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

Отчет по заданию №1

«Методы сортировки»

Вариант 2 / 2 / 1 / 5

Выполнил:
студент 104 группы
Дыскина М. А.

Преподаватель:
Гуляев Д. А.

Москва
2022

Содержание

Постановка задачи

В программе необходимо реализовать два метода сортировки массива чисел и провести их экспериментальное сравнение. При реализации каждого метода необходимо вычислить число сравнений элементов и число перемещений (обменов) элементов.

Для сравнения используются методы быстрой и медленной сортировки, а именно, пирамидальная сортировка и сортировка методом «пузырька». Элементами массива являются 64-разрядные целые числа (long long int). В результате каждой из сортировок числа должны быть упорядочены по невозрастанию.

Результаты экспериментов

В данном разделе приведены значения счетчиков сравнений и перемещений в методе "пузырька"и в пирамидальной сортировке. Номера, отвечающие за вид генерации массива: 1 - отсортирован по неубыванию модулей, 2 - отсортирован по невозрастанию модулей (по такому же принципу функции сортируют массив), 3 и 4 - массив, элементы которого не упорядочены.

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	45	45	45	45	45
	Перемещения	45	0	23	16	21
100	Сравнения	4950	4950	4950	4950	4950
	Перемещения	4950	0	2766	2247	2490.75
1000	Сравнения	499500	499500	499500	499500	499500
	Перемещения	499500	0	253583	251458	251135.25
10000	Сравнения	49995000	49995000	49995000	49995000	49995000
	Перемещения	49995000	0	25101663	25064129	25040198

Таблица 1: Результаты работы сортировки методом "пузырька"

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	61	87	74	80	75.5
	Перемещения	22	31	28	29	27.5
100	Сравнения	1606	2033	1805	1832	1819
	Перемещения	517	641	579	590	581.75
1000	Сравнения	27148	31985	29666	29768	29641.75
	Перемещения	8317	9709	9068	9103	9049.25
10000	Сравнения	388676	439821	413414	413027	413734.5
	Перемещения	116697	131958	124236	124114	124251.25

Таблица 2: Результаты работы пирамидальной сортировки

С теоретической точки зрения, число сравнений в строго обменном алгоритме (метод "пузырька"): $C = (n^2 - n)/2$, а минимальное, среднее и максимальное число перемещений элементов (присваиваний) равно, соответсвенно: $M_{min} = 0, M_{avg} = 3 * (n^2 - n)/2, M_{max} = 3 * (n^2 - n)/4$. [1]

В алгоритме пирамидальной сортировки среднее число перемещений элементов (присваиваний) равно: $(n/2) * log_2(n)$, максимальное число перемещений равно $n * log_2(n)$, а среднее число сравнений $2N * log_2 n$. [1]

Сортировка методом "пузырька"работает в среднем за O(n^2), а пирамидальная сортировка - это алгоритм сортировки с временем работы O($n * log_2 n$) , где n — количество элементов для сортировки. [1]

Структура программы и спецификация функций

Функция **BubbleSort** - сортировка методом "пузырька"

Выполняет сортировку n элементов массива, на который ссылается указатель a. Алгоритм состоит в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. Прототип функции: void BubbleSort(long long int *a, int n). Параметры функции: a - указатель на сортируемый массив, n - количество элементов в массиве.

Функция **HeapSort** - пирамидальная сортировка

Выполняет сортировку n элементов массива, на который ссылается указатель a. Алгоритм состоит из построения пирамиды по сортируемому массиву и поочерёдного извлечения минимума кучи. Прототип функции: void HeapSort (long long int *a, int n). Параметры функции: a - указатель на сортируемый массив, n - количество элементов в массиве.

Функция **sift**

Просеивание элементов через кучу (формирование кучи). Просеивание построено на том, что наименьший элемент кучи оказывается в корне. Прототип функции: static void sift(long long int *a, int n, int i). Параметры функции: a - указатель на сортируемый массив, n - количество элементов в массиве (размер кучи), i - корневой узел поддерева, которое преобразовывается в двоичную кучу.

