Министерство образования и науки 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)

ОТЧЕТ
По лабораторной работе №3
По курсу «Анализ алгоритмов»
Тема: «Алгоритмы сортировки»

Студент: Зейналов З. Г.
Группа: ИУ7-51
Преподаватель: Волкова Л.Л.

Москва, 2019г.

Оглавление

Введение

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма. Наша современная жизнь вплотную пересекается с большим объемом данных. Именно поэтому задача сортировки имеет высокое значение для упорядочивания данных и получения результата в удобном виде.

Целью данной лабораторной работы является исследование существующих алгоритмов сортировки массива и исследовать их трудоемкость.

Задачи работы:

Задачами данной лабораторной работы являются:

1. Реализовать три выбранных алгоритма сортировки;
2. Рассчитать трудоемкость одного из алгоритмов, привести рассчитанное значение трудоемкости для двух оставшихся алгоритмов из литературы;
3. Провести экспериментальный анализ по замерам времени.

1. **Аналитическая часть**

В данном разделе будут представлены описания алгоритмов, формулы и оценки сложностей алгоритмов

1.1 Описание алгоритмов

Алгоритм сортировки вставками:

Основная идея данного алгоритма заключается в том, что по левую сторону от рассматриваемого элемента, находится уже отсортированный массив. Для этого элемента выбирается место в этой части массива[2].

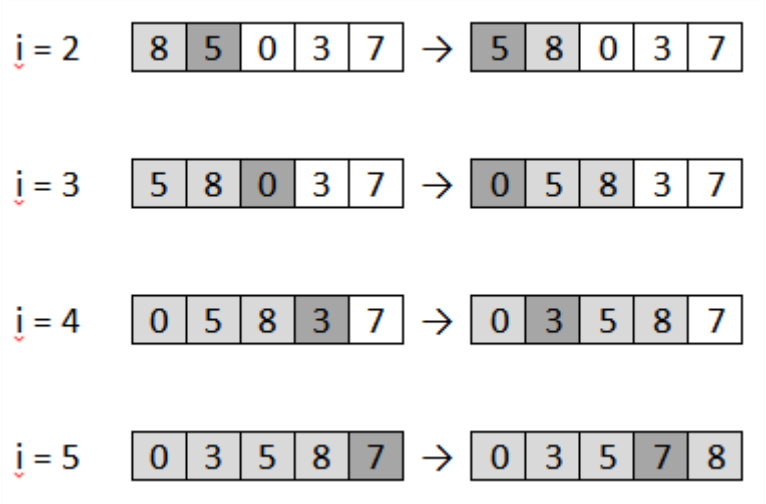

Рисунок 1 - Демонстрация работы алгоритма

Алгоритм «Гномья сортировка»

«Гномья сортировка основана на технике, используемой обыкновенным голландским садовым гномом. Вот как садовый гном сортирует ряд цветочных горшков. По существу, он смотрит на два соседних цветочных горшка, если они находятся в правильном порядке, то он переходит на один горшок вперед, иначе он меняет их местами и возвращается на один горшок назад. Граничные условия: если позади нет ни одного горшка – он шагает вперед, а если нет следующего горшка, тогда он закончил». [1]

Т.е алгоритм работы заключается в том, что у нас существует некий указатель, способный ходить как в одну сторону, так в другую. Таким образом, если мы возьмем 2 элемента массива A: A[i - 1] и A[i], происходит их сравнение, в случае, если они стоят на своих местах (упорядочены), то счетчик i увеличиваем, в обратном случае меняем их местами и уменьшаем счетчик, затем проделываем то же самое.

Алгоритм быстрой сортировки

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Любопытный факт: улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.

Общая идея алгоритма состоит в следующем :

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы [3].

Мы же в нашей реализации будем использовать 2 разделения на «большие либо равные» и на «меньшие».

1.2 Описание модели вычислений

Выберем следующую модель вычислений:

1. Трудоемкость базовых операций
   Операции $+, -, *, /, \%, =, >, <, \geq, \leq, ==, \neq, [\,], +=, -=$ - имеют стоимость 1.
2. Трудоемкость условного перехода
   Условный переход имеет стоимость 0, при этом оцениваем расчет условия:

$$
\begin{aligned}
&if\ (n\ \%\ 2 == 1)\\
&\{\\
&\quad \text{Тело1}\\
&\}\\
&else\\
&\{\\
&\quad \text{Тело2}\\
&\}
\end{aligned}
$$

$$
f_{if} = f_{\text{условия}} + \begin{cases} f_{\text{тело1}}, \text{при нечетном } N \\ f_{\text{тело2}}, \text{при четном} \end{cases}
$$

Трудоемкость цикла

$f_{\text{цикла}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N\left(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}\right)$,

где N – число повторений цикла

1.3 Вывод

В данном разделе была приведена теория, необходимая для понимания и реализации алгоритмов, а также выбрана модель вычислений, в соответствии с которой будут проведены дальнейшие вычисления трудоемкости алгоритмов.

