Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Генерация случайных чисел. Сравнение сортировок.

Выполнил:

студент группы 3821Б1ФИ3

Палатин И.А

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2021г.

# Содержание

[Содержание 1](#_Toc88648576)

[Введение 2](#_Toc88648577)

[Постановка задачи 3](#_Toc88648578)

[Руководство пользователя 4](#_Toc88648579)

[Первая программа 4](#_Toc88648580)

[Вторая программа 6](#_Toc88648581)

[Руководство программиста 9](#_Toc88648582)

[*Описание структуры кода программ* 9](#_Toc88648583)

[Первая программа 9](#_Toc88648584)

[Вторая программа 12](#_Toc88648585)

[Описание структур данных 19](#_Toc88648586)

[Описание алгоритмов 19](#_Toc88648587)

[Сортировка «пузырьком» 19](#_Toc88648588)

[Сортировка «вставками» 20](#_Toc88648589)

[Быстрая сортировка 21](#_Toc88648590)

[Эксперименты 26](#_Toc88648591)

[Заключение 28](#_Toc88648592)

[Список литературы 29](#_Toc88648593)

[Приложение 30](#_Toc88648594)

[Приложение 1.1 30](#_Toc88648595)

[Приложение 1.2 35](#_Toc88648596)

# Введение

Программирование — процесс создания компьютерных программ. По выражению одного из основателей языков программирования Никлауса Вирта, «Программы = алгоритмы + структуры данных». Программирование основывается на использовании языков программирования, на которых записываются исходные тексты программ.

Массивы один из самых распространённых типов данных, который позволяет удобно работать с большим количеством чисел одно типа. Изучение, построение и оптимизация алгоритмов работы с массивами — это задача, которая естественным образом встает перед программистом, работающим с массивами.

В настоящее время существует множество различных сортировок (т.е. алгоритмов сортировки), которые работают с разной эффективностью и применяются в различных ситуациях. В настоящей работе на языке программирования «С» будет реализовано три вида наиболее известных сортировок: пузырьковая (Bubble Sort), сортировка вставками (Insertion Sort) и быстрая сортировка (Quick Sort), а также будет произведено сравнение их быстродействия при одинаковых входных данных.

# Постановка задачи

Реализовать сортировки массивов данных (тип данных определяется преподавателем(long double)) задаваемых случайно

Реализовать сортировки. Сравнить время работы, сделать выводы.

Первая программа создает текстовый файл с записанными в него числами. Программа принимает количество чисел n, максимальное и минимальное значение.

Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия.

# Руководство пользователя

## Первая программа

После запуска программы необходимо выбрать нужный способ создания массива. Будет предложено три варианта генерации массива: 1) Генерация массива из случайных чисел. 2) Ввод массива с клавиатуры. 3) Копирование массива из файла. (см. рис. 1). Пользователю необходимо выбрать нужный способ, введя число.

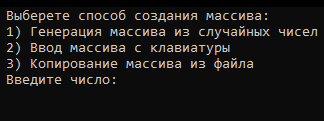


Рис. 1 Программа после запуска

Если пользователь напишет в консоли число 1, то ему необходимо будет ввести размер массива, а также его минимальное и максимальное значение. Затем будет создан текстовый документ, который будет заполнен случайными числами, с заданными параметрами (см. рис. 2).

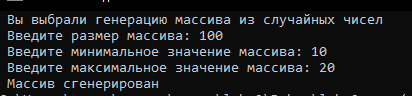


Рис. 2 Генерация массива из случайных чисел.

Если же пользователь напишет в консоли число 2, то после этого указатель в консоли переместиться на новую строку и пользователь сможет начать ввод чисел с клавиатуры. Числа передаются программе по одному, и как только пользователь введет все нужные числа, выведется строка: «Массив сгенерирован». Пример такой работы пользователя с интерфейсом программы приведен на рисунке 3.

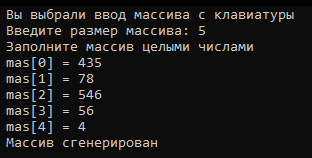


Рис. 3 Ввод массива с клавиатуры.

Если пользователь напишет в консоли число 3, то массив будет скопирован из одного файла и передан в другой - новый файл. (см. рис. 4).

