МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева»

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Алгоритмы и структуры данных

Алгоритмы сортировки данных во внешней памяти.

Руководитель _____ В. В. Тынченко
подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 _____ К.В. Трифонов
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

Цель работы:

Изучение алгоритмов сортировки данных во внешней памяти, особенностей их программной реализации и эффективности работы на различных наборах исходных данных.

Общая постановка задачи.

Разработать и реализовать программу, предназначенную для исследования времени работы и поведения двух выбранных методов внешней сортировки. При выборе методов, подлежащих реализации, необходимо ознакомиться с принципом оценивания результатов работы, изложенным ниже.

Провести исследование параметров работы, указанных в варианте задания методов сортировки на различных наборах данных. Представить отчет, содержащий результаты исследования и полученные выводы.

Требования к функциональным возможностям программы.

Программа должна содержать меню, позволяющее выбирать один из двух режимов работы программы:

1) сортировка файла данных, сформированных случайным образом;
2) режим накопления статистических данных

В **первом режиме** требуется предоставить пользователю следующие возможности:

- задавать размер числовой последовательности, содержащейся в файле;
- указывать диапазон значений элементов последовательности;
- выбирать метод внешней сортировки.

Результаты работы программы в данном режиме:

- вывести на экран количество сравнений и перестановок элементов массива.

Во **втором режиме** пользователь должен иметь возможность:

- выбирать способ формирования элементов последовательности, содержащейся в файле (случайные значения, упорядоченная последовательность значений, значения расположены в обратном порядке);
- задавать диапазон и шаг изменения размера последовательности;
- выбирать метод внешней сортировки.

Результаты работы программы в данном режиме:
Для каждого значения размера формируется набор данных и сортируется выбранным методом. В файл с указанным именем выводятся значения времени сортировки для каждого количества элементов в наборе.

Для получения оценки **«отлично»** необходимо программно реализовать следующие методы сортировки данных во внешней памяти: сортировка прямым слиянием; многофазная сортировка. Оценить быстродействие указанных методов и степень естественности их поведения.

Код программы:

Описание вспомогательных функций:

```
void getFile(string filename, int file_size, int n1, int n2){
    ofstream in(filename);
    srand(time(0));
    for (int i = 0; i < file_size; i++){
        in << n1 + rand()%(n2-n1+1)<<" ";
    }
    in.close();
}
```

Функция, создающая файл с заданным именем, и заполняет его нужным количеством случайных чисел в диапазоне.

```
void getUFile(string filename, int file_size){
    ofstream in(filename);
    int temp;
    for (int i = 0; i < file_size; i++){
            temp = i + 1;
            in << temp<<" ";
    }
    in.close();
}
```

Функция, создающая отсортированный файл с заданным именем.

```
void getUrFile(string filename, int file_size){
    ofstream in(filename);
    int temp;
    for (int i = 0; i < file_size; i++){
            temp = file_size-i;
            in << temp<<" ";
    }
    in.close();

}
```

Функция, создающая отсортированный в обратном порядке файл с заданным именем.

```
void readFile(string filename, int file_size){
    ifstream in(filename);
    int temp;
    in>>temp;
    for (int i = 0; i < file_size; i++){
        cout<<temp<<" ";
        in>>temp;
    }
    cout<<"\n";
    in.close();
}
```

Функция вывода на экран файла с заданным именем.

Методы сортировки:

Сортировка прямым слиянием:

```
//Сортировка прямым слиянием
void DirectSort(string filename, int file_size) {
    int count = 0;
```

Цикл слияния/деления нужно повторять log2(size) раз (глубина алгоритма).

```
    for (int i = 1; i < file_size; i *= 2) {
```

Деление происходит до тех пор, пока в исходном файле имеются числа, по принципу перетаскивания частей размером в size делённое на текущую глубину.

```
        //делим файлы для слияния
        int temp;
        ifstream in(filename);
        ofstream out1("buffer1.txt"), out2("buffer2.txt");
        in >> temp;
        while (!in.eof()) {
            for (int I = 0; I < i && !in.eof(); I++) {
                out1 << temp << " ";
                in >> temp;
            }
            for (int J = 0; J < i && !in.eof(); J++) {
                out2 << temp << " ";
                in >> temp;
            }
        }
        if (i == 1)
            out2 << temp << " ";
        in.close();
        out1.close();
        out2.close();
```

