Балтийский государственный технический университет
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

Практическая работа №3
по дисциплине «Структуры и организация данных»
на тему «Оценка эффективности алгоритмов»
часть 1 «Алгоритмы сортировки»

вариант 5

Выполнил:
Студент Вяткин Н. А.
Группа О722Б

Преподаватель:
Гладевич А. А.

Санкт-Петербург
2023 г.

Уровень сложности – повышенный. Провести сравнение указанных алгоритмов сортировки массивов, содержащих *N1, N2, N3* и *N4* элементов, по указанному в вариативной части критерию и объему требуемой дополнительно памяти.

Дополнительно провести анализ того, как наличие повторяющихся ключей во входной последовательности влияет на трудоемкость каждого из рассматриваемых алгоритмов сортировки.

Порядок: по возрастанию элементов. Методы: пузырька, шейкера, быстрая сортировка, сортировка естественным слиянием. N1=20000, N2=50000, N3=80000, N4=110000. Критерий – количество присваиваний.

1) Алгоритм сортировки – сортировка пузырьком: повторяем проходы по массиву, сдвигая каждый раз наименьший элемент оставшейся последовательности к левому концу массива.

Трудоемкость сортировки пузырьком по количеству присваиваний:

Случай	**Ситуация, соответствующая случаю**	**Обоснование**	**Ожидаемое число присваиваний элементов массива**	**Асимптотическая оценка сложности по количеству присваиваний**	**Ожидаемое число вспомогательных присваиваний**
наилучший	массив упорядочен	перемещение элементов не требуется, но требуется пройтись по всем элементам массива	0 N1: 0 N2: 0 N3: 0 N4: 0	$\Omega(n^2)$	$1/2(n^2+3n-2)$ N1:200029999 N2:1250074999 N3:3200119999 N4:6050164999
наихудший	массив упорядочен в обратном порядке	Требуется поочередно перемещать меньший элемент к началу, пройтись по всем элементам массива	$3/2(n^2-n)$ N1: 599970000 N2:3749925000 N3: 9599880000 N4: 18149835000	$O(n^2)$	$1/2(n^2+3n-2)$ N1:200029999 N2:1250074999 N3:3200119999 N4:6050164999
средний	неупорядоченный массив	Требуется пройтись по массиву, находя меньшие элементы перемещать их на нужные места, пройтись по всем элементам массива	$3/4(n^2-n)$ N1: 299985000 N2: 1874962500 N3: 4799940000 N4: 9074917500	$O(n^2)$	$1/2(n^2+3n-2)$ N1:200029999 N2:1250074999 N3:3200119999 N4:6050164999

Пространственная сложность – O(1) (две переменных цикла и вспомогательная переменная).

Сортировка пузырьком является устойчивой.

При повторениях ключей количество присваиваний должно уменьшаться, так как

элементы с одинаковыми значениями менять местами не надо. Следовательно, с увеличением числа повторов ключей сортировка массива, упорядоченного в обратном направлении, должна стать не более трудоемкой, чем сортировка неупорядоченного массива.

2) Алгоритм шейкерной сортировки: разновидность пузырьковой сортировки. Отличается тем, что просмотры элементов выполняются один за другим в противоположных направлениях, при этом большие элементы стремятся к концу массива, а маленькие - к началу.

Трудоемкость шейкерной сортировки по количеству присваиваний:

Случай	**Ситуация, соответствующая случаю**	**Обоснование**	**Ожидаемое число присваиваний элементов массива**	**Асимптотическая оценка сложности по количеству присваиваний**	**Ожидаемое число вспомогательных присваиваний**
наилучший	массив упорядочен	перемещение элементов не требуется, но требуется пройтись по всем элементам массива	0 N1: 0 N2: 0 N3: 0 N4: 0	Ω(n)	n+6 N1: 20006 N2: 50006 N3: 80006 N4: 110006
наихудший	массив упорядочен в обратном порядке	требуется переместить все элементы, поочередно меняя местами элементы, проходя справа налево, и слева направо	$3/2*(n^2 -n) -3$ N1: 599969997 N2: 3749924997 N3: 9599879997 N4:18149834997	$O(n^2)$	n^2+n-3 N1: 400019997 N2: 2500049997 N3: 6400079997 N4: 12100109997
средний	неупорядоченный массив	требуется проходить слева направо, перемещая меньшие элементы, и затем проходить справа, налево также перемещая элементы, и так далее пока не будет отсортировано	$3/4*(n^2-n)$ N1: 299985000 N2: 1874962500 N3: 4799940000 N4: 9074917500	$O(n^2)$	$(n^2+n-3)/2$ N1: 200009999 N2: 1250024999 N3: 3200039999 N4: 6050054999

Пространственная сложность – O(1), (три переменных цикла и две вспомогательные переменные).

Шейкерная сортировка является устойчивой.

При повторениях ключей количество присваиваний должно уменьшаться, так как элементы с одинаковыми значениями менять местами не надо. Следовательно, с увеличением числа повторов ключей сортировка массива, упорядоченного в обратном направлении, должна стать не более трудоемкой, чем сортировка неупорядоченного массива.

