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

# Отчет по заданию №1

## «Методы сортировки»

## Вариант 3 / 1 / 2 / 3

Выполнил:
студент 101 группы
Боровко Н. А.

Преподаватель:
Дудина И. А.

Москва
2021

# Содержание

**Постановка задачи** **2**

**Результаты экспериментов** **3**
Теоретическая оценка . . . . . 3
Сравнение результатов . . . . . 4

**Структура программы и спецификация функций** **6**
Описание структуры теста . . . . . 6
Описание основных функций программы . . . . . 6
Описание второстепенных функций программы . . . . . 7

**Отладка программы, тестирование функций** **9**

**Анализ допущенных ошибок** **10**
Генерация случайного double . . . . . 10
Компаратор для qsort . . . . . 10

**Список цитируемой литературы** **11**

# Постановка задачи

Задачей является реализация сортировок методами **простого выбора** и **Шелла**, экспериментальное сравнение их эффективности друг с другом и с соответствующей теоретической оценкой. Ключевые моменты анализа:

- элементы сортируемого массива имеют тип *double*,
- сортировка выполняется по неубыванию с учетом знака,
- сортировка простого выбора используется в своем классическом виде,
- для сортировки Шелла используется последовательность[1] длин промежутков, предложенная Томасом Хиббардом, $k$-ый член которой вычисляется по формуле $2^k - 1$ (в обратном порядке).

# Результаты экспериментов

## Теоретическая оценка

1. *Сортировка простого выбора* - для этой сортировки можно привести точную оценку и числа сравнений, и числа присваиваний для массива длины $N$.

   (a) Сравнения. Нетрудно посчитать, что на $i$-ом шаге происходит $(i-1)$ сравнение, поэтому всего их $(N-1)+(N-2)+...+1 = \sum_{i=1}^{N-1} i = \frac{(N-1)+1}{2}(N-1) = \frac{1}{2}N(N-1) = \frac{1}{2}(N^2-N)$.

   (b) Обмены. На каждом проходе потребуется один обмен, чтобы нужный элемент встал на свое место, всего проходов $N-1$, так как последний элемент уже будет на нужном месте, поэтому итоговая оценка $N-1$.

2. *Сортировка Шелла* - для этой сортировки и выбранной нами последовательности существует[1] оценка математического ожидания числа сравнений и перенумераций.

   (a) Сравнения. Оценочная формула в данном случае имеет вид $kN^{5/4} + 0.15kN^{5/4}\log_2 N$, где $k \approx 1.07$.

   (b) Обмены. Формула оценки сравнений справедлива и для числа перенумераций, которое в условиях нашей задачи является удвоенным числом обменом, в связи с чем формула оценки обменов имеет вид $\frac{kN^{5/4}+0.15kN^{5/4}\log_2 N}{2}$, где $k \approx 1.07$.

## Сравнение результатов

| **N** | **Параметр** | **Номер массива** | | | | **Среднее значение** | **Теор. оценка** |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | | |
| 10 | Сравнения | 45 | 45 | 45 | 45 | 45 | 45 |
| | Обмены | 0 | 5 | 7 | 8 | 5 | 9 |
| 100 | Сравнения | 4950 | 4950 | 4950 | 4950 | 4950 | 4950 |
| | Обмены | 0 | 50 | 95 | 96 | 60 | 99 |
| 1000 | Сравнения | 499500 | 499500 | 499500 | 499500 | 499500 | 499500 |
| | Обмены | 0 | 500 | 991 | 993 | 621 | 999 |
| 10000 | Сравнения | 49995000 | 49995000 | 49995000 | 49995000 | 49995000 | 49995000 |
| | Обмены | 0 | 5000 | 9988 | 9988 | 6244 | 9999 |

Таблица 1: Результаты работы сортировки простого выбора

| **N** | **Параметр** | **Номер массива** | | | | **Среднее значение** | **Теор. оценка** |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | | |
| 10 | Сравнения | 19 | 25 | 27 | 28 | 25 | 28 |
| | Обмены | 0 | 11 | 9 | 15 | 9 | 14 |
| 100 | Сравнения | 480 | 614 | 786 | 796 | 669 | 675 |
| | Обмены | 0 | 192 | 355 | 374 | 230 | 337 |
| 1000 | Сравнения | 7987 | 10511 | 14102 | 14437 | 11759 | 15011 |
| | Обмены | 0 | 3424 | 6677 | 7001 | 4275 | 7505 |
| 10000 | Сравнения | 113631 | 144824 | 239455 | 235170 | 183270 | 320267 |
| | Обмены | 0 | 36778 | 131038 | 126766 | 73645 | 160133 |

