Национальный исследовательский университет
«Высшая школа экономики»

Отчет по заданию №1

«Методы сортировки»

Вариант 2 / 1 / 2 + 4

Выполнил:
студент 213-1 группы
Ермишин Н. В.

Преподаватель:
Батузов К. А.

Москва
2022

Содержание

1. Постановка задачи **2**

2. Результаты экспериментов **3**

3. Структура программы и спецификация функций **5**

4. Отладка программы, тестирование функций **6**

5. Анализ допущенных ошибок **7**

Список литературы **8**

1. Постановка задачи

Суть задачи состоит в реализации двух методов сортировке массивов чисел и последующем сравнении их на количество обменов и сравнений на случайных массивах заданной длины и типа. В рамках распределения вариантов, необходимо реализовать сортировку методом *простого выбора* и рекурсивную реализацию *быстрой сортировки*. Данные реализации должны обрабатывать 64-разрядные целые числа (`long long int`). В процессе работы сортировок числа должны упорядочиваться по неубыванию. Для корректного подсчета количества сравнений и обменов необходимо реализовать отдельные функции. Также нужно реализовать функцию генерации массива и тестирования массива на отсортированность.

2. Результаты экспериментов

	Номер сгенерированного массива				Среднее
n	**1**	**2**	**3**	**4**	**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				Среднее
n	**1**	**2**	**3**	**4**	**значение**
10	9	9	9	9	9
100	99	99	99	99	99
1000	999	999	999	999	999
10000	9999	9999	9999	9999	9999

Таблица 2: Количество обменов при сортировке методом *простого выбора*

	Номер сгенерированного массива				Среднее
n	**1**	**2**	**3**	**4**	**значение**
10	31	34	36	42	35,75
100	606	610	917	853	746,5
1000	9009	9016	13715	13116	11214
10000	125439	125452	181011	184918	154205

Таблица 3: Количество сравнений при рекурсивной реализации *быстрой сортировки*

	Номер сгенерированного массива				Среднее
n	**1**	**2**	**3**	**4**	**значение**
10	6	11	11	11	9,75
100	63	112	175	182	133
1000	511	1010	2563	2610	1673,5
10000	5904	10904	33492	33258	20889,5

Таблица 4: Количество обменов при рекурсивной реализации *быстрой сортировки*

Сортировка методом *простого выбора* в любом случае делает $\mathbf{(n^2 - n)/2}$ сравнений (т.к. оно не зависит от порядка ключей)[1](стр 100-101) и $\mathbf{n - 1}$ обмен (т.к. обмен происходит на каждой итерации в любом случае - в рамках конкретной реализации). Таким образом, при асимптотическом приближении имеем $\mathbf{n^2}$ сравнений (см.Табл. 1) и $\mathbf{n}$ обменов (см.Табл. 2) для сортировки методом *прямого выбора*. Анализ подтвердил, что данные теоретического анализа сходятся с данными, полученными эмпирическим путём.

Сортировка *quicksort* в лучшем случае делает $\mathbf{n \cdot \log_2(n)}$ сравнений и делает $\mathbf{n \cdot \log_2(n)/6}$ обменов (в этом случае при каждом рекурсивном вызове выбирается медиана массива, всего требуется $\mathbf{\log_2(n)}$ рекурсивных вызовов), в среднем делает $\mathbf{2\ln(2) \cdot n \cdot \log_2(n)}$ сравнений и $\mathbf{\ln(2) \cdot n \cdot \log_2(n)/3}$ обменов (при случайном выборе граничного элемента получаем константу $\mathbf{2 \cdot \ln(2)}$), а в худшем случае делает $\mathbf{n \cdot (n - 1)}$ сравнений и $\mathbf{n \cdot (n - 1)/6}$ обменов (в этом случае массив длины $\mathbf{n}$ делится на массивы длин $\mathbf{1}$ и $\mathbf{n - 1}$). Таким образом, при асимптотическом приближении имеем $\mathbf{n \cdot \log_2(n)}$ (см.Табл. 3 и Табл. 4) сравнений и обменов в лучшем и среднем случае и $\mathbf{n^2}$ сравнений и обменов в худшем случае для *быстрой сортировки* [1](стр 118-120). Анализ подтвердил, что данные теоретического анализа сходятся с данными, полученными эмпирическим путём.

