МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра О7 Информационные системы и программная инженерия
шифр наименование кафедры, по которой выполняется работа

Дисциплина **Структуры и организация данных**
наименование дисциплины

УЧЕБНО-ПРАКТИЧЕСКАЯ РАБОТА №3

тема «Оценка эффективности алгоритмов»
часть 1 «Алгоритмы сортировки»
Вариант 7

ОБУЧАЮЩИЙСЯ

группы И934Б

Есипов М.А.
подпись фамилия и инициалы

12.05.2024
дата сдачи

ПРОВЕРИЛ

ученая степень, ученое звание, должность

Фамилия И.О.
подпись фамилия и инициалы

Оценка / балльная оценка

дата проверки

Санкт-Петербург
2024 г.

Уровень сложности – базовый. Провести сравнение указанных алгоритмов сортировки массивов, содержащих N1(10000), N2(18000), N3(30000) и N4(60000) элементов, по указанному в вариативной части критерию и объему требуемой дополнительно памяти.

Порядок: по убыванию элементов. Методы: пузырька, пузырька с фиксацией места обмена, шейкера, быстрая сортировка. N1=10000, N2=18000, N3=30000, N4=60000. Критерий – количество сравнений.

1) Алгоритм сортировки попарным обменом (пузырьковая):

Выполняется проход по массиву с сравнением и сменой мест пары соседних элементов. Этот процесс продолжается до тех пор, пока не будут упорядочены все элементы массива.

Трудоемкость сортировки попарным обменом по количеству сравнений:

Случай	**Ситуация, соответст-вующая случаю**	**Обоснование**	**Ожидаемое число сравнений элементов массива**	**Асимптотическая оценка сложности по количеству сравнений**	**Ожидаемое число вспомогат. сравнений**
наилучший	массив упорядочен	смена мест элементов не требуется	$(n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$	$O(n^2)$	$((n^2+ n)/2)-1$ N1: 50004999 N2: 162008999 N3: 450014999 N4: 1800029999
наихудший	массив упорядочен в обратном порядке	требуется поменять местами все элементы	$(n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$	$O(n^2)$	$((n^2+ n)/2)-1$ N1: 50004999 N2: 162008999 N3: 450014999 N4: 1800029999
средний	неупорядо-ченный массив	требуется поменять местами некоторые элементы	$(n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$	$O(n^2)$	$((n^2+ n)/2)-1$ N1: 50004999 N2: 162008999 N3: 450014999 N4: 1800029999

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

2) Алгоритм сортировки попарным обменом с фиксацией места обмена:

Сравнение и смена мест пары соседних элементов. Этот процесс продолжается до тех пор, пока не будут упорядочены все элементы, однако просмотры заканчиваются на определенном индексе k, левее которого все пары соседних элементов уже находятся в желаемом порядке.

Трудоемкость сортировки попарного обмена с фиксацией места обмена по

количеству сравнений:

Случай	**Ситуация, соответст-вующая случаю**	**Обоснование**	**Ожидаемое число присваиваний элементов массива**	**Асимптотическая оценка сложности по количеству присваиваний**	**Ожидаемое число вспомогат. присваиваний**
наилучший	элементы массива упорядочены	смена мест элементов не требуется	n-1 N1: 9999 N2: 17999 N3: 29999 N4: 59999	O(n)	n-1 N1: 9999 N2: 17999 N3: 29999 N4: 59999
наихудший	массив упорядочен в обратном порядке	требуется поменять местами все элементы	$((n^2+ n)/2)-1$ N1: 50004999 N2: 162008999 N3: 450014999 N4: 1800029999	$O(n^2)$	$((n^2+ n)/2)-1$ N1: 50004999 N2: 162008999 N3: 450014999 N4: 1800029999
средний	неупорядо-ченный массив	требуется поменять местами некоторые элементы	$\approx (n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$	$O(n^2)$	$\approx (n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$

Пространственная сложность – O(1) (переменная цикла, вспомогательная переменная для обмена и вспомогательная переменная для фиксации места последнего обмена).

3) Алгоритм сортировки шейкером:

Сравнение и смена мест пары соседних элементов. Этот процесс продолжается до тех пор, пока не будут упорядочены все элементы, при чем при проходе массива чередуются направления просмотра.

Трудоемкость сортировки шейкером по количеству сравнений:

Случай	**Ситуация, соответст-вующая случаю**	**Обоснование**	**Ожидаемое число сравнений элементов массива**	**Асимптотическая оценка сложности по количеству сравнений**	**Ожидаемое число вспомогат. сравнений**
наилучший	массив упорядочен	смена мест элементов не требуется, необходимо лишь 1 раз пройти по массиву	n-1 N1: 9999 N2: 17999 N3: 29999 N4: 59999	O(n)	n N1: 10000 N2: 18000 N3: 30000 N4: 60000
наихудший	массив упорядочен в обратном порядке	требуется поменять местами все элементы, на каждой итерации массив будет проходится в обе стороны, и	$(n^2- n)/2$ N1: $4.9995*10^7$ N2: $1.61991*10^8$ N3: $4.49985*10^8$ N4: $1.79997*10^9$	$O(n^2)$	$n^2/2$ N1: $5*10^7$ N2: $1.62*10^8$ N3: $4.5*10^8$ N4: $1.8*10^9$

		производится замены			
средний	массив не упорядочен	требуется поменять местами некоторые элементы	$\approx (n^2 - n\log_2 n)/2$ N1: $4.99336*10^7$ N2: $1.61873*10^8$ N3: $4.49777*10^8$ N4: $1.79955*10^9$	$O(n^2)$	$\approx (n^2 - n\log_2 n)/2$ N1: $4.99336*10^7$ N2: $1.61873*10^8$ N3: $4.49777*10^8$ N4: $1.79955*10^9$

Пространственная сложность – O(1) (переменная цикла, вспомогательные переменные, обозначающие левую и правую границу сортировки, вспомогательная переменная для обмена и вспомогательная переменная для фиксации места последнего обмена).

4) Алгоритм сортировки quicksort (быстрая сортировка):

Массив разделяется на группы, в одну часть помещаются все элементы, большие опорного элемента, а в другую – меньшие. Данная последовательность действий повторяется пока массив не будет отсортирован

Трудоемкость сортировки quicksort по количеству сравнений:

Случай	**Ситуация, соответст-вующая случаю**	**Обоснование**	**Ожидаемое число сравнений элементов массива**	**Асимптотическая оценка сложности по количеству сравнений**	**Ожидаемое число вспомогат. сравнений**
наилучший	каждый опорный элемент является медианой сортируемой части	на каждом шаге разделение происходит так, что в левой части оказываются элементы, большие или равные опорному, а в правой - элементы, меньшие или равные опорному.	nlogn N1: 132877 N2: 254443 N3: 446180 N4: 952360	O(nlogn)	
наихудший	каждый опорный элемент является минимальным или максимальным в сортируемой части	массив делится на 2 части, в одной из которых нет ни одного элемента, что сильно повышает сложность алгоритма	n^2 N1: $10*10^7$ N2: $3.24*10^8$ N3: $9*10^8$ N4: $3.6*10^9$	$O(n^2)$	
средний	каждый опорный элемент является случайным числом		nlogn*2ln2 N1: 184206 N2: 352733 N3: 618537 N4: 1320250	O(nlogn)	

Для нашей функций быстрой сортировки самыми лучшими будут случаи, при которых массив отсортирован в обратном или правильном порядке, в этом случае выборе центрального элемента массива будет крайне выгоден. Худшим случаем является тот, при котором массив не отсортирован вообще.

Пространственная сложность – O(n) (функция рекурсивная, глубина рекурсии может доходить до n; две переменных цикла и вспомогательная переменная, являющаяся опорным элементом, а также вспомогательная переменная для обмена).

