Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №3**по дисциплине «Структуры и организация данных»  
на тему «Оценка эффективности алгоритмов»  
часть 1 «Алгоритмы сортировки»  
  
вариант 4

Выполнил:  
Студент Антонов Н. М.  
Группа И924Б   
  
Преподаватель:  
Гладевич А.А.

Санкт-Петербург  
2023г.

Уровень сложности – базовый. Провести сравнение указанных алгоритмов сортировки массивов, содержащих N1, N2, N3 и N4 элементов, по указанному в вариативной части критерию и объему требуемой дополнительно памяти.

Порядок: по возрастанию элементов.

Методы: пузырька, шейкера, быстрая сортировка, сортировка естественным слиянием.

N1 = 10000, N2 = 50000, N3 = 100000, N4 = 150000. Критерий – количество сравнений.

1. Алгоритм сортировки пузырьком:

Алгоритмпроходит по списку несколько раз, сравнивая два соседних элемента и меняя их местами, если они находятся в неправильном порядке.

Трудоемкость сортировки пузырьком по количеству сравнений:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Случай** | **Ситуация, соответст-вующая случаю** | **Обоснование** | **Ожидаемое число**  **сравнений элементов массива** | **Асимптотическая оценка сложности по количеству сравнений** | **Ожидаемое число вспомогат.**  **сравнений** |
| Наилучший  Средний  Наихудший | Массив может быть любым | Сортировка не зависит от упорядоченности элементов, она работает всегда одинаково | N1: 49995000  N2: 1249975000  N3: 499950000  N4: 11249925000 | O(n2) | N1: 50015001  N2: 1250075001  N3: 5000150001  N4: 11250225001 |

Пространственная сложность – O(1) (две переменных цикла и вспомогательная переменная).

1. Алгоритм шейкерной сортировки:

Алгоритм последовательно просматривает элементы массива слева направо, сравнивая соседние элементы и, если они находятся в неправильном порядке, меняет их местами. Затем алгоритм движется в обратном направлении — справа налево, опять выполняя сравнения и перемещения. Этот процесс продолжается до тех пор, пока не будет достигнут правильный порядок элементов.

Трудоемкость шейкерной сортировки по количеству сравнений:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Случай** | **Ситуация, соответст-вующая случаю** | **Обоснование** | **Ожидаемое число сравнений элементов массива** | **Асимптотическая оценка сложности по количеству сравнений** | **Ожидаемое число вспомогат. сравнений** |
| Наилучший | Массив упорядочен по возрастанию | Массив уже полностью отсортирован | *n-1*  N1: 9999  N2: 49999  N3: 99999  N4: 149999 | Ω(n) | *n+2*  N1: 10002  N2: 50002  N3: 100002  N4: 150002 |
| Наихудший | Массив упорядочен по убыванию | Потребуется отсортировать массив в обратном порядке | N1: 49995000  N2: 1249975000  N3: 4999950000  N4: 11249925000 | O(n2) | N1: 33343334  N2: 833383334  N3: 3333433334  N4: 7500150000 |
| Средний | Частично упорядоченный массив | Массив может быть частично упорядочен | N1: 33330000  N2: 833317000  N3: 3333300000  N4: 7499950000 | O(n2) | N1: 33330000  N2: 833317000  N3: 3333300000  N4: 7499950000 |

Пространственная сложность – O(1) (две переменных цикла и 3 вспомогательных переменных).

1. Алгоритм быстрой сортировки:

Алгоритм сначала выбирает опорный элемент, далее массив делится на две подгруппы: меньшие опорного элемента и большие опорного элемента. Далее рекурсивно повторяется алгоритм, пока подгруппы не станут размером меньше или равному 1.

