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

Отчет по заданию №1

«Методы сортировки»

Вариант 3 / 3 / 3 / 5

Выполнил:
студент 106 группы
Ясько М. О.

Преподаватели:
Корухова Л. С.
Манушин Д. В.

Москва
2024

Содержание

Постановка задачи **2**

Результаты экспериментов **3**
Сортировка Шелла 3
Пирамидальная сортировка 3
Выводы 4

Структура программы и спецификация функций **5**

Отладка программы, тестирование функций **7**
Отладка генераторов 7
Отладка сортировок 8
Пирамидальная сортировка 8

Анализ допущенных ошибок **9**

Список литературы **10**

Постановка задачи

Задача эксперимента - создать и сравнить два метода сортировки: сортировку Шелла и пирамидальную сортировку. Сравниваться будут массивы элементов типа double по неубыванию модулей чисел. Для сравнения методов сортировки они будут использованны на парах одинаковых массивов размером в 10, 100, 1000 и 10000 элементов в 4-х разных вариантах, два из которых случайны, а оставшиеся два заранее упорядочены по неубыванию и невозрастанию. Перед сравнением результаты будут записаны в таблицы такого вида:

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения					
	Перемещения					
100	Сравнения					
	Перемещения					
1000	Сравнения					
	Перемещения					
10000	Сравнения					
	Перемещения					
100000	Сравнения					
	Перемещения					

Таблица 1: Шаблон таблицы с результатами сортировки

Результаты экспериментов

Массивы были построенны с использовнием сида srand(1709061327)

Сортировка Шелла

Сортировка Шелла является модификацией сортировки вставкой. Массив сортируется вставкойшагай перешагивая на каждый d-й элемент, где d постепенно уменьшается. В данном эксперименте была использована стандартная последовательность уменьшения шага ($d_1 = n/2$, $d_2 = n/4$, ... , $d_i = n/2^i$). Теоретическая асимптотическая оценка времени выполнения $\mathcal{O}(n^2)$

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	22	27	32	27	27
	Перемещения	0	13	12	11	9
100	Сравнения	503	668	906	854	733
	Перемещения	0	260	442	404	276.5
1000	Сравнения	8006	11716	15613	14394	12432
	Перемещения	0	4700	8128	6899	4932
10000	Сравнения	120005	172578	258508	261759	203212.5
	Перемещения	0	62560	143533	146811	88226
100000	Сравнения	1500006	2244582	4275152	4455471	3118803
	Перемещения	0	844557	2825823	3006001	1669095

Таблица 2: Результаты работы сортировки Шелла

Пирамидальная сортировка

В пирамидальной сортировке алгоритм выполнения имеет следующий вид:

Шаг 1: Преобразовать массив в max-heap

Шаг 2: Переставить первый (наибольший) элемент в конец

Шаг 3: Преобразовать в max-heap массив всех элементов,
кроме переставленных в конец

Шаг 4: Повторять порядок действий с шага 2,
пока не закончатся элементы в куче

Теоретическая асимптотическая оценка времени алгоритма - $\mathcal{O}(n \log n)$

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	41	35	38	40	38.5
	Перемещения	31	22	29	30	28
100	Сравнения	1081	944	1028	1028	1020
	Перемещения	641	517	581	581	580
1000	Сравнения	17583	15965	16811	16826	56796
	Перемещения	9709	8317	9037	9076	9035
10000	Сравнения	244460	226682	235360	235418	235480
	Перемещения	131957	116697	124276	124236	124291,5
100000	Сравнения	3112500	2926640	3019457	3019942	3019635
	Перемещения	1650698	1497435	1574782	1575245	1574540

Таблица 3: Результаты работы пирамидальной сортировки

Выводы

Экспериментальные данные подтверждают теоретические оценки. На малых размерах массива сортировка Шелла выигрывает из-за большой константы у пирамидальной сортировки, так как в ней совершаются дополнительные действия для превращения в max-heap. Но благодаря своей асимптоматике пирамидалная сортировка проявляет себя гораздо лучше на больших размерах массивов.