Текст программы:

```
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <istream>
#include <fstream>
#include <ctime>
#include <conio.h>

using namespace std;


const int ESC_KEY = 27;
int quicksortCounter = 0; //глобальная переменная для подсчета кол-ва сравнений в quicksort
int extraQuicksortCounter = 0; //глобальная переменная для подсчета кол-ва доп. сравнений в quicksort
size_t allocatedMemory=0;         //глобальная переменная для подсчета выделенной памяти



void codeExit();          //прототип функции для досрочного завершения программы

template <typename T>
void printArray(T* arr, int n);

template <typename T>
void BubbleSort(T* arr, int n);

template <typename T>
void BubbleSortUpgraded(T* arr, int n);

template <typename T>
void ShakerSort(T* arr, int n);

template <typename T>
void QuickSort(T* arr, int n);

template <typename T>
void ReverseQuickSort(T* arr, int n);

int main(){

  setlocale(LC_ALL, "Russian");

  string filename = "test_numbers.txt";

  ifstream file(filename);
```

```
        if (!file.is_open()){
          cout << "Error open file" << endl;
          cout << "Завершение работы программы...";
          exit(0);                              //в случае, если файл не удалось
открыть, программа досрочно завершается
        }


        for (int n=1; n<=4; n++){
          int startTime, endTime;
          int size;
          if (n>1){
            codeExit();
          }
          cout << "Введите число N" << n <<": ";
          cin >> size;
          int *numbers = new int[size];      //создание динамического массива
размера size

          for (int i=0;i<size;i++){
            int value;
            file >> value;
            numbers[i]=value;                //запись данных из файла в массив
numbers
          }


          cout << "-----------------------------------------------------------
--------\n";
          cout << "ПУЗЫРЬКОВАЯ СОРТИРОВКА (" << size << " элементов):\n";

          cout << "\n\n* * * Применение функции для неотсортированного массива
* * *\n";
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);             //вывод первых 8 элементов массива
до сортировки
          cout << "\n-----------------------------------------------------------
----------";
          BubbleSort(numbers,size);         //сортировка массива пузырьком
          cout << "+----------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);             //вывод первых 8 элементов массива
до сортировки
          cout << endl << endl;
          system("Pause");                   //программа ставится на паузу


          cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
          cout << "\n-----------------------------------------------------------
----------";
          BubbleSort(numbers,size);
          cout << "-----------------------------------------------------------
--------\n";
          cout << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для массива, отсортированного в
обратном порядке * * *\n";
          ReverseQuickSort(numbers,size);      //сортировка массива в обратном
```

```
порядке
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n----------------------------------------------------------
---------";
          BubbleSort(numbers,size);
          cout << "----------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");
          file.clear();
          file.seekg(0);
          system("cls");    //очистка экрана

          for (int i=0;i<size;i++){   //заполнение массива новыми данными
            int value;
            file >> value;
            numbers[i]=value;
          }


          cout << "----------------------------------------------------------
--------\n";
          cout << "ПУЗЫРЬКОВАЯ СОРТИРОВКА С ФИКСАЦИЕЙ МЕСТА ОБМЕНА (" << size <<
" элементов):\n";

          cout << "\n\n* * * Применение функции для неотсортированного массива
* * *\n";
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n----------------------------------------------------------
----------";
          BubbleSortUpgraded(numbers,size); //сортировка массива пузырьком с
фиксацией места обмена
          cout << "----------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
          cout << "\n----------------------------------------------------------
----------";
          BubbleSortUpgraded(numbers,size);
          cout << "----------------------------------------------------------
--------\n";
          cout << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для массива, отсортированного в
обратном порядке * * *\n";
          ReverseQuickSort(numbers,size);
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n----------------------------------------------------------
---------";
          BubbleSortUpgraded(numbers,size);
```

```
          cout << "-------------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");
          file.clear();
          file.seekg(0);
          system("cls");

          for (int i=0;i<size;i++){
            int value;
            file >> value;
            numbers[i]=value;
          }


          cout << "-------------------------------------------------------------
--------\n";
          cout << "ШЕЙКЕРНАЯ СОРТИРОВКА (" << size << " элементов):\n";

          cout << "\n\n* * * Применение функции для неотсортированного массива
* * *\n";
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n-------------------------------------------------------------
----------";
          ShakerSort(numbers,size); //сортировка массива шейкером
          cout << "-------------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки:  ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
          cout << "\n-------------------------------------------------------------
----------";
          ShakerSort(numbers,size);
          cout << "-------------------------------------------------------------
--------\n";
          cout << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для массива, отсортированного в
обратном порядке * * *\n";
          ReverseQuickSort(numbers,size);
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n-------------------------------------------------------------
---------";
          ShakerSort(numbers,size);
          cout << "-------------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");
          file.clear();
          file.seekg(0);
```

```
          system("cls");

          for (int i=0;i<size;i++){
            int value;
            file >> value;
            numbers[i]=value;
          }

          cout << "-------------------------------------------------------------
--------\n";
          cout << "БЫСТРАЯ СОРТИРОВКА (" << size << " элементов):\n";

          cout << "\n\n* * * Применение функции для неотсортированного массива
* * *\n";
          cout << "\nПервые 8 элементов массива до сортировки: ";
          printArray(numbers,8);
          cout << "\n-------------------------------------------------------------
----------\n";
          startTime = clock();        //получение информации о текущем времени
          QuickSort(numbers,size);    //сортировка быстрой сортировкой
          endTime = clock();          //получение информации о текущем времени
(после работы функции)
          cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
          cout << "Выделенная память: " << allocatedMemory << " байт\n";
          cout << "Количество сравнений: " << quicksortCounter << endl;
          cout    <<    "Количество    вспомогательных    сравнений:    "    <<
extraQuicksortCounter << endl;
          quicksortCounter = 0;
          extraQuicksortCounter=0;
          cout << "-------------------------------------------------------------
--------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
          cout << "\n-------------------------------------------------------------
----------\n";
          startTime = clock();
          QuickSort(numbers,size);
          endTime = clock();
          cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
          cout << "Выделенная память: " << allocatedMemory << " байт\n";
          cout << "Количество сравнений: " << quicksortCounter << endl;
          cout    <<    "Количество    вспомогательных    сравнений:    "    <<
extraQuicksortCounter << endl;
          quicksortCounter = 0;
          extraQuicksortCounter = 0;
          cout << "-------------------------------------------------------------
--------\n";
          cout << endl;
          system("Pause");


          cout << "\n\n* * * Применение функции для массива, отсортированного в
обратном порядке * * *\n";
          ReverseQuickSort(numbers,size);
          cout << "\nПервые 8 элементов массива до сортировки: ";
```

```
          printArray(numbers,8);
          cout << "\n-------------------------------------------------------------
---------\n";
          startTime = clock();
          QuickSort(numbers,size);
          endTime = clock();
          cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
          cout << "Выделенная память: " << allocatedMemory << " байт\n";
          cout << "Количество сравнений: " << quicksortCounter << endl;
          cout    <<    "Количество    вспомогательных    сравнений:    "    <<
extraQuicksortCounter << endl;

          quicksortCounter = 0;
          extraQuicksortCounter = 0;
          cout << "-------------------------------------------------------------
-------\n";
          cout << "Первые 8 элементов массива после сортировки: ";
          printArray(numbers,8);
          cout << endl << endl;
          system("Pause");
          file.clear();
          file.seekg(0);
          system("cls");

          delete [] numbers;
        }

        file.close();     //закрытие файла
        return 0;
      }

      template <typename T>
      void printArray(T* arr, int n){
        for (int i=0; i<n;i++){
          cout << arr[i] << " ";
        }
      }


      template <typename T>
      void BubbleSort(T* arr,int n){
        int startTime, endTime;

        startTime = clock();

        int assignmentCount=0, extraAssignmentCount = 0;
        for (int i=1;i<n;i++){
          extraAssignmentCount++;     //увеличение счетчика доп. сравнений
          for (int j=n-1;j>=i;j--){
            extraAssignmentCount++;  //увеличение счетчика доп. сравнений
            assignmentCount++;        //увеличение счетчика сравнений
            if(arr[j]>arr[j-1]){
              T temp = arr[j];
              arr[j] = arr[j-1];
              arr[j-1] = temp;
            }
          }
        }
        endTime = clock();

        size_t allocatedMemory;
        allocatedMemory = sizeof(int)*2 + sizeof(T); //память, выделенная под
```

```
массив + доп переменные (в данном случае - i, j и temp)

        cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
        cout << "Выделенная память: " << allocatedMemory << " байт\n";
        cout << "Количество сравнений: " << assignmentCount << endl;
        cout << "Количество вспомогательных сравнений: " << extraAssignmentCount
<< endl;
      }


      template <typename T>
      void BubbleSortUpgraded(T* arr, int n){
        int startTime, endTime;

        startTime = clock();

        int i=1,k;
        int assignmentCount=0, extraAssignmentCount = 0;
        do{
          k=0;
          for (int j=n-1;j>=i;j--){
            extraAssignmentCount++;   //увеличение счетчика доп. сравнений

            assignmentCount++;  //увеличение счетчика сравнений
            if(arr[j]>arr[j-1]){
              T temp = arr[j];
              arr[j] = arr[j-1];
              arr[j-1] = temp;
              k = j;      //запоминаем место последнего обмена
            }
          }
          i = k;         //запоминаем место последнего обмена как левую границу
        }while(k);

        endTime = clock();

        size_t allocatedMemory;
        allocatedMemory = sizeof(int)*3 + sizeof(T); //память, выделенная под
массив + доп переменные (в данном случае - i, j, k и temp)

        cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
        cout << "Выделенная память: " << allocatedMemory << " байт\n";
        cout << "Количество сравнений: " << assignmentCount << endl;
        cout << "Количество вспомогательных сравнений: " << extraAssignmentCount
<< endl;
      }

      template <typename T>
      void ShakerSort(T* arr, int n){
        int startTime, endTime;

        startTime = clock();

        int left = 1,right = n-1,k = n-1;
        int assignmentCount=0, extraAssignmentCount = 0;
        do{

          for(int j=right;j>=left;j--){ //просматриваем справа налево
            extraAssignmentCount++;   //увеличение счетчика доп. сравнений
            assignmentCount++;    //увеличение счетчика сравнений
```

```
            if(arr[j]>arr[j-1]){
              T temp = arr[j];
              arr[j] = arr[j-1];
              arr[j-1] = temp;
              k=j;
            }
          }
          left = k+1;
          for(int j=left;j<=right;j++){   //просматриваем слева направо
            extraAssignmentCount++;     //увеличение счетчика доп. сравнений
            assignmentCount++;    //увеличение счетчика сравнений

            if(arr[j]>arr[j-1]){
              T temp = arr[j];
              arr[j] = arr[j-1];
              arr[j-1] = temp;
              k=j;
            }
          }
          right = k-1;
          extraAssignmentCount++;   //увеличение счетчика доп. сравнений
        }while (left<=right);   //пока есть что просматривать

        endTime = clock();

        size_t allocatedMemory;
        allocatedMemory = sizeof(int)*4 + sizeof(T); //память, выделенная под
массив + доп переменные (в данном случае - i, j и temp)

        cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
мс\n";
        cout << "Выделенная память: " << allocatedMemory << " байт\n";
        cout << "Количество сравнений: " << assignmentCount << endl;
        cout << "Количество вспомогательных сравнений: " << extraAssignmentCount
<< endl;

      }

      template <typename T>
      void QuickSort(T* arr, int n){

        int i = 0,j = n-1;
        T pivot = arr[n/2]; //опорный элемент
        do
        {
          while (arr[i]>pivot) {  //слева находим элемент больше опорного
            i++;
            quicksortCounter++;
          }
          while (pivot>arr[j]){   //справа находим элемент меньше опорного
            j--;
            quicksortCounter++;
          }
          extraQuicksortCounter++;
          if (i<=j){
            T temp = arr[i];
            arr[i] = arr[j];    //меняем найденные элементы местами
            arr[j] = temp;
            i++;
            j--;
          }
          extraQuicksortCounter++;
        } while (i<j);  // по завершении цикла слева от опорного элемента ключи,
```

```
меньшие x, а справа - большие

        extraQuicksortCounter++;
        //сортируем так же левую и правую части
        if (j>0) QuickSort(arr,j+1);

        extraQuicksortCounter++;
        if (i<n-1) QuickSort(arr+i,n-i);

        allocatedMemory += sizeof(int)*2 + sizeof(T)*2;
      }

      template <typename T>
      void ReverseQuickSort(T* arr, int n){  //функция для сортировки массива в
обратном порядке относительно условия
        int i,j;
        T pivot = arr[n/2];
        i=0;
        j = n-1;
        do
        {
          while (arr[i]<pivot) i++;
          while (pivot<arr[j]) j--;
          if (i<=j){
            T temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
          }
        } while (i<j);
        if (j>0) ReverseQuickSort(arr,j+1);
        if (i<n-1) ReverseQuickSort(arr+i,n-i);
      }

      void codeExit(){
        cout << "Для досрочного завершения программы нажмите клавишу esc. Для
продолжения работы нажмите любую другую клавишу." << endl;
        if (getch()==ESC_KEY){                        //если пользователь нажал на
клавишу esc, выполнение программы завершается
          cout << "Досрочное завершение работы программы...";
          exit(0);
        }
      }
```

Результаты работы программы:

При запуске программа запрашивает у пользователя число N1

Рисунок 1 – Введите число N1

Далее на экран выводится информация о применении пузырьковой сортировки для массива размера N1.