3) Алгоритм быстрой сортировки: Выбрать опорный элемент, произвести перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Трудоемкость сортировки быстрой сортировка по количеству присваиваний:

Случай	**Ситуация, соответствующая случаю**	**Обоснование**	**Ожидаемое число присваиваний элементов массива**	**Асимптотическая оценка сложности по количеству присваиваний**	**Ожидаемое число вспомогательных присваиваний**
Средний	Не каждый опорный элемент является медианой сортируемой части. Массив упорядочен по возрастанию.	Массив уже упорядочен, поэтому будет происходить только само обмен при i=j, количество вызовов зависит только от выбора опорного элемента	½*n*log(n/6) N1: 35228 N2: 98020 N3: 164997 N4: 234478	O(nlogn)	$n*\log_2 n+n$ N1: 305755 N2: 830482 N3: 1383016 N4: 1952185
Средний	Не каждый опорный элемент является медианой сортируемой части. Массив упорядочен по убыванию.	Количество вызовов функции зависит только от выбора опорного элемента, и нам необходимо поменять местам все элементы относительно опорного в каждом проходе.	4,6*(n/6)*logn N1: 65949 N2: 180127 N3: 300722 N4: 425157	O(nlogn)	$n*\log_2 n+n$ N1: 305755 N2: 830482 N3: 1383016 N4: 1952185
Средний	Неупорядоченный массив, последовательность делится так, что не каждый	Количество вызовов функции зависит только от выбора опорного элемента и от количества элементов в	3,15*n*log(n/6) N1: 221941 N2: 617528 N3: 1039480 N4: 1477210	O(nlogn)	$(n*\log_2 n+n)*3/2$ N1: 458632 N2: 1245723 N3: 2074524 N4: 2928277

	опорный элемент является медианой	последовательности и ее самой, нам нужно искать элементы для обмена.			
Наилучший	Каждый опорный элемент является медианой сортируемой части, массив упорядочен по возрастанию	Потребуется $\log_2 n$ проходов для сортировки массива, и массив упорядочен, поэтому будут происходить само обмены при i=j	n+n*1/2 N1: 30000 N2: 750000 N3: 120000 N4: 1650000	O(nlogn)	$n*\log_2 n+n$ N1: 305755 N2: 830482 N3: 1383016 N4: 1952185
Наилучший	Каждый опорный элемент является медианой сортируемой части, массив упорядочен по убыванию.	Потребуется $\log_2 n$ проходов для сортировки массива, меняем местами элементы с разных концов, так как массив упорядочен по убыванию.	2*(n+n*1/2) N1: 60000 N2: 150000 N3: 240000 N4: 330000	O(nlogn)	$n*\log_2 n+n$ N1: 305755 N2: 830482 N3: 1383016 N4: 1952185
Наихудший	Каждый опорный элемент является минимальным или максимальным в сортируемой части.	Массив будет разбиваться на части из 1 элемента и n-1. Потребуется n разделений	$n^2/4$ N1: 100000000 N2: 625000000 N3: 1600000000 N4: 3025000000	$O(n^2)$	n^2 N1: 400000000 N2: 2500000000 N3: 6400000000 N4: 12100000000

Пространственная сложность – O(logn) (две переменные цикла и две вспомогательные переменные, рекурсивные вызовы функции).

Быстрая сортировка является не устойчивой.

При увеличении повторов ключей количество присваиваний особо не изменится, по сравнению с случаем без повторов, так как, алгоритм зависит только от выбора опорного элемента, а не от самой последовательности элементов.

4) Алгоритм сортировки естественным слиянием:

Трудоемкость сортировки естественным слиянием по количеству присваиваний:

Используются два вспомогательных массива, элементы из основного массива

последовательно считываются в вспомогательные. Пока последовательно считываемые числа упорядочены, они записываются в один вспомогательный массив. Как только заканчивается отсортированный подмассив, числа записываются в другой вспомогательный массив, как только считаны все числа из основного массива в вспомогательные(подмассива) массивы, начинается обработка вспомогательных массивов, при этом сравниваются элементы из вспомогательных массивов между собой, по результату сравнений, в основной массив записывается меньший элемент, сравнения происходят в пределах вспомогательных массивов, это все повторяется до тех пор пока массив не будет полностью упорядочен.

Случай	**Ситуация, соответствующая случаю**	**Обоснование**	**Ожидаемое число присваиваний элементов массива**	**Асимптотическая оценка сложности по количеству присваиваний**	**Ожидаемое число вспомогательных присваиваний**
наилучший	массив упорядочен	Нам не надо ничего записывать в вспомогательные серии и потом соединять все обратно в нужном порядке.	0 N1: 0 N2: 0 N3: 0 N4: 0	O(n)	n+10 N1: 20010 N2: 50010 N3: 80010 N4: 110010
наихудший	массив упорядочен в обратном порядке	Массив будет разбиваться много раз на фрагменты из 1 (максимальная серия состоит из 1 элемента) элемента и число проходов будет равно $\log_2 n$	16n при n <=50000 18n при n> 50000 N1: 320000 N2: 800000 N3: 1440000 N4: 1980000	O(nlogn)	16n*2,75 при n <50001 18n*2,75 при n> 50000 N1: 880000 N2: 2200000 N3: 3960000 N4: 5445000
средний	неупорядоченный массив	Массив будет разбиваться на серии, но эти серии будут разных размеров, и не все из одного элемента.	14n при n <=20000 16n при n>50000 N1: 280000 N2: 800000 N3: 1280000 N4: 1760000	O(nlogn)	17n*2,5 при n <50001 18n*2,5 при n> 50000 N1: 850000 N2: 2125000 N3: 3600000 N4: 4950000

Пространственная сложность – O(n) (5 переменных для индексов, две переменные для флагов, вспомогательный массив, указатель на исходный массив).

