

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Сортировки**

Студент гр. 9382

\_\_\_\_\_

Михайлов Д.А.

Преподаватель

\_\_\_\_\_

Камалеев Р.М.

Санкт-Петербург

2020

### **Цель работы.**

Ознакомиться с основными характеристиками и особенностями типов данных стек и очередь, изучить особенности их реализации на языке программирования C++. Разработать программу, использующую иерархические списки и их рекурсивную обработку, высчитывающую значение выражения.

### **Задание (вариант 32)**

«Болты и гайки». Имеется куча перемешанных  $n$  болтов и  $n$  гаек, отличающихся диаметрами. Нужно быстро найти все соответствующие пары болтов и гаек. Известно, что каждая гайка подходит по диаметру ровно к одному болту, и наоборот. Нет ни двух болтов одинакового диаметра, ни двух гаек одинакового диаметра.

Попробовав навинтить гайку на болт, можно определить, что из них больше (или они соответствуют друг другу), но невозможно (а в алгоритме – нельзя) непосредственно сравнить два болта или две гайки.

Простой алгоритм –  $O(n^2)$  . Это не быстро. Предложить аналограндомизированной быстрой сортировки –  $O(n \log n)$ .

### **Основные теоретические положения.**

Одной из быстрых сортировок, имеющих среднюю сложность  $O(n \log n)$ , является алгоритм быстрой сортировки QuickSort.

Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

### **Описание алгоритма нахождения пар**

Алгоритм представляет собой сортировку двух массивов таким образом, чтобы массивы были одинаковы, то есть элементы с одинаковым индексом образовывали пару в условиях задачи. Алгоритм основан на базе быстрой сортировки QuickSort. В первом массиве фиксируется опорный элемент, который сравнивается с элементами другого массива. В итоге массив разделяется на элементы, меньшие опорного, большие опорного, а равный опорному помещается в конце массива. Далее проводится аналогичная операция, но опорный элемент берется последним из второго массива, то есть равный опорному элементу из первого, и сравнивается с элементами первого массива. В итоге оба массива поделены на пару найденных равных элементов, на элементы, меньшие этой пары, и большие. Алгоритм рекурсивно запускается для последних двух частей массива.

В алгоритме предусмотрен вывод промежуточных результатов, в том числе пошаговый режим, при котором выводится информация о всех сравнениях и перестановках, совершенных алгоритмом. Полный список функций, реализующих визуализацию алгоритма и некоторые его функции, приведен в табл. 1.

Функция	Назначение
<code>int strCount(string &amp;in)</code>	Считает количество чисел, разделенных пробелом в строке, а также проверяет корректность строки
<code>int checkStr(string&amp;bolts, string &amp;nuts)</code>	Проверяет совпадение строк по

	количеству чисел в них, а также обрабатывает ошибки на корректность строк, возвращает кол-во чисел
<code>dArr makeArray(string &amp;in, int l)</code>	Создает динамический массив чисел из строки
<code>string getStrFromArray(dArr &amp;input)</code>	Возвращает строку, состоящую из элементов массива, разделенных пробелом
<code>string getStrCmpAction(dArr &amp;bolts, int ptr1, dArr &amp;nuts, int ptr2)</code>	Возвращает строку, состоящую из двух массивов, в которых два элемента выделены фигурными скобками.
<code>string getStrSwapAction (dArr &amp;input, int ptr1,int ptr2)</code>	Возвращает строку, состоящую из элементов массива, среди которых два элемента выделены фигурными скобками
<code>int checkPairs(dArr &amp;bolts, dArr&amp;nuts)</code>	Проверяет, совпадают ли в отсортированных массивах элементы с одинаковыми индексами, то есть проверяет пары
<code>string getPairs(dArr &amp;bolts, dArr&amp;nuts)</code>	Возвращает строку, в которой содержатся все пары в формате $(b_i, n_i)$ , где $b_i$ и $n_i$ – элементы двух массивов с индексом $i$

### Оценка сложности алгоритма

В лучшем случае, когда при каждом разбиении оба массива будут разбиваться пополам, глубина рекурсии  $N \approx \log_2 n$ , где  $n$  – длина массивов. Во время работы функция проводит  $K_i = 2n + 2K_{i+1}$  сравнений: сначала опорный элемент одного массива с элементами другого, затем наоборот, после чего

функция рекурсивно вызывается для двух сегментов массивов. Тогда сложность алгоритма составит  $O(n * \log 2 n)$ .

