**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №15

Вариант №11

Выполнил студент группы РИС-20-2б

Савельев Виталий Антонович

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с другими, более эффективными алгоритмами сортировки массивов, как:

1. Алгоритм Хоара;
2. Алгоритм Шелла;

**Постановка задачи**

Задача – реализовать на языке C++ функции, которые сортируют массивы с структурами.

* Составить свою структуру, содержащие несколько элементов;
* Реализовать 2 функции, которые осуществляют сортировку элементов;
* Организовать вывод массива структур до и после сортировки;

Структура «Раскладка клавиатуры»:

* Название раскладки;
* Автор раскладки;
* Язык раскладки;
* Год выпуска;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

**Реализация программы:**

Для использования операторов ввода-вывода необходимо подключить два заголовочных файла:

1. iostream, содержащий объекта потоков ввода-вывода: std::cin и std::cout;
2. string, позволяющий работать со строками;

Также, для удобства было подключено пространство имён std, который в себе содержит описанные выше потоки ввода-вывода. Заголовочные файлы подключаются при помощи директивы #include.

#include <iostream>

#include <string>

using namespace std;

Сама структура, описанная в постановке задача объявляется следующим образом:

struct KeyboardLayout

{

string Name;

string Author;

string Language;

int ReleaseYear;

};

Поскольку, во время выполнения программы потребуется хранить количество элементов, объявляется глобальная переменная, которая хранить количество элементов массива.

int \_itemCount = 0;

Для того, чтобы осуществлять удобный ввод-вывод данных с клавиатуры, были реализованы функции:

* Функция ввода натурального числа (длина массива не может быть отрицательной или равной нулю);
* Функция ввода и инициализации элементов массива;
* Функция вывода всех элементов массива и его порядкового номера;

Функция ввода натурального числа проверяет введённое число, если оно меньше единицы, то система попросит новый ввод.

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

Функция ввода массива структур принимает на ввод натуральное число – количество элементов массива и создаёт новый динамический массив. Далее через цикл for идёт ввод полей каждого элемента массива. После инициализации массива функция возвращает полученный массив.

KeyboardLayout\* InputNewLayouts()

{

cout << "Input the count of new keyboard layouts:" << endl;

\_itemCount = ReadNaturalNum();

KeyboardLayout\* layouts = new KeyboardLayout[\_itemCount];

for (int i = 0; i < \_itemCount; i++)

{

//Заполнение полей элемента

}

cout << endl;

return layouts;

}

Ввод полей элемента:

cout << endl << "Layout's name: ";

cin.ignore();

getline(cin, layouts[i].Name);

cout << "Layout's author: ";

getline(cin, layouts[i].Author);

cout << "Layout's language: ";

getline(cin, layouts[i].Language);

cout << "Layout's release year: ";

layouts[i].ReleaseYear = ReadNaturalNum();

Функция вывода элементов массива устроена очень просто: через цикл for идёт последовательный вывод полей элементов через строку.

void PrintKeyboardLayouts(KeyboardLayout\* layouts)

{

for (int i = 0; i < \_itemCount; i++)

{

cout << "Keyboad layout #" << i + 1 << ":\n";

cout << "Name: \"" << layouts[i].Name << "\";\n";

cout << "Author: \"" << layouts[i].Author << "\";\n";

cout << "Language: \"" << layouts[i].Language << "\";\n";

cout << "Rel. Year: \"" << layouts[i].ReleaseYear

<< "\";\n\n";

}

}

Поскольку в обоих алгоритмах необходимо менять элементы местами, была реализована перегруженная функция Swap, которая меняет два элемента местами.

void Swap(KeyboardLayout& first, KeyboardLayout& second)

{

KeyboardLayout buffer = first;

first = second;

second = buffer;

}

void Swap(int& first, int& second)

{

int buffer = first;

first = second;

second = buffer;

}

**Основная функция main:**

В основной функции будут происходить следующие действия:

1. Ввод элементов нового массива с клавиатуры;
2. Вывод полученного массива на экран;
3. Сортировка массива с использованием одного из методов сортировки;
4. Вывод сортированного массива на экран;

int main()

