МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский государственный университет им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет по заданию №1

На тему «Методы сортировки»

Вариант 1 4 1 4

Работу выполнил:
Студент 1 курса 106 группы
Факультета вычислительной математики и кибернетики
Наклескин Никита Владимирович

Преподаватель:
Корухова Людмила Сергеевна

Москва
2022

Содержание

Постановка задачи

Требуется реализовать функции сортировки методом "пузырька" и методом быстрой сортировки (рекурсивная реализация) для целых чисел в порядке невозрастания абсолютных величин для исследования и сравнения эффективности алгоритмов. Тип элементов массива long long int. После реализации алгоритмов требуется эмпирически сравнить их асимптотическую сложность путем сравнения числа перестановок и сравнений в результате работы алгоритмов для различного количества входных данных. Генерация исходных массивов реализуется через отдельные функции. При исследовании будут сравниваться 3 различных вида входных данных.

1. Случайные числа
2. Упорядоченные от N до 1
3. Упорядоченные от 1 до N

После реализации алгоритмов необходимо провести исследование эффективности алгоритмов путем сравнения числа сравнений и перестановок и представить данные в таблице вида:

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

Результаты экспериментов

В результате проведённых экспериментов была подтверждена асимптотическая оценка алгоритма быстрой сортировки и сортировки методом «пузырька». При оценке производилось сравнение числа перестановок и сравнений в результате работы алгоритмов на одних данных. Для сравнения использовалось 3 вида различных данных, рандомно сгенерированные последовательности чисел, упорядоченные последовательности чисел и числа, упорядоченные в обратном порядке. В результате можно сделать вывод, что алгоритм сортировки методом пузырька неэффективен при больших объемах данных так как его асимптотическая сложность $O(n^2)$. Наименьшей эффективности алгоритм достигает с упорядоченными в обратном порядке данными. Так же можно однозначно сказать, что алгоритм быстрой сортировки более эффективен, чем сортировка методом «пузырька». Асимптотическая сложность алгоритма быстрой сортировки в усредненном случае составляет

$O(n * log(n))$. Однако в наихудшем случае его сложность может доходить до $O(n^2)$.

Быстрая сортировка, рекурсивная реализация:

n	Параметр	Номер сгенерированного массива			Среднее значение
		1	2	3	
10	Сравнения	17	22	15	18
	Перемещения	12	0	5	5
100	Сравнения	374	486	392	417
	Перемещения	168	0	50	72
1000	Сравнения	7323	7996	7005	7441
	Перемещения	2433	0	500	977
10000	Сравнения	95855	113644	103657	104385
	Перемещения	32102	0	5000	12367

Сортировка методом пузырька

n	Параметр	Номер сгенерированного массива			Среднее значение
		1	2	3	
10	Сравнения	45	9	81	45
	Перемещения	22	0	45	22
100	Сравнения	8910	99	9801	6270
	Перемещения	8910	0	4950	4620
1000	Сравнения	937062	999	998001	645354
	Перемещения	248710	0	499500	249403
10000	Сравнения	98470152	9999	99980001	66153384
	Перемещения	25282188	0	49995000	25092396

Структура программы и спецификации функций

Функция gen_rand

```
void gen_rand(int n)
{
    srand(time(NULL));
    FILE* input = fopen("data/data1.txt", "a");
    for(int i = 0; i < n; i++)
    {
        fprintf(input, "%d\n", (-1) * n + rand()%(2 * n + 1));
    }
    fclose(input);
}
```

Функция получает на вход число n - количество чисел, которое необходимо сгенерировать. В результате работы программы она создает файл, в котором содержится n случайно сгенерированных чисел.

Функция gen_rev

```
void gen_rev(int n)
{
    FILE* input = fopen("data/data2.txt", "a");
    for(int i = n; i > 0; i--)
    {
        fprintf(input, "%d\n", i);
    }
    fclose(input);
}
```

Функция получает на вход число n - количество чисел, которое необходимо сгенерировать. В результате работы программы она создает файл, в котором содержится n чисел, упорядоченных от n до 1.