2. Конструкторская часть

В данном разделе будут размещены схемы алгоритмов и расчет трудоемкости алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2 – 6 приведены схемы алгоритмов, описывающие их работу.

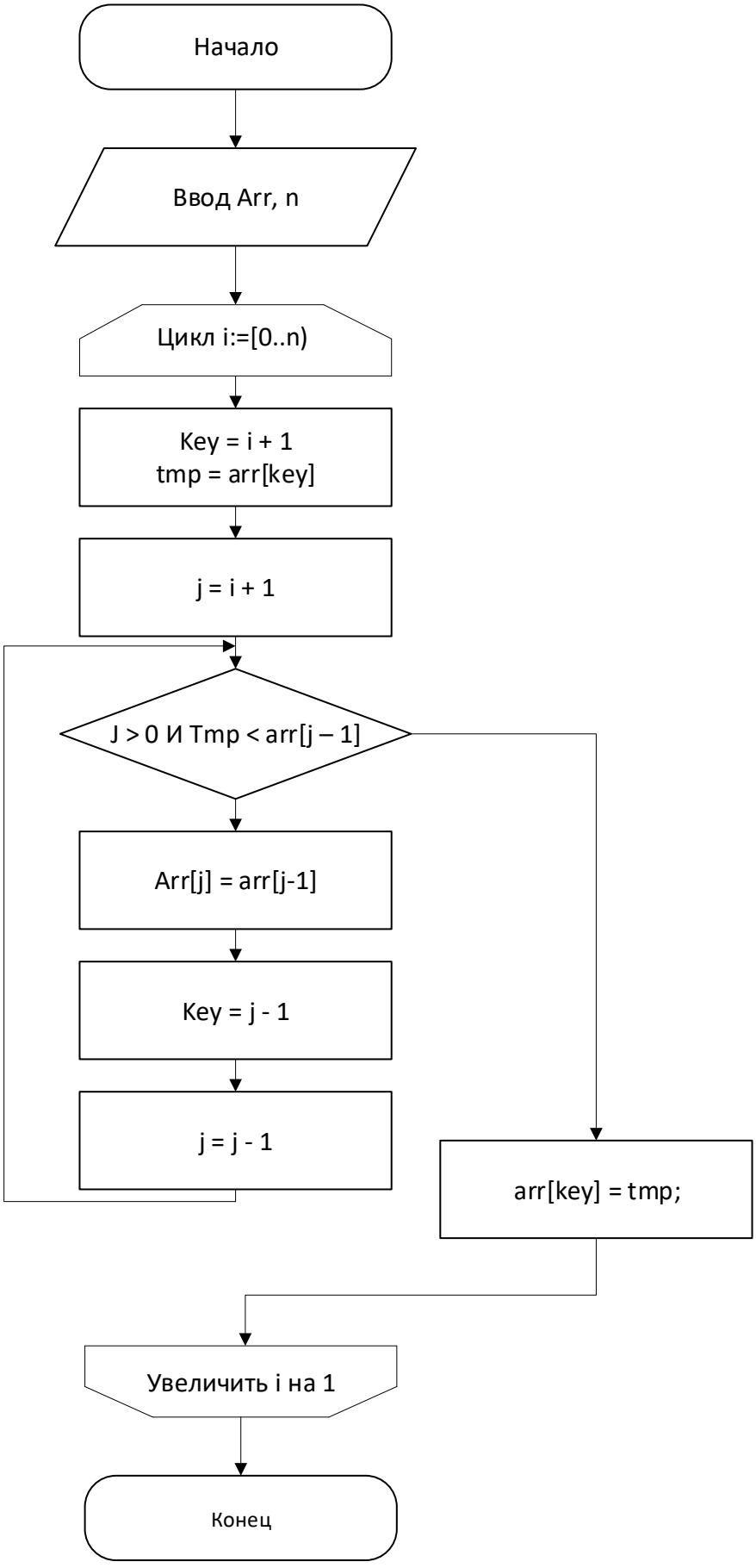

Рисунок 2 - Схема работы алгоритма сортировки вставками

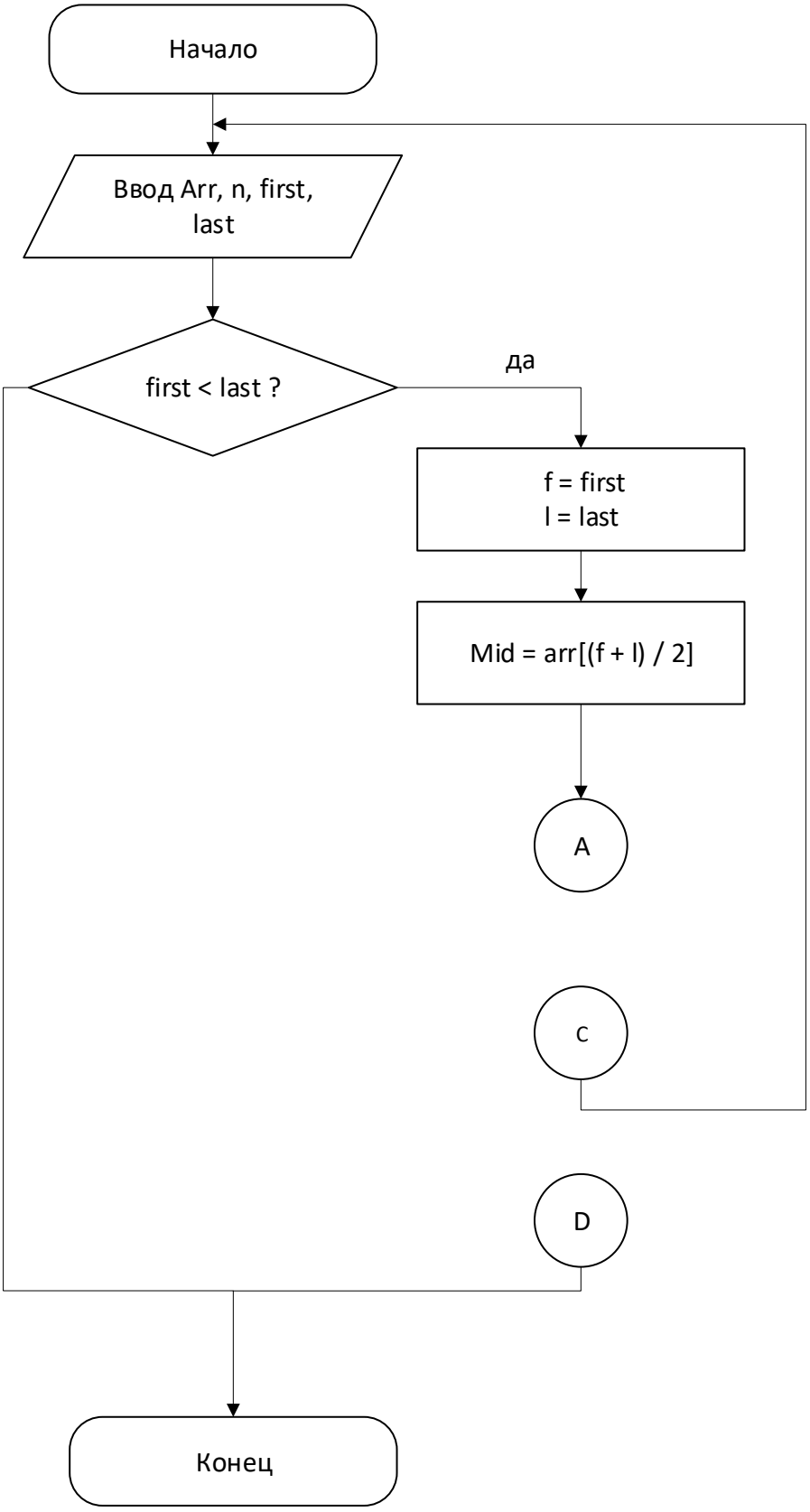

Рисунок 3 - Схема алгоритма быстрой сортировки

