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)

ОТЧЕТ

По лабораторной работе №3

По курсу: «Анализ алгоритмов»

Тема: «Алгоритмы сортировки»

Студент: Керимов А. Ш.

Группа: ИУ7-54Б

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

Москва, 2019

Оглавление

Введение **2**

1 Аналитическая часть **4**

1.1 Модель вычислений 4

1.2 Описание алгоритмов 4

1.2.1 Сортировка пузырьком 4

1.2.2 Сортировка вставками 5

1.2.3 Сортировка выбором 5

2 Конструкторская часть **6**

2.1 Разработка алгоритмов 6

2.2 Трудоёмкость алгоритмов 7

3 Технологическая часть **9**

3.1 Средства реализации 9

3.2 Листинг кода 9

3.3 Тестирование функций 11

4 Исследовательская часть **12**

4.1 Технические характеристики 12

4.2 Время выполнения алгоритмов 12

Заключение **16**

Литература **17**

Введение

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Сортировки используются в самом широком спектре задач, включая обработку коммерческих, сейсмических, космических и прочих данных [1]. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными и т. п.

Сортировка применяется во всех без исключения областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

В учебнике [2, стр. 21] Д. Кнута упоминается что «по оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Исходя из этих статистических данных, можно заключить, что либо (i) сортировка имеет много важных применений, либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки».

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным.

К примеру, на сайте [3] можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

Задачи работы

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить сортировки пузырьком, вставками и выбором;
- реализовать каждую из этих сортировок;
- рассчитать их трудоемкость;

- сравнить их временные характеристики экспериментально;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. +, -, /, %, ==, !=, <, >, <=, >=, [], *, ++, -- – трудоемкость 1;
2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе;} \end{cases} \tag{1.1}$$

3. трудоемкость цикла рассчитывается, как $f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$;
4. трудоемкость вызова функции равна 0.

1.2 Описание алгоритмов

1.2.1 Сортировка пузырьком

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, но есть модифицированная версия, где если окажется, что обмены больше не нужны, значит проходы прекращаются. 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.

1.2.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [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 [4, стр. 38–45].

1.2.3 Сортировка выбором

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;

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: пузырьком, вставками и выбором. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки выбором, вставками и пузырьком соответственно.