Функция **swap**

Обмен значениями двух переменных. Прототип функции: void swap (long long int *a, long long int *b). Параметры функции: a - указатель на первую переменную, b - указатель на вторую переменную.

Функции-компараторы **cmp1** и **cmp2** для встроенной функции qsort

Сравнение двух элементов. Прототип функции: int cmp1(const void *a, const void *b). Возвращаемое значение: если a меньше, равен или больше, чем b, функция должна вернуть отрицательное значение, ноль или положительное значение, соответственно (для функции cmp2 наоборот, так как cmp2 используется для сортировки qsort в обратном порядке) Параметры функции: a, b - указатели на элементы массива.

Функция **gen**

Генерация массива тремя способами, в зависимости от введенного параметра: массив, отсортированный по неубыванию; массив, отсортированный по невозрастанию; массив, элементы которого не упорядочены. Прототип функции: void gen (int n, int MassivType). Параметры функции: n - длина массива, MassivType - целое число, задающее порядок построения элементов в массиве.

В качестве счетчиков числа сравнений и перемещений для каждой из сортировок использовались глобальные переменные: long long int comp = 0 и long long int mov = 0. Для средних значений этих показателей использовались глобальные переменные: long double compB, movB, compP, movP.

Отладка программы, тестирование функций

Производилось поочередное тестирование каждой фукнции сортировки на 4 различных массивах (элементы отсортированы по неубыванию, по невозрастанию и два массива с неупорядоченными элементами), генерация которых была осуществлена с помощью функции void gen (int n, int MassivType). Для получения случайных чисел в массиве использовалась функция rand() стандартной библиотеки, возвращающая случайное целое число в диапазоне от 0 до 32767. Для инициализации генератора случайных чисел в функции main() был сделан вызов srand(time(NULL)). Так как используемым типом элементов массива был long long int (64-разрядные целые числа), то для генерации случайного 64-битного целого числа было использовано выражние вида rand() *rand() * rand() * rand() * rand().

Также на предыдщем этапе тестирования функций было составлено 3 массива в функции main(). Массив [7, 0, 3, -2, 6, 9, -5, 8, 1, 4] был использован для проверки функции BubbleSort. Результат выполнения: [9, 8, 7, 6, 4, 3, 1, 0, -2, -5]. Массив [-9, -3, 4, -10, -1, -6, 8, 8, -8, -6] был использован для проверки функции HeapSort. Реузльтат выполнения: [8, 8, 4, -1, -3, -6, -6, -8, -9, -10]. Массив [5], содержащий один элемент, был использован для проверки корректности работы счетчиков сравнений и перемещений для каждого метода сортировки. Результат выполнения: 0 0, 0 0.

Анализ допущенных ошибок

1. Не обнуляла счетчики (глобальные переменные) при вхождении в функции сортировок. Это было проблемой, так как с помощью функции gen для одного размера n создавалось сразу 4 массива с разным изначальным порядком элементов. При прохождении каждого из этих массивов поочередно через две функции сортировки, глобальные переменные не обнулялись и накапливали неверное значение. Исправила эту ошибку, поместив обнуление счетчиков в начало каждой из двух функций сортировок.

2. Изначально увеличивала счетчики сравнений и перемещений только внутри условных конструкций, которые сравнивали значения элементов. Но сравнение все равно осуществляется, даже при неподходящем результате, следовательно, счетчик должен увеличиться.

3. Изначально функции-компараторы для qsort работали только с типом int, что противоречило использованию типа long long int элементов исходного массива. Ошибка была исправлена.

4. Не сразу был учтен случай массива с одним элементом. Проблема была решена добавлением дополнительной проверки в функцию пирамидальной сортировки (функция методом "пузырька"с массивом из одного элемента работала верно)

Список литературы

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