{

KeyboardLayout\* layouts = InputNewLayouts();

PrintKeyboardLayouts(layouts);

<функция сортировки>(layouts);

cout << "Sorted array:" << endl;

PrintKeyboardLayouts(layouts);

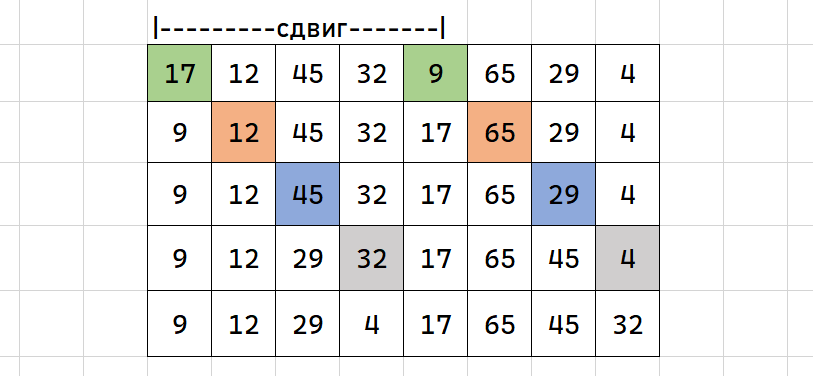
return 0;

}

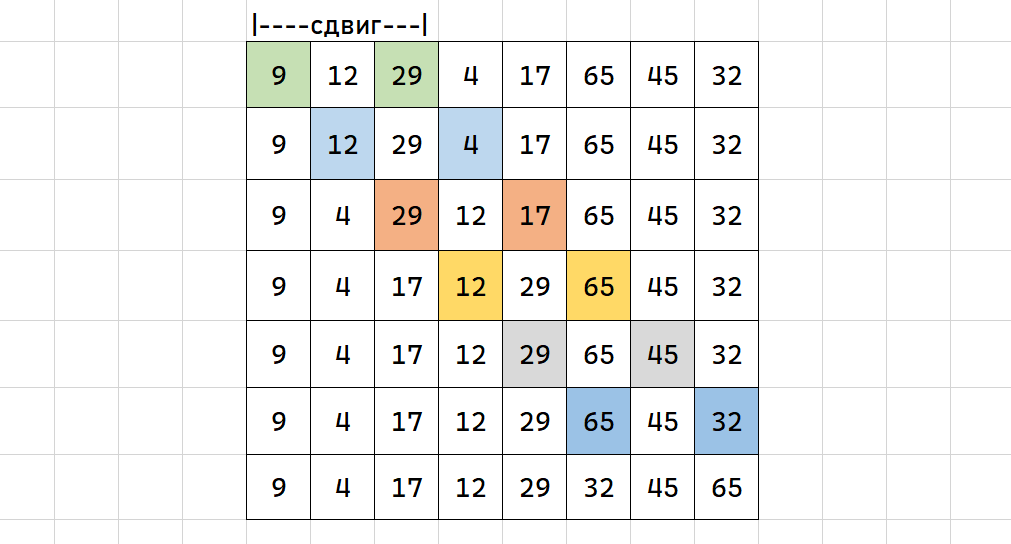
**Метод Шелла – Shell’s sort:**

Алгоритм Шелла является не самым быстрым алгоритмом сортировки, однако в сравнении с другими «быстрыми» алгоритмами сортировки, риск того, что скорость алгоритма при определённых условиях сдеградирует до скорости сортировки «Пузырьком» намного ниже. Данный метод считается самым универсальным, но ни в чём не выдающимся.

Метод работает по принципу сдвига, который в начале равен половине длины массива. На каждой итерации идёт сравнение элемента с текущем индексом и элемента с индексом со сдвигом, если левый элемент больше правого, то их необходимо поменять местами. Так будет происходить пока элемент со сдвигом не станет последним в массиве.



После того, как элемент со сдвигом стал последним, идёт возврат на начало массива, но длина сдвига уменьшается вдвое.





