

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
Национальный исследовательский Нижегородский государственный университет им.  
Н.И. Лобачевского

Институт информационных технологий, математики и механики

## **Отчет по лабораторной работе**

### **«Сортировки»**

**Выполнил:**

студент группы 382003-1  
Маслов А.Е.

**Проверил:**

ассистент каф. МОСТ,  
Волокитин В.Д.

Нижний Новгород  
2020

# Содержание

Постановка задачи .....	3
Метод решения.....	4
Руководство пользователя.....	5
Описание программной реализации.....	6
Подтверждение корректности .....	7
Результаты экспериментов.....	8
Заключение .....	8
Приложение .....	10

## **Постановка задачи**

Реализовать алгоритмы сортировки: выбором, Шелла, слиянием и поразрядной сортировки для чисел типа float. Также необходимо посчитать количество сравнений и перестановок элементов в каждой сортировке. Необходимо проверить работу алгоритмов при различных состояниях его элементов (лучший, средний и худший случаи) так, чтобы показать их теоретическую сложность в каждом случае.

## Метод решения

1. Сортировка выбором: находится максимальный элемент массива и меняется местами с последним элементом (для сортировки по возрастанию). Далее происходит тоже самое, только массив на каждой следующей итерации рассматривается без последнего элемента. После последней итерации получим отсортированный массив.  
Худшее время:  $O(n^2)$ , Лучшее время:  $O(n^2)$ , Среднее время:  $O(n^2)$ , Затраты памяти:  $O(n)$  всего,  $O(1)$  дополнительно.
2. Сортировка Шелла: выбирается шаг равный половине размера массива, элементы с одинаковыми остатками от деления индекса на шаг образуют одну группу, в этой группе они сортируются сортировкой вставками, первая итерация окончена. На каждой следующей итерации шаг уменьшается вдвое и происходит тоже самое, что и на первой итерации. Когда шаг станет равен единице, реализуется обычная сортировка вставками и алгоритм завершается.  
Худшее время:  $O(n^2)$ , Лучшее время:  $O(n \cdot \log^2(n))$ , Среднее время: зависит от выбранных шагов, Затраты памяти:  $O(n)$  всего,  $O(1)$  дополнительно.
3. Сортировка слиянием: в функцию слияния подается указатель на массив и его длина (половины массива должны быть отсортированы), функция слияния перезаписывает массив так, что он получается отсортированным. Функция сортировки вызывает сама себя для левой и правой части массива, пока их размеры больше 1, затем вызывает функцию слияния для этих частей, которые она отсортировала в рекурсии.  
Худшее время:  $O(n \cdot \log(n))$ , Лучшее время:  $O(n \cdot \log(n))$ , Среднее время:  $O(n \cdot \log(n))$ , Затраты памяти:  $O(n)$  вспомогательных.
4. Поразрядная сортировка: числа в массиве рассматриваются по разрядам, начиная с наименьшего. Числа сортируются устойчивой сортировкой по цифрам, стоящим в рассматриваемом разряде. На следующей итерации рассматриваемый разряд увеличивается и происходит тоже самое, что и на прошлой итерации. Когда рассмотрен максимальный разряд сортировка завершена.  
Худшее время:  $O(n \cdot k)$ , Лучшее время:  $O(n)$ , Затраты памяти:  $O(n+k)$ .

## **Руководство пользователя**

Пользователь должен ввести число элементов массива, затем его элементы, выбрать сортировку. В консоль будет выведен несортированный массив и отсортированный массив, число перестановок и сравнений.

