МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №3

по дисциплине: Алгоритмы и структуры данных

тема:«Сравнительный анализ методов сортировки (Pascal/C)»

Выполнил: ст. группы ПВ-212
Гринченко Алина Сергеевна

Проверил:
Черников Сергей Викторович
Синюк Василий Григорьевич

Белгород 2022 г.

тема:«Сравнительный анализ методов сортировки (Pascal/C)»

Вариант 4

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

Задания

1. Изучить временные характеристики алгоритмов.
2. Изучить методы сортировки:
   - 2.1 включением;
   - 2.2 выбором
   - 2.3 обменом:
     - 2.3.1 улучшенная обменом 1;
     - 2.3.2 улучшенная обменом 2;
   - 2.4 Шелла;
   - 2.5 Хоара;
   - 2.6 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 таблицы 9, клетки которой содержат количество операций сравнения при выполнении алгоритма сортировки массива с заданным количеством элементов. Провести эксперимент для упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов (для каждого типа массива заполнить отдельную таблицу).
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.

Задание 3, 4

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <assert.h>

int N = 500;

// обмен значений двух переменных по адресам a и b
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

//сортировка включением
int insertionSort(int *a, const int size) {
    int count = 0;
    for (size_t i = 1; ++count && i < size; i++) {
        int t = a[i];
        int j = i;
        while (++count && j > 0 && ++count && a[j - 1] > t) {
            a[j] = a[j - 1];
            j--;
        }
        a[j] = t;
    }
    return count;
}

//сортировка выбором
int selectionSort(int *a, int size) {
    int count = 0;
    for (int i = 0; ++count && i < size - 1; i++) {
        int minPos = i;
        for (int j = i + 1; ++count && j < size; j++)
            if (++count && a[j] < a[minPos])
                minPos = j;
        swap(&a[i], &a[minPos]);
    }
    return count;
}

//сортировка обменом
int bubbleSort(int *a, int size) {
    int count = 0;
    for (size_t i = 0; ++count && i < size - 1; i++)
        for (size_t j = size - 1; ++count && j > i; j--)
            if (++count && a[j - 1] > a[j])
                swap(&a[j - 1], &a[j]);

    return count;
}

//Улучшенная обменом 1
/*После каждого прохода в сортировке обменом может быть сделана проверка,были
 ли совершены перестановки в течение данного прохода.Если
 перестановок не было, то это означает,что
 массив упорядочен и дальнейших проходов не требуется.*/
int bubbleSort1(int *a, int size) {
```

```
60      int count = 0;
61      int f;
62      int i = 1;
63      do {
64          f = 0;
65          for (size_t j = size - 1; ++count && j >= i; j--) {
66              if (++count && a[j - 1] >= a[j]) {
67                  swap(&a[j - 1], &a[j]);
68                  f = 1;
69              }
70          }
71          i++;
72      } while ((++count && f == 1) && (++count && i < size));
73
74      return count;
75  }
76
77  /*В отличие от улучшенной сортировки обменом 1 здесь в течение
       проходафиксируется
78   последний элемент, участвующий в обмене.В
79   очередном проходе этот элемент и все предшествующие в сравнении не участвуют,тк
80  .. все элементы до этой позиции уже отсортированы.*/
81  int bubbleSort2(int *a, int size) { //Улучшенная обменом 2
82      int count = 0;
83      int i = 1;
84      do {
85          int k = size;
86          for (size_t j = size - 1; ++count && j >= i; j--) {
87              if (a[j - 1] > a[j] && ++count) {
88                  swap(&a[j - 1], &a[j]);
89                  k = j;
90              }
91          }
92          i = k;
93      } while (++count && i < size); //Если перестановок не было, выйти из
       алгоритма
94
95      return count;
96  }
97
98  //сортировка Шелла
99  int shellSort(int *a, int size) {
100     int count = 0;
101     for (int d = size / 2; ++count && d > 0; d /= 2)
102         for (int i = d; ++count && i < size; ++i)
103             for (int j = i - d; (++count && j >= 0) && (++count && a[
       j] > a[j + d]); j -= d)
104                 swap(&a[j], &a[j + d]);
105     return count;
106 }
107
108 int QSort(int a[], int L, int R) {
109     int count = 0;
110     int x = a[L], i = L, j = R; // в качестве разделителя выбираем первый
       элемент
111     while (++count && i <= j) {
112         while (++count && a[i] < x)
113             i++;
114         while (++count && a[j] > x)
115             j--;
```

