МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени. М. В. ЛОМОНОСОВА ФАКУЛЬТЕТ
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ОТЧЕТ ПО ЗАДАНИЮ №1

«Методы сортировок»

Вариант 1 4 1 4

Исполнитель: Студент 106 группы
Арутюнов Эдгар Арамович
Преподаватели:
Соловьев Михаил Александрович
Корухова Людмила Сергеевна

МОСКВА
2018

Содержание

Постановка задачи

Реализовать два метода сортировки массива чисел и провести их экспериментальное сравнение. Тип элементов массива 32-битный int. Методы быстрой сортировки и пузырьковой. Для каждого из реализуемых методов необходимо предусмотреть возможность работы с массивами длины от 1 до $N(N \geq 1)$. Значение N в зависимости от варианта задания либо фиксировано, либо память под массив следует выделять динамически (рекомендуется последнее).

При реализации каждого метода вычислить ***число сравнений*** элементов и ***число перемещений*** (обменов) элементов.

Сравнение методов сортировки необходимо проводить на одних и тех же исходных массивах, при этом следует рассмотреть массивы разной длины. Для вариантов с фиксированным значением N рассмотреть, как минимум, $n = 10, 100, 1000, 10000$
Генерация исходных массивов для сортировки реализуется отдельной функцией, создающей в зависимости от заданного параметра и заданной длины конкретный массив, в котором:

- элементы уже упорядочены (1);
- элементы упорядочены в обратном порядке (2);
- расстановка элементов случайна (3, 4).

Результаты экспериментов

В результате проведенных экспериментов была подтверждена ассимтотическая оценка алгоритмов $Quick\,Sort - O(N \log N)$ и $Bubble\,Sort - O(N^2)$

Quick Sort

N	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	25	28	27	33	28,25
	Перемещения	0	5	6	7	4,50
100	Сравнения	543	548	736	634	615,25
	Перемещения	0	50	126	130	76,50
1000	Сравнения	8498	8506	11314	11819	10034,25
	Перемещения	0	500	2040	2034	1143,50
10000	Сравнения	119535	119548	169682	164404	143292,25
	Перемещения	0	5000	28131	28033	15291,00

В среднем, при увеличении N в ***10*** раз, число сравнений увеличивается в ***≈17*** раз, число перемещений в увеличивается в ***≈15*** раз.

Bubble Sort

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	9	90	54	72	56,25
	Перемещения	0	45	19	27	22,75
100	Сравнения	99	9900	8910	8514	6855,75
	Перемещения	0	4950	2427	2633	2502,50
1000	Сравнения	999	999000	939060	956043	723775,50
	Перемещения	0	499500	243397	248415	247828,00
10000	Сравнения	9999	999000	98350164	97870212	49307343,75
	Перемещения	0	49995000	24989315	24885204	24967379,75

В среднем, при увеличении N в ***10*** раз, число сравнений увеличивается в ***99*** раз, число перемещений в увеличивается в ***103*** раза.

Как мы видим, ассимтотическая оценка алгоритмов подтверждается на практике.

Структура программы и спецификации функций

В моей программе используются средующие функции:

- `int cmp(int a, int b, int type_sort)`
  Компаратор. Возвращает ***1***, при ***a*** < ***b*** и ***0*** в противном случае.
  Помимо переменных ***a*** и ***b*** принимает переменную ***type_sort***, которая отвечает за тип сортировки(Быстрая или Пузырьковая), которая вызвала функцию сравнения.

- `void swap(int *a, int *b, int type_sort)`
  Меняет ячейки, на которые указывают ***a*** и ***b***, местами.
  Помимо указателей ***a*** и ***b*** принимает переменную ***type_sort***, которая отвечает за тип сортировки(Быстрая или Пузырьковая), которая вызвала функцию обмена.

- **void** Qsort(**int** *a, **int** n)
  Сортирует массив из ***n*** элементов, на первый элемент которого указывает ***a***, методом *быстрой* сортировки

- **void** Bsort(**int** *a, **int** n)
  Сортирует массив из ***n*** элементов, на первый элемент которого указывает ***a***, методом *пузырьковой* сортировки

- **void** gen_ascend(**int** *a, **int** n)
  Заполняет массив из ***n*** элементов, на первый элемент которого указывает ***a***, возрастающей последовательностью целых чисел. Память под ***a*** должна быть выделена ДО вызова.