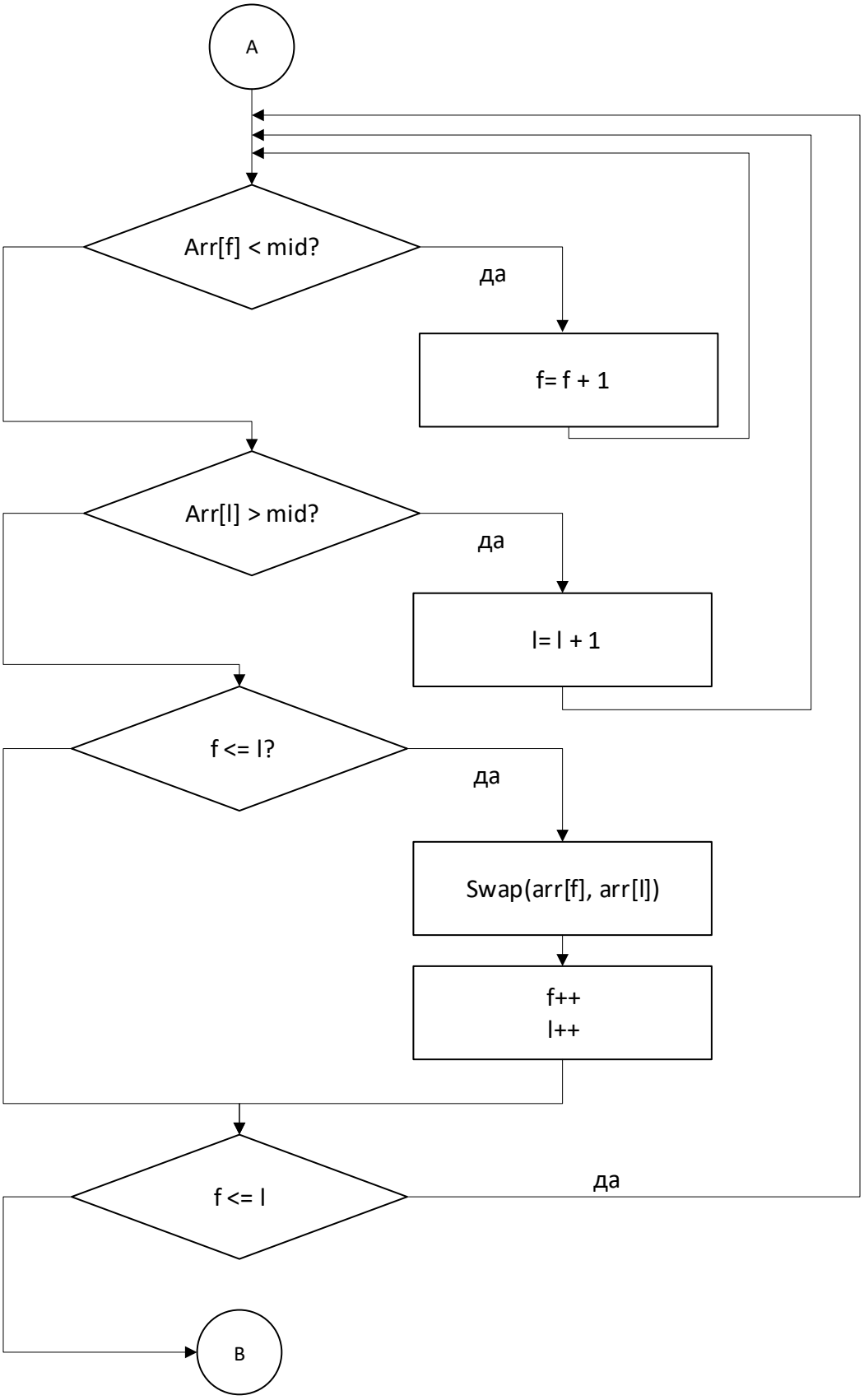

Рисунок 4 - схема работы алгоритма быстрой сортировки (продолжение 1)

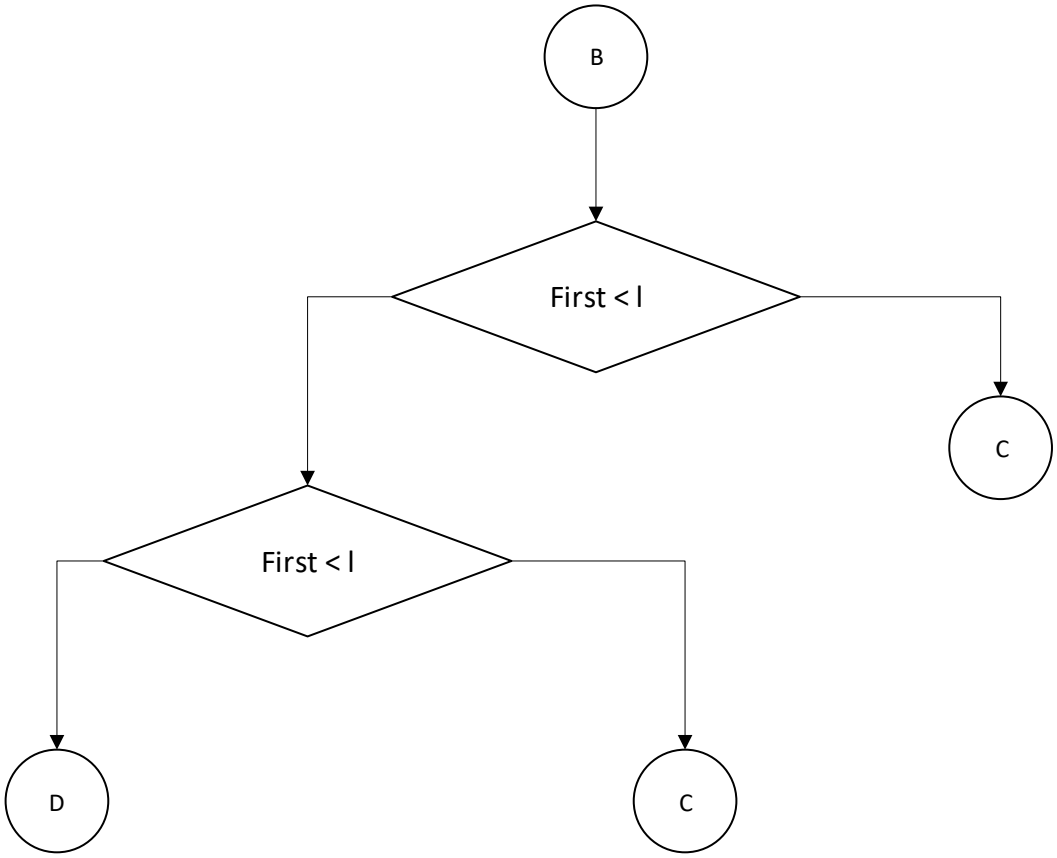

Рисунок 5 - Схема работы быстрой сортировки (продолжение 2)