```
        if (++count && i <= j) {
            swap(&a[i], &a[j]);
            i++;
            j--;
        }
    }
    if (++count && L < j)
        count += QSort(a, L, j);
    if (++count && i < R)
        count += QSort(a, i, R);

    return count;
}

//функция сортировки методом Хоара
int HoarSort(int a[], int n) {
    return QSort(a, 0, n);
}

int Sift(int a[], int L, int R) {
    int count = 0;

    int i, j, x;
    i = L;
    j = 2 * L + 1;
    x = a[L];
    if (++count && (j < R) && (++count && a[j] < a[j + 1]))
        j++;

    while (++count && (j <= R) && (++count && x < a[j])) {
        swap(&a[i], &a[j]);
        i = j;
        j = 2 * j + 1;

        if (++count && (j < R) && (++count && a[j] < a[j + 1]))
            j++;
    }

    return count;
}

//пирамидальная функция сортировки
int HeapSort(int A[], int nn) {
    int count = 0;
    int L, R, x;
    L = nn / 2;
    R = nn - 1;

    //Построение пирамиды из исходного массива
    while (++count && L > 0) {
        L = L - 1;
        count += Sift(A, L, R);
    }

    //Сортировка: пирамида => отсортированный массив
    while (++count && R > 0) {
        x = A[0];
        A[0] = A[R];
        A[R] = x;
        R--;
```

```
176             count += Sift(A, L, R);
177         }
178 
179         return count;
180     }
181 
182     void output(int *a) {
183         for (int i = 0; i < N; ++i) {
184             printf("%d ", a[i]);
185         }
186         printf("\n");
187     }
188 
189     //Обратно упорядоченный массив
190     void BackSorted(int *a) {
191         for (int i = 0; i < N; ++i) {
192             a[i] = N - i;
193         }
194     }
195 
196     //упорядоченный массив
197     void Sorted(int *a) {
198         for (int i = 0; i < N; ++i) {
199             a[i] = i;
200         }
201     }
202 
203     //Неупорядоченный массив
204     void Unsorted(int *a) {
205         for (int i = 0; i < N; ++i) {
206             a[i] = rand() % 100;
207         }
208     }
209 
210     FILE *csvFiles[3];
211 
212     typedef int (*sort_func)(int *, int);
213 
214     typedef void(*gen_f)(int *);
215 
216 
217     int TIME_TEST(int *a, int n, gen_f gen, sort_func s) {
218         gen(a);
219         return s(a, N);
220     }
221 
222     int isSorted(int *a) {
223         for (int i = 1; i < N; i++) {
224             if (a[i - 1] > a[i]) {
225                 return 0;
226             }
227         }
228 
229         return 1;
230     }
231 
232     #define ARRAY_SIZE(array) \
233     sizeof(array) / sizeof(int)
234 
235     const int sizes[] = {5, 10, 15, 20, 25, 30, 35, 40, 45};
```

```
236 const int sizes_count = ARRAY_SIZE(sizes);
237
238 typedef struct {
239     char name[100];
240     sort_func func;
241 } sort_f;
242
243 int main() {
244     for (int i = 0; i < 3; i++) {
245         char filename[100];
246         sprintf(filename, "data%d.csv", i);
247         csvFiles[i] = fopen(filename, "w");
248     }
249
250     int a[1000];
251
252     gen_f gen_funcs[3] = {
253         BackSorted,
254         Sorted,
255         Unsorted,
256     };
257
258     sort_f sort_funcs[8] = {
259         {"insertion sort", insertionSort},
260         {"selection sort", selectionSort},
261         {"bubble sort",    bubbleSort},
262         {"bubble sort 1",  bubbleSort1},
263         {"bubble sort 2",  bubbleSort2},
264         {"shell sort",     shellSort},
265         {"Hoar sort",      HoarSort},
266         {"heap sort",      HeapSort},
267     };
268
269     for (int g_f = 0; g_f < 3; g_f++) {
270         fprintf(csvFiles[g_f], "Sort name;");
271         for (int i = 0; i < sizes_count; i++) {
272             fprintf(csvFiles[g_f], "%d;", sizes[i]);
273         }
274         fprintf(csvFiles[g_f], "\n");
275
276         for (int s_f = 0; s_f < 8; s_f++) {
277
278             fprintf(csvFiles[g_f], "%s;", sort_funcs[s_f].name);
279
280             for (int size = 0; size < sizes_count; size++) {
281                 N = sizes[size];
282                 int comps = TIME_TEST(a, N, gen_funcs[g_f],
283                 sort_funcs[s_f].func);
284                 assert(isSorted(a));
285
286                 fprintf(csvFiles[g_f], "%d;", comps);
287             }
288
289             fprintf(csvFiles[g_f], "\n");
290         }
291     }
292
293     for (int i = 0; i < 3; i++) {
294         fclose(csvFiles[i]);
295     }
```