В среднем случае, когда разбиения массивов будут неравноценными, глубина рекурсии  $N \approx \log c n$ , где  $c$  – некоторая константа. В конечном счете, это приводит к сложности алгоритма  $O(n * \log n)$ . Так как в функции проводится  $2n$  сравнений, а также рекурсивно функция вызывается два раза, то фактически число итераций можно оценить как  $4n * \log n$ .

В худшем случае, когда разбиения массивов будет таковым, что все элементы либо меньше, либо больше опорного, глубина рекурсии будет  $N \approx n$ . В таком случае, сложность алгоритма возрастает до  $O(n^2)$ . Рост занимаемой памяти напрямую зависит от глубины рекурсии, при этом в функции не создаются копии массивов, соответственно, сложность алгоритма по памяти в среднем случае  $O(\log n)$ , в худшем –  $O(n)$ .

Был проведен ряд тестов программы с фиксацией числа итераций во время работы алгоритма. Входные данные были таковы, чтобы избежать худшего случая. Для оценки теоретического числа итераций  $N$  для  $n$  элементов в массиве использовалась функция  $f(n) = 4n * \log n$ . График зависимости числа итераций  $N$  от количества элементов  $n$  и график функции  $f(n)$  изображены на рис.1

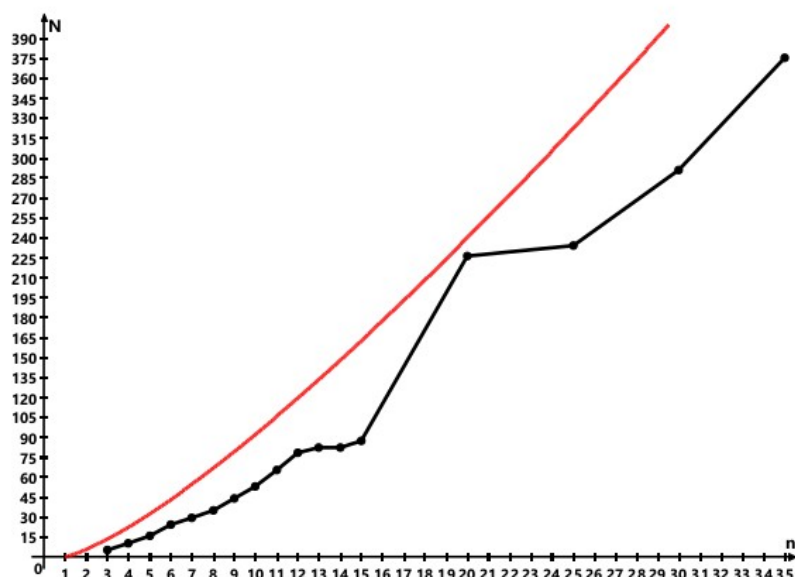


Рисунок 1 – График зависимости числа итераций от длины массивов

Из графиков видно, что практическое число итераций N растет так же, как функция  $f(n) = 4n * \log n$ . График числа итераций имеет скачкообразную форму, связанную с тем, что алгоритм имеет нестабильную сложность.

