**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**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

автоматизированныхсистем

Лабораторная работа №3

дисциплина: Алгоритмы и структуры данных

тема: «Сравнительный анализ методов сортировки»

Выполнил ст. группы ВТ-

Проверил: проф. Синюк В.Г.

Белгород 20

**Цель работы:** изучение методов сортировки массивов иприобретение навыков в проведении сравнительного анализа различных методов сортировки.

**Задание**

# 1. Изучить временные характеристики алгоритмов.

2. Изучить методы сортировки:

1) включением;

2) выбором;

3) обменом:

3.1) улучшенная обменом 1;

3.2) улучшенная обменом 2;

4) Шелла;

5) Хоара;

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

Заголовочный файл

#ifndef **ASD\_3\_SORT\_SORT\_H**#define **ASD\_3\_SORT\_SORT\_H  
void** Sis(**int** \*A,**int** nn); *//прямое включение***void** StrSel(**int** \*A,**int** nn);*//прямое выбором***void** BblSort(**int** \*A,**int** nn);*// пузырьком***void** ShellSort(**int** \*a, **int** n); *//методом Шела=***void** HoarSort(**int** \*a, **int** n); *//Хоара***void** HeapSort(**int \***A,**int** nn); *// пирамидальная  
void change\_Sort\_1 (int \*a, int n) //Улучшенная обменом 1*

*void change\_Sort\_2 (int \*a, int n) //Улучшенная обменом 2*#endif *//ASD\_3\_SORT\_SORT\_H*

Модуль

#include **"sort.h"**#include **<math.h>***// функция сортировки включением***void** Sis(**int** \*A,**int** nn){  
 **int** i,j,k;  
 **for** ( j=1; j<nn; j++ ){  
 k = A[j];  
 i = j -1;  
 **while** ( k < A[i] && i >= 0){  
 A[i+1] = A[i];  
 i -= 1;  
 }  
 A[i+1] = k;  
 }  
}  
  
*//функциясортировкивыбором***void** StrSel(**int** \*A,**int** nn){  
 **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];  
 }  
 A[k] = A[i];  
 A[i] = x;  
 }  
}  
  
*// функциясортировкиобменом***void** BblSort(**int** \*A,**int** nn){  
 **int** i,j,k,p;  
 **for** ( i=0; i<nn-1; i++ ){  
 p = 0;  
 **for** (j=nn-1; j>i; j--)  
 **if** (A[j] <A[j-1]){  
 k = A[j];  
 A[j] = A[j-1];  
 A[j-1] = k;  
 p = 1;  
 }*/\* Если перестановок не было, то сортировка выполнена\*/* **if**( p== 0)  
 **break**;  
 }  
}  
  
*/\* функция сортировки методом Шелла \*/***void** ShellSort(**int** \*a, **int** n){  
 **int** i,j,k,hh,t,s;  
 **int** h [1000];  
 t = round(log(n)/log(3))-1;

**if** (t < 1){  
 t = 1;  
 }  
 h[t-1] = 1;  
 **for** (k=t-1; k>= 1; k--) {  
 h[k-1] = 3\*h[k]+1;  
 }  
 **for** (s=t-1;s>=0;s--){  
 hh = h[s];  
 **for** (j = hh;j<=n;j++){  
 i = j-hh;  
 k = a[j];  
 **while** ((k <= a[i])&&(i >= 0))  
 {a[i+hh] = a[i];  
 i = i-hh;};  
 a[i+hh] = k;  
 }  
 }  
  
}  
  
**void** QSort(**int** \*a, **int** L, **int** R){  
 **int** x=a[L], i= L, j= R, t; *//в качестве разделителя выбираем первый элемент* **while** (i<=j){  
 **while** (a[i]<x)  
 i++;  
 **while** (a[j]>x)  
 j--;  
 **if** (i<=j){  
 t = a[i];  
 a[i] = a[j];  
 a[j] = t;  
 i++;  
 j--;  
 }  
 }  
 **if** (L<j)  
 QSort(a,L,j);  
 **if** (i<R)  
 QSort(a,i,R);  
}  
*//функция сортировки методом Хоара***void** HoarSort(**int** \*a, **int** n){  
 QSort(a,1,n);  
}  
  
**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++;  
 }  
}  
  
*/\*пирамидальная функциясортировки\*/***void** HeapSort(**int \***A,**int** nn){  
 **int** L,R,x,i;  
 L = nn/2 ;  
 R = nn-1;  
 */\* Построение пирамиды из исходного массива\*/* **while** ( L>0 ){  
 L = L -1;  
 Sift(A,L,R);  
 }*/\* Сортировка: пирамида => отсортированный массив\*/* **while** ( R>0 ){  
 x = A[0];  
 A[0] = A[R];  
 A[R] = x;  
 R--;  
 Sift(A,L,R);  
 }  
}