В результате анализа и сравнения можно понять, что сортировка *простым выбором* немного эффективнее на малых массивах ($\mathbf{n \sim 100}$), в то время как *быстрая сортировка* в среднем лучше работает на больших массивах ($\mathbf{n > 100}$)

3. Структура программы и спецификация функций

Список функций и описание функциональности:

1. `static int qsort_cmp_d(const void *a, const void *b)` - возвращает разность чисел `long long int`, расположенных по адресам `a` и `b`.

2. `static int qsort_cmp_r(const void *a, const void *b)` - возвращает разность чисел `long long int`, расположенных по адресам `a` и `b`, со знаком минус.

3. `static int _cmp(long long int a, long long int b)` - возвращает разность чисел `a` и `b`, увеличивает счетчик `cmp_counter` на 1.

4. `static void _swap(long long int *a, long long int *b)` - меняет местами значения по адресам `a` и `b`, увеличивает счетчик `swap_counter` на 1.

5. `static long long int *gen_arr(int param, int n)` - возвращает указатель на сгенерированный случайный массив длины `n`, который может являться отсортированным в прямом или обратном порядке в зависимости от `param`.

6. `static void selection_sort(int n, long long int *a)` - выполняет сортировку массива длины `n` по адресу `a` методом простого выбора.

7. `static void qs_aux(int l, int r, long long int *a)` - рекурсивная функция быстрой сортировки массива по адресу `a`. Выполняет сортировку подмассива `a[l:r]`.

8. `static void quick_sort(int n, long long int *a)` - обертка для рекурсивной функции быстрой сортировки `qs_aux`. Сортирует массив длины `n` по адресу `a`.

9. `static int is_sorted(long long int *a_initial, long long int a_sorted, int n)` - возвращает 1, если массив по адресу `a_sorted` является отсортированным массивом по адресу `a_initial`, 0 в противном случае. Оба массива длины `n`.

10. `static long long int *cp_arr(int n, long long int *a)` - возвращает указатель на копию массива `a` длины `n`.

11. `int main (void)` - основная функция программы. Через стандартный поток ввода принимает значение длины массива от пользователя, на основе которого проводит анализ. Возвращает 0 в случае успешного исполнения, в противном случае иное значение.

4. Отладка программы, тестирование функций

Изначально были спроектированы и написаны функция-генератор `gen_arr` и функция проверки отсортированности `is_sorted`, а также вспомогательные функции `qsort_cmp_d` и `qsort_cmp_r`. Функция генерации была проверена вызовами на произвольных значениях `n` с различными параметрами. Функция тестирования сортировки была проверена на группе массивов (в том числе был проверен факт того, что массивы обязательно должны быть поэлементно равными для положительного ответа).

После этого был изучен соответствующий материал по сортировкам. Сортировки были реализованы и оттестированы на группе произвольных массивов разных длин. Также на этом шаге были реализованы функции сравнения `_cmp` и обмена `_swap`.

Далее была написана основная функция программы - `main`, а также вспомогательная функция `cp_arr`. Тестирование `main` производилось поблочно, использовались инструменты отладки *IDE CLion*.

5. Анализ допущенных ошибок

В первой версии выгруженного кода была допущена ошибка - существовала неиспользуемая переменная `tmp`. Из-за этого автоматическая проверка *github classroom* выдавала ошибку. После удаление переменной коммит был успешно загружен, автоматическая проверка не выдала ошибок.

Благодаря этой ошибке была также найдена проблема, из-за которой при локальной компиляции *gcc* не использовал предоставленные флаги.

Список литературы

[1] Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.