Трудоемкость быстрой сортировки по количеству сравнений:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Случай** | **Ситуация, соответст-вующая случаю** | **Обоснование** | **Ожидаемое число сравнений элементов массива** | **Асимптотическая оценка сложности по количеству сравнений** | **Ожидаемое число вспомогат. сравнений** |
| наилучший | Каждый опорный элемент является медианой сортируемой части | Достаточно совершать вызовы только медианы | *n\*log(n)*  N1: 132877  N2: 780482  N3: 1660960  N4: 2579190 | Ω(n\*log(n)) | *n/2 \* log(n/2)*  N1: 61438  N2: 365241  N3: 780482  N4: 1214600 |
| наихудший | Каждый опорный элемент является минимальным или максимальным в сортируемой части | На каждом этапе массив из n элементов будет делиться на левую часть, состоящую из n-1 элементов, а правая часть будет одним единственным числом, будут совершаться n2 проходов | N1: 49995000  N2: 1249975000  N3: 4999950000  N4: 11249925000 | O(n2) | N1: 24997500  N2: 624987500  N3: 2499975000  N4: 5624962500 |
| средний | При делении массива на подгруппы, получившиеся массивы будут не очень сильно отличаться по длине | Разделяй и властвуй принцип | N1: 177169  N2: 1040642  N3: 2214620  N4: 3438920 | O(n\*log(n)) | N1: 88584  N2: 520321  N3: 1107310  N4: 1719460 |

Пространственная сложность – O(n) (создается временный массив размером n).

1. Алгоритм сортировки естественным слиянием:

Алгоритм начинается с деления исходного списка на отдельные подсписки. Это делается путем прохода по списку и нахождения участков, где элементы находятся в возрастающем порядке. Когда обнаруживается участок, где элементы идут в порядке убывания, он считается концом текущего подсписка. Затем этот подсписок добавляется в список отсортированных подсписков. Если список имеет только один подсписок (т.е. все элементы уже упорядочены), алгоритм сортировки считается завершенным.

Трудоемкость сортировки естественным слиянием по количеству сравнений:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Случай** | **Ситуация, соответст-вующая случаю** | **Обоснование** | **Ожидаемое число сравнений элементов массива** | **Асимптотическая оценка сложности по количеству сравнений** | **Ожидаемое число вспомогат. сравнений** |
| Наилучший | Массив уже полностью отсортирован | В этом случае сортировка естественным слиянием может быстро обнаружить отсортированные подсписки и объединит их | *1*  N1: 1  N2: 1  N3: 1  N4: 1 | Ω(n) | *n-1*  N1: 9999  N2: 49999  N3: 99999  N4: 149999 |
| Наихудший | Массив обратно упорядочен | В этом случае каждый элемент будет рассматриваться как отдельный подсписок, и при каждом объединении будет создаваться новый подсписок | N1: 49995000  N2: 1249975000  N3: 4999950000  N4: 11249925000 | O(n2) | N1: 50015001  N2: 1250075001  N3: 5000150001  N4: 11250225001 |
| Средний | Массив частично упорядочен | Некоторые подмассивы будут уже находиться в правильном порядке | *3\*n\*log(n)*  N1: 398631  N2: 2341450  N3: 4982890  N4: 7737570 | O(n\*log(n)) | N1: 465069  N2: 2731690  N3: 5813370  N4: 9027170 |

Пространственная сложность – O(n) (создается временный массив размером n).

Текст программы:

Файл Bubble.cpp:

#include <iostream>

#include <fstream>

#include <algorithm>

#include <ctime>

#include <Windows.h>

#define MAXSTACK 2048

using namespace std;

unsigned long long int memory, cntosn, cntvtr;

void BubbleSort(long\* arr, int n) {

int x = 0;

cntosn = 0; // Счетчик основных сравнений

cntvtr = 0; // Счетчик второстепенных сравнений

for (int i = 1; cntvtr++, i < n; i++)

for (int j = n - 1; cntvtr++, j >= i; j--)

if (cntosn++, arr[j - 1] < arr[j])

{

x = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = x;

}

memory = 3 \* sizeof(int);

cout << "Основные сравнения: " << cntosn << endl;

cout << "Второстепенные сравнения: " << cntvtr << endl;

cout << "Количество затраченной памяти: " << memory << " bytes" << endl;

}

int main(int argc, char\* argv[]) {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int n1 = 10000, n2 = 50000, n3 = 100000, n4 = 150000;

long\* arr1 = new long[10000], \* arr2 = new long[50000], \* arr3 = new long[100000], \* arr4 = new long[150000];

int w;

clock\_t start = 0, end = 0;

char menu;

ifstream infile("C:\\Users\\NiKiToS\\Downloads\\test\_numbers (2).txt");

if (!infile)

