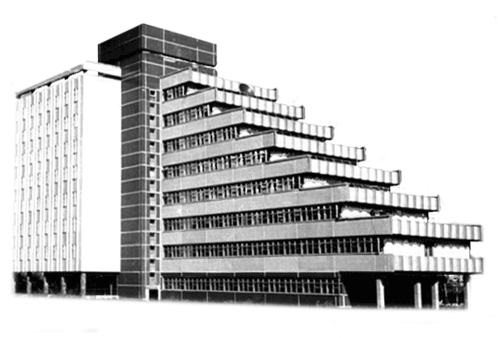
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное агентство по образованию

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»



Лабораторная работа № 5

по дисциплине «Информатика и программирование (I часть)»

Направление подготовки: 230105 - «Программное обеспечение вычислительной техники

и автоматизированных систем»

Выполнил слушатель: Коренский А.А.

Группа: 1

Вариант: 2

Дата сдачи: .03.2015

Преподаватель: Прытков Д.В.

Новосибирск, 2015г.

# Задание

Сортировка Шелла. Частичную сортировку с заданным шагом, начиная с заданного элемента оформить в виде функции. Алгоритм частичной сортировки - вставка погружением.

# Теоретический материал

Сортировка – упорядочение элементов структуры данных в порядке возрастания - уже давно стала классикой программирования. Здесь мы не собираемся “изобретать велосипед” или обсуждать глубинные достоинства или недостатки различных алгоритмов сортировки. Наша задача – увидеть все многообразие и классифицировать его. В этом обзоре, безусловно, будут пробелы, тем более, что некоторые алгоритмы сортировки используют сложные структуры данных, а мы ограничиваемся пока только массивами. Тексты программ большинства сортировок приведены в “Вопросах без ответов”, поэтому нужно потрудиться, чтобы установить соответствие между словесным описанием алгоритма и текстом программы.

Но для начала зададимся жизненно важным вопросом: а зачем нужна сортировка? Ответ простой: если данные не упорядочены, то найти что-либо нас интересующее можно только путем последовательного перебора всех элементов. Для обычного массива фрагмент программы, определяющий, имеет ли один из его элементов заданное значение, выглядит так:

for (i=0; i<n; i++)

if (A[i]==B) break;

if (i != n) ...найден...

То, что мы получаем в данном фрагменте только факт наличия элемента массива с данным значением, не играет никакой роли для понимания сущности поиска данных. В реальных программах “элементами массива” являются, конечно, не простые переменные, а более сложные образования (например, структурированные переменные). Та часть элемента данных, которая идентифицирует его и используется для поиска, называется *ключом*. Остальная часть несет в себе содержательную информацию, которая извлекается и используется из найденного элемента данных.

Ключ – часть элемента данных, которая используется для его идентификации и поиска среди множества других таких элементов.

Приведенный фрагмент программы обеспечивает в неупорядоченном массиве *последовательный* или *линейный поиск* и среднее количество просмотренных элементов для массива размерности n будет равно n/2. Если же элементы данных упорядочены, то найти интересующий нас элемент можно значительно быстрее. Алгоритм, основанный на делении пополам текущего интервала поиска, называется алгоритмом бинарного или *двоичного поиска*. В основе его лежит тот факт, что при однократном сравнении искомого элемента и некоторого элемента массива мы можем определить, справа или слева от текущего следует искать. Чтобы количество сравнений было минимальным, нужно выбирать элемент на середине интервала, в котором производится поиск. Приведем словесное описание алгоритма:

-искомый интервал поиска делится пополам и по значению элемента массива в точке деления определяется, в какой части следует искать значение на следующем шаге цикла;

-для выбранного интервала поиск повторяется;

-при “сжатии” интервала в 0 поиск прекращается;

-в качестве начального интервала выбирается весь массив.

Алгоритмы сортировки можно классифицировать по нескольким признакам.

