

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 .....	9
Приложение .....	10

## **Постановка задачи**

Реализовать алгоритмы сортировки: выбором, Шелла, слиянием и поразрядной сортировки для чисел типа float.

## 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.
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.
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.
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.

## **Руководство пользователя**

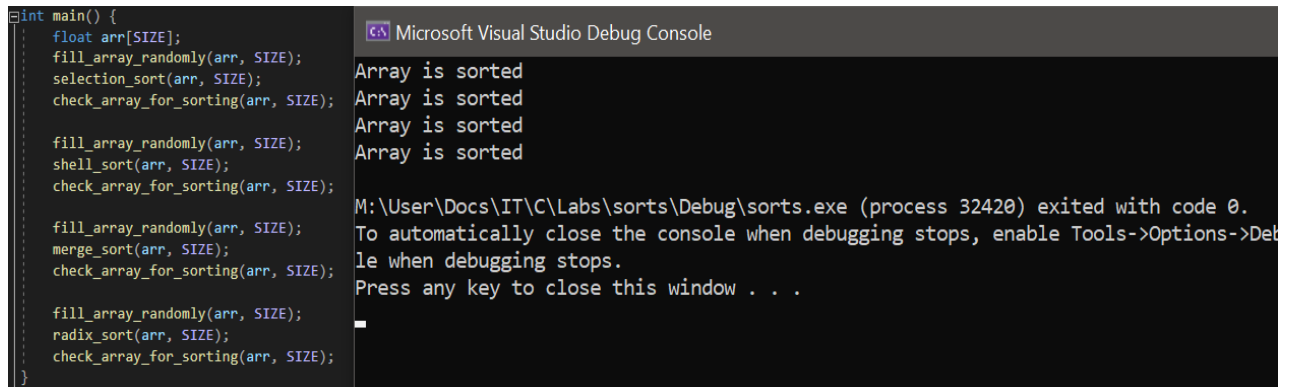
Пользователь должен ввести число элементов массива, затем его элементы, выбрать сортировку. В консоль будет выведен несортированный массив и отсортированный массив, число перестановок и сравнений.

## **Описание программной реализации**

Один файл main.c содержащий весь код.

## 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**

По данным экспериментов видно, что сортировки работают за разумное время на моем ноутбуке.



## **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);
}
```