Сортировка естественным слиянием является неустойчивой.

Сортировка происходит по средству сравнения элементов, поэтому при увеличение повторяющихся ключей, количество присваиваний будет уменьшаться. Следовательно, сортировка упорядоченного массива по убыванию с повторениями с увеличение повторяющихся ключей будет менее трудоемкой чем сортировка неупорядоченного

массива.

Текст программы:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <chrono>

using namespace std;

void CreateArr(ifstream &, unsigned long int, unsigned long int *&); // создание массива
void CreateFile(unsigned long int *, unsigned long int, string);      // создание файла из массива

void BubbleSort(unsigned long int *, unsigned long int, unsigned long long int &, unsigned long long int &, unsigned long int &, unsigned long int &);       // сортировка пузырьком
void ShakerSort(unsigned long int *, unsigned long int, unsigned long long int &, unsigned long long int &, unsigned long int &, unsigned long int &);       // сортировка шейкера
void QuickSort(unsigned long int *, unsigned long int, unsigned long long int &, unsigned long long int &, unsigned long int &, unsigned long int &);        // быстрая сортировка
void NaturalMergeSort(unsigned long int *, unsigned long int, unsigned long long int &, unsigned long long int &, unsigned long int &, unsigned long int &); // сортировка естесвенным слиянием

struct Arrays
{
    unsigned long int *arr;
    unsigned long int N;
    string Type;
    string Repeat;
    string Name;
};

// создание массивов из файлов
void CreateArraysFromFiles(string, string, string, string, struct Arrays *, int);
void CreateStruct(string *, string *, struct Arrays *, unsigned long int *);
// использования сортировок для нескольких массивов
void UseSort(struct Arrays *, void (*)(unsigned long int *, unsigned long int, unsigned long long int &, unsigned long long int &, unsigned long int &, unsigned long int &));

int main()
{
    setlocale(LC_ALL, "");
     unsigned long int size[4] = {20000, 50000, 80000, 110000}; // размерность массивов
    string Types[3] = {
        "По возврастанию",
        "По убыванию",
        "Не упорядоченное",
    }; // типы файлов
        string Names[24] = {"AN1elements.txt", "AN2elements.txt", "AN3elements.txt", "AN4elements.txt", "DN1elements.txt", "DN2elements.txt", "DN3elements.txt", "DN4elements.txt", "N1elements.txt", "N2elements.txt", "N3elements.txt", "N4elements.txt", "R10A.txt", "R100A.txt", "R500A.txt", "R1000A.txt", "R10D.txt", "R100d.txt", "R500D.txt", "R1000D.txt", "R10.txt", "R100.txt", "R500.txt", "R1000.txt"};
```

```
        struct Arrays Arr[24]; // структура массивов
        int menu = 0;
        CreateStruct(Names, Types, Arr, size);

        cout << "Выберите сортировку для тестирования" << endl
             << "1. Сортировка пузырьком" << endl
             << "2. Сортировка шейкера" << endl
             << "3. Быстрая сортировка" << endl
             << "4. Сортировка естественным слиянием" << endl;
        do
        {
            cout << "- ";
            cin >> menu;
            switch (menu)
            {
            case 1:
            {

                cout << "Тип сортировки: Сортировка пузырьком" << endl;
                UseSort(Arr, BubbleSort);
                break;
            }
            case 2:
            {
                cout << "Тип сортировки: Шейкерная сортировка" << endl;
                UseSort(Arr, ShakerSort);
                break;
            }
            case 3:
            {
                cout << "Тип сортировки: Быстрая сортировка" << endl;
                UseSort(Arr, QuickSort);
                break;
            }
            case 4:
            {
                cout << "Тип сортировки: Сортировка естественным слиянием" <<
endl;
                UseSort(Arr, NaturalMergeSort);
                break;
            }
            }
        } while (menu <= 0 || menu > 4);
        for (int i = 0; i < 24; i++)
            delete[] Arr[i].arr;
        return 0;
    }
    void CreateStruct(string *s, string *type, struct Arrays *Arr, unsigned
long int *size)
    {
        int i, j, l = 0;
        ifstream in;
        for (i = 0, j = 0; i < 24; i++, j++)
        {
            if (j == 4)
            {
                j = 0;
                l++;
            }
            if (i == 12)
            {
                for (l = 0; l < 4; l++)
                    size[l] = 110000;
```

```
                l = 0;
            }
            Arr[i].N = size[j];
            Arr[i].Name = "t" + to_string(i);
            Arr[i].Type = type[l];
            in.open(s[i]);
            CreateArr(in, Arr[i].N, Arr[i].arr);
            if (i >= 12)
                Arr[i].Repeat = "С повторениями";
            else
                Arr[i].Repeat = "Без повторений";
            in.close();
        }
    }
    // вызываем поочередно для всех массивов выбранную сортировку
    void UseSort(struct Arrays *Arr, void (*Sort)(unsigned long int *a,
unsigned long int n, unsigned long long int &cAs, unsigned long long int &cAu,
unsigned long int &cS, unsigned long int &cMS))
    {
        unsigned long long int countAssign, countAuxiliary;
        unsigned long int size = 0, msize = 0;
         cout << "| Количество элементов массива | Тип элементов в массиве
|    Тип файла    | Количество присваиваний элементов массива | Количество
вспомогательных присваиваний | Время сортировки | Объем выделенной памяти |" <<
endl;
        cout << "----------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------" << endl;
        for (int i = 0; i < 24; i++)
        {
            countAssign = 0;
            countAuxiliary = 0;
            size = 0;
            msize = 0;
            auto start = std::chrono::steady_clock::now(); // начало отсчета
              Sort(Arr[i].arr, Arr[i].N, countAssign, countAuxiliary, size,
msize);
            auto end = std::chrono::steady_clock::now(); // конец отсчета
           auto t = std::chrono::duration_cast<std::chrono::milliseconds>(end
- start);

