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»

Отчет по лабораторной работе №3 по дисциплине "Анализ алгоритмов"

Тема Анализ алгоритмов сортировки

Студент Малышев И. А.

Группа ИУ7-51Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2021 г.

Оглавление

Введение

Сортировка является одной из важнейших задач обработки информации. Под сортировкой понимают процесс упорядочивания элементов по какому-либо признаку в заданном массиве элементов. Например, текстовые данные можно отсортировать в лексикографическом порядке, а числовые в порядке неубывания или невозрастания.

В настоящее время многие программные системы работают с большим объёмом данных, поэтому возникает задача ускорения процесса обработки. Именно эту задачу и решают алгоритмы сортировки. Так, время поиска в отсортированном массиве пропорционально логарифму количества элементов, а в неотсортированном - пропорционально количеству элементов, что значительно медленнее.

Важной характеристикой любого алгоритма сортировки является скорость его работы, то есть время, за которое данные будут отсортированы. Время сортировки будет зависеть от количества сравнений и перестановок элементов, а также от длины массива данных.

Поэтому **целью** данной работы является получить навыки сравнительного анализа на примере 3 алгоритмов сортировки: сортировки пузырьком, сортировки выбором и быстрой сортировки.

Для достижения поставленной цели необходимо решить следующие **задачи**:

- изучить алгоритмы сортировки;
- провести сравнительный анализ алгоритмов на основе теоретических расчётов;
- реализовать алгоритмы сортировки;
- протестировать реализованные алгоритмы;
- провести сравнительный анализ реализаций алгоритмов по затраченному процессорному времени.

1 Аналитическая часть

В этом разделе приведён обзор и анализ алгоритмов сортировки.

1.1 Сортировка пузырьком

Алгоритм состоит из последовательных проходов по массиву. За один проход элементы всех пар стоящих рядом элементов сравнивают друг с другом и, если они стоят в неверном порядке, меняют их местами. Проходы повторяются $N - 1$ раз, где N - длина массива. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим "наибольшим элементом", а наименьший элемент массива перемещается на одну позицию к началу массива ("всплывает" до нужной позиции, как пузырёк в воде – отсюда и название алгоритма).

Из описания алгоритма очевидна модификация, ускоряющая его: если за проход не было обменов, то все элементы стоят на своих местах, следовательно, можно закончить выполнение сортировки (т. н. Сортировка пузырьком с флагом).

При этом важно заметить, что алгоритм является устойчивым, то есть одинаковые по заданному признаку элементы останутся в том же порядке, что и до сортировки.

1.2 Сортировка выбором

Идея алгоритма заключается в том, что массив делят на 2 части: отсортированную и неотсортированную. Части отделяет текущий элемент. Текущий элемент сравнивают с каждым элементом из неотсортированной части. Среди них находят минимальный элемент, после этого производится обмен местами с текущим элементом и граница смещается на один элемент к концу массива. Алгоритм заканчивается тогда, когда отсортированной частью массива будет являться весь массив.

Важно заметить, что алгоритм не является устойчивым, т. е. элементы, которым не нужно менять порядок, всё равно будут его менять во время работы алгоритма.

1.3 Быстрая сортировка

Идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но может зависеть его эффективность. Обычно выбирают средний элемент.

2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на два отрезка: «меньшие опорного» и «равные и большие».

3. Для двух отрезков рекурсивно применяем ту же последовательность действий, пока длина отрезка больше единицы.

1.4 Вывод

В данном разделе были описаны идеи трёх алгоритмов сортировки: сортировки пузырьком, выбором и быстрой сортировки.

2 | Конструкторская часть

В этом разделе приводятся схемы алгоритмов сортировки, приведённых в аналитической части, и расчёты их трудоёмкости.

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2 и 2.3 показаны схемы алгоритмов сортировки пузырьком, выбором и быстрой сортировки соответственно.

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса;
2. оценка трудоемкости цикла: $F_{ц} = a + N * (a + F_{тела})$, где a - условие цикла;
3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов.

