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

ОТЧЕТ ПО ЗАДАНИЮ №1

«Методы сортировки»

Вариант 2 3 1 4

Исполнитель: студент 106 группы
Шурыгин Всеволод Евгеньевич
Преподаватель:
Манушин Дмитрий Валерьевич

МОСКВА
2021

Содержание

Постановка задачи

Требуется реализовать два метода сортировки массива чисел и провести их экспериментальное сравнение. Генерация исходных массивов для сортировки реализуется отдельной функцией, создающей в зависимости от заданного параметра и заданной длины конкретный массив, в котором:
 элементы уже упорядочены (параметр 1);
 элементы упорядочены в обратном порядке (параметр 2);
 расстановка элементов случайна (параметр 3, 4).

Данные (элементы массива) — 64-разрядные целые числа (long long int), память выделяется динамически функцией malloc.

Вид сортировки: числа упорядочиваются по неубыванию модулей, то есть при сравнении элементов не учитывается знак.

Исследуемые два вида сортировки:

- Метод «пузырька» (см. [1] 130-132; [2] 27-28; [3] 101-102)
- Быстрая сортировка Хоара, рекурсивная реализация (см. [3] 114-117)

Результаты экспериментов оформить на основе нескольких запусков программы в виде следующей сводной таблицы:

Название метода сортировки

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения					
	Перемещения					
100	Сравнения					
	Перемещения					
1000	Сравнения					
	Перемещения					
10000	Сравнения					
	Перемещения					

Результаты экспериментов

В результате экспериментов была подтверждена асимптотическая оценка алгоритмов: быстрой сортировки — O(N · logN), сортировки “пузырьком” — O(N^2).

Пузырьковая сортировка

n	Параметр	Номер сгенерированного массива				Среднее значение
		1	2	3	4	
10	Сравнения	9	45	45	41	35
	Перемещения	0	45	27	14	21,5
100	Сравнения	99	4950	4935	4797	3696
	Перемещения	0	4950	2298	2361	2403
1000	Сравнения	999	499500	499434	498905	374709,5
	Перемещения	0	499500	246947	250634	249271
10000	Сравнения	9999	49995000	49981797	49989224	37494005
	Перемещения	0	49995000	24836278	24986725	24954501

По асимптотике общее число операций (сравнения+перемещения) соответственно для N должно быть порядка N^2: 10^2, 10^4, 10^6, 10^8: 10, 100, 1000, 10000. Результаты порядка: 60 (100), 6000 (10000), 620000 (1000000), 62000000 (100000000) подтверждают эту оценку.

Быстрая сортировка Хоара

n	Параметр	Номер сгенерированного массива				Среднее значение
		1	2	3	4	
10	Сравнения	31	34	33	45	35,75
	Перемещения	0	5	7	8	5
100	Сравнения	606	610	936	812	741
	Перемещения	0	50	145	138	84
1000	Сравнения	9009	9016	13401	12424	10962,5
	Перемещения	0	500	2304	2332	1284
10000	Сравнения	125439	125452	187387	179207	154371,5
	Перемещения	0	5000	30483	30744	16557

Общее число операций (сравнения+перемещения) соответственно для N: 10, 100, 1000, 10000 должно быть порядка N · logN: 10*3.32, 100*6.64, 1000*9.96, 10000*13.28. Результаты порядка: 40 (33), 820 (664), 12000 (9960), 170000 (132800) подтверждают эту оценку. Быстрая сортировка асимптотически “лучше” сортировки “пузырьком”, но на уже отсортированном массиве делает операций больше (см. таблицу).

Структура программы и спецификации функций

Функции, реализованные для выполнения задания:

- **void swap (long long *a, long long *b)** – меняет местами элементы массива, на которые указывают a и b.
- **int compareIncrease (const void *x1, const void *x2)** – для функции qsort из стандартной библиотеки компаратор должен возвращать 0, если аргументы равны, значение меньше нуля – если первый аргумент должен предшествовать второму (для сортировки по возрастанию меньше), больше нуля в ином случае.
- **int compareDecrease (const void *x1, const void *x2)** – функция сравнения элементов массива для qsort по невозрастанию.
- **void arrayGeneration (long long *a, int n, int param) -** генерация массива с псевдослучайными числами, от значения параметра зависит их порядок следования (см. выше в постановке задачи), при значении параметра 1 или 2 - используем qsort и 2 функции, описанные выше.
- **int absCompare (long long x1, long long x2, unsigned test)** – компаратор для сравнения чисел по абсолютной величине (проверяет, является ли модуль первого числа больше модуля второго).
- **void bubbleSort (long long *a, int n)** – сортировка “пузырьком”.
- **void qSort (long long *a, int left, int right)** – быстрая сортировка Хоара с параметрами начала и конца сортируемого подмассива (для рекурсивной реализации).
- **void quickSort (long long *a, int n)** – функция, принимающая в соответствиями с постановкой задачи только число элементов и сам массив.
- **int checkArrayAbsAscending (long long *a, int n)** – проверка, является ли сортировка по возрастанию модулей корректной.