{

cout << "Error openning file";

return 0;

}

system("cls");

do {

cout << "Сортировка пузырьком" << endl;

cout << "Сколько элементов сортировать?" << endl;

cout << "1. 10000" << endl;

cout << "2. 50000" << endl;

cout << "3. 100000" << endl;

cout << "4. 150000" << endl;

cout << "0. Выход" << endl;

cin >> menu;

switch (menu) {

case '1':

infile.seekg(0);

for (int i = 0; i < n1; i++) {

infile >> w;

arr1[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

BubbleSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

BubbleSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n1 / 2; i++)

arr1[i] = arr1[n1 - i - 1];

BubbleSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '2':

infile.seekg(0);

for (int i = 0; i < n2; i++) {

infile >> w;

arr2[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

BubbleSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

BubbleSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n2 / 2; i++)

arr2[i] = arr2[n2 - i - 1];

BubbleSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '3':

infile.seekg(0);

for (int i = 0; i < n3; i++) {

infile >> w;

arr3[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

BubbleSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

BubbleSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n3 / 2; i++)

arr3[i] = arr3[n3 - i - 1];

BubbleSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '4':

infile.seekg(0);

for (int i = 0; i < n4; i++) {

infile >> w;

arr4[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

BubbleSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

BubbleSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n4 / 2; i++)

arr4[i] = arr4[n4 - i - 1];

BubbleSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '0':

break;

default:

cout << "INVALID CHOICE" << endl;

break;

}

} while (menu != '0');

infile.close();

return 0;

}

Файл Merge.cpp:

#include <iostream>

#include <fstream>

#include <algorithm>

#include <ctime>

#include <Windows.h>

using namespace std;

unsigned long long int memory, cntosn, cntvtr;

void mergeSort(long\* a, int n)

{

memory = 0;

cntosn = 0;

cntvtr = 0;

int split; //индекс по которому делим массив

int last, end;

long\* p = a, \* tmp;

char flag = 0, sorted = 0;

int pos1, pos2, pos3;

tmp = new long[n];

do

{

end = n;

pos2 = pos3 = 0;

do/\* если есть более 1 элемента \*/

{

p += pos2;

end = n - pos3;

for (split = 1; split < end && p[split - 1] <= p[split]; split++)/\*первая серия\*/

cntvtr++;

cntosn++;

if (cntvtr++, split == n)

{

sorted = 1;

break;

}

pos1 = 0;

pos2 = split;

while (pos1 < split && pos2 < end)/\*идет слияние, пока есть хоть один элемент в каждой серии\*/

{

cntvtr++;

cntosn++;

if (p[pos1] > p[pos2])

{

tmp[pos3++] = p[pos2++];

}

else

{

tmp[pos3++] = p[pos1++];

}

}

/\* одна последовательность закончилась - копировать остаток другой в конец буфера \*/

while (pos2 < end && tmp[pos3 - 1] <= p[pos2])/\* пока вторая последовательность не пуста \*/

{

cntvtr++;

cntosn++;

tmp[pos3++] = p[pos2++];

}

while (pos1 < split)/\* пока первая последовательность не пуста \*/

{

tmp[pos3++] = p[pos1++];

}

} while (pos3 < n);

if (sorted) {

break;

cntosn++;

}

p = tmp;

tmp = a;

a = p;

flag = !flag;

} while (split < n);

if (flag)

{

cntosn++;

for (pos1 = 0; pos1 < n; pos1++)

tmp[pos1] = a[pos1];

delete[] a;

}

else

{

delete[] tmp;

}

memory += 6 \* sizeof(int);

memory += 2 \* sizeof(long);

memory += 2 \* sizeof(char);

cout << "Основные сравнения: " << cntosn << endl;

cout << "Второстепенные сравнения: " << cntvtr << endl;

cout << "Количество затраченной памяти: " << memory << " bytes" << endl;

}

int main(int argc, char\* argv[]) {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int n1 = 10000, n2 = 50000, n3 = 100000, n4 = 150000;

long\* arr1 = new long[10000], \* arr2 = new long[50000], \* arr3 = new long[100000], \* arr4 = new long[150000];

int w;

clock\_t start = 0, end = 0;

char menu;

ifstream infile("C:\\Users\\NiKiToS\\Downloads\\test\_numbers (2).txt");

if (!infile)

