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 инженер					
шифр наименование кафедры, по которой выполняется работа					
Дисциплина	Структуры	и организация данных наименование дисциплины			
		паписнование двецивины			
У	′ЧЕБНО-	-ПРАКТИЧЕСКАЯ РАБОТА	L		
		№3			
тема	а «Опенк	ка эффективности алгоритмо	B>>		
101/1			D ,,		
	часть 1	«Алгоритмы сортировки»			
		Вариант 7			
		1			
		ОБУЧА	ЮЩИЙСЯ		
		группы	ы <u>И934Б</u>		
		Есипов М	ЛА		
		подпись фамилия и ини			
		12.05.2024			
		дата сдачи			
		7	ПРОВЕРИЛ		
		,	II ODEI IIJI		
		ученая степень, ученое звание, долж	кность		
		Фамилия	ı И.О.		
		подпись фамилия и и			
		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	O(n ²)	((n ² + n)/2)-1 N1: 50004999
			N2: 1.61991*10 ⁸ N3: 4.49985*10 ⁸ N4: 1.79997*10 ⁹		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	Асимптотичес кая оценка сложности по количеству присваивани й	Ожидаемое число вспомогат. присваиваний
наилучший	элементы	смена мест	n-1	O(n)	n-1
	массива	элементов не			
	упорядочены	требуется	N1: 9999		N1: 9999
			N2: 17999		N2: 17999
			N3: 29999		N3: 29999
			N4: 59999		N4: 59999
наихудший	массив	требуется	$((n^2+n)/2)-1$	$O(n^2)$	$((n^2+n)/2)-1$
	упорядочен в	поменять			
	обратном	местами все	N1: 50004999		N1: 50004999
	порядке	элементы	N2: 162008999		N2: 162008999
			N3: 450014999		N3: 450014999
			N4: 1800029999		N4: 1800029999
средний	неупорядо-	требуется	$\approx (n^2 - n)/2$	$O(n^2)$	$\approx (n^2 - n)/2$
	ченный массив	поменять	N1: 4.9995*10 ⁷		N1: 4.9995*10 ⁷
		местами	N2: 1.61991*10 ⁸		N2: 1.61991*10 ⁸
		некоторые	N3: 4.49985*10 ⁸		N3: 4.49985*10 ⁸
		элементы	N4: 1.79997*10 ⁹		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	Асимптотичес кая 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	требуется	$\approx (n^2 - n \log_2 n)/2$	$O(n^2)$	$\approx (n^2 - n \log_2 n)/2$
	упорядочен	поменять местами	N1: 4.99336*10 ⁷		N1: 4.99336*10 ⁷
		некоторые	N2: 1.61873*10 ⁸		N2: 1.61873*10 ⁸
		элементы	N3: 4.49777*10 ⁸		N3: 4.49777*10 ⁸
			N4: 1.79955*10 ⁹		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	Асимптотиче ская 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	nlogn N1: 132877 N2: 254443 N3: 446180 N4: 952360	O(nlogn)	
наихудший	каждый опорный элемент является минимальным или максимальным в сортируемой части	опорному. 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. cравнений 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;</pre>
        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++) {</pre>
         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;</pre>
        system("Pause");
                                     //программа ставится на паузу
        cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
        cout << "\n-----
        BubbleSort(numbers, size);
        cout << "-----
----\n";
        cout << endl;</pre>
        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;</pre>
        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 MECTA 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;</pre>
        system("Pause");
       cout << "\n\n* * * Применение функции для отсортированного массива *
* *\n";
       cout << "\n-----
       BubbleSortUpgraded(numbers, size);
       cout << "-----
----\n";
       cout << endl;</pre>
        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);
```

```
----\n";
       cout << "Первые 8 элементов массива после сортировки: ";
       printArray(numbers,8);
       cout << endl << endl;</pre>
       system("Pause");
       file.clear();
       file.seekg(0);
       system("cls");
       for (int i=0;i<size;i++) {</pre>
         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;</pre>
       system("Pause");
       cout << "\n^* * \n^* * Применение функции для отсортированного массива *
* *\n";
       cout << "\n-----
----";
       ShakerSort(numbers, size);
       cout << "-----
----\n";
       cout << endl;</pre>
       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;</pre>
       system("Pause");
       file.clear();
       file.seekg(0);
```

