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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Алгоритмы сортировки

Студент гр. 9303 _____ Низовцов Р. С.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург
2020

Цель работы.

Написание алгоритма сортировки в соответствии с вариантом задания, теоретическая оценка сложности алгоритма.

Задание.

Вариант 4

Пузырьковая сортировка оптимизированная; сортировка чёт-нечет.

Основные теоретические положения.

Сортировка простыми обменами, сортиро́вка пузырько́м — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются $N - 1$ раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

Алгоритм чёт-нечет представляет вариацию алгоритма пузырьковой сортировки. В последовательном варианте он применяется редко, поскольку он сложнее и интуитивно менее понятен, чем алгоритм пузырька. Интересен он тем, что допускает естественное распараллеливание.

Описание алгоритма.

Как и в алгоритме пузырька внешний цикл задает n проходов по сортируемому массиву. На каждом проходе, как и в алгоритме пузырька, происходит сравнение и обмен двух соседних элементов. Но есть два важных отличия:

- на каждом проходе производится *n/2* независимых сравнений соседних пар, так что никакой элемент пары не участвует в дальнейших сравнениях на данном проходе;

- Проходы делятся на четные и нечетные. На четных проходах обмен начинается с пары (a_{n-1}, a_{n-2}) . На нечетном проходе производится сдвиг и начальной парой является пара (a_{n-2}, a_{n-3}) . (Предполагается, что нумерация элементов массива начинается с нуля).

В отличие от "пузырька", где на i -м проходе первые i элементов занимают свои места, в алгоритме "чет - нечет" элементы гарантировано занимают свои места после выполнения всех n проходов. Для самого легкого элемента достаточно n - 1 проход для "всплытия" в вершину массива, так как на каждом проходе элемент поднимается вверх на одну позицию. Для следующего за ним элемента может понадобиться в самом неблагоприятном случае ровно N проходов. На первом проходе элемент может опуститься на последнее место (например в случае инверсного массива), на втором проходе остаться на последнем месте, поскольку не будет участвовать в сравнениях, а затем начнет подниматься и за n - 2 прохода станет на свое место.

Теоретическая оценка сложности алгоритма: в идеальном случае она равна O(n), так как мы производим ровно (n) параллельных проверок для получения итога. Но для достижения идеального случая нам понадобится (n/2) потоков для параллельного сравнивания пар. Следовательно, при наихудшем случае сложность алгоритма будет равна $O(n^2/2)$.

Выполнение работы.

Для выполнения программы были реализована функция OddEvenSortSeq.

В функции void OddEvenSortSeq(int*, int, ofstream&) производится сортировка методом чет-нечет. Функция принимает массив данных для сортировки, его длины и поток для вывода в файл. Далее циклично происходит проверка пар элементов с промежуточным выводом данных. Исходный код программы представлен в приложении А. Результаты тестирования включены в приложение Б.

Выводы.

Ознакомился с алгоритмом сортировки «чет-нечет», изучил его особенности, высчитал его сложность и реализовал программу, решающую поставленную задачу с помощью данного алгоритма.

Была реализована программа, включающая в себя функцию OddEvenSortSeq для сортировки элементов, хранящихся в массиве. Программа выполняет запись результата в файл, а также производит проверку и обработку полученных данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>

using namespace std;

void OddEvenSortSeq(int* mas, int size, ofstream &outFile)
{
    int n = size;
    int temp = 0;
    for(int i = 0; i < n; i++){
        cout << mas[i] << " ";
        outFile << mas[i] << " ";
    }
    cout << endl;
    outFile << endl;
    for (int k = 0; k < n; k++){
        if (k % 2 == 0)
            for (int j = 0; j + 1 < size; j += 2){
                if (mas[j] > mas[j + 1]){
                    temp = mas[j];
                    mas[j] = mas[j + 1];
                    mas[j + 1] = temp;
                }
            }
        else
            for (int j = 1; j + 1 < size; j += 2){
                if (mas[j] > mas[j + 1]){
                    temp = mas[j];
                    mas[j] = mas[j + 1];
                    mas[j + 1] = temp;
                }
            }
        for(int i = 0; i < size; i++){
            cout << mas[i] << " ";
            outFile << mas[i] << " ";
        }
        cout << "  <-" << k+1 << " C€P°Pi" << endl;
        outFile << "  <-" << k+1 << " C€P°Pi" << endl;
    }
    cout << endl;
    outFile << endl;
}

int main(){
    srand(time(NULL));

    cout << "Result was saved in result.txt" << endl;
    ofstream outFile("/home/rostislav/result.txt");
```

```
    cout << "If you want create new array write Continue, else write End"
<< endl;

    string line;
    while(getline(cin, line)){
        if(line == "End")
            break;
        else if(line != "Continue")
            continue;

        int size = rand()%10 + 5;
        int* mas = new int[size];
        for(int i = 0; i < size; i++){
            mas[i] = rand()%100;
        }
        OddEvenSortSeq(mas, size, outFile);
        delete [] mas;
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 — Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарий
1.	39 15 32 63 69	15 39 32 63 69 <-1 шаг 15 32 39 63 69 <-2 шаг 15 32 39 63 69 <-3 шаг 15 32 39 63 69 <-4 шаг 15 32 39 63 69 <-5 шаг	Программа работает корректно
2.	94 33 9 24 8 3 2 29 44 45	33 94 9 24 3 8 2 29 44 45 <-1 шаг 33 9 94 3 24 2 8 29 44 45 <-2 шаг 9 33 3 94 2 24 8 29 44 45 <-3 шаг 9 3 33 2 94 8 24 29 44 45 <-4 шаг 3 9 2 33 8 94 24 29 44 45 <-5 шаг 3 2 9 8 33 24 94 29 44 45 <-6 шаг 2 3 8 9 24 33 29 94 44 45 <-7 шаг 2 3 8 9 24 29 33 44 94 45 <-8 шаг 2 3 8 9 24 29 33 44 45 94 <-9 шаг 2 3 8 9 24 29 33 44 45 94 <-10 шаг	Программа работает корректно
3.	65 7 50 10 4	7 65 10 50 4 <-1 шаг 7 10 65 4 50 <-2 шаг 7 10 4 65 50 <-3 шаг 7 4 10 50 65 <-4 шаг 4 7 10 50 65 <-5 шаг	Программа работает корректно

4.	27 90 58 18 5 43 34 26 0 28 60 10 4	27 90 18 58 5 43 26 34 0 28 10 60 4 <-1 шаг 27 18 90 5 58 26 43 0 34 10 28 4 60 <-2 шаг 18 27 5 90 26 58 0 43 10 34 4 28 60 <-3 шаг 18 5 27 26 90 0 58 10 43 4 34 28 60 <-4 шаг 5 18 26 27 0 90 10 58 4 43 28 34 60 <-5 шаг 5 18 26 0 27 10 90 4 58 28 43 34 60 <-6 шаг 5 18 0 26 10 27 4 90 28 58 34 43 60 <-7 шаг 5 0 18 10 26 4 27 28 90 34 58 43 60 <-8 шаг 0 5 10 18 4 26 27 28 34 90 43 58 60 <-9 шаг 0 5 10 4 18 26 27 28 34 43 90 58 60 <-10 шаг 0 5 4 10 18 26 27 28 34 43 58 90 60 <-11 шаг 0 4 5 10 18 26 27 28 34 43 58 60 90 <-12 шаг 0 4 5 10 18 26 27 28 34 43 58 60 90 <-13 шаг	Программа работает корректно