{

cout << "Error openning file";

return 0;

}

system("cls");

do {

cout << "Сортировка естественным слиянием" << endl;

cout << "Сколько элементов сортировать?" << endl;

cout << "1. 10000" << endl;

cout << "2. 50000" << endl;

cout << "3. 100000" << endl;

cout << "4. 150000" << endl;

cout << "0. Выход" << endl;

cin >> menu;

switch (menu) {

case '1':

infile.seekg(0);

for (int i = 0; i < n1; i++) {

infile >> w;

arr1[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

mergeSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

mergeSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n1 / 2; i++)

arr1[i] = arr1[n1 - i - 1];

mergeSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '2':

infile.seekg(0);

for (int i = 0; i < n2; i++) {

infile >> w;

arr2[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

mergeSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

mergeSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n2 / 2; i++)

arr2[i] = arr2[n2 - i - 1];

mergeSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '3':

infile.seekg(0);

for (int i = 0; i < n3; i++) {

infile >> w;

arr3[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

mergeSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

mergeSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n3 / 2; i++)

arr3[i] = arr3[n3 - i - 1];

mergeSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '4':

infile.seekg(0);

for (int i = 0; i < n4; i++) {

infile >> w;

arr4[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

mergeSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

mergeSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n4 / 2; i++)

arr4[i] = arr4[n4 - i - 1];

mergeSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '0':

break;

default:

cout << "INVALID CHOICE" << endl;

break;

}

} while (menu != '0');

infile.close();

return 0;

}

Файл QuickSort.cpp:

#include <iostream>

#include <fstream>

#include <algorithm>

#include <ctime>

#include <cmath>

#include <Windows.h>

#define MAXSTACK 2048

using namespace std;

unsigned long long int memory, cntosn, cntvtr;

void quicksort(long\* a, int n) {

cntosn = 0;

cntvtr = 0;

memory = 0;

const long M = log(n) / log(2) + 1;

long i, j, left, right, s, x, w;

struct stack { long left, right; } \*stack;

stack = new struct stack[M];

s = 0;

stack[0].left = 0;

stack[0].right = n - 1;

memory += sizeof(stack);

memory += sizeof(int);

do /\*выбор из стека последнего запроса\*/

{

left = stack[s].left;

right = stack[s].right;

s--;

do /\*разделение а[left]... a[right]\*/

{

i = left; j = right;

x = a[(left + right) / 2];

do

{

while (cntosn++, a[i] < x) i++;

while (cntosn++, x < a[j]) j--;

if (cntvtr++, i <= j)

{

w = a[i]; a[i] = a[j]; a[j] = w;

i++; j--;

}

} while (cntvtr++, i < j);

if (cntvtr++, i < right && (cntvtr++, right - i >= j - left)) /\*если правая часть не меньше левой\*/

{ /\*запись в стек границ правой части\*/

s++;

stack[s].left = i;

stack[s].right = right;

right = j; /\*теперь left и right ограничивают левую часть\*/

}

else if (cntvtr++, j > left && (cntvtr++, j - left > right - i)) /\*если левая часть больше правой\*/

{ /\*запись в стек границ левой части\*/

s++;

stack[s].left = left;

stack[s].right = j;

left = i; /\*теперь left и right ограничивают правую часть\*/

}

else left = right; /\*делить больше нечего, интервал "схлопывается"\*/

} while (cntvtr++, left < right);

} while (cntvtr++, s > -1);

delete[] stack;

cout << "Основные сравнения: " << cntosn << endl;

cout << "Второстепенные сравнения: " << cntvtr << endl;

cout << "Количество затраченной памяти: " << memory << " bytes" << endl;

}

int main(int argc, char\* argv[]) {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int n1 = 10000, n2 = 50000, n3 = 100000, n4 = 150000;

long\* arr1 = new long[10000], \* arr2 = new long[50000], \* arr3 = new long[100000], \* arr4 = new long[150000];

int w;

clock\_t start = 0, end = 0;

char menu;

ifstream infile("C:\\Users\\NiKiToS\\Downloads\\test\_numbers (2).txt");

if (!infile)