Отладка программы, тестирование функций

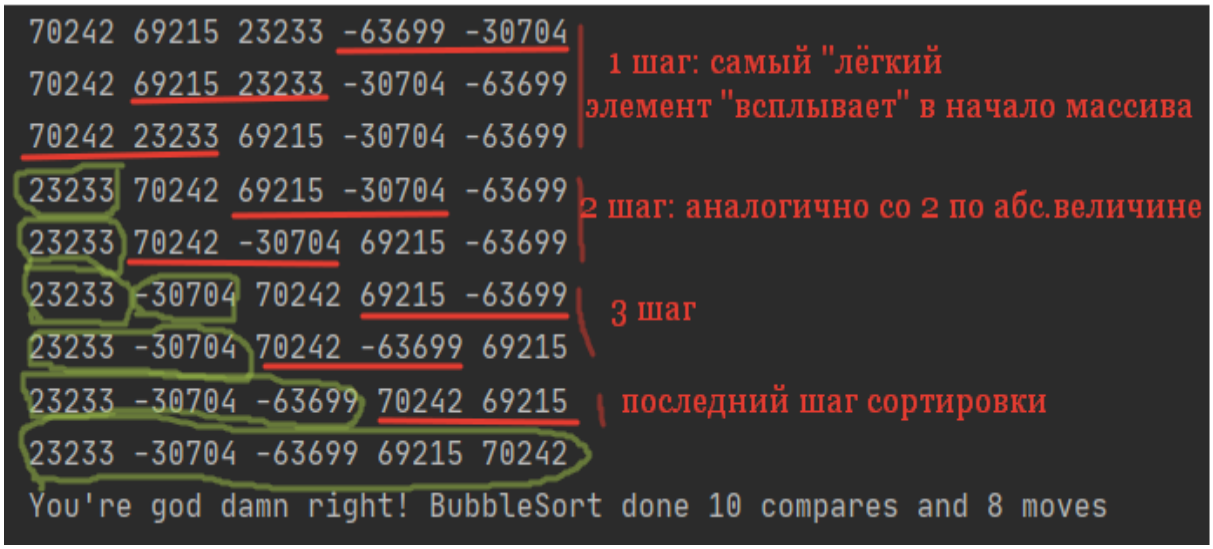
Правильность работы сортировок проверялась функцией **checkArrayAbsAscending (long long *a, int n)**, которой на вход подаётся потенциально отсортированный массив. Функция с помощью цикла проверяет, что все элементы отсортированы корректно – по возрастанию модулей.

Также в процессе доработки программы была использована функция qsort из стандартной библиотеки, принимающую в качестве компаратора функцию **absCompare**. В финальной же версии программы qsort оставлена в другом качестве - используется функцией **arrayGeneration** для проверки работы программы на случайном массиве чисел, отсортированных (не по абсолютной величине, а с учётом знака) по неубыванию или невозрастанию.

