



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра прикладной математики
Практическое задание № 5+
по дисциплине «Структуры данных и алгоритмы»

АЛГОРИТМЫ СОРТИРОВКИ

Бригада 2 ТАДЖИБАЕВ ЗАВКИДДИН

Группа ПМ-25

Вариант 7

Преподаватель ТРАКИМУС ЮРИЙ ВИКТОРОВИЧ

Новосибирск, 2023

1 Задание

Упорядочить таблицу, построенную в практическом задании «ТАБЛИЦЫ» (варианты 3б, 3в), по новому ключу – по возрастанию номеров команд, не вошедших в претенденты ни на первое, ни на последнее место.

Для упорядочения использовать метод:

- а) сортировка вставками;
- б) Сортировка пузырьком;
- в) Сортировка выбором.

2 Анализ программы

Входные данные: В файле "Table.txt" содержится хеш-таблица. Каждый элемент этой таблицы состоит из номера команды и количества её вхождений в первую, последнюю и топ-3 местах рейтинга.

Выходные данные: Упорядоченная таблица по возрастанию номеров команд, не вошедших в претенденты ни на первое, ни на последнее место, "Не удалось открыть файл Table.txt.", "Файл Table.txt пуст."

Анализ работы программы : Данная программа предоставляет пользователю возможность сортировать данные из файла с помощью различных методов сортировки. После выбора метода сортировки и выполнения соответствующего алгоритма, отсортированные данные выводятся на экран. Пользователь может продолжать работу с программой или завершить ее. Insertion sort (Сортировка вставками) : Алгоритм сортировки вставками. Проходит по элементам массива по одному за раз, начиная со второго. Для каждого элемента, алгоритм сравнивает его с предыдущими элементами и вставляет его в правильную позицию в уже отсортированной части массива. Алгоритм продолжает проходить по всем элементам до тех пор, пока не достигнет конца массива, упорядочивая все элементы по мере прохождения. Bubble sort (Сортировка пузырьком): Алгоритм сортировки пузырьком. Сравнивает пары соседних элементов и меняет их местами, если они находятся в неправильном порядке. Этот процесс повторяется до тех пор, пока все элементы не будут расположены в правильном порядке. Selection sort (Сортировка выбором): Алгоритм сортировки выбором. На каждом шаге выбирает минимальный элемент из неотсортированной части массива и меняет его местами с первым неотсортированным элементом. Этот процесс повторяется до тех пор, пока весь массив не будет отсортирован.

Замер времени работы алгоритмов: В программе используется стандартная библиотека <chrono> для замера времени работы алгоритмов. Замер времени производится без учета времени на считывание и вывод данных. Каждый тест проводится несколько раз для усреднения результатов и учета возможных погрешностей, связанных с вычислительными способностями и другими системными факторами, влияющими на время работы алгоритма. Количество данных увеличивается в 10 раз, начиная с 1000 элементов, для наглядности зависимости времени от количества данных. Результаты замера времени выводятся на экран.

3 Программа

```
#include <stdio.h>
#include <locale.h>
#include <windows.h>
#include <algorithm>
#include <chrono>

using namespace std;
```

```

using namespace std::chrono;

const UINT N = 1000000;
UINT K = 0, L = 0, H = 0, k1 = 0, k2 = 0;

struct elem
{
    UINT Num, first, last, top;
    elem(UINT _Num = 0, UINT _first = 0, UINT _last = 0, UINT _top = 0) :
        Num(_Num), first(_first), last(_last), top(_top) { };
};

struct table
{
    elem *e[N]{ };
    UINT size = 0;

    void Insertion_sort(UINT l, UINT r)
    {
        for (UINT i = l + 1; i <= r; i++)
        {
            elem *key = e[i];
            int j = i - 1;

            for ( ; j >= 0 && e[j]->Num > key->Num; j--, k1++)
                e[j + 1] = e[j];

            e[j + 1] = key;
        }
    }

    void Bubble_sort(UINT l, UINT r)
    {
        for (UINT i = l; i < r; i++)
            for (UINT j = l; j < r - i; j++)
                if (e[j]->Num > e[j + 1]->Num)
                {
                    swap(e[j], e[j + 1]);
                    k2++; k1++;
                }
    }

    void Selection_sort(UINT l, UINT r)
    {
        for (UINT i = l; i < r; i++)
        {
            UINT minIndex = i;

            for (UINT j = i + 1; j <= r; j++)
                if (e[j]->Num < e[minIndex]->Num)
                {

