ФЕДЕРАЛЬНОЕ Государственное АВТОНОМНОЕ образовательное учреждение

высшего образования

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра математического и программного обеспечения информационных систем

**Отчет по лабораторной работе №2**

**по дисциплине: «Структуры и алгоритмы компьютерной обработки данных»**

**Тема работы «Сравнительный анализ методов сортировки»**

студента очного отделения

2 курса 12001801 группы

Джинганин Олег Юрьевич

Проверил(а):

Курлов Василий Васильевич

Гайворонский Виталий Александрович

Белгород 2019

**Лабораторная работа №2**

**Вариант 14**

**Цель работы:** Изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

**Задание**

1. Изучить временные характеристики алгоритмов.

2. Изучить методы сортировки:

1) вставками;

2) выбором;

3) обменом;

1) улучшенная обменом 1;

2) улучшенная обменом 2;

4) Шелла;

5) Хоара;

6) пирамидальная.

3. Программно реализовать методы сортировки массивов.

4. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов сортировки.

5. Провести эксперименты по определению временных характеристик алгоритмов сортировки. Результаты экспериментов представить в виде таблиц 3.1, 3.2 и 3.3, клетки которых содержат количество операций сравнения при выполнении алгоритма сортировки массива с заданным количеством элементов.

6. Построить график зависимости количества операций сравнения от количества элементов в сортируемом массиве.

7. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.

8. Определить порядок функций временной сложности алгоритмов сортировки при сортировке упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов.

**Выполнение работы**

Таблица 3.1

Упорядоченный массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Вставками | 4 | 9 | 14 | 19 | 24 | 29 | 34 | 39 | 44 |
| Выбором | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| Обменом | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Обменом 1 | 4 | 9 | 14 | 19 | 24 | 29 | 34 | 39 | 44 |
| Обменом 2 | 6 | 11 | 16 | 21 | 26 | 32 | 36 | 41 | 46 |
| Шелла | 7 | 22 | 34 | 62 | 78 | 94 | 143 | 162 | 184 |
| Хоара | 7 | 12 | 17 | 22 | 27 | 32 | 37 | 42 | 47 |
| Пирамидальная | 14 | 39 | 67 | 99 | 128 | 166 | 203 | 242 | 275 |

