

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
Инженерно-физический факультет  
Кафедра автоматизированных систем обработки информации и  
управления

ОТЧЕТ ПО ПРАКТИКЕ

Программная реализация численного метода  
*Сортировки Быстрая и Слиянием. (вариант 4)*

2 курс, группа ИВТ АСОИУ

Выполнил:

\_\_\_\_\_ А. Е. Колесник  
«\_\_\_» \_\_\_\_\_ 2025 г.

Руководитель:

\_\_\_\_\_ С. В. Теплоухов  
«\_\_\_» \_\_\_\_\_ 2025 г.

Майкоп, 2025 г.

# 1. Введение.

## 1.1. Задание

Сортировки Быстрая и Слиянием.

## 1.2. Теория: быстрая сортировка

"Быстрая сортировка" хоть и была разработана более 40 лет назад, является наиболее широко применяемым и одним из самых эффективных алгоритмов. Метод основан на подходе "разделяй-и-властвуй". Общая схема такова:

- из массива выбирается некоторый опорный элемент  $a[i]$ ,
- запускается процедура разделения массива, которая перемещает все ключи, меньшие, либо равные  $a[i]$ , влево от него, а все ключи, большие, либо равные  $a[i]$  - вправо,
- теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правому,
- для обоих подмассивов: если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру. В конце получится полностью отсортированная последовательность.

Разделение массива:

На входе массив  $a[0] \dots a[N]$  и опорный элемент  $p$ , по которому будет производиться разделение.

1. Введем два указателя:  $i$  и  $j$ . В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.
2. Будем двигать указатель  $i$  с шагом в 1 элемент по направлению к концу массива, пока не будет найден элемент  $a[i] \geq p$ . Затем аналогичным образом начнем двигать указатель  $j$  от конца массива к началу, пока не будет найден элемент  $a[j] \leq p$ .
3. Далее, если  $i \leq j$ , меняем  $a[i]$  и  $a[j]$  местами и продолжаем двигать  $i, j$  по тем же правилам...
4. Повторяем шаг 3, пока  $i \leq j$ .

Рассмотрим работу процедуры для массива  $a[0] \dots a[6]$  и опорного элемента  $p = a[3]$ .



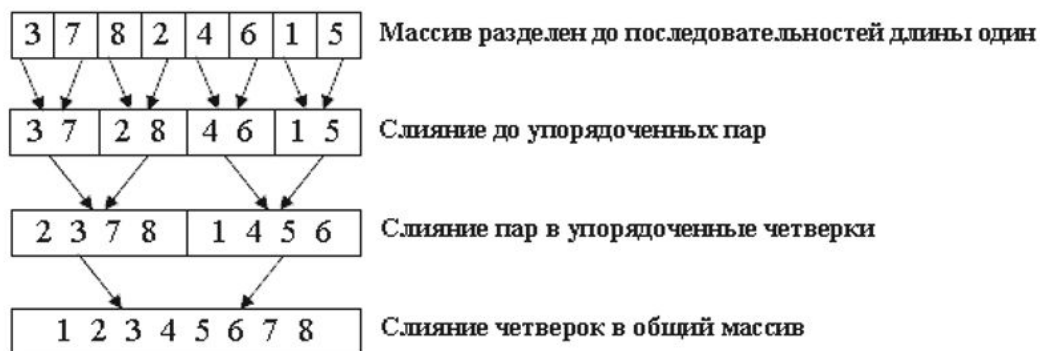
Теперь массив разделен на две части: все элементы левой меньше либо равны  $p$ , все элементы правой - больше, либо равны  $p$ . Разделение завершено.

### 1.3. Теория: сортировка слиянием

Сортировка слиянием также построена на принципе "разделяй-и-властвуй" однако реализует его несколько по-другому, нежели quickSort. А именно, вместо разделения по опорному элементу массив просто делится пополам.

Функция merge на месте двух упорядоченных массивов  $a[lb] \dots a[split]$  и  $a[split+1] \dots a[ub]$  создает единый упорядоченный массив  $a[lb] \dots a[ub]$ .

Пример работы алгоритма на массиве 3 7 8 2 4 6 1 5..



Рекурсивный алгоритм обходит получившееся дерево слияния в прямом порядке. Каждый уровень представляет собой проход сортировки слияния - операцию, полностью переписывающую массив.

Обратим внимание, что деление происходит до массива из единственного элемента. Такой массив можно считать упорядоченным, а значит, задача сводится к написанию функции слияния merge.

Один из способов состоит в слиянии двух упорядоченных последовательностей при помощи вспомогательного буфера, равного по размеру общему количеству имеющихся в них элементов. Элементы последовательностей будут перемещаться в этот буфер по одному за шаг.



## 2. Ход работы

### 2.1. Код приложения быстрой сортировки

```
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    setlocale(LC_ALL, "RU");
    vector<int> arr = {};
    int n, a;
    cout << "Заполнение массива: \n";
    cout << "Ввод количества значений: \n";
    cin >> n;
    cout << "Ввод значений (целые числа от -100 до 100): \n";
    for (int i = 0; i < n; i++)
    {
        cout << ": ";
        cin >> a;
        if (not(-100 <= a and a <= 100))
```

```

        {
            cout << "Введите целое число от -100 до 100\n";
            i--;
        }
        else arr.push_back(a);
    }

    cout << "Исходный массив: ";
    for (int num : arr) cout << num << " ";
    cout << endl;
    quickSort(arr, 0, arr.size() - 1);
    cout << "Отсортированный массив: ";
    for (int num : arr) cout << num << " ";
    cout << endl;
    return 0;
}

```

## 2.2. Код приложения сортировки слиянием

```

#include <iostream>
#include <vector>
#include <math.h>
using namespace std;

void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0;
    int j = 0;
    int k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
    }
}