```

```

        k1++;
        minIndex = j;
    }

    if (minIndex != i)
    {
        swap(e[i], e[minIndex]);
        k2++;
    }
}

void insert(elem *el)
{
    if (el->first || el->last)
        e[size] = el;

    else
    {
        for (UINT i = size; i > H; i--)
            e[i] = e[i - 1];
        e[H] = el;
        H++;
    }
    size++;
}

inline bool isEmpty() { return size > 0; }

void print_table()
{
    printf_s("   №   First, %%   Last, %%   Top, %%\n");
    for (UINT i = 0; i < size; i++)
        printf_s("%14d %10.1f %10.1f %10.1f\n", e[i]->Num, e[i]->first
            * 100. / K, e[i]->last * 100. / K, e[i]->top * 100. / K);
}

} T{ }, T2{ };

bool input()
{
    FILE *f = NULL;

    fopen_s(&f, "Table.txt", "r");
    if (f)
    {
        UINT Num = 0, first = 0, last = 0, top = 0;
        for (elem *e = NULL;
            fscanf_s(f, "%d %d %d %d ", &Num, &first, &last, &top) == 4;
            e = new elem(Num, first, last, top), T.insert(e), K++);
    }
}

```

```

        fclose(f);
        return true;
    }
    else
        return false;
}

void measureSort(table &T, table &T2, const char *sortName, void (table::
*sortFunc)(UINT, UINT))
{
    T = T2;

    auto start = high_resolution_clock::now();
    (T.*sortFunc)(0, H - 1);
    (T.*sortFunc)(H, T.size - 1);
    auto end = high_resolution_clock::now();

    duration<double> duration = end - start;
    printf_s("%-15s%13d %13d %13d %13.8lf\n", sortName, K, k1, k2,
        duration.count());

    k1 = 0; k2 = 0; T = T2;
}

int main()
{
    setlocale(0, " ");
    UINT cp = GetConsoleCP(), outcp = GetConsoleOutputCP();
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    if (input())
    {
        if (T.isEmpty())
        {
            T2 = T;
            bool exitFlag = false;
            for ( ; !exitFlag; )
            {
                printf_s("Выберите способ сортировки:\n1 - Insertion_sort\n2
- Bubble_sort\n3 - Selection_sort\n4 - Таблица\n");
                USHORT a = 0;
                L = 0;
                scanf_s("%hu", &a);
                switch (a)
                {
                    case 1: T.Insertion_sort(0, H - 1);
                        T.Insertion_sort(H, T.size - 1); break;
                    case 2: T.Bubble_sort(0, H - 1);
                        T.Bubble_sort(H, T.size - 1); break;
                    case 3: T.Selection_sort(0, H - 1);

```

```

        T.Selection_sort(H, T.size - 1); break;
    case 4: T = T2;
        printf_s("      Название      Количество      Сравне-
ния Перестановки      Время, с\n");
        measureSort(T, T2, "Insertion_sort", &table::
            Insertion_sort);
        measureSort(T, T2, "Bubble_sort", &table::
            Bubble_sort);
        measureSort(T, T2, "Selection_sort", &table::
            Selection_sort);
        break;
    default: printf_s("Некорректный вариант сортировки\n");
}
if (a <= 3) T.print_table();
printf_s("\nХотите продолжить работу:\n1 - ДА\n0 - НЕТ\n");
UCHAR c = 0;
scanf_s("%hhu", &c);
exitFlag = c == 0;
}
}
else
    printf_s("Файл Table.txt пуст.");
}
else
    printf_s("Не удалось открыть файл Table.txt.");

SetConsoleCP(cp);
SetConsoleOutputCP(outcp);
return 0;
}

```

4 Набор тестов

| № | Входные данные | Назначение |
|---|--------------------------------|---------------------------------------|
| 1 | | Файл Table.txt отсутствует. |
| 2 | | Файл Table.txt пуст. |
| 3 | 11 0 0 1 9 1 2 3 7 0 0 1 | Проверка работоспособности программы. |

5 Результаты работы программы

| № | Ввод/Вывод программы |
|---|--|
| 1 | Не удалось открыть файл Table.txt. |
| 2 | Файл Table.txt. пуст. |
| 3 | Выберите способ сортировки: 1 - Insertion_sort 2 - Bubble_sort 3 - Selection_sort 4 - Таблица 1 |

| № | First, % | Last, % | Top3, % | |
|----|----------|---------|---------|--|
| 7 | 0.0 | 0.0 | 33.3 | |
| 11 | 0.0 | 0.0 | 33.3 | |
| 9 | 33.3 | 66.7 | 100.0 | |