## Описание программной реализации

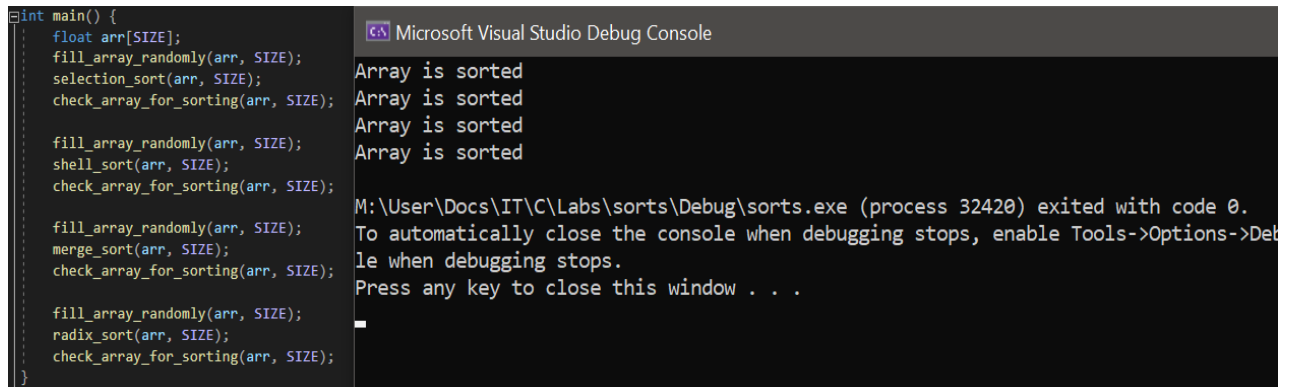
Один файл main.c содержащий весь код.

Описание функций:

- `fill_array_randomly` – принимает первым аргументом указатель на массив, вторым размер этого массива, заполняет массив псевдослучайными числами.
- `show_array` - принимает первым аргументом указатель на массив, вторым размер этого массива, выводит в консоль все элементы массива.
- `check_array_for_sorting` - принимает первым аргументом указатель на массив, вторым размер этого массива, проверяет массив на сортировку.
- `find_max_index` - принимает первым аргументом указатель на массив, вторым размер этого массива, возвращает индекс максимального элемента.
- `swap` – принимает указатели на переменные, значения которых надо поменять местами и меняет их.
- `merge` - принимает первым аргументом указатель на массив (половины массива должны быть отсортированы), вторым размер этого массива, функция перезаписывает массив так, что он получается отсортированным. Является вспомогательной функцией.
- `create_counters` - принимает первым аргументом указатель на массив, вторым размер этого массива, вспомогательная функция для подсчёта значений каждого байта у элементов.
- `radix_pass` - принимает первым аргументом номер байта от 0 до 3, длину массива, третьим исходный массив, четвертым дополнительный массив, пятым указатель на элемент массива подсчёта, начиная с которого будет происходить сортировка по текущему разряду, вспомогательная функция для сортировки одного разряда.
- `input_array` - принимает первым аргументом указатель на массив, вторым размер этого массива, считывает числа и записывает их в массив.
- `selection_sort` - принимает первым аргументом указатель на массив, вторым размер этого массива, сортирует массив алгоритмом выбора.
- `shell_sort` - принимает первым аргументом указатель на массив, вторым размер этого массива, сортирует массив алгоритмом Шелла.
- `merge_sort` - принимает первым аргументом указатель на массив, вторым размер этого массива, сортирует массив алгоритмом слияния.
- `radix_sort` - принимает первым аргументом указатель на массив, вторым размер этого массива, сортирует массив алгоритмом по разрядной сортировки.

## Подтверждение корректности

Для подтверждения корректности в программе я заполнял массив на 100000 элементов “случайными” числами и сортировал его сортировками, каждый раз я видел следующие:



```
int main() {  
    float arr[SIZE];  
    fill_array_randomly(arr, SIZE);  
    selection_sort(arr, SIZE);  
    check_array_for_sorting(arr, SIZE);  
  
    fill_array_randomly(arr, SIZE);  
    shell_sort(arr, SIZE);  
    check_array_for_sorting(arr, SIZE);  
  
    fill_array_randomly(arr, SIZE);  
    merge_sort(arr, SIZE);  
    check_array_for_sorting(arr, SIZE);  
  
    fill_array_randomly(arr, SIZE);  
    radix_sort(arr, SIZE);  
    check_array_for_sorting(arr, SIZE);  
}
```

Microsoft Visual Studio Debug Console

Array is sorted  
Array is sorted  
Array is sorted  
Array is sorted

M:\User\Docs\IT\C\Labs\sorts\Debug\sorts.exe (process 32420) exited with code 0.  
To automatically close the console when debugging stops, enable Tools->Options->Debu  
le when debugging stops.  
Press any key to close this window . . .

## Результаты экспериментов

### 1) Сортировка выбором:

Сложность алгоритма: лучший случай –  $O(n^2)$ , средний и худший случаи –  $O(n^2)$ .