```
system("cls");
        for (int i=0;i<size;i++) {</pre>
          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 << "
Mc\n";
        cout << "Выделенная память: " << allocatedMemory << " байт\n";
        cout << "Количество сравнений: " << quicksortCounter << endl;
             << "Количество вспомогательных
                                                  сравнений:
extraOuicksortCounter << endl;
        quicksortCounter = 0;
        extraQuicksortCounter=0;
        cout << "-----
----\n";
        cout << "Первые 8 элементов массива после сортировки: ";
        printArray(numbers, 8);
        cout << endl << endl;</pre>
        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 << "
Mc\n";
        cout << "Выделенная память: " << allocatedMemory << " байт\n";
        cout << "Количество сравнений: " << quicksortCounter << endl;
        cout << "Количество вспомогательных сравнений:
                                                                   <<
extraQuicksortCounter << endl;</pre>
        quicksortCounter = 0;
        extraQuicksortCounter = 0;
        cout << "-----
----\n";
        cout << endl;</pre>
        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 << "
Mc\n";
         cout << "Выделенная память: " << allocatedMemory << " байт\n";
         cout << "Количество сравнений: " << quicksortCounter << endl;
         cout
              << "Количество вспомогательных сравнений:
extraQuicksortCounter << endl;</pre>
         quicksortCounter = 0;
         extraQuicksortCounter = 0;
         cout << "-----
----\n";
         cout << "Первые 8 элементов массива после сортировки: ";
         printArray(numbers, 8);
         cout << endl << endl;</pre>
         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 << "
Mc\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;
                         //запоминаем место последнего обмена
              k = \dot{j};
          }
          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 << "
Mc\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;
          for(int j=right; j>=left; j--) { //просматриваем справа налево
            extraAssignmentCount++; //увеличение счетчика доп. сравнений
            assignmentCount++; //увеличение счетчика сравнений
```

```
if(arr[i]>arr[i-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</pre>
        endTime = clock();
        size t allocatedMemory;
        allocatedMemory = sizeof(int)*4 + sizeof(T); //память, выделенная под
массив + доп переменные (в данном случае - і, ј и temp)
        cout << "\nВремя выполнения алгоритма: " << endTime - startTime << "
Mc\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
            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 х, 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);</pre>
        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++;</pre>
          while (pivot<arr[j]) j--;</pre>
          if (i<=j) {
            T temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
          }
        } while (i<j);</pre>
        if (j>0) ReverseQuickSort(arr,j+1);
        if (i<n-1) ReverseQuickSort(arr+i,n-i);</pre>
      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.

```
Введите число N1: 10000
ПУЗЫРЬКОВАЯ СОРТИРОВКА (10000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 185 мс
.
Выделенная память: 12 байт
Количество сравнений: 49995000
Количество вспомогательных сравнений: 50004999
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 64 мс
Выделенная память: 12 байт
роиделенная намяты. 12 оди
Количество сравнений: 49995000
Количество вспомогательных сравнений: 50004999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10008243 10016775 10017577 10020064 10023046 10025298 10026380
Время выполнения алгоритма: 153 мс
Выделенная память: 12 байт
Количество сравнений: 49995000
Количество вспомогательных сравнений: 50004999
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 999068<u>18 99898308 99846957 99836722</u>
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2 – Пузырьковая сортировка, 10000 элементов

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

```
ПУЗЫРЬКОВАЯ СОРТИРОВКА С ФИКСАЦИЕЙ МЕСТА ОБМЕНА (10000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 144 мс
Выделенная память: 16 байт
Количество сравнений: 49910659
Количество вспомогательных сравнений: 49910659
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 16 байт
Количество сравнений: 9999
Количество вспомогательных сравнений: 9999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10008243 10016775 10017577 10020064 10023046 10025298 10026380
Время выполнения алгоритма: 163 мс
Выделенная память: 16 байт
Количество сравнений: 50004999
Количество вспомогательных сравнений: 50004999
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . . .
```

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

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

```
ШЕЙКЕРНАЯ СОРТИРОВКА (10000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 125 мс
Выделенная память: 20 байт
Количество сравнений: 33728972
Количество вспомогательных сравнений: 33731529
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 20 байт Количество сравнений: 9999
Количество вспомогательных сравнений: 10000
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10008243 10016775 10017577 10020064 10023046 10025298 10026380
.
Выделенная память: 20 байт
Количество сравнений: 49995000
Количество вспомогательных сравнений: 50000000
Первые 8 элементов массива до сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 4 — Шейкерная сортировка, 10000 элементов

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

```
БЫСТРАЯ СОРТИРОВКА (10000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 1 мс
Выделенная память: 167280 байт
Количество сравнений: 101737
Количество вспомогательных сравнений: 94066
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 261744 байт
Количество сравнений: 113631
Количество вспомогательных сравнений: 23616
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10008243 10016775 10017577 10020064 10023046 10025298 10026380
Время выполнения алгоритма: 0 мс
Выделенная память: 356224 байт
Количество сравнений: 103644
Количество вспомогательных сравнений: 33618
Первые 8 элементов массива после сортировки: 99993573 99969552 99968561 99919815 99906818 99898308 99846957 99836722
Для продолжения нажмите любую клавишу .
```