В итоге получается отсортированный по возрастанию массив.

Реализация данного алгоритма на языке C++ является очень простой: в цикле while уменьшается сдвиг в два раза (пока тот не станет равным нулю). Во вложенном цикле for идёт проход по элементам и сравнивается левый и правый элемент. Если вышеописанное условие выполняется, происходит замена элементов с использованием вышеописанной функции Swap.

void ShellSort(KeyboardLayout\* layouts)

{

int offset = \_itemCount;

while (offset > 0)

{

offset /= 2;

for (int i = 0; i < \_itemCount - offset - 1; i++)

if (layouts[i].ReleaseYear > layouts[i + offset +

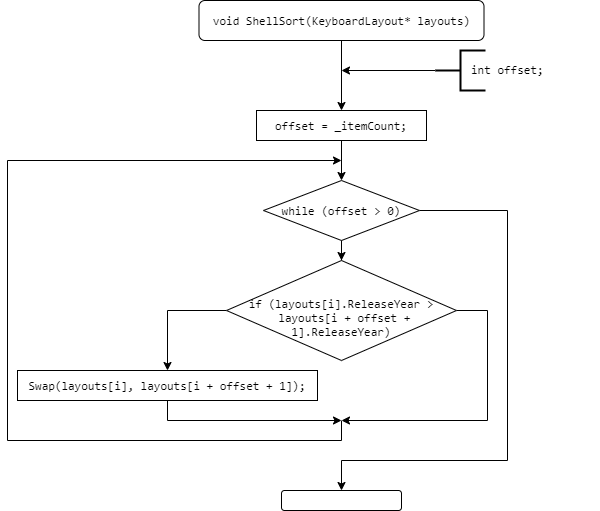
1].ReleaseYear)

Swap(layouts[i], layouts[i + offset + 1]);

}

}

**Блок-схема** алгоритма Шелла:



**Метод Хоара – Quick sort:**

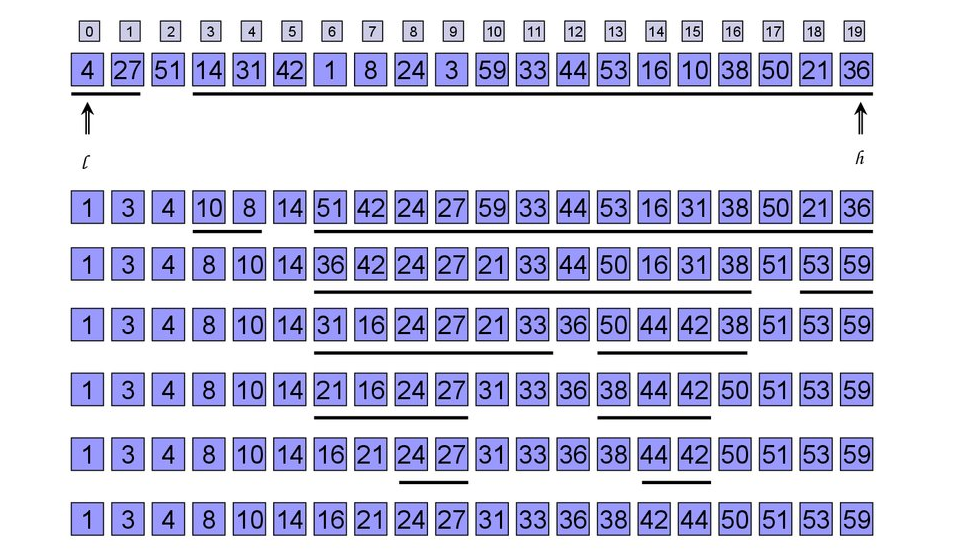
Метод Хоара является дальнейшим развитием пузырькового метода сортировки массивов. Также этот алгоритм считается самым быстрым в сравнении со другими алгоритмами сортировки, однако скорость работа алгоритма сильно зависит от следующих факторов:

1. Наличие частичной упорядоченности массива;
2. Выбор опорной точки;

При наличии первого условия или неудачного выбора опорной точки, скорость алгоритма сильно деградирует до скорости самых медленных алгоритмов сортировки (например, сортировки «Пузырьком».

