**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 автоматизированных систем

Лабораторная работа №3

дисциплина: Алгоритмы и Структуры данных

тема: «Сравнительный анализ методов сортировки (Pascal/C)»

Выполнил: ст. группы ПВ-21

Донцов Александр Алексеевич

Проверил: Синюк Василий Григорьевич

Белгород 2018

**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.

**Выполнение**

#include "pch.h"

#include <stdio.h>

#include <time.h>

#define N 45\*1000

void superinput(int a[N]);//ленивый ввод

void input(int a[N]);//ввод массива чисел

void output(int a[N]);//вывод массива чисел

void swap(int \*a, int \*b);//замена a и b

void inclusion(int a[N]);//сортировка включением

void selection(int a[N]);//сортировка выбором

void exchange(int a[N]);//сортировка обменом

void exchange1(int a[N]);//улучшенная сортировка обменом 1

void exchange2(int a[N]);//улучшенная сортировка обменом 2

void shell(int a[N]); //сортировка Шелла

void squick(int a[N]); //сортировка Хоара

void quick(int a[], int L, int R); //быстрая сортировка

void HeapSort(int A[]);// пирамидальная функция сортировки

void sift(int A[], int L, int R);

int main() {

int a[N];

int c = 0;

/\*superinput3(a);

//output(a);

clock\_t t = clock();

selection(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);

superinput3(a);

//output(a);

t = clock();

exchange(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);

superinput3(a);

//output(a);

t = clock();

exchange1(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);

superinput2(a);

//output(a);

t = clock();

exchange2(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);

superinput3(a);

//output(a);

t = clock();

shell(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);

//superinput3(a);

//output(a);

//t = clock();

//squick(a);

//t = clock() - t;

//output(a);

//printf("time = %d\n", t);

superinput3(a);

//output(a);

t = clock();

HeapSort(a);

t = clock() - t;

//output(a);

printf("time = %d\n", t);\*/

superinput(a);

//output(a);

clock\_t t = clock();

HeapSort(a);

output(a);

t = clock() - t;

printf("time = %d", t);

return 0;

}

void superinput1(int a[N]) {

int i;

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

a[i] = i;

}

void superinput2(int a[N]) {

int i;

for (i = 0; i < N; i++) {

a[i] = rand();

}

}

void superinput3(int a[N]) {

int i, j = N;

for (i = 0; i < N; i++) {

a[i] = j;

j--;

}

}

void input(int a[N]) { //ввод массива чисел+

int i;

printf("Input arr: ");

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

scanf\_s("%d", &a[i]);

return;

}

void output(int a[N]) { //вывод массива чисел+

int i;

printf("Output arr: ");

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

printf("%d ", a[i]);

printf("\n");

return;

}

void swap(int \*a, int \*b) {

int c;

c = \*a;

\*a = \*b;

\*b = c;

return;

}

void inclusion(int a[N]) {//сортировка включением+

int i, k;

int time = 0;

for (i = 1; i < N; i++) {

k = i;

while (k != 0) {

time++;

if (a[k] < a[k - 1]) {

swap(&a[k], &a[k - 1]);

}

else {

break;

}

k--;

}

}

printf("time = %d\n", time);

return;

}

void selection(int a[N]){//сортировка выбором+

int i, j;

int time = 0;

for (i = 0; i < N; i++) {

for (j = i + 1; j < N; j++) {

time++;

if (a[i] > a[j])

swap(&a[i], &a[j]);

}

}

printf("time = %d\n", time);

return;

}

void exchange(int a[N]) { //сортировка обменом+

int i, j;

int time = 0;

for (i = 0; i < N; i++) {

for (j = N - 1; j != i; j--) {

time++;

if (a[j - 1] > a[j])

swap(&a[j - 1], &a[j]);

}

}

printf("time = %d\n", time);

return;

}

void exchange1(int a[N]) { //улучшенная сортировка обменом 1

int i = 0, j;

int flag = 1;

int time = 0;

while (i < N && flag) {

flag = 0;

for (j = N - 1; j != i; j--) {

if (a[j - 1] > a[j]) {

swap(&a[j - 1], &a[j]);

flag = 1;