Функция gen_sorted

```
void gen_sorted(int n)
{
    FILE* input = fopen("data/data3.txt", "a");
    for(int i = 1; i <= n; i++)
    {
        fprintf(input, "%d\n", i);
    }
    fclose(input);
}
```

Функция получает на вход число n - количество чисел, которое необходимо сгенерировать. В результате работы программы она создаёт файл, в котором содержится n чисел, упорядоченных от 1 до n.

Функция bubble_sort

Функция получает на вход числовой массив a, число n - количество элементов в массиве и вспомогательный массив count[2] в котором будет храниться количество сравнений и перестановок в результате работы алгоритма, где count[0] - количество сравнений, а count[1] - количество перестановок. Алгоритм сортирует массив путем поочередного сравнения соседних пар чисел, таким образом после каждого прохода массива самое маленькое число будет оказываться в конце массива. Асимптотическая сложность алгоритма $O(n^2)$.

Функция qsort_alg

Функция получает на вход числовой массив a, число first - индекс начала сортируемой области массива, число last - индекс конца сортируемой области массива, число n - количество элементов в массиве и вспомогательный массив count[2] в котором будет храниться количество сравнений и перестановок в результате работы алгоритма, где count[0] - количество сравнений, а count[1] - количество перестановок. Алгоритм сортирует массив путем разбиения его на две части с средним элементом. Справа от этого элемента будут находится числа, меньшие, чем сам элемент, а слева большие. Далее алгоритм аналогичным образом отсортирует левую и правые части массива. Таким образом алгоритм будет выполняться рекурсивно, пока массив не будет отсортирован.

Отладка программы, тестирование функций

Пошаговый пример работы метода «пузырька»

```
void bubble_sort(long long* a, int n, int count[2])
{
    long long c;
    long long count_cmp = 0, count_swap = 0;
    for(int j = 0; j < n - 1; j++)
    {
        for(int i = 0; i < n - 1; i++)
        {
            count_cmp++;
            if(abs(a[i]) < abs(a[i + 1]))
            {
                count_swap++;
                c = a[i];
                a[i] = a[i + 1];
                a[i + 1] = c;
            }
        }
    }
    count[0] = count_cmp;
    count[1] = count_swap;
}
```

Возьмем массив: **9 2 8 0 2 5 1 3 -10 -2** и пошагово отсортируем его по невозрастанию абсолютных величин. 9 >= 2, все верно, идем дальше. 2 <= 8, значит меняем их местами. Получаем: **9 8 2 0 2 5 1 3 -10 -2**. 2 >= 0, значит идем дальше, 0 <= 2, меняем их местами. Получаем **9 8 2 2 0 5 1 3 -10 -2**. 0 <= 5, меняем их местами. Получаем: **9 8 2 2 5 0 1 3 -10 -2**. 0 <= 1, меняем их местами. Получаем: **9 8 2 2 5 1 0 3 -10 -2**. 0 <= 3, меняем их местами. Получаем: **9 8 2 2 5 1 3 0 -10 -2**. 0 <= abs(-10), меняем их местами. Получаем: **9 8 2 2 5 1 3 -10 0 -2**. 0 <= abs(-2), меняем их местами. Получаем: **9 8 2 2 5 1 3 -10 -2 0**. В результате данных перестановок самое маленькое число ушло в конец массива. На этом этапе вложенный цикл закончит работу и массив начнет снова сортироваться с первого элемента. Продолжим выполнение алгоритма. 9 >= 8, все верно, идем дальше. 2 >= 2, все верно, идем дальше. 2 <= 5, меняем их местами. Получаем: **9 8 2 5 2 1 3 -10 -2 0**. 2 >= 1, все верно, идем дальше 1 <= 3, меняем их местами. Получаем: **9 8 2 5 2 3 1 -10 -2 0**. 1 <= abs(-10), меняем их местами. Получаем: **9 8 2 5 2 3 -10 1 -2 0**. 1 <= abs(-2), меняем их местами. Получаем: **9 8 2 5 2 3 -10 -2 1 0**. Продолжаем сортировку массива с первого элемента. 9 >= 8, все верно, идем дальше. 8 >= 2, все верно, идем дальше. 2 <= 5, меняем их местами. Получаем: **9 8 5 2 2 3 -10 -2 1 0**. 2 >= 2, все верно, идем дальше. 2 <= 3, меняем их местами. Получаем: **9 8 5 2 3 2 -10 -2 1 0**. 2 < 10, меняем их местами. Получаем: **9 8 5 2 3 -10 2 -2 1 0**. 2 >= abs(2), все верно, идем дальше. abs(-2) >= 1, все верно, идем дальше. 1 >= 0, все верно, идем дальше. Продолжаем сортировку массива с первого элемента. 9 >= 8, все верно, идем дальше. 8 >= 5, все верно, идем дальше. 5 >= 2, все верно, идем дальше. 2 <= 3, меняем их местами. Получаем: **9 8 5 3 2 -10 2 -2 1 0**. 2 <= abs(-10), меняем их местами. Получаем: **9 8 5 3 -10 2 2 -2 1 0**. 2 >= 2, все верно, идем дальше. 2 >= abs(-2), все верно, идем дальше. abs(-2) >= 1, все верно, идем дальше. 1 >= 0. Продолжаем сортировку массива с первого элемента. 9 >= 8, все верно, идем дальше. 8 >= 5, все верно, идем дальше. 5 >= 3, все верно, идем дальше. 3 <= abs(-10), меняем их местами. Получаем: **9 8 5 -10 3 2 2 -2 1 0**. 3 >= 2, все верно, идем дальше. 2 >= 2, все верно, идем дальше. 2 >= abs(-2), все верно, идем дальше. abs(-2) >= 1, все