Алгоритм работает по следующим правилам:

1. Выбирается опорная точка – pivot;
2. Устанавливается два указателя – левый и правый;
3. Если левый указатель указывает на элемент меньший, чем pivot, то указатель сдвигается вправо.
4. В противном случае идёт переключение на правый указатель.
5. Если правый указатель указывает на элемент больший, чем pivot, то указатель сдвигается влево.
6. Если и это условие не выполняется, элементы на которые указывают левый и правый указатели меняются местами.
7. После того, как pivot был достигнут, массив делится на две части и идёт сортировка в них, рекурсивная функция.



Поскольку функция является рекурсивной, она принимает в качестве аргументов не только сам массив, но и границы – левый крайний и правый крайний индекс.

void HoareSort(KeyboardLayout\* layouts, int leftBorderIndex, int rightBorderIndex)

В начале выполнения функции объявляются левый и правый указатели (индексы) равные левому и правому краю, соответственно. Также объявлена опорная точка – pivot, равная среднему элементу массива.

int

leftPoint = leftBorderIndex,

rightPoint = rightBorderIndex,

pivot = layouts[(leftPoint + rightPoint) / 2].ReleaseYear;

Затем идёт проверка на значения индекса левого и правого края, на случай если пользователь ошибочно ввёл данные не в том порядке.

if (leftBorderIndex > rightBorderIndex)

Swap(leftBorderIndex, rightBorderIndex);

Далее запускается цикл, который проверяет сошлись ли левый и правый указатели. В цикле первые двигается левый указатель. Левый указатель будет двигаться до тех пор, пока значение элемента, на который он указывает, меньше, чем значение опорной точки.

После того, как условие перестало выполняться, начинает двигаться правый указатель. Правый указатель двигается до тех пор, пока значение элемента, на который он указывает, больше, чем значение опорной точки.

do

{

while ((layouts[leftPoint].ReleaseYear < pivot) &&

(leftPoint < rightBorderIndex))

leftPoint += 1;

while ((layouts[rightPoint].ReleaseYear > pivot) &&

(rightPoint > leftBorderIndex))

rightPoint -= 1;

//дальнейшее действия происходят здесь

}

while (leftPoint <= rightPoint);

После того, как и это условие перестало выполняться, необходимо поменять местами элементы, на которые указывают указатели и сдвинуть указатель на единицу. Но перед этим необходимо убедиться, что левый указатель находится левее правого.

if (leftPoint <= rightPoint)

{

Swap(layouts[leftPoint], layouts[rightPoint]);

leftPoint += 1;

rightPoint -= 1;

}

После того, как осуществился выход из цикла, необходимо произвести сортировку двух подмножеств массива, так как была достигнута промежуточная точка.

Сортировка правого подмножества:

if (leftBorderIndex < rightPoint)

HoareSort(layouts, leftBorderIndex, rightPoint);

Сортировка левого подмножества:

if (leftPoint < rightBorderIndex)

HoareSort(layouts, leftPoint, rightBorderIndex);

**Полный исходный код алгоритма Хоара:**

void HoareSort(KeyboardLayout\* layouts, int leftBorderIndex, int rightBorderIndex)

{

int

leftPoint = leftBorderIndex,

rightPoint = rightBorderIndex,

pivot = layouts[(leftPoint + rightPoint) / 2].ReleaseYear;

if (leftBorderIndex > rightBorderIndex)

Swap(leftBorderIndex, rightBorderIndex);

do

{

while ((layouts[leftPoint].ReleaseYear < pivot) && (leftPoint < rightBorderIndex))

leftPoint += 1;

while ((layouts[rightPoint].ReleaseYear > pivot) && (rightPoint > leftBorderIndex))

rightPoint -= 1;

if (leftPoint <= rightPoint)

{

Swap(layouts[leftPoint], layouts[rightPoint]);

leftPoint += 1;

rightPoint -= 1;