Структура программы и спецификация функций

Здесь приведён список использованных функций, а также пояснительные комментарии к ним. Функция генерации массива приведена полностью для пояснения принципа её работы.

```
// Счётчики сравнений и перемещений
int comp_count, swap_count;

/* Вспомагательные функции */

// Модуль 64-битной десятичной дроби
double dabs(double a);

// Функция сравнения для qsort
int dcomp(const void *a, const void *b);

// Вывод массива
void print_arr(double *arr, int n);

// Перестановка переменных
void swap(double *a, double *b);

// Переворот массива для обратного порядка
void reverse(double *arr, int n);

double *arrgen(int type, int n) {
    double *arr = malloc(sizeof(double) * n);

    for (int i = 0; i < n; i++) { // генерирует случайные числа от -100000000 до 10
        arr[i] = ((double)rand() / (double)RAND_MAX) * 200000000 - 100000000;
    }

    switch (type) {          // 3 вида генериации : 1 - неубывающий, -1 - невозраста
    case 1:
        qsort(arr, n, sizeof(double), dcomp);
        break;

    case -1:
        qsort(arr, n, sizeof(double), dcomp);
        reverse(arr, n);
        break;
    }

    return arr;
}
```

```c
// Сортировка Шелла со стандартным выбором шага
void shell_sort(double *arr, int n);

// Функция приведения к max-куче
void heapify(int i, double *arr, int n);

// Пирамидальная сортировка
void pyramid_sort(double *arr, int n);


// Специальные функции для примера в следующем сегменте

// Функция выписывающая LaTeX код для таблицы
void verbosinator(int a, int b, const char *s, double *arr);


// Обе функции используют verbosinator
void verbose_heapify(int i, double *arr, int n);

void verbose_pyramid_sort(double *arr, int n);

/* Функция выполняющая тесты */
void show_results(int n, int type, char *message) {
    double *arr1 = arrgen(type, n);
    double *arr2 = malloc(n * sizeof(double));

    printf("%s \n", message);

    /* Генерирует массив и клонирует его для проверки обих сортировок*/
    memcpy(arr2, arr1, n * sizeof(double));

    swap_count = comp_count = 0;
    shell_sort(arr1, n);
    printf("\tShell sort:\n");
    printf("\tAmount of swaps:\t%d\n", swap_count);
    printf("\tAmount of comparisons:\t%d\n\n", comp_count);

    swap_count = comp_count = 0;
    pyramid_sort(arr2, n);
    printf("\tPyramid sort:\n");
    printf("\tAmount of swaps:\t%d\n", swap_count);
    printf("\tAmount of comparisons:\t%d\n\n", comp_count);

    free(arr1);
```

```
    free(arr2);
}

void test_generation(int n) {
    /* Функция выводящаярезультвты генерации (дляпроверки на правильную генерацию)
}

int main()
{
    /*   Задание сида, генерация массивов и активация тестов   */
}
```

Отладка программы, тестирование функций

Отладка генераторов

- Для отладки функции генерации массива `arrgen()` она была запущена для массива размером в 1000 элементов, и на полученных значениях был построен график (рис. 1) с помощью Desmos, из которого видно, что генерируемые значения распределены достаточно равномерно в диапазоне $[-10^8; 10^8]$.