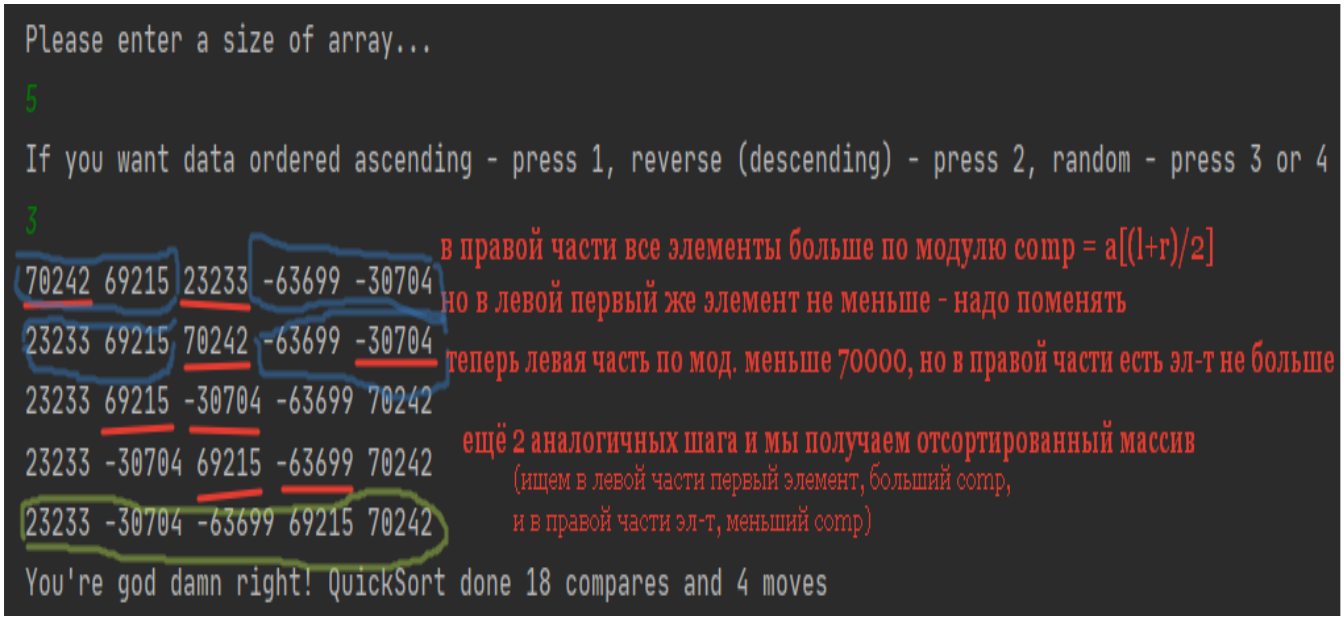
Массив сортируется. Но чтобы убедиться в корректности реализации конкретного метода, разберём шаги сортировок на примере небольшого массива (пусть n=5).

Сгенерированный массив: 70242 69215 23233 -63699 -30704

1) Сортировка “пузырьком”



2) Быстрая сортировка Хоара



Анализ допущенных ошибок

1) На начальной стадии несколько раз не компилировалось из-за синтаксических ошибок (например, писал ull вместо llu), выводило неверное число сравнений и перемещений (в printf выводил указатели).
2) Излишняя экономия память на типах данных, которая привела к проблемам, в частности с unsigned n, left, right, когда делал n-1 и когда уменьшалось j в цикле в быстрой сортировке, после нуля становилось max_unsigned, а не -1.
3) Не учёл, что abs(LLONG_MIN) вызовет переполнение - некорректно работало. Пришлось писать отдельную функцию AbsCompare. Также понадобилась дополнительная проверка x2==LLONG_MIN в AbsCompare.
4) В compare для библиотечной qsort добавил третий параметр для сортировки по неубыванию/невозрастанию, хотя по стандарту должно быть два.
5) Написал swap и compare inline функциями как в c++ по привычке - undefined reference.
6) Утечка памяти – добавил free(a).
7) Codestyle: неконсистентное именование фунций: часть в CamelCase, часть в lowerCamelCase, также не у всех функций по имени был понятен результат ее работы (compare1 и compare2 ничего не сообщают о том, как именно они сравнивают аргументы, CompareIncrease/CompareDecrease стали более понятными, из названия PsevdoRandom не ясно, что эта функция заполняет массив – заменено на arrayGeneration, TestFailed также можно назвать в соответствии с выполняемой работой, например CheckArrayAscending).
8) С генерацией 64-битных целых чисел по шаблону, предложенному в задании rand()*rand()*rand()*rand*rand() возникало сразу 2 проблемы: arrayGeneration читает элементы неинициализированного массива, когда в цикле из 5 итераций умножал на Rand(), а также Undefined Behavior (integer overflow) signed integer overflow: 2086435344 *1569882090 cannot be represented in type 'int' в rand()*rand(). Поэтому случайное число в массиве теперь формируется не по байтам, а по битам, отдельно аккуратно комментарий со старшим битом знака, чтобы в итоге охватывался весь диапазон от -2^63 до 2^63-1.
9) Была немного улучшена сортировка “пузырьком” - с досрочным выходом из цикла при отсутствии перемещений элементов, чтобы число сравнений было не заранее фиксированным (для отчёта).

Литература

1. Кнут Д. Искусство программирования для ЭВМ. Том 3. — М.: Мир, 1978.
2. Лорин Г. Сортировка и системы сортировки. — М.: Наука, 1983.
3. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.
4. Кормен Т., Лейзерсон Ч., Ривест Р, Штайн К. Алгоритмы: построение и анализ. Второе издание. М.”Вильямс», 2005.