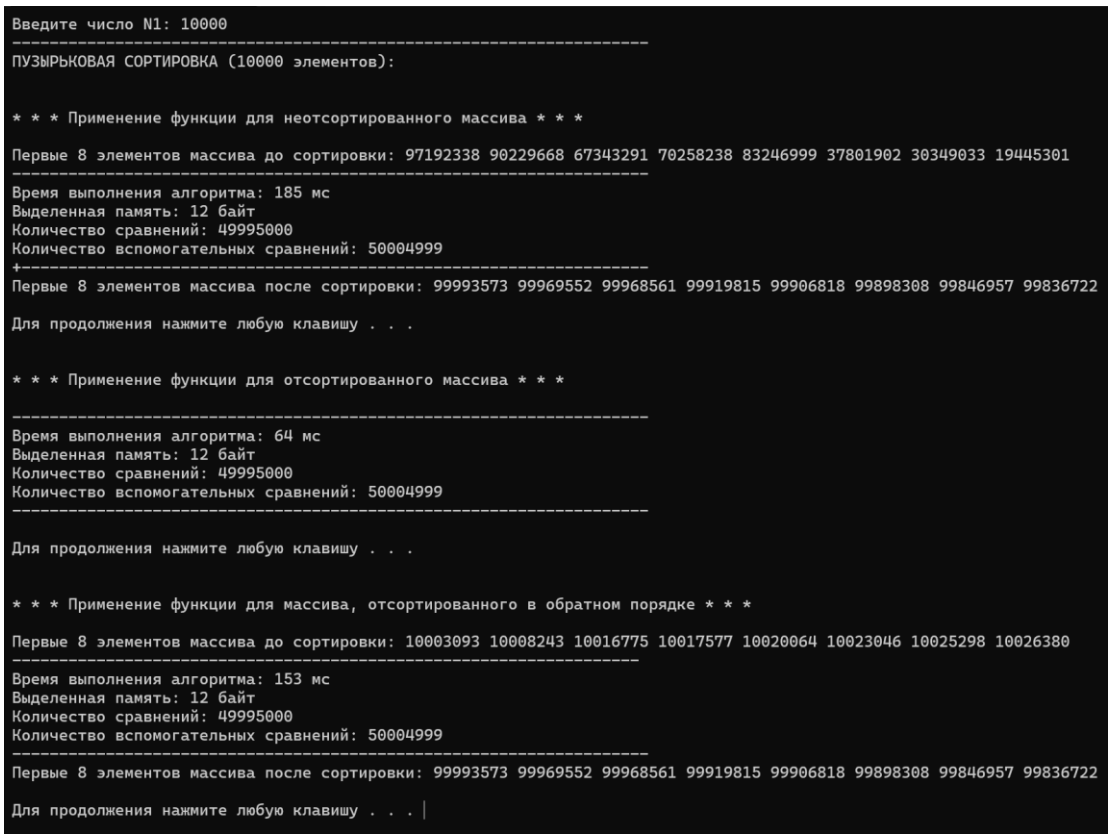

Рисунок 2 – Пузырьковая сортировка, 10000 элементов

Далее, после нажатия любой клавиши запускается пузырьковой сортировки с фиксацией места обмена для массива размера N1.

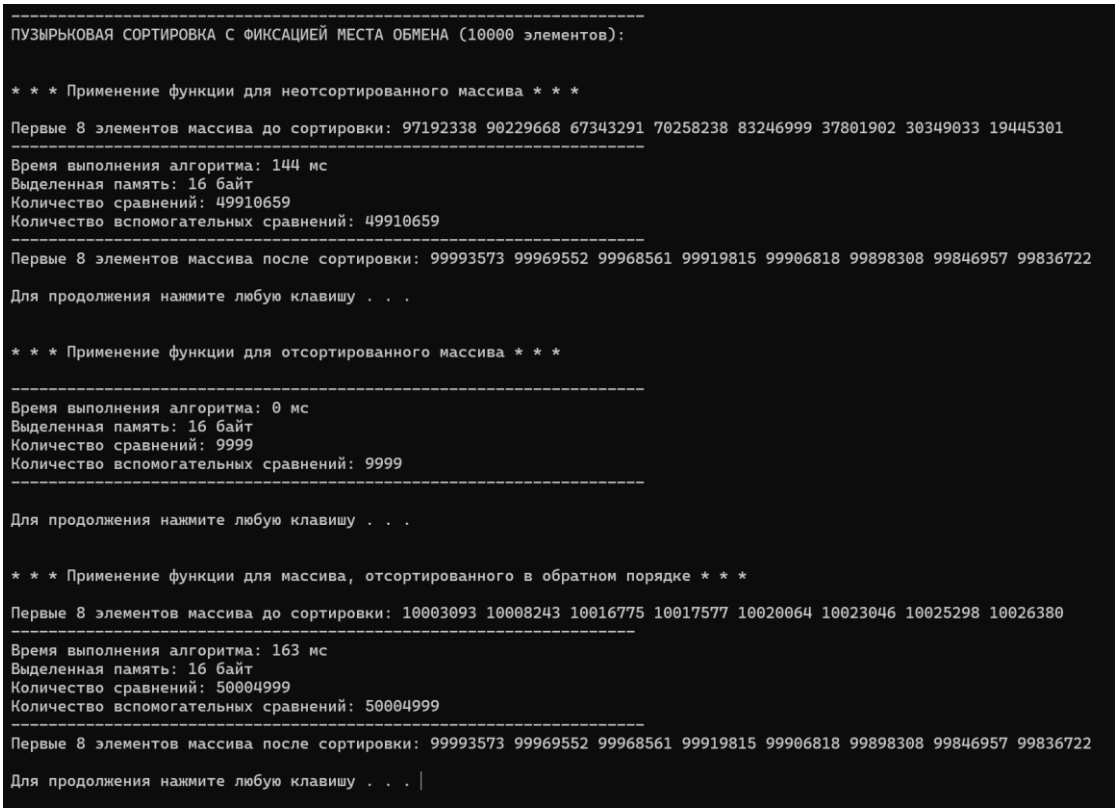

Рисунок 3 – Пузырьковая сортировка с фиксацией места обмена, 10000 элементов

Далее, после нажатия любой клавиши запускается шейкерной сортировки с для массива размера N1.

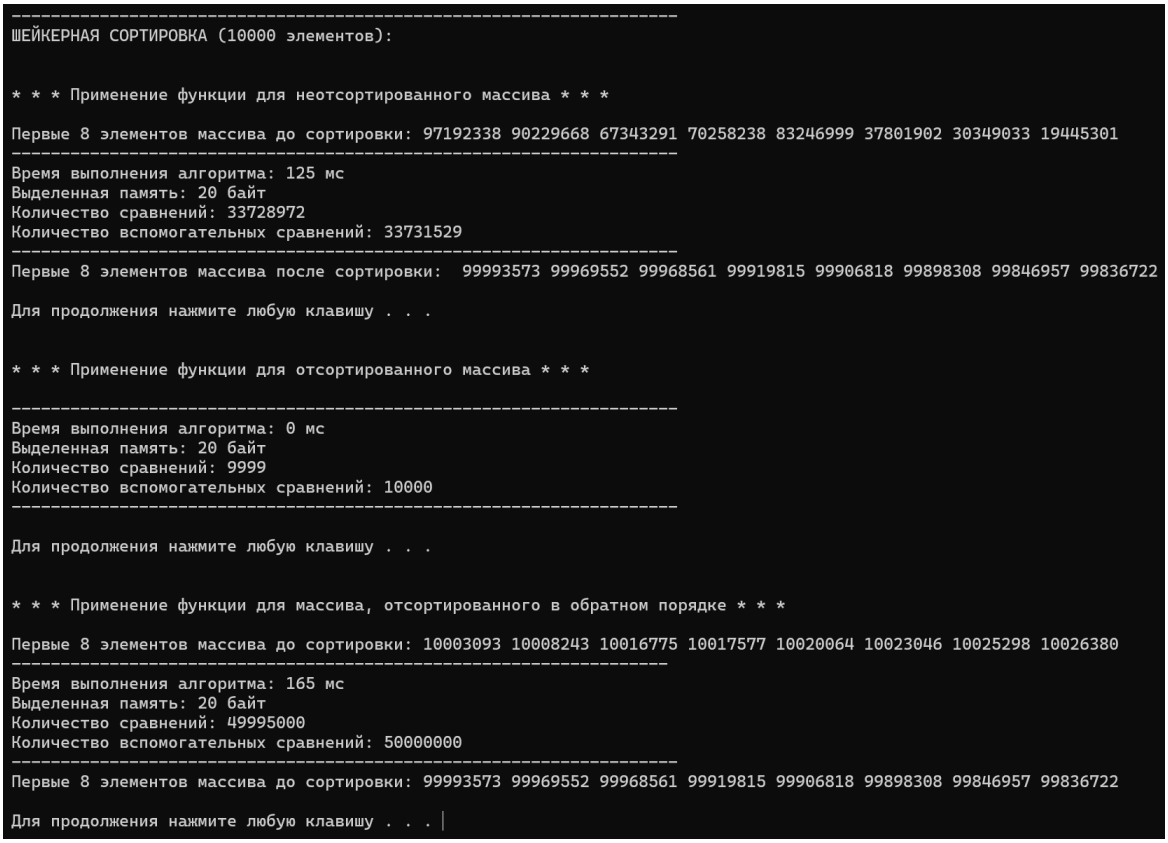

Рисунок 4 – Шейкерная сортировка, 10000 элементов

Далее, после нажатия любой клавиши запускается быстрой сортировки с фиксацией места обмена для массива размера N1.