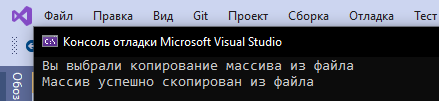


Рис. 4 Копирование массива из файла

При любом способе ввода результатом работы программы будет текстовый файл, который наполнен некоторым количеством чисел. Например, для набора из 10 случайных чисел, сгенерированных в диапазоне от -10 до 10, файл будет выглядеть как показано на рисунке 5.

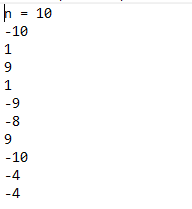


Рис. 5 Текстовый файл с сгенерированными числами

## Вторая программа

При запуске второй программы перед пользователем будет выведен интерфейс, (печать, сортировка, сброс, выход) в котором ему необходимо выбрать нужное действие (см. рис. 6).

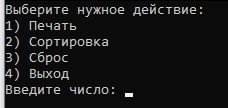


Рис.6 Интерфейс программы

Если пользователь напишет в консоли число 2, то программа предложит выбрать следующие сортировки: пузырьком, вставками и быстрая. Пользователю необходимо выбрать нужную сортировку (см. рис. 8).

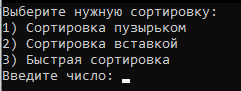
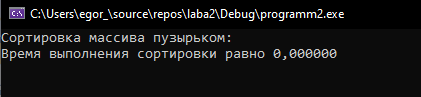


Рис. 8 Выбор сортировки

Например, пользователю нужно выбрать сортировку пузырьком. Тогда необходимо ввести число, соответствующее данной сортировке. Затем на экране будет выведено время данной сортировки (см. рис. 9)



# Руководство программиста

# *Описание структуры кода программ*

# *Первая программа*

1. Подключение библиотек
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <time.h>

2. Задание переменных

1. int main()
2. {
3. int i;
4. int maximum, minimum;
5. int size;
6. long double RAND\_MAX\_F = RAND\_MAX;

3. Диалог с пользователем

1. printf("Сколько элементов должен содержать массив ");
2. scanf("%d",&size);
3. printf("Укажите минимальное значение массива ");
4. scanf("%d", &minimum);
5. printf("Укажите максимальное значение массива ");
6. scanf("%d", &maximum);

4. Потенциальные ошибки пользователя

1. if (minimum > maximum) {
2. printf("Минимальное значение не может быть выше минимального");
3. return 0;
4. }
5. if (size <= 0 ) {
6. printf("Генерируемый массив не может быть пустым");
7. return 0;
8. }

5. Процесс создания файла

1. srand(time(NULL));
2. FILE\* file = 0;
3. file = fopen("/Users/prog/textFile.txt", "w");
4. for(i=0; i<size; i++){
5. fprintf(file, "%Lf\n", (rand() / RAND\_MAX\_F) \* ((double)maximum - (double)minimum) + maximum);
6. }
7. fclose(file);
8. return 0;
9. }

### Вторая программа

1. Подключение библиотек, с которыми предстоит работать.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <conio.h>

#include<locale.h>

#include<iso646.h>

#include<math.h>

clrscr()

{

system("@cls||clear");

}

2. Быстрый метод сортировки

1. void fastSort(long double\* array3, int start, int end)
2. {
3. printf("Пожалуйста, подождите. Идет процесс сортировки\n");
4. int left = start;
5. int right = end;
6. long double middle = array3[(left + right) / 2];
7. while (left <= right)
8. {
9. while (array3[left] < middle)
10. {
11. left++;
12. }
13. while (array3[right] > middle)
14. {
15. right--;
16. }
17. if (left <= right)
18. {
19. double buff = array3[left];
20. array3[left] = array3[right];
21. array3[right] = buff;
22. left++;
23. right--;
24. }
25. }

3. Пузырьковая сортировка

1. void bSort(long double\* array1, int size)
2. {
3. printf("Пожалуйста, подождите. Идет процесс сортировки\n");
4. int a, b;
5. long double c;
6. for (a = 0; a < size; a++)
7. {
8. for (b = 0; b < size - (a + 1); b++)
9. {
10. if (array1[b] > array1[b + 1])
11. {
12. c = array1[b];
13. array1[b] = array1[b + 1];
14. array1[b + 1] = c;
15. }
16. }
17. }
18. }
19. if (start < right)
20. fastSort(arr3, start, right);
21. if (end > left)
22. fastSort(arr3, left, end);
23. }

