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: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверил:
асс. Солонченко Роман Евгеньевич

Белгород 2023 г.

Лабораторная работа №3 «Сравнительный анализ методов сортировки (Pascal/C)»

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

Содержание отчета:

- Тема лабораторной работы;

- Цель лабораторной работы;

- Условия задач и их решение;

- Вывод.

Листинг программы:

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))

/*============СОРТИРОВКА ВКЛЮЧЕНИЕМ============*/

/* функция сортировки включением  */
long long sis(int A[], int nn) {
    long long number_of_comparisons = 0;
    int i, j, k;

    for (j = 1; j < nn; j++) {
        k = A[j];
        i = j - 1;

        while (k < A[i] && i >= 0) {
            number_of_comparisons++;

            A[i + 1] = A[i];
            i -= 1;
        }

        number_of_comparisons++;

        A[i + 1] = k;
    }

    return number_of_comparisons + (nn - 1);
}
```

```
/*==============СОРТИРОВКА ВЫБОРОМ===============*/

/*функция сортировки выбором  */
long long str_sel(int A[], int nn) {
    long long number_of_comparisons = 0;
    int i, j, x, k;

    for (i = 0; i < nn - 1; i++) {
        x = A[i];
        k = i;

        for (j = i + 1; j < nn; j++)
            if (A[j] < x) {
                k = j;
                x = A[k];
            }

        number_of_comparisons += (nn - (i + 1));

        A[k] = A[i];
        A[i] = x;
    }

    return number_of_comparisons + (nn - 1);
}
```

```
/*============СОРТИРОВКА ОБМЕНОМ=============*/

/*  функция сортировки обменом  */
long long bbl_sort(int A[], int nn) {
    long long number_of_comparisons = 0;
    int i, j, k, p;

    for (i = 0; i < nn - 1; i++) {
        p = 0;

        for (j = nn - 1; j > i; j--) {
            number_of_comparisons++;

            if (A[j] < A[j - 1]) {
                k = A[j];
                A[j] = A[j - 1];
                A[j - 1] = k;
                p = 1;
            }
        }

        number_of_comparisons += (nn - i);

        /* Если перестановок не было, то сортировка выполнена */
        if (!p)
            break;
    }

    return number_of_comparisons + (nn - 1);
}
```

```cpp
long long bbl_sort1(int arr[], int n) {
    long long number_of_comparisons = 0;
    int temp;
    bool swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            number_of_comparisons++;

            if (arr[j] > arr[j + 1]) {
                // меняем элементы местами
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        number_of_comparisons += (n - i);

        // если на текущей итерации не было ни одного обмена,
        // то массив уже отсортирован и можно завершить процесс
        if (swapped == false)
            break;
    }

    return number_of_comparisons + (n - 1);
}
```

```
long long bbl_sort2(int arr[], int n) {
    long long number_of_comparisons = 0;
    int i, j, temp;
    int last_swap_index = n - 1;

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

        for (int j = 0; j < last_swap_index; j++) {
            number_of_comparisons++;

            if (arr[j] > arr[j + 1]) {
                // меняем элементы местами
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                current_swap_index = j;
            }
        }

        number_of_comparisons += (last_swap_index + 1);

        // если на текущей итерации не было ни одного обмена,
        // то массив уже отсортирован и можно завершить процесс
        if (current_swap_index == -1)
            break;

        last_swap_index = current_swap_index;
    }

    return number_of_comparisons + (n - 1);
}

/*============СОРТИРОВКА МЕТОДОМ ШЕЛЛА=============*/

// Функция вычисления натурального логарифма
double ln(double x) {
    // Вычисляем сумму ряда Тейлора, используя 1000 слагаемых
    double sum = 0;

    for (int i = 1; i <= 1000; i++)
        sum += pow((-1), (i + 1)) * pow((x - 1), i) / i;

    return sum;
}
```

```
// Функция для сортировки массива методом Шелла
long long shell_sort(int arr[], int n) {
    long long number_of_comparisons = 0;

    // Начинаем с большего шага
    for (int gap = n / 2; gap > 0; gap /= 2) {
        number_of_comparisons++;


        // Проходим по элементам массива с шагом gap
        for (int i = gap; i < n; i++) {
            // Сохраняем текущий элемент в переменную temp
            int temp = arr[i];

            // Сдвигаем предыдущие элементы, которые больше
текущего, на один шаг вперед
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -=
gap) {
                number_of_comparisons++;
                arr[j] = arr[j - gap];
            }

            number_of_comparisons++;
            // Вставляем текущий элемент на правильную позицию
            arr[j] = temp;
        }

        number_of_comparisons += (n - gap);
    }

    return number_of_comparisons + 1;
}
```

```c
/*============СОРТИРОВКА МЕТОДОМ ХОАРА=============*/
// Функция для обмена двух элементов массива
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