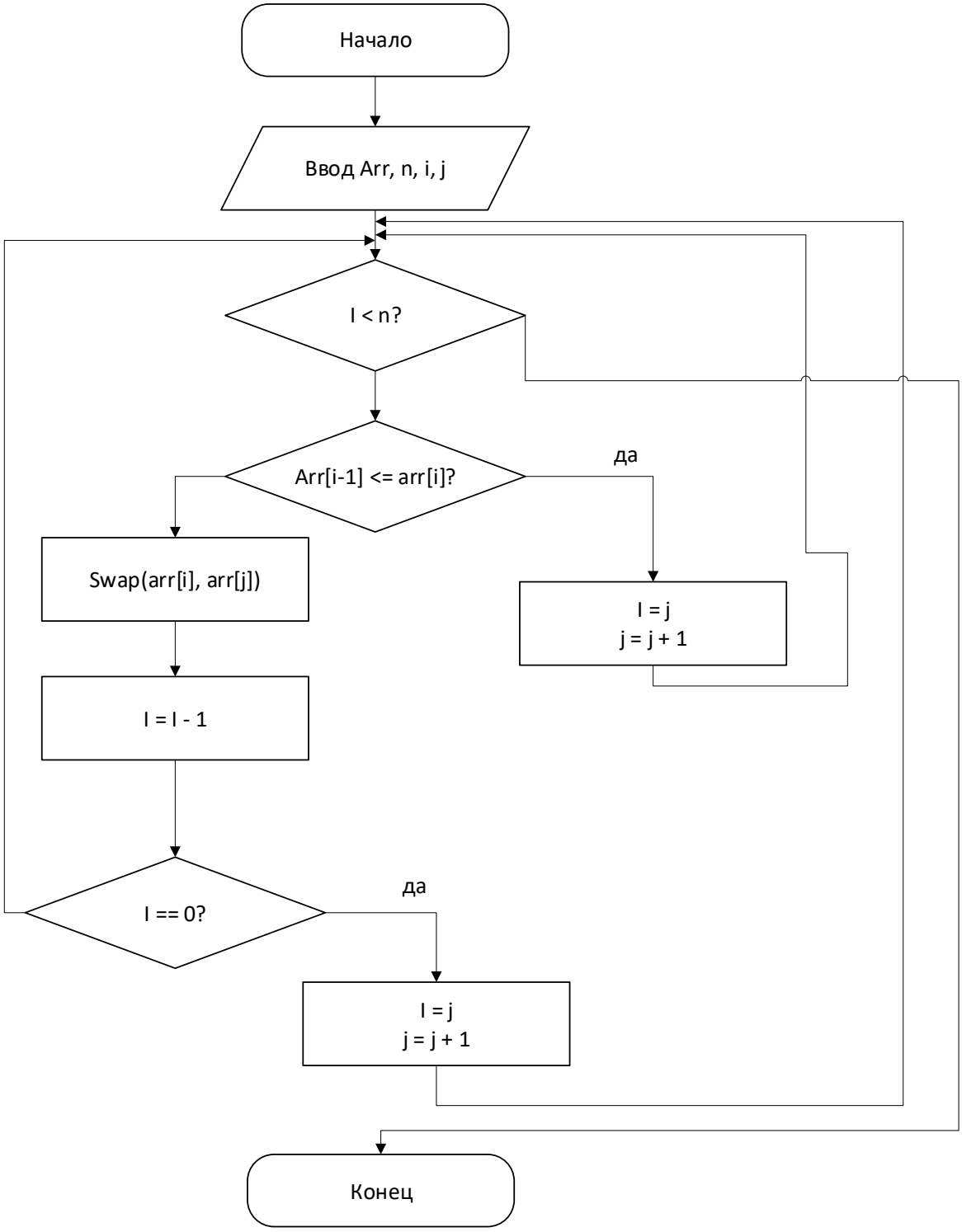

Рисунок 6 - Схема алгоритма Гномьей сортировки

2.2 Расчет трудоемкости

Проведем расчёт трудоемкости для алгоритма сортировки вставками:

- Лучший случай, когда массив уже отсортирован
$$f_{\text{стд}} = 1 + 2 + N(2 + 2 + 2 + 2) \approx O(n)$$
- Худший случай
$$f_{\text{стд}} = 1 + 2 + N(2 + 2 + 2 + 2 + (N - 1)(4 + 3 + 4)) \approx O(n^2)$$

Алгоритм Гномьей сортировки

Алгоритм концептуально простой, не требует вложенных циклов. Время работы $O(n^2)$. На практике алгоритм может работать так же быстро, как и сортировка вставками. [1]

Алгоритм быстрой сортировки

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых под массивов достигнут 1, составит $\log_2 n$. В результате количество сравнений, совершаемых быстрой сортировкой, было бы равно значению рекурсивного выражения $C_n = 2 * C_{\frac{n}{2}} + n$, что даёт общую сложность алгоритма $O(n \log n)$. [3]

2.3 Вывод

В данном разделе были приведены схемы алгоритмов и расчет их трудоемкости.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода.