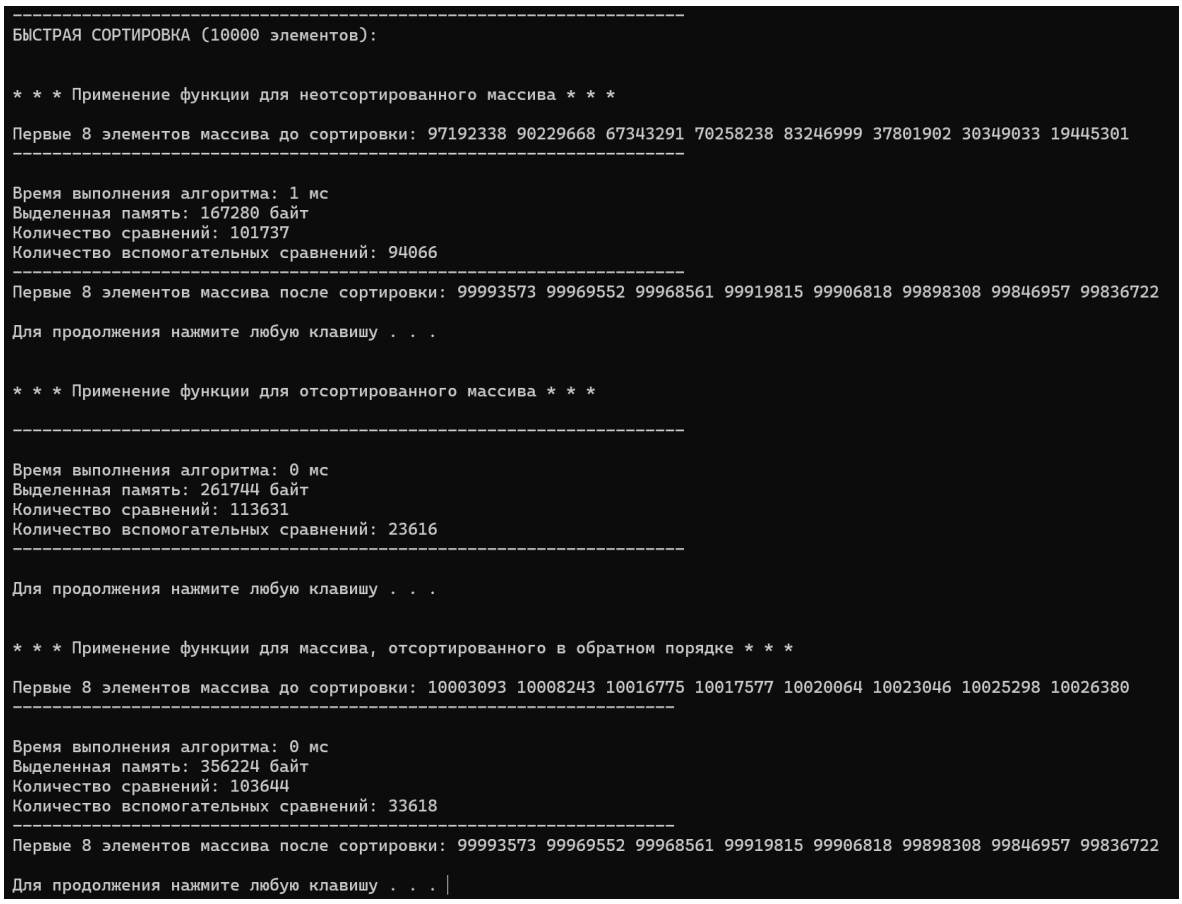

Рисунок 5 – Быстрая сортировка, 10000 элементов

У после выполнения всех сортировок у пользователя есть возможность завершить выполнение программы досрочно нажатием клавиши esc

