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

Отчет по заданию №1

«Методы сортировки»

Вариант 3/4/2+4

Выполнил:
студент 103 группы
Шибаев П.П.

Преподаватель:
Кузьменкова Е.А.

Москва
2020

Содержание

Постановка задачи

Цель данной работы - произвести эмпирическое сравнение двух методов сортировки:

- сортировки простым выбором (simple selection sort)
- быстрой сортировки в рекурсивной реализации (recursive quicksort)

Сортировка производилась на массивах чисел типа double в порядке неубывания абсолютных значений чисел.

Код работы доступен по ссылке: github.com/shibaeff/SortsReport.

Результаты экспериментов

Были произведены запуски на 3 типах массивов: отсортированном, отсортированном в обратном порядке и случайно сгенерированном. Номер колонки 1 соответствует отсортированному, 2 - обратно отсортированному, а 3 и 4 - случайно сгенерированным. Усредненные значения количества сравнений и перемещений для различных n (размеров массивов) приведены ниже.

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	45	45	45	45	45.00
	Перемещения	0	5	7	8	5.00
100	Сравнения	4950	4950	4950	4950	4950.00
	Перемещения	0	50	97	93	60.00
1000	Сравнения	499500	499500	499500	499500	499500.00
	Перемещения	0	500	995	995	622.50
10000	Сравнения	49995000	49995000	49995000	49995000	49995000.00
	Перемещения	0	5000	9990	9991	6245.25

Таблица 1: Результаты работы сортировки простым прямым выбором

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	63	58	37	40	49.50
	Перемещения	9	9	10	9	9.25
100	Сравнения	5148	5098	736	806	2947.00
	Перемещения	99	99	163	159	130.00
1000	Сравнения	501498	500998	12171	11697	256591.00
	Перемещения	999	999	2393	2392	1695.75
10000	Сравнения	50014998	50009998	166091	169521	25090152.00
	Перемещения	9999	9999	31659	31592	20812.25

Таблица 2: Результаты работы быстрой сортировки

Переходя к теоретическому рассмотрению сортировок, заметим, что сортировка выбором делает $n-1+n-2+n-3+...+1+0 = O(\frac{n^2}{2})$ сравнений и $O(n)$ обменов. Формальное подробное объяснение дается Робертом Сэджвиком [1]. Анализ быстрой сортировки сложен, поэтому мы лишь сошлемся на общеизвестные оценки [1]: $O(n \log n)$ - как для числа сравнений, так и для числа обменов.

Экспериментальные данные наших реализаций согласуются с теоретическими оценками. Для построения аппроксимирующих функций использовались данные запусков на случайных массивах, что важно для эмпирического анализа алгоритмов [1]. Ниже на графиках представлены данные запусков алгоритмов на случайных массивах различных размеров.

Рис.1 Зависимость количества сравнений от n

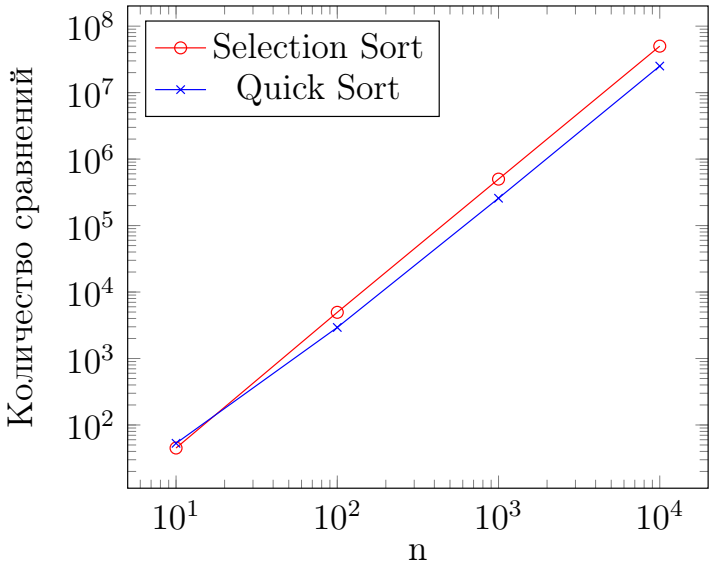
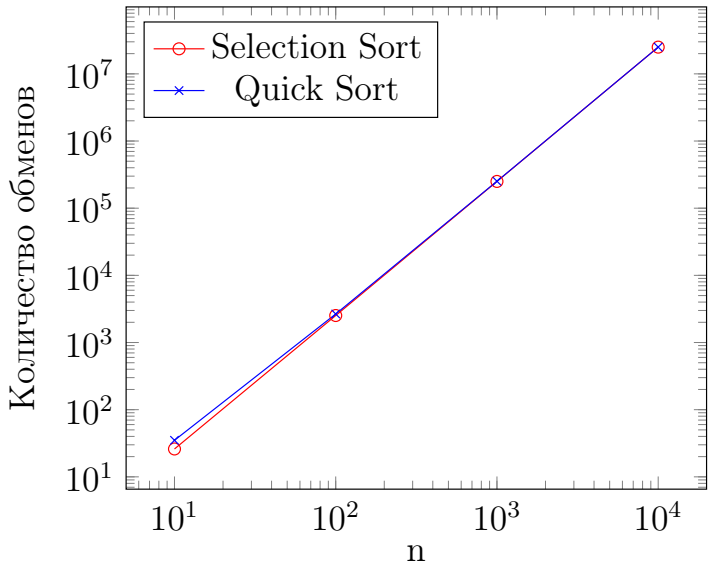


