Пермский национальный исследовательский политехнический университет.

Лабораторная работа.

«Сортировки и поиски».

Выполнил: студент группы РИС-23-2б

Вековшинин Иван Николаевич

Проверила: доцент кафедры ИТАС

О.А. Полякова.

2024 г.

**Разработка алгоритма сортировки.**

1. **Постановка задачи №1:**

Отсортировать массив методами: слияние, подсчет, блочная, быстрая, естественное слияние, многофазное слияние.

Осуществить поиск методами: линейная, бинарная, интерполяционная, Бойера-Мура, Кнута-Морриса-Пратта.

1. **Листинг программы**

**Сортировка:**

**Слияние:**

#include<iostream>

using namespace std;

//функция, сливающая массивы

void Merge(int\* A, int first, int last) {

int middle, start, final, j;

int\* mas = new int[100];

middle = (first + last) / 2; //вычисление среднего элемента

start = first; //Начало левой части

final = middle + 1; //начало правой части

for (j = first; j <= last; j++) {

if ((start <= middle) && ((final > last) || (A[start] < A[final])))

{

mas[j] = A[start];

start++;

}

else

{

mas[j] = A[final];

final++;

}

}

for (j = first; j <= last; j++) {

A[j] = mas[j];

}

delete[] mas;

}

//рукурсивная процедура сортировки

void Sort(int\* A, int first, int last) {

if (first < last) {

Sort(A, first, (first + last) / 2); //сортировка левой части

Sort(A, (first + last) / 2 + 1, last); //сортировка правой части

Merge(A, first, last); //слияние двух частей

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

srand(time(0));

int i, n = 25;

int\* A = new int[n]{50, 1, 24, 32, 3, 33, 53, 12, 13, 0, 11, -30, 23, 6, 56, 17, 13, 81, 44, 33, 100, 7, 8};

Sort(A, 0, n - 1);

cout << endl << "Упорядоченный массив: ";

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

cout << A[i] << " ";

}

return 0;

}

**Методом подсчета:**

#include<iostream>

using namespace std;

void sort\_counting\_method(int A[], int N)

{

int max = A[0];

int min = A[0];

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

if (A[i] > max) max = A[i];

else

{

if (A[i] < min) min = A[i];

}

}

int L = max - min + 1;

int\* B = new int[L] {};

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

B[A[i] - min]++;

}

int k = 0;

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

for (int j = 0; j < B[i]; j++) {

A[k++] = i + min;

}

}

delete[] B;

}

int main()

{

setlocale(LC\_ALL, "Ru");

int N = 25;

int\* A = new int[N] {0, 4, 3, 2, 1, 6, 5, 4, 2, 3, 0, 1, 4, 6, 5, 4, 5, 2, 1, 0, 6, 5, 3, 1, 0};

sort\_counting\_method(A, N);

cout << "Массив после сортировки: " << endl;

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

cout << A[i] << " ";

}

cout << endl;

delete[] A;

return 0;

}

**Методом блоков:**

#include<iostream>

using namespace std;

void bucketSort(int arr[], int n) {

int max\_value = arr[0];

int min\_value = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > max\_value) max\_value = arr[i];

if (arr[i] < min\_value) min\_value = arr[i];

}

int bucket\_count = ((max\_value - min\_value) / 10) + 1;

int\*\* buckets = new int\* [bucket\_count];

for (int i = 0; i < bucket\_count; i++) {

buckets[i] = new int[n];

for (int j = 0; j < n; j++)

{

buckets[i][j] = max\_value + 1;

}

}

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

int bucket\_index = ((arr[i] - min\_value) / 10);

for (int j = 0; j < n; j++)

{

buckets[bucket\_index][i] = arr[i];

}

}

for (int i = 0; i < bucket\_count; i++) {

for (int j = 0; j < n; j++) {

for (int k = j + 1; k < n; k++) {

if (buckets[i][j] > buckets[i][k]) {

int temp = buckets[i][j];

buckets[i][j] = buckets[i][k];

buckets[i][k] = temp;

}

}

}

}

int index = 0;

for (int i = 0; i < bucket\_count; i++) {

for (int j = 0; j < n; j++) {

if (buckets[i][j] != max\_value + 1) {

arr[index++] = buckets[i][j];

}

}

}

for (int i = 0; i < bucket\_count; i++) {

delete[] buckets[i];

}

delete[] buckets;

}

int main()