}

}

while (leftPoint <= rightPoint);

if (leftBorderIndex < rightPoint)

HoareSort(layouts, leftBorderIndex, rightPoint);

if (leftPoint < rightBorderIndex)

HoareSort(layouts, leftPoint, rightBorderIndex);

}

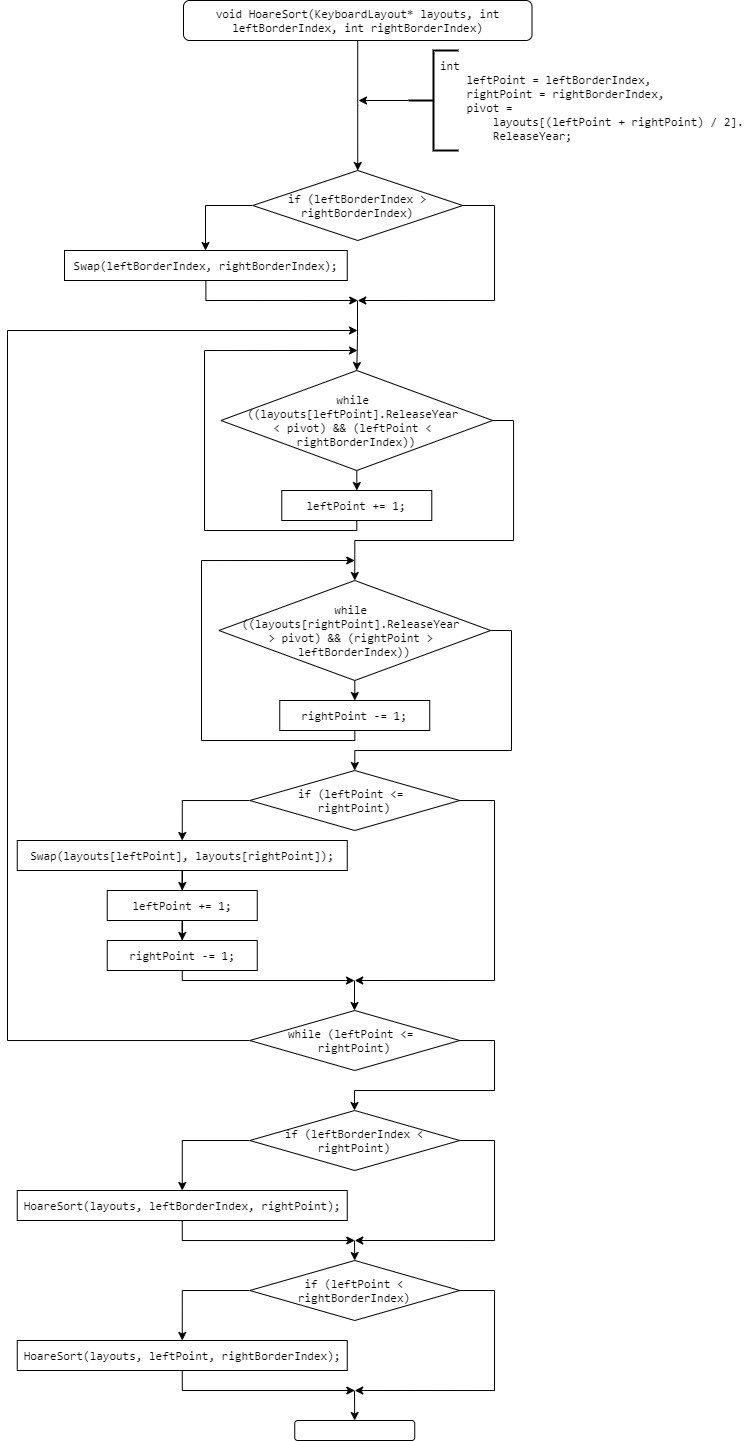
void HoareSort(KeyboardLayout\* layouts)