Вид сортировки по размещению элементов: внутренняя -в памяти, внешняя -в файле данных.

Вид сортировки по виду структуры данных, содержащей сортируемые элементы: сортировка массивов, массивов указателей, списков и других структур данных.

Способ выбора элементов:

-*сортировка подсчетом*: определяется количество элементов, больших или меньших данного;

-*сортировка вставками*: очередной элемент помещается по месту своего расположения в выходную последовательность (массив);

-*сортировка выбором*: выбирается очередной минимальный элемент и помещается в конец последовательности;

-*обменная сортировка*: выполняется путем перестановки элементов по определенному правилу;

-*сортировка слиянием*: последовательность (массив) регулярно распределяется в несколько последовательностей, которые затем объединяются (слияние).

# Описание алгоритма

Цель данной программы – для массивов размерами 100, 1000, 10000 выполнить сортировку Шелла (частичная сортировка с заданным шагом, начиная с заданного элемента) и сортировку *погружение вставками*; затем вывести на экран количество перестановок по каждому массиву для визуального сравнения эффективности методов. Алгоритм реализации задания представлен ниже.

Достижение заданной цели реализовано с помощью двух функций

*ShellSort()* и *PogrujSort()*, каждая из которых отвечает за один из методов сортировки. Функция *main()* вычисляет массив необходимой длинны, заполняя его случайными числами, а затем передает его на вход в вышеуказанные функции для дальнейшей сортировки.

Функцию *ShellSort()* можно условно разбить на четыре цикла *for*. Первый – обеспечивает выполнение сортировки для каждого массива. Второй – вычисляет интервал сортировки, деля его на два, пока он не станет равен единице. Третий – обеспечивает перебор всех элементов массива для всех интервалов. Четвертый – реализует сам процесс сортировки: с помощью проверки на выход за пределы массива и сравнения двух элементов (отстоящих друг от друга на вычисленный интервал) на меньшее значение, (в случае истинности обоих проверок) происходит «погружение» i-го элемента до своей позиции в уже упорядоченную часть массива.

Функция *PogrujSort()* реализует метод сортировки *вставка погружением*. Ее также можно условно разбить на три цикла *for*. Первый – обеспечивает выполнение сортировки для каждого массива. Второй – обеспечивает перебор всех элементов массива. Третий – поочередно сортирует соседние элементы массива, «погружая» их на соответствующую позицию в уже упорядоченную часть массива.

# 4. Описание реализации

Подключаемые библиотеки:

#include "stdio.h"

#include "conio.h"

#include "stdlib.h"

#include "math.h"

#include “windows.h”

Используемые функции:

int main() – вызывает функцию *ShellSort ()*;

printf() – вывод форматированного текста на экран;

gotoxy() – для перехода к заданным координатам экрана (библиотека windows.h);

getch() – ожидание ввода символа с клавитуры.

Используемые конструкции:

if – условный оператор, для создания ветвления ;

for – оператор цикла;

switch – оператор-переключатель;

#define – препроцессорная директива (подстановка значения по всему коду программы).

# 5. Пример работы программы

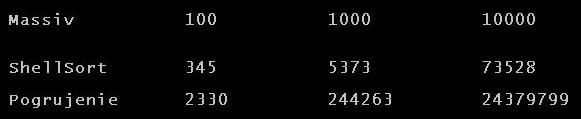


Рисунок 1. Результат работы функций *ShellSort ()* и *PogrujSort()*

На рисунке 1 показан результат работы функций *ShellSort ()* и *PogrujSort()*. На экран выводится количество перестановок для массивов из 100, 1000, 10000 элементов. После чего визуально можно убедиться в эффективности *сортировки Шелла* (частичная сортировка – *вставка погружением*) по отношению к сортировке *вставка погружением*.

**Выводы**

Перед выполнением этой работы, был изучен теоретический материал по данным темам. Затем необходимо было продумать алгоритм будущей программы, для общего представления ее конструкции. После чего, я приступил к написанию исходного кода.

