Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №2

по курсу «Анализ алгоритмов»

Сортировка массивов

Выполнил: Сорокин А.П., гр. ИУ7-52Б

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

Москва, 2019 г.

Оглавление

Введение

В настоящее время в любом программном проекте необходимо обрабатывать большое число однотипных данных. Такие данные удобно обрабатывать, используя массив - именованную последовательность однотипных данных. Благодаря тому, что элементы массива расположены последовательно, упрощается их обработка. Очень часто требуется сортировать данные по какому-либо признаку, или ключу. В отсортированных массивах значительно быстрее можно выполнять поиск определённых данных по ключу. Существует множество алгоритмов сортировки массивов, которые применяются при различных условиях в зависимости от задач, стоящих перед программой. В данной лабораторной работе изучаются некоторые из алгоритмов сортировки массивов на предмет их эффективности.

1. Аналитическая часть

1.1 Задачи

Цель лабораторной работы - изучение трех алгоритмов сортировки массивов. Были выбраны следующие алгоритмы:

- сортировка вставками;
- сортировка слиянием;
- быстрая сортировка.

Для того чтобы добиться поставленной цели, были поставлены следующие задачи:

- изучить и реализовать алгоритмы сортировок массивов;
- оценить трудоёмкости алгоритмов;
- выполнить сравнительный анализ алгоритмов на основе трудоёмкости и используемой памяти;
- оценить и сравнить эффективности алгоритмов по времени.

1.2 Описание алгоритмов

На вход алгоритмов подаётся последовательность n чисел $a_1, a_2, a_3, .., a_N$. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с N элементами. На выходе алгоритмы должны вернуть перестановку исходной последовательности $a'_1, a'_2, a'_3, .., a'_N$, чтобы выполнялось соотношение $a'_1 \leq a'_2 \leq ... \leq a'_N$

1.2.1 Сортировка вставками

В алгоритме сортировки вставками элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.
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.

1.2.2 Сортировка слиянием

В алгоритме сортировки слиянием массив разбивается на два массива одинакового размера (от среднего элемента). Каждая из этой частей разбивается таким же образом до тех пор, пока длина массива не станет равной 1. Затем выполняется слияние двух подмассивов: на каждом шаге берётся меньший из двух первых элементов подмассивов и записывается в результирующий. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1. Как только заканчивается один из подмассивов, оставшиеся элементы второго записываются в конец результирующего. [1]

1.2.3 Быстрая сортировка

В алгоритме быстрое сортировки исходный массив разбивается на два отрезка по опорному элементу: на одном отрезке находятся все элементы, меньшие опорного, во втором - большие (или равные). Для выставления правильного порядка относительно опорного элемента выполняется перестановка. При этом в массиве в начале идёт отрезок, в котором расположены меньшие элементы, а затем - в котором большие или равные. Затем для каждого из этих отрезков выполняется то же разбиение рекурсивно. Разделение на отрезка продолжается, пока длина отрезка больше единицы. [2]

1.2.4 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

- операции, имеющие трудоемкость 1: <, >, =, <=, =>, ==, ! =,+, −, ∗, /, +=, −=, ∗=, /=, [ ];
- оператор условного перехода имеет трудоёмкость, равную трудоёмкости операторов тела условия;
- оператор цикла for имеет трудоемкость:

$$F_{for} = F_{init} + F_{check} + N * (F_{body} + F_{inc} + F_{check}), \tag{1.1}$$

  где F_{init} – трудоёмкость инициализации, F_{check} – трудоёмкость проверки условия, F_{inc} – трудоёмкость инкремента аргумента, F_{body} – трудоёмкость операций в теле цикла, N – число повторений. [3]

2. Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов трёх алгоритмов сортировки массивов. Сортировка выполняется по возрастанию.