Слияние двух файлов в один файл-серию.

```
        //Слияние
        ifstream Input_file1("buffer1.txt"), Input_file2("buffer2.txt");
        ofstream outFile(filename);
        int a, b, I, J;
        Input_file1 >> a;
        Input_file2 >> b;
        while (!Input_file1.eof() && !Input_file2.eof()) {
            //Сливаем пока в файлах есть элементы
            while (I < i && J < i && !Input_file1.eof() && !Input_file2.eof()) {
                //Слияние файлов в один по порядку
                if (a < b) {
                    outFile << a << " ";
                    Input_file1 >> a;
                    I++;
                }
                else {
```

```
                    outFile << b << " ";
                    Input_file2 >> b;
                    J++;
                }
            }
            //Слияние оставшихся элементов в одном из двух файлов
            while (I < i && !Input_file1.eof()) {
                outFile << a << " ";
                Input_file1 >> a;
                I++;
            }
            while (J < i && !Input_file2.eof()) {
                outFile << b << " ";
                Input_file2 >> b;
                J++;
            }
            I = J = 0;
        }
        while (!Input_file1.eof()) {
            outFile << a << " ";
            Input_file1 >> a;
        }
        while (!Input_file2.eof()) {
            outFile << b << " ";
            Input_file2 >> b;
        }
        Input_file1.close();
        Input_file2.close();
        outFile.close();
        count++;
        }
    cout<<"Всего слияний: "<<count<<"\n";
    remove("buffer1.txt");
    remove("buffer2.txt");
}
```

Многофазная сортировка:

Определение глубины алгоритма и максимального числа серий.

```
//Многофазная сортировка
void PolyphaseSort(string filename) {
    FILE* _s;
    fopen_s(&_s, filename.c_str(), "r");
    int cur, _prev, num = 1;
    fscanf_s(_s, "%d", &cur);
    while (!feof(_s)) {
        _prev = cur;
        fscanf_s(_s, "%d", &cur);
        if (cur < _prev)
            num++;
    }
    fclose(_s);
    int min = 0, max = 1, Fib = 1, deep = 0;
    //Определение глубины алгоритма по Фибоначчи
    while (Fib < num) {
        min = max;
        max = Fib;
        Fib = min + max;
        deep++;
    }
    //Максимальное число серий
    int maxS = (min + max) - num;
```

Распределение исходного файла во временные:

```
    FILE* s, * buf1, * buf2;
    fopen_s(&s, filename.c_str(), "r");
    fopen_s(&buf1, "buffer1.txt", "w");
    int elem, prev, sI = 0, merges;
    fscanf_s(s, "%d", &elem);
    while (sI != min) {
        prev = elem;
        fprintf_s(buf1, "%d ", elem);
        fscanf_s(s, "%d", &elem);
        if (elem < prev)
            sI++;
    }
    fclose(buf1);
    fopen_s(&buf2, "buffer2.txt", "w");
    while (!feof(s)) {
        fprintf_s(buf2, "%d ", elem);
        fscanf_s(s, "%d", &elem);
    }
    fprintf_s(buf2, "%d ", elem);
    fclose(s);
    for (int i = maxS; i > 0; i--) {
        fprintf_s(buf2, "%d ", 0);
        fprintf_s(buf2, "%d ", 1);
    }
    fclose(buf2);
```

```
    //Создаем массив названий временных файлов
    char* Buff[][3] = { {"buffer1.txt", "buffer2.txt", "buffer3.txt"},
                        {"buffer2.txt", "buffer3.txt", "buffer1.txt"},
                        {"buffer3.txt", "buffer1.txt", "buffer2.txt"} };
```