*void change\_Sort\_1 (int \*a, int n) { //Улучшенная обменом 1   
 int f,k,i,j;  
 i =1;  
 do {  
 f = 0;  
 for (j = n-1; j >= i; j--) {  
 if (a[j - 1] >= a[j]) {  
 k = a[j - 1];   
 a[j - 1] = a[j];  
 a[j] = k;  
 f = 1;  
 }  
 }  
 i++;  
 } while ((f=1) && (i < n));  
}  
  
void change\_Sort\_2 (int \*a, int n) { //Улучшенная обменом 2   
 int i,j,k,t, z;  
 i = 1;  
 k = n;  
 do {  
 z = k + 1;  
 for (j = n - 1; j >= i; j--) {*

*p++;  
 if (a[j - 1] >= a[j]) {  
 t = a[j - 1];   
 a[j - 1] = a[j];  
 a[j] = t;  
 k = j - 1;  
 }  
 }  
 i = k + 1;  
 } while (z != i); //Если перестановок не было, выйти из алгоритма  
}*

Основная программа

#include **<stdio.h>**#include **"sort.h"**#include **<time.h>**#include **<stdlib.h>  
const long int** N = 45\*1000;  
  
**void** output(**int** \*a){  
 **for** (**int** i = 0; i < N; ++i) {  
 printf(**"%d "**,a[i]);  
 }  
 printf(**"\n"**);  
}  
**void** input1(**int** \*a){  
 **for** (**int** i = 0; i < N; ++i) {  
 a[i]=N-i;  
 }  
}  
**void** input2( **int** \*a){  
 **for** (**int** i = 0; i < N; ++i) {  
 a[i]=i;  
 }  
}  
**void** input3(**int** \*a){  
 **for** (**int** i = 0; i < N; ++i) {  
 a[i]=rand()%100;  
 }  
}  
**int** main() {  
 **int** a[N];  
  
 input1(a);  
 clock\_t t = clock();  
 HeapSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 HeapSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 HeapSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 input1(a);  
 t = clock();  
 Sis(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 Sis(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );;  
  
 input3(a);  
 t = clock();  
 Sis(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 input1(a);  
 t = clock();  
 StrSel(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 StrSel(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 StrSel(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 input1(a);  
 t = clock();  
 BblSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 BblSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 BblSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 input1(a);  
 t = clock();  
 ShellSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 ShellSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 ShellSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 input1(a);  
 t = clock();  
 HoarSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 HoarSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 HoarSort(a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );

input1(a);  
 t = clock();  
 change\_Sort\_1 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 change\_Sort\_1 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 change\_Sort\_1 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );

input1(a);  
 t = clock();  
 change\_Sort\_2 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input2(a);  
 t = clock();  
 change\_Sort\_2 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
 input3(a);  
 t = clock();  
 change\_Sort\_2 (a,N);  
 t=clock()-t;  
 printf(**"%f\n"**, (**double**) t );  
  
  
 **return** 0;  
}

Задание 5

**Результаты экспериментов (Обратно упорядоченный массив)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве \*1000 | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Включением | 82 | 324 | 721 | 1202 | 176 | 2381 | 3236 | 8021 | 17022 |
| Выбором | 62 | 221 | 535 | 1120 | 2415 | 3416 | 4952 | 6580 | 9871 |
| Обменом | 124 | 411 | 1223 | 2540 | 3751 | 6218 | 9354 | 12410 | 16450 |
| Обменом 1 | 94 | 408 | 985 | 1624 | 3642 | 6233 | 9325 | 12401 | 16423 |
| Обменом 2 | 167 | 420 | 1084 | 1725 | 3748 | 6314 | 9451 | 13441 | 17235 |
| Шелла | 1 | 1 | 1 | 3 | 4 | 6 | 8 | 9 | 11 |
| Хоара | 36 | 107 | 211 | 488 | 628 | 789 | 950 | 1110 | 1414 |
| Пирамидальная | 12 | 15 | 19 | 26 | 34 | 47 | 59 | 71 | 82 |

**Результаты экспериментов (Упорядоченный массив)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве \*1000 | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Включением | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Выбором | 47 | 192 | 405 | 689 | 941 | 1635 | 2895 | 4210 | 6490 |
| Обменом | 49 | 208 | 641 | 1398 | 1842 | 2574 | 4412 | 6271 | 8440 |
| Обменом 1 | 54 | 310 | 781 | 1481 | 1842 | 2325 | 3451 | 5412 | 6841 |
| Обменом 2 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| Шелла | 3 | 8 | 13 | 18 | 23 | 28 | 33 | 38 | 43 |
| Хоара | 21 | 142 | 687 | 1041 | 1441 | 2301 | 2840 | 3512 | 4721 |
| Пирамидальная | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 10 | 12 |