}

}

i++;

}

printf("time = %d\n", time);

return;

}

void exchange2(int a[N]) { //сортировка обменом

int i = 0, j;

int flag = 1;

int time = 0;

while (i < N && flag) {

flag = 0;

for (j = N - 1; j != i; j--) {

time++;

if (a[j - 1] > a[j]) {

swap(&a[j - 1], &a[j]);

flag = j;

}

}

i = j + 1;

}

printf("time = %d\n", time);

return;

}

void shell(int a[N]) { //сортировка Шелла+

int time = 0;

int h[1000]; int count = 0;

h[count] = 1;

while (h[count] < N) {

h[count + 1] = 3 \* h[count] + 1;

count++;

}

int j;

//цикл, отвечающий за изменение шагов

for (int k = count; k >= 0; k--) {

//обычнвя сортировка вставкой

for (int i = 0; i < N; i = i + h[k]) {

j = i;

while ((++time) && (j - h[k] >= 0) && (a[j] < a[j - h[k]])) {

swap(&a[j], &a[j - 1]);

j = j - h[k];

}

}

}

printf("time = %d\n", time);

return;

}

void squick(int a[N]) { //сортировка Хоара

int time = 0;

quick(a, 0, N - 1, &time);

printf("time = %d\n", time);

}

void quick(int a[], int L, int R, int \*time) { //быстрая сортировка

int x = a[L], i = L, j = R, t; // в качестве разделителя выбираем первый элемент

while (i <= j) {

while (a[i] < x) {

\*time++;

i++;

}

while (a[j] > x) {

\*time++;

j--;

}

time++;

if (i <= j) {

swap(&a[i], &a[j]);

i++;

j--;

}

}

if (L < j)

quick(a, L, j, time);

if (i < R)

quick(a, i, R, time);

}

void HeapSort(int A[]) {// пирамидальная функция сортировки

int L, R, x, i;

int time = 0;

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

while (L > 0) { // Построение пирамиды из исходного массива

L = L - 1;

sift(A, L, R, &time);

}

while (R > 0) { // Сортировка: пирамида => отсортированный массив

x = A[0]; A[0] = A[R]; A[R] = x;

R--;

sift(A, L, R, &time);

}

printf("time = %d\n", time);

return;

}

/\* ============================================ \*/

void sift(int A[], int L, int R, int \*time) {

int i, j, x, k;

i = L;

j = 2 \* L + 1;

x = A[L];

time++;

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;

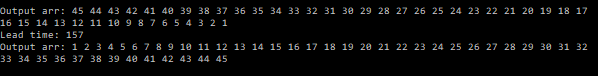
time++;

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

j++;