Таблица 3.2

Массив, упорядоченный в обратном порядке

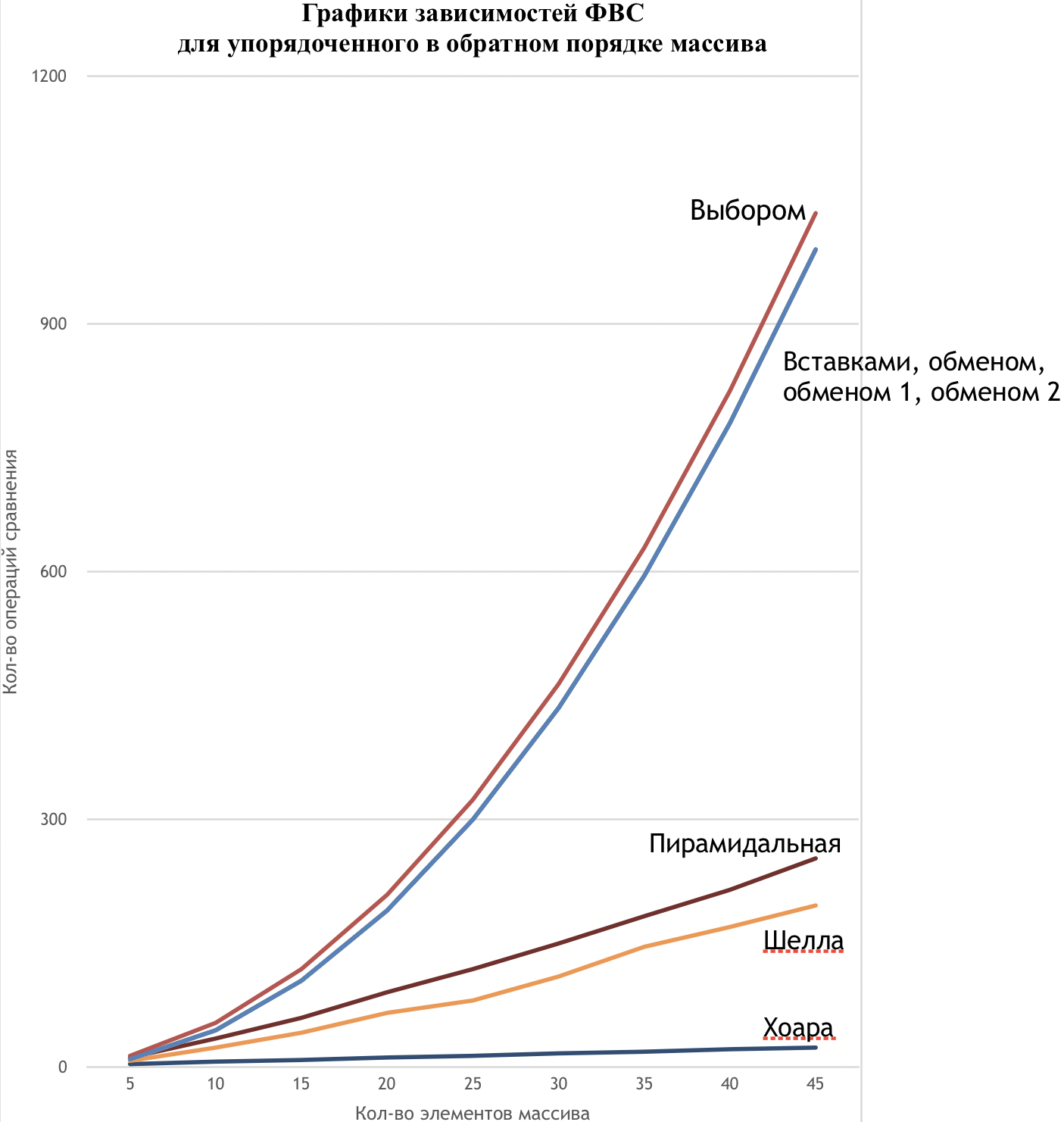
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Вставками | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Выбором | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| Обменом | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Обменом 1 | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Обменом 2 | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Шелла | 8 | 24 | 42 | 66 | 81 | 110 | 146 | 170 | 196 |
| Хоара | 4 | 7 | 9 | 12 | 14 | 17 | 19 | 22 | 24 |
| Пирамидальная | 12 | 35 | 60 | 91 | 119 | 150 | 183 | 215 | 253 |