{

cout << "Error openning file";

return 0;

}

system("cls");

do {

cout << "Быстрая сортировка" << endl;

cout << "Сколько элементов сортировать?" << endl;

cout << "1. 10000" << endl;

cout << "2. 50000" << endl;

cout << "3. 100000" << endl;

cout << "4. 150000" << endl;

cout << "0. Выход" << endl;

cin >> menu;

switch (menu) {

case '1':

infile.seekg(0);

for (int i = 0; i < n1; i++) {

infile >> w;

arr1[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

quicksort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

quicksort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n1 / 2; i++)

arr1[i] = arr1[n1 - i - 1];

quicksort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '2':

infile.seekg(0);

for (int i = 0; i < n2; i++) {

infile >> w;

arr2[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

quicksort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

quicksort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n2 / 2; i++)

arr2[i] = arr2[n2 - i - 1];

quicksort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '3':

infile.seekg(0);

for (int i = 0; i < n3; i++) {

infile >> w;

arr3[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

quicksort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

quicksort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n3 / 2; i++)

arr3[i] = arr3[n3 - i - 1];

quicksort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '4':

infile.seekg(0);

for (int i = 0; i < n4; i++) {

infile >> w;

arr4[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

quicksort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

quicksort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n4 / 2; i++)

arr4[i] = arr4[n4 - i - 1];

quicksort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '0':

break;

default:

cout << "INVALID CHOICE" << endl;

break;

}

} while (menu != '0');

infile.close();

return 0;

}

Файл Shaker.cpp:

#include <iostream>

#include <fstream>

#include <algorithm>

#include <ctime>

#include <Windows.h>

using namespace std;

unsigned long long int memory, cntosn, cntvtr;

void cocktailSort(long a[], int n) {

int k = n - 1, left = 1, right = n - 1, x, j = 0;

cntosn = 0;

cntvtr = 0;

memory = 0;

memory += sizeof(int) \* 5;

int start = 0;

int end = n - 1;

do

{

for (j = right; cntvtr++, j >= left; j--)/\*сначала просматриваем справа налево\*/

{

if (cntosn++, a[j - 1] > a[j])

{

x = a[j - 1];

a[j - 1] = a[j];

a[j] = x;

k = j;

}

}

left = k + 1;

for (j = left; cntvtr++, j <= right; j++)/\*теперь просматриваем слева направо\*/

{

if (cntosn++,a[j - 1] > a[j])

{

x = a[j - 1];

a[j - 1] = a[j];

a[j] = x;

k = j;

}

}

right = k - 1;

} while (cntvtr++, left < right); /\*так до тех пор, пока есть что просматривать\*/

cout << "Основные сравнения: " << cntosn << endl;

cout << "Второстепенные сравнения: " << cntvtr << endl;

cout << "Количество затраченной памяти: " << memory << " bytes" << endl;

}

int main(int argc, char\* argv[]) {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int n1 = 10000, n2 = 50000, n3 = 100000, n4 = 150000;

long\* arr1 = new long[10000], \* arr2 = new long[50000], \* arr3 = new long[100000], \* arr4 = new long[150000];

int w;

clock\_t start = 0, end = 0;

char menu;

ifstream infile("C:\\Users\\NiKiToS\\Downloads\\test\_numbers (2).txt");

if (!infile)

{

cout << "Error openning file";

return 0;

}

system("cls");

do {

cout << "Сортировка Шейкера" << endl;

cout << "Сколько элементов сортировать?" << endl;

cout << "1. 10000" << endl;

cout << "2. 50000" << endl;

cout << "3. 100000" << endl;

cout << "4. 150000" << endl;

cout << "0. Выход" << endl;

cin >> menu;

switch (menu) {

case '1':

infile.seekg(0);

for (int i = 0; i < n1; i++) {

infile >> w;

arr1[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

cocktailSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

cocktailSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n1 / 2; i++)

arr1[i] = arr1[n1 - i - 1];

cocktailSort(arr1, n1);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '2':

infile.seekg(0);

for (int i = 0; i < n2; i++) {

infile >> w;

arr2[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

cocktailSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

cocktailSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n2 / 2; i++)

arr2[i] = arr2[n2 - i - 1];

cocktailSort(arr2, n2);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '3':

infile.seekg(0);

for (int i = 0; i < n3; i++) {

infile >> w;

arr3[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

cocktailSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

cocktailSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n3 / 2; i++)

arr3[i] = arr3[n3 - i - 1];

cocktailSort(arr3, n3);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '4':

infile.seekg(0);

for (int i = 0; i < n4; i++) {

infile >> w;

arr4[i] = w;