2.2.1 Сортировка пузырьком с флагом

Построчная оценка трудоемкости сортировки пузырьком с флагом (Табл. 2.1).

Табл. 2.1 Построчная оценка трудоемкости для алгоритма сортировки пузырьком с флагом

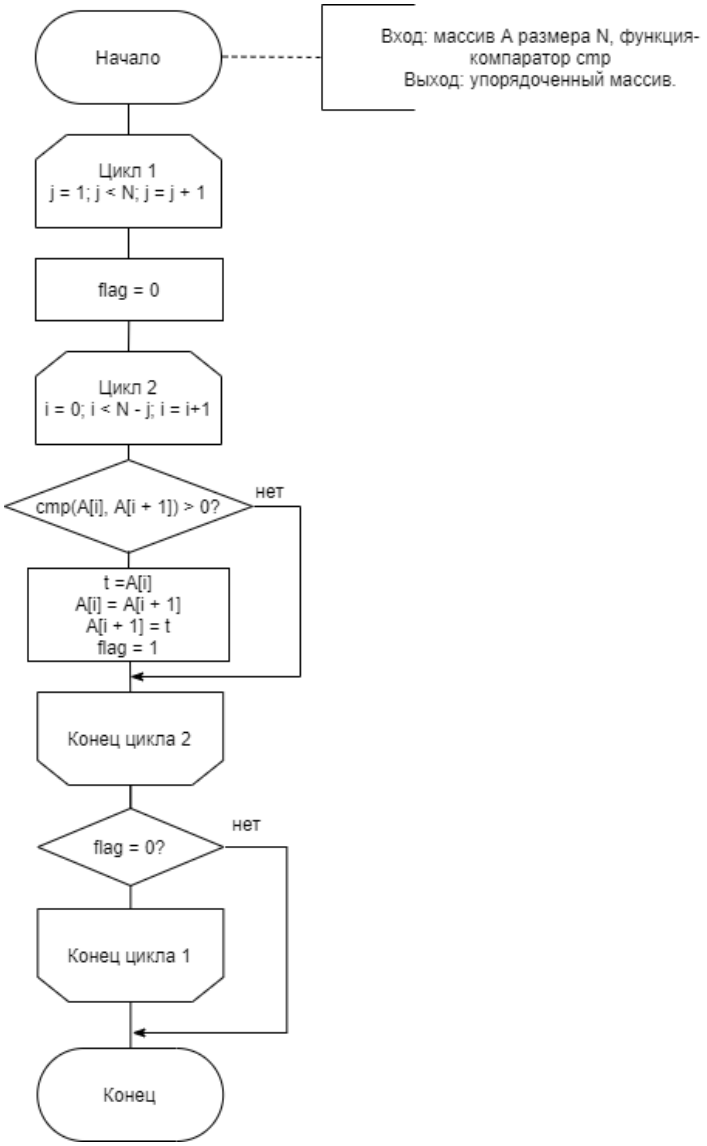

Рис. 2.1: Схема сортировки пузырьком

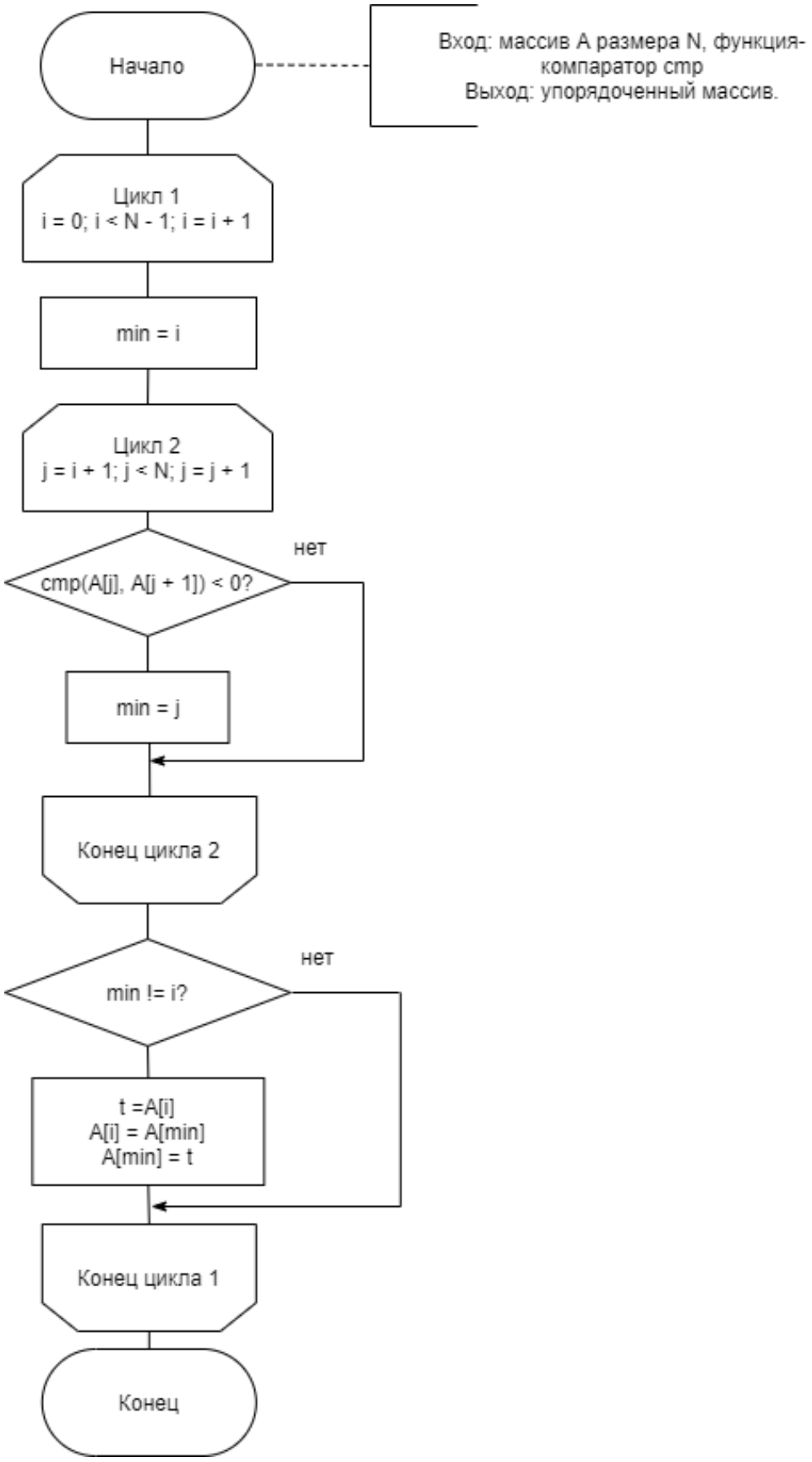

Рис. 2.2: Схема сортировки выбором