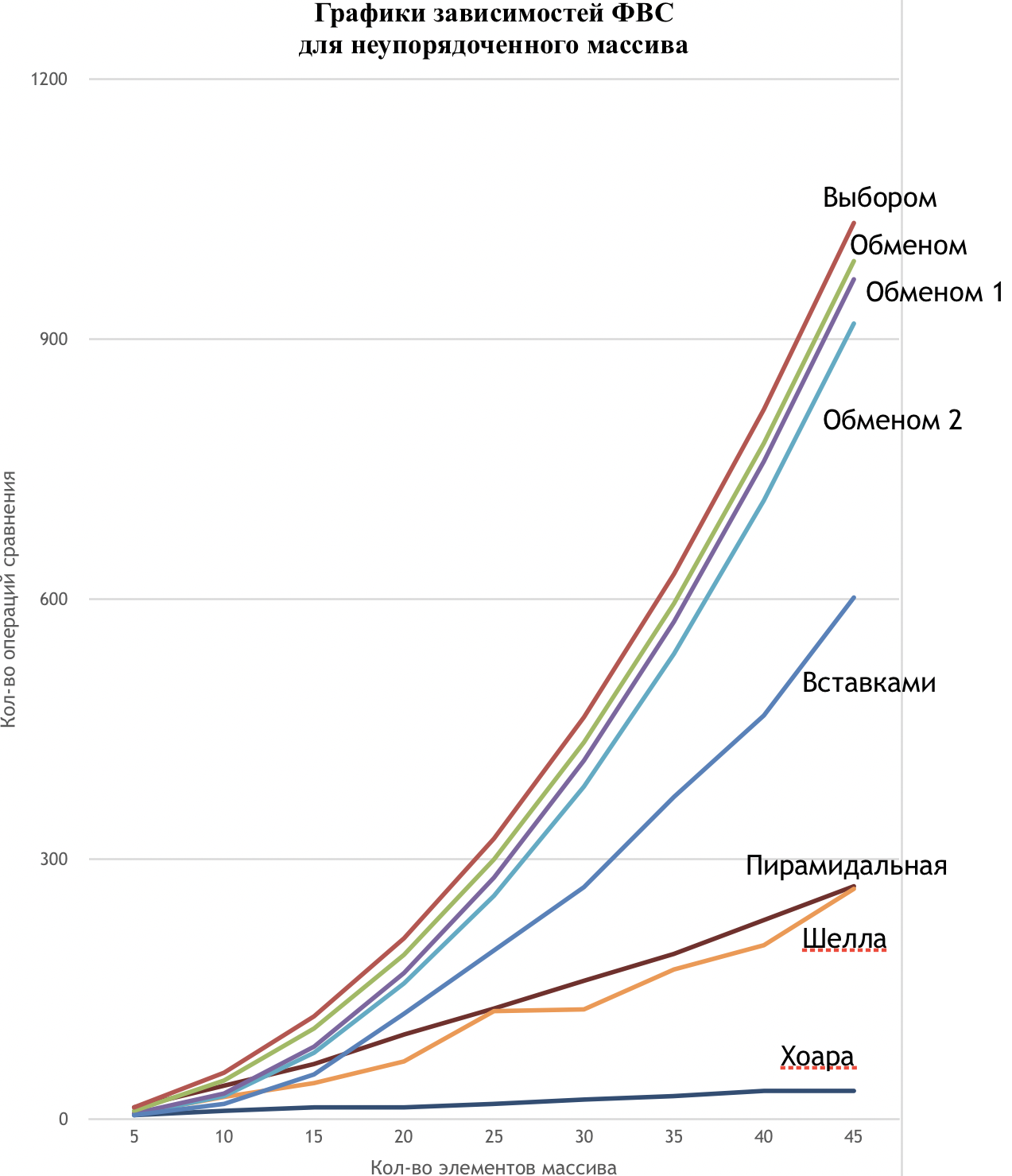
Таблица 3.3

Неупорядоченный массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Вставками | 5 | 18 | 52 | 122 | 195 | 268 | 372 | 466 | 602 |
| Выбором | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| Обменом | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| Обменом 1 | 7 | 30 | 84 | 169 | 279 | 414 | 574 | 759 | 969 |
| Обменом 2 | 6 | 27 | 77 | 157 | 258 | 384 | 537 | 714 | 918 |
| Шелла | 7 | 26 | 42 | 67 | 125 | 127 | 173 | 201 | 266 |
| Хоара | 5 | 10 | 14 | 14 | 18 | 23 | 27 | 33 | 33 |
| Пирамидальная | 14 | 39 | 64 | 98 | 128 | 160 | 191 | 230 | 269 |