Далее поочерёдно выполняем слияние первого и второго, второго и третьего и третьего и первого файлов в серии.

```
    //Выполняем поочерёдное слияние
    for (merges = 0; merges < deep; merges++) {
        FILE* in1, * in2, * out, * temp;
        fopen_s(&in1, Buff[merges % 3][0], "r"); fopen_s(&in2, Buff[merges %
3][1], "r"); fopen_s(&out, Buff[merges % 3][2], "w");
        int elem1, elem2, prev1, prev2;
        fscanf_s(in1, "%d", &elem1);
        fscanf_s(in2, "%d", &elem2);
        while (!feof(in1)) {
            //Проверка каждой серии на последовательность
            bool flag1 = true, flag2 = true;
            while (flag1 && flag2) {
                if (elem1 < elem2) {
                    fprintf_s(out, "%d ", elem1);
                    prev1 = elem1;
```

```
                fscanf_s(in1, "%d", &elem1);
                if (elem1 < prev1 || feof(in1))
                    flag1 = false;
            }
            else {
                fprintf_s(out, "%d ", elem2);
                prev2 = elem2;
                fscanf_s(in2, "%d", &elem2);
                if (elem2 < prev2 || feof(in2))
                    flag2 = false;
            }
        }
        while (flag1 && !feof(in1)) {
            fprintf_s(out, "%d ", elem1);
            prev1 = elem1;
            fscanf_s(in1, "%d", &elem1);
            if (elem1 < prev1)
                flag1 = false;
        }
        while (flag2 && !feof(in2)) {
            fprintf_s(out, "%d ", elem2);
            prev2 = elem2;
            fscanf_s(in2, "%d", &elem2);
            if (elem2 < prev2)
                flag2 = false;
        }
    }
    fopen_s(&temp, "_buffer.txt", "w");
    while (!feof(in2)) {
        fprintf_s(temp, "%d ", elem2);
        fscanf_s(in2, "%d", &elem2);
    }
    fclose(in1); fclose(in2); fclose(out); fclose(temp);
    fopen_s(&temp, "_buffer.txt", "r");
    fopen_s(&in2, Buff[merges % 3][1], "w");
    while (fscanf_s(temp, "%d", &elem1) != EOF)
        fprintf_s(in2, "%d ", elem1);
    fclose(in2);
    fclose(temp);
    remove("_buffer.txt");
}
int null = maxS;
int one = null;
if ((merges - 1) % 3 == 0)
    fopen_s(&buf1, "buffer3.txt", "r");
else if ((merges - 1) % 3 == 1)
    fopen_s(&buf1, "buffer1.txt", "r");
else
    fopen_s(&buf1, "buffer2.txt", "r");
fopen_s(&s, filename.c_str(), "w");
//Запись в файл вывода с проверкой на нули
```

```
    while (fscanf_s(buf1, "%d", &elem) != EOF) {
        if (elem == 0 && null > 0) {
            null--;
        }
        else if (elem == 1 && one > 0) {
            one--;
        }
        else
            fprintf_s(s, "%d ", elem);
    }
    fclose(buf1);fclose(s);
    //Чистка временных файлов
    remove("buffer1.txt"); remove("buffer2.txt"); remove("buffer3.txt");
    cout<<"Всего слияний: "<<deep - 1<<"\n";
}
```

Меню тестирования в функции main():

```
1) Сортировка одного файла, сформированного случайным образом
2) Режим накопления статистических данных
0) Выход
```

Первый режим работы программы:

```
case 1:{
                int n;
                cout<<"Задайте длину файла: ";
                cin>>n;
                cout<<"Введите диапозон генерации от n1 до n2 (n1_n2):";
                int n1, n2;
                cin>>n1>>n2;
                getFile("numbers.txt",n,n1,n2);
                int inp;
                cout<<"1) Сортировка прямым слиянием Merge Sort\n"
                <<"2) Многофазная сортировка Polyphase Merge Sort\n";
                cin>>inp;
                int t_1 = clock();
                switch (inp){
                    case 1:{
                        DirectSort("numbers.txt",n);
                        break;
                    }
                    case 2:{
                        PolyphaseSort("numbers.txt");
                        break;
                    }
                }
                int t_2 = clock();
                int ans = t_2 - t_1;
                cout<<n<< " эл отсортированы за "<<ans<<"ms\n";
                readFile("numbers.txt",n);
                break;
            }
```

Создаётся файл длинной n со случайными значениями от n1 до n2, далее он сортируется и выводится на экран.

Пример работы режима с заполнением файла длинной 10 эл случайными числами от 1 до 5 (вывод данных в консоль и файл numbers.txt):

```
1
Задайте длину файла: 10
Введите диапозон генерации от n1 до n2 (n1_n2):1 5
1) Сортировка прямым слиянием Merge Sort
2) Многофазная сортировка Polyphase Merge Sort
2
Всего слияний: 2
10 эл отсортированы за 3ms
1 3 3 3 3 3 4 4 4 5
```

numbers.txt

```
1 3 3 3 3 3 4 4 4 5 5
```

Второй режим работы программы:

```
case 2:{
                int size;
                int in2,in3;
                int a1,b1,l;
                cout<<"1) Заполнение случайными значениями\n"
                <<"2) Заполнение упорядоченными значениями\n"
                <<"3) Заполнение обратноупорядоченными значениями\n";
                cin>>in2;
                cout<<"Введите диапазон (от _ до _) и шаг через пробел:\n";
                cin>>a1>>b1>>l;
                int N = (b1-a1)/l+1;
                for (int i = 0; i<N-1; i++){
                    size = ((i)*l)+a1;
                    switch (in2){
                        case 1:{
                            getFile("numbers" + to_string(i+1) +
".txt",size,1,10);
                            break;
                        }
                        case 2:{
                            getUFile("numbers" + to_string(i+1) + ".txt",size);
                            break;
                        }
                        case 3:{
                            getUrFile("numbers" + to_string(i+1) + ".txt",size);
                            break;
                        }
                    }
                }
                cout<<"Всего заполнено "<<N-1<<" массивов.\n";
```