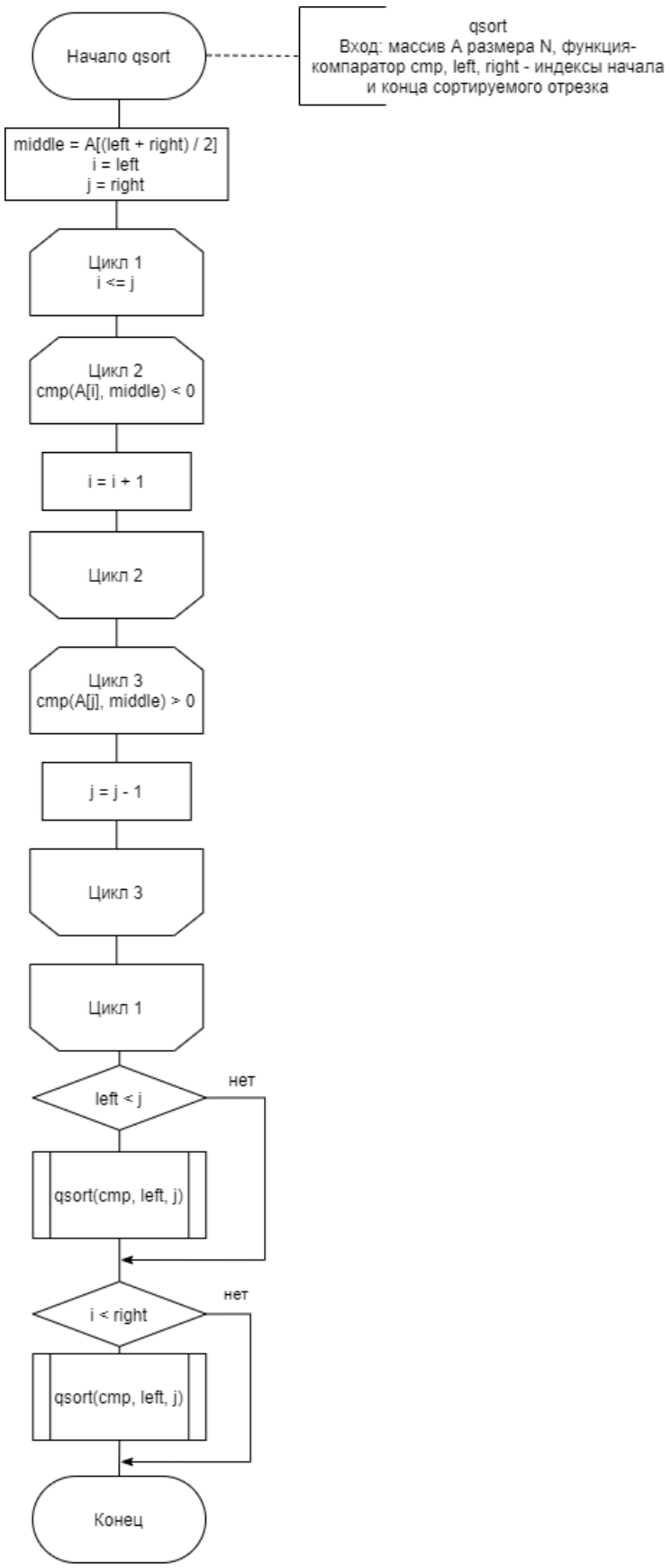

Рис. 2.3: Схема быстрой сортировки

Код	Вес
bool flag = false;	1
for (int i = 1; i < Length; i++)	3
{	0
flag = false;	1
for (int j = 0; j < Length - i; j++)	4
{	0
if(arr[i] < arr[i + 1])	4
{	0
tmp = arr[i];	2
arr[i] = arr[i + 1];	3
arr[i + 1] = tmp;	2
flag = true;	1
}	0
}	0
if (flag == false) break;	1
}	0

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $1 + 2 + 1 * (1 + 1 + 3 + (n - 1) * (4 + 1) + 1) = 5n + 4 = O(n)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен, все внутренние циклы будут состоять из N - j итераций.
Трудоемкость: $1+2+(n-1)*(1+3+(n-2)*(1+4+2+3+3+1)+1)/2 = 7n^2-18.5n+14.5 = O(n^2)$

2.2.2 Сортировка выбором

Построчная оценка трудоемкости сортировки выбором (Табл. 2.2).

