Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчёт по заданию №1

«Методы сортировки»

Вариант 3 / 1 / 2 + 5

Выполнил:
студент 103 группы
Воронин И. И.

Преподаватель:
Малышев Н. Е.

Москва
2026

Содержание

1. Постановка задачи 3
2. Результаты экспериментов 4
   2.1. Теоретическая основа сортировки простым выбором 5
   2.2. Теоретическая основа пирамидальной сортировки 5
3. Структура программы и спецификация функций 7
   3.1. Константы 7
   3.2. Вспомогательные функции 7
   3.3. Методы сортировок 8
   3.4. Функция `main` 8
   3.5. Особенности описания функций. 8
4. Отладка программы, тестирование функций 9
   4.1. Error handler для упрощения отладки 9
   4.2. Тестирование функций 9
5. Анализ допущенных ошибок 10
   5.1. Неправильная работа HeapSort 10
   5.2. Использование Clang вместо GCC 10
6. Вывод 11
7. Список литературы 12

1. Постановка задачи

Требуется реализовать методы сортировки чисел, а также сравнить их с помощью экспериментов. Элементами списка являются действительные числа типа `double`, сортировка происходит по неубыванию.

В рамках выполнения этой работы будут рассмотрены 2 метода сортировки: простым выбором и пирамидальная(сортировка кучей). Требуется реализовать тестирующую функцию для данных алгоритмов. Тестирующая функция будет принимать на вход исходный массив, другой массив(отсортированный алгоритмом), а также длину исходного массива. Исходя из этих данных функция проверяет корректность работы конкретного алгоритма и выводит ответ. Также финальная программа должна выводить статистику по количеству перемещений и сравнений для конкретных тестируемых случаев в каждом из алгоритмов.

2. Результаты экспериментов

В данном разделе приведены результаты экспериментов по измерению количества сравнений и перемещений элементов для метода простого выбора и пирамидальной сортировки, а также их сравнение с теоретическими оценками.

	Номер сгенерированного массива				
n	**1**	**2**	**3**	**4**	**Среднее значение**
10	45	45	45	45	45
100	4950	4950	4950	4950	4950
1000	499500	499500	499500	499500	499500
10000	49995000	49995000	49995000	49995000	49995000

Таблица 1. Количество сравнений при сортировке методом простого выбора

	Номер сгенерированного массива				
n	**1**	**2**	**3**	**4**	**Среднее значение**
10	0	5	8	7	5
100	0	50	91	94	58,75
1000	0	500	991	990	620,25
10000	0	5000	9990	9993	6245,75

Таблица 2. Количество перемещений при сортировке методом простого выбора

	Номер сгенерированного массива				
n	**1**	**2**	**3**	**4**	**Среднее значение**
10	41	35	40	41	39,25
100	1081	944	1032	1050	1026,75
1000	17583	15965	16857	16871	16819
10000	244460	226682	235394	235462	235499,5

Таблица 3. Количество сравнений при сортировке пирамидальным методом

	Номер сгенерированного массива				
n	**1**	**2**	**3**	**4**	**Среднее значение**
10	30	21	28	29	27
100	640	516	587	600	585,75
1000	9708	8316	9083	9102	9052,25
10000	131956	116696	124276	124297	124306,25

Таблица 4. Количество перемещений при сортировке пирамидальным методом

2.1. Теоретическая основа сортировки простым выбором

Теоретическая оценка числа сравнений для метода простого выбора составляет $C(n) = \frac{n(n-1)}{2}$ для любого входного массива [1]. Это объясняется тем, что алгоритм всегда просматривает весь остаток массива в поисках минимума, независимо от расположения элементов. Экспериментальные данные полностью подтверждают эту оценку: при $n = 10$ получено ровно $45 = 10 \cdot \frac{9}{2}$ сравнений, при $n = 100$: 4950, при $n = 1000$: 499500, при $n = 10000$: 49995000. Таким образом, число сравнений метода простого выбора не зависит ни от начального порядка элементов, ни от конкретного набора данных.

Число перемещений метода простого выбора сильно зависит от входных данных. Для упорядоченного массива перемещения не производятся, потому что каждый минимум уже стоит на своём месте. Для массива, упорядоченного в обратном порядке, число обменов составляет $\frac{n}{2}$, что соответствует теоретической оценке [1]. Для случайных массивов число перемещений близко к n, что согласуется с оценкой $O(n)$ для среднего случая.