{

setlocale(LC\_ALL, "rus");

int n = 25;

int\* mas = new int[n]{0, 10, 1, 34, 50, 49, 35, 12, 15, 28, 32, 27, 14, 47, 41, 36, 37, 43, 23, 24, 7, 1, 11, 14, 24};

bucketSort(mas, n);

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

cout << mas[i] << " ";

}

cout << endl;

delete [] mas;

return 0;

}

**Быстрая сортировка:**

#include<iostream>

using namespace std;

void quickSort(int\* mas, int size) {

//Указатели в начало и в конец массива

int i = 0;

int j = size - 1;

int pivot = mas[size / 2];//ключевой элемент

//Делим массив

do

{

while (mas[i] < pivot) { //В левой части пропускаме элементы, которые меньше ключевого

i++;

}

while (mas[j] > pivot) { //В правой части пропускаем элементы, которые болье ключевого

j--;

}

if (i <= j) {

swap(mas[i], mas[j]);

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, если осталось, что сортировать

if (j > 0) {

quickSort(mas, j + 1);//левая часть

}

if (i < size) {

quickSort(&mas[i], size - i);

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int i, n = 25;

int\* A = new int[n] {101, 3, 1, 24, 45, 343, 22, 34, 4, 3, 3 ,3443, 43, 4, 232, 34, 3, 343 ,34, 4, 4, 4, 13, 23, 25};

quickSort(A, n + 1);

cout << "Упорядоченный массив: ";

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

cout << A[i] << " ";

}

cout << endl;

return 0;

}

**Сортировка методом Шелла:**

#include<iostream>

using namespace std;

void printMas(int mas[], int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

void ShellSort(int mas[], int size) {

int d = size / 2;

bool f = true;

while (f)

{

f = false;

for (int i = 0; i < size - d; i++)

{

if (mas[i] > mas[i + d])

{

int tmp = mas[i];

mas[i] = mas[i + d];

mas[i + d] = tmp;

f = true;

}

}

if (d > 1)

{

d = d / 2;

}

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25;

int\* A = new int[n] {25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

cout << "Исходный массив:\n";

printMas(A, n);

ShellSort(A, n);

cout << "Упорядоченный массив:\n";

printMas(A,n);

return 0;

}

**Сортировка методом естественного слияния:**

#include<iostream>

using namespace std;

void printMas(int mas[], int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

void SortSeries(int\* mas, int indexS1, int indexS2){

int insertIndex;

bool flag = false;

for (int i = 0; i <= indexS1; i++)//проход по первой серии

{

for (int j = indexS1+1; j <= indexS2; j++)//проход по второй серии

{

if (mas[i] >= mas[j])

{

insertIndex = j;

flag = true;

}

}

if (flag)

{

int tmp = mas[i];

for (int k = i; k < insertIndex; k++)

{

mas[k] = mas[k + 1];

}

mas[insertIndex] = tmp;

indexS1--;

i--;

flag = false;

}

}

}

void NaturalSort(int\* mas, int size) {

int s1 = mas[0], s2 = 0;

int indexS1, indexS2;

bool flag = true;

for (int i = 1; i < size; i++)

{

if (s1 < mas[i] && flag)

{

s1 = mas[i];

indexS1 = i;

}

else

{

if (s2 < mas[i] || flag)

{

flag = false;//s1 сформирован. начинаем формировать s2

s2 = mas[i];

indexS2 = i;

}

else

{

//с1 и с2 сформированы, далее необходимо сортировать эти серии

SortSeries(mas, indexS1, indexS2);

flag = true;

s1 = mas[i - 1];

indexS1 = i - 1;

s2 = mas[i];

indexS2 = i;

}

}

}

SortSeries(mas, indexS1, indexS2);//последний проход оставшихся с1 и с2

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25;

int\* A = new int[n] {16, 51, 8, 15, 24, 2, 1, 12, 7, 10, 3, 1, 24, 45, 43, 22, 34, 4, 3, 3, 33, 43, 4, 23, 34};

cout << "Исходный массив:\n";

printMas(A, n);

NaturalSort(A, n);

cout << "Упорядоченный массив:\n";

printMas(A,n);

return 0;

}

**Сортировка многофазным (трехфазным) методом:**

#include<iostream>

#include<vector>

using namespace std;

void printMas(vector <int> mas, int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

void divisionIntoSeries(int size, int\* a1, int\* a2, int\* empty\_series1, int\* empty\_series2)