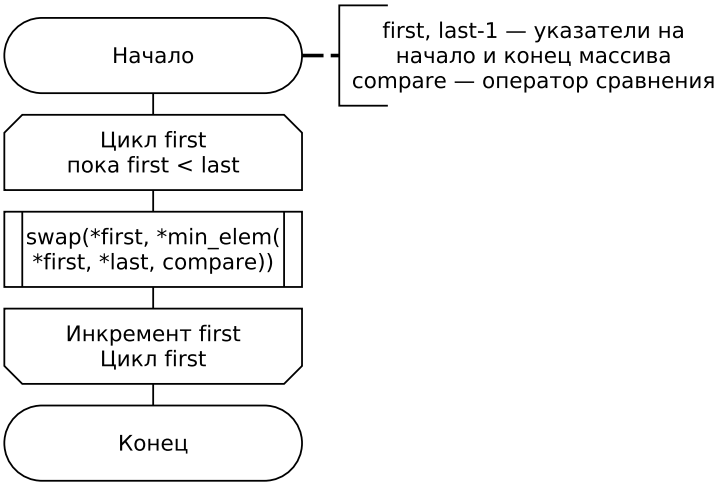

Рис. 2.1: Схема алгоритма сортировки выбором

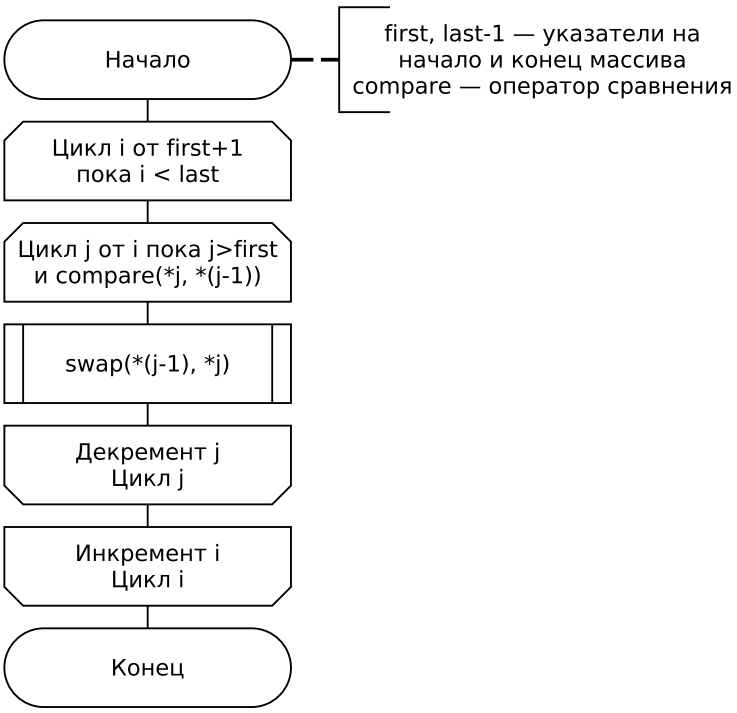

Рис. 2.2: Схема алгоритма сортировки вставками

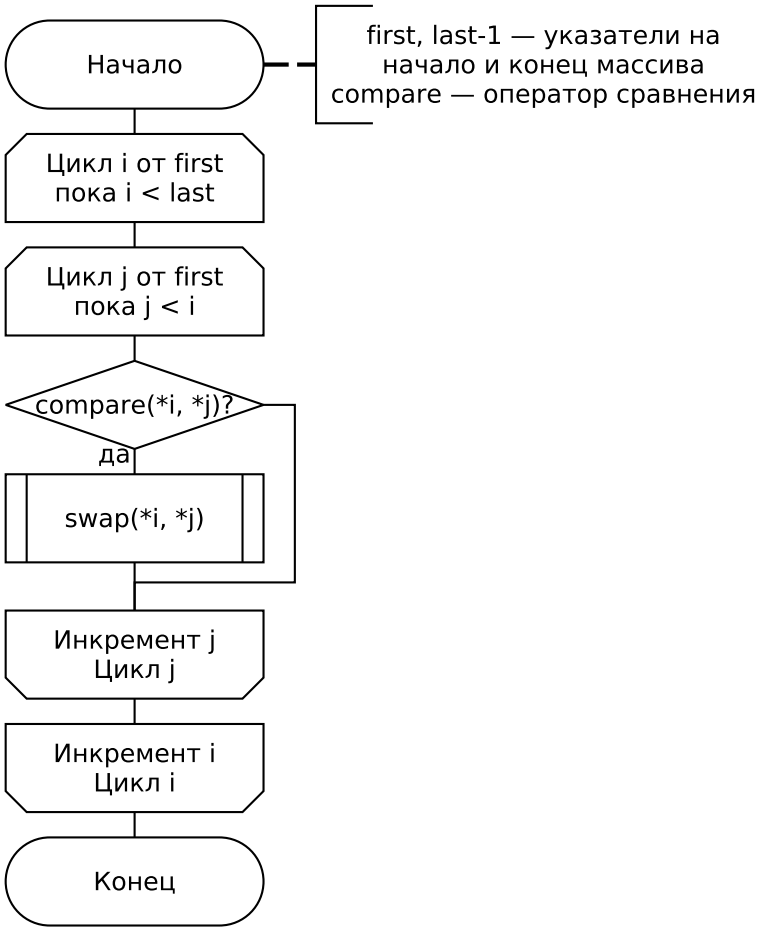

Рис. 2.3: Схема алгоритма сортировки пузырьком

2.2 Трудоёмкость алгоритмов

Пусть n — длина массива (расстояние между `first` и `last`) и оператор сравнения в точности есть оператор меньше. Лучший и худший случай для данных алгоритмов одинаковы и равны соответственно случаям отсортированного и обратно отсортированного массивов.

Подробно рассчитаем трудоёмкость алгоритма сортировки выбором. В цикле `first` делается n сравнений `first < last` и n инкрементов переменной `first`. В теле цикла `first` выполняется

$$\sum_{k=1}^{n} 2k = n(n+1) \tag{2.1}$$

разыменовываний указателя и n вызовов процедуры `swap`, которая выполняет 3 присваивания каждый раз.