}

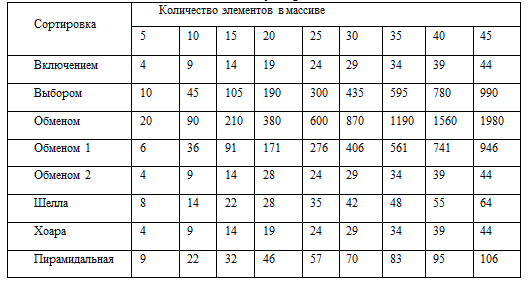
}

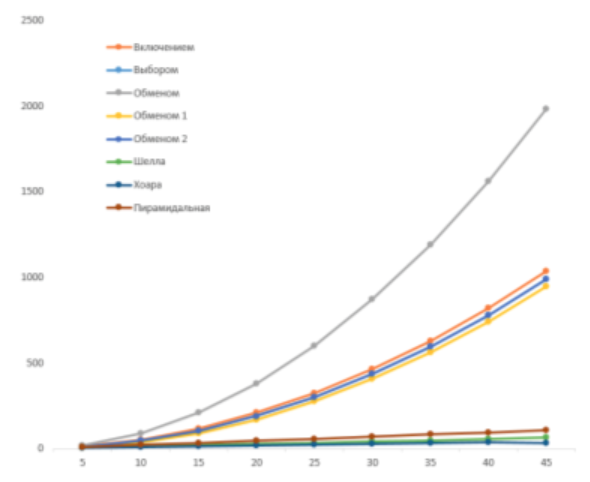


**Результаты экспериментов**

Упорядоченные по возрастанию

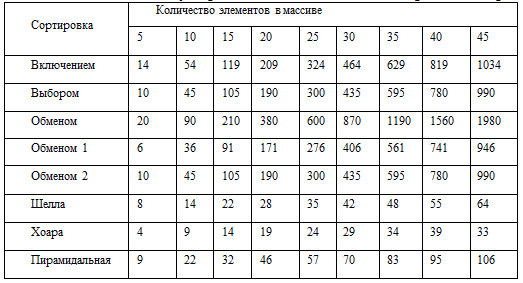
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве \* 1000 | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Включением | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Выбором | 2 | 5 | 12 | 19 | 29 | 40 | 56 | 74 | 96 |
| Обменом+ | 3 | 7 | 14 | 24 | 37 | 54 | 76 | 130 | 135 |
| Обменом 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Обменом 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Шелла | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Хоара | 3 | 7 | 14 | 19 | 32 | 40 | 54 | 74 | 88 |
| Пирамидальная | 0 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |

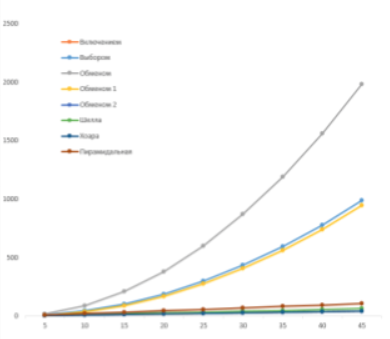




Неупорядоченные

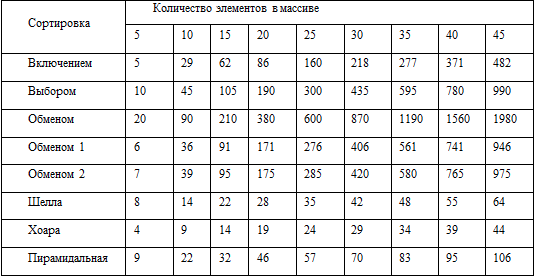
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве \* 1000 | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Включением | 8 | 33 | 80 | 160 | 258 | 364 | 497 | 569 | 771 |
| Выбором | 9 | 36 | 93 | 187 | 295 | 408 | 553 | 713 | 895 |
| Обменом | 9 | 38 | 97 | 211 | 327 | 479 | 526 | 737 | 885 |
| Обменом 1 | 9 | 44 | 99 | 198 | 315 | 449 | 571 | 833 | 955 |
| Обменом 2 | 9 | 38 | 101 | 205 | 343 | 419 | 551 | 753 | 934 |
| Шелла | 4 | 13 | 28 | 55 | 83 | 127 | 199 | 280 | 369 |
| Хоара | 3 | 7 | 14 | 19 | 32 | 40 | 54 | 74 | 95 |
| Пирамидальная | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |

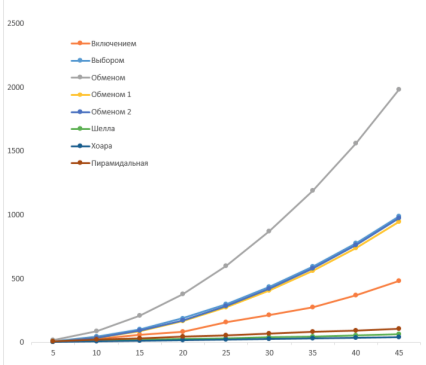




Упорядоченные по убыванию

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Количество элементов в массиве \* 1000 | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Включением | 16 | 63 | 181 | 315 | 432 | 630 | 863 | 1180 | 1359 |
| Выбором | 16 | 62 | 154 | 301 | 418 | 608 | 905 | 1136 | 1396 |
| Обменом | 16 | 80 | 179 | 253 | 412 | 589 | 791 | 1011 | 1322 |
| Обменом 1 | 16 | 66 | 146 | 275 | 443 | 560 | 748 | 974 | 1280 |
| Обменом 2 | 16 | 71 | 155 | 268 | 385 | 585 | 795 | 1011 | 1260 |
| Шелла | 5 | 20 | 48 | 81 | 177 | 180 | 239 | 307 | 392 |
| Хоара | 1 | 7 | 13 | 19 | 30 | 40 | 54 | 74 | 90 |
| Пирамидальная | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 2 |





**Стэк**

#include "pch.h"

#include <stdio.h>

#include <stdlib.h>

#define STACK\_MAX\_SIZE 20

#define STACK\_OVERFLOW -100//выход за границы массива

#define STACK\_UNDERFLOW -101

#define StakOk 0

#define StakEmpty 1

#define StakFull 2

typedef struct Stack\_tag {

int data[STACK\_MAX\_SIZE];

size\_t size;

} Stack\_t;

void printStackValue(const int value);

void printStack(const Stack\_t \*stack, void(\*printStackValue)(const int));

void StakInit(Stack\_t \*s);

void push(Stack\_t \*stack, const int value);

int pop(Stack\_t \*stack);

int isEmpty(Stack\_t \*s);

int isFull(Stack\_t \*s);

int StakError;

int main()

{

}

void printStackValue(const int value) {

printf("%d", value);

}

void printStack(const Stack\_t \*stack, void(\*printStackValue)(const int)) {

int i;

int len = stack->size - 1;

printf("stack %d > ", stack->size);

for (i = 0; i < len; i++) {

printStackValue(stack->data[i]);

printf(" | ");

}

if (stack->size != 0) {

printStackValue(stack->data[i]);

}

printf("\n");

}

void StakInit(Stack\_t \*s) {

s->size = 0;

StakError = StakOk;

}

void push(Stack\_t \*stack, const int value) {//Кладём новый элемент на стек.

if (isFull(stack)) {

exit(STACK\_OVERFLOW);

}

stack->data[stack->size] = value;

stack->size++;

}

int pop(Stack\_t \*stack) {//возвращает элемент с вершины

if (isEmpty(stack)) {

exit(STACK\_UNDERFLOW);

}

stack->size--;

return stack->data[stack->size];

}

int isEmpty(Stack\_t \*s) {

return (s->size == 0) ? 1 : 0;

}

int isFull(Stack\_t \*s) {

return (s->size == STACK\_MAX\_SIZE) ? 1 : 0;

}

**Циклическая очередь**

#include "pch.h"

#include <stdio.h>

#include <stdlib.h>

#define MAX\_ARR 20

#define OVERFLOW -1 //переполнение массива

#define Empty -2 //пустой массив

typedef struct {

int arr[MAX\_ARR];

int first;

int last;

int size;

}turn;

void push(turn \*a, int value);

int pop(turn \*a);

void prinTurn(turn \*a);

int isEmpty(turn \*a);

int isFull(turn \*a);

int main()

{

turn a;

a.first = 0;

a.last = 0;

a.size = 0;

push(&a, 1);

push(&a, 2);

push(&a, 3);

prinTurn(&a);

}

void push(turn \*a, int value) {

a->size++;

if (isFull(a))

exit(OVERFLOW);

a->last++;

if (a->last == MAX\_ARR)

a->last = 0;

a->arr[a->last - 1] = value;

}

int pop(turn \*a) {

if (isEmpty(a))

exit(Empty);

int s;

s = a->arr[a->first];

a->first++;

if (a->first == 20)

a->first = 0;

return s;

}

void prinTurn(turn \*a) {

int i;

for (i = 0; i < a->size; i++) {

printf("%d ", pop(a));

}

return;

}

int isEmpty(turn \*a) {

return (a->size == 0) ? 1 : 0;

}

int isFull(turn \*a) {

return (a->size == MAX\_ARR) ? 1 : 0;

}