{

bool flag = true;

vector <int> chisla\_fib{ 0,1 };

for (int i = 0; i < chisla\_fib.size() && flag; i++)

{

if (chisla\_fib[i + 1] < size)

{

chisla\_fib.push\_back(chisla\_fib[i] + chisla\_fib[i + 1]);

}

else if (chisla\_fib[i + 1] == size)

{

\*a1 = chisla\_fib[i - 1];

\*a2 = chisla\_fib[i];

flag = false;

}

else

{

\*empty\_series1 = ((chisla\_fib[i-1] + chisla\_fib[i]) - size) / 2;

int ostatokOtDelNa2\_empty\_series = ((chisla\_fib[i - 1] + chisla\_fib[i]) - size) % 2;

\*empty\_series2 = \*empty\_series1;

if (ostatokOtDelNa2\_empty\_series > 0)

{

\*empty\_series1 += 1;

}

\*a1 = chisla\_fib[i - 1] - \*empty\_series1;

\*a2 = chisla\_fib[i] - \*empty\_series2;

flag = false;

}

}

chisla\_fib.clear();

chisla\_fib.shrink\_to\_fit();//хз, пока не понимаю. возможо, полная очистка памяти

}

void sort\_buble(vector <int>\* mas, int n)

{

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if ((\*mas)[j] > (\*mas)[j + 1]) {

// меняем элементы местами

auto temp = (\*mas)[j];

(\*mas)[j] = (\*mas)[j + 1];

(\*mas)[j + 1] = temp;

}

}

}

}

void merge(int kolSymbolsInSeriesMedium, int kolSymbolsInSeriesMax, vector <int>\* medium\_file, vector <int>\* max\_file, vector <int>\* min\_file)

{

while (!(\*medium\_file).empty() && !(\*max\_file).empty())

{

for (int i = 0; i < kolSymbolsInSeriesMedium; i++)

{

min\_file->push\_back((\*medium\_file).at(0));

medium\_file->erase(medium\_file->begin() + 0);

}

for (int i = 0; i < kolSymbolsInSeriesMax; i++)

{

min\_file->push\_back((\*max\_file).at(0));

max\_file->erase(max\_file->begin() + 0);

}

}

if (medium\_file->size() < max\_file->size() && (medium\_file->size() != 0 || max\_file->size() != 0))

{

sort\_buble(min\_file, min\_file->size());

merge(kolSymbolsInSeriesMax, (kolSymbolsInSeriesMedium + kolSymbolsInSeriesMax), max\_file, min\_file, medium\_file);

}

else if (medium\_file->size() > max\_file->size() && (medium\_file->size() != 0 || max\_file->size() != 0))

{

sort\_buble(min\_file, min\_file->size());

merge((kolSymbolsInSeriesMedium + kolSymbolsInSeriesMax), kolSymbolsInSeriesMax, medium\_file, min\_file, max\_file);

}

else if (medium\_file->size() == max\_file->size())

{

sort\_buble(min\_file, min\_file->size());

}

}

void threePhaseSort(vector <int>\* mas, int size) {

int a1, a2, empty\_series1 = 0, empty\_series2 = 0, empty = 0;

//узнаем, сколько должно быть серий исходя из количества элементов в данном массиве

divisionIntoSeries(size, &a1, &a2, &empty\_series1, &empty\_series2);

//количество пустых

empty = empty\_series1 + empty\_series2;

//создаем места для хранения временных серий

vector <int> file1(a1 + empty\_series1);

vector <int> file2(a2 + empty\_series2);

vector <int> file3;

//распределяем элементы по сериям (для 25 элементов массива - 16 + 5\*(пустых) серий и 8 + 5\*(пустых) серий)

for (int i = 0; i < a1; i++)

{

file1[i] = (\*mas)[0];

mas->erase(mas->begin() + 0);

}

for (int i = 0; i < a2; i++)

{

file2[i] = (\*mas)[0];

mas->erase(mas->begin() + 0);

}

//сортировка

if (file1.size() < file2.size()) {

merge(1, 1, &file1, &file2, &file3);

}

else {

merge(1, 1, &file2, &file1, &file3);

}

int numberNotEmptyFile;

if (!file1.empty()) numberNotEmptyFile = 1;

else if (!file2.empty()) numberNotEmptyFile = 2;

else numberNotEmptyFile = 3;

//удаляем нули, которые образовались при добавлении пустых серий

switch (numberNotEmptyFile)