2.2. Теоретическая основа пирамидальной сортировки

Теоретическая оценка числа сравнений для пирамидальной сортировки составляет $O(n \log n)$ в наихудшем и среднем случаях [1]. Экспериментальные данные подтверждают логарифмический характер роста: при увеличении n в 10 раз число сравнений возрастает примерно в 11–13 раз, что соответствует множителю $\log_2 10 \approx 3,32$. Число перемещений также растёт как $O(n \log n)$: при $n = 10000$ среднее значение составило $124306,25$, что близко к $n \log_2 n \approx 132877,12$.

В отличие от метода простого выбора, пирамидальная сортировка имеет зависимость результатов от начального порядка элементов: для уже упорядоченного массива число сравнений и перемещений выше, чем для массива в обратном порядке. Это объясняется особенностями построения пирамиды и последующего извлечения элементов[1].

Таким образом, метод простого выбора имеет строго квадратичную сложность $O(n^2)$ по числу сравнений и линейную $O(n)$ по числу перемещений, тогда как пирамидальная сортировка обеспечивает сложность $O(n \log n)$ как по сравнениям, так и по перемещениям на любом входном массиве. При малых n разница между методами незначительна. При $n = 10000$ пирамидальная сортировка совершает гораздо меньше сравнений, что делает её более эффективной для работы с большими данными.

3. Структура программы и спецификация функций

Вся программа располагается в одном файле `sort.c`. Структурно в файле можно выделить 4 логических блока:

1. Глобальные переменные
2. Вспомогательные функции
3. Методы сортировки
4. Функция `main(void)`

3.1. Константы

Для облегчения разработки и отладки программы было принято выделить следующие глобальные переменные:

- `randD = 20000` - константа, отвечающая за диапазон, в котором будут генерировать числа типа `double` (соответственно от -10000 до 10000 в данном случае)
- `swaps` - переменная, отвечающая за подсчет количества перестановок в коде(учитываются только перестановки в случаях, когда `a[i] != a[j]`)
- `comps` - переменная, отвечающая за подсчет количества сравнений в методах сортировок
- `test` - переменная, принимающая значения от 1 до 4. Отвечает за номер теста, на котором проверяется метод сортировки. Вынесена в глобальные для облегчения отладки

3.2. Вспомогательные функции

1. `static void fillArr(double* a, int n)` - функция отвечает за заполнения массива определенными числами в зависимости от теста
2. `static int compare(double a, double b)` - одна из 2-х функций сравнения. Создана для облегчения подсчета количества сравнений.
3. `static int qComp(const void* a, const void* b)` - вторая функция сравнения. Создана для корректной работы встроенной в язык C быстрой сортировки. Используется для проверки корректности работы сортировок в файле `sort.c`
4. `static void swap(double* a, double* b)` - функция, отвечающая за подсчет количества перестановок в коде
5. `static void heapify(double* a, int n, int i)` - вспомогательная функция для пирамидальной сортировки. Строит для массива `a` «кучу» для максимума в массиве
6. `static int testW(double* a, double* sa, int n)` - функция для проверки корректности работы методов сортировки. Сначала проверяет на то, что массив `sa` отсортирован. Далее на то, что никакие элементы исходного массива `a` не пропали из `sa`.

7. `static void printRes(double *a, double *sa, int n, int st)` - функция созданная для «человеческого» вывода результатов тестирования методов сортировки

3.3. Методы сортировок

1. `static void selectionSort(double* a, int n)` - реализация сортировки выбором. Сложность алгоритма: $O(n^2)$ [1]
2. `static void heapSort(double *a, int n)` - реализация пирамидальной сортировки. Сложность алгоритма: $O(n \log(n))$ [1]

3.4. Функция `main`

1. Запуск генерации псевдорандомных чисел при помощи `srand(time(NULL))`
2. Проверка корректности введенного числа `n`, отвечающего за размер массивов, которые используется для тестирования функций сортировки
3. Выделение памяти для массивов, которые используются при работе методов сортировки
4. Цикл, прогоняющий конкретные тесты для методов сортировки и выводящий соответствующие результаты

3.5. Особенности описания функций.

Каждая функция имеет подробное описание, как внутренних компонентов, так и сигнатур. Это облегчает дальнейшее использование функций при переиспользование кода в других проектах. Также данное решение увеличивает читабельность кода в большинстве современных IDE. Пример см. Рис. 1.