Табл. 2.2 Построчная оценка трудоемкости для алгоритма сортировки выбором

Код	Вес
int min = 0;	1
for (int i = 0; i < Length - 1; i++)	4
{	0
min = i;	1
for (int j = i + 1; j < arr.Length; j++)	4
{	0
if(arr[j] < arr[min])	3
{	0
min = j;	1
}	0
}	0
if (min != i)	1
{	0
tmp = arr[i];	2
arr[i] = arr[min];	2
arr[min] = tmp;	2
}	0
}	0

Лучший случай: отсортированный массив, обмены не производятся.
Трудоемкость: $1 + 3 + (n-1)(1 + 3 + (n-2)(1 + 3)) = 4n^2 - 8n + 8 = O(n^2)$

Худший случай: массив отсортирован в обратном порядке, все обмены будут произведены.
Трудоемкость: $1 + 3 + (n-1)(1 + 3 + (n-2)(1 + 3 + 1 + 2 + 2 + 2)) = 11n^2 - 29n + 22 = O(n^2)$

2.2.3 Быстрая сортировка

Лучший случай: сбалансированное дерево вызовов $O(n * log(n))$. В сбалансированном варианте при каждой операции разделения массив делится на две одинаковые части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит log2n. В результате количество сравнений, совершаемых быстрой сортировкой, равно значению рекурсивного выражения $C_n = 2 * C_{n/2} + n$, что дает общую сложность O(nlogn) [1].

Худший случай: несбалансированное дерево $O(n^2)$. В несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и $n - 1$, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. В этом случае потребуется $\sum_{i=0}^{n}(n-i) = O(n^2)$ операций, то есть сортировка будет выполняться за квадратичное время [1].

2.3 Вывод

На основе описания алгоритмов, данного в аналитическом разделе, были построены схемы трёх алгоритмов сортировки, оценены их трудёмкости в лучшем и худшем случаях. Таким образом, были получены следующие трудоёмкости:

- сортировка пузырьком: лучший - $O(n)$, худший - $O(n^2)$;

- сортировка выбором: лучший - $O(n^2)$, худший - $O(n^2)$;
- быстрая сортировка: лучший - $O(nlogn)$, худший - $O(n^2)$.

3 Технологическая часть

В данном разделе приводятся реализации алгоритмов, схемы которых были разработаны в конструкторской части. Кроме того, обосновывается выбор технологического стека и проводится тестирование реализованных алгоритмов.

3.1 Средства реализации

В качестве языка программирования был выбран C#, а среду разработки – Visual Studio, т. к. я знаком с данным языком и имею представление о тестировании программ в данном языке. Время работы алгоритмов было замерено с помощью библиотеки System.Diagnostics, класса Stopwatch, который имеет методы для расчёта процессорного времени [2].

3.2 Входные и выходные данные

На вход ПО получает массив сравнимых элементов. На выходе – тот же массив, но отсортированный в заданном порядке.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов сортировки.

Листинг 3.1: Функция быстрой сортировки

```
1  static int _Partition(int[] arr, int start, int end)
2  {
3    int p = arr[(end - start) / 2 + start];
4    int i = start - 1;
5    int j = end + 1;
6
7    while (true)
8    {
9      do i++; while (arr[i] < p);
10     do j--; while (arr[j] > p);
11
12     if (i >= j) return j;
13
14     Swap(ref arr[i], ref arr[j]);
15   }
```

```
}

static void _QuickSort(int[] arr, int start, int end)
{
  int i;
  if (start < end)
  {
    i = _Partition(arr, start, end);

    _QuickSort(arr, start, i);
    _QuickSort(arr, i + 1, end);
  }
}

static void QuickSort(int[] array)
{
  _QuickSort(array, 0, array.Length - 1);
}
```

Листинг 3.2: Функция сортировки массива пузырьком с флагом

```
static void BubleFlagSort(int[] arr)
{
  bool flag;

  for (int i = 1; i < arr.Length; i++)
  {
    flag = false;

    for (int j = 0; j < arr.Length - i; j++)
    {
      if (arr[j] > arr[j + 1])
      {
        Swap(ref arr[j], ref arr[j + 1]);
        flag = true;
      }
    }

    if (!flag)
    break;
  }
}
```