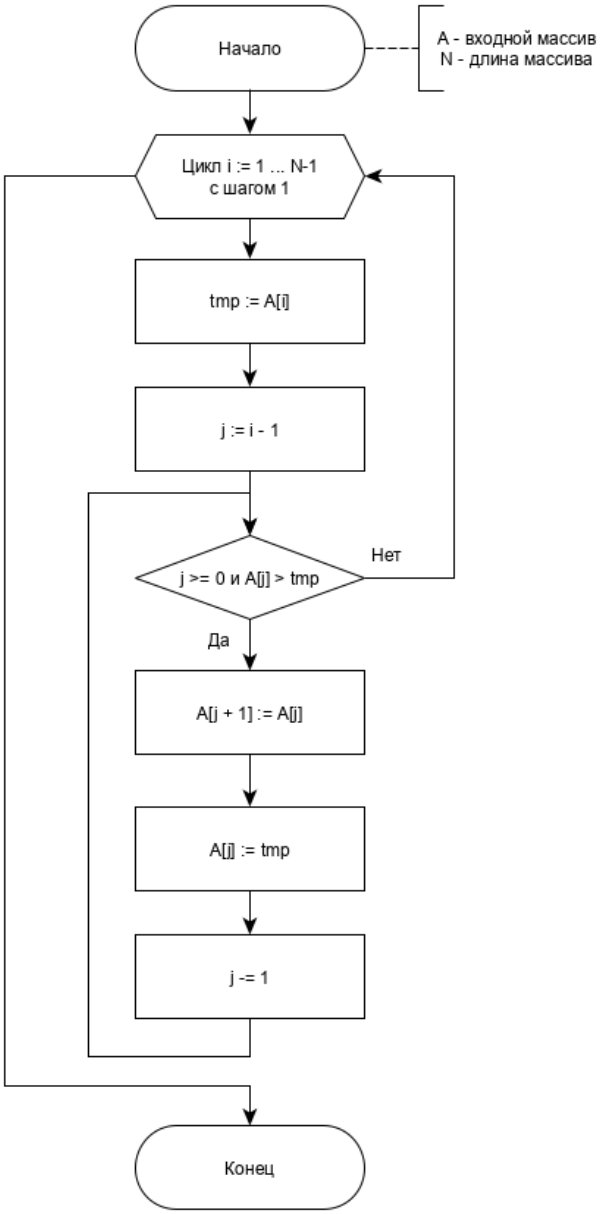

Рис. 2.1: Схема алгоритма сортировки вставками

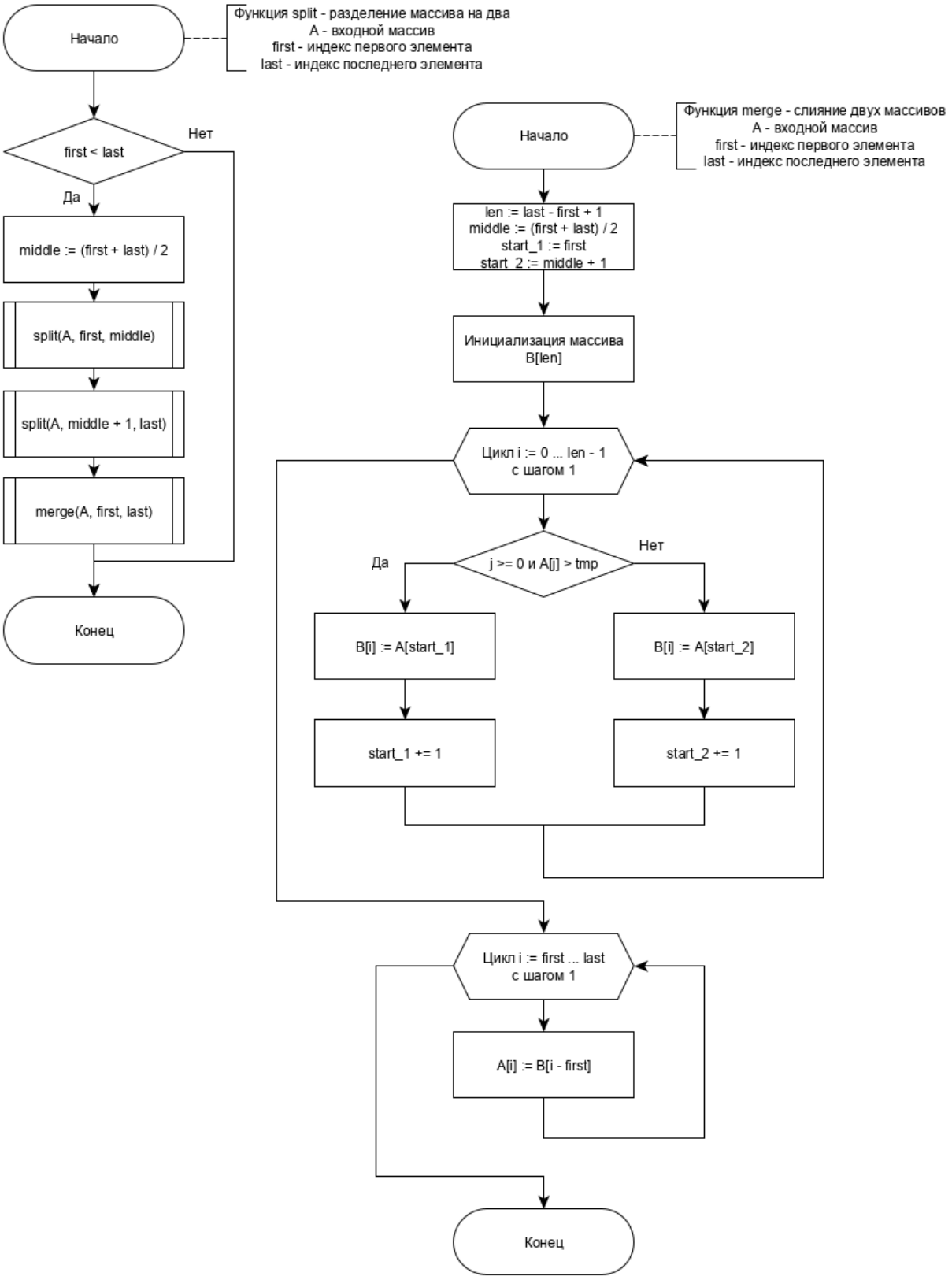

Рис. 2.2: Схема алгоритма сортировки слиянием

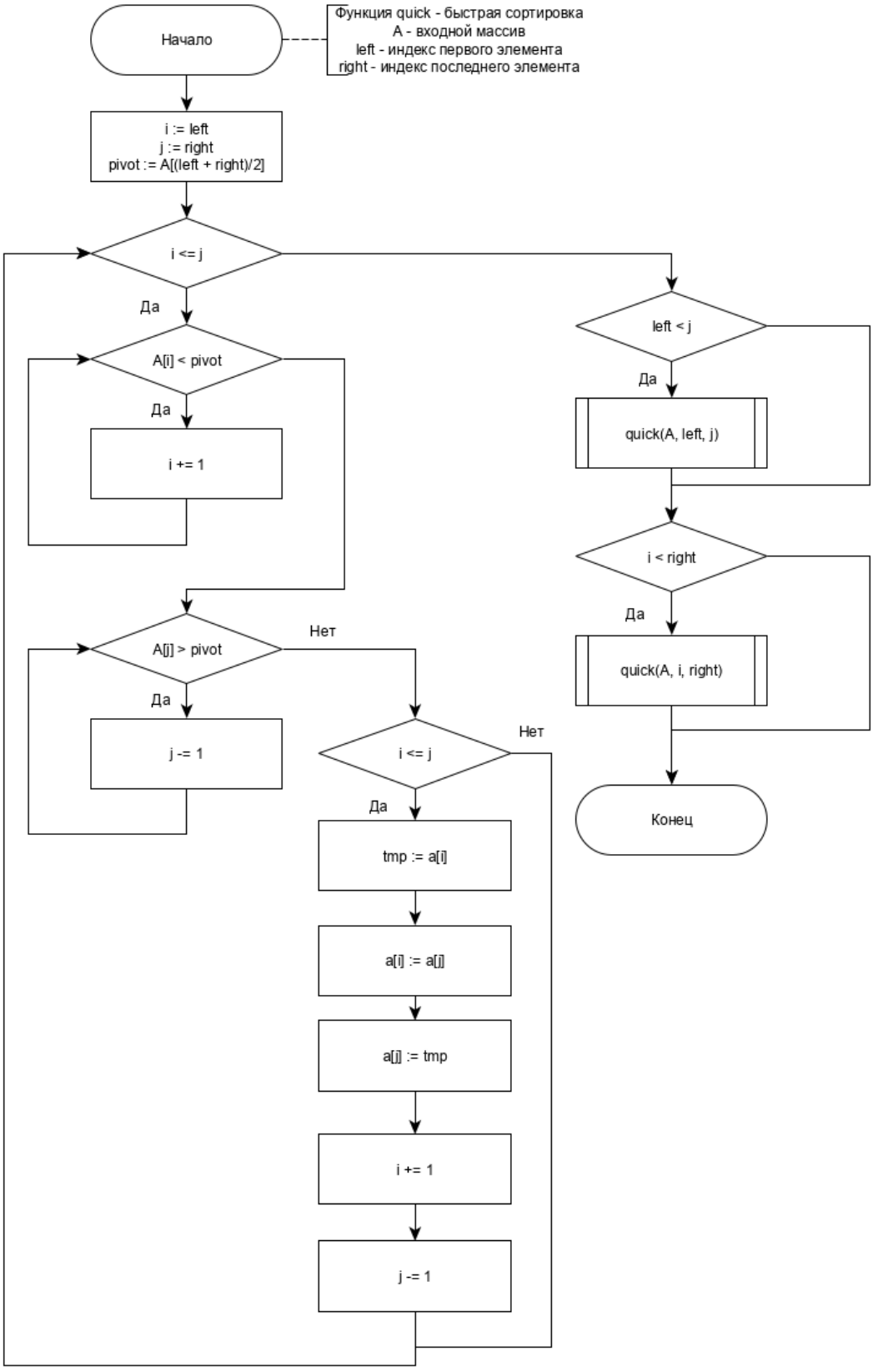

Рис. 2.3: Схема алгоритма быстрой сортировки

2.2 Оценка трудоёмкости

Пусть дан массив A длиной N. Рассмотрим трудоемкость трёх алгоритмов сортировки.

2.2.1 Сортировка вставками

1. Худший случай - массив отсортирован в обратном порядке:
   - Число сравнений в теле цикла: $(N-1)\frac{N}{2}$