4. Сортировка вставками

1. void sortInsertion(long double\* array2, int size)
2. {
3. printf("Пожалуйста, подождите. Идет процесс сортировки\n");
4. long double b;
5. int f, k;
6. for (f = 1; f < size; f++)
7. {
8. for (k = i; k > 0; and array2[k - 1] > array2[k]; k--)
9. {
10. b = array2[k];
11. array2[k] = array2[k - 1];
12. array2[k - 1] = b;
13. }
14. }
15. }
16. const long double RAND\_MAX\_F = RAND\_MAX;
17. int stringCount(FILE\* strC) {
18. int result = 0;
19. while (!ferror(strC) and !feof(strC)) {
20. if (fgetc(strC) == '\n')
21. result++;
22. }
23. rewind(strC);
24. return result;
25. }

5. Алгоритм

1. int main()
2. {
3. int size, flag = 0, flag2 = 0, i;
4. double startTime, endTime, time\_on\_prog;
5. long double\* MASSIVE1, \* MASSIVE2;
6. FILE\* file = fopen("Users/Prog/TextFile.txt", "r");
7. setlocale(LC\_ALL, "Rus");
8. srand(time(NULL));
9. printf\_s("Читаем файл\n\n");
10. size = stringCount(file);
11. MASSIVE1 = (long double\*)malloc(size \* sizeof(long double));
12. MASSIVE2 = (long double\*)malloc(size \* sizeof(long double));
13. if (MASSIVE1 == 0 or MASSIVE2 == 0)
14. return 1;
15. for (int i = 0; i < size; i++)
16. fscanf\_s(file, "%lf", &MASSIVE1[i]);
17. memcpy(MASSIVE2, MASSIVE1, sizeof(long double) \* size);
18. do
19. {
20. printf("Выберете действие:\n1. Сортировка\n2. Распечатать числа\n3. Сброс\n4. Выйти\n\nВЫБОР: ");
21. do
22. {
23. scanf\_s("%d", &flag);
24. if (flag < 1 or flag > 4)
25. {
26. printf("Ошибка. Введите значение снова\n\nВЫБОР:");
27. }
28. } while (flag < 1 or flag > 4);
29. if (flag == 1)
30. {
31. clrscr();
32. printf("Выберите способ сортировки:\n1. Пузырьком\n2. Вставками\n3. Быстрая\n4. Вернуться в меню\n\nВыбор:");
33. do
34. {
35. scanf\_s("%d", &flag2);
36. if (flag2 < 1 or flag2 > 4)
37. {
38. printf("Неверный ввод. Введите снова\n Выбор:");
39. }
40. } while (flag2 < 1 or flag2 > 4);
41. if (flag2 == 1)
42. {
43. clrscr();
44. startTime = clock();
45. bSort(MASSIVE2, size);
46. endTime = clock();
47. time\_on\_prog = (double)(endTime - startTime) / 1000;
48. printf("Соритровка завершена\n");
49. printf\_s("Время выполнения %f\n\n", time\_on\_prog);
50. }
51. else if (flag2 == 2)
52. {
53. clrscr();
54. startTime = clock();
55. sortInsertion(MASSIVE2, size);
56. endTime = clock();
57. time\_on\_prog = (double)(endTime - startTime) / 1000;
58. printf("Сортировка завершена\n");
59. printf\_s("Время выполнения %f\n\n", time\_on\_prog);
60. }
61. else if (flag2 == 3)
62. {
63. clrscr();
64. startTime = clock();
65. fastSort(MASSIVE2, 0, size - 1);
66. endTime = clock();
67. time\_on\_prog = (double)(endTime - startTime) / 1000;
68. printf("Сортировка завершена\n");
69. printf\_s("Время выполнения %f\n\n", time\_on\_prog);
70. }
71. else
72. {
73. clrscr();
74. }
75. }
76. if (flag == 2)
77. {
78. for (i = 0; i < size; i++)
79. printf\_s("MASSIVE1[%d] = %lf\n", i, MASSIVE2[i]);
80. }
81. if (flag == 3)
82. {
83. clrscr();
84. memcpy(MASSIVE2, MASSIVE1, sizeof(long double) \* size);
85. printf\_s("Сортировка успешно сброшена.\n\n");
86. }
87. if (flag == 4)
88. {
89. printf("Программа завершена\n");
90. return 0;
91. }
92. } while (flag == 1 or flag == 2 or flag == 3 or flag == 4);
93. fclose(file);
94. free(MASSIVE1);
95. free(MASSIVE2);
96. }