```
296
297     return 0;
298 }
```

Задание 5

Результаты экспериментов (Неупорядоченный массив)

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	29	109	239	419	649	929	1259	1639	2069
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	29	109	239	419	649	929	1259	1639	2069
Обменом 1	32	117	252	437	672	957	1292	1677	2112
Обменом 2	34	119	254	439	674	959	1294	1679	2114
Шелла	31	91	153	249	303	404	577	643	716
Хоара	53	128	233	358	513	688	893	1118	1373
Пирамидальная	38	103	181	280	365	478	582	687	810

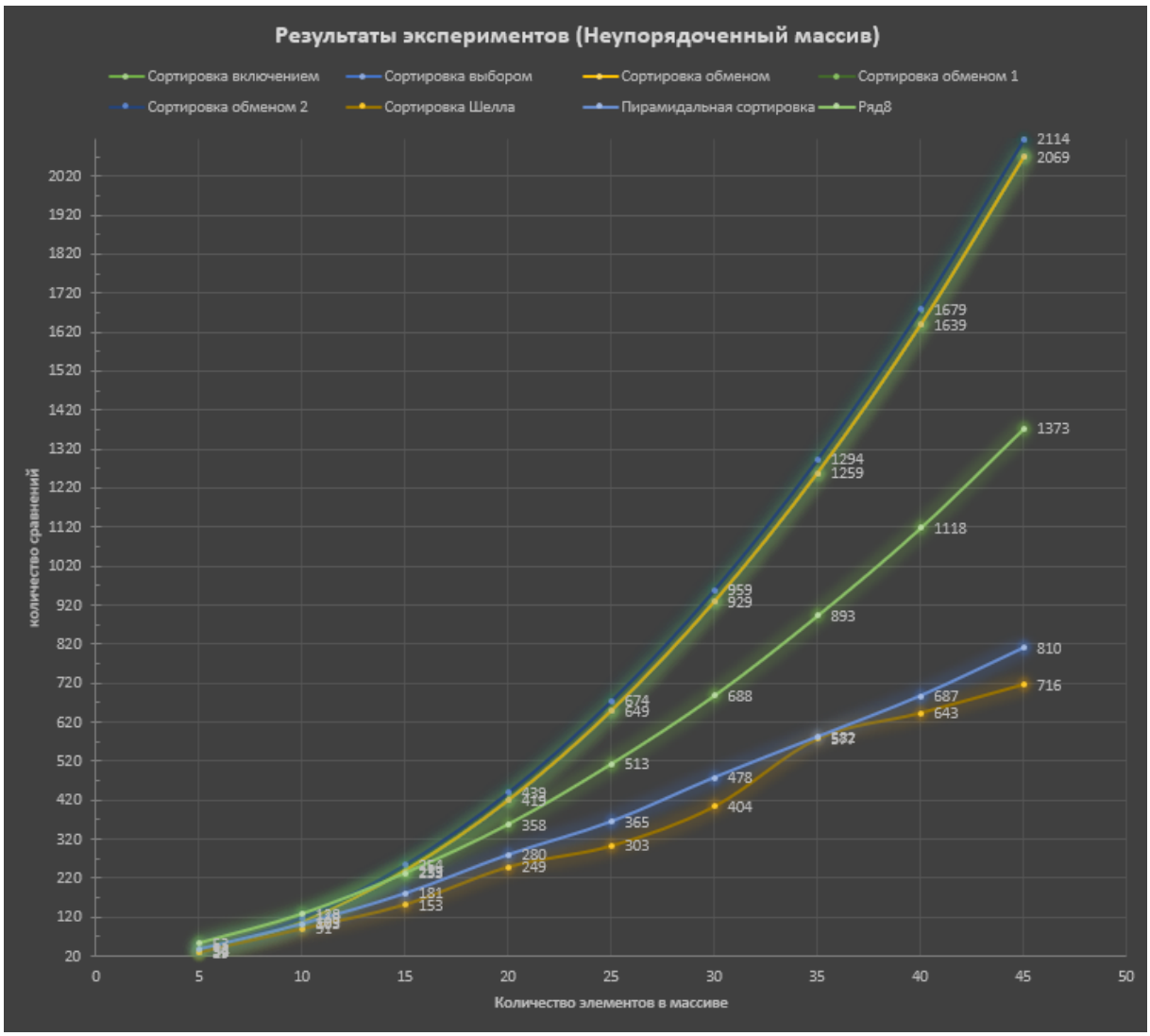
Результаты экспериментов (Упорядоченный массив)

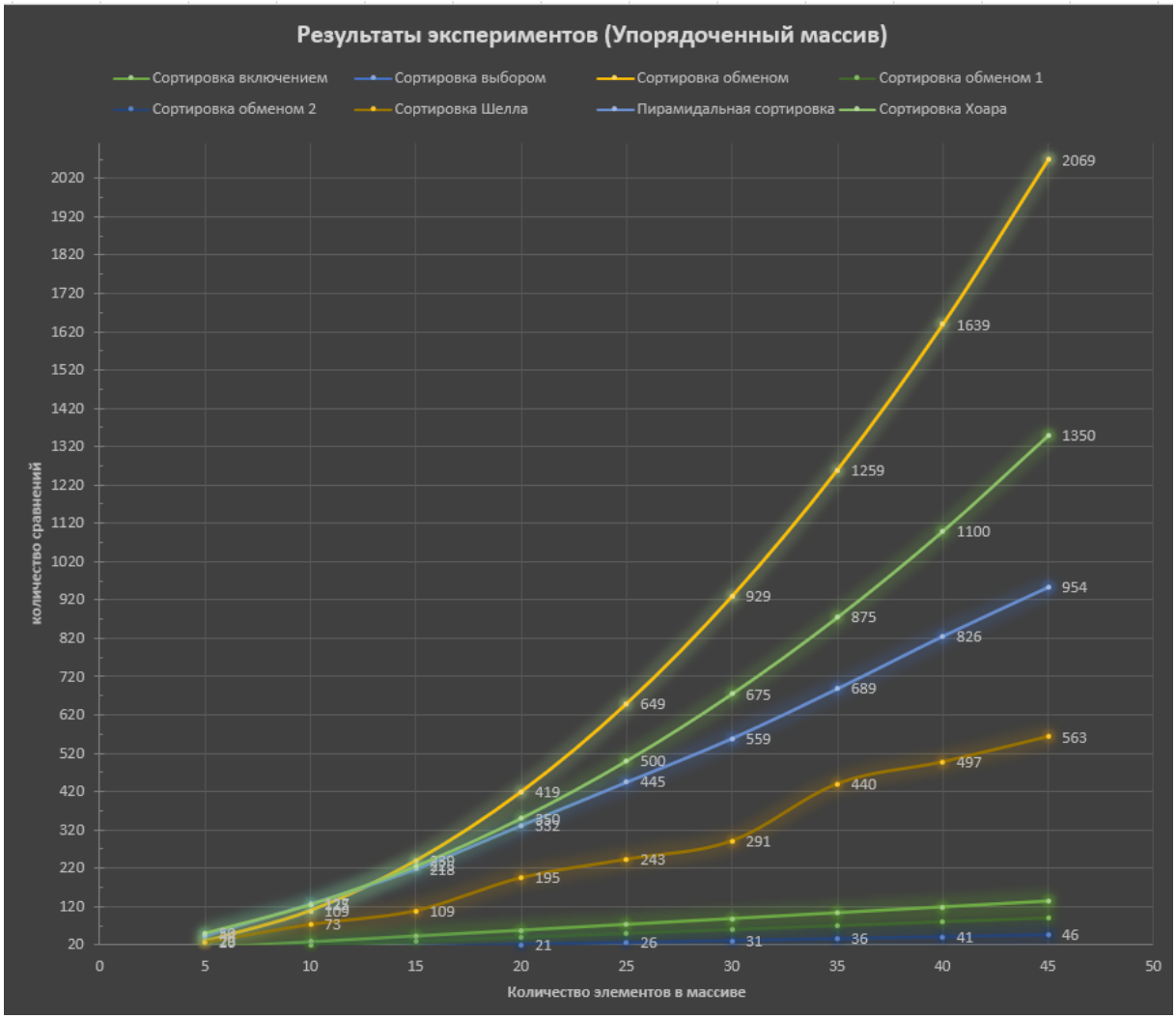
Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	13	28	43	58	73	88	103	118	133
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	29	109	239	419	649	929	1259	1639	2069
Обменом 1	10	20	30	40	50	60	70	80	90
Обменом 2	6	11	16	21	26	31	36	41	46