**Результаты экспериментов (Неупорядоченный массив)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве \*1000 | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Включением | 41 | 174 | 345 | 874 | 1214 | 1641 | 2480 | 3024 | 3874 |
| Выбором | 54 | 164 | 475 | 1121 | 1457 | 1678 | 2784 | 3841 | 6475 |
| Обменом | 154 | 847 | 1654 | 3102 | 4345 | 5784 | 7845 | 9744 | 12450 |
| Обменом 1 | 142 | 804 | 105 | 3214 | 4357 | 6147 | 9595 | 11407 | 13420 |
| Обменом 2 | 3 | 4 | 5 | 8 | 10 | 11 | 13 | 14 | 21 |
| Шелла | 25 | 2 | 3 | 8 | 7 | 9 | 9 | 10 | 12 |
| Хоара | 1 | 2 | 3 | 7 | 8 | 9 | 10 | 15 | 17 |
| Пирамидальная | 2 | 3 | 3 | 8 | 9 | 9 | 10 | 19 | 23 |

**Не упорядоченный массив (кол-во элементов \*1000)**

**Упорядоченный массив (кол-во элементов \*1000)**

**Массив упорядоченный в обратном порядке (кол-во элементов \*1000)**

Задание 6

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

Включением – лучший случай:N-1; средний, худший:(N-1)\*N/2

Выбором – (N-1)\*N/2

Обменом – (N-1)\*N/2

Обменом 1 – лучший случай:N-1; средний, худший:(N-1)\*N/2

Обменом 2 - лучший случай:N-1; средний, худший:(N-1)\*N/2

Шелла– зависит от выбранных шагов

Хоара–удачный разделитель – N+2\*(N/2)+…+m\*(N\*m), m = logN

Пирамидальная - [log2N]+[log2(N – 1)]+…+[log22]

Задание 7

Функции временной сложности

Включением – лучший случай:O(N); средний, худший: O(N2)

Выбором – O(N2)

Обменом – O(N2)

Обменом 1 – лучший случай:O(N); средний, худший: O(N2)

Обменом 2 - лучший случай:O(N); средний, худший: O(N2)

Шелла - О(N(log2N)).

Хоара – удачный разделитель – O(N·log2N), в худшем - O(N2)

Пирамидальная - O(N·log2N).

**Очередь**

Заголовочный файл

#ifndef **ASD\_3\_SORT\_FIFO\_H**#define **ASD\_3\_SORT\_FIFO\_H***//Размер очереди*#define **FIFO\_SIZE** 10  
  
*//Очередь в порядке***static const short** FIFO\_OK = 0;  
  
*//Очередь переполнена***static const short** FIFO\_FULL = 1;  
  
*//Очередь пуста***static const short** FIFO\_EMPTY = 2;  
  
*//Переменная ошибки***extern short** FIFO\_ERROR;  
  
**typedef int** t\_baseType;  
  
**typedef struct**{  
  
 t\_baseType Buf[**FIFO\_SIZE**];  
  
 **short** Uk1;  
 **short** Uk2;  
  
 **short** n;  
  
} FIFO;  
  
*//Операция инициализации***void** InitFIFO(FIFO \*q);  
  
**void** PutFIFO(FIFO \*q, t\_baseType E);  
  
**void** GetFIFO(FIFO \*q, t\_baseType \*E);  
  
*//Возвращает 1 если очередь q пуста  
//Иначе 0***short** EmptyFIFO(FIFO \*q);  
  
*//Возвращает 0 если очередь q полная  
//Иначе 1***short** FullFIFO(FIFO \*q);  
  
#endif *//ASD\_3\_SORT\_FIFO\_H*

Модуль

#include **"FIFO.h"***//Переменная для ошибок***short** FIFO\_ERROR;  
  
*//Операция инициализации***void** InitFIFO(FIFO \*q)  
{  
  
 q->Uk1 = 0;  
 q->Uk2 = 0;  
 q->n = 0;  
 FIFO\_ERROR = FIFO\_OK;  
  
}  
  
**void** PutFIFO(FIFO \*q, t\_baseType E)  
{  
  
 **if**(FullFIFO(q)){  
 q->Buf[q->Uk1] = E;  
 q->Uk1 = (q->Uk1 + 1) % **FIFO\_SIZE**;  
 ++q->n;  
 FIFO\_ERROR = FIFO\_OK;  
 }  
  
}  
  
**void** GetFIFO(FIFO \*q, t\_baseType \*E)  
{  
 **if**(EmptyFIFO(q)){  
 \*E = q->Buf[q->Uk2];  
 q->Uk2 = (q->Uk2 + 1) % **FIFO\_SIZE**;  
 --q->n;  
 FIFO\_ERROR = FIFO\_OK;  
 }  
}  
  
  
**short** EmptyFIFO(FIFO \*q)  
{  
 **if**(q->n==0){  
 FIFO\_ERROR = FIFO\_OK;  
 **return** 0;  
 } **else** {  
 FIFO\_ERROR = FIFO\_EMPTY;  
 **return** 1;  
 }  
  
}  
  
  
**short** FullFIFO(FIFO \*q)  
{  
 **if**(q->n== **FIFO\_SIZE**){  
 FIFO\_ERROR = FIFO\_FULL;  
 **return** 0;  
 } **else** {  
 FIFO\_ERROR = FIFO\_OK;  
 **return** 1;  
 }}