Функция `min_elem` возвращает указатель на минимальный элемент диапазона, длина которого постепенно уменьшается с n до 1. Следовательно, суммарное количество сравнений в этой функции $n(n+1)$, а кол-во присваиваний в лучшем случае n (n раз минимальному значению присваивается первый элемент), в худшем —

$$\sum_{k=1}^{n} k + 1 = \frac{1}{2}n(n+3). \tag{2.2}$$

Кол-во разыменовываний указателя — $n(n+1)$.

Итого, трудоёмкость алгоритма сортировки выбором в лучшем случае:

$$n + n + n(n+1) + 3n + n(n+1) + n + n(n+1) = 3n^2 + 9n = O(n^2), \tag{2.3}$$

в худшем:

$$n+n+n(n+1)+3n+n(n+1)+\frac{1}{2}n(n+3)+n(n+1) = \frac{7n^2}{2}+\frac{17n}{2} = O(n^2). \tag{2.4}$$

Рассчитывая аналогично, трудоёмкость алгоритма сортировки пузырьком в лучшем и худшем случае — $O(n^2)$, сортировки вставками в лучшем случае — $O(n)$, в худшем — $O(n^2)$.

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были выделены и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран высокопроизводительный язык C++ [5], так как он предоставляет широкие возможности для эффективной реализации алгоритмов.

Для генерации псевдослучайных чисел использована функция из листинга 3.4.

Для замера времени работы алгоритма использованы точнейшие функции библиотеки `std::chrono`.

3.2 Листинг кода

В листингах 3.1, 3.2 и 3.3 приведены листинги алгоритма сортировки пузырьком, вставками и выбором соответственно. Функция замера времени работы алгоритма приведена в листинге 3.5.

```
template <typename Iterator, typename Compare = std::less<>>
void bubble_sort(Iterator first, Iterator last, Compare compare = Compare()) {
	for (auto i = first; i < last; ++i) {
		for (auto j = first; j < i; ++j) {
			if (compare(*i, *j)) {
				std::iter_swap(i, j);
			}
		}
	}
}
```

Листинг 3.1: Алгоритм сортировки пузырьком

```
template <typename Iterator, typename Compare = std::less<>>
void insertion_sort(Iterator first, Iterator last, Compare compare = Compare()) {
	for (auto i = first + 1; i < last; ++i) {
		for (auto j = i; j > first && compare(*j, *(j - 1)); --j) {
			std::iter_swap(j, j - 1);
```

```
            }
        }
}
```

Листинг 3.2: Алгоритм сортировки вставками

```
template <typename Iterator, typename Compare = std::less<>>
void selection_sort(Iterator first, Iterator last, Compare compare = Compare()) {
        for (; first < last; ++first) {
                std::swap(*std::min_element(first, last, compare), *first);
        }
}
```

Листинг 3.3: Алгоритм сортировки выбором

```
int dont_try_to_guess() {
        static thread_local std::mt19937 generator(std::random_device{}());
        static thread_local std::uniform_int_distribution<int> distribution(-1000, 1000);
        return distribution(generator);
}
```

Листинг 3.4: Продвинутый генератор псевдослучайных чисел

```
double count_time(sort_func_type<std::vector<int>::iterator> sort_func, const
    std::vector<int>& vector) {
        constexpr size_t N = 100;

        long long time = 0;
        for (int i = 0; i != N; ++i) {
                auto tmp = vector;

                const auto start = std::chrono::high_resolution_clock::now();
                sort_func(tmp.begin(), tmp.end(), std::less<>());
                const auto end = std::chrono::high_resolution_clock::now();

                time += std::chrono::duration_cast<std::chrono::nanoseconds>(end -
                    start).count();
        }

        return static_cast<double>(time) / 1.0e6 / N;
}
```

Листинг 3.5: Функция замера времени работы алгоритмов

3.3 Тестирование функций

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

Входной массив	Результат	Ожидаемый результат
[1,2,3,4]	[1,2,3,4]	[1,2,3,4]
[3,2,1]	[1,2,3]	[1,2,3]
[5,6,2,4, − 2]	[−2,2,4,5,6]	[−2,2,4,5,6]
[4]	[4]	[4]
[]	[]	[]

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

4.1 Технические характеристики

- Операционная система: Ubuntu 19.10 64-bit.
- Память: 3,8 GiB.
- Процессор: Intel® Core™ i3-6006U CPU @ 2.00GHz

4.2 Время выполнения алгоритмов

Алгоритмы тестировались с помощью функции замера процессорного времени `std::chrono::high_resolution_clock::now()`. Чтобы уменьшить случайные отклонения в измерениях, время считалось среднее для 5 запусков функций.

