

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

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

Студент гр. 9304

Кузнецов Р.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

### **Цель работы.**

Ознакомиться с различными алгоритмами сортировки, написать программу, сортирующую массив вводимых данных с помощью реализованного алгоритма сортировки.

### **Задание.**

Вариант 24.

Написать программу, сортирующую массив входных данных с помощью последовательной реализации битонной сортировки.

### **Объяснение работы алгоритма.**

При запуске программа предлагает ввести пользователю последовательность, которую необходимо отсортировать, после чего запускается функция `bitonicSort`, сортирующая данную последовательность. Она сначала заполняет сортируемый массив элементами, равные максимальному, до размера ближайшего числа, равного степени двойки, так как алгоритм работает только с массивами, чья длина равна степени двойки. Затем в цикле создаются битонные последовательности размеров степеней двойки, начиная с 4 и заканчивая размером обрабатываемого массива. Битонные последовательности образуются с помощью функции `makeBitonic`, запускаемой в два потока, и склеиваются с помощью функции `mergeBitonic`, запускаемой также в два потока. В результате работы этого цикла создается битонная последовательность, равная по длине размеру обрабатываемого массива. Затем над этим массивом в цикле выполняется слияние с шагом от равного половине длины массива до 1 (уменьшается в 2 раза каждую итерацию), после чего мы получаем отсортированный массив. Затем добавленные фиктивные элементы отбрасываются, а полученный массив выводится в `stdout`.

Промежуточные данные выводятся в поток `stderr`, для синхронизации вывода в него двумя параллельными потоками используется `std::mutex`. Пример отладочного вывода представлен на рисунке 1.

```

Роман@Coodahter /cygdrive/d/main/GitHub/ADS-9304/Kuznetsov/lab4
$ ./main.exe
Enter a sequence: 20 25 6 16 30 29 23 19 7 21 28 2 15 13 10 14 31 8 1 12 4 32 9 11 18 22 26 27 5 3 24 17
[Making] BS size 2; Comparison distance: 1; dir = DESCENDING
[Making] BS size 2; Comparison distance: 1; dir = ASCENDING
20 25 16 6 29 30 23 19 7 21 28 2 13 15 14 10 8 31 12 1 4 32 11 9 18 22 27 26 3 5 24 17
[Making] BS size 4; Comparison distance: 2; dir = DESCENDING
[Making] BS size 4; Comparison distance: 2; dir = ASCENDING
[Merging] BS size 4; Comparison distance: 1; dir = DESCENDING
[Merging] BS size 4; Comparison distance: 1; dir = ASCENDING
6 16 20 25 30 29 23 19 2 7 21 28 15 14 13 10 1 8 12 31 32 11 9 4 18 22 26 27 24 17 5 3
[Making] BS size 8; Comparison distance: 4; dir = ASCENDING
[Making] BS size 8; Comparison distance: 4; dir = DESCENDING
[Merging] BS size 8; Comparison distance: 2; dir = DESCENDING
[Merging] BS size 8; Comparison distance: 2; dir = ASCENDING
6 16 20 19 23 25 30 29 21 28 15 14 13 10 2 7 1 4 9 8 12 11 32 31 26 27 24 22 18 17 5 3
[Merging] BS size 8; Comparison distance: 1; dir = DESCENDING
[Merging] BS size 8; Comparison distance: 1; dir = ASCENDING
6 16 19 20 23 25 29 30 28 21 15 14 13 10 7 2 1 4 8 9 11 12 31 32 27 26 24 22 18 17 5 3
[Making] BS size 16; Comparison distance: 8; dir = DESCENDING
[Making] BS size 16; Comparison distance: 8; dir = ASCENDING
[Merging] BS size 16; Comparison distance: 4; dir = DESCENDING
[Merging] BS size 16; Comparison distance: 4; dir = ASCENDING
6 10 7 2 13 16 15 14 23 21 19 20 28 25 29 30 27 26 31 32 18 17 24 22 11 12 8 9 1 4 5 3
[Merging] BS size 16; Comparison distance: 2; dir = DESCENDING
[Merging] BS size 16; Comparison distance: 2; dir = ASCENDING
6 2 7 10 13 14 15 16 19 20 23 21 28 25 29 30 31 32 27 26 24 22 18 17 11 12 8 9 5 4 1 3
[Merging] BS size 16; Comparison distance: 1; dir = DESCENDING
[Merging] BS size 16; Comparison distance: 1; dir = ASCENDING
2 6 7 10 13 14 15 16 19 20 21 23 25 28 29 30 32 31 27 26 24 22 18 17 12 11 9 8 5 4 3 1
[Merging] BS size 32; Comparison distance: 16; dir = ASCENDING
2 6 7 10 13 14 15 16 12 11 9 8 5 4 3 1 32 31 27 26 24 22 18 17 19 20 21 23 25 28 29 30
[Merging] BS size 32; Comparison distance: 8; dir = ASCENDING
2 6 7 8 5 4 3 1 12 11 9 10 13 14 15 16 19 20 21 23 24 22 18 17 32 31 27 26 25 28 29 30
[Merging] BS size 32; Comparison distance: 4; dir = ASCENDING
2 4 3 1 5 6 7 8 12 11 9 10 13 14 15 16 19 20 18 17 24 22 21 23 25 28 27 26 32 31 29 30
[Merging] BS size 32; Comparison distance: 2; dir = ASCENDING
2 1 3 4 5 6 7 8 9 10 12 11 13 14 15 16 18 17 19 20 21 22 24 23 25 26 27 28 29 30 32 31
[Merging] BS size 32; Comparison distance: 1; dir = ASCENDING
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Sorted sequence: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Sorted sequence validated successfully