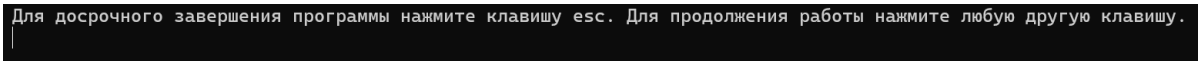

Рисунок 6 – Возможность досрочного завершения выполнения программы

Если пользователь нажал любую другую клавишу, программа запрашивает число N2

Рисунок 7 – Введите число N2

Далее повторяется последовательность действий, описанная выше

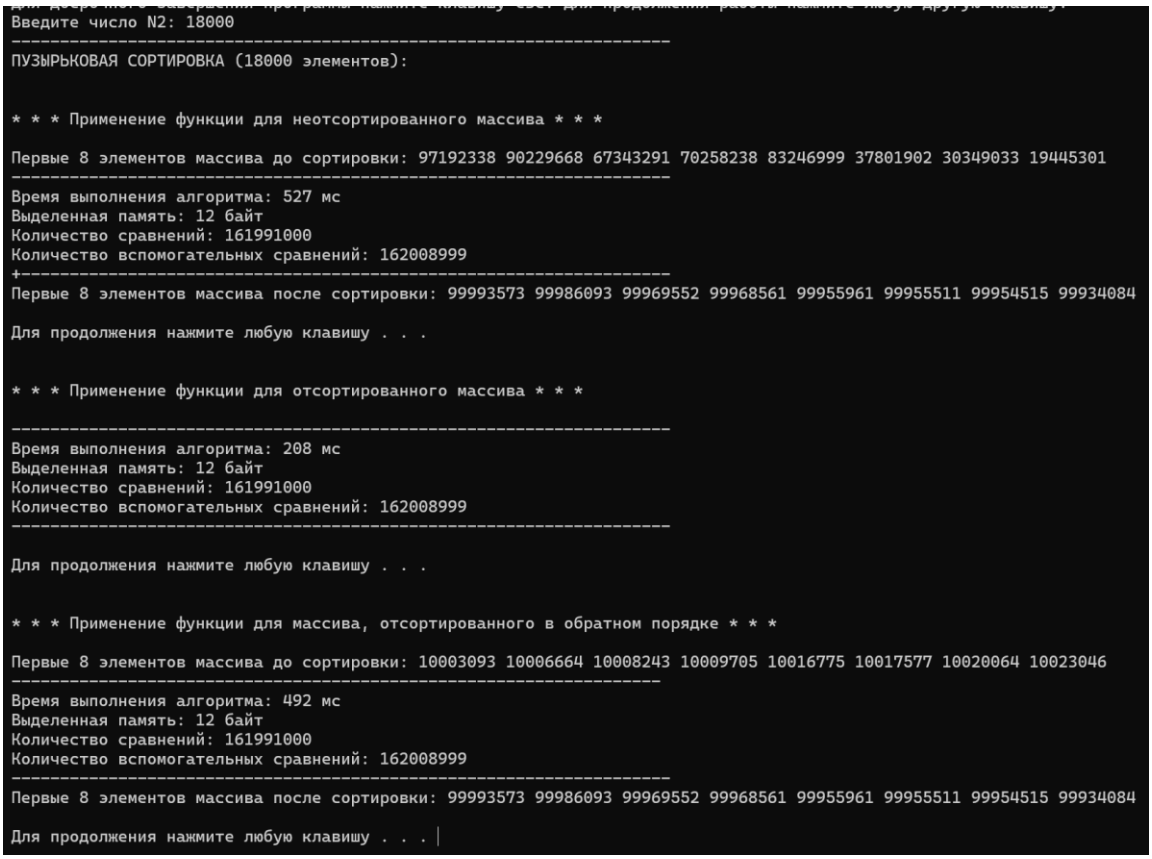

Рисунок 8 – Пузырьковая сортировка, 18000 элементов

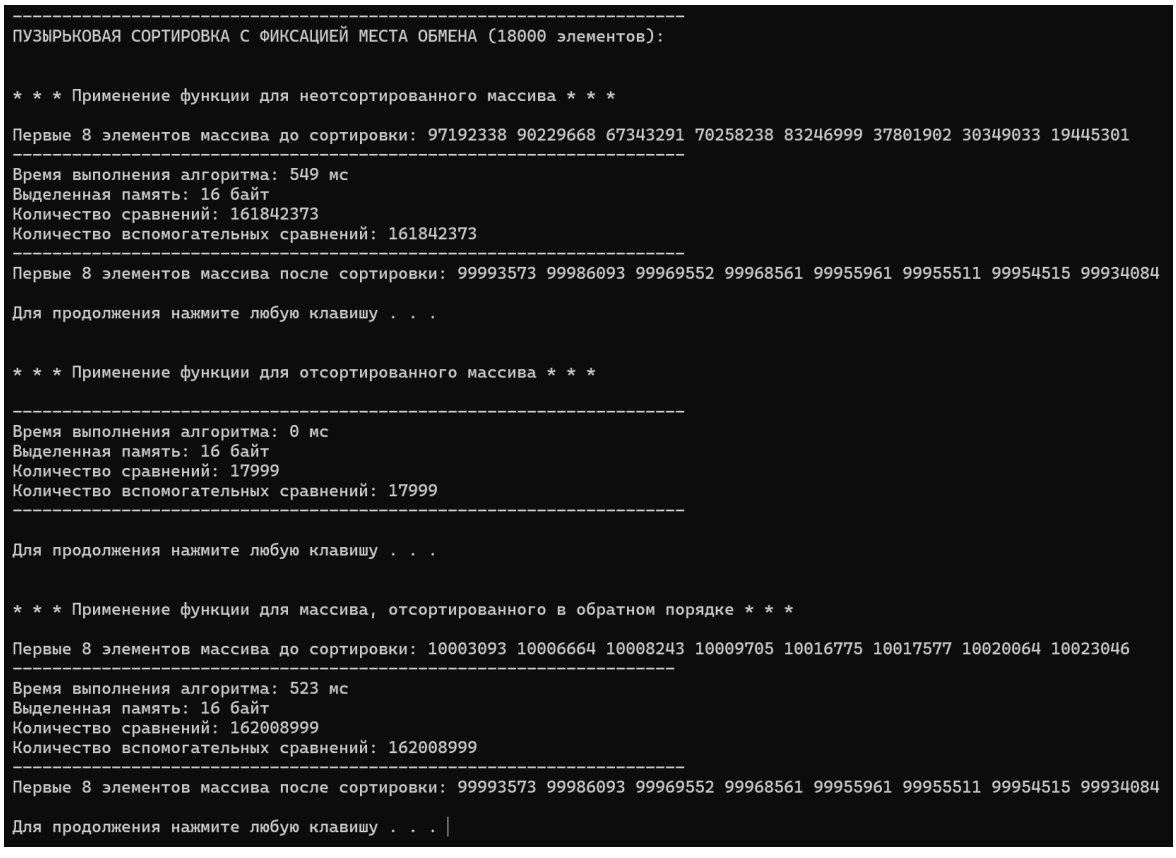

Рисунок 9 – Пузырьковая сортировка с фиксацией места обмена, 18000 элементов

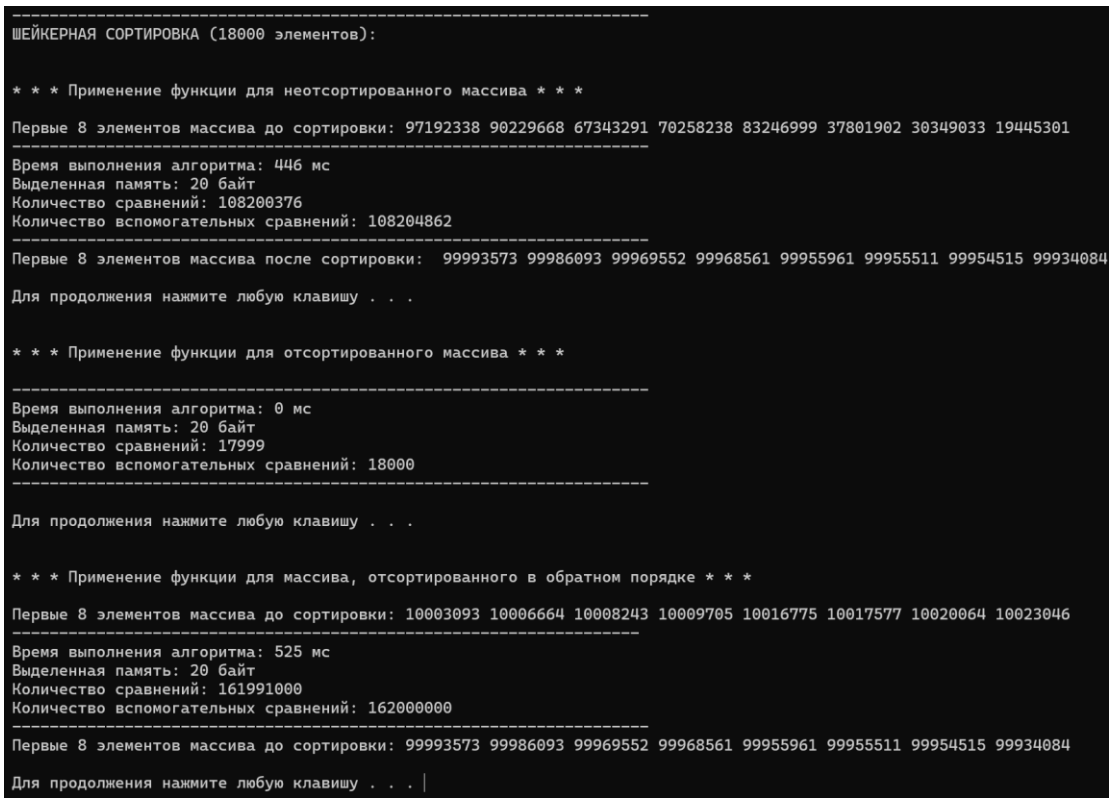

Рисунок 10 – Шейкерная сортировка, 18000 элементов

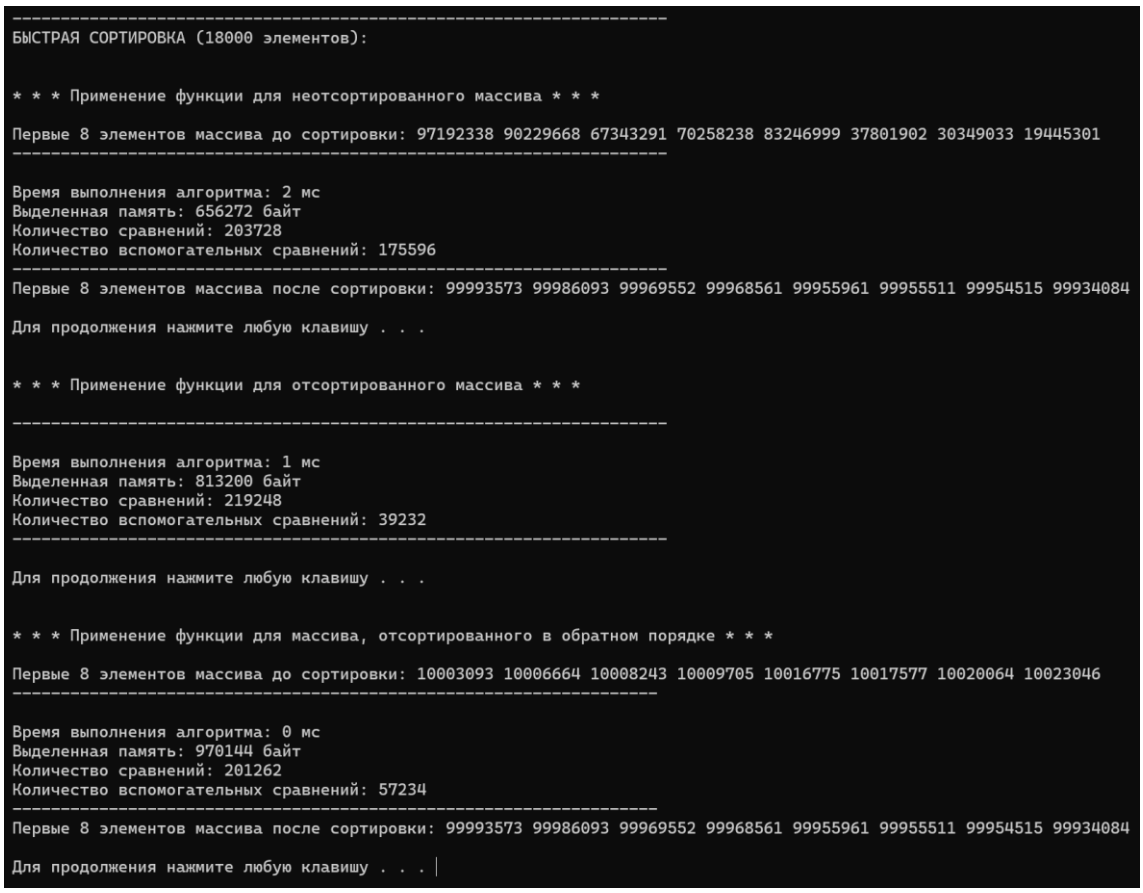

Рисунок 11 – Быстрая сортировка, 18000 элементов

Далее программа запрашивает число N3, после чего выводит на экран следующее:

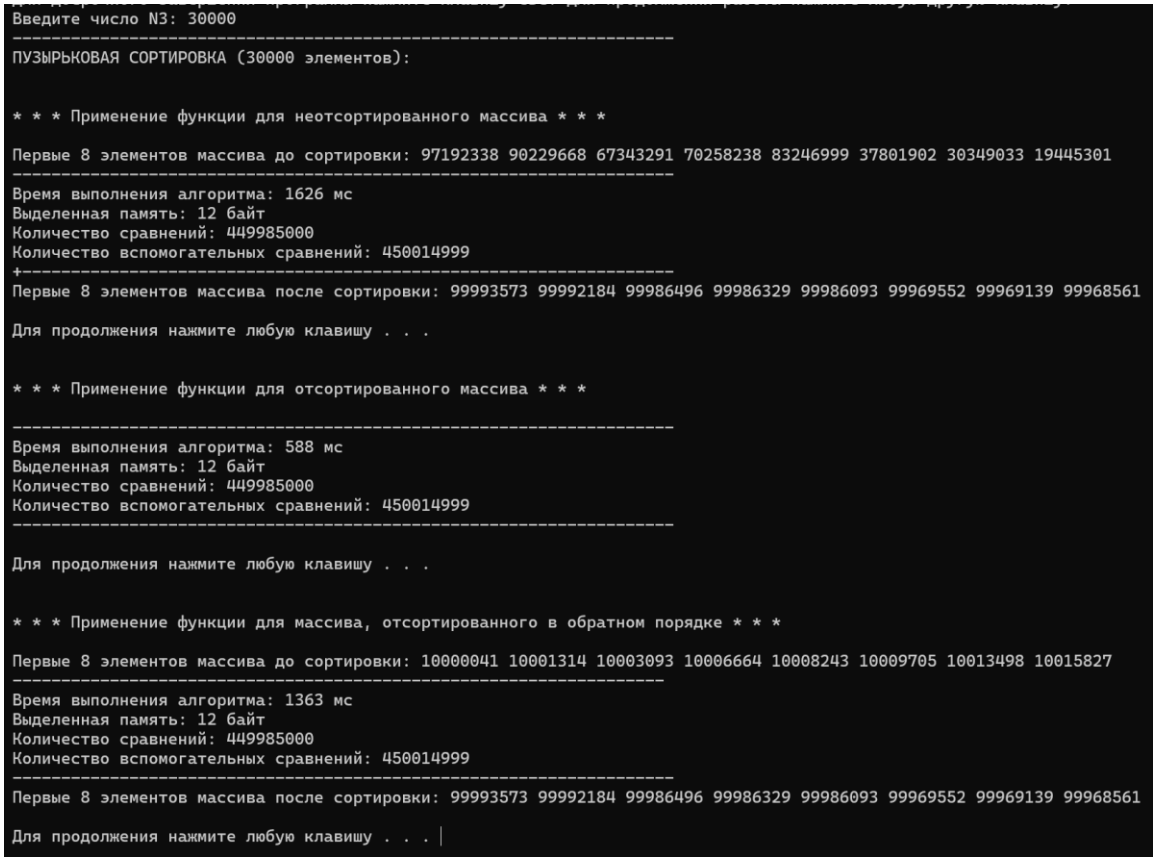

Рисунок 12 – Пузырьковая сортировка, 30000 элементов

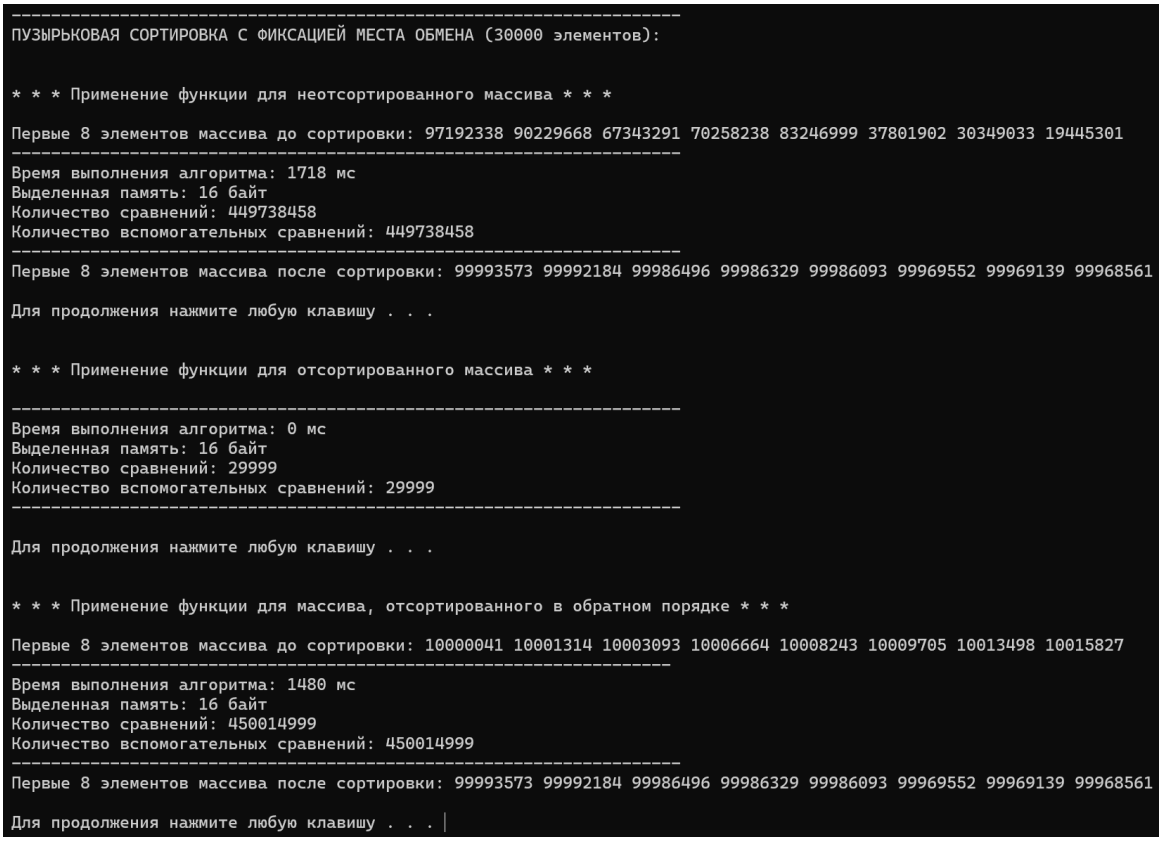

Рисунок 13 – Пузырьковая сортировка с фиксацией места обмена, 30000 элементов

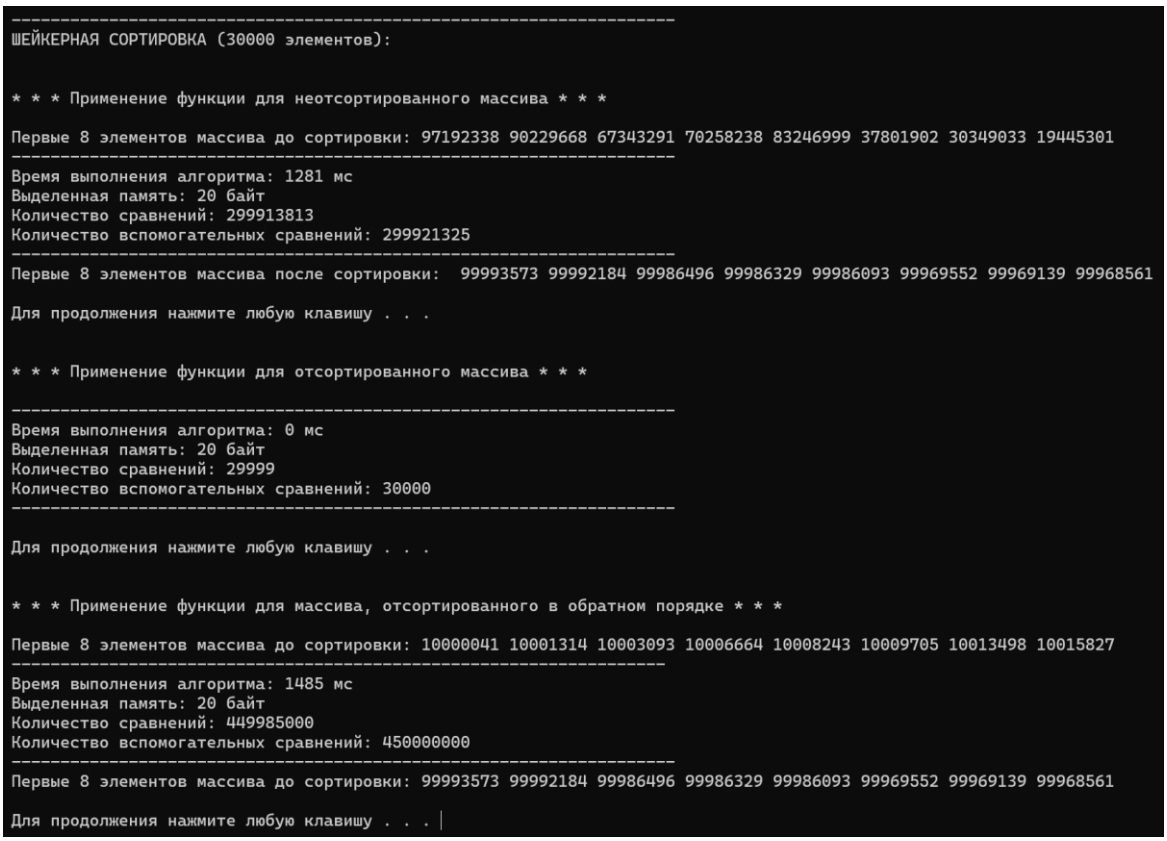

Рисунок 14 – Шейкерная сортировка, 30000 элементов

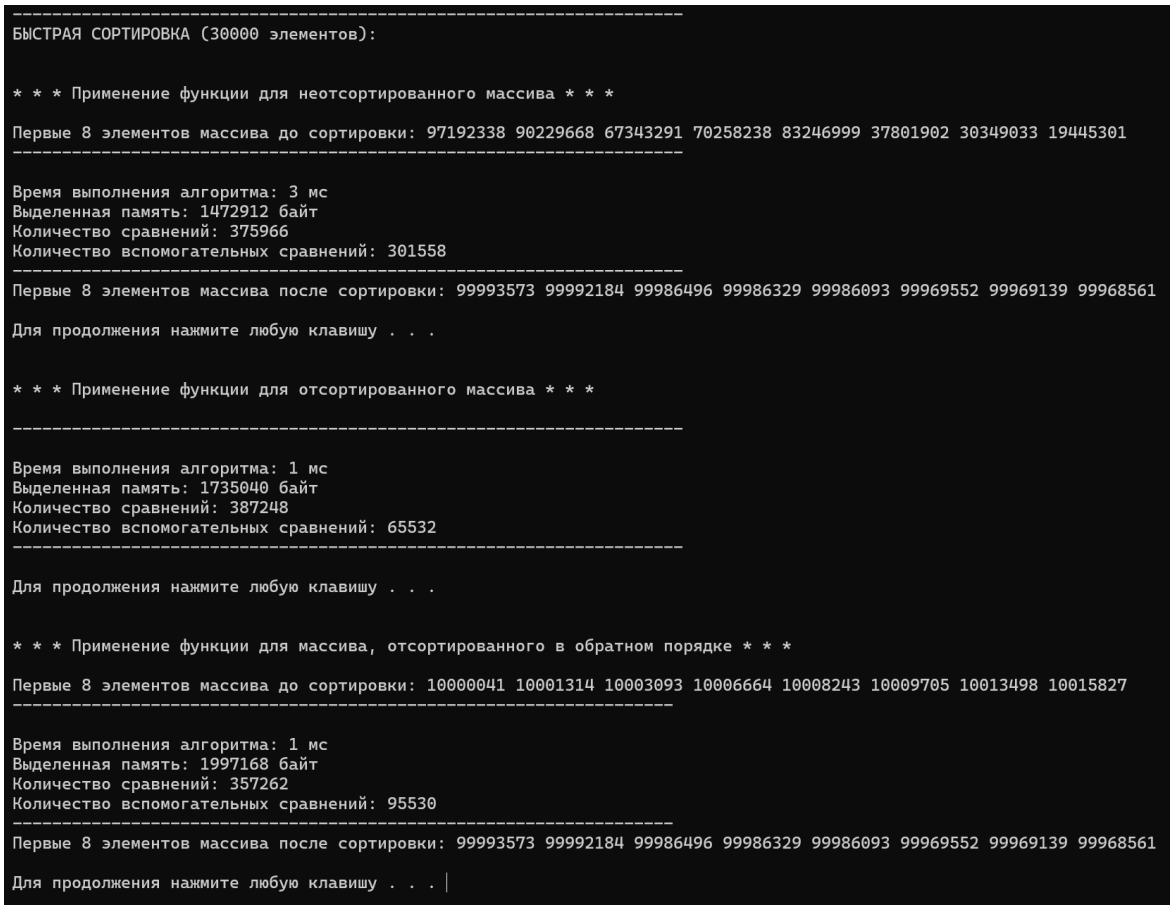

Рисунок 15 – Быстрая сортировка, 30000 элементов

Далее программа запрашивает число N3, после чего выводит на экран следующее

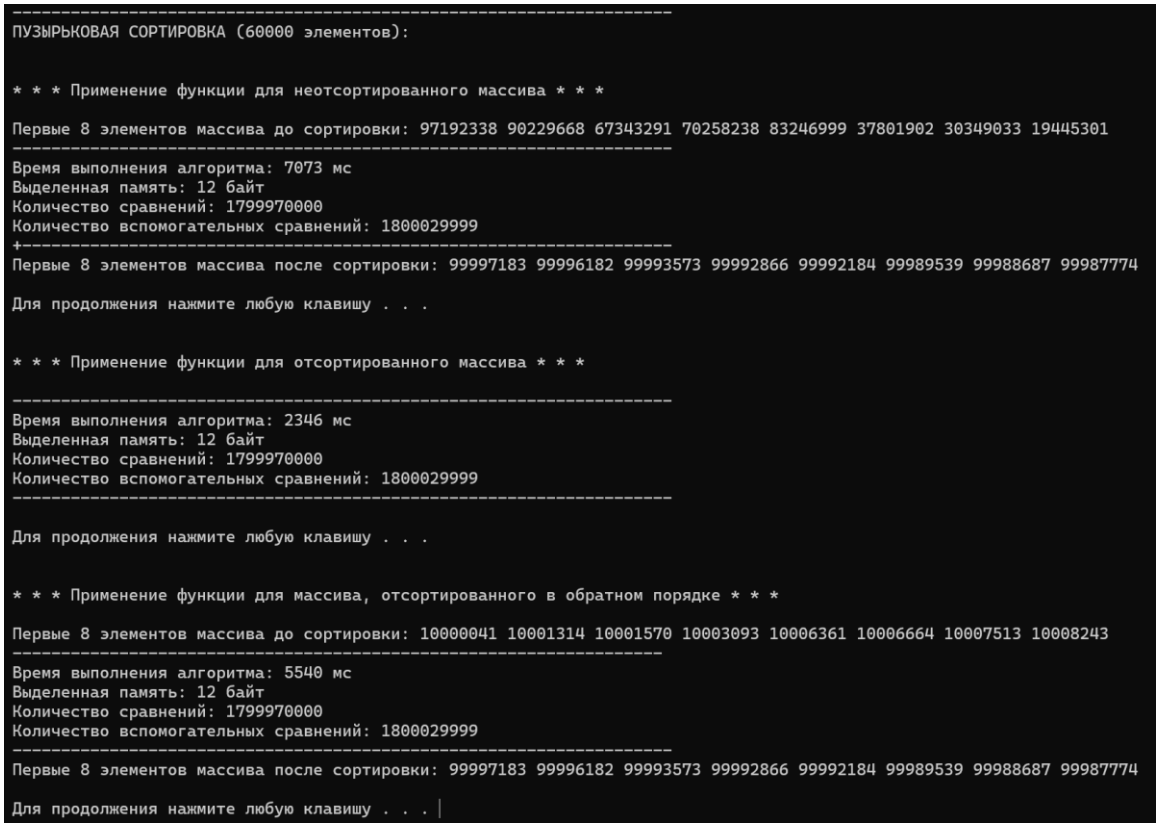

Рисунок 16 – Пузырьковая сортировка, 60000 элементов

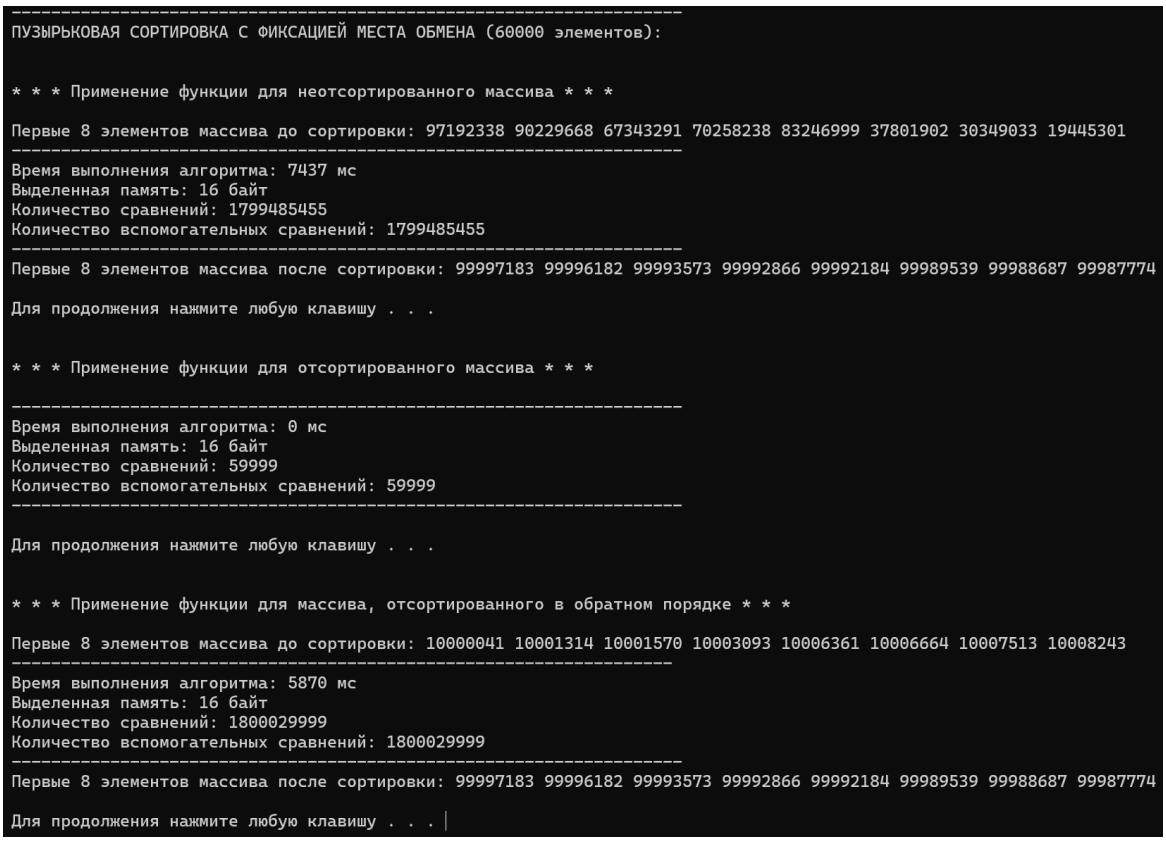

Рисунок 17 – Пузырьковая сортировка с фиксацией места обмена, 60000 элементов

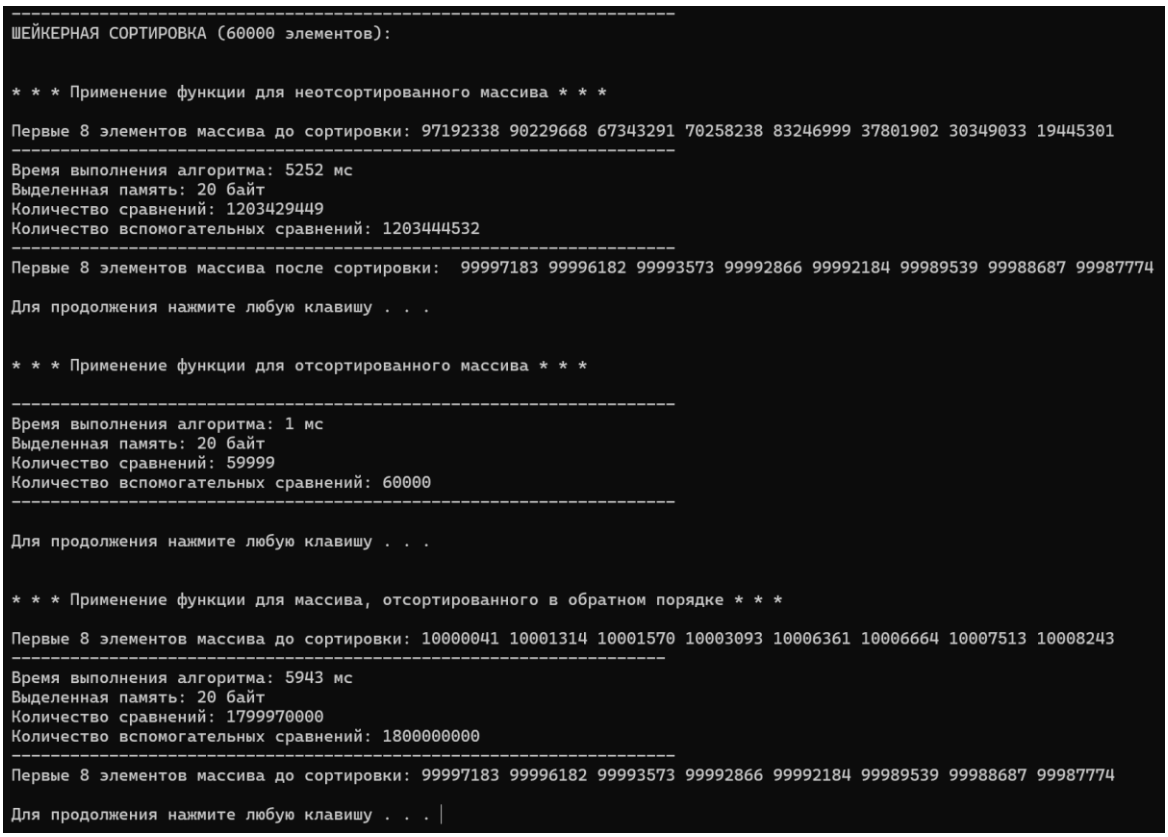

Рисунок 18 – Шейкерная сортировка, 60000 элементов

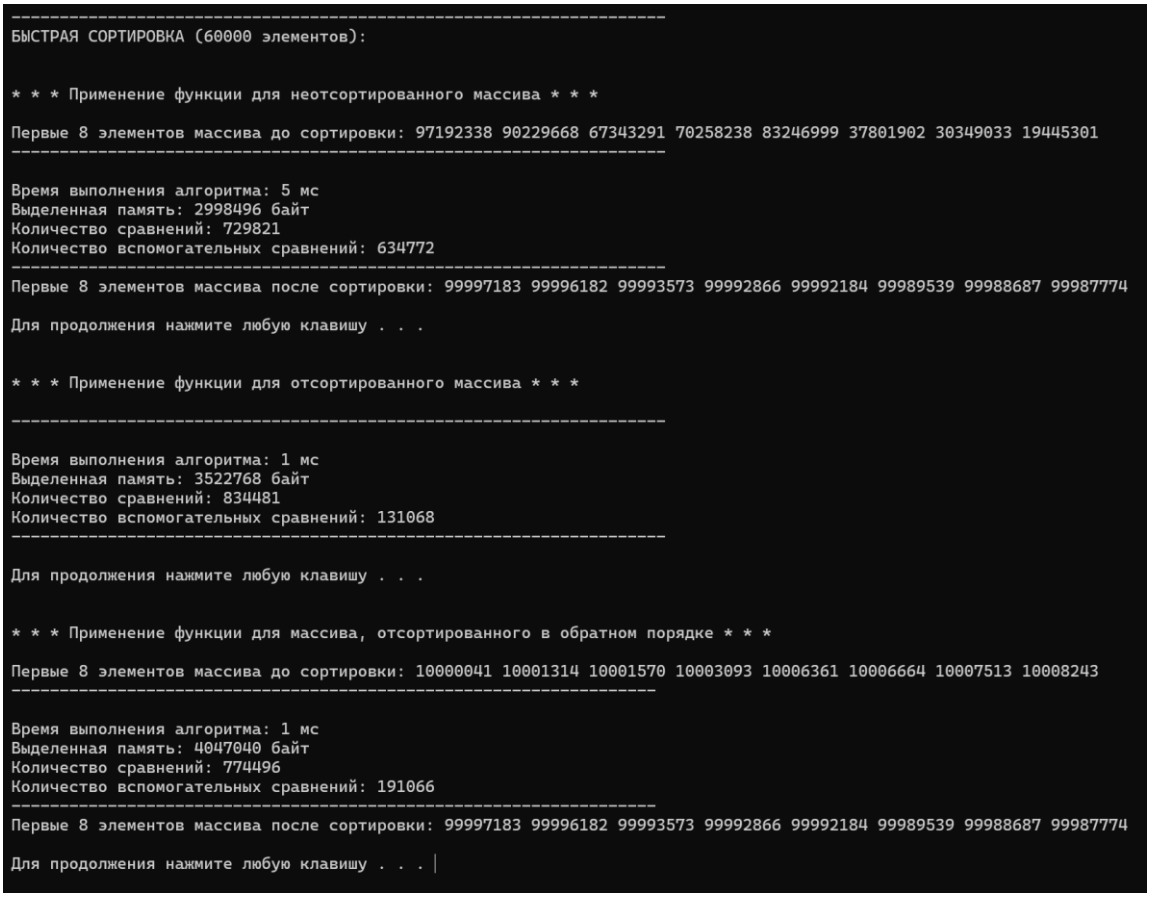

Рисунок 19 – Быстрая сортировка, 60000 элементов

После того, как выполнится быстрая сортировка для массива, отсортированного в обратном порядке, и пользователь нажмет на любую клавишу, выполнение программы закончится.

Анализ полученных данных:

Сравниваем полученные результаты с расчетными, делаем выводы.

1) Пузырьковая сортировка

   Ожидаемые и расчетные результаты совпали. Функция выполняет фиксированное количество операций, которое можно предсказать.

2) Пузырьковая сортировка с фиксацией места обмена

   Ожидаемые и расчетные результаты полностью совпали только для отсортированного массива и массива, отсортированного в обратном порядке, соответственно в этих случаях функция выполняет фиксированное количество операций, которое можно предсказать. Благодаря формуле также удалось приблизительно предсказать количество операций сравнения при сортировке изначального массива. Точно предсказать не удалось, так как количество операций напрямую зависит от того, насколько удачно расположены элементы в массиве.

3) Шейкерная сортировка

   Ожидаемые и расчетные результаты полностью совпали только для отсортированного массива и массива, отсортированного в обратном порядке, соответственно в этих случаях функция выполняет фиксированное количество операций, которое можно предсказать. Предсказать количество операций сравнения при сортировке изначального массива удалось лишь очень приблизительно, расхождение с ожидаемым оказалось достаточно велико. Точно предсказать не удалось, так как количество операций напрямую зависит от того, насколько удачно расположены элементы в массиве.

4) Быстрая сортировка

   Ожидаемые и расчетные результаты не совпали

Для каждого алгоритма сравниваем количество основных операций с количеством вспомогательных, анализируем, как изменяется соотношение при увеличении количества элементов, делаем выводы.

N…: неотсортированный массив, отсортированный массив, массив отсортированный в обратном порядке;