/* Эта функция принимает последний элемент в качестве опорного,
помещает
этот элемент в правильное положение в отсортированном массиве и
помещает
все меньшие (меньше опорного) элементы слева от него и все
большие
элементы справа от него */
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // опорный элемент
    int i = (low - 1); // индекс меньшего элемента

    for (int j = low; j <= high - 1; j++) {
        // Если текущий элемент меньше или равен опорному
        if (arr[j] <= pivot) {
            i++; // увеличиваем индекс меньшего элемента
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);
}

/* Функция для реализации алгоритма быстрой сортировки
arr[] --> Массив для сортировки,
low  --> Начальный индекс,
high  --> Конечный индекс */
long long q_sort(int arr[], int low, int high, long long
number_of_comparisons) {
    if (low < high) {
        /* pi - это разделительный индекс, arr[p] сейчас на
правильном месте */
        int pi = partition(arr, low, high);

        number_of_comparisons += (2 * (high - low));
        // Рекурсивно сортируем элементы до разделителя и после
разделителя
        q_sort(arr, low, pi - 1, number_of_comparisons);
        q_sort(arr, pi + 1, high, number_of_comparisons);
    }

    return number_of_comparisons + 1;
}
```

```
long long hoar_sort(int arr[], int high) {
    return q_sort(arr, 0, high, 0);
}

/*============ПИРАМИДАЛЬНАЯ СОРТИРОВКА=============*/


/* ============================================== */
void sift(int A[], int L, int R) {
    int i, j, x, k;

    i = L;
    j = 2 * L + 1;
    x = A[L];

    if ((j < R) && (A[j] < A[j + 1]))
        j++;

    while ((j <= R) && (x < A[j])) {
        k = A[i];
        A[i] = A[j];
        A[j] = k;
        i = j;
        j = 2 * j + 1;

        if ((j < R) && (A[j] < A[j + 1]))
            j++;
    }
}
```

```c
/* ************************************************** */

/* пирамидальная функция сортировки  */
long long heap_sort(int A[], int nn) {
    long long number_of_comparisons = 0;
    int L, R, x, i;

    L = nn / 2;
    R = nn - 1;

    /*  Построение пирамиды из исходного массива */
    while (L > 0) {
        number_of_comparisons++;
        L = L - 1;
        sift(A, L, R);
    }

    number_of_comparisons++;

    /* Сортировка: пирамида => отсортированный массив */
    while (R > 0) {
        number_of_comparisons++;
        x = A[0];
        A[0] = A[R];
        A[R] = x;
        R--;
        sift(A, L, R);
    }

    return number_of_comparisons + 1;
}

void generate_random_array(int *array, const size_t size) {
    srand(time(0));

    for (size_t i = 0; i < size; i++)
        array[i] = rand() % 10000;
}

int is_ordered(int *array, size_t size) {
    for (size_t i = 1; i < size; i++)
        if (array[i] < array[i - 1])
            return 0;

    return 1;
}
```

```c
void output_array(int *array, size_t size) {
    printf("[");

    for (size_t i = 0; i < size; i++) {
        printf("%d", array[i]);

        if (i < size - 1)
            printf(", ");
    }

    printf("]\n");
}

typedef struct sort_function {
    long long (*sort)(int[], int);

    char *name;
} sort_function;


typedef struct generation_function {
    void (*generate)(int *, size_t);