Листинг 3.3: Функция сортировки массива выбором

```
static void SelectionSort(int[] arr)
{
  int min;

  for (int i = 0; i < arr.Length - 1; i++)
  {
```

```
7       min = i;
8       for (int j = i + 1; j < arr.Length; j++)
9         if (arr[j] < arr[min])
10          min = j;
11
12      if (min != i)
13        Swap(ref arr[i], ref arr[min]);
14    }
15  }
```

3.4 Тестирование

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Таблица 3.1: Тестирование реализованных алгоритмов

Входной массив	Результат	Ожидаемый результат
$[-4, -4, -9, 9, 2]$	$[-9, -4, -4, 2, 9]$	$[-9, -4, -4, 2, 9]$
$[-5, 6, -1, 5, 8]$	$[-5, -1, 5, 6, 8]$	$[-5, -1, 5, 6, 8]$
$[7]$	$[7]$	$[7]$
$[]$	$[]$	$[]$
$[1, 0, -1]$	$[-1, 0, 1]$	$[-1, 0, 1]$

3.5 Вывод

В данном разделе были выбраны средства разработки и с помощью них реализованы три алгоритма сортировки: пузырьком, выбором и быстрая сортировка. Кроме того, было проведено тестирование реализованных алгоритмов.

4 Исследовательская часть

В данном разделе проводится сравнительный анализ реализованных алгоритмов по процессорному времени.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Windows 10 Home 64-bit;
- Оперативная память: 8 GB;
- Процессор: 3.0 GHz 6-ядерный процессор AMD Ryzen 5 4600h.

4.2 Время выполнения реализаций алгоритмов

Процессорное время выполнения алгоритма замерялось с помощью класса Stopwatch[2]: для более точного замера каждая сортировка на каждой длине массива выполнялась несколько раз, а затем время усреднялось. На рисунках 4.1, 4.2 и 4.3 показаны зависимости процессорного времени выполнения сортировок от размера массива. Размеры массива варьировались от 100 до 1000. Время приведено в микросекундах. В таблицах 4.1, 4.2 и 4.3 показано время выполнения алгоритмов в зависимости от размеров массива.

Таблица 4.1: Таблица времени выполнения сортировок на отсортированных данных (в мкс)

Размер	bsort	ssort	qsort
100	1.07	37.53	6.10
200	2.02	149.26	13.07
300	2.78	346.79	19.63
400	3.41	603.28	26.39
500	4.25	900.45	44.86
600	5.07	1279.26	41.92
700	5.92	1812.68	55.55
800	6.77	2395.87	63.59
900	7.97	2965.90	81.32
1000	7.16	3553.72	84.51