**2. Графики зависимостей ФВС**





**Листинг программы**

*#include <iostream>*

*#include <random>*

*using namespace std;*

*void SwapElement(int\* arr, int i)*

*{*

*int buff;*

*buff = arr[i - 1];*

*arr[i - 1] = arr[i];*

*arr[i] = buff;*

*}*

*int InsertSort(int\* arr,int n){*

*unsigned count = 0;*

*for (int i = 1; i < n; i++)*

*{*

*if (arr[i] >= arr[i - 1])*

*count++;*

*for (int j = i; j > 0 && arr[j - 1] > arr[j]; j--, count++)*

*SwapElement(arr, j);*

*}*

*return count;*

*}*

*int BubbleSort(int\* a, int n) // пузырек*

*{*

*unsigned count = 0;*

*for(int i=0;i<n-1;i++)*

*{*

*for(int j=n-1;j>i;j--)*

*{*

*count++;*

*if (a[j-1] > a[j])*

*SwapElement(a,j);*

*}*

*}*

*return count;*

*}*

*unsigned BubbleSortv1(int\* arr, short n)*

*{*

*unsigned count = 0;*

*bool check;*

*for (int i = 0; i < n - 1; i++)*

*{*

*check = false;*

*for (int j = (n - 1); j > i; j--)*

*{*

*count++;*

*if (arr[j - 1] > arr[j])*

*{*

*SwapElement(arr, j);*

*check = true;*

*}*

*}*

*if (check == false)*

*break;*

*}*

*return count;*

*}*

*unsigned BubbleSortv2(int\* arr, short n)*

*{*

*unsigned count = 0;*

*int last\_swaped\_el\_num;*

*for (int i = 0; i < n - 1; i++)*

*{*

*for (int j = (n - 1); j > i; j--)*

*{*

*count++;*

*if (arr[j - 1] > arr[j])*

*{*

*SwapElement(arr, j);*

*last\_swaped\_el\_num = j - 1;*

*}*

*else last\_swaped\_el\_num = j;*

*}*

*i = last\_swaped\_el\_num;*

*}*

*return count;*

*}*

*int siftDown(int \*numbers, int root, int bottom)*

*{*

*int count = 0;*

*int maxChild;*

*int done = 0;*

*while ((root \* 2 <= bottom) && (!done))*

*{*

*if (root \* 2 == bottom)*

*maxChild = root \* 2;*

*else if (numbers[root \* 2] > numbers[root \* 2 + 1])*

*maxChild = root \* 2;*

*else*

*maxChild = root \* 2 + 1;*

*count++;*

*if (numbers[root] < numbers[maxChild])*

*{*

*int temp = numbers[root];*

*numbers[root] = numbers[maxChild];*

*numbers[maxChild] = temp;*

*root = maxChild;*

*}*

*else*

*done = 1;*

*}*

*return count;*

*}*

*int heapSort(int \*numbers, int array\_size)*

*{*

*int count = 0;*

*for (int i = (array\_size / 2) - 1; i >= 0; i--)*

*count += siftDown(numbers, i, array\_size - 1);*

*for (int i = array\_size - 1; i >= 1; i--, count++)*

*{*

*int temp = numbers[0];*

*numbers[0] = numbers[i];*

*numbers[i] = temp;*

*count += siftDown(numbers, 0, i - 1);*

*}*

*return count;*

*}*

*int QuicksortHoar(int \*mas, int first, int last) // Хоара*

*{*

*int mid, count;*

*int counts = 0;*

*int f = first, l = last;*

*mid = mas[(f + l) / 2]; //вычисление опорного элемента*

*do*

*{*

*while (mas[f]<mid){*

*f++;*

*counts++;*

*}*

*while (mas[l]>mid){*

*counts++;*

*l--;*

*}*

*if (f <= l) //перестановка элементов*

*{*

*counts++;*

*count = mas[f];*

*mas[f] = mas[l];*

*mas[l] = count;*

*f++;*

*l--;*

*}*

*} while (f<l);*

*counts++;*

*if (first<l) QuicksortHoar(mas, first, l);*

*counts++;*

*if (f<last) QuicksortHoar(mas, f, last);*

*return counts;*

*}*

*int SortBySelection(int\* arr, short n)*

*{*

*int count = 0;*

*int value, key;*

*for (int i = 0; i < n - 1; i++)*

*{*

*value = arr[i];*

*key = i;*

*for (int j = i + 1; j < n; j++)*

*{*

*count++;*

*if (arr[j] < arr[key])*

*key = j;*

*}*

*count++;*

*if (key != i)*

*{*

*arr[i] = arr[key];*

*arr[key] = value;*

*}*

*}*

*return count;*

*}*

*unsigned ShellSort(int\* arr, short n)*

*{*

*unsigned count = 0;*

*int step = n / 2;*

*while (step > 0)*

*{*

*for (int i = 0; i < (n - step); i++)*

*{*

*if (arr[i] < arr[i + step])*

*count++;*

*for (int j = i; j >= 0 && arr[j] > arr[j + step]; j--, count++)*

*{*

*int temp = arr[j];*

*arr[j] = arr[j + step];*

*arr[j + step] = temp;*

*}*

*}*

*step /= 2;*

*}*

*return count;*

*}*

*void GetArray(int\* arr, const short sz)*

*{*

*for (int i = 0; i < sz; i++)*

*{*

*arr[i] = rand();*

*}*

*}*

*void GetDisArray(const int\* arr1, int\* arr2, const short sz)*

*{*

*for (int i = 0; i < sz; i++) arr2[i] = arr1[i];*

*}*

*void GetOrderArray(const int\* arr1, int\* arr2, const short sz)*

*{*

*for (int i = 0; i < sz; i++)*