            cout << "|" << setw(30) << Arr[i].N << "|";
            cout << setw(25) << Arr[i].Type << "|";
            cout << setw(15) << Arr[i].Repeat << "|";
            cout << setw(43) << countAssign << "|";
            cout << setw(41) << countAuxiliary << "|";
            cout << setw(18) << t.count() / 1000. << "|";
            cout << setw(25) << msize << "|" << endl;
            cout << "----------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------"    <<
endl;
            CreateFile(Arr[i].arr, Arr[i].N, Arr[i].Name);
        }
    }

    // считать нужное количество чисел из файла
    void CreateArr(ifstream &file, unsigned long int N, unsigned long int
*&Arr)
    {
        if (file.is_open())
        {
            Arr = new (nothrow) unsigned long int[N];
```

```
            if (!Arr)
            {
                cout << "Ошибка выделения памяти" << endl;
                return;
            }
            for (int i = 0; i < N; i++)
                file >> Arr[i];
        }
        else
            cout << "Файл не найден" << endl;
    }

    // создать файл с массива
    void CreateFile(unsigned long int *Arr, unsigned long int N, string s)
    {
        ofstream file(s);
        if (file.is_open())
            for (int i = 0; i < N; i++)
                file << Arr[i] << " ";
        else
            cout << "Не удалось создать файл" << endl;
        file.close();
    }
    // сортировка пузырьком
    void BubbleSort(unsigned long int *arr, unsigned long int n, unsigned long
long int &n1, unsigned long long int &n2, unsigned long int &size, unsigned long
int &msize)
    {
        int i, j;
        unsigned long int x;
        for (++n2, i = 1; i < n; ++n2, i++)
            for (++n2, j = n - 1; j >= i; ++n2, j--)
                if (arr[j - 1] > arr[j])
                {
                    x = arr[j - 1];
                    arr[j - 1] = arr[j];
                    arr[j] = x;
                    n1 += 3;
                }
        size = sizeof(i) + sizeof(j) + sizeof(x);
        msize = size;
    }

    // сортировка шейкера
    void ShakerSort(unsigned long int *a, unsigned long int n, unsigned long
long int &n1, unsigned long long int &n2, unsigned long int &size, unsigned long
int &msize)
    {
        unsigned long long int j, k = n - 1, left = 1, right = n - 1, x;
        n2 += 3;
        do
        {
           for (++n2, j = right; j >= left; ++n2, j--) /*сначала просматриваем
справа налево*/
                if (a[j - 1] > a[j])
                {
                    x = a[j - 1];
                    a[j - 1] = a[j];
                    a[j] = x;
                    n1 += 3;
                    k = j;
                    ++n2;
                }
```

```
            left = k + 1;
            ++n2;
           for (++n2, j = left; j <= right; ++n2, j++) /*а теперь просматриваем
слева направо*/
                if (a[j - 1] > a[j])
                {
                    x = a[j - 1];
                    a[j - 1] = a[j];
                    a[j] = x;
                    n1 += 3;
                    k = j;
                    ++n2;
                }
            right = k - 1;
            ++n2;
          } while (left < right); /*и так до тех пор, пока есть что
просматривать*/
         size = sizeof(j) + sizeof(k) + sizeof(left) + sizeof(right) +
sizeof(x);
        msize = size;
    }

    // быстрая сортировка
    void QuickSort(unsigned long int *a, unsigned long int n, unsigned long
long int &n1, unsigned long long int &n2, unsigned long int &size, unsigned long
int &msize)
    {
        unsigned long long int x, w, i, j;
        x = a[n / 2]; /* опорный элемент */
        ++n2;
        i = 0;
        ++n2;
        j = n - 1;
        ++n2;
        size += sizeof(x) + sizeof(w) + sizeof(i) + sizeof(j);
        if (size > msize)
            msize = size;
        do
        {
            while (a[i] < x)
            {
                i++; /* слева находим элемент больше опорного */
                ++n2;
            }
            while (x < a[j])
            {
                j--; /* справа находим элемент меньше опорного */
                ++n2;
            }
            if (i <= j)
            {
                w = a[i];
                ++n1;
                a[i] = a[j];
                ++n1;
                a[j] = w; /* меняем найденные элементы местами */
                ++n1;
                i++;
                ++n2;
                j--;
                ++n2;
            }
        } while (i < j); /* по завершении цикла слева от опорного элемента
```

```
ключи, меньшие x, а справа - большие */
          /* сортируем так же левую и правую части */
          if (j > 0)
              QuickSort(a, j + 1, n1, n2, size, msize);
          if (i < n - 1)
              QuickSort(a + i, n - i, n1, n2, size, msize);
          size -= size;
      }