Таблица 2: Результаты работы сортировки Шелла

На основании данных таблицы можно сказать следующее. Сортировка обменом несколько выигрывает у сортировки Шелла по числу обменов, т. к. оно у нее строго линейно, но по сравнениям проигрывает гораздо сильнее. Уже при 10000 элементах это число приближается к 50 млн против 200 тыс у сортировки Шелла, поэтому ее практическое применение не является рациональным. Сортировка Шелла показала себя лучше.

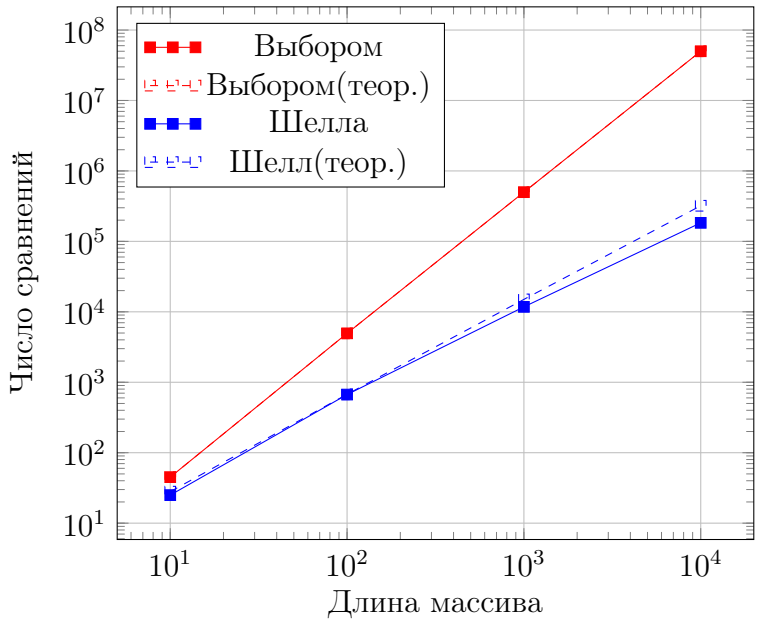


Рис. 1: Анализ зависимости числа сравнений от длины массива

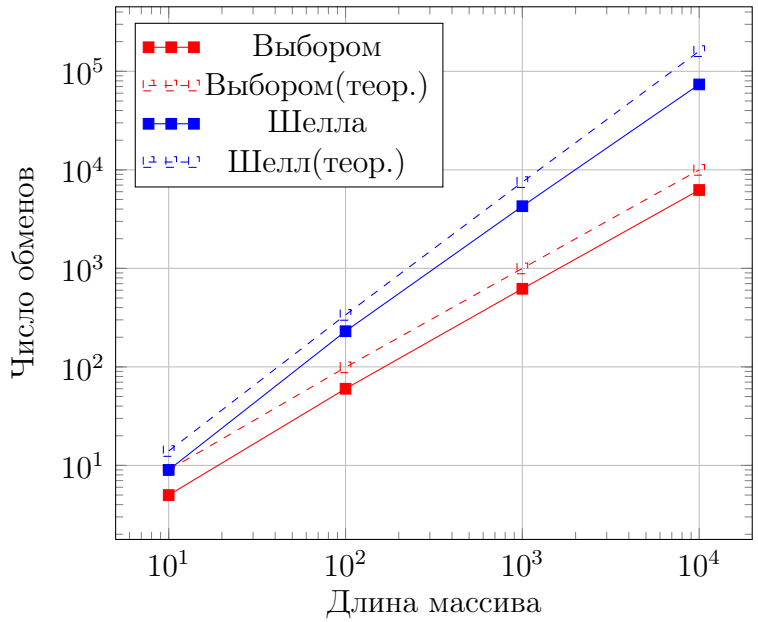


Рис. 2: Анализ зависимости числа обменов от длины массива

# Структура программы и спецификация функций

## Описание структуры теста

Программа содержит в себе струкруту test, в которой хранится вся информация для всех 3 сортировок на одном массиве. Структура содержит следующие поля:

1. `int size` - размер каждого массива.
2. `int type` - тип массива(1 - неубывающий, -1 - невозрастающий, иначе - полностью случайный).
3. `double *arr` - указатель на контрольный массив, к которому будет применен qsort.
4. `double *arr_selection` - указатель на массив, к которому будет применен sort_selection.
5. `double *arr_shell` - указатель на массив, к которому будет применен sort_shell.
6. `int selection_valid` - переменная, хранящая в себе 0/1 - корректность работы sort_selection.
7. `int shell_valid` - переменная, хранящая в себе 0/1 - корректность работы sort_shell.
8. `int swaps_shell` - переменная, хранящая в себе количество обменов в результате работы сортировки Шелла.
9. `int swaps_selection` - переменная, хранящая в себе количество обменов в результате работы сортировки выбором.
10. `int compares_shell` - переменная, хранящая в себе количество сравнений в результате работы сортировки Шелла.
11. `int compares_selection` - переменная, хранящая в себе количество сравнений в результате работы сортировки выбором.

## Описание основных функций программы

1. `void generate_test(test *a, int number)` - на вход принимаются указатель на тест и его номер, функция выделяет динамическую память для каждого указателя и записывает в каждый массив одинаковую последовательность чисел.
2. `void do_test(test *a)` - на вход принимается указатель на тест, функция выполняет сортировку выбором, сортировку Шелла и встроенную qsort, записывает в нужные переменные количества сравнений и обменов и выполняет проверку корректности, сверяя каждый массив с контрольным массивом, отсортированным с помощью qsort.

3. `void do_report(test *a, int number)` - на вход принимаются указатель на тест и его номер, функция печатает на стандартный поток вывода отчет о работе каждой сортировки на этом тесте(количество сравнений, обменов, корректность).

4. `void erase_test(test *a)` - на вход принимается указатель на тест, функция высвобождает всю использованную динамическую память.

5. `void sort_selection(double *arr, int size)` - на вход принимаются указатель на массив из double и его размер, программы выполняет сортировку выбором и подсчитывает число сравнений и обменов.

6. `void sort_shell(double *arr, int size)` - на вход принимаются указатель на массив из double и его размер, программы выполняет сортировку Шелла и подсчитывает число сравнений и обменов.

## Описание второстепенных функций программы

1. `double gen_rand_double(void)` - функция генерирует случайный double, используя union.

2. `double * gen_rand_array(int size, int type)` - на вход принимаются размер нужного массива и его тип(возрастающий, убывающий, случайный), функция генерирует массив из случайных элементов double в динамической памяти и возвращает указатель на него.

3. `int arrays_equal(double *a, double *b, int size)` - на вход принимаются 2 указателя на массивы и их размер, функция сравнивает их на идентичность и выводит соответственно 0 или 1. Нам требуется сортировка по неубыванию, поэтому такой метод гарантирует корректность выполнения сортировки.

4. `int compare(double a, double b)` - на вход принимаются 2 элемента типа double, функция сравнивает их и возвращает 0 или 1. Также производится увеличение глобальной переменной, отвечающей за количество сравнений для данного теста.

5. `void swap(double *a, double *b)` - на вход принимаются 2 указателя на double, функция производит обмен и увеличивает переменную, отвечающую за количество обменов для данного теста.

6. `int comp_up(const void *a, const void *b)` - стандартный компаратор для использования в встроенном qsort для сортировки массива из double по неубыванию.

7. `int comp_down(const void *a, const void *b)` - стандартный компаратор для использования в встроенном qsort для сортировки массива из double по невозрастанию.