   - Число сравнений в заголовках циклов: $(N-1)\frac{N}{2}$
   - Общее число сравнений: $(N-1)N$
   - Число присваиваний в заголовках циклов: $(N-1)\frac{N}{2}$
   - Число обменов: $(N-1)\frac{N}{2}$
   - Итоговая трудоёмкость: $2(N-1)N = 2N^2 - N$
2. Лучший случай - массив уже отсортирован:
   - Число сравнений в теле цикла: $(N-1)\frac{N}{2}$
   - Число сравнений в заголовках циклов: $(N-1)\frac{N}{2}$
   - Общее число сравнений: $(N-1)N$
   - Число присваиваний в заголовках циклов: $(N-1)\frac{N}{2}$
   - Число обменов: 0
   - Итоговая трудоёмкость: $\frac{3}{2}(N-1)N = \frac{3}{2}N^2 - \frac{3}{2}N$

2.2.2 Сортировка слиянием

И в худшем, и в лучшем случаях число операций глубина рекурсии $h = logN$:

- Трудоёмкость разбиения: 5
- Трудоёмкость инициализации пределов и середины: 9
- Трудоёмкость цикла слияния: $2 + N \cdot (15 + 2) = 2 + 17N$
- Трудоёмкость переноса результат слияния: $2 + N \cdot (3 + 2) = 5N + 2$
- Итоговая трудоёмкость: $logN \cdot (5 + 9 + 2 + 17N + 5N + 2) = 22NlogN + 18logN$