      // сортировка естественным слиянием
      void NaturalMergeSort(unsigned long int *a, unsigned long int n, unsigned
long long int &n1, unsigned long long int &n2, unsigned long int &size, unsigned
long int &msize)
      {
          int split; /* индекс, по которому делим массив */
          int end;
          unsigned long int *p = a, *tmp;
          char flag = 0, sorted = 0;
          int pos1, pos2, pos3;
          tmp = (unsigned long int *)malloc(n * sizeof(unsigned long int));
           size = sizeof(split) + sizeof(end) + sizeof(p) + sizeof(flag) +
sizeof(sorted) + sizeof(pos1) + sizeof(pos2) + sizeof(pos3) + sizeof(tmp) * n;
          msize = size;
          n2 += 4;
          do /* если есть более 1 элемента */
          {
              end = n;
              ++n2;
              pos2 = pos3 = 0;
              n2 += 2;
              do
              {
                  p += pos2;
                  ++n2;
                  end = n - pos3;
                  ++n2;
                  for (++n2, split = 1; split < end && p[split - 1] <= p[split];
++n2, split++)
                      ; /*первая серия*/
                  if (split == n)
                  {
                      sorted = 1;
                      ++n2;
                      break;
                  }
                  pos1 = 0;
                  ++n2;
                  pos2 = split;
                  ++n2;
                   while (pos1 < split && pos2 < end) /*идет слияние, пока есть
хоть один элемент в каждой серии*/
                  {
                      if (p[pos1] < p[pos2])
                      {
                          tmp[pos3++] = p[pos1++];
                          ++n1;
                          n2 += 2;
                      }
                      else
                      {
                          tmp[pos3++] = p[pos2++];
                          ++n1;
                          n2 += 2;
```