Результаты замеров приведены в таблицах 4.1, 4.2 и 4.3. На рисунках 4.1, 4.2 и 4.3 приведены графики зависимостей времени работы алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных.

Вывод

Алгоритм сортировки вставками работает лучше остальных двух на случайных числах и уже отсортированных, практический интерес, конечно, представляет лишь первый случай, на котором сортировка вставками почти на четверть быстрее сортировки пузырьком и на тринадцатую часть быстрее сортировки выбором.

Размер	Время сортировки, мс		
	Пузырьком	Вставками	Выбором
100	0.1736	0.0050	0.1908
200	0.6910	0.0102	0.7477
300	1.5523	0.0147	2.1193
400	2.7321	0.0199	2.9662
500	4.3130	0.0245	4.6916
600	6.2448	0.0294	6.7079
700	8.6443	0.0343	9.0672
800	11.1454	0.0391	11.9809
900	13.9159	0.0440	15.0407
1000	17.2003	0.0489	18.5356
1100	21.0066	0.0538	22.4010
1200	24.9522	0.0601	26.5965
1300	29.2257	0.0635	31.3609
1400	33.7588	0.0684	36.1941
1500	38.7270	0.0733	41.5358
1600	44.1570	0.0781	47.2253
1700	49.7883	0.0830	53.3138
1800	55.6949	0.0879	59.7656
1900	62.2464	0.0929	66.6882
2000	68.6274	0.0979	73.6054

Таблица 4.1: Время работы алгоритмов сортировки на отсортированных данных

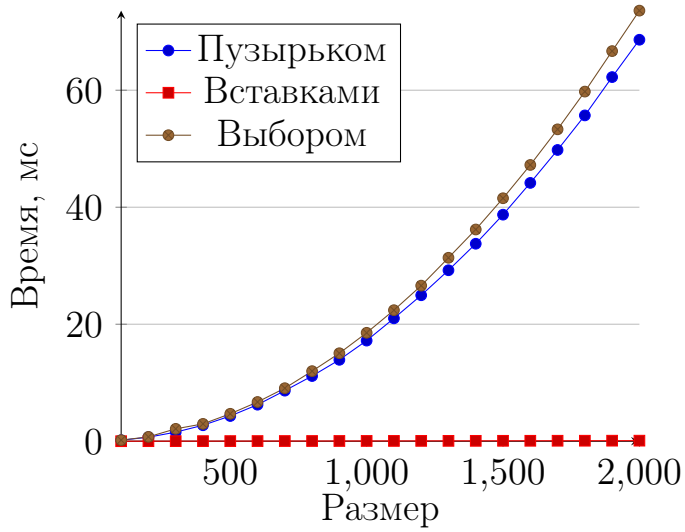

Рис. 4.1: Зависимость времени работы алгоритма сортировки от размера отсортированного массива

Размер	Время сортировки, мс		
	Пузырьком	**Вставками**	**Выбором**
100	0.2853	0.3480	0.1913
200	1.1316	1.3933	0.7523
300	2.5332	3.1253	1.6752
400	4.4974	5.5913	3.0934
500	7.1483	8.7589	4.6663
600	10.1921	12.6856	6.6764
700	13.8548	17.2417	9.0759
800	18.0085	22.5361	11.8481
900	22.9204	28.2434	14.9634
1000	28.3231	34.8271	18.4970
1100	34.1663	42.1950	22.3798
1200	40.8743	50.0964	26.6364
1300	47.6459	58.8373	31.2274
1400	55.2414	68.2595	36.1873
1500	63.5400	78.4059	41.6423
1600	72.3277	89.2412	47.3785
1700	81.4981	100.8099	53.4440
1800	91.3522	113.0405	59.8551
1900	101.9674	126.0761	66.7505

Таблица 4.2: Время работы алгоритмов сортировки на обратно отсортированных данных

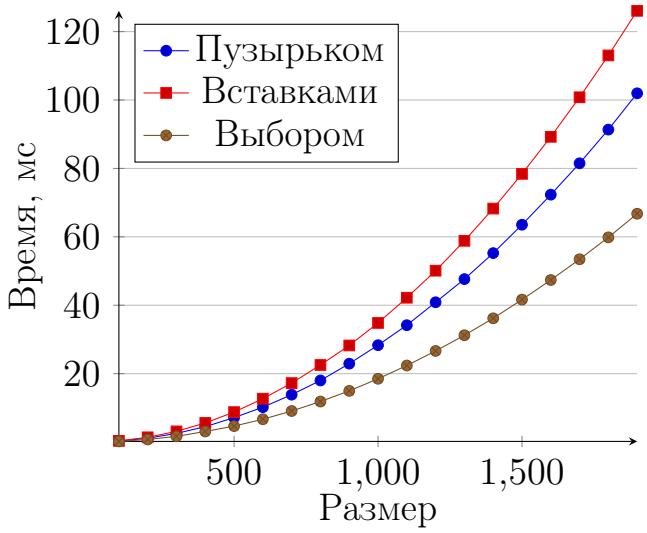

Рис. 4.2: Зависимость времени работы алгоритма сортировки от размера обратно отсортированного массива

Размер	Время сортировки, мс		
	Пузырьком	Вставками	Выбором
100	0.2341	0.1821	0.1963
200	0.9040	0.6844	0.7616
300	2.0532	1.6254	1.6954
400	3.6103	2.8699	3.0190
500	5.7835	4.4966	4.7507
600	8.0549	6.0996	6.8376
700	10.9769	8.2941	9.2248
800	14.3362	11.1200	11.8765
900	18.2116	14.5456	15.0222
1000	22.3888	16.9061	18.5311
1100	27.5257	21.3481	22.4303
1200	32.2910	25.2237	26.6552
1300	37.7618	29.2964	31.3004
1400	43.9901	34.9689	36.2359
1500	50.3503	39.5095	41.6308
1600	57.1336	44.9778	48.5256
1700	64.2833	49.7107	53.4662
1800	72.2257	56.4045	59.8987
1900	80.6056	61.4675	66.9760
2000	88.4554	67.5460	73.9807

Таблица 4.3: Время работы алгоритмов сортировки на случайных данных

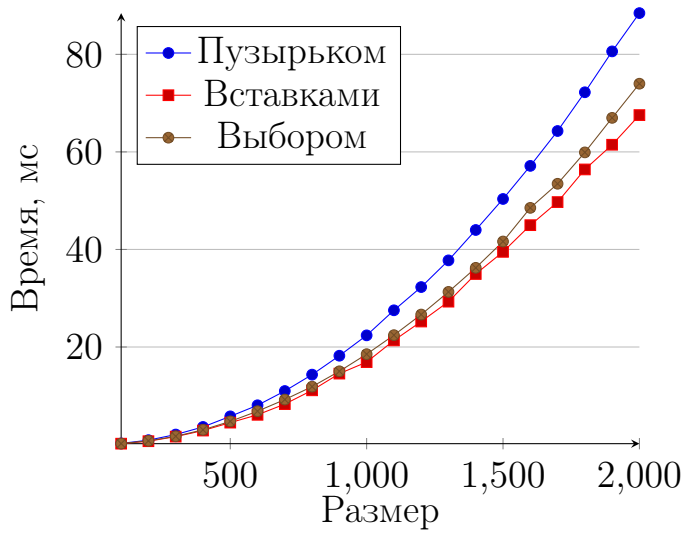

Рис. 4.3: Зависимость времени работы алгоритма сортировки от размера случайного массива

Заключение

В рамках лабораторной работы были реализованы и изучены сортировки пузырьком, вставками и выбором. Была оценена их трудоемкость, произведены замеры времени работы реализованных алгоритмов и сравнена их эффективность по времени. Наиболее эффективной оказалась сортировка вставками, выигрывая по времени на случайных данных пузырьковую сортировку на 23,6%, выбором — на 7,6%. Так же было отмечено, что сортировка выбором работает всегда за примерно одинаковое время, поэтому в худшем случае обгоняет остальные два алгоритма.

Литература

[1] Сортировка и фильтрация таблицы в Excel с помощью средства чтения с экрана. URL: `http://shorturl.at/zCLZ9` (дата обращения: 25.11.2019).

[2] Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования*. с. 834.

[3] Sort Benchmark. URL: `http://sortbenchmark.org/` (дата обращения: 25.11.2019).

[4] Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К. Алгоритмы: построение и анализ. Вильямс, 2013.

[5] Working Draft, Standard for Programming Language C++. URL: `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf`. 2017.