{

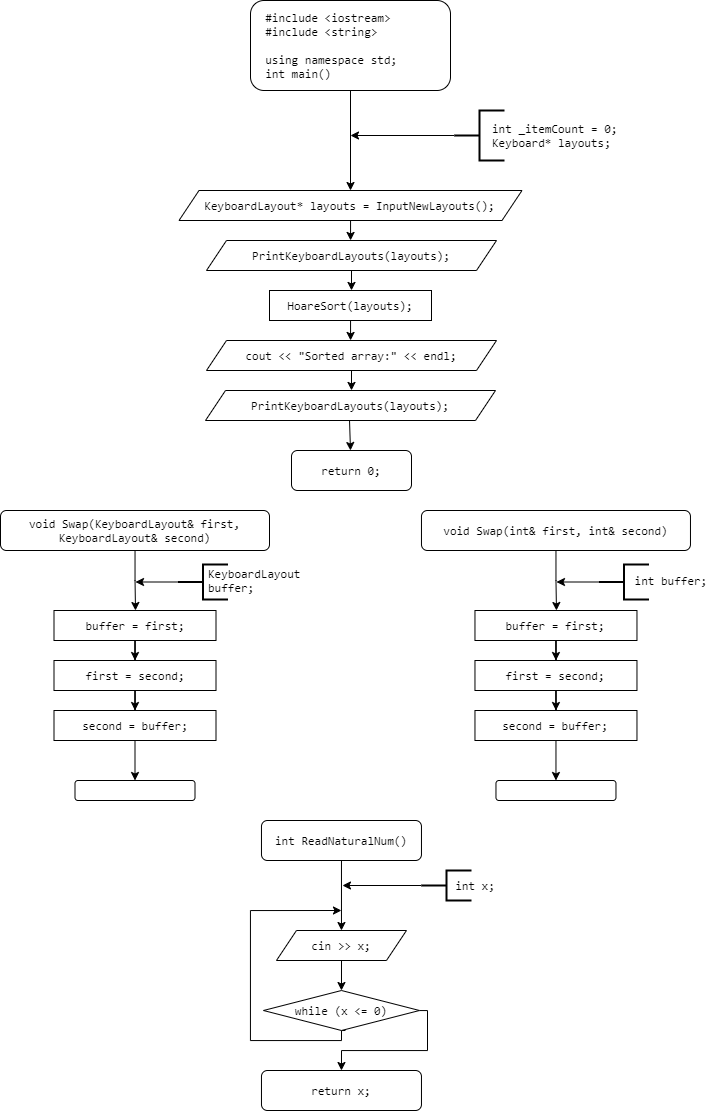
HoareSort(layouts, 0, \_itemCount - 1);