В ходе выполнения лабораторной работы были изучены принципы сортировки массивов, освоены методы подсчета затрат на различные виды сортировок.

После было проведено контрольное тестирование программы, которое показало правильность ее работы.

**Приложение. Текст программы с комментариями**

#include "stdio.h"

#include "conio.h"

#include "math.h"

#include "stdlib.h"

#include <windows.h>

#define N 10000

//Лабораторная работа №5 вариант 2

//Задание: сортировка Шелла. Частичную сортировку с заданным шагом, начиная

//с заданного элемента оформить в виде функции. Алгоритм частичной

//сортировки – вставка погружением. Если имеется базовый алгоритм сортировки

//то аналогично оформить базовый алгоритм и произвести сравнение

//эффективности.

void gotoxy(int x, int y){ //перемещение курсора в координаты 'x' и 'y'

COORD scrn;

HANDLE hOuput = GetStdHandle(STD\_OUTPUT\_HANDLE);

scrn.X = x; scrn.Y = y;

SetConsoleCursorPosition(hOuput,scrn);

}

int main(){

int ShellSort(int A[]), PogrujSort(int B[]); //объявление функций

int A[N], B[N], i;

for(i=0;i<N;i++){ //формирует элементы массива случайным образом

A[i]=rand()%43;

B[i]=A[i];

}

ShellSort(A); //запуск сортировки Шелла(погружение вставками)

PogrujSort(B); //запуск сортировки погружение вставками

getch();

}

int ShellSort(int A[]){

int i, d, k, j, t, z=0, zs=0, H=10, v;

gotoxy(1,1);

printf("Massiv 100 1000 10000\n\n");

for(v=0;v<3;v++){ //цикл для 100, 1000, 10000 повторений

H\*=10;

for(d=H/2;d>=1;d/=2){ //вычисляет d (интервалл сортировки массива

//в методе Шелла) при каждой итерации

for(i=0;i<H;i++){ //сортирует весь массив с вычесленным d

for(t=H/d,j=t;j>0;j--){ //вычисляет количество интервалов для i-го элемента

if (((i+j\*d)<=(H-1)) && (A[i+(j-1)\*d]<A[i+j\*d])){ //проверка на выход за пределы массива и на меньшее значение

k=A[i+(j-1)\*d]; //A[i+(j-1)\*d] - элемент с меньшим индексом

A[i+(j-1)\*d]=A[i+j\*d]; //j\*d - задает интервал и его количество

A[i+j\*d]=k; //A[i+j\*d] - элемент с большим индексом

z++; //счетчик перестановок

}

}

}

zs+=z; //суммируем количество перестановок (для статистики)

z=0; //обнуляем счетчик

}

switch (v){

case 0: gotoxy(1,4); printf("ShellSort %d", zs); break;

case 1: gotoxy(30,4); printf("%d", zs); break;

case 2: gotoxy(44,4); printf("%d\n", zs); break;

}

}

}

int PogrujSort(int B[]){

int i, d, k, j, t, z=0, zs=0, H=10, v;

for(v=0;v<3;v++){ //цикл для 100, 1000, 10000 повторений

H\*=10;

for(i=1;i<=H-1;i++){ //проход по всем элементам массива

for(j=i;j>0;j--){ //сортирует часть массива до i-го элемента

if (B[j]>B[j-1]){ //меняет местами в случае

//большего значения

k=B[j-1];

B[j-1]=B[j];

B[j]=k;

z++; //счетчик перестановок

}

}

zs+=z; //суммируем количество перестановок (для статистики)

z=0; //обнуляем счетчик

}

switch (v){

case 0: gotoxy(1,6); printf("Pogrujenie %d", zs); break;

case 1: gotoxy(30,6); printf("%d", zs); break;

case 2: gotoxy(44,6); printf("%d", zs); break;

}

}

}

Защита:

1)

2)