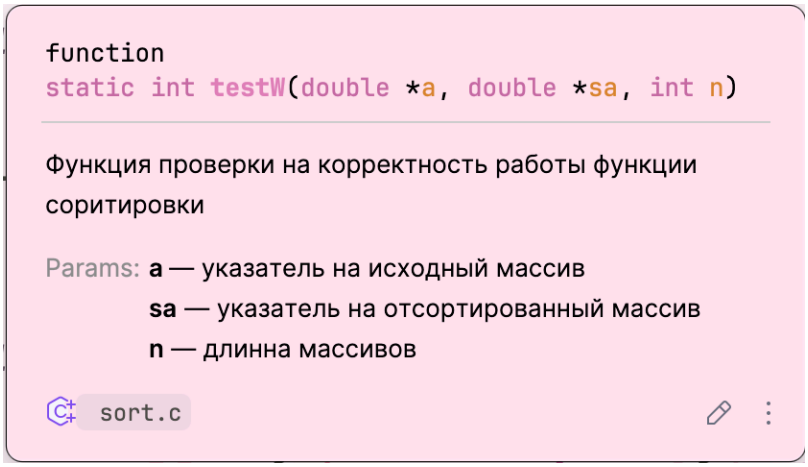


Рис. 1. Пример отображения описания функции в Clion 2025.2.1

4. Отладка программы, тестирование функций

4.1. Error handler для упрощения отладки

В файле `sort.c` кроме методов реализации и тестирующих функций присутствует небольшой error handler. При возникновение ошибок, программа экстренно завершается с определенными кодами ошибок. Ниже описаны реализованные коды ошибок и их описание:

- `Exit code 1` - вместо целого числа n программа получила на вход что-то другое
- `Exit code 2` - введенное пользователем число n оказалось меньше 1(массива не может существовать)
- `Exit code 3` - не удалось выделить память для основного массива или массивов для сортировок
- `Exit code 4` - ошибка выделения памяти для массивов во время проверки корректности работы алгоритмов

4.2. Тестирование функций

Функции тестирования алгоритмов помогли выявить критическую ошибку в работе пирамидальной сортировки

5. Анализ допущенных ошибок

5.1. Неправильная работа HeapSort

Функция `testW(double* a, double* sa, int n)` помогло выявить некорректность работы функции пирамидальной сортировки, вследствие чего функция `heapSort(double *a, int n)` была полностью переписана. Это помогло значительно сократить время на отладку, потому что было сразу понятно, где нужно было искать ошибку.

5.2. Использование Clang вместо GCC

Главной ошибкой при разработке проекта было использование компьютера на базе Apple Silicon. Изначальная отладка и разработка была произведена при помощи системы сборки Cmake и компилятора Clang. Использование данного стека обосновано разработкой на архитектуре Arm64 и MacOS 26.3. Ввиду особенностей архитектуры и принуждения операционной системы использовать LLVM и сопутствующие ему компоненты, в разработке и отладки программы было невозможно использовать некоторые флаги компиляции. Ниже приведены все использованные флаги:

```
-std=gnu99 -O2 -Wall -Werror -Wformat-security -Wignored-qualifiers -Winit-self -Wswitch-default -Wpointer-arith -Wtype-limits -Wempty-body -Wstrict-prototypes -Wold-style-definition -Wmissing-field-initializers -Wnested-externs -Wmissing-prototypes
```

Для последующего деплоя файла `sort.c` было проведено тестирование работоспособности программы в системе godbolt.org с заявленными в требованиях флагами компиляции.

6. Вывод

В ходе выполнения данной работы были реализованы и исследованы два метода сортировки: метод простого выбора и пирамидальная сортировка. Для каждого из алгоритмов была проведена серия экспериментов при различных размерах исходного массива($n = 10, 100, 1000, 10000$) на четырёх типах входных массивов, а результаты сопоставлены с теоретическими оценками. Экспериментальные данные полностью подтвердили теоретические оценки сложности: число сравнений метода простого выбора точно соответствует формуле $C(n) = \frac{n(n-1)}{2}$ и не зависит от входных данных, тогда как число перемещений изменяется от 0 до $\frac{n}{2}$. Пирамидальная сортировка демонстрирует рост числа сравнений и перемещений порядка $O(n \log n)$, что подтверждается экспериментальными значениями. При малых n оба метода показывают сопоставимую производительность. Однако при $n = 10000$ пирамидальная сортировка показывает большую производительность в сравнении с сортировкой простым выбором на больших объемах данных. Также была реализована тестирующая функция, которая позволила выявить и устранить ошибку в реализации пирамидальной сортировки, что подтвердило важность систематического тестирования алгоритмов.

7. Список литературы

[1] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Canada: ADDISON-WESLEY, 1998.