Шелла	26	73	109	195	243	291	440	497	563
Хоара	50	125	225	350	500	675	875	1100	1350
Пирамидальная	44	127	218	332	445	559	689	826	954

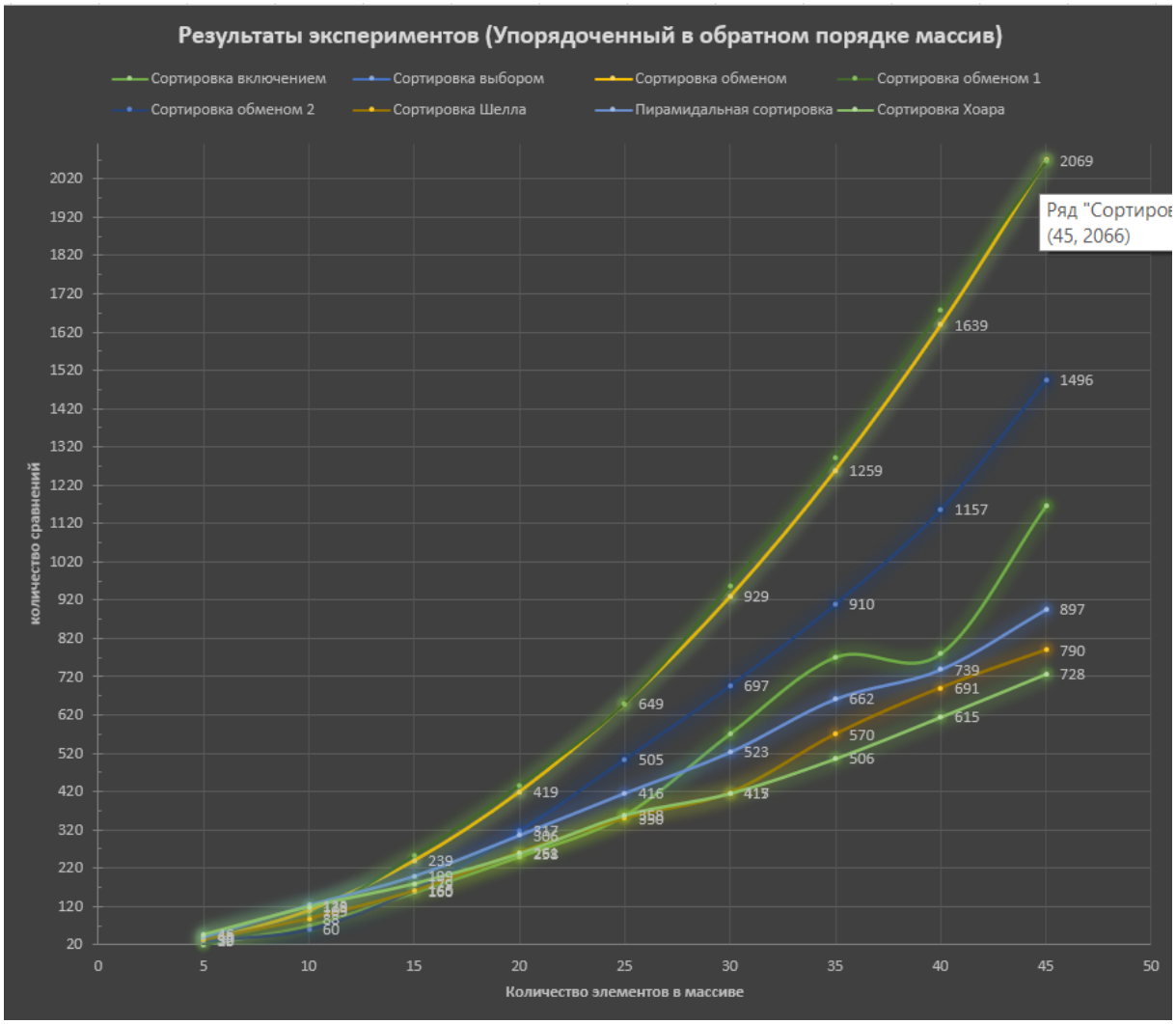
Результаты экспериментов (Упорядоченный в обратном порядке)

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	21	71	157	251	359	572	772	781	1168
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	29	109	239	419	649	929	1259	1639	2069
Обменом 1	19	95	252	436	650	957	1291	1677	2066
Обменом 2	30	60	163	317	505	697	910	1157	1496
Шелла	32	88	160	261	350	417	570	691	790
Хоара	46	119	179	258	358	415	506	615	728
Пирамидальная	40	123	199	306	416	523	662	739	897

Задание 6


Результаты экспериментов (Неупорядоченный массив)
Сортировка включением
Сортировка выбором
Сортировка обменом
Сортировка обменом 1
Сортировка обменом 2
Сортировка Шелла
Пирамидальная сортировка
Ряд8
Количество сравнений