*{*

*arr2[i] = arr1[i];*

*}*

*for (int i = 1; i < sz; i++)*

*{*

*for (int j = i; j > 0 && arr2[j - 1] > arr2[j]; j--)*

*SwapElement(arr2, j);*

*}*

*}*

*void GetReorderArray(const int\* arr1, int\* arr2, short sz)*

*{*

*int temp;*

*for (int i = 0; i < sz; i++)*

*{*

*arr2[i] = arr1[i];*

*}*

*for (int i = 0; i < sz - 1; i++)*

*{*

*for (int j = i + 1; j < sz; j++)*

*{*

*if (arr2[i] < arr2[j])*

*{*

*temp = arr2[i];*

*arr2[i] = arr2[j];*

*arr2[j] = temp;*

*}*

*}*

*}*

*}*

*void GetInfo(const int\* arr, short sz)*

*{*

*cout << endl;*

*for (int i = 0; i < sz; i++)*

*{*

*cout << arr[i] << "-";*

*}*

*cout << endl;*

*}*

*int main(int argc, const char \* argv[])*

*{*

*setlocale(0, "");*

*while (cin.get() != 'q') {*

*cout << "\nВведите размер массива --> ";*

*int size;*

*cin >> size;*

*srand(50);*

*int\* default\_arr = new int[size];*

*GetArray(default\_arr, size);*

*cin.clear();*

*cout << "Анализ сортировки вставками" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка вставками: " << InsertSort(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка вставками: " << InsertSort(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка вставками: " << InsertSort(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки пузырьком" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSort(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSort(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSort(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки пузырьком улучшенной версии 1" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv1(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv1(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv1(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки пузырьком улучшенной версии 2" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv2(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv2(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пузырьком: " << BubbleSortv2(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки Хоара" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка Хоара: " << QuicksortHoar(ordered\_arr, 0, size-1);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка Хоара: " << QuicksortHoar(reordered\_arr, 0, size-1);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка Хоара: " << QuicksortHoar(disordered\_arr, 0, size-1);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки выбором" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка выбором: " << SortBySelection(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка выбором: " << SortBySelection(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка выбором: " << SortBySelection(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки Шелла" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка Шелла: " << ShellSort(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка Шелла: " << ShellSort(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка Шелаа: " << ShellSort(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Анализ сортировки пирамидой" << endl;*

*int \*ordered\_arr = new int[size];*

*int \*reordered\_arr = new int[size];*

*int \*disordered\_arr = new int[size];*

*GetDisArray(default\_arr, disordered\_arr, size);*

*GetOrderArray(disordered\_arr, ordered\_arr, size);*

*GetReorderArray(ordered\_arr, reordered\_arr, size);*

*cout << "Упорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пирамидой: " << heapSort(ordered\_arr, size);*

*cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";*

*cout << "Сортировка пирамидой: " << heapSort(reordered\_arr, size);*

*cout << "\nНеупорядоченный массив размера " << size << ":\n";*

*cout << "Сортировка пирамидой: " << heapSort(disordered\_arr, size);*

*GetInfo (disordered\_arr, size);*

*cout << endl;*

*GetInfo (ordered\_arr, size);*

*cout << endl;*

*GetInfo (reordered\_arr, size);*

*cout << endl;*

*delete[] ordered\_arr;*

*delete[] reordered\_arr;*

*delete[] disordered\_arr;*

*cout << "Введите любой символ чтобы проанализировать другой массив (q - выход) --> ";*