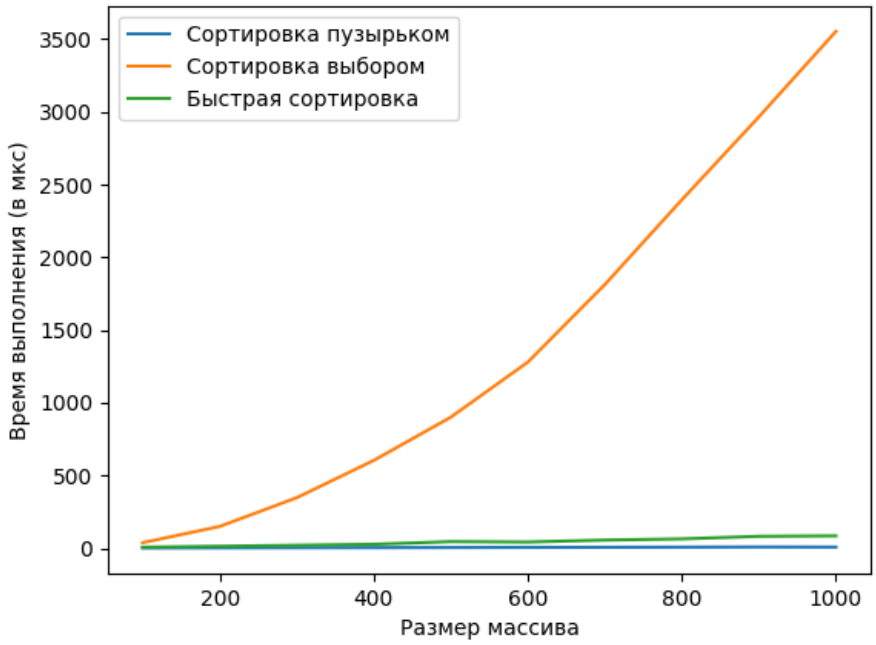

Рис. 4.1: Зависимость времени выполнения сортировок от размера массива на отсортированных данных

Таблица 4.2: Таблица времени выполнения сортировок на отсортированных в обратном порядке данных (в мкс)

Размер	bsort	ssort	qsort
100	94.97	40.37	6.40
200	430.85	138.77	13.17
300	756.08	342.28	20.99
400	1566.99	613.07	34.24
500	2425.09	918.80	45.05
600	3417.56	1242.70	49.46
700	4512.78	1742.92	51.72
800	5886.49	2448.24	68.06
900	7473.29	2983.23	85.06
1000	9070.99	3681.87	89.79

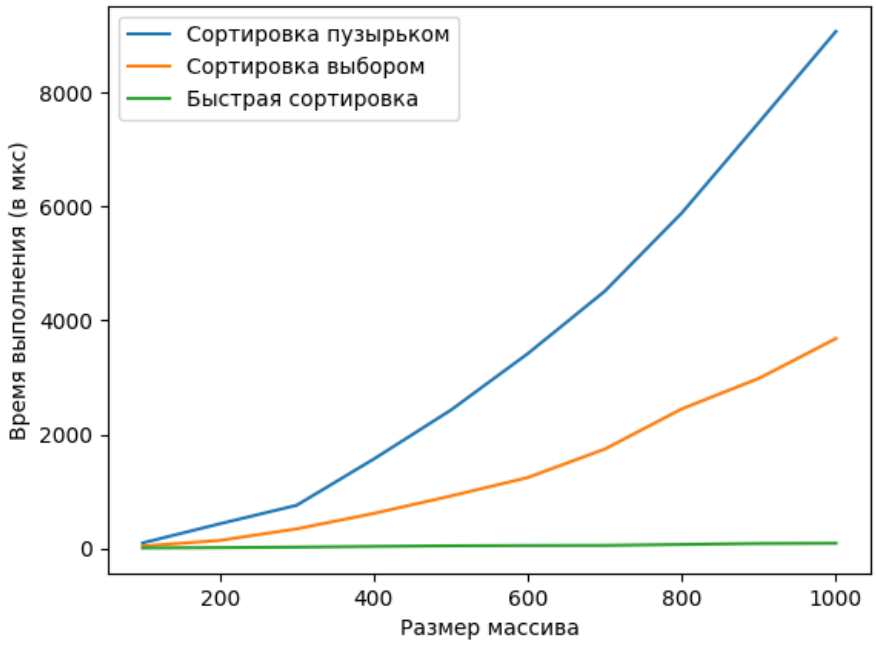

Рис. 4.2: Зависимость времени выполнения сортировок от размера массива на отсортированных в обратном порядке данных

Таблица 4.3: Таблица времени выполнения сортировок на случайных данных (в мкс)

Размер	bsort	ssort	qsort
100	78.83	44.63	7.61
200	291.00	184.57	17.25
300	665.56	365.30	27.60
400	1160.13	610.15	57.50
500	1819.05	910.41	76.51
600	2630.66	1284.73	84.09
700	3494.18	1806.15	132.21
800	4617.10	2393.96	172.21
900	5990.72	3002.44	180.78
1000	7337.07	3626.99	227.67