Рисунок 5 – Быстрая сортировка, 10000 элементов

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

```
Для досрочного завершения программы нажмите клавишу esc. Для продолжения работы нажмите любую другую клавишу.
```

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

```
Введите число N2: 18000
ПУЗЫРЬКОВАЯ СОРТИРОВКА (18000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 8324699<u>9 37801902 30349033 19445301</u>
Время выполнения алгоритма: 527 мс
Выделенная память: 12 байт
Быделенная іншялів. 12 Один
Количество сравнений: 161991000
Количество вспомогательных сравнений: 162008999
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 999559<u>61 99955511 99954515 99934084</u>
Для продолжения нажмите любую клавишу .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 208 мс
Выделенная память: 12 байт
Количество сравнений: 161991000
Количество вспомогательных сравнений: 162008999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10006664 10008243 10009705 10016775 10017577 10020064 10023046
Время выполнения алгоритма: 492 мс
.
Выделенная память: 12 байт
Количество сравнений: 161991000
Количество вспомогательных сравнений: 162008999
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – Пузырьковая сортировка, 18000 элементов

```
ПУЗЫРЬКОВАЯ СОРТИРОВКА С ФИКСАЦИЕЙ МЕСТА ОБМЕНА (18000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 549 мс
Выделенная память: 16 байт
Количество сравнений: 161842373
Количество вспомогательных сравнений: 161842373
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 999<u>54515 99934084</u>
Для продолжения нажмите любую клавишу . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 16 байт
Количество сравнений: 17999
Количество вспомогательных сравнений: 17999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10006664 10008243 10009705 10016775 10017577 10020064 10023046
Время выполнения алгоритма: 523 мс
Выделенная память: 16 байт
Количество сравнений: 162008999
Количество вспомогательных сравнений: 162008999
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . . .
```

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

```
ШЕЙКЕРНАЯ СОРТИРОВКА (18000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 446 мс
  ıделенная память: 20<sup>°</sup> байт
рыделенная намять 20 один
Количество сравнений: 108200376
Количество вспомогательных сравнений: 108204862
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Выделенная память: 20 байт
Количество сравнений: 17999
Количество вспомогательных сравнений: 18000
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10006664 10008243 10009705 10016775 10017577 10020064 10023046
Время выполнения алгоритма: 525 мс
Выделенная память: 20 байт
общеленная память: 20 байт
Количество сравнений: 161991000
Количество вспомогательных сравнений: 162000000
Первые 8 элементов массива до сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . .
```

Рисунок 10 – Шейкерная сортировка, 18000 элементов

```
БЫСТРАЯ СОРТИРОВКА (18000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 2 мс
Выделенная память: 656272 байт
Количество сравнений: 203728
Количество вспомогательных сравнений: 175596
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 1 мс
Выделенная память: 813200 байт
Количество сравнений: 219248
Количество вспомогательных сравнений: 39232
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10003093 10006664 10008243 10009705 10016775 10017577 10020064 10023046
Время выполнения алгоритма: 0 мс
Выделенная память: 970144 байт
Количество сравнений: 201262
Количество вспомогательных сравнений: 57234
Первые 8 элементов массива после сортировки: 99993573 99986093 99969552 99968561 99955961 99955511 99954515 99934084
Для продолжения нажмите любую клавишу . . .
```

Рисунок 11 – Быстрая сортировка, 18000 элементов

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

```
Введите число N3: 30000
ПУЗЫРЬКОВАЯ СОРТИРОВКА (30000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 1626 мс
Выделенная память: 12 байт
Количество сравнений: 449985000
Количество вспомогательных сравнений: 450014999
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 588 мс
Выделенная память: 12 байт
Количество сравнений: 449985000
Количество вспомогательных сравнений: 450014999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10003093 10006664 10008243 10009705 10013498 10015827
Время выполнения алгоритма: 1363 мс
Выделенная память: 12 байт
Количество сравнений: 449985000
Количество вспомогательных сравнений: 450014999
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу . . .
```

Рисунок 12 – Пузырьковая сортировка, 30000 элементов

```
ПУЗЫРЬКОВАЯ СОРТИРОВКА С ФИКСАЦИЕЙ МЕСТА ОБМЕНА (30000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 1718 мс
Выделенная память: 16 байт
Количество сравнений: 449738458
Количество вспомогательных сравнений: 449738458
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99966952 99969139 99968561
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 16 байт
Количество сравнений: 29999
Количество вспомогательных сравнений: 29999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10003093 10006664 10008243 10009705 10013498 10015827
Время выполнения алгоритма: 1480 мс
Выделенная память: 16 байт
Количество сравнений: 450014999
Количество вспомогательных сравнений: 450014999
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу .
```

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