Рис.2 Зависимость количества обменов от n



Далее, можно заметить, что количество сравнений хорошо аппроксимируется функциями $l_1 = 0.5n^2$ для сортировки выбором и $l_2 = 553n \log n - 1.14 \cdot 10^6$ для быстрой сортировки. У быстрой сортировки довольно большая константа, поэтому ее преимущества ощутимы при $n > 100$, однако это фактор зависит от конкретного процессора. Количество обменов аппроксимируется функциями $l_3 = 2568n - 817939$ для сортировки выбором и $l_4 = 2.77 \cdot 10^2 \cdot n \log n - 5.71 \cdot 10^5$. В рассматриваемом диапазоне данных l_3 и l_4 практически неотличимы. Очевидно, по совокупности факторов преимущество стоит отдать быстрой сортировке.

Структура программы и спецификация функций

Основную ценность представляют реализации сортировок, сделанные на основе описаний классических алгоритмов [1]. Все функции работают с числами типа double. Однако в реализации сортировок удалось добиться определенной гибкости через использование указателей на функции. В качестве аргументов в алгоритм сортировки передаются указатели на функцию-компаратор и функцию обмена. В программе эти функции представлены в двух видах: с подсчетом и без. Функции с подсчетом являются обертками над базовыми.

Приведем список реализованных функций.

double rand_double (void) - генерирует случайный элмент массива

double * generate_sorted (size_t n) - генерирует отсортированный массив заданной длины

double * generate_reverse (size_t n) - генерирует обратно отсортированный массива заданной длины

double * generate_random (size_t n) - генерирует случайный массив заданной длины

double * deep_copy (const double *src, size_t n) - выделяет память под новый массив копирует в него содержимое исходного

void print_arr (double *arr, size_t n) - печатает массив на стандартный поток вывода

bool cmp (double other, double another) - компаратор

double max (double a, double b) - возвращает максимум из двух чисел

bool equals (double other, double another) - возвращает истину, если числа равны

bool check_trivial (const double *src, const double *sorted, size_t n) - проверяет корректность сортировки тривиального массива

bool check_reverse (const double *src, const double *sorted, size_t n) - проверяет корректность сортировки обратно отсортированного массива

bool permucheck (double *src, double *sorted, size_t n) - проверяет, что массив src явлеяется перестановкой элементов массива sorted

bool is_sorted (double *sorted, size_t n) - проверяет, что в массиве не нарушен порядок

bool check_correct (double *src, double *sorted, size_t n) - проверяет корректность сортировки массива sorted

void swap_with_count (double *other, double *another) - обмен двух элементов местами с увеличением глобального счетчика обменов

bool cmp_with_count (double other, double another) - компаратор с глобальным счетчиком обменов

double avg (const int arr[], int n) - возвращает среднее значение элементов массива avg

void print_case (int size, int swaps[], int cmps[]) - печатает часть финального вывода, которая соответветствует переданным значениям

void zero (void) - обнуляет глобальные счетчики обменов и сравнений

void driver (void(*method)(double *, size_t, bool(*)(double, double), void(*)(double *, double *))) - производит запуск тестов, проверку корректность

проверки работы, выводит статистику запусков для произвольного метода сортировки method

void selection (double *arr, size_t sz, bool(*cmp)(double, double), void(*swap)(double *, double *)) - классическая сортировка выбором с произвольным компаратором и функцией обмена

size_t partition (double *arr, size_t l, size_t r, bool(*cmp)(double, double), void(*swap)(double *, double *)) - функция, осуществляющая операцию разбиения из быстрой сортировки

void quick_sort_r (double *arr, int l, int r, bool(*cmp)(double, double), void(*swap)(double *, double *)) - рекурсивная быстрая сортировка

void quick_sort (double *arr, size_t sz, bool(*cmp)(double, double), void(*swap)(double *, double *)) - обертка на рекурсивной быстрой сортировкой

void run_with_print (void) - запуск сортировок для "визуального"тестирования

Отладка программы, тестирование функций

Для тестирования тривиальных случаев (исходный массив отсортирован и отсортирован в обратном порядке) были написаны специальные тестирующие функции, которые проверяют, что элементы следуют в прямом и обратном порядках соответственно. Случай со случайным массивом проверяется с помощью двух вспомогательных функций: is_sorted и permucheck. Первая из них проверяет, что порядок следования элементов в массиве корректен, а вторая за $O(n^2)$ попарно сравнивает элементы исходного и обработанного массивов, убеждаясь, что второй массив есть перестановка элементов первого.

Также были написаны вспомогательные функции для печати массива на консоль и визуального сравнения массивов при маленьком n.

Анализ допущенных ошибок

В процессе разработки часто возникали нерациональные идеи усложнения организации тестирования и служебных функций. Часть из них была в какой-то степени реализована, однако с учетом большого объема уже написанного кода (который тоже надо отлаживать и выверять) такая работа была приостановлена. Это говорит о необходимости продумывания архитектурных решений, а также минимальных требований к результату еще до написания кода как такого.

Список литературы

[1] Robert Sedgewick and Michael Schidlowsky. 1998. Algorithms in Java, Third Edition, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.