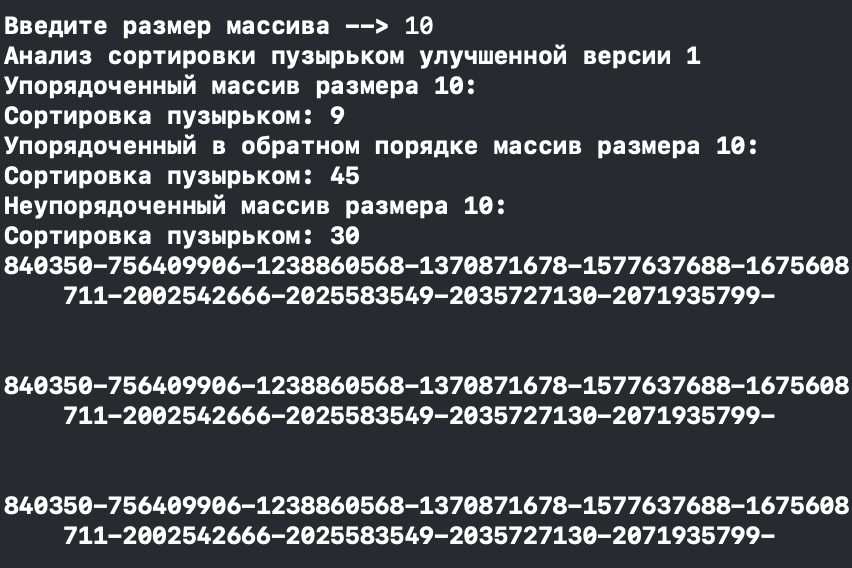
*cin.get();*

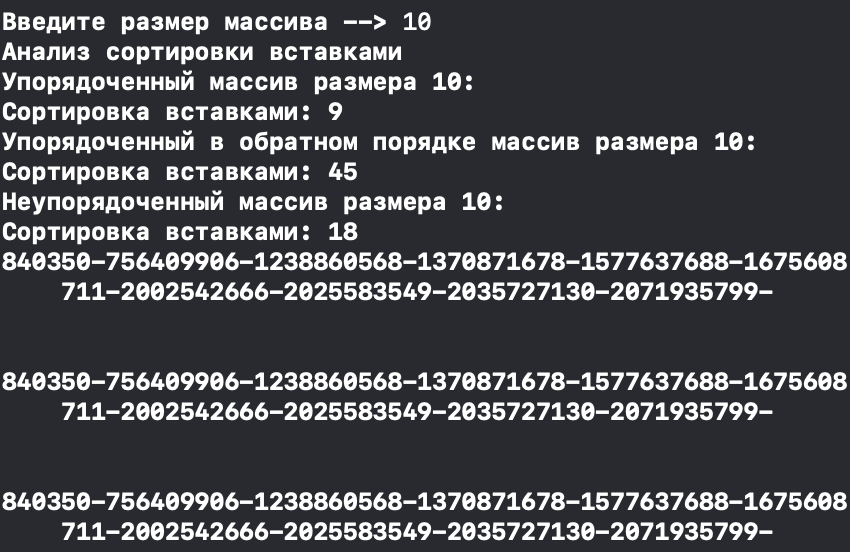
*}*

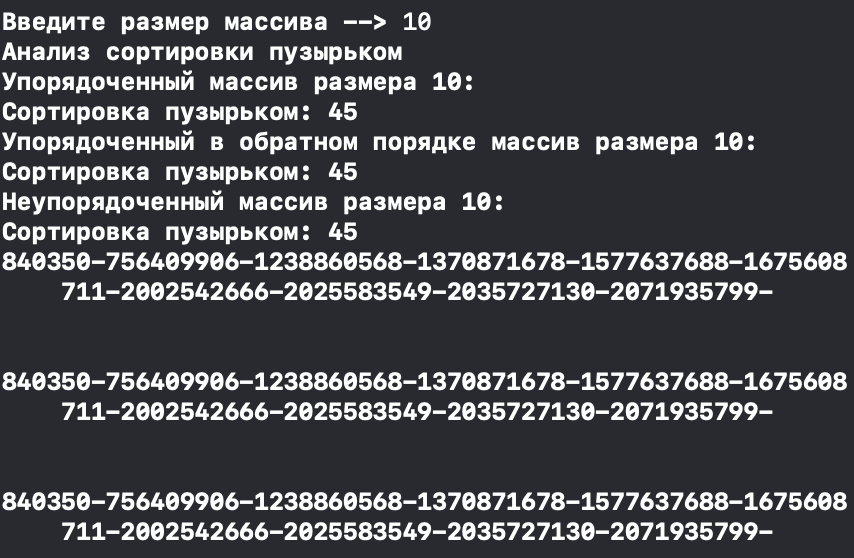
*return 0;*

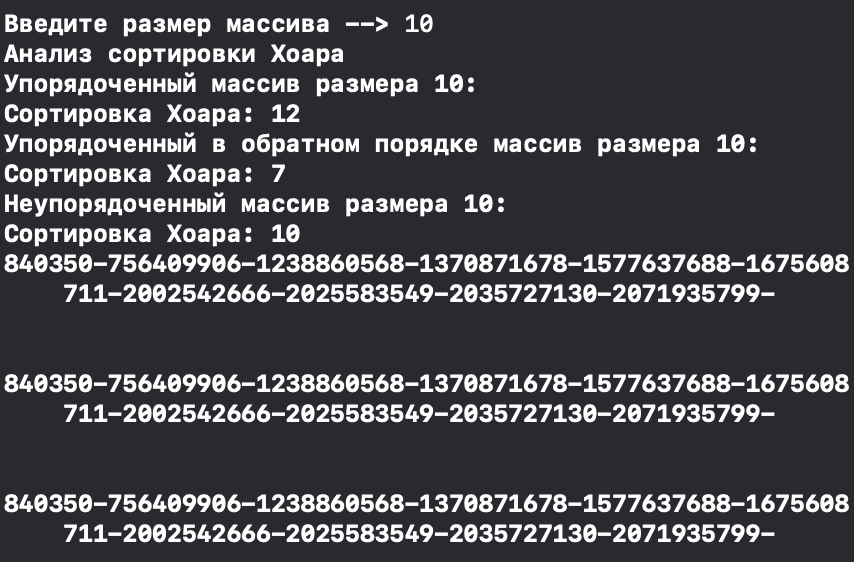
*}*

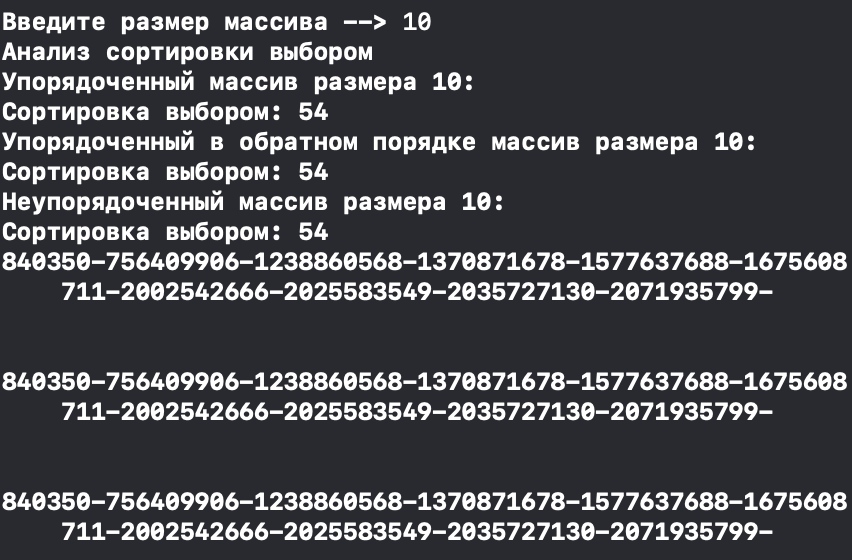
**Скриншоты работы:**

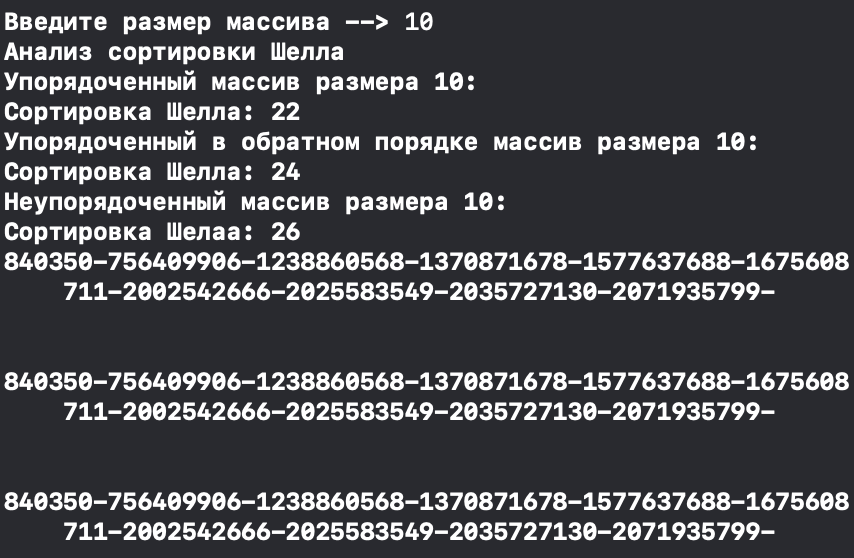
****

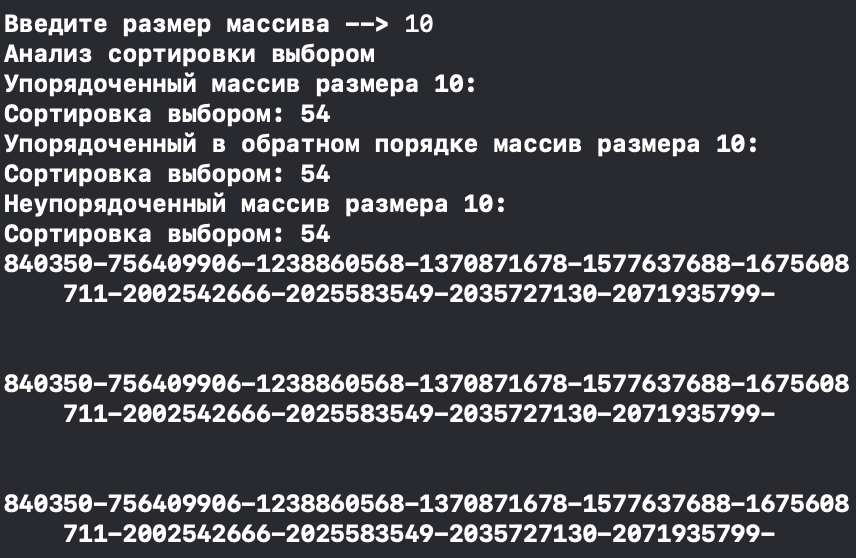
****

****

****

****

****

****

**Вывод:**

Исходя из полученных результатов и графиков зависимостей ФВС, можно сделать следующие выводы по основным методам сортировки.

Чем больше элементов в массиве отсортировано, тем эффективнее работает сортировка вставками. В упорядоченном массиве сортировка вставками производит O(N) число сравнений, а их среднее число равняется приблизительно O(N2).