верно, идем дальше. 1 >= 0, все верно, идем дальше. Продолжаем сортировку массива с первого элемента. 9 >= 8, все верно, идем дальше. 8 >= 5, все верно, идем дальше. 5 <= abs(-10), меняем их местами. Получаем: **9 8 -10 5 3 2 2 -2 1 0**. 5 >= 3, все верно, идем дальше. 3 >= 2, все верно, идем дальше. 2 >= 2, все верно, идем дальше. 2 >= abs(-2), все верно, идем дальше. abs(-2) >= 1, все верно, идем дальше. 1 >= 0, все верно, идем дальше. Продолжаем сортировку массива с первого элемента. 9 >= 8, все верно, идем дальше. 8 <= abs(-10), меняем их местами. Получаем: **9 -10 8 5 3 2 2 -2 1 0**. 8 >= 5, все верно, идем дальше. 3 >= 2, все верно, идем дальше. 2 >= 2, все верно, идем дальше. 2 >= abs(-2), все верно, идем дальше. abs(-2) >= 1, все верно, идем дальше. 1 >= 0, все верно, идем дальше. Продолжаем сортировку массива с первого элемента. 9 <= abs(-10), меняем их местами. Получаем: **-10 9 8 5 3 2 2 -2 1 0**. 9 >= 8, все верно, идем дальше. 8 >= 5, все верно, идем дальше. 3 >= 2, все верно, идем дальше. 2 >= 2, все верно, идем дальше. 2 >= abs(-2), все верно, идем дальше. abs(-2) >= 1, все верно, идем дальше. 1 >= 0, все верно, идем дальше. Массив отсортирован.В результате получилось 81 сравнение и 20 перестановок.

Пошаговый пример работы алгоритма быстрой сортировки

```
void qsort_alg(long long* a, int first, int last, int count[2])
{
    if(last > first)
    {
        long long count_cmp = 0, count_swap = 0;
        int left = first, right = last;
        int mid = abs(a[(left + right) / 2]);
        while(left <= right)
        {
            while(abs(a[left]) > mid)
            {
                count_cmp++;
                left++;
            }
            while(abs(a[right]) < mid)
            {
                count_cmp++;
                right--;
            }
            if(left <= right)
            {
                if(left != right)
                {
                    count_swap++;
                    long long c = a[left];
                    a[left] = a[right];
                    a[right] = c;
                }
                left++;
                right--;
            }
        }
        count[0] += count_cmp;
        count[1] += count_swap;
        qsort_alg(a, first, right, count);
        qsort_alg(a, left, last, count);
    }
}
```