### Тестирование программы.

Входные массивы	Результат выполнения (сокращ.)
Болты: 2 56 38 4 8 6 18 72 10 Гайки: 10 4 8 56 38 6 72 18 2	Количество элементов в массивах гаек и болтов: 9 Глубина рекурсии: 20 Сортировка завершена Болты: 4 10 8 6 38 56 72 18 2 Гайки: 4 10 8 6 38 56 72 18 2 Найденные пары болтов и гаек: (4, 4) (10, 10) (8, 8) (6, 6) (38, 38) (56, 56) (72, 72) (18, 18) (2, 2)
Болты: 2 56 38 4 8 98 18 72 10	Болты: 4 10 8 56 38 98 72 18 2

Гайки: 10 4 8 56 38 6 72 18 2	Гайки: 4 10 8 6 38 56 72 18 2 Обнаружены непарные болты и гайки
Болты: 2 56 38 4 8 6 18 72 Гайки: 10 4 8 56 38 6 72 18 2	Ошибка! Количество болтов и гаек не совпадает
Болты: 2 56 38 4 10 8 6 18 72 10 Гайки: 10 4 8 56 38 6 72 10 18 2	Болты: 6 4 10 8 10 56 38 72 18 2 Гайки: 10 4 8 6 10 56 72 38 18 2 Ошибка! Обнаружены одинаковые болты Сортировка прервана
Болты: 2 56 38 4 -8 6 18 72 10 Гайки: 10 4 8 56 38 6 72 18 2	Ошибка! Болты могут быть заданы только цифрами

### **Выводы.**

В ходе выполнения лабораторной работы была написана программа, сортирующая два массива по парам, используя сравнения только элементов разных массивов. Был реализован быстрый алгоритм, основанный на сортировке QuickSort, имеющий среднюю сложность  $O(n * \log n)$ . Были реализованы пошаговый режим выполнения алгоритма, консольный и графический интерфейсы, обработка ошибок.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.C

```
#include <iostream>
#include<vector>
#include<string>
#include <fstream>
#include <sstream>

#define ERR_SYMBOL_B -1
#define ERR_SYMBOL_N -2
#define ERR_LENGTH -3

using namespace std;

struct dArr
{
    int length;
    int *base;
};

/**
 * Вычисляет количество элементов (чисел), разделенных пробелом, и
 * проверяет корректность формата
 * @param input - входная строка
 */
int strCount(string &input) {
    unsigned int count = 0;
    bool num = false;
    if (input.empty())
        return 0;
    for (char i : input) {
        if (i == ' ') {
            if (num) {
                count++;
                num = false;
            }
        } else {
            if (!isdigit(i))
                return ERR_SYMBOL_B;
            num = true;
        }
    }
    if (num)
        count++;
    return static_cast<int>(count);
}
```



```

/**
 * Проверяет совпадение строк по количеству чисел в них, а также
 * проверяют корректность входных данных
 * Возвращает количество чисел
 * @param bolts болты
 * @param nuts гайки
 */
int checkStr(string &bolts, string &nuts) {
    int nB = strCount(bolts);
    if (nB == -1)
        return ERR_SYMBOL_B;
    int nN = strCount(nuts);
    if (nN == -1)
        return ERR_SYMBOL_N;
    if (nN == nB)
        return nN;
    return ERR_LENGTH;
}

/**
 * Создает динамический массив чисел из принятой строки
 * @param input входная строка
 * @param length длина массива
 */
dArr makeArray(string &input, int length) {
    dArr output{};
    output.base = new int[length];
    output.length = length;
    stringstream temp;
    temp << input;
    for (int i = 0; i < length; i++) {
        temp >> output.base[i];
    }
    return output;
}

/**
 * Возвращает строку, состоящую из элементов массива, разделенных
 * пробелом
 * @param input
 */
string getStrFromArray(dArr &input) {
    string output;
    for (int i = 0; i < input.length; i++) {
        output += to_string(input.base[i]);
        output += " ";
    }
    return output;
}