При сортировке выбором число сравнений не зависит от упорядоченности элементов и равняется приблизительно O(N2).

При использовании сортировки обменом число сравнений элементов также не зависит от их первоначального порядка и равняется O(N2).

Число сравнений для улучшенной сортировки обменом №1 зависит от числа проходов, необходимых для сортировки. В худшем случае это число равняется O(N2). Если же массив уже упорядочен, выполняется всего один проход и N – 1 сравнение, что составляет O(N).

Число сравнений для улучшенной сортировки обменом №2 аналогичен сортировке обменом №1 и может лишь незначительно отличаться.

Сортировка Шелла при лучшем случае имеет O(Nlog2N) число сравнений, при худшем случае она деградирует до O(N2), но не деградирует при неудачных наборах данных.

Эффективность сортировки Хоара в значительной степени определяется правильностью выбора опорных (ведущих) элементов при формировании блоков. В худшем случае трудоемкость метода имеет ту же сложность, что и пузырьковая сортировка, то есть порядка O(n2). При оптимальном выборе ведущих элементов, когда разделение каждого блока происходит на равные по размеру части, трудоемкость алгоритма совпадает с быстродействием наиболее эффективных способов сортировки, то есть порядка O(N log N).

Несмотря на некоторую внешнюю сложность, пирамидальная сортировка является одной из самых эффективных для больших N. В худшем, лучшем и среднем случае требуется O(N·log2N) сравнений. На небольших N быстрее сортирвока Шелла.

New string for #5