Количество элементов в массиве
2114
2069
1679
1639
1373
1294
1259
1118
959
929
893
810
716
687
643
688
674
649
513
478
404
365
303
280
249
358
184
153
0
5
10
15
20
25
30
35
40
45
50
20
120
220
320
420
520
620
720
820
920
1020
1120
1220
1320
1420
1520
1620
1720
1820
1920
2020


Результаты экспериментов (Упорядоченный массив)
Сортировка включением
Сортировка выбором
Сортировка обменом
Сортировка обменом 1
Сортировка обменом 2
Сортировка Шелла
Пирамидальная сортировка
Сортировка Хоара
количество сравнений
Количество элементов в массиве
2069
1639
1350
1259
1100
954
929
875
826
689
675
649
563
559
500
497
445
440
419
350
332
291
243
218
195
109
73
46
41
36
31
26
21



Задание 7

Аналитические выражение функции зависимости количества операций сравнения от количества элементов в массиве.

включением	лучший случай: $N-1$; средний и худший: $(N-1)*N/2$
выбором	$(N-1)*N/2$
обменом	$(N-1)*N/2$
обменом 1	лучший случай: $N-1$; средний и худший: $(N-1)*N/2$
обменом 2	лучший случай: $N-1$; средний и худший: $(N-1)*N/2$
Шелла	зависит от выбранных шагов
Хоара	удачный разделитель:$N+2*(N/2)+\ldots+m*(N*m), m=logN$
пирамидальная	$[log_2N]+[log_2(N-1)]+\ldots+[log_22]$

Задание 8

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

Название сортировки	лучший случай	средний случай	худший случай
включением	$O(N)$	$O(N^2)$	$O(N^2)$
выбором	$O(N^2)$	$O(N^2)$	$O(N^2)$
обменом	$O(N^2)$	$O(N^2)$	$O(N^2)$
обменом 1	$O(N)$	$O(N^2)$	$O(N^2)$
обменом 2	$O(N)$	$O(N^2)$	$O(N^2)$
Шелла	$O(N)$	$O(N^2)$	$O(N * log^2 N)$
Хоара	$O(N * logN)$	$O(N^2)$	$O(N * logN)$
пирамидальная	$O(N * logN)$	$O(N * logN)$	$O(N * logN)$

Вывод: В ходе выполнения данной лабораторной работы были изучены методы сортировки, приобретены навыки в проведении сравнительного анализа различных методов сортировки