{

case 1: file1.erase(file1.begin(), file1.begin() + empty);

break;

case 2:file2.erase(file2.begin(), file2.begin() + empty);;

break;

case 3:file3.erase(file3.begin(), file3.begin() + empty);;

break;

}

mas->clear();

while (size > 0)

{

switch (numberNotEmptyFile)

{

case 1: {mas->push\_back(file1.at(0));

file1.erase(file1.begin() + 0); }

break;

case 2: {mas->push\_back(file2.at(0));

file2.erase(file2.begin() + 0); }

break;

case 3: {mas->push\_back(file3.at(0));

file3.erase(file3.begin() + 0); }

break;

}

size--;

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25;

vector <int> mas{16, 51, 8, 15, 24, 2, 1, 12, 7, 10, 3, 1, 24, 45, 43, 22, 34, 4, 3, 3, 33, 43, 4, 23, 34};

cout << "Исходный массив:\n";

printMas(mas, n);

threePhaseSort(&mas, n);

cout << "Упорядоченный массив:\n";

printMas(mas, n);

return 0;

}

**Алгоритмы поиска:**

**Линейный поиск:**

#include<iostream>

#include<vector>

using namespace std;

void printMas(int mas[], int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

int lineSearch(int mas[], int size, int desiredElement)

{

bool flag = false;

for (int i = 0; i < size; i++)

{

if (mas[i] == desiredElement)

{

flag = true;

return flag;

}

}

return flag;

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25, desiredElement;

int\* mas = new int[n] {16, 51, 8, 15, 24, 2, 1, 12, 7, 10, 3, 1, 24, 45, 43, 22, 34, 4, 3, 3, 33, 43, 4, 23, 34};

cout << "Исходный массив:\n";

printMas(mas, n);

cout << "Введите элемент, который необходимо найти:\n";

cin >> desiredElement;

if (lineSearch(mas, n, desiredElement))

cout << "Элемен найден!\n";

else cout << "Элемен не найден!\n";

return 0;

}

**Бинарный поиск:**

#include<iostream>

#include<vector>

using namespace std;

void printMas(int mas[], int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

void quickSort(int\* mas, int size) {

//Указатели в начало и в конец массива

int i = 0;

int j = size - 1;

int pivot = mas[size / 2];//ключевой элемент

//Делим массив

do

{

while (mas[i] < pivot) { //В левой части пропускаме элементы, которые меньше ключевого

i++;

}

while (mas[j] > pivot) { //В правой части пропускаем элементы, которые болье ключевого

j--;

}

if (i <= j) {

swap(mas[i], mas[j]);

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, если осталось, что сортировать

if (j > 0) {

quickSort(mas, j + 1);//левая часть

}

if (i < size) {

quickSort(&mas[i], size - i);

}

}

int lineSearch(int mas[], int sizeMas, int desiredElement, int first)

{

int pivot = sizeMas /2 + first;

if (pivot > 0)

{

if (mas[pivot] == desiredElement) return true;

else

{

if (mas[pivot] > desiredElement)

lineSearch(mas, sizeMas - sizeMas / 2 - 1, desiredElement, 0);

else lineSearch(mas, sizeMas - sizeMas / 2, desiredElement, pivot);

}

}

else return false;

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25, desiredElement;

int\* mas = new int[n] {16, 51, 8, 15, 24, 2, 1, 12, 7, 10, 3, 1, 24, 45, 43, 22, 34, 4, 3, 3, 33, 43, 4, 23, 34};

cout << "Исходный массив:\n";

printMas(mas, n);

cout << "Сортированный массив: \n";

quickSort(mas, n);

printMas(mas, n);

cout << "Введите элемент, который необходимо найти:\n";

cin >> desiredElement;

if (lineSearch(mas, n, desiredElement, 0))

cout << "Элемен найден!\n";

else cout << "Элемен не найден!\n";

return 0;

}

**Интерполяционный поиск:**

#include<iostream>

#include<vector>

using namespace std;

void printMas(int mas[], int size)

{

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

cout << mas[i] << " ";

}

cout << endl;

}

void quickSort(int\* mas, int size) {

//Указатели в начало и в конец массива

int i = 0;

int j = size - 1;

int pivot = mas[size / 2];//ключевой элемент

//Делим массив

do

{

while (mas[i] < pivot) { //В левой части пропускаме элементы, которые меньше ключевого

i++;

}

while (mas[j] > pivot) { //В правой части пропускаем элементы, которые болье ключевого

j--;