- **void** gen_decreas(**int** *a, **int** n)
  Заполняет массив из ***n*** элементов, на первый элемент которого указывает ***a***, убывающей последовательностью целых чисел. Память под ***a*** должна быть выделена ДО вызова.

- **void** gen_decreas(**int** *a, **int** n)
  Заполняет массив из ***n*** элементов, на первый элемент которого указывает ***a***, убывающей последовательностью целых чисел. Память под ***a*** должна быть выделена ДО вызова.

- **void** gen_rand{3,4}(**int** *a, **int** n)
  Заполняет массив из ***n*** элементов, на первый элемент которого указывает ***a***, произвольной последовательностью целых чисел. Память под ***a*** должна быть выделена ДО вызова.

- **int** check(**int** *a, **int** n)
  Проверяет массив из ***n*** элементов, на первый элемент которого указывает ***a***, на неубывание. Функция написана для тестирования и проверки корректности работы сортировок.

- **void** print_inf(**int** type_ar, **int** type_sort)
  1. Принимает на вход два числа: ***type_ar*** и ***type_sort.***
  2. Генерирует массив, тип которого определяется переменной ***type_ar***(возрастание/убывание/рандом).
  3. Сортирует его функцией, определенной переменной ***type_sort***(Быстрая и Пузырьковая).
  4. Выводит всю необходимую информацию: число сравнений, число перестановок и размер массива, для которого все выполнялось.

  При этом шаги 2, 3 и 4 проделываются последовательно для ***N*** = 10, 100, 1000, 10000, где ***N*** — размер массива.

- **void** (*gen_array[4]) (**int** *, **int**)
  Массив указателей на функции, генерирующие массивы.

- **void** (*sort[2]) (**int** *, **int**)
  Массив указателей на сортировки.

- **void** init_func()
  Инициализирует массивы указателей на функции.

- **void** print_ar (**int** *a, **int** n)
  Выводит массив из ***n*** элементов, на первый элемент которого указывает ***a***.

- **int** main()
  1. Вызываеся функция ***init_func()***
  2. Перебираются всевозможные комбинации из типов сортировок и типов массивов и на них вызывается функция ***printf_inf(…)***.

Отладка программы, тестирование функций

В связи с тем, что весь алгоритм был разбит на достаточно маленькие составляющие, представленные в виде функций, отладка программы не составила мне труда. При отладке той или иной части кода, я просто комментировал вызовы ненужных мне частей и оставлял лишь интересующие меня разделы.

Тестирование функций проводилось на сгенерированных случайным/упорядоченным образом наборах данных, а проверка корректности проводилась за счет специально написанных для этого функций. Так, например, в коде задействована функция ***check***, проверяющая правильность работы сортировок.

Анализ допущенных ошибок

Нынешняя версия программы сильно отличается от первоначальной. Повторяющиеся части кода я заменял на функции, и даже их мне иногда приходилось объединять в одну функцию. Для этого пришлось разобраться в указателях на функции. И это дало свои плюсы.
Во-первых, искать ошибки и исправлять их стало гораздо легче.
Во-вторых, я все-таки применил возможность языка Си, до этого казавшуюся мне ненужной.
В-третьих, код стал выглядеть по-человечески.

Также не с первого раза была написана сортировка ***Qsort***, не уделил достаточно внимания инвариантам. Хотя функция и работала корректно на маленьких тестах, ошибка была установлена функцией ***check*** на достаточно больших размерах данных(порядка 1000 и 10000). Причем далеко не на ранней стадии разработки.

Вывод

1. Функции надо тщательно тестировать в процессе, а не под конец работы над всем проектом.

2. Необходимо писать красивый, лаконичный и понятный код с комментариями.
Это облегчит поиск ошибок и работу товарищам, которые будут вынуждены разобраться в исходниках.

Литература

1. Трифанов Н. П., Пильщиков В. Н. Задания практикума на ЭВМ (1 курс) Методическая разработка (составители). —М.: ВМК МГУ, 2001.
2. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.Второе издание. —М.: «Вильямс», 2005
3. Кнут Д. Искусство программирования для ЭВМ. Том 3. М.: Мир, 1978.
4. Лорин Г. Сортировка и системы сортировки. — М.: Наука, 1983
5. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989