```

Рисунок 1 – Пример отладочного вывода

## Тестирование.

Для тестирования были написаны python и bash скрипты. Python скрипт принимает сначала отсортированный с помощью программы массив, затем исходный. После чего сортирует исходный массив сама и сравнивает с результатом работы программы. В случае соответствия выводится соответствующее сообщение, в случае несоответствия также выводится правильно отсортированный массив. Bash скрипт ответственен за поставку тестовых данных из папки Tests и результата работы программы тестовому скрипту. Запуск тестов производится при помощи команды «make test», компиляция программы с помощью «make compile»

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 3 4 6 3 5 9	1 3 3 4 5 6 9	Случайный набор
2.	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Одинаковые элементы
3.	12 65 2 53 24 53 2	2 2 12 24 53 53 65	Имеются дубликаты
4.	20 25 6 16 30 29 23 19 7 21 28 2 15 13 10 14 31 8 1 12 4 32 9 11 18 22 26 27 5 3 24 17	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32	Перемешаны все числа от 1 до 32
5.	1	1	Один элемент
6.	-4 1 -2 -3 4 3 0 -5 -1 2 5	-5 -4 -3 -2 -1 0 1 2 3 4 5	Отрицательные элементы
7.	-341 42 245 52 531 10 -124 952 -154 262 52 2345 14 351	-341 -154 -124 10 14 42 52 52 245 262 351 531 952 2345	Элементы побольше
8.	1 2 3 3 2 1	1 1 2 2 3 3	Только дубликаты
9.	1 2 3	1 2 3	Изначально отсортирован
10.	8 7 6 5 4 3 2 1	1 2 3 4 5 6 7 8	Отсортирован в обратном порядке

### Выводы.

В процессе выполнения работы было проведено ознакомление с различными алгоритмами сортировки, также был реализован один из них (битонная сортировка).

Была разработана программа, сортирующая массив входных данных по алгоритму битонной сортировки. У данного алгоритма сложностью  $O(n \log^2 n)$  имеется преимущество в том, что его можно выполнять параллельно, однако существенным недостатком является применимость только для последовательностей длины  $2^n$ , что может увеличить время выполнения алгоритма почти в три раза.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <thread>
#include <iterator>
#include <mutex>
#ifdef _DEBUG
std::mutex cerr_mutex; //Sorry, i have to...
#endif
template <typename T>
std::istream& operator>>(std::istream& in, std::vector<T>& seq) {
    seq.reserve(8);
    while (in.peek() != '\n' && in.peek() != EOF)
    {
        T tmp;
        in >> tmp;
        seq.push_back(tmp);
    }
    return in;
}
template <typename T>
std::ostream& operator<<(std::ostream& out, std::vector<T>& seq) {
    std::copy(seq.begin(), seq.end(), std::ostream_iterator<T>(out, "
));
    return out << '\n';
}
namespace bitSort {
    enum direction {
        ASCENDING, //0
        DESCENDING, //1
    };
    template <typename T>
    void makeBitonic(std::vector<T>& seq, const unsigned begin, const
unsigned compDist, const direction dir) {
#ifdef _DEBUG
        {const std::lock_guard<std::mutex> lock_cerr(cerr_mutex);
            std::cerr << "\033[35m[Making]\033[m BS size \033[33m" << 2 *
compDist << "\033[m; Comparison distance: \033[33m"
                << compDist << "\033[m; dir = " << (dir ?
"\033[31mDESCENDING" : "\033[32mASCENDING") << "\033[m\n";}
#endif
        for (unsigned i = begin; i < seq.size(); i += 4 * compDist)
            for (unsigned j = i; j < i + compDist; j++)
                if ((seq[j] > seq[j + compDist]) != dir)
                    std::swap(seq[j + compDist], seq[j]);
        }
    template <typename T>
    void mergeBitonic(std::vector<T> & seq, const unsigned begin,
const unsigned length, const unsigned compDist, const direction dir) {
```