}

if (i <= j) {

swap(mas[i], mas[j]);

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, если осталось, что сортировать

if (j > 0) {

quickSort(mas, j + 1);//левая часть

}

if (i < size) {

quickSort(&mas[i], size - i);

}

}

int lineSearch(int mas[], int key, int left, int right)

{

if ((mas[right] - mas[left]) != 0)

{

int pivot = left + ((key - mas[left]) \* (right - left) / (mas[right] - mas[left]));

if (mas[pivot] == key) return true;

else

{

if (mas[pivot] > key)

lineSearch(mas, key, left, pivot);

else lineSearch(mas, key, pivot + 1, right);

}

}

else return false;

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n = 25, key;

int\* mas = new int[n] {16, 51, 8, 15, 24, 2, 1, 12, 7, 10, 3, 1, 24, 45, 43, 22, 34, 4, 3, 3, 33, 43, 4, 23, 34};

cout << "Исходный массив:\n";

printMas(mas, n);

cout << "Сортированный массив: \n";

quickSort(mas, n);

printMas(mas, n);

cout << "Введите элемент, который необходимо найти:\n";

cin >> key;

if (lineSearch(mas, key, 0, n-1))

cout << "Элемен найден!\n";

else cout << "Элемен не найден!\n";

return 0;

}

**Бойера-Мура:**

#include<iostream>

#include<string>

#include<vector>

#include <chrono>

using namespace std::chrono;

using namespace std;

//об этом идет речь в imageSearch

//-----------

//|д|а|т|а|\*|

//-----------

//|3|2|1|2|4|

//-----------

int BoylerSearch(string str, int sizestr, string image, int sizeimage)

{

int\* mas = new int[sizeimage];

mas[sizeimage - 1] = sizeimage - 1;//заполняем звездочку количеством букв в строке

mas[sizeimage - 3] = 1;

mas[sizeimage - 2] = 0;

for (int i = sizeimage - 3 - 1; i >= 0; i--)//sizeStr - 3 - это третий элемент с конца, то есть "т"

{

for (int j = i + 1; j < sizeimage - 2; j++)//проверяем, встречались ли после данного элемента равные ему элементы

{

if (image[i] == image[j]) { mas[i] = mas[j]; break; }

else if (j == sizeimage - 2 - 1) mas[i] = sizeimage - 2 - i;

}

if (image[i] == image[sizeimage - 2]) mas[sizeimage - 2] = mas[i];

}

if (mas[sizeimage - 2] == 0) mas[sizeimage - 2] = sizeimage - 1;

for (int i = sizeimage - 2; i < sizestr; i++)

{

int k = sizeimage - 2;

for (int j = i; j >=0 ; j--)

{

if (str[j] == image[k] && k == 0) return j;

else

{

if (str[j] != image[k])

{

int m = 0;

while (str[j] != image[m] && m < sizeimage - 1)

{

m++;

}

i += mas[m] - 1;

break;

}

}

k--;

}

}

return 0;

}

int main()

{

setlocale(LC\_ALL, "Ru");

string str = "в траве сидел кузнечик", image = "сидел";

cout << "Строка: " << str << "\tПодстрока: " << image << "\n";

image += "\*";

system\_clock::time\_point start = system\_clock::now();

int index = BoylerSearch(str, str.length(), image, image.length());

if (index > 0)

cout << "Образ найден! Индекс строки = " << index << "\n";

else cout << "Образ не найден!\n";

duration<double> duration1 = system\_clock::now() - start;

cout << "\nВремя выполнения: " << duration1.count() << endl;

str = "в траве сидел кузнечик", image = "сидела";

cout << "\nСтрока: " << str << "\tПодстрока: " << image << "\n";

image += "\*";

index = BoylerSearch(str, str.length(), image, image.length());

if (index > 0)

cout << "Образ найден! Индекс строки = " << index << "\n";

else cout << "Образ не найден!\n";

return 0;

}

**Кнута-Морриса-Пратта:**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<int> buildPrefixTable(string& pattern) {

int m = pattern.size();

vector<int> prefixTable(m, 0);

int j = 0;

for (int i = 1; i < m; i++) {

while (j > 0 && pattern[i] != pattern[j]) {

j = prefixTable[j - 1];

}

if (pattern[i] == pattern[j]) {

j++;

}

prefixTable[i] = j;

}

return prefixTable;