2.2.3 Быстрая сортировка

1. Худший случай - каждый раз выбирается один отрезок длиной 1:
   - Глубина рекурсии: $h = N$
   - Выбор опорного элемента: 4
   - Трудоёмкость цикла: $2 + N \cdot (2 + 1) = 3N + 2$
   - Перестановка элементов: 10

- Итоговая трудоёмкость: $N \cdot (3N + 16) = 3N^2 + 16N$

2. Лучший случай - каждый раз массив разделяется на два одинаковых по длине отрезка:

   - Глубина рекурсии: $h = logN$
   - Выбор опорного элемента: 4
   - Трудоёмкость цикла: $2 + 2 \cdot \frac{N}{2} \cdot (2 + 1) = 3N + 2$
   - Перестановка элементов: 10

   - Итоговая трудоёмкость: $logN \cdot (3N + 16) = 3NlogN + 16logN$

2.3 Замер используемой памяти

В каждом из алгоритмов требуется хранить исходный массив A длиной N. Таким образом, под хранение массива требуется N байт памяти.
Однако рекурсивные алгоритмы сортировки при любых случаях требуют использование дополнительной памяти под временное хранение элементов массива. Длина таких временных массивов зависит от N.

2.4 Сравнительный анализ алгоритмов

Исходя из проведённых анализов, можно заметить, что несмотря на то, что алгоритм вставками не использует большое количество дополнительной памяти (только под итераторы циклов), данный алгоритм уступает в трудоёмкости в лучших случаях рекурсивным алгоритмам сортировки слиянием и быстрой сортировки. Однако данные рекурсивные алгоритмы требуют дополнительно память, соизмеримую с размером массива (напрямую зависит от N).
Также стоит отметить, что выбор опорного элемента в алгоритме быстрой сортировки может коренным образом повлиять на трудоёмкость алгоритма. В худшем случае данный алгоритм уступает алгоритму вставками и по трудоёмкости, и по памяти.

3. Технологическая часть

3.1 Требования к программному обеспечению

На вход подаются размер массива и сам массив. На выход программа выдаёт три массива, которые являются результатами работы трёх различных алгоритмов сортировки. Сортировка выполняется по возрастанию.

3.2 Средства реализации

Для реализации программы был использован язык C++ [4]. Для замера процессорного времени была использована функция rdtsc() из библиотеки stdrin.h.

3.3 Реализации алгоритмов

В листингах 3.1, 3.2, 3.3 представлены коды реализации алгоритмов сортировки массивов.

Листинг 3.1: Сортировка вставками

```
1  void array_sort_insert(int * const arr, size_t n)
2  {
3    for (size_t i = 1; i < n; i++)
4    {
5      int tmp = arr[i];
6      int j = i - 1;
7      while (j >= 0 && arr[j] > tmp)
8      {
9        arr[j + 1] = arr[j];
10       arr[j] = tmp;
11       j--;
12     }
13   }
14 }
```

Листинг 3.2: Сортировка слиянием

```
1  void array_merge(int * const arr, size_t first, size_t last)
2  {
3    size_t len = last - first + 1;
4    size_t middle = (first + last) / 2;
5    size_t start_1 = first;
6    size_t start_2 = middle + 1;
7
8    int *arr_tmp = new int[len];
9
```

```
10   for (size_t i = 0; i < len; i++)
11     if ((start_1 <= middle) && ((start_2 > last) ||
12       (arr[start_1] < arr[start_2])))
13     {
14       arr_tmp[i] = arr[start_1];
15       start_1++;
16     }
17     else
18     {
19       arr_tmp[i] = arr[start_2];
20       start_2++;
21     }
22
23   for (size_t i = first; i <= last; i++)
24     arr[i] = arr_tmp[i - first];
25   delete [] arr_tmp;
26 }
27
28 void array_sort_merge(int * const arr, size_t first, size_t last)
29 {
30   if (first < last)
31   {
32     size_t middle = (first + last) / 2;
33     array_sort_merge(arr, first, middle);
34     array_sort_merge(arr, middle + 1, last);
35     array_merge(arr, first, last);
36   }
37 }
```