}

cout << "МАССИВ НЕУПОРЯДОЧЕН" << endl;

start = clock();

cocktailSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН ПО ВОЗРАСТАНИЮ" << endl;

start = clock();

cocktailSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

cout << "МАССИВ УПОРЯДОЧЕН В ОБРАТНОМ ПОРЯДКЕ" << endl;

start = clock();

for (int i = 0; i < n4 / 2; i++)

arr4[i] = arr4[n4 - i - 1];

cocktailSort(arr4, n4);

end = clock();

cout << "Затраченное время: " << double(end - start) / CLOCKS\_PER\_SEC << endl << endl;

break;

case '0':

break;

default:

cout << "INVALID CHOICE" << endl;

break;

}

} while (menu != '0');

infile.close();

return 0;

}

Результаты работы программы:

Bubble.cpp

При запуске программ нужно выбрать, массив из скольких элементов нужно отсортировать, что показано на рисунке 1.

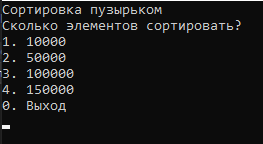


Рисунок 1 – Главное меню

При нажатии «1» сортируется массив на 10.000 элементов. Сначала сортируется неупорядоченный массив, затем отсортированный по возрастанию, а после отсортированный в обратном порядке. Выводится количество основных и второстепенных сравнений, количество затраченной памяти и время выполнения сортировки, что показано на рисунке 2.

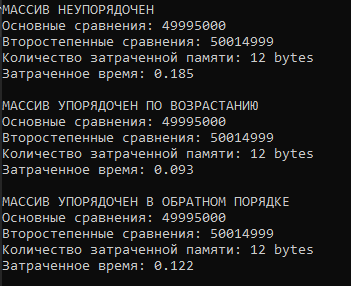


Рисунок 2 – Сортировка трех разновидностей упорядоченности массива

При нажатии «2», «3» и «4» производится та же работа, но для массива из 50.000, 100.000 и 150.000 элементов соответственно.

Merge.cpp:

При запуске программ нужно выбрать, массив из скольких элементов нужно отсортировать, что показано на рисунке 3.

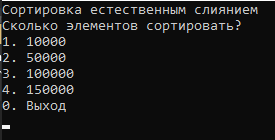


Рисунок 3 – Главное меню

При нажатии «1» сортируется массив на 10.000 элементов. Сначала сортируется неупорядоченный массив, затем отсортированный по возрастанию, а после отсортированный в обратном порядке. Выводится количество основных и второстепенных сравнений, количество затраченной памяти и время выполнения сортировки, что показано на рисунке 4.

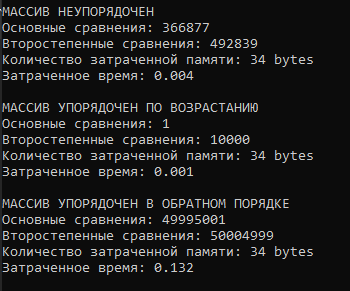


Рисунок 4 – Сортировка трех разновидностей упорядоченности массива

При нажатии «2», «3» и «4» производится та же работа, но для массива из 50.000, 100.000 и 150.000 элементов соответственно.

QuickSort.cpp:

При запуске программ нужно выбрать, массив из скольких элементов нужно отсортировать, что показано на рисунке 5.

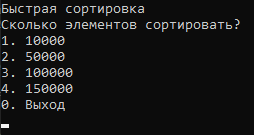


Рисунок 5 – Главное меню

При нажатии «1» сортируется массив на 10.000 элементов. Сначала сортируется неупорядоченный массив, затем отсортированный по возрастанию, а после отсортированный в обратном порядке. Выводится количество основных и второстепенных сравнений, количество затраченной памяти и время выполнения сортировки, что показано на рисунке 6.