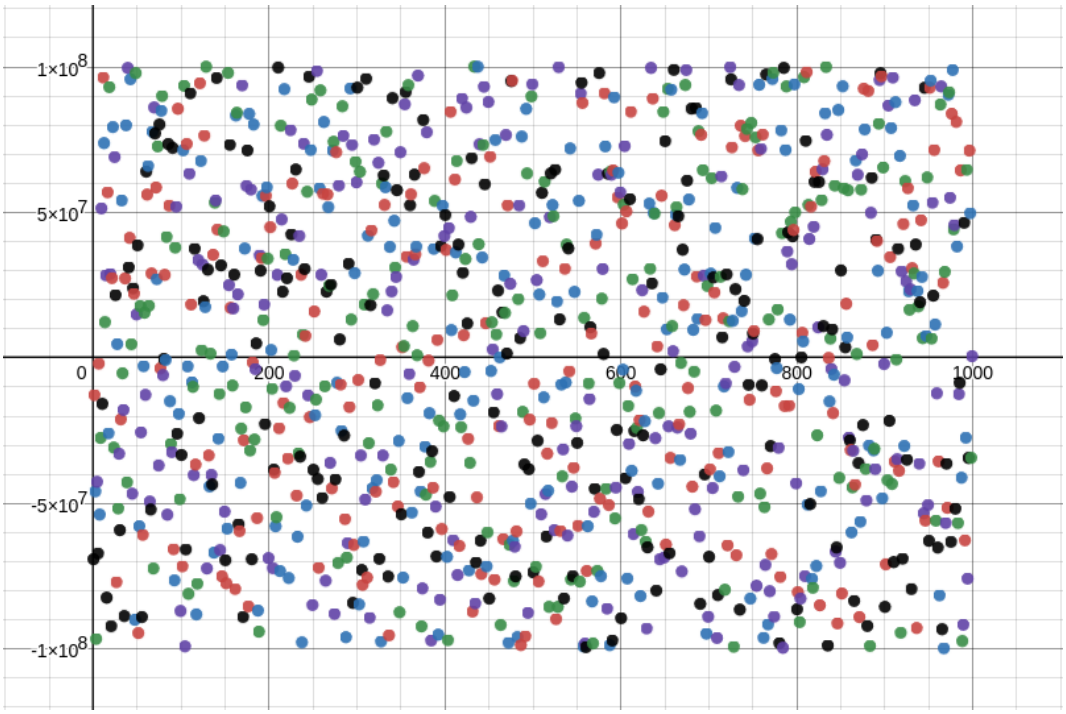

Рис. 1: График распределения 1000 значений `gen_elem()`

Отладка сортировок

Для демонстрации корректности подсчёта количества сравнений и перестановок продемонстрируем работу сортировок на примере конкретного массива. Для простоты восприятия массив был выбран небольшой длины и с небольшими значениями. В таблицах сравниваемые элементы выделены синим цветом, а перемещаемые значения подчёркнуты. В двух левых колонках указаны количества сделанных сравнений и перестановок на данный момент. Таблица была сгенерирована автоматически при помощи эквивалентных verbose функций.

Пирамидальная сортировка

Сравн.	Перест.	0	1	2	3	4
Исходный		3.1	1.0	6.4	3.5	7.6
Построение кучи						
1	0	3.1	**1.0**	6.4	**3.5**	7.6
2	0	3.1	1.0	6.4	**3.5**	**7.6**
2	1	3.1	1.0	6.4	3.5	7.6
3	1	**3.1**	**7.6**	6.4	3.5	1.0
4	1	3.1	**7.6**	**6.4**	3.5	1.0
4	2	3.1	7.6	6.4	3.5	1.0
5	2	7.6	**3.1**	6.4	**3.5**	1.0
6	2	7.6	3.1	6.4	**3.5**	**1.0**
6	3	7.6	3.1	6.4	3.5	1.0
Сортировка						
6	4	7.6	3.5	6.4	3.1	1.0
7	4	**1.0**	**3.5**	6.4	3.1	7.6
8	4	1.0	**3.5**	**6.4**	3.1	7.6
8	5	1.0	3.5	6.4	3.1	7.6
8	6	6.4	3.5	1.0	3.1	7.6
9	6	**3.1**	**3.5**	1.0	6.4	7.6
10	6	3.1	**3.5**	**1.0**	6.4	7.6
10	7	3.1	3.5	1.0	6.4	7.6
10	8	3.5	3.1	1.0	6.4	7.6
11	8	**1.0**	**3.1**	3.5	6.4	7.6
11	9	1.0	3.1	3.5	6.4	7.6
11	10	3.1	1.0	3.5	6.4	7.6
11	11	1.0	3.1	3.5	6.4	7.6
Итоговый		1.0	3.1	3.5	6.4	7.6

Получившиеся значения совпали с выводимыми программой. Таким образом, подсчёт производится корректно.

Анализ допущенных ошибок

- Изначально, при выполнении сортировки Шелла, из-за неправильного порядка выполнения функции выполнялись лишние перемещения, что приводило, например, к ненулевому колличеству перемещений в уже отсортированном массиве.

Список литературы

Википедия : Сортировка Шелла
Хабр : Пирамидальные сортировки
Лорин Г. Сортировка и системы сортировки. — М.: Наука, 1983.
Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.