

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
имени М. В. ЛОМОНОСОВА  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ОТЧЕТ ПО ЗАДАНИЮ №1

**«Методы сортировки»**

**Вариант 2 4 3 4**

Исполнитель: студент 106 группы

Магомедов Абдуррахман

Преподаватели:

Соловьев Михаил Александрович

Корухова Людмила Сергеевна

Манушин Дмитрий Валерьевич

МОСКВА

2020

## Оглавление

Постановка задачи.....	3
Результаты экспериментов.....	4
Графики.....	6
Структура программы и спецификации функций.....	8
Отладка программы, тестирование функций.....	10
Анализ допущенных ошибок.....	11
Литература.....	12

## Постановка задачи

Необходимо реализовать два метода сортировки чисел типа `long long int` по невозроанию модулей и провести их сравнение.

Первый метод сортировки – быстрая сортировка или quicksort, второй – сортировка Шелла (shellsort). Память будет выделяться динамически. При выполнении сортировки каждого вида будет подсчитываться число сравнений и перемещений элементов.

Рассматриваются длины массивов  $10$ ,  $10^2$ ,  $10^3$ ,  $10^4$ ,  $10^5$ ,  $10^6$ ,  $10^7$ ,  $10^8$ . Для каждой длины массива будет генерироваться четыре массива:

1. массив, в котором элементы упорядочены
2. массив, в котором элементы упорядочены в обратном порядке
- 3, 4. массивы со случайной расстановкой элементов

Результаты запуска программы будут записаны в таблицу.

Код доступен на <https://github.com/abrikos110/polygon/tree/master/sorts-homework>

## Результаты экспериментов

quicksort

swaps	comp.	data n		avg. swaps	avg. comp.
0	35	1	10	7,7	35,5
0	671	1	100	124,7	762,5
0	9976	1	1000	1701,0	12102,3
0	134219	1	10000	22523,7	154496,5
0	1668927	1	100000	276655,3	1927299,3
0	19951424	1	1000000	3257297,3	23815362,5
0	233759915	1	10000000	37737659,0	274242144,5
0	2666003370	1	100000000	429541838,0	3120492494,8
5	30	2	10		
50	670	2	100		
500	9972	2	1000		
5000	134214	2	10000		
50000	1668926	2	100000		
500000	19951420	2	1000000		
5000000	233759910	2	10000000		
50000000	2666003368	2	100000000		
10	42	3	10		
162	813	3	100		
2316	14130	3	1000		
31074	177464	3	10000		
390539	2179320	3	100000		
4632993	27961392	3	1000000		
53927058	317513594	3	10000000		
619542986	3551494913	3	100000000		
8	35	4	10		
162	896	4	100		
2287	14331	4	1000		
31497	172089	4	10000		
389427	2192024	4	100000		
4638899	27397214	4	1000000		
54285919	311935159	4	10000000		
619082528	3598468328	4	100000000		

shellsort

swaps	comp.	data	n	avg. swaps	avg. comp.
0	22	1	10	16,3	30,5
0	327	1	100	674,7	799,0
0	4550	1	1000	14738,3	15208,5
0	60156	1	10000	270035,7	260155,3
0	758531	1	100000	3363528,0	3241572,3
0	9310065	1	1000000	40079349,3	38920849,0
0	108851935	1	10000000	457750059,0	449340435,3
0	1255334358	1	100000000	5471859631,3	5314134574,8
25	38	2	10		
520	778	2	100		
7690	11390	2	1000		
80958	137708	2	10000		
984680	1662744	2	100000		
13933052	22320677	2	1000000		
132894650	237412758	2	10000000		
1587165582	2744265881	2	100000000		
16	35	3	10		
772	1065	3	100		
19128	23303	3	1000		
369106	425883	3	10000		
4418005	5137526	3	100000		
53323091	62196841	3	1000000		
620122606	725492690	3	10000000		
7277665982	8491922258	3	100000000		
8	27	4	10		
732	1026	4	100		
17397	21591	4	1000		
360043	416874	4	10000		
4687899	5407488	4	100000		
52981905	61855813	4	1000000		
620232921	725604358	4	10000000		
7550747330	8765015802	4	100000000		

В подсчёте средних значений количества обменов **не учитывались** результаты на отсортированном массиве, так как количество обменов равно 0. Серьёзных противоречий экспериментальных данных с теоретическими оценками сложности алгоритмов обнаружено не было

$n$  – размер массива,  $\text{comp.}$  – количество сравнений,  $\text{swaps}$  – количество обменов,  $\text{data}$  – тип заполнения массива (см. Постановка задачи),  $\text{avg. swaps}$  – среднее количество обменов на данных всех типов,  $\text{avg. comp.}$  – среднее количество сравнений

## Графики

Для просмотра графиков необходим цвет. Графики строились в логарифмическом масштабе с помощью `matplotlib`

Количество обменов ( $\text{swaps}$ ) `shellsort` на отсортированном массиве (`shellsort, sorted`) равно 0

Все четыре графика сравнений ( $\text{comparisons}$ ) `quicksort` очень близки друг к другу и накладываются друг на друга

Легенда:

- чёрный цвет – `shellsort, sorted`
- зелёный – `shellsort, sorted, reversed`
- синий – `shellsort, random`
- красный – `quicksort, sorted`
- жёлтый – `quicksort, sorted, reversed`
- magenta – `quicksort, random`



## Структура программы и спецификации функций

### **some\_funcs.h:**

- `long long int rand_ll(unsigned only_nonnegative);` // Возвращает

неотрицательное случайное число, если `only_nonnegative != 0`, иначе

возвращает случайное число во всём диапазоне допустимых значений типа `long long int`

- `void swap_vp(void *a, void *b, int sz);` // Меняет местами `sz` байт

по соответствующим указателям. Области памяти не должны пересекаться.