3.1 Требования к программному обеспечению

На вход подаются 2 целочисленные матрицы, на выходе должен возвращаться результат их умножения либо сообщение о невозможности их умножения.

3.2 Средства реализации

В качестве языка программирования был выбран C++ (компилятор g++) в связи с его широким функционалом и быстротой работы, а так же благодаря привычному для меня синтаксису и семантики языка. Среда разработки - Qt. Время работы процессора замеряется с помощью функции представленной на листинге 1.

Листинг 1 - Функция замера времени

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

    return d;
}
```

3.3 Листинг кода

Ниже на листингах 2 – 4 будут представлены реализации алгоритмов.

Листинг 2. Реализация алгоритма сортировки вставками

```
void arraySort::insertionSort()
{
	for (size_t i = 0; i < this->arraySize - 1; i++)
	{
		int key = i + 1;
		int tmp = this->Array[key];
		size_t j = i + 1;
		while (j > 0 && tmp < this->Array[j - 1])
		{
			this->Array[j] = this->Array[j - 1];
			key = j - 1;
			j = j - 1;
		}
		this->Array[key] = tmp;
	}
}
```

Листинг 3. Реализация алгоритма Гномьей сортировки

```
void arraySort::GnomeSort(size_t i, size_t j)
{
    if (i > this->arraySize || j > this->arraySize)
    {
        return;
    }
    while (i < this->arraySize)
    {
        if (this->Array[i - 1] <= this->Array[i])
        {
            i = j;
            j++;
        }
        else
        {
            int t = this->Array[i];
            this->Array[i] = this->Array[i - 1];
            this->Array[i - 1] = t;
            i--;
            if (i == 0)
            {
                i = j;
                j++;
            }
        }
    }
}
```

Листинг 4. Реализация алгоритма быстрой сортировки