## Описание структур данных

Одним из ключевых элементов второй программы является динамический массив, хранящий переменные типа int, в которой записываются числа, полученные из файла и с которым по ходу программы, происходит основная доля работы. Именно элементы массива подвергаются сортировке, а затем записываются в текстовый файл. Помимо этого, присутствуют различные счетчики типа int, без которых невозможна работа сортировок. Важнейшим элементом интерфейса является массив символов, в который записываются команды, вводимые пользователем во время работы программы.

При написании программы были использованы стандартные библиотеки языка «C»: stdlib.h, stdio.h, time.h.

## Описание алгоритмов

### Сортировка «пузырьком»

На вход алгоритму подается число элементов в сортируемом массиве, а также указатель на начало массива. На первой итерации внешнего цикла рассматривается нулевой элемент массива: он сравнивается с последующим элементом и, если оказывается больше него, то меняется с ним местами, а если оказывается меньше, то рассматривается уже следующий элемент. Эта процедура повторяется для каждого элемента массива, который в начале i-ой итерации внешнего цикла оказывается на месте j в массиве до тех пор, пока он будет отсортирован.

|  |
| --- |
| void bSort(long double\* array1, int size)  {      printf("Пожалуйста, подождите. Идет процесс сортировки\n");      int a, b;      long double c;      for (a = 0; a < size; a++)      {          for (b = 0; b < size - (a + 1); b++)          {              if (array1[b] > array1[b + 1])              {                  c = array1[b];                  array1[b] = array1[b + 1];                  array1[b + 1] = c;              }          }      }  }      if (start < right)          fastSort(arr3, start, right);      if (end > left)          fastSort(arr3, left, end);  } |

Сортировка Bubble Sort считается наиболее простой и в тоже время наиболее время затратной. Её эффективность оценивается как от размера входных данных, то есть количества элементов в исходном массиве. И в самом деле это можно легко заметить, ведь внешний цикл повторяется n раз и в каждом из таких циклов есть вложенный цикл, который повторяется n раз, отсюда можно сделать вывод, что всего совершается проверок.

### Сортировка «вставками»

На вход алгоритма подается число элементов в массиве и указатель в начало массива. Данный алгоритм помещает как бы разделяет исходный массив на отсортированную и неотсортированную часть, а затем по очереди помещает в отсортированную часть массива оставшиеся элементы массива на соответствующие им места уже в упорядоченном виде, то есть вставляет каждый новый элемент на свое место в уже отсортированном массиве по очереди сравнивая его с упорядоченной последовательностью до тех пор, пока он не займет нужное место.

|  |
| --- |
| void sortInsertion(long double\* array2, int size)  {      printf("Пожалуйста, подождите. Идет процесс сортировки\n");      long double b;      int f, k;      for (f = 1; f < size; f++)      {          for (k = i; k > 0; and array2[k - 1] > array2[k]; k--)          {              b = array2[k];              array2[k] = array2[k - 1];              array2[k - 1] = b;          }      }  } |