Количество элементов	50	100	1000	2000	5000
Лучший случай	49	99	999	1999	4999
Лучший случай	0	0	0	0	0
Средний случай	579	2604	254926	989021	6239697
Средний случай	534	2511	253931	987030	6234707
Худший случай	1225	4950	499500	1999000	12497500
Худший случай	1225	4950	499500	1999000	12497500

По таблице видно, что в лучшем случае количество сравнений растёт квадратично ( $O(n^2)$ ), в среднем и худшем случаях – квадратично ( $O(n^2)$ ), что согласуется с теоретическими данными.

### 2) Сортировка Шелла:

Сложность алгоритма: лучший и средний случаи –  $O(n \cdot \log^2(n))$ , худший случай –  $O(n^2)$ .

Количество элементов	50	100	1000	10000	100000
Лучший случай	237	573	8977	123617	1568929
Лучший случай	0	0	0	0	0
Средний случай	316	611	11660	137736	1662906
Средний случай	61	148	2278	31258	412637

По таблицам можно сделать вывод, что результаты эксперимента совпадают с теоретической оценкой  $O(n \cdot \log^2(n))$ .

### 3) Сортировка слиянием:

Сложность алгоритма:  $O(n \cdot \log(n))$  во всех случаях.

Количество элементов	50	100	1000	2000	5000
Лучший случай	133	316	4932	10864	29804
Лучший случай	286	672	9976	21952	61808
Средний случай	219	542	8678	19401	55243
Средний случай	286	672	9976	21952	61808

Таблицы подтверждают теоретическую сложность сортировки  $O(n \cdot \log(n))$ .

### 4) Поразрядная сортировка:

Сложность алгоритма:  $O(n)$  во всех случаях (считаются перестановки элементов массива).

Количество элементов	50	100	1000	10000	100000
Количество сравнений	0	0	0	0	0
Количество перестановок	250	500	5000	50000	500000

Нетрудно заметить, что число перестановок растёт линейно относительно  $n$ , что совпадает с теоретической сложностью.



## **Заключение**

Сортировки готовы к применению на других проектах.

## Приложение

```
void selection_sort(float array[], int size)
{
    for (int last_unsorted_index = size - 1; last_unsorted_index > 0; last_unsorted_index--)
    {
        int max_elem_index = find_max_index(array, last_unsorted_index);
        swap(&array[max_elem_index], &array[last_unsorted_index]);
    }
};

void shell_sort(float array[], int size)
{
    int i, j, step;
    float temporary;
    for (step = size / 2; step > 0; step = step / 2)
    {
        for (i = step; i < size; i++)
        {
            temporary = array[i];
            for (j = i; j >= step; j = j - step)
            {
                if (temporary < array[j - step])
                {
                    array[j] = array[j - step];
                }
                else
                {
                    break;
                }
            }
            array[j] = temporary;
        }
    }
}

void merge_sort(float array[], int size)
{
    float* left_array_part = array;
    float* right_array_part = &array[size / 2];
    int left_array_part_size = size / 2;
    int right_array_part_size = size - (size / 2);
    if (size < 2)
    {
        return;
    }
    merge_sort(left_array_part, left_array_part_size);
    merge_sort(right_array_part, right_array_part_size);
    merge(left_array_part, size);
}

void radix_sort(float* array, int size)
{
    float* temporary_array = (float*)malloc(size * sizeof(float));
    int* counters = (int*)malloc(1024 * sizeof(int));
    int* count;
    int k = 0;
    create_counters(array, counters, size);
    for (unsigned short i = 0; i < sizeof(float); i++)
    {
        count = counters + 256 * i;
        radix_pass(i, size, array, temporary_array, count);
        for (int j = 0; j < size; j++)
        {
            array[j] = temporary_array[j];
        }
    }
    while (array[k] >= 0 && k < size && !(k > 0 && array[k] <= 0 && array[k - 1] > 0)) k++;
    for (int i = 0; i < size - k; i++)
    {
        array[i] = temporary_array[size - 1 - i];
    }
    for (int i = 0; i < k; i++)
    {
        array[size - k + 1 + i] = temporary_array[i];
    }
    memcpy(array, temporary_array, size * sizeof(float));
    free(temporary_array);
    free(counters);
}
```