Создание N файлов пользовательской длинны N = (b1-a1)/L+1, где b1 и a1 границы диапазона длин массивов и L – шаг. Последующее заполнение в зависимости от выбора пользователя, используя методы функции, описанные ранее.

Пользователь выбирает метод сортировки:

```
                cout<<"1) Сортировка прямым слиянием Merge Sort\n"
                <<"2) Многофазная сортировка Polyphase Merge Sort\n";
                cin>>in2;
                ofstream fout("results.txt");
                int t1 = clock();
                switch (in2){
                    case 1:{
                        for (int i = 0; i<N-1; i++){
                            size = ((i)*l)+a1;
                            int t_1 = clock();
                            DirectSort("numbers" + to_string(i+1) + ".txt",size);
```

```
                    int t_2 = clock();
                    int ans = t_2 - t_1;
                    cout<<" эл отсортированы за "<<ans<<"ms\n";
                    fout<<size<<" "<<ans<<"\n";
                }
                break;
            }
            case 2:{
                for (int i = 0; i<N-1; i++){
                    size = ((i)*l)+a1;
                    int t_1 = clock();
                    PolyphaseSort("numbers" + to_string(i+1) + ".txt");
                    int t_2 = clock();
                    int ans = t_2 - t_1;
                    cout<<" эл отсортированы за "<<ans<<"ms\n";
                    fout<<size<<" "<<ans<<"\n";
                }
                break;
            }
        }
        int t2 = clock();
        int answ = (t2-t1);
        fout.close();
        cout<<"Общее время: "<<answ<<"ms\n";
        cout<<"Очистить файлы сортировки? 1/0\n";
        int clear;
        cin>>clear;
        if (clear){
            for (int i = 0; i<N-1; i++){
                remove(("numbers" + to_string(i+1) + ".txt").c_str());
            }
        }
        break;
    }
```

Записывается время t1, затем каждый файл сортируется выбранной функцией, далее записывается время t2, время выполнения выводится экран и в файл *results.txt*, далее выводится на экран общее время, за которое были отсортированы все N файлов. Пользователю также предлагается удалить или оставить файлы.

Пример работы режима в диапазоне от 1000 до 20000 эл с шагом 1000 эл (вывод данных в консоль и в файл results.txt:

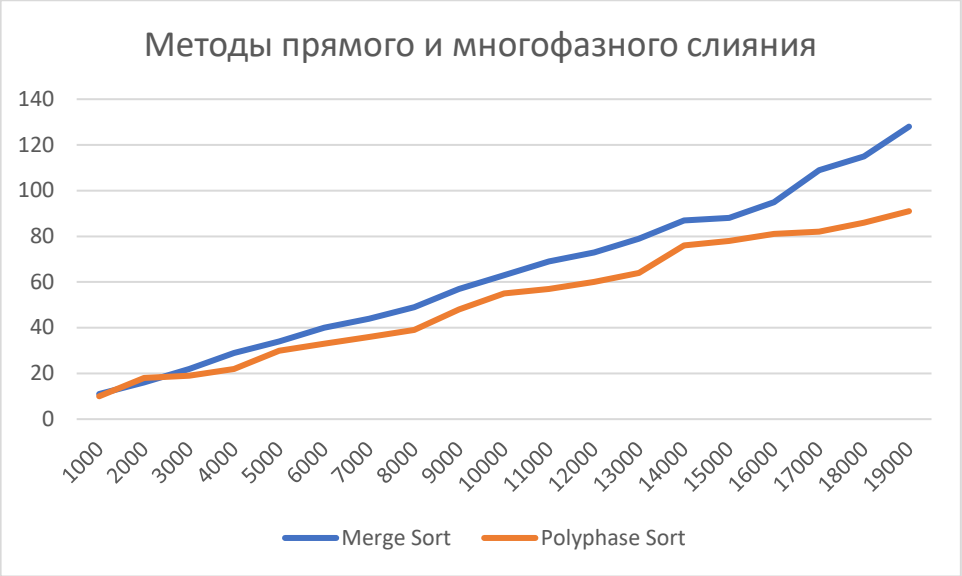
```
1) Сортировка одного файла, сформированного случайным образом
2) Режим накопления статистических данных
0) Выход
2
1) Заполнение случайными значениями
2) Заполнение упорядоченными значениями
3) Заполнение обратноупорядоченными значениями
1
Введите диапазон (от _ до _) и шаг через пробел:
1000 10000 1000
Всего заполнено 9 массивов.
1) Сортировка прямым слиянием Merge Sort
2) Многофазная сортировка Polyphase Merge Sort
1
Всего слияний: 10
1000 эл отсортированы за 10ms
Всего слияний: 11
2000 эл отсортированы за 16ms
Всего слияний: 12
3000 эл отсортированы за 22ms
Всего слияний: 12
4000 эл отсортированы за 28ms
Всего слияний: 13
5000 эл отсортированы за 36ms
Всего слияний: 13
6000 эл отсортированы за 39ms
Всего слияний: 13
7000 эл отсортированы за 45ms
Всего слияний: 13
8000 эл отсортированы за 48ms
Всего слияний: 14
9000 эл отсортированы за 57ms
Общее время: 303ms
Очистить файлы сортировки? 1/0
```