Листинг 3.3: Быстрая сортировка

```
1  void array_sort_quick(int * const arr, size_t left, size_t right)
2  {
3    unsigned i = left, j = right;
4    int pivot = arr[(left + right) / 2];
5
6    while (i <= j)
7    {
8      while ((i < j) && (arr[i] < pivot))
9      i++;
10
11     while ((i < j) && (arr[j] > pivot))
12     j--;
13
14     if (i <= j)
15     {
16       int tmp = arr[i];
17       arr[i] = arr[j];
18       arr[j] = tmp;
19       if (i < right)
20         i++;
21       if (j > left)
22         j--;
23     }
```

```
24    };
25
26    if (left < j)
27      array_sort_quick(arr, left, j);
28
29    if (i < right)
30      array_sort_quick(arr, i, right);
31  }
```

3.4 Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице 3.1. Все алгоритмы должны выдавать на выходе одинаковые результаты. Сортировка выполняется по возрастанию.

Таблица 3.1: Функциональные тесты

Массив	**Ожидание**
$[1]$	$[1]$
$[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$	$[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$
$[9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0]$	$[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$
$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$	$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$[5\ 4\ 0\ 2\ -1\ 4]$	$[-1\ 0\ 2\ 4\ 4\ 5]$
$[5\ 4\ 0\ 2\ -1\ 3]$	$[-1\ 0\ 2\ 3\ 4\ 5]$

В результате проверки реализации всех алгоритмов сортировки прошли все поставленные функциональные тесты.

4. Экспериментальная часть

4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы, демонстрирующий корректную работу алгоритмов.

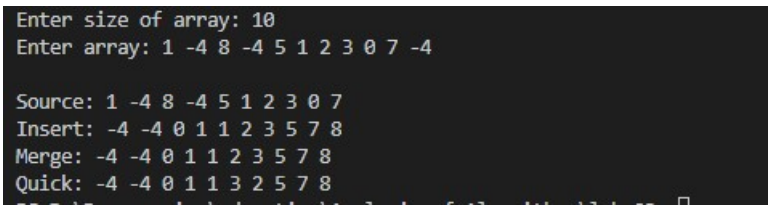

Рис. 4.1: Пример работы программы

4.2 Сравнение времени работы алгоритмов

4.2.1 Лучшее время

Для сравнения лучшего времени работы алгоритмов сортировки массивов были использованы отсортированные массивы длиной от 100 до 1000 с шагом 50. Эксперимент для более точного результата повторялся 1000 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.2.

Таблица 4.1: Лучшее время работы алгоритмов сортировки массивов в тактах процессора

Размер массива	Сортировка вставками	Сортировка слиянием	Быстрая сортировка
100	666	28985	5165
200	1621	59163	11456
300	2701	96452	17989
400	3036	129188	25595
500	3580	159770	31671
600	4316	198736	39545
700	5130	229305	48898
800	5779	272229	57898
900	6368	304811	64339
1000	7272	340182	71289