```
                        if (p[pos2] < p[pos2 - 1])
                            break;
                    }
                }
                 /* одна последовательность закончилась - копировать остаток
другой в конец буфера */
                while (pos2 < end && tmp[pos3 - 1] <= p[pos2]) /* пока вторая
последовательность не пуста */
                {
                    tmp[pos3++] = p[pos2++];
                    ++n1;
                    n2 += 2;
                }
               while (pos1 < split) /* пока первая последовательность не пуста
*/
                {
                    tmp[pos3++] = p[pos1++];
                    ++n1;
                    n2 += 2;
                }
            } while (pos3 < n);
            if (sorted)
                break;
            p = tmp;
            n2++;
            tmp = a;
            n2++;
            a = p;
            n2++;
            flag = !flag;
            ++n2;
        } while (split < n);
        if (flag)
        {
            for (++n2, pos1 = 0; pos1 < n; ++n2, pos1++)
            {
                tmp[pos1] = a[pos1];
                n1++;
            }
            free(a);
        }
        else
            free(tmp);
    }
```

Результаты работы программы:

Тип сортировки: Сортировка пузырьком

Количество элементов массива	Тип элементов в массиве	Тип файла	Количество присваиваний элементов массива	Количество вспомогательных присваиваний	Время сортировки	Объем выделенной памяти
20000	По возрастанию	Без повторений	0	200029999	0.652	12
50000	По возрастанию	Без повторений	0	1250074999	3.567	12
80000	По возрастанию	Без повторений	0	3200119999	8.554	12
110000	По возрастанию	Без повторений	0	6050164999	16.616	12
20000	По убыванию	Без повторений	599970000	200029999	1.03	12
50000	По убыванию	Без повторений	3749925000	1250074999	6.388	12
80000	По убыванию	Без повторений	9599880000	3200119999	16.227	12
110000	По убыванию	Без повторений	18149835000	6050164999	30.704	12
20000	Не упорядоченное	Без повторений	298672155	200029999	1.317	12
50000	Не упорядоченное	Без повторений	1872093714	1250074999	8.583	12
80000	Не упорядоченное	Без повторений	4743815133	3200119999	27.919	12
110000	Не упорядоченное	Без повторений	8877739059	6050164999	41.438	12
110000	По возрастанию	С повторениями	0	6050164999	16.18	12
110000	По возрастанию	С повторениями	0	6050164999	16.32	12
110000	По возрастанию	С повторениями	0	6050164999	16.612	12
110000	По возрастанию	С повторениями	0	6050164999	16.308	12
110000	По убыванию	С повторениями	18148350000	6050164999	30.425	12
110000	По убыванию	С повторениями	18133500000	6050164999	30.617	12
110000	По убыванию	С повторениями	18067500000	6050164999	30.206	12
110000	По убыванию	С повторениями	17985000000	6050164999	30.224	12
110000	Не упорядоченное	С повторениями	9082273053	6050164999	43.623	12
110000	Не упорядоченное	С повторениями	9093250236	6050164999	43.395	12
110000	Не упорядоченное	С повторениями	9027195807	6050164999	41.298	12
110000	Не упорядоченное	С повторениями	8991932838	6050164999	41.117	12

Рисунок 1 – Сортировка пузырьком

Тип сортировки: Шейкерная сортировка

Количество элементов массива	Тип элементов в массиве	Тип файла	Количество присваиваний элементов массива	Количество вспомогательных присваиваний	Время сортировки	Объем выделенной памяти
20000	По возрастанию	Без повторений	0	20006	0	40
50000	По возрастанию	Без повторений	0	50006	0	40
80000	По возрастанию	Без повторений	0	80006	0	40
110000	По возрастанию	Без повторений	0	110006	0	40
20000	По убыванию	Без повторений	599969997	400019997	1.756	40
50000	По убыванию	Без повторений	3749924997	2500049997	10.162	40
80000	По убыванию	Без повторений	9599879997	6400079997	25.64	40
110000	По убыванию	Без повторений	18149834997	12100109997	48.89	40
20000	Не упорядоченное	Без повторений	298672155	232752236	1.231	40
50000	Не упорядоченное	Без повторений	1872093714	1456017717	8.007	40
80000	Не упорядоченное	Без повторений	4743815133	3692706499	20.088	40
110000	Не упорядоченное	Без повторений	8877739059	6911988237	36.626	40
110000	По возрастанию	С повторениями	0	110006	0	40
110000	По возрастанию	С повторениями	0	110006	0	40
110000	По возрастанию	С повторениями	0	110006	0	40
110000	По возрастанию	С повторениями	0	110006	0	40
110000	По убыванию	С повторениями	18148349997	12099614997	49.124	40
110000	По убыванию	С повторениями	18133499997	12094664997	48.669	40
110000	По убыванию	С повторениями	18067499997	12072664997	48.705	40
110000	По убыванию	С повторениями	17984999997	12045164997	48.025	40
110000	Не упорядоченное	С повторениями	9082273053	7061207452	36.992	40
110000	Не упорядоченное	С повторениями	9093250236	7074548625	36.811	40
110000	Не упорядоченное	С повторениями	9027195804	7035242624	36.674	40
110000	Не упорядоченное	С повторениями	8991932838	7012487819	37.315	40

Рисунок 2 – Шейкерная сортировка

Тип сортировки: Быстрая сортировка

Количество элементов массива	Тип элементов в массиве	Тип файла	Количество присваиваний элементов массива	Количество вспомогательных присваиваний	Время сортировки	Объем выделенной памяти
20000	По возврастанию	Без повторений	35424	306288	0	448
50000	По возврастанию	Без повторений	98301	848316	0.003	480
80000	По возврастанию	Без повторений	141696	1385106	0.007	512
110000	По возврастанию	Без повторений	196605	1956621	0.001	512
20000	По убыванию	Без повторений	65424	306305	0.004	448
50000	По убыванию	Без повторений	173298	848329	0.002	480
80000	По убыванию	Без повторений	261696	1385125	0.012	512
110000	По убыванию	Без повторений	361602	1956635	0.007	512
20000	Не упорядоченное	Без повторений	221697	453931	0.011	576
50000	Не упорядоченное	Без повторений	589089	1293186	0.01	576
80000	Не упорядоченное	Без повторений	1007469	1935988	0.016	544
110000	Не упорядоченное	Без повторений	1410135	2816533	0.023	544
110000	По возврастанию	С повторениями	725997	2575148	0.002	544
110000	По возврастанию	С повторениями	1230897	2417541	0.001	544
110000	По возврастанию	С повторениями	1502817	2256983	0.008	512
110000	По возврастанию	С повторениями	1667487	2256293	0.014	512
110000	По убыванию	С повторениями	890994	2575139	0.005	544
110000	По убыванию	С повторениями	1395894	2417485	0.001	544
110000	По убыванию	С повторениями	1667814	2256729	0.005	512
110000	По убыванию	С повторениями	1832484	2255789	0.01	512
110000	Не упорядоченное	С повторениями	1669029	2850244	0.023	512
110000	Не упорядоченное	С повторениями	1995792	2710641	0.018	576
110000	Не упорядоченное	С повторениями	2213292	2787285	0.009	512
110000	Не упорядоченное	С повторениями	2309688	2662593	0.032	512

Рисунок 3 – Быстрая сортировка

Тип сортировки: Сортировка естественным слиянием