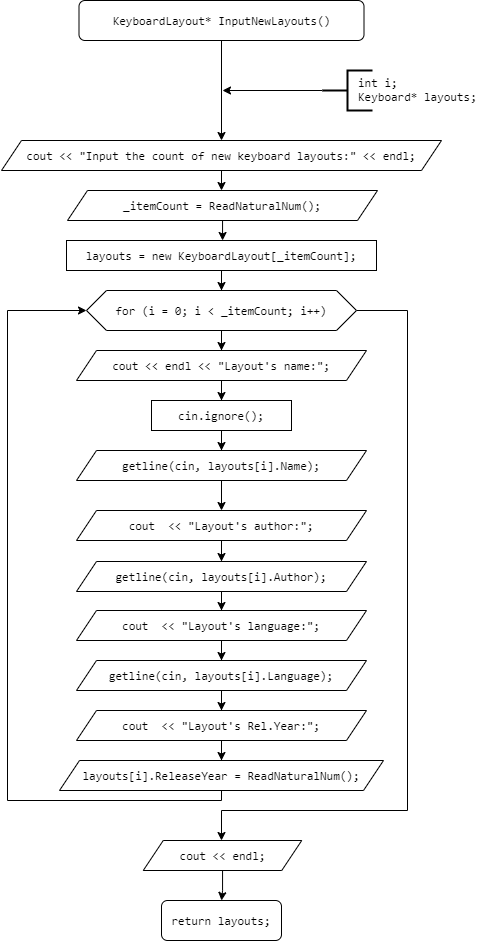
}

**Блок-схема** алгоритма Хоара:



**Блок-схема** основных функций программы:

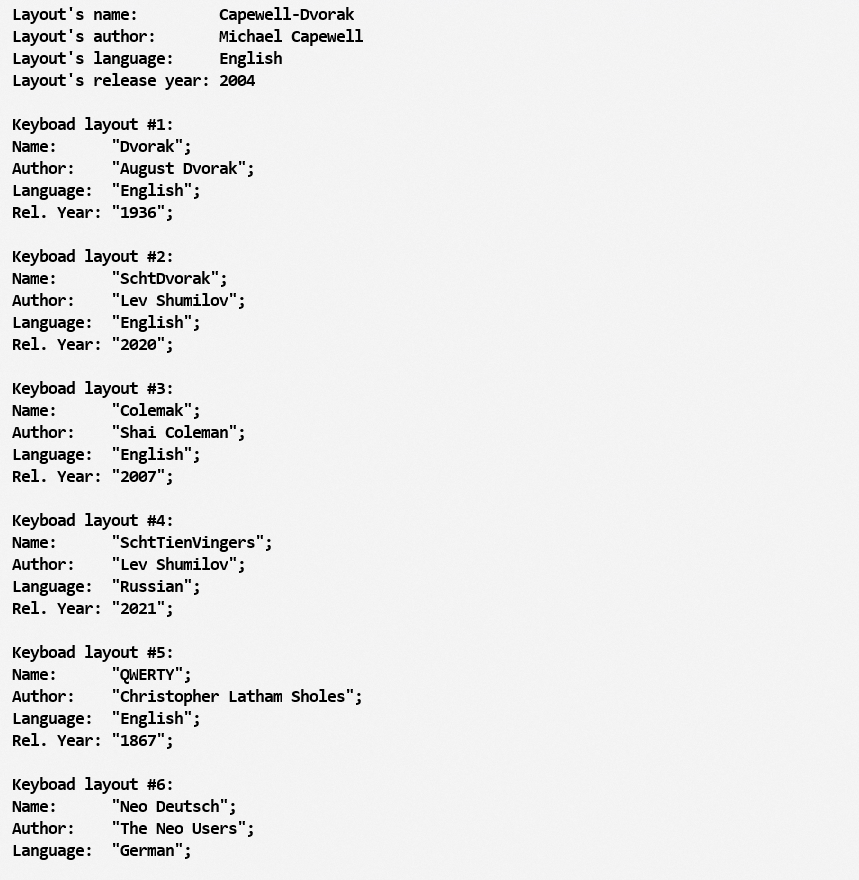