```
void arraySort::quickSort(size_t first, size_t last)
{
    int mid, count;
    int f = first, l = last;
    mid = this->Array[(f + l) / 2]; //вычисление опорного элемента
    do
    {
        while (this->Array[f] < mid)
            f++;
        while (this->Array[l] > mid)
            l--;
        if (f<=l) //перестановка элементов
        {
            count = this->Array[f];
            this->Array[f] = this->Array[l];
            this->Array[l] = count;
            f++;
            l--;
        }
    } while (f < l);
    if (first < l) quickSort(first, l);
    if (f < last) quickSort(f, last);
}
```

3.4 Вывод

В данном разделе были приведены требования к программному обеспечению, средства реализации и листинги реализаций алгоритмов.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1 Примеры работы

Ниже на рисунках 7 – 10 представлены примеры работы программы с использованием реализованных алгоритмов.

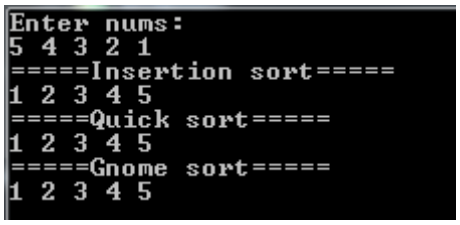

Рисунок 7 - Пример работы алгоритма №1

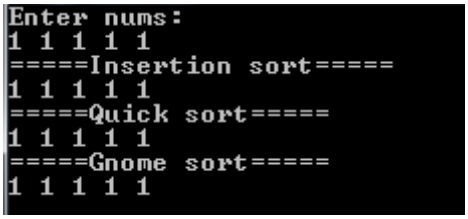

Рисунок 8 - Пример работы алгоритма №1

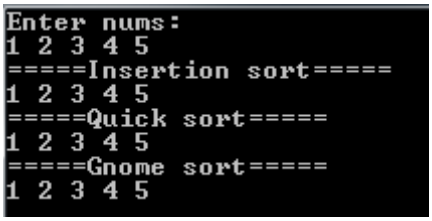

Рисунок 9 - Пример работы алгоритма №3

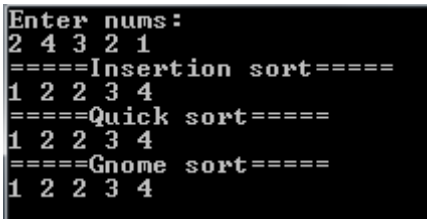

Рисунок 10 - Пример работы алгоритма №4

Программа успешно прошла все тестовые случаи, все полученные результаты совпали с ожидаемыми.

4.2 Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализаций алгоритмов будет использована следующая формула $t = \frac{Tn}{N}$, где t – время выполнения, N – количество замеров. Неоднократное измерение времени необходимо для построения более гладкого графика.

Количество замеров будет взято равным 100.

Тестирование будет проведено на одинаковых входных данных. 1) массивы размерностями от 100x100 до 800x800 с шагом 100.

4.3 Сравнительный анализ на материале экспериментальных данных

Ниже на рисунках 7 – 9 представлен сравнительный временной анализ для различно отсортированных массивов.

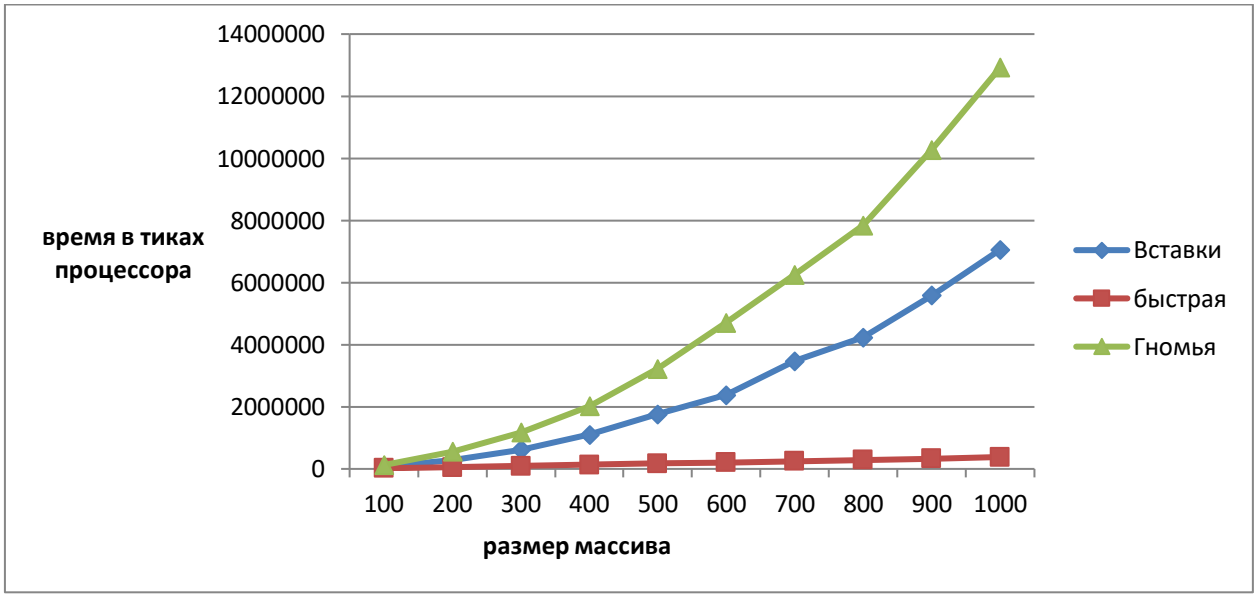

Рисунок 11 - Сравнительный временной анализ алгоритмов на массиве со случайными числами