Количество элементов массива	Тип элементов в массиве	Тип файла	Количество присваиваний элементов массива	Количество вспомогательных присваиваний	Время сортировки	Объем выделенной памяти
20000	По возврастанию	Без повторений	0	20010	0	80026
50000	По возврастанию	Без повторений	0	50010	0.001	200026
80000	По возврастанию	Без повторений	0	80010	0	320026
110000	По возврастанию	Без повторений	0	110010	0	440026
20000	По убыванию	Без повторений	320000	880920	0.001	80026
50000	По убыванию	Без повторений	800000	2267634	0.007	200026
80000	По убыванию	Без повторений	1440000	3923286	0.002	320026
110000	По убыванию	Без повторений	1980000	5367090	0.012	440026
20000	Не упорядоченное	Без повторений	280000	770893	0.001	80026
50000	Не упорядоченное	Без повторений	800000	2092624	0.01	200026
80000	Не упорядоченное	Без повторений	1280000	3483585	0.01	320026
110000	Не упорядоченное	Без повторений	1760000	4762284	0.01	440026
110000	По возврастанию	С повторениями	0	110010	0.001	440026
110000	По возврастанию	С повторениями	0	110010	0.003	440026
110000	По возврастанию	С повторениями	0	110010	0	440026
110000	По возврастанию	С повторениями	0	110010	0	440026
110000	По убыванию	С повторениями	1540000	4058080	0.001	440026
110000	По убыванию	С повторениями	1320000	3312508	0.006	440026
110000	По убыванию	С повторениями	880000	2338950	0.002	440026
110000	По убыванию	С повторениями	880000	2173504	0.001	440026
110000	Не упорядоченное	С повторениями	1760000	4761873	0.014	440026
110000	Не упорядоченное	С повторениями	1760000	4761532	0.018	440026
110000	Не упорядоченное	С повторениями	1760000	4762308	0.023	440026
110000	Не упорядоченное	С повторениями	1760000	4761558	0.02	440026

Рисунок 4 – Сортировка естественным слиянием

Анализ полученных данных:

1. Пузырьковая сортировка

1) Рассчитанные результаты совпадают с результатами работы программы в наилучшем и худшем случае, в среднем случае совпадает только количество вспомогательных присваиваний, так как количество основных присваиваний зависит от последовательности.

2) Количество основных присваиваний меньше вспомогательных только в наилучшем случае, так как нам не надо менять местами элементы. В остальных случаях, количество основных присваиваний больше количества вспомогательных, так как в наихудшем случае нам нужно поменять местами n элементов, а среднем случае как минимум n/2, на что понадобится больше присваиваний чем на проход по всему массиву, количество вспомогательных присваиваний меньше основных присваиваний примерно в 3 раза в наихудшем случае, и в 1,5 раза в среднем случае. При увеличении количества элементов, количество вспомогательных и основных присваиваний также увеличится, с таким же соотношением, в зависимости от упорядоченности массива.

3) Во всех случаях асимптотика сортировки пузырьком является $O(n^2)$, что хуже чем в других сортировках, такую же асимптотику имеет сортировка шейкером в среднем и наихудшем случае, и быстрая сортировка в наихудшем случае. Количество основных присваиваний по сравнению с шейкерной сортировкой совпадает в среднем и наихудшем случае, а по сравнению с быстрой сортировок, количество основных присваиваний больше в наихудшем и среднем случае.

4) Пузырьковая сортировка обладает наихудшими показателями по времени, где время варьируется от 0,6с до 41с, худшее время сортировки программа показывает при сортировке неупорядоченного массива, так как время тратится на поиск меньших элементов, и обмен элементов в последовательности. По сравнению с другими алгоритмами пузырьковая сортировка показывает худшие показатели по времени, но в наихудшем случае, работает быстрее пузырьковой сортировки в наихудшем случае. Эта сортировка имеет худшие показатели из всех четырёх, и за время выполнения программы делает большее количество присваиваний в наилучшем случае, чем остальные три алгоритма, и также в остальных случаях по сравнению с быстрой сортировкой и естественным слиянием.

5) Пространственная сложность O(1), что означает малую затратность алгоритма по памяти, он задействует самое небольшое количество памяти по сравнению с другими алгоритмами, но именно поэтому он является самым медленным по времени.

6) При повторяющихся ключах прогноз не совпал с поведением программы, количество присваиваний элементов массива и время сортировки уменьшилось, но не

значительно. Это связано с тем, что одинаковые элементы менять местами не надо, но при сортировке с повторениями нам надо поменять все большие элементы с меньшими, нужное количество раз в зависимости от количества повторений. Также при увлечении количества повторяющихся ключей, сортировка упорядоченного по убыванию массива не становится менее трудоемкой, чем сортировка неупорядоченного массива, так как количество присваиваний не изменится так, чтобы это было правдой, так как нам надо делать все равно столько же вспомогательных присваиваний, и меньше основных присваиваний только в зависимости от количества повторений.

2. Шейкерная сортировка

1) Рассчитанные результаты совпадают с результатами работы программы в наилучшем и худшем случае, в среднем случае результаты не совпали, так как количество присваиваний зависит от входной последовательности.

2) Количество основных присваиваний меньше вспомогательных только в наилучшем случае, так как нам не надо менять местами элементы, в других случаях количество основных присваиваний больше вспомогательных присваиваний. В остальных случаях нам надо поменять местами все элементы, и пройтись справа, налево перемещая все наименьшие элементы и потом слева направо пройтись также перемещая, поэтому количество основных присваиваний будет больше вспомогательных, разница в случаях только, от количества элементов которое необходимо поменять местами. Количество вспомогательных присваиваний отличается от основных на n в лучшем случае, в наихудшем случае отличается почти в 1,5 раза, в среднем случае количество основных присваиваний больше вспомогательных примерно в 1, 3 раза. При увеличении количества элементов в массиве, соотношение основных и вспомогательных присваиваний будет зависеть только от случая.

3) В лучшем случае, алгоритм обладает хорошей асимптотикой O(n), что является более хорошим результатом, чем в пузырьковой сортировке и быстрой, и таким же как сортировке естественным слиянием. Поэтому количество присваиваний в шейкерной сортировке в лучшем случае меньше чем в пузырьковой сортировке и быстрой сортировке, и такое же как в сортировке естественным слиянием. В среднем и худшем случае асимптотика $O(n^2)$, что означает, что алгоритм в этих случаях достаточно трудоемкий, по трудоёмкости он такой же как все случаи в пузырьковой сортировке, и быстрой сортировке в наихудшем случае, но количество присваиваний в наихудшем случае в шейкерной сортировке больше чем в других алгоритмах.