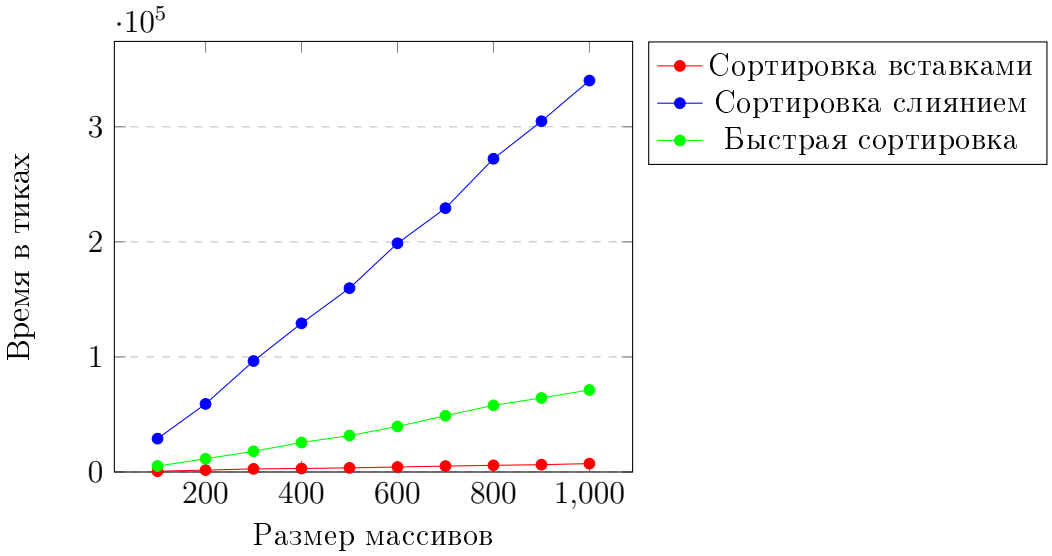

Рис. 4.2: График лучшего времени работы алгоритмов сортировки массивов

Можно заметить, что при уже отсортированном массиве на входе по времени выигрывает самый простой в реализации алгоритм - сортировка вставками, он быстрее быстрой сортировки в 10 раз, а сортировки слиянием - в 50 раз. Также можно заметить, что алгоритм сортировки слиянием значительно проигрывает по времени двум другим алгоритмам из-за большого числа сравнений, которое выполняется в данном алгоритме независимо от входных данных.

4.2.2 Среднее время

Для сравнения среднего времени работы алгоритмов сортировки массивов были использованы случайно сгенерированные массивы длиной от 100 до 1000 с шагом 50. Эксперимент для более точного результата повторялся 1000 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунке 4.3.

Таблица 4.2: Среднее время работы алгоритмов сортировки массивов в тактах процессора

Размер массива	Сортировка вставками	Сортировка слиянием	Быстрая сортировка
100	24574	42900	5228
200	102615	96159	12812
300	205502	144138	19526
400	352258	188735	27116
500	540030	241566	33963
600	783849	296684	42716
700	1050632	350260	51462
800	1317654	386488	58100
900	1638597	431168	63785
1000	2035657	498586	72418

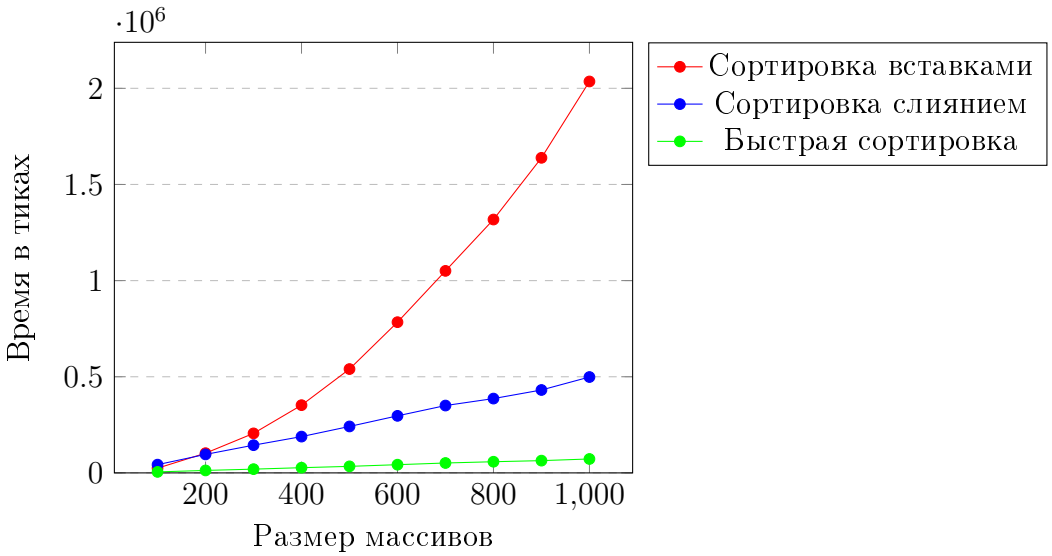

Рис. 4.3: График среднего времени работы алгоритмов сортировки массивов

По среднему времени рекурсивные алгоритмы выигрывают простую сортировку вставками, что следует из трудоёмкости алгоритмов. Разница во времени растёт с увеличением размера массива: так при размере массива 500 сортировкой вставками медленее сортировки слиянием в 2,2 раза, а при размере 100 - в 4 раза. При этом можно заметить, что алгоритм сортировки вставками работает примерно за то же время, что и алгоритм сортировки слиянием, при размерах, меньших 200. Сортировками слиянием в среднем работает в 2 раза медленнее быстрой сортировки.