Основные сравнения … вспомогательные сравнения;

1) Пузырьковая сортировка

   N1: 49995000 < 50004999, 49995000 < 50004999, 49995000 < 50004999

   N4: 1799970000 < 1800029999, 1799970000 < 1800029999, 1799970000 < 1800029999

   Вывод: несмотря на увеличение числа элементов, соотношение вспомогательных сравнений к основным практически не изменяется

2) Пузырьковая сортировка с фиксацией места обмена

   N1: 49910659 = 49910659, 9999 = 9999, 50004999 = 50004999

   N4: 1799485455 = 1799485455, 59999 = 59999, 1800029999 = 1800029999

   Вывод: соотношение неизменно и равно 1 к 1

3) Шейкерная сортировка

   N1: 33728972 < 33731529, 9999 < 10000, 49995000 < 50000000

   N4: 1203429449 < 1203444532, 59999 < 60000, 1799970000 < 1800000000

   Вывод: несмотря на увеличение числа элементов, соотношение вспомогательных сравнений к основным практически не изменяется

4) Быстрая сортировка:

   N1: 101737 > 94066, 113631 > 23616, 103644 > 33618

   N4: 729821 > 634772, 834481 > 131068, 774496 > 191066

   Вывод: чем больше элементов в сортируемом массиве, тем сильнее увеличивается отношение основных сравнений к вспомогательным (в пользу основных сравнений)

Сравнение трудоемкости алгоритмов по асимптотике:

Исходя из расчетов, полученных из таблицы и данных, полученных в результате выполнения программы можно заметить, что наилучшую асимптотическую трудоемкость

имеет быстрая сортировка, т.к. на ее выполнение тратится наименьшее количество времени и внутри алгоритма выполняется наименьшее количество операций сравения (справедливо и для худшего, и для лучшего, и для среднего случаев). Второй по эффективности можно назвать Шейкерную сортировку (так как она значительно эффективнее работает в худшем и лучшем случаях). Далее идет улучшенная пузырьковая сортировка, (значительно быстрее чем стандартная пузырьковая сортировка при работе с отсортированным массивом). Худшей же является пузырьковая сортировка.

Сравнение алгоритмов по времени работы и количеству операций сравнения

1) Пузырьковая сортировка

   Быстрее всего сортировка выполняется для уже отсортированного в правильном порядке массива (лучший случай), при этом отсортированный в обратном порядке массив (худший случай) сортируется быстрее чем изначальный (неотсортированный) массив (средний случай). Во всех трех случаях количество операции сравнения совпадает.

2) Пузырьковая сортировка с фиксацией места обмена

   Быстрее всего сортировка выполняется для уже отсортированного в правильном порядке массива (лучший случай), при этом отсортированный в обратном порядке массив (худший случай) сортируется быстрее чем изначальный (неотсортированный) массив (средний случай). Количество операций сравнения значительно влияет на результат, только если оно сильно отличается. При небольшом отличии количества операций присваивания (как между средним и худшим случаями) разница незначительна. Алгоритм работает со скоростью, идентичной обычной пузырьковой сортировке, значительно быстрее чем в п. 1 функция выполняется лишь для уже отсортированного массива.