4) Время выполнения шейкерной сортировки в зависимости от случая изменяется от 0с до 37с, в наилучшем случае время минимальное из того что требуется всего один

проход по массиву. По сравнению с другими алгоритмами данный является более быстрым, чем сортировка пузырьком, но медленнее быстрой сортировки и естественным слиянием. За время выполнения алгоритм в среднем и наихудшем случае, выполняет большее количество присваиваний, чем другие алгоритмы.

5) Пространственная сложность O(1), следовательно алгоритм не задействует много памяти, в связи с этим с увеличением количества элементов, время выполнения сортировки растет. Задействует меньше памяти по сравнению с быстрой сортировкой и естественным слиянием. По памяти алгоритм является одним из лучших из этих четырех.

6) При повторяющихся ключах прогноз не совпал, количество присваиваний элементов уменьшилось, но не значительно, время работы программы в наихудшем случае при повторяющихся ключах изменилось чуть больше, чем на секунду. При увеличении повторяющихся ключей шейкерная сортировка становится менее трудоемкой, но не значительно, поэтому нельзя считать ее менее трудоемкой сортировки неупорядоченного массива.

3. Быстрая сортировка.

1) Рассчитанное количество основных и вспомогательных присваиваний не совпало с результатом работы программы, так как быстрая сортировка зависит от выбора опорного элемента и разделения исходного массива на части.

2) Количество основных присваиваний меньше количества вспомогательных присваиваний, примерно в 9 раз в среднем случае, когда массив упорядочен по возрастанию, в данном случае происходит обмен элементов массива, когда i=j, и элемент массива меняется сам собой, когда массив упорядочен в обратном порядке количество основных присваиваний меньше вспомогательных присваиваний примерно в 4,5 раза, когда массив не упорядочен меньше примерно в 2 раза. При увеличении количества элементов, количество присваиваний будет увеличиваться в зависимости от выбора опорного элемента (является ли он медианой).

3) Худшая трудоемкость алгоритма $O(n^2)$ достигается только при неудачных выборах опорного элемента (он максимальный или минимальный), в этом случае количество присваиваний будет наибольшим, но даже так количество присваиваний будет меньше чем в пузырьковой сортировке и сортировке шейкером, хотя эти алгоритмы имеют такую же трудоемкость в данном случае. В лучшем и среднем случае трудоемкость O(nlogn), что лучше, чем в пузырьковой сортировке в аналогичных случаях, но количество присваиваний в быстрой сортировке в разы меньше, при сравнений с шейкерной сортировкой, в лучшем случае уступает, а в среднем случае превосходит, при сравнение с сортировкой естественным слиянием количество вспомогательных присваиваний при неупорядоченном массиве меньше в быстрой сортировке при одинаковой трудоемкости, а в лучшем случае

наоборот, так как трудоемкость в быстрой сортировке хуже.

4) Время выполнения программы варьируется от 0с до 0,012с, что является одним из лучших показателей по времени среди этих четырёх сортировок, уступает только сортировке естественным слиянием при сортировке упорядоченного массива по возрастанию. В связи с небольшим временем выполнения программа производит меньше присваиваний, чем другие алгоритмы. Быстрая сортировка и время ее выполнения зависит от выбора опорного элемента.

5) Пространственная сложность O(logn), показывает, что алгоритм, более затратный по памяти, чем пузырьковая и шейкреная сортировка, и менее затратный, чем сортировка естественным слиянием.

6) Прогноз совпал, повторяющиеся ключи и их увеличение не влияет на трудоемкость, так как алгоритм меньше зависит от последовательности, чем от выбора опорного элемента, поэтому время выполнения изменяется не значительно.

4. Сортировка естественным слиянием

1) Рассчитанные результаты совпали с программными полностью для наилучшего случая, при наихудшем и среднем случае совпало только количество основных присваиваний, так как массив в любом случае нам нужно перезаписать полностью. Количество вспомогательных присваиваний не совпадает, так как зависит от распределения на серии.

2) Количество основных присваиваний меньше во всех случаях, в наилучшем случае на примерно на n, в наихудшем и среднем случае примерно в 2,75 раза. Количество вспомогательных присваиваний больше, так как происходит разделение на серии в циклах, а основные присваивания происходят при переписывании элементов массива уже в отсортированном порядке. При увеличении количества элементов в массиве, количество присваиваний также увеличится и соотношение основных и вспомогательных присваиваний останется тем же, в зависимости от случая.

3) По асимптотике алгоритм является лучшим из рассматриваемых, в лучшем случае O(n), трудоемкость минимально, количество присваиваний n, все остальные алгоритм при сортировке упорядоченного массива, показывают большую трудоемкость, кроме шейкерной сортировки. В наихудшем и среднем случае O(nlogn), лучшая трудоемкость из всех для этих случаев, но для неупорядоченного массива необходимое количество присваиваний больше чем у быстрой сортировки. Из-за меньшей трудоемкости, алгоритм совершает меньшее количество присваиваний, по сравнению с лучшим и худшим случаем быстрой сортировки, всеми случаями пузырьковой сортировки, и средним и худшим случаем шейкерной сортировки.

4) Время выполнения варьируется от 0с до 0,012с, что является одним из лучших показателей, выигрывает по времени у быстрой сортировки при сортировке упорядоченного по возрастанию массива, у пузырьковой и шейккерной во всех случаях. Минимальное время связано с меньшим количеством присваиваний.

5) Пространственная сложность алгоритма O(n), задействует некоторое количество памяти, большее чем пузырьковая, шейкерная и быстрая сортировка, в связи с этим можно утверждать, что данный алгоритм будет более быстрым, чем пузырьковая и шейкерная.

6) Прогноз совпал, количество присваиваний уменьшилось, время выполнение сортировки уменьшилось. В случае наличия одинаковых ключей обратно упорядоченная последовательность уже не является худшим случаем (ведь элементы с одинаковыми ключами будут стоять рядом и в этом случае), и наибольшая трудоемкость будет достигнута при случайном расположении ключей. В целом на наборах неуникальных ключей сортировка естественным слиянием будет производиться быстрее, чем на неповторяющихся ключах.

Вывод: по результатам проведенного анализа самым эффективным из рассмотренных алгоритмов по соотношению время-память является алгоритм быстрой сортировки.