    char *name;
} generation_function;
```

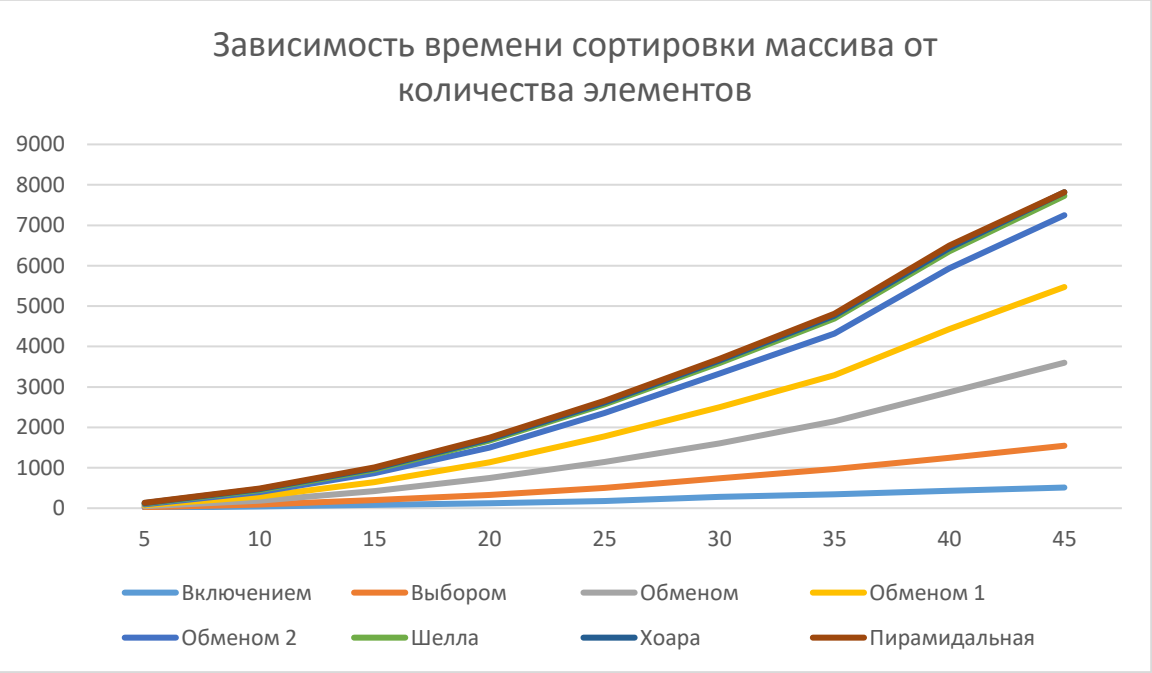
```c
void check_time(long long (*sort_func)(int *, int), void
(*generate_func)(int *, size_t), size_t size,
                char *experiment_name) {
    static size_t run_counter = 1;
    static int inner_buffer[100000000];

    generate_func(inner_buffer, size);

    printf("Run #%zu | ", run_counter++);
    printf("Name: %s\n", experiment_name);

    long long count_comparison = sort_func(inner_buffer, size);

    printf("Status: ");

    if (is_ordered(inner_buffer, size)) {
        printf("OK! Count comparison %lld\n\n",
count_comparison);

        char filename[256];

        sprintf(filename, "data/%s.csv", experiment_name);

        FILE *f = fopen(filename, "a");

        if (f == NULL) {
            printf("File open error %s", filename);

            exit(1);
        }

        fprintf(f, "%llu; %lld\n", size, count_comparison);

        fclose(f);
    } else {
        printf("Wrong!\n");

        output_array(inner_buffer, size);
        exit(1);
    }
}
```

```
void time_experiment() {
    sort_function sorts[] = {
            {sis,        "sis"},
            {str_sel,    "str_sel"},
            {bbl_sort,   "bbl_sort"},
            {bbl_sort1,  "bbl_sort1"},
            {bbl_sort2,  "bbl_sort2"},
            {shell_sort, "shell_sort"},
            {hoar_sort,  "hoar_sort"},
            {heap_sort,  "heap_sort"},
    };
    const unsigned FUNCS_N = ARRAY_SIZE(sorts);
    generation_function generation[] = {
            {generate_random_array, "random"}
    };
    const unsigned CASES_N = ARRAY_SIZE(generation);

    for (size_t size = 5; size <= 45; size += 5) {
        printf("************************************\n");
        printf("Size: %llu\n", size);

        for (size_t i = 0; i < FUNCS_N; i++)
            for (size_t j = 0; j < CASES_N; j++) {
                static char filename[128];

                sprintf(filename, "%s_%s_time", sorts[i].name,
generation[j].name);

                check_time(sorts[i].sort,
generation[j].generate, size,
                           filename);
            }

        printf("\n");
    }
}

int main() {
    time_experiment();

    return 0;
}
```

Временные характеристики алгоритмов.

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	11	40	84	122	175	278	343	428	512
Выбором	14	54	119	209	324	464	629	819	1034
Обменом	25	93	223	415	645	865	1178	1623	2053
Обменом 1	25	105	223	394	633	893	1138	1558	1873
Обменом 2	21	105	215	362	575	827	1028	1508	1777
Шелла	20	54	92	170	218	263	373	420	478
Хоара	11	21	31	41	51	61	71	81	91
Пирамидальная	8	16	23	31	38	46	53	61	68

График зависимости ФВС:



Порядок ФВС:

Сортировки	Порядок ФВС
Включением	$O(N^2)$
Выбором	$O(N^2)$
Обменом	$O(N^2)$
Обменом 1	$O(N^2)$
Обменом 2	$O(N^2)$
Шелла	$O(N * \log^2 N)$
Хоара	$O(N * \log N)$
Пирамидальная	$O(N * \log N)$

Вывод: в ходе выполнения лабораторной работы были изучены методы сортировки массивов и приобретены навыки в проведении сравнительного анализа различных методов сортировки.