Возьмем массив: **9 2 8 0 2 5 1 3 -10 -2** и пошагово отсортируем его по невозрастание абсолютных величин. В качестве разделительного элемента будет 5. Идем по массиву с правой стороны. 9 > 5, все верно, идем дальше. 2 < 5, неверно, переходим на правую сторону и ищу элемент, больший чем 5. abs(-2) < 5, все верно, идем дальше. abs(-10) > 5, неверно, меняем 2 и -10 местами. Получаем: **9 -10 8 0 2 5 1 3 2 -2**. Идем дальше. 8 > 5, все верно. 0 < 5, неверно, переходим на правую сторону. 3 < 5, все верно, идем дальше. 1 < 5, все верно, идем дальше. 5 = 5, неверно, меняем 5 и 0 местами. Получаем: **9 -10 8 5 2 0 1 3 2 -2**. Поскольку дальше не будет выполняться условие left <= right, то функция уйдет в рекурсию. Рассмотрим правую часть рекурсии: **9 -10 8**. В качестве разделительного элемента будет -10. 9 < abs(-10), неверно, переходим в правую часть. 8 < abs(-10), все верно, идем дальше. -10 == -10, неверно, меняем 9 и -10 местами. Получаем: **-10 9 8**. Поскольку дальше не будет выполняться условие left <= right, то функция уйдет в рекурсию. С левой стороны длина массива будет равна 1, следовательно -10 - наибольшее число и функция не уйдет в рекурсию в эту

сторону. Рассмотрим правую часть рекурсии: **9 8**. В качестве разделительного элемента будет 8. Данная пара чисел упорядочена, значит, никаких перестановок не будет. Рассмотрим левую часть: **2 0 1 3 2 -2**. Разделительным элементом будет 3. 2 < 3, неверно, переходим в правую часть. abs(-2) < 3, верно, идем дальше. 2 < 3, верно, идем дальше. 3 == 3, неверно, меняем 2 и 3 местами. Получаем: **3 0 1 2 2 -2**. С левой стороны длина массива будет равна 1, следовательно 3 - наибольшее число и функция не уйдет в рекурсию в эту сторону. Рассмотрим правую часть: **0 1 2 2 -2**. В качестве разделительного элемента будет 2. 0 < 2, неверно, переходим в правую часть. abs(-2) == 2, неверно, меняем 0 и -2 местами. Получаем: **-2 1 2 2 0**. 1 < 2, неверно, переходим в правую часть. 2 == 2, неверно, меняем 1 и 2 местами. Получаем: **-2 2 2 1 0**. Далее функция уйдет в рекурсию и закончит работу без дальнейших перестановок. Массив отсортирован. Результат: **-10 9 8 5 3 2 -2 2 1 0**. В результате работы алгоритма было выполнено 5 перестановок и 19 сравнений.

Анализ допущенных ошибок

В процессе выполнения задания мною было допущено несколько ошибок, некоторые из которых потребовали достаточно много времени на поиск и устранение.

Неправильная работа с файлами.

В изначальном варианте программы я просто заново сканировал файл в массив, не закрывая файл. В результате в алгоритм быстрой сортировки массив поступал уже отсортированный, а я не мог понять причину, по которой программа не считает количество перестановок, хотя массив сортируется.

Неправильный qsort

После нахождения и исправления ошибки с файлами я обнаружил, что алгоритм не до конца сортирует массив. Насколько я понял было не правильно задано условия входа в рекурсию. В процессе устранения этой ошибки алгоритм был полностью переписан.

Так же в процессе сдачи задание преподаватель указал мне на то, что алгоритм переставляет числа даже в отсортированном массиве. Причиной стало неправильно прописанное условие перестановки. Программа переставляло число с самим собой. Ошибка была устранена добавлением дополнительного условия.

Функция abs() для INT_MIN

В процессе сдачи задания преподаватель указал на то, что функция abs() некорректно работает для INT_MIN. Решением проблемы был переход на 64 битные числа и функцию llabs().

Неправильное оформление заголовочного файла

В процессе создания заголовочного файла sort_func.h я забыл прописать необходимые библиотеки для его работы.

Ошибки по кодстайлу

В программе использовалось различное обозначение для массива count[2] (int* count и int count[2]), которое впоследствии было приведено к одному виду.

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.