```

```

/**
 * Возвращает строку, состоящую из двух массивов, в которых два элемента
выделены фигурными скобками
 * @param bolts массив болтов
 * @param ptr1 позиция в массиве болтов
 * @param nuts массив гаек
 * @param ptr2 позиция в массиве гаек
 */
string getStrCmpAction(dArr &bolts, int ptr1, dArr &nuts, int ptr2) {
    string output;

    output +=
    "-----\n";
    output += "Сравниваем элементы:\nБолты:\n";
    for (int i = 0; i < bolts.length; i++) {
        if (i == ptr1)
            output += "{";
        output += to_string(bolts.base[i]);
        if (i == ptr1)
            output += "}";
        output += " ";
    }
    output += "\nГайки:\n";
    for (int i = 0; i < nuts.length; i++) {
        if (i == ptr2)
            output += "{";
        output += to_string(nuts.base[i]);
        if (i == ptr2)
            output += "}";
        output += " ";
    }
    return output;
}

/**
 * Возвращает строку, состоящую из элементов массива, среди которых два
элемента выделены фигурными скобками
 * @param input массив элементов
 * @param ptr1 позиция первого элемента
 * @param ptr2 позиция второго элемента
 */
string getStrSwapAction(dArr &input, int ptr1, int ptr2) {
    string output;
    output += "Меняем элементы местами: ";
    for (int i = 0; i < input.length; i++) {
        if (i == ptr1 || i == ptr2)
            output += "{";
        output += to_string(input.base[i]);
    }
}

```

```

        if (i == ptr1 || i == ptr2)
            output += "}";
        output += " ";
    }
    output += "\n";
    return output;
}

/**
 * Проверяет, совпадают ли в отсортированных массивах элементы с
 * одинаковыми индексами (проверка пар)
 * @param bolts массив болтов
 * @param nuts массив гаек
 */
int checkPairs(dArr &bolts, dArr &nuts) {
    for (int i = 0; i < bolts.length; i++) {
        if (bolts.base[i] != nuts.base[i])
            return 0;
    }
    return 1;
}

/**
 * Возвращает строку, в которой содержатся все пары в формате (Bi , Ni),
 * где Bi и Ni – элементы двух массивов с индексом i
 * @param bolts массив болтов
 * @param nuts массив гаек
 */
string getPairs(dArr &bolts, dArr &nuts) {
    string output;
    for (int i = 0; i < bolts.length; i++) {
        if (bolts.base[i] == nuts.base[i]) {
            output += "(";
            output += to_string(bolts.base[i]);
            output += ", ";
            output += to_string(nuts.base[i]);
            output += ") ";
        }
    }
    return output;
}

/**
 * Сортирует массивы на основе алгоритма QuickSort
 * @param bolts массив болтов
 * @param nuts массив гаек
 * @param start начальная позиция
 * @param length длина массива
 * @param n количество рекурсий

```

```

*/
int qsortNB(dArr &bolts, dArr &nuts, int start, int length, int &n) {
    int ret = 0;
    string output;
    bool equiv = false;
    int bigger = start;
    if (length - start > 1) {
        n++;
        int center = nuts.base[length - 1];
        int i = start;
        while (i < length - 1) {
            output = getStrCmpAction(bolts, i, nuts, length - 1);
            if (bolts.base[i] < center) {
                output += "Болт меньше, перемещаем в левую часть\n";
                output += getStrSwapAction(bolts, i, bigger);
                swap(bolts.base[i], bolts.base[bigger]);
                output += "Результат:\nБолты:\n";
                output += getStrFromArray(bolts);
                output += "\nГайки:\n";
                output += getStrFromArray(nuts);
                bigger++;
            }
            if (bolts.base[i] == center) {
                if (!equiv)
                    equiv = true;
                else {
                    cout << "Ошибка! Обнаружены одинаковые болты" <<
endl;

                    return 2;
                }
                output += "Болт имеет такой же размер, перемещаем в
конец\n";

                output += getStrSwapAction(bolts, i, length - 1);
                swap(bolts.base[i], bolts.base[length - 1]);
                output += "Результат:\nБолты:\n";
                output += getStrFromArray(bolts);
                output += "\nГайки:\n";
                output += getStrFromArray(nuts);
                continue;
            }
            cout << output << endl;
            i++;
        }
        equiv = false;
        center = bolts.base[length - 1];
        bigger = start;
        i = start;
        while (i < length - 1) {
            output = getStrCmpAction(bolts, length - 1, nuts, i);