```

#ifdef _DEBUG
    {const std::lock_guard<std::mutex> lock_cerr(cerr_mutex);
        std::cerr << "\033[34m[Merging]\033[m BS size \033[33m" << length
        << "\033[m; Comparison distance: \033[33m"
            << compDist << "\033[m; dir = " << (dir ?
"\033[31mDESCENDING" : "\033[32mASCENDING") << "\033[m\n"; }
    }
#endif
    for (unsigned i = begin; i < seq.size(); i += 2 * length)
        for (unsigned j = i; j < i + length; j += 2 * compDist)
            for (unsigned k = j; k < j + compDist; k++)
                if ((seq[k] > seq[k + compDist]) != dir)
                    std::swap(seq[k + compDist], seq[k]);
    }
    template <typename T>
    void debug_print(std::ostream& out, const std::vector<T>& seq,
const unsigned bsSize) {
#ifdef _DEBUG
        if (!bsSize) {
            std::copy(seq.begin(), seq.end(), std::ostream_iterator<T>(out,
" "));
            out << '\n';
            return;
        }
        char color[][6] = { "\033[32m", "\033[31m" };
        bool pick = 0;
        for (auto it = seq.begin(); it != seq.end(); it+=bsSize) {
            out << color[pick];
            pick = !pick;
            for (auto jt = it; std::distance(it, jt) < bsSize; jt++)
                out << *jt << ' ';
            }
            out << "\033[m\n";
        }
    };
    template <typename T>
    void bitonicSort(std::vector<T>& seq) {
        size_t oldSize = seq.size(),
            newSize = 1 <<
static_cast<size_t>(std::ceil(std::log2(seq.size())));
        seq.resize(newSize, *std::max_element(seq.begin(), seq.end()));
        //bitonic sort only works with sequences of 2^x length
        for (size_t i = 2; i < newSize; i *= 2) { // i = Bitonic Sequence
size
            std::thread makeAscending(bitSort::makeBitonic<T>, std::ref(seq),
0, i / 2, bitSort::ASCENDING);
            bitSort::makeBitonic<T>(seq, i, i / 2, bitSort::DESCENDING);
            makeAscending.join();
            for (size_t j = i / 4; j > 0; j /= 2) { //merging
                std::thread mergeAscending(bitSort::mergeBitonic<T>,
std::ref(seq), 0, i, j, bitSort::ASCENDING);
                bitSort::mergeBitonic<T>(seq, i, i, j, bitSort::DESCENDING);
                mergeAscending.join();
                if(j>1)
                    bitSort::debug_print(std::cerr, seq, 0);
            }
            bitSort::debug_print(std::cerr, seq, i);
        }
    }
}

```

```

    for (int j = newSize / 2; j > 0; j /= 2) { //final merge
        bitSort::mergeBitonic(seq, 0, newSize, j, bitSort::ASCENDING);
        bitSort::debug_print(std::cerr, seq, 0);
    }
    seq.resize(oldSize); //remove fictitious elements
}
int main() {
    std::vector<int> seq;
#ifdef TEST
    std::cout << "Enter a sequence: ";
#endif
    std::cin >> seq;
#ifdef _DEBUG
    std::vector<int> test = seq;
#endif
    bitonicSort(seq);
#ifdef TEST
    std::cout << "Sorted sequence: ";
#endif
    std::cout << seq;
#ifdef _DEBUG
    std::sort(test.begin(), test.end());
    if (std::equal(seq.begin(), seq.end(), test.begin(), test.end()))
        std::cerr << "\033[32mSorted sequence validated
successfully\033[m\n";
    else
        std::cerr << "\033[31mERROR: sequence wasn't sorted
correctly\033[m\n";

#endif
    return 0;
}

```