Хотите продолжить работу:

1 - ДА
0 - НЕТ
1

Выберите способ сортировки:

1 - Insertion_sort
2 - Bubble_sort
3 - Selection_sort
4 - Таблица
2

| № | First, % | Last, % | Top3, % | |
|----|----------|---------|---------|--|
| 7 | 0.0 | 0.0 | 33.3 | |
| 11 | 0.0 | 0.0 | 33.3 | |
| 9 | 33.3 | 66.7 | 100.0 | |

Хотите продолжить работу:

1 - ДА
0 - НЕТ
1

Выберите способ сортировки:

1 - Insertion_sort
2 - Bubble_sort
3 - Selection_sort
4 - Таблица
3

| № | First, % | Last, % | Top3, % | |
|----|----------|---------|---------|--|
| 7 | 0.0 | 0.0 | 33.3 | |
| 11 | 0.0 | 0.0 | 33.3 | |
| 9 | 33.3 | 66.7 | 100.0 | |

Хотите продолжить работу:

1 - ДА
0 - НЕТ
1

Выберите способ сортировки:

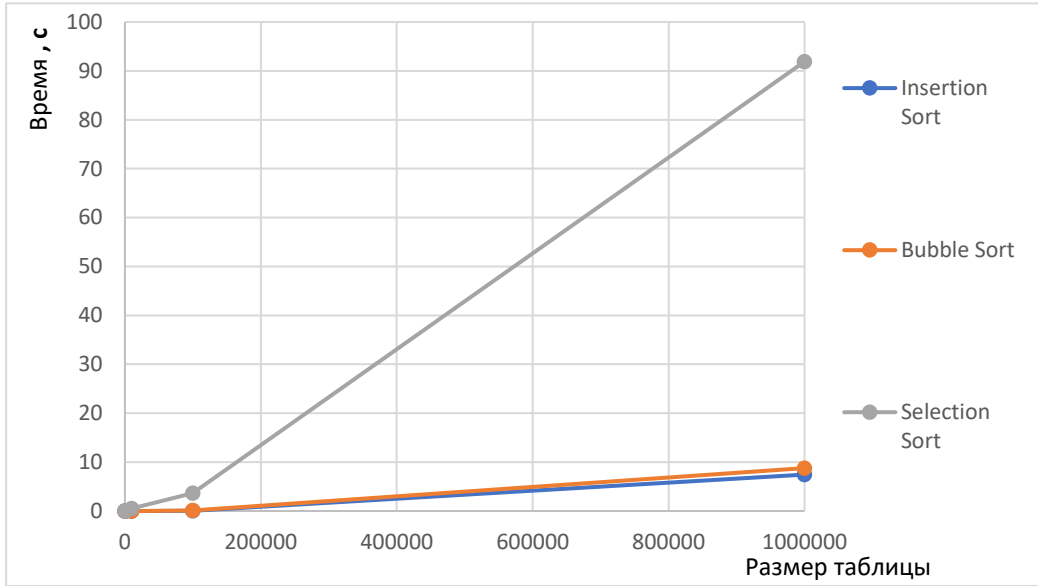
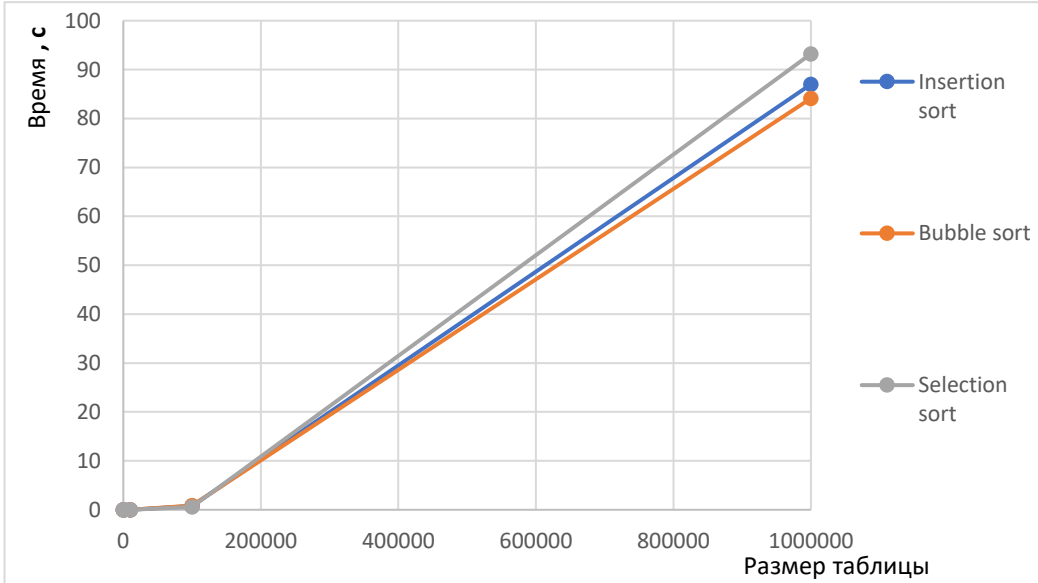
1 - Insertion_sort
2 - Bubble_sort
3 - Selection_sort
4 - Таблица
4