```

```

        if (nuts.base[i] < center) {
            output += "Гайка меньше, перемещаем в левую часть\n";
            output += getStrSwapAction(nuts, i, bigger);
            swap(nuts.base[i], nuts.base[bigger]);
            output += "Результат:\nБолты:\n";
            output += getStrFromArray(bolts);
            output += "\nГайки:\n";
            output += getStrFromArray(nuts);
            bigger++;
        }
        if (nuts.base[i] == center) {
            if (!equiv)
                equiv = true;
            else {
                cout << "Ошибка! Обнаружены одинаковые гайки" <<
endl;

                return 2;
            }
            output += "Гайка имеет такой же размер, перемещаем в
конец\n";

            output += getStrSwapAction(nuts, i, length - 1);
            swap(nuts.base[i], nuts.base[length - 1]);
            output += "Результат:\nБолты:\n";
            output += getStrFromArray(bolts);
            output += "\nГайки:\n";
            output += getStrFromArray(nuts);
            continue;
        }
        cout << output << endl;
        i++;
    }
    ret = qsortNB(bolts, nuts, start, bigger, n);
    if (ret)
        return ret;
    ret = qsortNB(bolts, nuts, bigger, length - 1, n);
}
return ret;
}

int sortBN(dArr &bolts, dArr &nuts, int &n) {
    return qsortNB(bolts, nuts, 0, bolts.length, n);
}

int main() {
    int load;
    string bolts;
    string nuts;
    dArr boltsArr{};
    dArr nutsArr{};

```

```

int n;
bool isError;
    cout << "\x1b[32m***** Болты и гайки
*****\x1b[0m" << endl;
    cout << "Пожалуйста, выберите способ ввода данных: 1 - из файла, 0 -
из консоли, любое другое значение - выход из программы" << endl;
    cout << "Ваш выбор: ";
    cin >> load;
    if (load == 1) {
        string filePath = "input.txt";
        ifstream fin(filePath, ios::in);
        if (fin) {
            getline(fin, bolts);
            getline(fin, nuts);
        } else {
            cout << "Ошибка! Входной файл не найден" << endl;
            return 0;
        }
        fin.close();
    } else if (load == 0) {
        cout << "Введите массив болтов через пробел: " << endl;
        cin.ignore(1);
        getline(cin, bolts);
        cout << "Введите массив гаек через пробел: " << endl;
        getline(cin, nuts);
    } else
        return 0;
    int num = checkStr(bolts, nuts);
    switch (num) {
        case 0:
            cout << "Ошибка! Введен пустой массив" << endl;
            isError = true;
            break;
        case ERR_SYMBOL_B:
            cout << "Ошибка! Болты могут быть заданы только цифрами" <<
endl;
            isError = true;
            break;
        case ERR_SYMBOL_N:
            cout << "Ошибка! Гайки могут быть заданы только цифрами" <<
endl;
            isError = true;
            break;
        case ERR_LENGTH:
            cout << "Ошибка! Количество болтов и гаек не совпадает" <<
endl;
            isError = true;
            break;
        default:

```

```

        isError = false;
        break;
    }
    if (!isError) {
        boltsArr = makeArray(bolts, num);
        nutsArr = makeArray(nuts, num);
        if (!sortBN(boltsArr, nutsArr, n)) {
            if (!checkPairs(boltsArr, nutsArr))
                cout << "Обнаружены непарные болты и гайки" << endl;
            else {
                cout << "Количество элементов в массивах гаек и болтов: " << num << endl;
                cout << "Глубина рекурсии: " << n << endl;
                cout << "Сортировка завершена" << endl;
                cout << "Болты:" << endl;
                cout << getStrFromArray(boltsArr) << endl;
                cout << "Гайки:" << endl;
                cout << getStrFromArray(nutsArr) << endl;
                cout << "Найденные пары болтов и гаек:" << endl;
                cout << "\x1b[31m" << getPairs(boltsArr, nutsArr) << "\x1b[0m" << endl;
            }
        }
        else {
            cout << "Сортировка прервана\n";
        }
    }
    return 0;
}

```