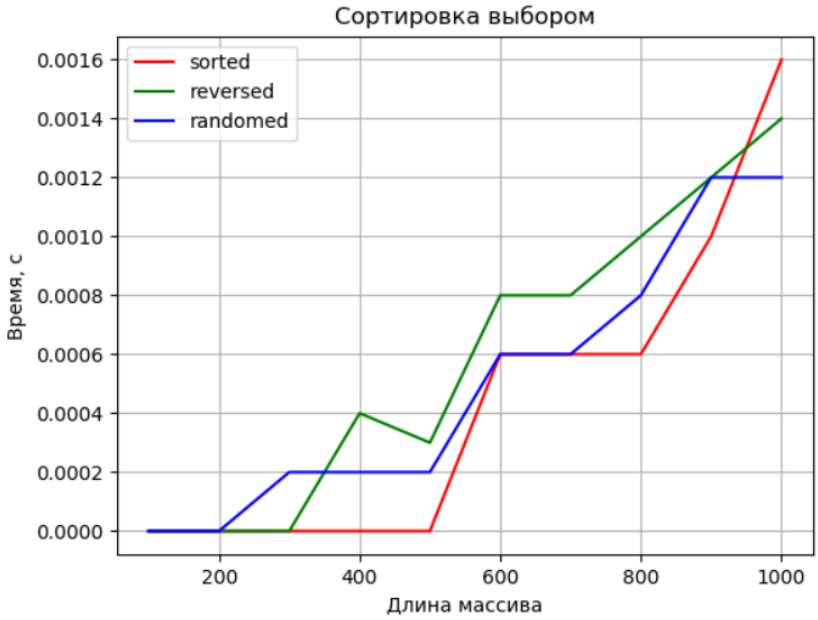

Рис. 6: Сортировка выбором

4.4 Выводы по экспериментальной части

Были протестированы алгоритмы сортировки на массивах размерностей $100, 200, \ldots, 1000$.

В результате тестирования было получено, что лучшее время сортировки показывают на отсортированном массиве, при обработке 1000 элементов сортировки вставками и пузырьком отработали за 0с, сортировка выбором выдала результат 0.0012с. Худшие результаты были получены при обработке обратно отсортированных массивов: все алгоритмы отработали за 0.0038с, при обработке случайного массива алгоритмом пузырька значения колебались, для 1000 элементов алгоритм показал время 0.0003с, в то время как для массива размерностью 200 - 0.0004с. Сортировка выбором и вставками обработали массив 1000 элементов за 0.0017с

При сравнении времени работы алгоритмов можно сделать вывод, что на массивах размерами до 200 элементов выдают результат, приближенный к 0 с. При обработке массивов большего размера лучший результат показала сортировка выбором, 0.0014с для обработки обратно отсортированного массива из 1000 элементов, в то время как сортировка пузырьком отработала за 0.004с, а вставками за 0.0026с

Заключение

Таким образом, в ходе работы были реализованы алгоритмы сортировки пузырьком, вставками и выбором. Был проведен анализ алгоритмов, а также протестировано время работы каждого алгоритма для массив разных размеров. В результате тестирования были сделаны выводы о трудоемкости алгоритмов.

Также были сделаны выводы о времени работы каждого из алгоритмов - было получено, что лучшее время сортировки показывают на отсортированном массиве, при обработке 1000 элементов сортировки вставками и пузырьком отработали за 0с, сортировка выбором выдала результат 0.0012с. Худшие результаты были получены при обработке обратно отсортированных массивов: все алгоритмы отработали за 0.0038с, при обработке случайного массива алгоритмом пузырька значения колебались, для 1000 элементов алгоритм показал время 0.0003с, в то время как для массива размерностью 200 - 0.0004с. Сортировка выбором и вставками обработали массив 1000 элементов за 0.0017с

При сравнении времени работы алгоритмов был сделан вывод, что на массивах размерностью до 200 элементов выдается результат, приближенный к 0 с. При обработке массивов большего размера лучший результат показала сортировка выбором, 0.0014с для обработки обратно отсортированного массива из 1000 элементов, в то время, как сортировка пузырьком отработала за 0.004с, а вставками за 0.0026с

Список литературы

[1] Марла Силли «Школа Флайледи. Как навести порядок в доме и в жизни»

[2] Кнут Д. Э. 5.2.1 Сортировка путём вставок "Искусство программирования. Том 3. Сортировка и поиск"

[3] Роберт Седжвик. Часть III. Глава 6. "Элементарные методы сортировки: 6.2 Сортировка выбором "

[4] Стивен С. Скиена "Алгоритмы. Руководство по разработке"

[5] Генри С. Уоррен мл. "Алгоритмические трюки для программистов"

[6] Джордж Хайнеман, Гари Поллис, Стэнли Селков "Алгоритмы. Справочник с примерами на C, C++, Java и Python"

[7] International Standard ISO - https://isocpp.org/std/the-standard