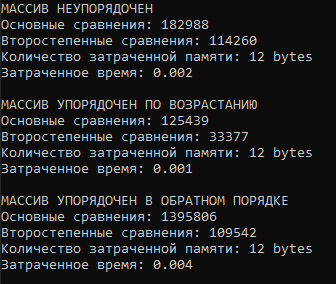


Рисунок 6 – Сортировка трех разновидностей упорядоченности массива

При нажатии «2», «3» и «4» производится та же работа, но для массива из 50.000, 100.000 и 150.000 элементов соответственно.

Shaker.cpp

При запуске программ нужно выбрать, массив из скольких элементов нужно отсортировать, что показано на рисунке 7.

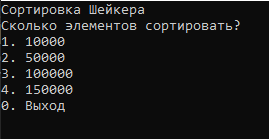


Рисунок 7 – Главное меню

При нажатии «1» сортируется массив на 10.000 элементов. Сначала сортируется неупорядоченный массив, затем отсортированный по возрастанию, а после отсортированный в обратном порядке. Выводится количество основных и второстепенных сравнений, количество затраченной памяти и время выполнения сортировки, что показано на рисунке 8.

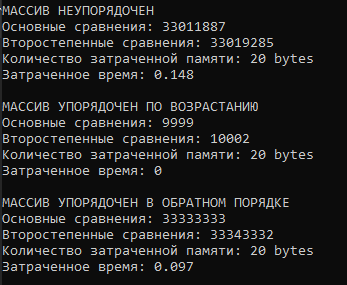


Рисунок 8 – Сортировка трех разновидностей упорядоченности массива

При нажатии «2», «3» и «4» производится та же работа, но для массива из 50.000, 100.000 и 150.000 элементов соответственно.

При выборе несуществующего пункта меню выводится ошибка, рисунок 9.



Рисунок 9 – Выбор несуществующего пункта меню

При нажатии «0» происходит выход из программы.

Анализ полученных данных:

Полученные данные в сортировке пузырьком полностью соответствуют расчетным. Количество основных сравнений всегда меньше второстепенных, вне зависимости от количества элементов в массиве.

Полученные данные в шейкерной сортировке отличаются от расчетных с небольшой погрешностью из-за неучтенного коэффициента. В лучшем случае количество второстепенных сравнений больше основных, в худшем основных сравнений больше второстепенных, с среднем их количество одинаково.

Полученные данные в быстрой сортировке отличаются от расчетных с небольшой погрешностью из-за неучтенного коэффициента. В лучшем случае количество основных сравнений больше второстепенных, в худшем случае количество основных сравнений больше второстепенных, в среднем количество основных сравнений также больше второстепенных.

Полученный данные в сортировке естественным слиянием отличаются от расчетных с небольшой погрешностью из-за неучтенного коэффициента. В лучшем случае основных сравнений почти не происходит, в худшем второстепенных сравнений больше, чем основных, в среднем второстепенных сравнений также больше, чем основных.

По трудоемкости:

В худшем случае самая оптимальная сортировка – быстрая сортировка, так как происходит меньше второстепенных сравнений.

В среднем случае самая оптимальная сортировка – быстрая сортировка, так как происходит меньше как основных, так и второстепенных сравнений.

В лучшем случае самая оптимальная сортировка – естественным слиянием, так как происходит наименьшее количество основных и второстепенных сравнений.

По времени:

В худшем случае быстрая сортировка самая эффективная.

В среднем случае быстрая сортировка и сортировка естественным слиянием работают одинаково.

В лучшем случае сортировка шейкера, быстрая сортировка и сортировка естественным слиянием работают одинаково.

По пространственной сложности:

Лучшая пространственная сложность у сортировки пузырьком и сортировки шейкера, они используют минимальное количество дополнительной памяти O(1).

Худшая пространственная сложность у быстрой сортировки и сортировки естественным слиянием, так как они используют количество памяти O(n), где n – размер дополнительно создаваемого массива.

Вывод: по результатам проведенного анализа самым эффективным из рассмотренных алгоритмов по соотношению время-память является алгоритмы быстрой сортировки и сортировки естественным слиянием.