Из рис. 7 следует, что на массиве со случайными числами алгоритм быстрой сортировки работает быстрее алгоритма сортировки вставками и гномьей сортировки в $\sim\frac{n}{50}$ раз, . Алгоритм же вставок работает быстрее алгоритма гномьей сортировки в ~2 раза.

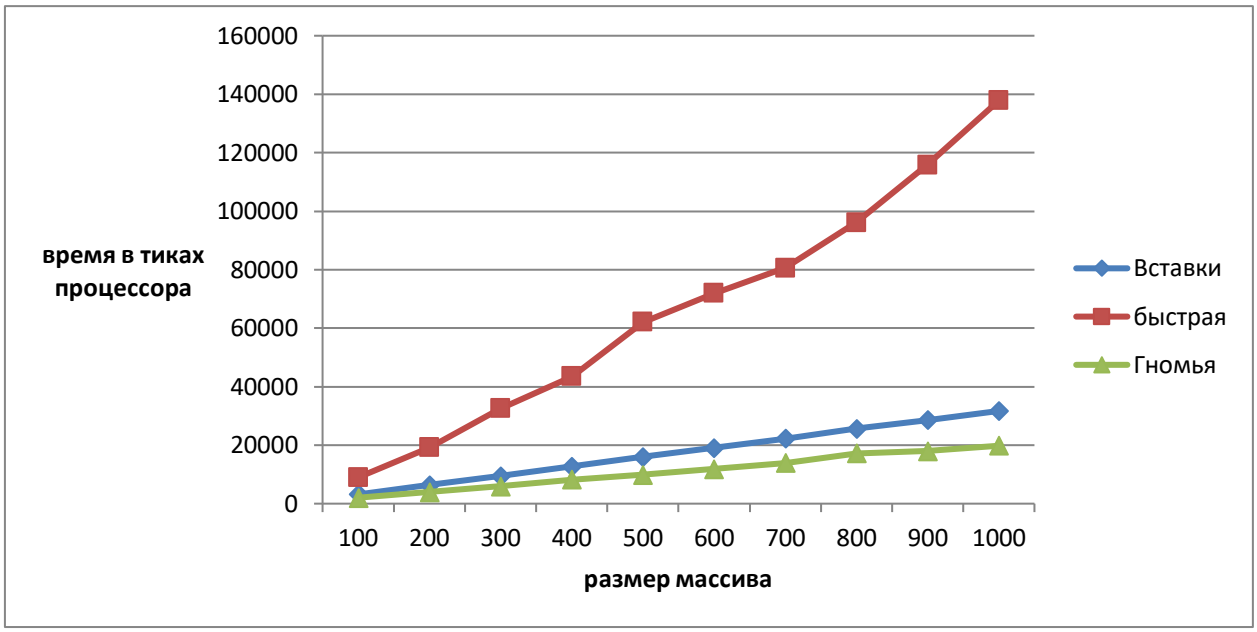

Рисунок 12 - Сравнительный временной анализ алгоритмов на массиве с числами упорядоченными по возрастанию

Из рис. 8 следует, что на уже отсортированном массиве алгоритм Гномьей сортировки показывает лучшие результаты. Алгоритм вставок проигрывает по скорости в ~2 раза алгоритму Гномьей сортировки, однако выигрывает значительно у алгоритма быстрой сортировки в 3 – 4 раза. Таким образом, алгоритм быстрой сортировки показывает худшие результаты на отсортированном массиве чисел.

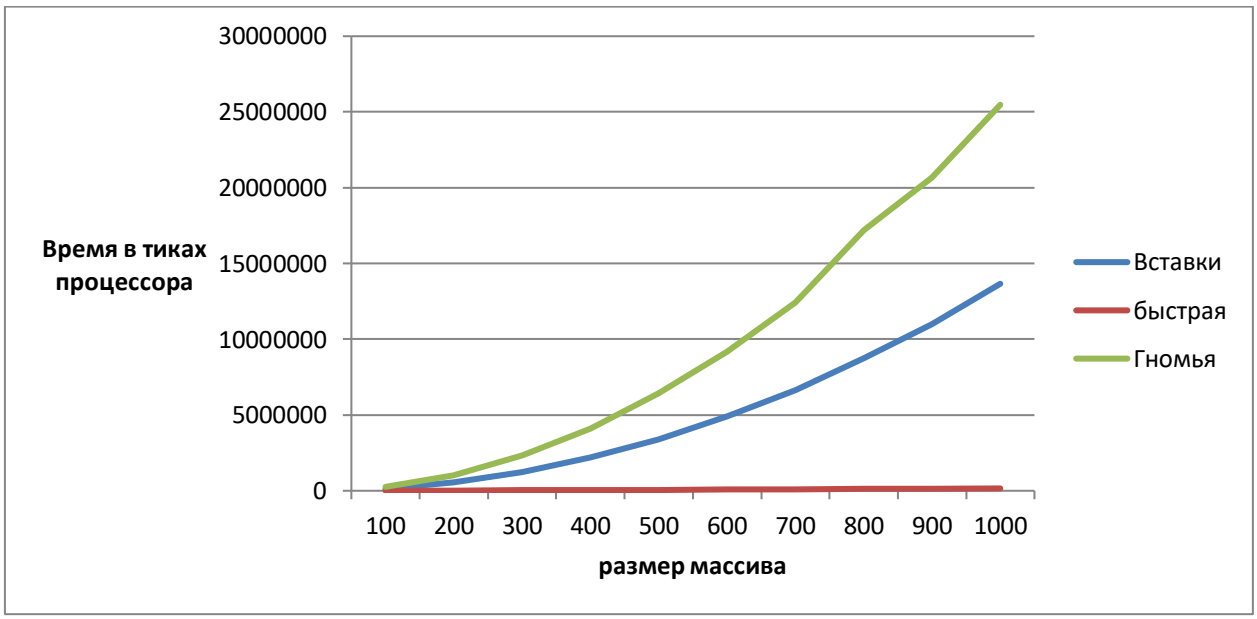

Рисунок 13 - Сравнительный временной анализ алгоритмов на массиве с числами упорядоченными по возрастанию

На рис. 9 использовался уже отсортированный в обратном порядке массиве и ситуация здесь следующая: Алгоритмы вставок и гномьей сортировки квадратично увеличиваются относительно алгоритма быстрой сортировки.

4.4 Вывод

В данном разделе были приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

Заключение

В ходе работы были изучены и реализованы алгоритмы сортировки: вставками, быстрая сортировка и быстрая сортировка. Данные алгоритмы были реализованы и были оценены их трудоемкости. Был произведен сравнительный временной анализ перечисленных алгоритмов. В результате алгоритм сортировки вставками и Гномьей сортировки показывают наиболее приближенные результаты и имеют трудоемкость в лучшем случае $O(n)$ и в худшем случае $O(n^2)$. Алгоритм же быстрой сортировки имея сложность $O(n \log n)$ показывает лучшие результаты на отсортированных массивах не в порядке возрастания.

Литература

1. Kvodo Computing Scince & Discrete Match[Электронный ресурс]// Гномья сортировка. URL: http://kvodo.ru/gnome-sorting.html
2. Kvodo Computing Scince & Discrete Match[Электронный ресурс]// Сортировка вставками. URL: http://kvodo.ru/sortirovka-vstavkami-2.html
3. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 174–179. — 576 с. — ISBN 978-5-8459-0987-9