```

```

        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    setlocale(LC_ALL, "RU");
    vector<int> arr = {};
    int n, a;
    cout << "Заполнение массива: \n";
    cout << "Ввод количества значений: \n";
    cin >> n;
    cout << "Ввод значений (целые числа от -100 до 100): \n";
    for (int i = 0; i < n; i++)
    {
        cout << ": ";
        cin >> a;
        if (not(-100 <= a and a <= 100))

```

```

        {
            cout << "Введите целое число от -100 до 100\n";
            i--;
        }
        else arr.push_back(a);
    }

    cout << "Исходный массив: ";
    for (int num : arr) cout << num << " ";
    cout << endl;

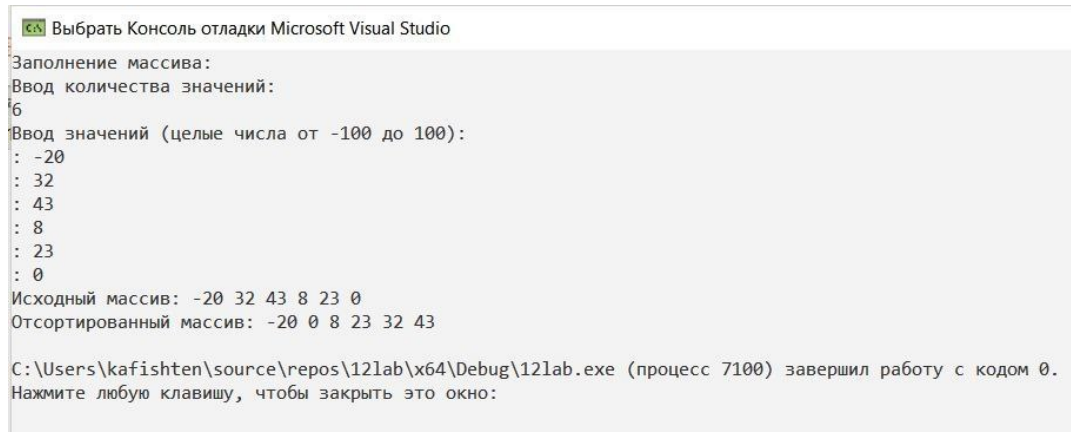
    mergeSort(arr, 0, arr.size() - 1);

    cout << "Отсортированный массив: ";
    for (int num : arr) cout << num << " ";
    cout << endl;
    return 0;
}

```

### 3. Примеры работы программ:

#### 3.1. Быстрая сортировка:



```

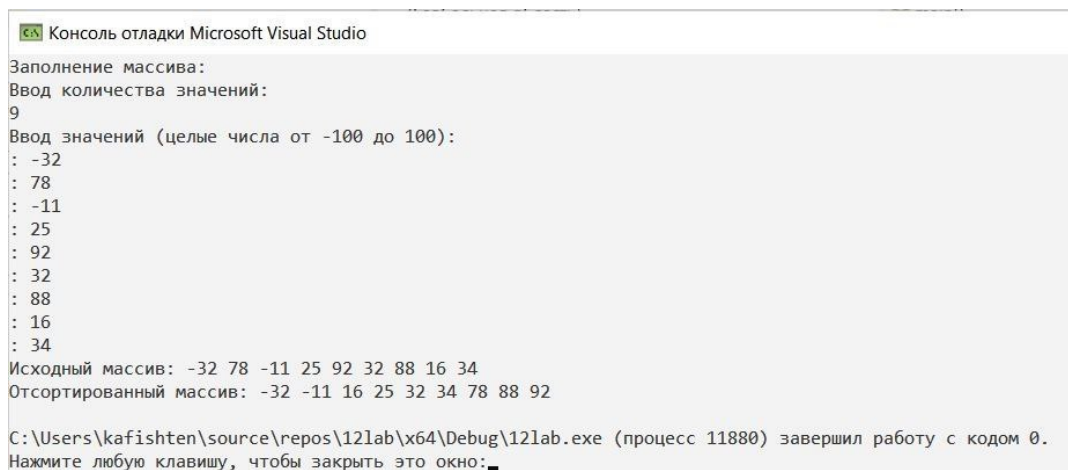
Выбрать Консоль отладки Microsoft Visual Studio
Заполнение массива:
Ввод количества значений:
6
Ввод значений (целые числа от -100 до 100):
: -20
: 32
: 43
: 8
: 23
: 0
Исходный массив: -20 32 43 8 23 0
Отсортированный массив: -20 0 8 23 32 43

C:\Users\kafishten\source\repos\12lab\x64\Debug\12lab.exe (процесс 7100) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:

```

Рис. 1. Пример работы программы быстрой сортировки с ручным вводом значений

## 3.2. Сортировка слиянием:



```
Консоль отладки Microsoft Visual Studio
Заполнение массива:
Ввод количества значений:
9
Ввод значений (целые числа от -100 до 100):
: -32
: 78
: -11
: 25
: 92
: 32
: 88
: 16
: 34
Исходный массив: -32 78 -11 25 92 32 88 16 34
Отсортированный массив: -32 -11 16 25 32 34 78 88 92

C:\Users\kafishten\source\repos\12lab\x64\Debug\12lab.exe (процесс 11880) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:■
```

Рис. 2. Пример работы программы сортировки слиянием с ручным вводом значений.

## Список литературы

- [1] Львовский С.М. Набор и верстка в системе  $\text{\LaTeX}$ . — 3-е издание, исправленное и дополненное, 2003 г.
- [2] Воронцов К.В.  $\text{\LaTeX}$  в примерах. 2005 г.
- [3] GitHub: <https://github.com/Kafishten/praktika2>