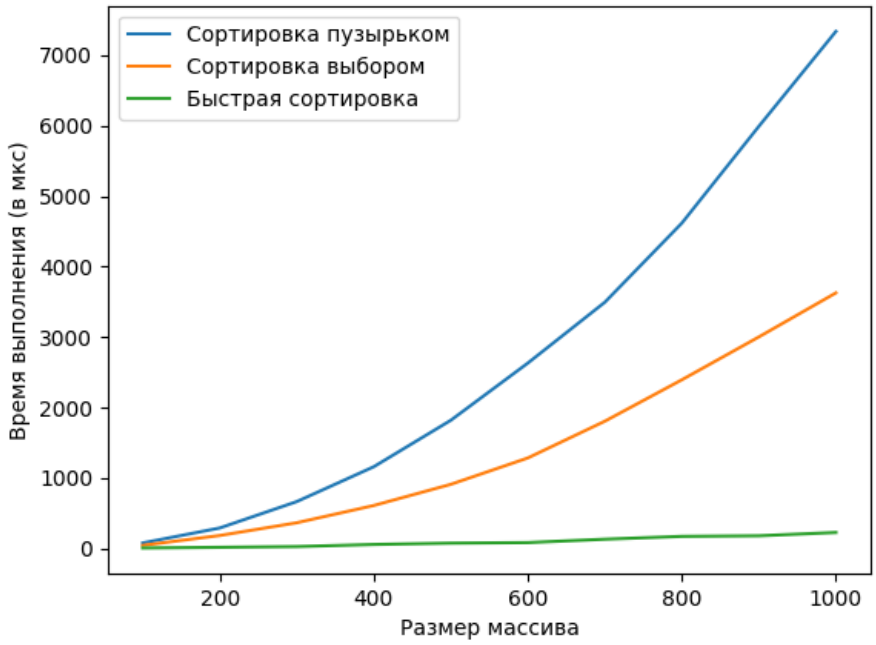

Рис. 4.3: Зависимость времени выполнения сортировок от размера массива на произвольных данных

4.3 Вывод

Как и ожидалось в результате оценки трудоемкости алгоритмов, сортировка пузырьком работает очень быстро на уже отсортированном массиве (лучший случай - оценка O(n)) и очень медленно на отсортированном в обратном порядке (худший случай - оценка O(n^2)). Время сортировки выбором на всех трёх видах массивов примерно одинаково, поскольку лучший и худший случаи работают за квадратичное время. Быстрая сортировка работает значительно лучше двух других сортировок во всех трёх приведённых случаях.

Заключение

В рамках данной лабораторной работы были решены следующие задачи:

- изучены 3 алгоритма сортировки: пузырьком, выбором, быстрая сортировка
- реализованы 3 алгоритма сортировки: пузырьком, выбором, быстрая сортировка;
- протестированы реализованные алгоритмы;
- проведён сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов;
- проведён сравнительный анализ алгоритмов по затраченному процессорному времени.

Цель работы достигнута: получены навыки сравнительного анализа на примере 3 алгоритмов сортировки: сортировки пузырьком, сортировки выбором и быстрой сортировки.

На основании анализа трудоемкости алгоритмов было показано, что алгоритм сортировки пузырьком имеет наименьшую сложность (линейную) в уже отсортированном массиве и квадратичную в отсортированном обратно, сортировка выбором имеет квадратичную сложность в лучшем, худшем и среднем случаях, а быстрая сортировка работает в лучшем и среднем случае значительно лучше (O(nlogn)) двух других алгоритмов, однако при неудачно выбранном опорном элементе и она может достигать квадратичной сложности. Те же выводы были подверждены экспериментально на основе замеров процессорного времени работы алгоритмов. Для сортировки рекомендуется быстрая сортировка.

Литература

1. Кормен Т. Алгоритмы: построение и анализ [Текст] / Кормен Т. - Вильямс, 2014. - 198 с. - 219 с.

2. Свойство Process.UserProcessorTime [Электронный ресурс]. Режим доступа: https://docs.microsoft.c ru/dotnet/api/system.diagnostics.stopwatch?view=net-5.0. Дата обращения: 02.10.2021