Используется глобальная `extern` переменная для подсчёта числа вызовов

- `long long int median3(long long int, long long int, long long`

`int);` // Возвращает медиану трёх чисел (использовалась в `quicksort`)

### **shellsort.h:**

- `void insertionsort(int n, long long int *a, int step, int (*less)(long`

`long int, long long int));` // Сортировка вставками по неубыванию на массиве `a`

длины `n` с шагом `step` (`a[0]`, `a[step]`, `a[2*step]`, ..., `a[(n-1)/step*step]`) и

функцией сравнения `less`, которая возвращает значение `!=0`, если первый

аргумент должен стоять раньше второго, иначе возвращает `0`

- `int step(int n, int i);` // `i`-ый (нумерация с `0`) шаг для сортировки

вставками массива длины `n` в сортировке Шелла. Здесь используется

последовательность  $f(j) = (2^j - 3)(2^{j+1} - 3)$ , которая обеспечивает асимптотику

$O(n^{4/3})$  в худшем случае [1]. Выбирается максимальное `j`, для которого  $f(j) < n$ ,

возвращается  $f(j-i)$

- `void shellsort(int n, long long int *a, int (*less)(long long int, long`

`long int), int (*step)(int, int));` // Параметры аналогичны `insertionsort`, только

есть дополнительный параметр указателя на функцию шага для сортировки

вставками

### **quicksort.h**



- `int partition(int n, long long int *a, int (*less)(long long int, long long int), long long int pivot);` // Функция разделяет массив на две части: если  $0 \leq i < p$ , то  $a[i] \leq pivot$ , если  $p \leq i < n$ , то  $a[i] \geq pivot$

- `void quicksort(int n, long long int *a, int (*less)(long long int, long long int));` // Параметры аналогичны `insertionsort`, сортирует массив с помощью `quicksort`. Средняя сложность –  $O(n \log n)$ , худшая –  $O(n^2)$  [3]

#### **main.c:**

- `long long int lliabs(long long int x);` // Возвращает абсолютное значение числа

- `int less(long long int a, long long int b);` // Возвращает `lliabs(a) > lliabs(b)` (для соответствия сортировки заданию). Используется глобальная переменная для подсчёта количества вызовов

- `void fill_array(int n, long long int *a, int filltype, int (*less)(long long int, long long int));` // Заполняет массив `a` длины `n` в соответствии с номером типа заполнения `filltype` (см. раздел Постановка задачи)

- `int main(void);` // Выделяет необходимую память, тестирует сортировки на массивах разной длины и выводит результат в `stdout`, освобождает память

## Отладка программы, тестирование функций

Функция `rand_llt` тестировалась построением графиков распределения, ошибки определялись на глаз. Проверялся знак возвращаемого значения `rand_llt(1)`

Функции `swar_vp`, `median3` отдельно не проверялись. В комментариях `median3` написано, что происходит в коде

Функции `insertionsort`, `step` отдельно не проверялись, функция `shellsort` проверялась на правильность сортировки

Функции `partition`, `quicksort` проверялись на соответствие требованиям, были найдены ошибки (см. раздел Анализ допущенных ошибок)

Функции `lliabs`, `less`, `fill_array` отдельно не проверялись

Функция `main` проверялась вручную с просмотром выводимых данных, были найдены ошибки

## Анализ допущенных ошибок

Функция `partition` много раз переписывалась, так как она иногда не обеспечивала правильное разделение массива, что приводило к неправильной сортировке

В функции `main` сначала тестировались `shellsort` и `quicksort` на разных массивах. После исправления этой ошибки появилась другая: `shellsort` запускался после `quicksort` на том же массиве, что приводило к тестированию `shellsort` на отсортированном массиве (ошибка была найдена просмотром данных — количество перемещений было равно 0). Сейчас эти ошибки исправлены

## Литература

1. Sedgewick, R. (1986). *A new upper bound for Shellsort*. *Journal of Algorithms*, 7(2), 159–173. doi:10.1016/0196-6774(86)90001-5
2. Incerpi, J., & Sedgewick, R. (1985). *Improved upper bounds on shellsort*. *Journal of Computer and System Sciences*, 31(2), 210–224. doi:10.1016/0022-0000(85)90042-x
3. <https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0>
4. <https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0>
5. <https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0>
6. <https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0>
7. Образец отчёта по заданию 1 (лежит в папке <https://github.com/abrikos110/polygon/tree/master/sorts-homework/docs> )