4.2.3 Худшее время

Для сравнения среднего времени работы алгоритмов сортировки массивов были использованы отсортированные в обратном порядке массивы длиной от 100 до 1000 с шагом 50. Эксперимент для более точного результата повторялся 1000 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.3 и на рисунке 4.4.

Таблица 4.3: Худшее время работы алгоритмов сортировки массивов в тактах процессора

Размер массива	Сортировка вставками	Сортировка слиянием	Быстрая сортировка
100	46751	31584	5327
200	179939	73565	12361
300	393600	108477	19410
400	701277	141741	28971
500	1139428	213607	36273
600	1533193	212551	45503
700	1993591	237489	48489
800	2581057	277803	58035
900	3368935	319167	108210
1000	3993100	348872	71125

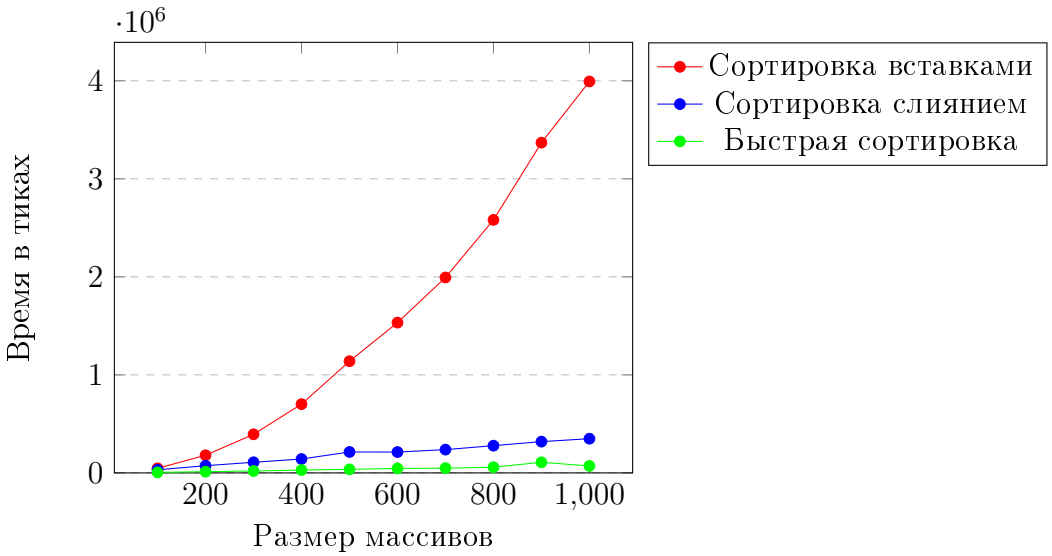

Рис. 4.4: График худшего времени работы алгоритмов сортировки массивов

В худшем случае лишь увеличивается разница во времени выполнения: сортировка слиянием работает медленее быстрой сортировки в 5 раз, при размере массива 500 сортировкой вставками медленее сортировки слиянием в 6,2 раза, а при размере 100 - в 12 раз.

Заключение

В ходе лабораторной работы были изучены и реализованы три алгоритма сортировки: вставками, слиянием и быстрая сортировка. Был проведён сравнительный анализ алгоритмов, который показал, что при различных исходных данных применительны каждый из данных алгоритмов сортировки. Также экспериментально были подтверждены временные различия в работе алгоритмов.

Литература

[1] Левитин А. В. Глава 4. Метод декомпозиции: Сортировка слиянием // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006

[2] Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006

[3] Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р.М. Ривест: – МЦНТО, 1999.

[4] https://cppreference.com/ [Электронный ресурс]