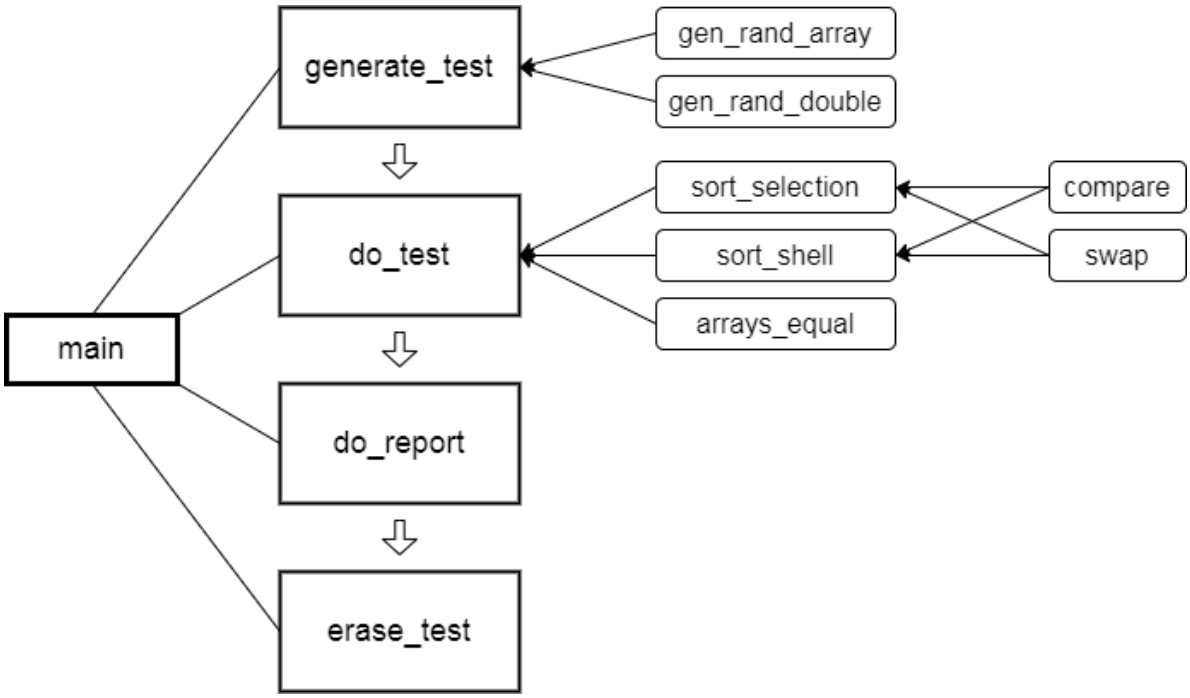


Рис. 3: Схема взаимодействия функций программы

# Отладка программы, тестирование функций

При генерации теста случайный массив создается в 3 одинаковых вариациях - для сортировки выбором (1), для сортировки Шелла (2) и для втроенного qsort (3). Впоследствии производится проверка на эквивалентность массивов (1) и (3), (2) и (3) при помощи функции arrays_equal:

```
int arrays_equal(double *a, double *b, int size)
{
    for (int i = 0; i < size; i++) {
        if (a[i] != b[i])
            return 0;
    }
    return 1;
}
```

Отладка функций сортировок производилась путем сравнения массивов, отведенных для соответствующей сортировки, с контрольным массивом, к которому был применен qsort при помощи функции, обозначенной выше. Изначально работа сортировок была протестирована на 10 разных массивах длины от 10 до 10000. После этого можно с уверенностью сказать, что сортировки работают правильно. Но проверка корректности оставлена и автоматически выполняется для каждого теста.

# Анализ допущенных ошибок

## Генерация случайного double

Первоначально я генерировал double при помощи перемножения 4-ех rand() для получения случайного long long и последующего преобразования этих 8 байт к 8 байтам double через взятие адреса, но на практике этот метод охватывает лишь часть возможных значений double, и также требуется убедиться, что double будет корректным числом, потому что часть значений в нем отведены под NaN и INF, поэтому было решено остановиться на реализации через union:

```
union {
    double a;
    unsigned char b[sizeof(double)];
} u;
u.a = NAN;
while (!isfinite(u.a)) {
    for (unsigned i = 0; i < sizeof(u.b); i++)
        u.b[i] = rand() % 256;
}
return u.a;
```

## Компаратор для qsort

Ошибка связана с тем, что компаратор для qsort обязан возвращать целое отрицательное, равное нулю или положительное число, если первый аргумент меньше, равен или больше второго. Если бы масссив был из элементов int, то можно было бы просто вернуть разность этих элементов, например так: `return ( *(int*)x1 - *(int*)x2 );`, но поскольку у нас элементы типа double, а прототип компаратора возвращает int, пришлось сравнивать числа в компараторе вручную:

```
if (*(double *)a > *(double *)b)
        return 1;
    if (*(double *)a < *(double *)b)
        return -1;
    return 0;
```

# Список литературы

[1] А. А. Паперпов, Г. В. Стасевич. Об одном методе упорядочивания информации в запоминающих устройствах цифровых машин. Проблемы передачи информации, том 1, выпуск 3, 1965, Москва. Статья.