| Название | Количество | Сравнения | Перестановки | Время, с |
|----------------|------------|-----------|--------------|------------|
| Insertion_sort | 3 | 2 | 0 | 0.00000470 |
| Bubble_sort | 3 | 1 | 1 | 0.00000180 |
| Selection_sort | 3 | 1 | 1 | 0.00000200 |

Хотите продолжить работу:

1 - ДА
0 - НЕТ
0

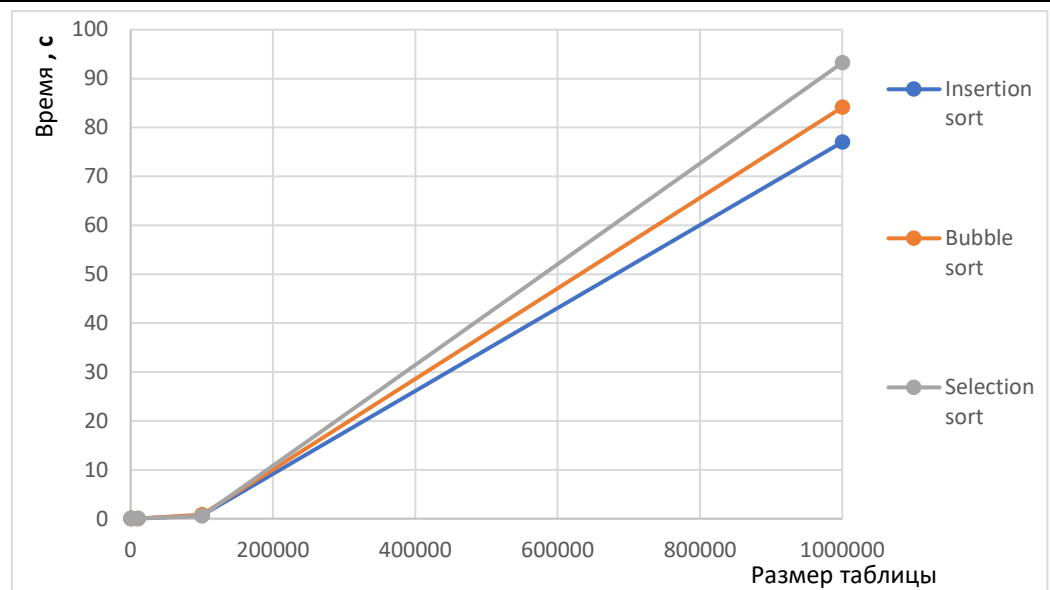
6 График сложности алгоритма

| № | Инверсии, % | Зависимость времени работы от входных данных |
|---|-------------|---|
| 1 | 0 | <div><p>На этом графике значения времени выполнения алгоритмов подтверждают, что все данные соответствуют теоретической сложности алгоритмов при сортировке уже упорядоченных данных. Insertion Sort и Bubble Sort имеют линейную сложность $O(n)$, и время выполнения пропорционально количеству входных данных. Selection Sort имеет квадратичную сложность $O(n^2)$, и время выполнения увеличивается пропорционально квадрату количества входных данных. Данные в графике подтверждают это соответствие.</p></div> |
| 2 | 50 | <div></div> |

На этом графике значения времени выполнения алгоритмов также соответствуют теоретической сложности частично упорядоченных данных. Как и ожидалось, все алгоритмы имеют квадратичную сложность $O(n^2)$

3

100



Аналогично предыдущим графикам, значения времени выполнения в графике соответствуют теоретической сложности алгоритмов при сортировке данных с отношением обратного порядка. Все алгоритмы имеют квадратичную сложность $O(n^2)$