```
ШЕЙКЕРНАЯ СОРТИРОВКА (30000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 1281 мс
Выделенная память: 20 байт
Количество сравнений: 299913813
Количество вспомогательных сравнений: 299921325
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу . . .
 * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 20 байт
Количество сравнений: 29999
Количество вспомогательных сравнений: 30000
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10003093 10006664 10008243 10009705 10013498 <u>10015827</u>
Время выполнения алгоритма: 1485 мс
Выделенная память: 20 байт
Количество сравнений: 449985000
Количество вспомогательных сравнений: 450000000
Первые 8 элементов массива до сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 14 – Шейкерная сортировка, 30000 элементов

```
БЫСТРАЯ СОРТИРОВКА (30000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 3 мс
Выделенная память: 1472912 байт
Количество сравнений: 375966
Количество вспомогательных сравнений: 301558
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 999<u>69139 99968561</u>
Для продолжения нажмите любую клавишу . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 1 мс
Выделенная память: 1735040 байт
Количество сравнений: 387248
Количество вспомогательных сравнений: 65532
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10003093 10006664 10008243 10009705 10013498 10015827
Время выполнения алгоритма: 1 мс
Выделенная память: 1997168 байт
Количество сравнений: 357262
Количество вспомогательных сравнений: 95530
Первые 8 элементов массива после сортировки: 99993573 99992184 99986496 99986329 99986093 99969552 99969139 99968561
Для продолжения нажмите любую клавишу . . .
```

Рисунок 15 – Быстрая сортировка, 30000 элементов

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

```
ПУЗЫРЬКОВАЯ СОРТИРОВКА (60000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 7073 мс
.
Выделенная память: 12 байт
Быделенная намяты. 12 один
Количество сравнений: 1799970000
Количество вспомогательных сравнений: 1800029999
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 2346 мс
.
Выделенная память: 12 байт
Количество сравнений: 1799970000
Количество вспомогательных сравнений: 1800029999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10001570 10003093 10006361 10006664 10007513 10008243
Время выполнения алгоритма: 5540 мс
Выделенная память: 12 байт
Количество сравнений: 1799970000
Количество вспомогательных сравнений: 1800029999
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу .
```

Рисунок 16 – Пузырьковая сортировка, 60000 элементов

```
ПУЗЫРЬКОВАЯ СОРТИРОВКА С ФИКСАЦИЕЙ МЕСТА ОБМЕНА (60000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 7437 мс
Выделенная память: 16 байт
Количество сравнений: 1799485455
Количество вспомогательных сравнений: 1799485455
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 0 мс
Выделенная память: 16 байт
Количество сравнений: 59999
Количество вспомогательных сравнений: 59999
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10001570 10003093 10006361 10006664 10007513 10008243
Время выполнения алгоритма: 5870 мс
Выделенная память: 16 байт
Количество сравнений: 1800029999
Количество вспомогательных сравнений: 1800029999
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . . |
```

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

```
ШЕЙКЕРНАЯ СОРТИРОВКА (60000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 5252 мс
Выделенная память: 20 байт
Количество сравнений: 1203429449
Количество вспомогательных сравнений: 1203444532
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 1 мс
Выделенная память: 20 байт
Количество сравнений: 59999
Количество вспомогательных сравнений: 60000
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10001570 10003093 10006361 10006664 10007513 10008243
Выделенная память: 20 байт
Количество сравнений: 1799970000
Количество вспомогательных сравнений: 1800000000
Первые 8 элементов массива до сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . .
```

Рисунок 18 – Шейкерная сортировка, 60000 элементов

```
БЫСТРАЯ СОРТИРОВКА (60000 элементов):
* * * Применение функции для неотсортированного массива * * *
Первые 8 элементов массива до сортировки: 97192338 90229668 67343291 70258238 83246999 37801902 30349033 19445301
Время выполнения алгоритма: 5 мс
Выделенная память: 2998496 байт
Количество сравнений: 729821
Количество вспомогательных сравнений: 634772
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для отсортированного массива * * *
Время выполнения алгоритма: 1 мс
Выделенная память: 3522768 байт
Количество сравнений: 834481
Количество вспомогательных сравнений: 131068
Для продолжения нажмите любую клавишу . . .
* * * Применение функции для массива, отсортированного в обратном порядке * * *
Первые 8 элементов массива до сортировки: 10000041 10001314 10001570 10003093 10006361 10006664 10007513 10008243
Время выполнения алгоритма: 1 мс
Выделенная память: 4047040 байт
Количество сравнений: 774496
Количество вспомогательных сравнений: 191066
Первые 8 элементов массива после сортировки: 99997183 99996182 99993573 99992866 99992184 99989539 99988687 99987774
Для продолжения нажмите любую клавишу . . . |
```

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.
- Пузырьковая сортировка с фиксацией места обмена
 Пространственная сложность этого метода составляет О(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) \rightarrow 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).