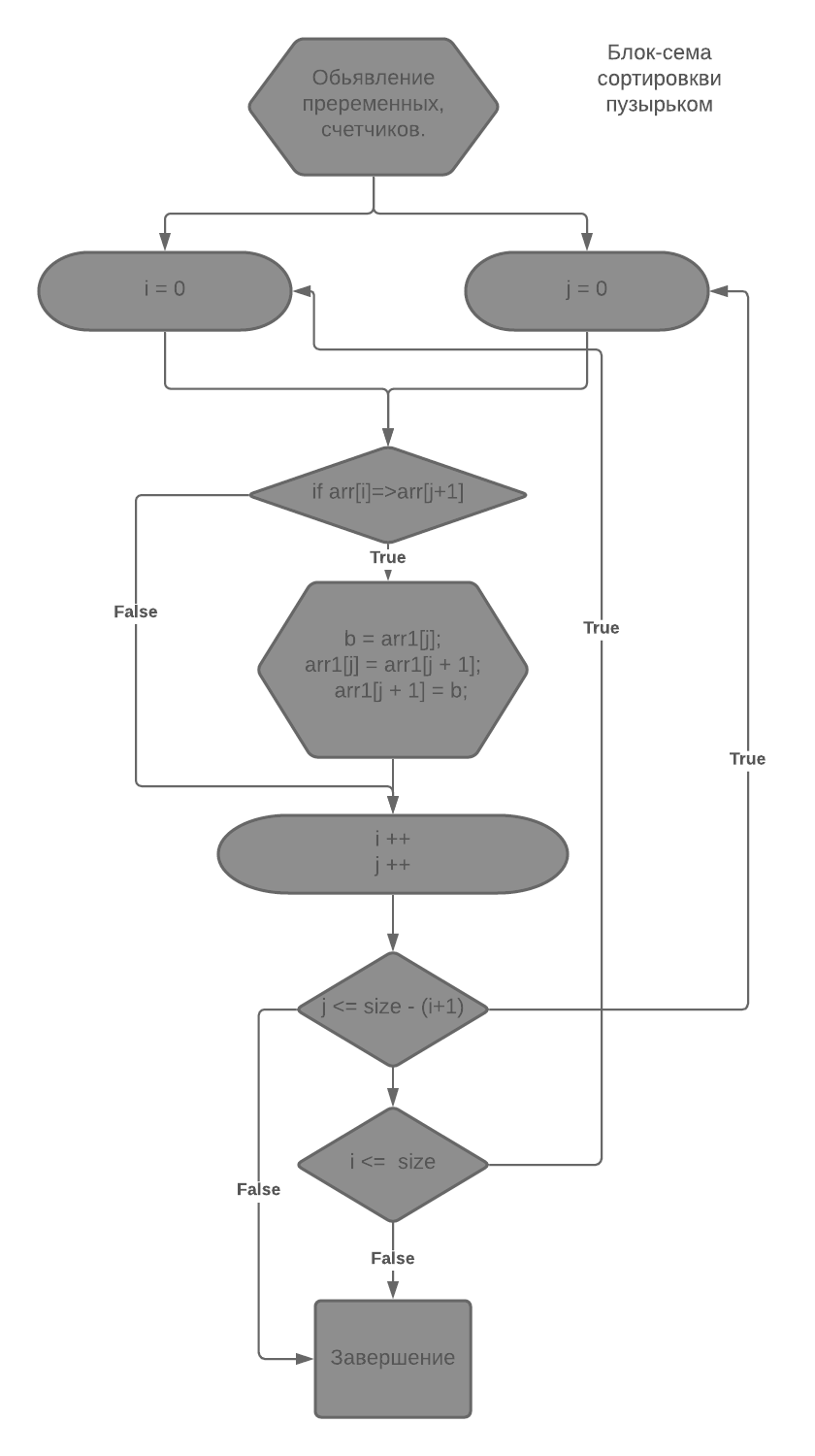
В худшем случае, как и в среднем, сортировка вставками затрачивает времени на свою работу. Данный алгоритм по совей сложности схож с сортировкой пузырьком, ведь в нем производятся схожие по структуре процессы: сравнение элементов и их сдвиги. Однако на практике такой алгоритм нередко оказывается быстрее сортировки пузырьком, так как в ней вне зависимости от того какой был исходный массив выполняется n проверок для каждого элемента с первого до( n – 1)-ого, в то время как сортировки вставкой выполняют проверки для новых элементов до тех пор, пока они не займут своего места и в изначально частично упорядоченном массиве могу работать значительно быстрее.