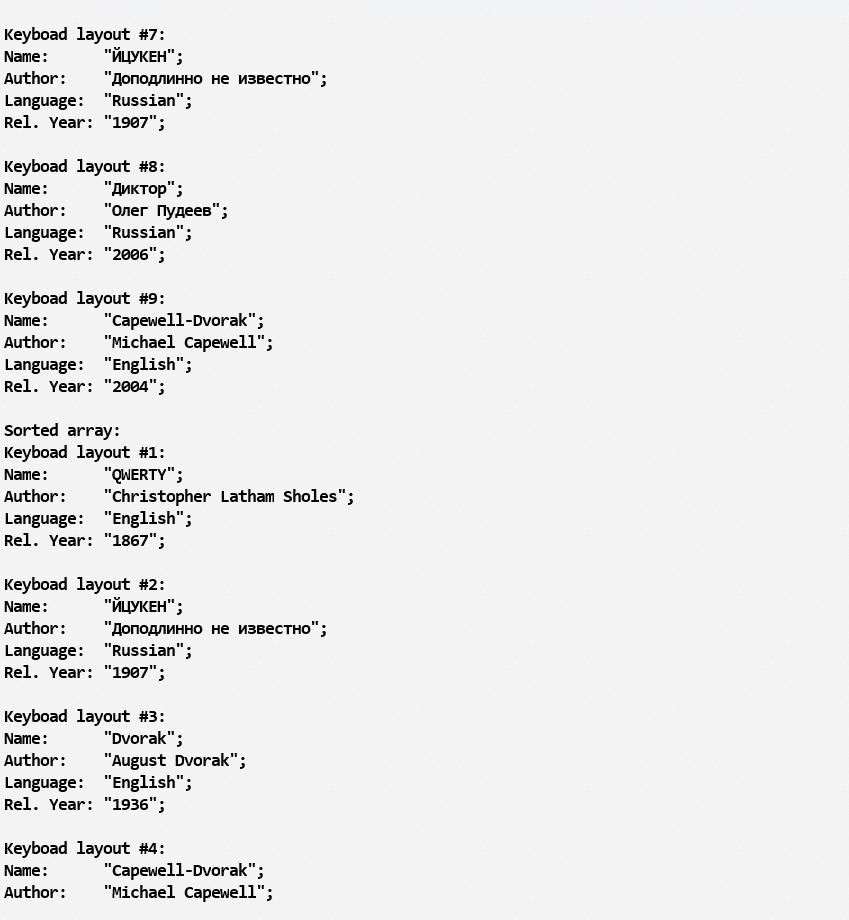


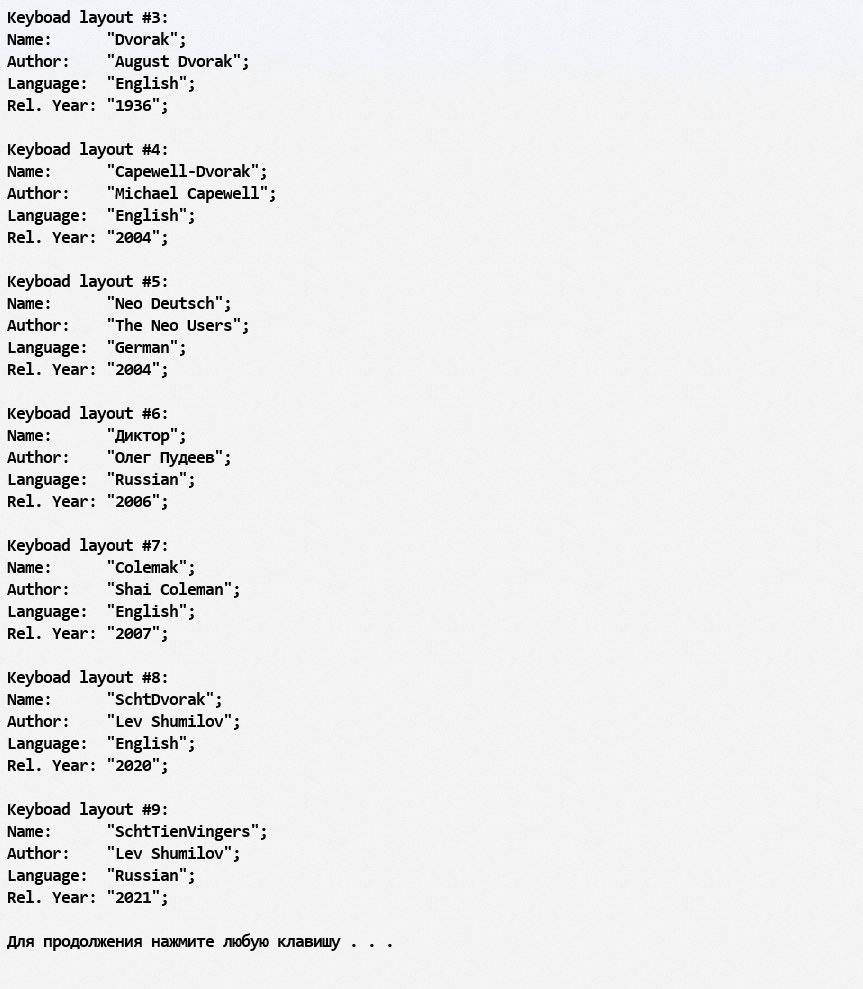
**Скриншоты** работы алгоритмов сортировки:

Алгоритм Шелла:









Алгоритм Хоара:

