Лабораторная работа №3
По курсу: "Анализ алгоритмов"
По теме: "Алгоритмы сортировки массива"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Цель лабораторной работы: изучение алгоритмов сортировки массивов, сравнительный анализ времени работы данных алгоритмов, анализ трудоемкости алгоритмов.

Задачи работы:

1. реализация следующих алгоритмов сортировки массивов – сортировка вставками, сортировка пузырьком и сортировка выбором;
2. оценка трудоемкости алгоритмов;
3. анализ времени работы программы;
4. сравнительный анализ работы алгоритмов для массивов размера от 100 до 1000 элементов.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмова, а также составлена модель для вычисления трудоемкости..

1.1 Описание алгоритмов

Сортировка вставками (Insertion Sort) — это простой алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Сложность алгоритма: $O(n^2)$. [1]

На вход алгоритма подаётся последовательность n чисел: $a_1, a_2, ..., a_n$. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности $a'_1, a'_2, ..., a'_n$, чтобы выполнялось следующее соотношение $a'_1 \leq a'_2 ... \leq a'_n$[2].

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма [3].

Сортировка пузырьком (Bubble Sort) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: $O(n^2)$. [1]

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен 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).

Трудоемкость тела внутреннего цикла не зависит от количества элементов в массиве, поэтому оценивается как O(1). В результате выполнения внутреннего цикла, наибольший элемент смещается в конец массива неупорядоченной части, поэтому через N таких вызовов массив в любом случае окажется отсортирован. Если же массив отсортирован, то внутренний цикл будет выполнен лишь один раз.

Лучший случай (отсортированный массив):
$O(\sum_{j=1}^{n-1} 1) = O(n)$

Рандомный и худший (массив, отсортированный в обратном порядке) случаи:
$O(\sum_{j=n-1}^{0} \sum_{j=1}^{i} 1) = O(n^2)$

Сортировка выбором (Selection Sort) —На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае $O(n^2)$, предполагая что сравнения делаются за постоянное время. [1]

Шаги алгоритма:

1. находим номер минимального значения в текущем списке;
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы;

1.2 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

1. Базовые операции имеют трудоемкость 1 (<, >, =, <=, =>, ==, +, -, *, /);
2. Оператор if имеет трудоемкость, равную трудоемкости тела оператора;
3. Оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где F_{body} – трудоемкость операций в теле цикла, а F_{check} – трудоемкость проверки условия.

1.3 Вывод

Были рассмотрены поверхностно алгоритмы сортировки «вставками», «пузырьком», «выбором».

2 Конструкторская часть

В данной части представлены схемы алгоритмов.

2.1 Схемы алгоритмов

Алгоритм сортировка вставками:

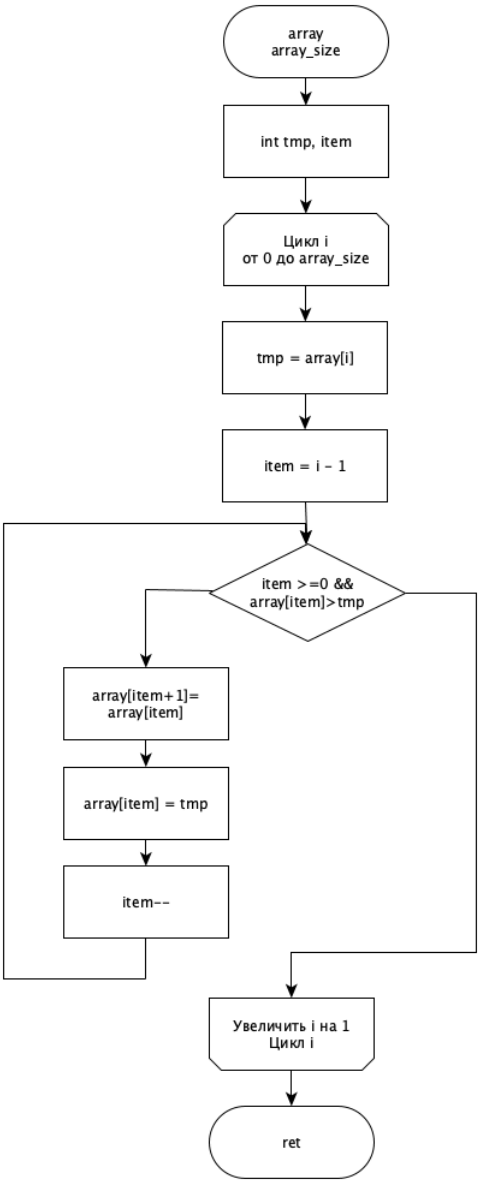

Рис. 1: Сортировка вставками

Алгоритм сортировка пузырьком:

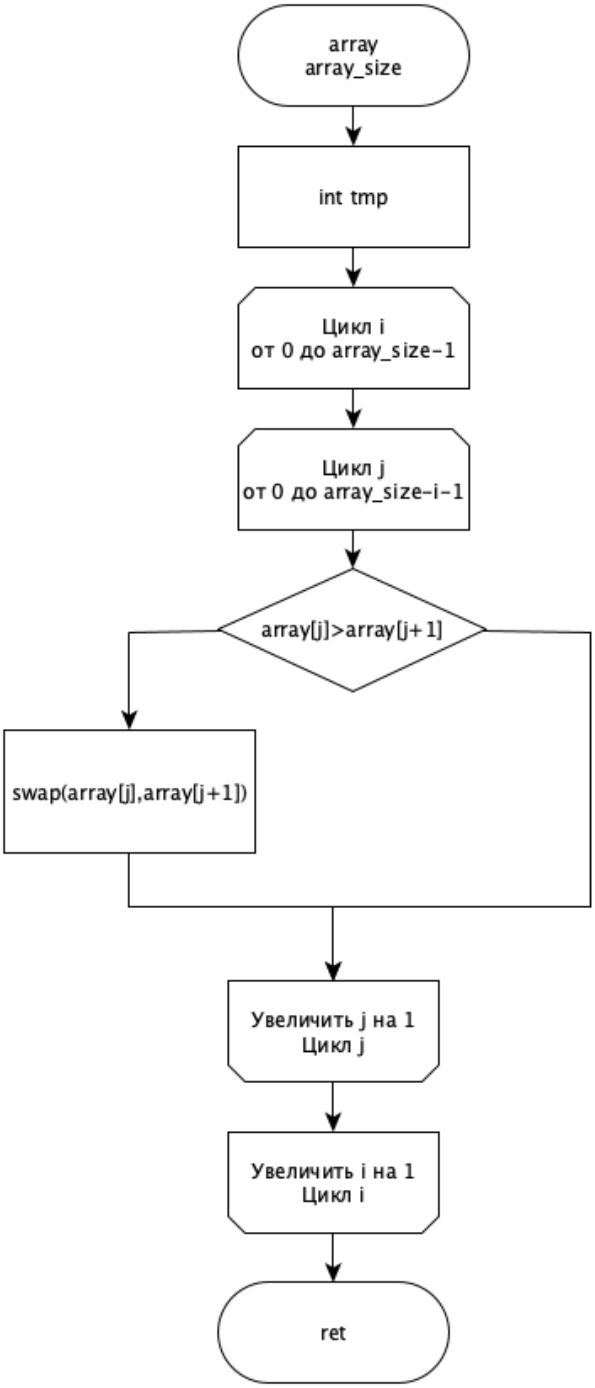

Рис. 2: Сортировка пузырьком

Алгоритм сортировка выбороми:

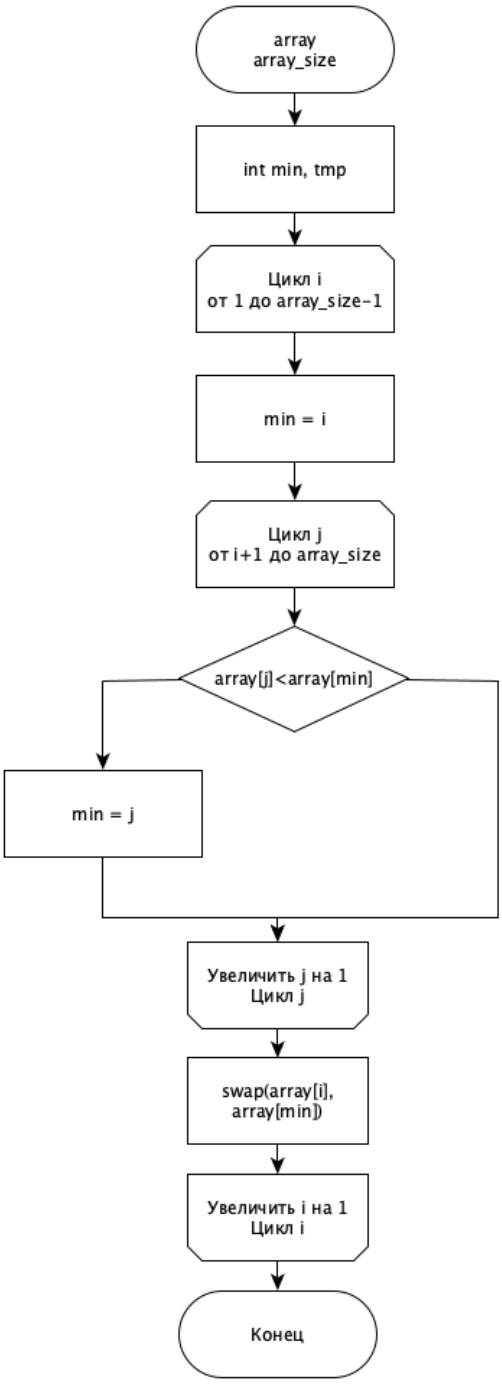

Рис. 3: Сортировка пузырьком

2.2 Вывод

В данной части были рассмотрены схемы алгоритмов.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Требования к программному обеспечению

Входные данные: массив чисел, длина массива. Выходные данные: произведение двух матриц.

Рис. 4: IDEF0 диаграмма сортировки массивов

3.2 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

Для замера процессорного времени используется функция, возвращающая количество тиков.

Листинг 1: Функция замера количества тиков

```
1 unsigned long long tick()
2 {
3     unsigned long long d;
4     __asm__ __volatile__ ("rdtsc" : "=A" (d));
5     return d;
6 }
```

3.3 Листинги функций

Листинг алгоритма сортировки вставками:

Листинг 2: Сортировка вставками

```
void insertion_sort(int* array, int array_size)
{
    int tmp, item;
    for (int i = 1; i < array_size; i++)
    {
        tmp = array[i];
        item = i - 1;
        while (item >= 0 && array[item] > tmp)
        {
            array[item + 1] = array[item];
            array[item] = tmp;
            item--;
        }
    }
}
```

Листинг алгоритма сортировки пузырьком:

Листинг 3: Сортировка пузырьком

```
void bubble_sort(int* array, int array_size)
{
    int tmp;
    for (int i = 0; i < array_size - 1; i++)
    {
        for (int j = 0; j < array_size - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
            }
        }
    }
}
```

Листинг алгоритма сортировки выбором:

Листинг 4: Сортировка выбором

```
void selection_sort(int* array, int array_size)
{
    int min, tmp;
    for (int i = 0; i < array_size - 1; i++)
    {
      min = i;
      for (int j = i + 1; j < array_size; j++)
      {
        if (array[j] < array[min])
          min = j;
      }
      tmp = array[i];
      array[i] = array[min];
      array[min] = tmp;
    }
}
```

3.4 Оценка трудоемкости

Сортировка вставками:
$F = 3 + (N-1) + 9W = 9W + 13N - 10$, где W - число заходов в цикл while
Лучший случай - отсортированный массив - $W = 0$
$F = 13N - 10$
Худший случай - массив, отсортированный в обратную сторону:
$W = \sum_{i=1}^{n-1} i = \frac{(N-1)N}{2}$
$F = 9 \cdot (N-1) \cdot N/2 + 13N - 10 = 4.5N^2 + 8.5N - 10$

3.5 Вывод

В данном разделе были рассмотрены листинги алгоритмов и рассчитана их трудоемкость.

4 Исследовательская часть

В данной части представлены результаты исследования быстродействия алгоритмов.

4.1 Постановка эксперимента

Для экспериментов использовались массивы, размер которых варьируется от 1000 до 10000 с шагом 1000.

Количество повторов каждого эксперимента равно 20. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

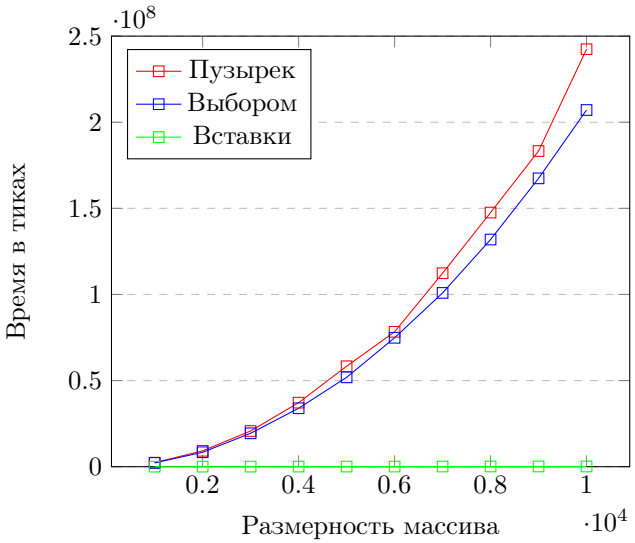

Рис. 5: График лучших случаев сортировок

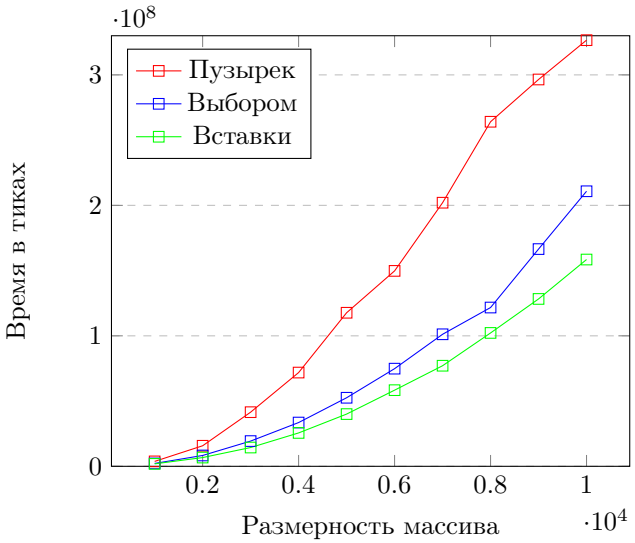

Рис. 6: График рандомных случаев сортировок

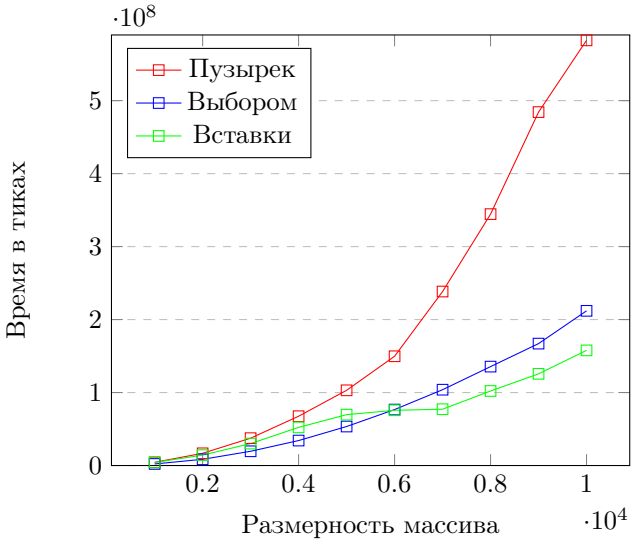

Рис. 7: График худших случаев сортировок

4.2 Вывод

В результате проведенного эксперимента был получен следующий вывод: алгоритм вставок справляется лучше всех во всех трех случаях.

Заключение

В данной лабораторной работе были выполнены следующие задачи:

1. реализованы следующие алгоритмы сортировки массивов – сортировка вставками, сортировка пузырьком и сортировка выбором;
2. была приведена оценка трудоемкости алгоритмов;
3. выполнен анализ времени работы программы на экспериментальных данных;
4. выполнен сравнительный анализ работы алгоритмов для массивов размера от 100 до 1000 элементов.

Список литературы

[1] Кнут Д. Э. 5.2.1 Сортировка путём вставок // Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.

[2] Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. 2.1. Сортировка вставкой // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 3-е изд. — М.: Вильямс, 2013. — С. 38-45. — ISBN 5-8459-1794-8.

[3] Макконнелл Дж. Основы современных алгоритмов = Analysis of Algorithms: An Active Learning Approach / Под ред. С. К. Ландо. — М.: Техносфера, 2004. — С. 72-76. — ISBN 5-94836-005-9.