### Быстрая сортировка

Как и в других случаях, на вход этому алгоритму подается число элементов и указатель на начало массива. Однако в отличии от двух других сортировок он работает по совершенно другому принципу. Для начала выбирается средний по положению элемент входного массива – ключ. Затем просматриваются элементы массива до ключа и после него. Как только слева находится элемент больше ключа, и справа находится элемент меньше ключа, то они меняются местами. Это происходит до тек пор, пока все элементы слева от ключа не будут меньше него, а все справа не станут больше. Таким образом мы имеем два массива: массив с числами меньше ключа и массив с числами больше ключа. Затем происходит рекурсивный вызов сортировки для каждого из этих массивов. Этот процесс происходит до тех пор, пока все элементы массива не окажутся отсортированными, то есть пока один из массивов на некоторой итерации не станет одноэлементным.

|  |
| --- |
| void fastSort(long double\* array3, int start, int end)  {      printf("Пожалуйста, подождите. Идет процесс сортировки\n");      int left = start;      int right = end;      long double middle = array3[(left + right) / 2];      while (left <= right)      {          while (array3[left] < middle)          {              left++;          }          while (array3[right] > middle)          {              right--;          }          if (left <= right)          {              double buff = array3[left];              array3[left] = array3[right];              array3[right] = buff;              left++;              right--;          }      } |

Данный алгоритм принципиально отличается от двух других. Это касается и его сложности. В сравнении с квадратичными сортировками (пузырьковая, вставками), быстрая сортировка более чем оправдывает свое название, ведь её худшее время работы , в то время как среднее время работы составляет , что быстрее чем в случае с квадратичными сортировками.



# Эксперименты

В качестве эксперимента посмотрим на практике какой из алгоритмов сортировки работает быстрее с набором случайной сгенерированных чисел.

Для этого с помощью первой программы создадим текстовый файл с большим количеством чисел (см. рис. 10).

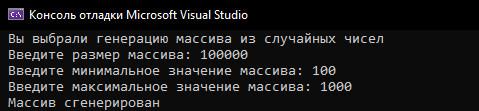


Рис. 12 Параметры генерации чисел

Затем запустим программу два и с её помощью отсортируем элементы в текстовом файле с помощью сортировки пузырьком (см. рис. 11)

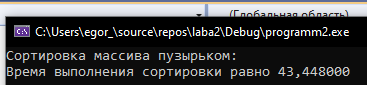


Рис. 11 Запуск и завершение сортировки

Сортировка пузырьком продлилась примерно 43 секунды. После её завершения возвращаем файл к первоначальному виду и сортируем его с помощью сортировки вставкой. Затем повторяем это процедуру для быстрой сортировки.

Полученные в экспериментах данные приведены в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип сортировки** | **Сложность** | **Размер входных данных** | **Время работы, с** |
| Пузырьком |  | 100000 | ≈43 |
| Вставками |  | 100000 | ≈10 |
| Быстрая |  | 100000 | ≈0,027 |
| Пузырьком |  | 5000 | ≈0,1 |
| Вставками |  | 5000 | ≈0,032 |
| Быстрая |  | 5000 |  |

Таблица 1. Сравнение времени работы сортировок

Все эти данные были получены на одной и той же машине, при одном и том же диапазоне генерации случайных чисел.

# Заключение

В ходе данной лабораторной работы, на языке программирования «С» были написаны две программы, позволяющие работать с текстовыми файлами: заполнять их числами и получать числа из них. Реализованы три алгоритма сортировки числовых массивов: сортировка пузырьком, вставками и быстрая, а также был проведен эксперимент по сравнению их быстродействия.

По результатам экспериментов можно сделать вывод о том, что самая быстрая сортировка, как при больших, так и при маленьких входных данных, это «быстрая» сортировка, что соответствует теоретическим данным. Далее по быстродействию идет сортировка «вставками», которая выполнилась вдвое быстрее пузырьковой при небольшом размере входных данных, и почти в пять раз быстрее при значительном количестве обрабатываемых данных. Ну и наиболее медленной оказалась сортировка «пузырьком», которая выполнялась дольше всех других сортировок при одинаковом характере и количестве входных данных.

# Список литературы

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ — 2-е. — М.: Вильямс, 2005. — 1296 с.
2. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / под ред. В. Т. Тертышного (гл. 5) и. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с.
3. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.
4. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.