3) Шейкерная сортировка

   Быстрее всего сортировка выполняется для уже отсортированного в правильном порядке массива (лучший случай), изначальный (неотсортированный) массив (средний случай) сортируется быстрее чем массив, отсортированный в обратном порядке (худший случай). В данной сортировке прослеживается зависимость времени работы от количества операций сравнения – чем меньше операций, тем меньше время выполнения. Изначальный массив сортируется значительно быстрее чем в п. 1) и в п. 2), тогда как массив, отсортированный в обратном порядке, сортируется либо за идентичное, либо за незначительно большее время.

4) Быстрая сортировка

Самый быстрый алгоритм из рассмотренных. Одинаково быстро выполняется для отсортированного массива и массива, отсортированного в обратном порядке; дольше всего выполняется сортировка изначального, неотсортированного, массива. Помимо наименьшего времени выполнения, в этом методе также и наименьшее количество операций сравнения.

Вывод: количество операций присваивания влияет на время выполнения алгоритма, однако это влияние становится заметно лишь тогда, когда в одном случае операций сравнения в разы меньше, чем в другом.

Сравнение пространственной сложности алгоритмов:

1) Пузырьковая сортировка.
   Пространственная сложность этого метода составляет O(1). Он не требует дополнительной памяти, кроме нескольких переменных для временных операций.
2) Пузырьковая сортировка с фиксацией места обмена
   Пространственная сложность этого метода составляет O(1). Он не требует дополнительной памяти, кроме нескольких переменных для временных операций.

3) Шейкерная сортировка
   Пространственная сложность этого метода составляет O(1). Он не требует дополнительной памяти, кроме нескольких переменных для временных операций.

4) Быстрая сортировка
   Пространственная сложность этого метода составляет O(n). Поскольку данный алгоритм рекурсивный, ему требуется дополнительная память (при каждом вызове функции внутри себя, в стек помещаются переменные, под которые была выделена память)
   Вывод: все три алгоритма, имеющие пространственную сложность O(1), а соответственно, затрачивающее меньшее количество памяти, выполняют сортировку значительно дольше чем алгоритм с пространственной сложностью O(n). Из этого следует, что в данном случае: выше пространственная сложность (требуется больше памяти) → сортировка выполняется быстрее

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

(быстрая сортировка тратит много памяти, хотя в разы быстрее работает).