results.txt

```
1000 10
2000 16
3000 22
4000 28
5000 37
6000 41
7000 43
8000 50
9000 62
```

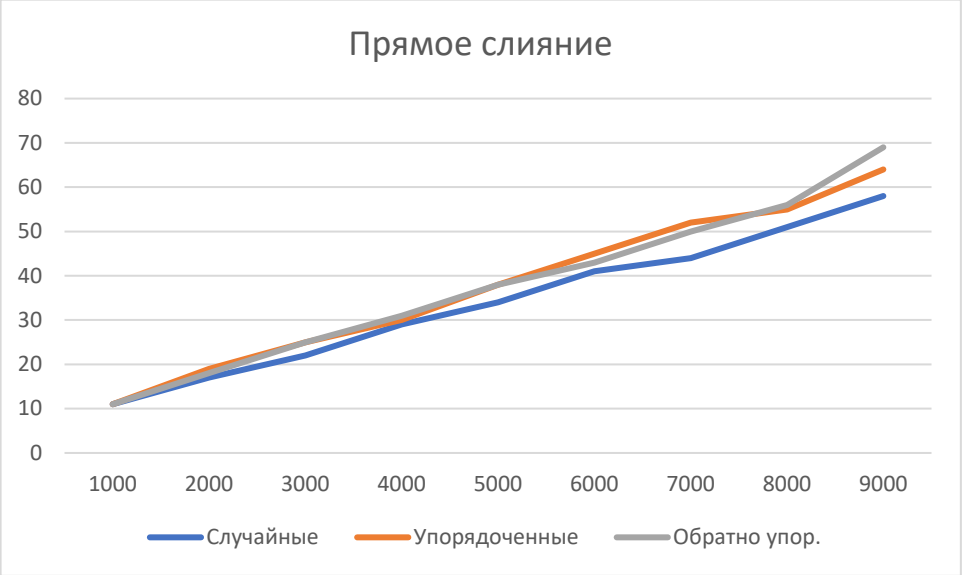
Исследование работы алгоритмов shaker sort и insert sort

График зависимости времени от длины массива, заполненного случайными значениями:



Число делений файла у обоих методов равно log2(size), однако многофазная сортировка слиянием обеспечивает более высокий эффективный коэффициент уменьшения количества запусков за счет неравномерного распределения отсортированных запусков между N – 1 рабочими файлами.

График зависимости времени для прямой сортировки от длины файла, заполненного случайными, упорядоченными и обратно упорядоченными значениями:



Время на сортировку файла, заполненного случайными или обратно упорядоченными значениями не сильно больше, чем упорядоченного. Можно сделать вывод, что алгоритм имеет **не естественное** поведение.

График зависимости времени для сортировки многофазным слиянием от длины файла, заполненного случайными, упорядоченными и обратно упорядоченными значениями:

Сортировка упорядоченных файлов занимает на порядок меньше времени, чем случайно заполненных. Также сортировка обратно упорядоченной последовательности занимает наибольшее время. Можно сделать вывод, что алгоритм имеет **естественное** поведение.

Вывод

Изучение алгоритмов сортировки во внешней памяти является важным шагом в области оптимизации обработки больших объемов данных. Многофазное слияние, являясь доработанным наследником метода прямого слияния, показывает преимущественно лучшие результаты. Также многофазное слияние имеет естественное поведение, по сравнению с методом сортировки прямым слиянием.