}

void kmpSearch(string text, string pattern) {

vector<int> prefixTable = buildPrefixTable(pattern);

int n = text.size();

int m = pattern.size();

int i = 0;

int j = 0;

while (i < n) {

if (text[i] == pattern[j]) {

i++;

j++;

if (j == m) {

cout << "Образ найден! Индекс символа: " << i - j << endl;

j = prefixTable[j - 1];

}

}

else {

if (j != 0) {

j = prefixTable[j - 1];

}

else {

i++;

}

}

}

}

int main() {

setlocale(LC\_ALL, "Ru");

string str = "ABCABCDABABCDABCDABDE", image = "ABCDABD";

kmpSearch(str, image);

return 0;

}

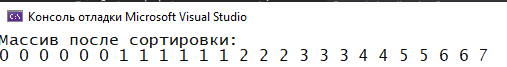
1. **Результаты работы**

**Сортировка:**

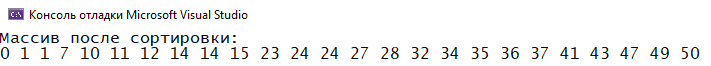
**Слияние:**



**Методом подсчета:**



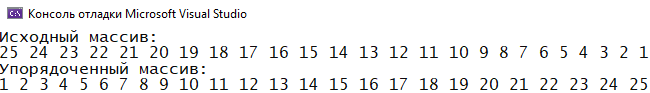
**Методом блоков:**



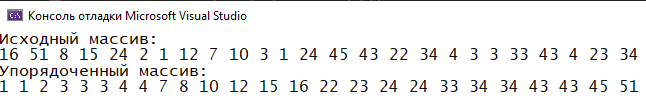
**Быстрая сортировка:**



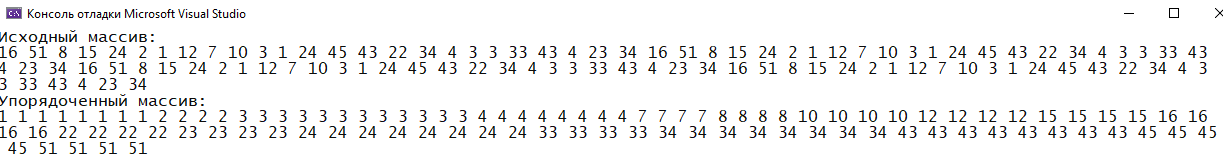
**Сортировка методом Шелла:**



**Сортировка методом естественного слияния:**

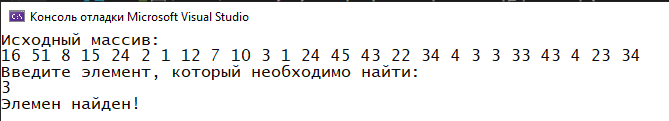


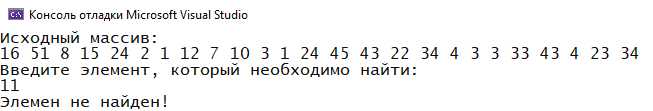
**Сортировка многофазным (трехфазным) методом:**



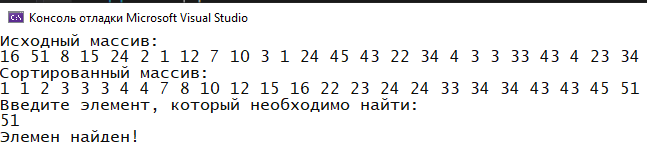
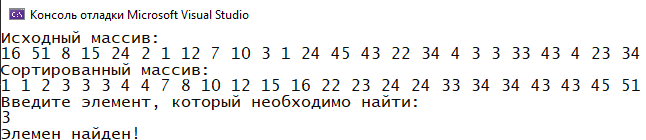
**Алгоритмы поиска:**

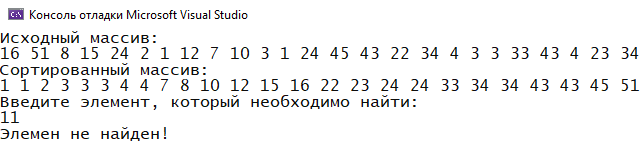
**Линейный поиск:**



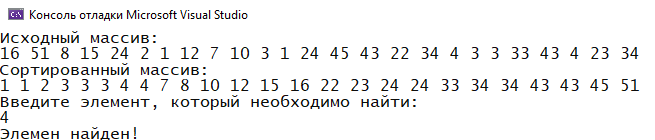


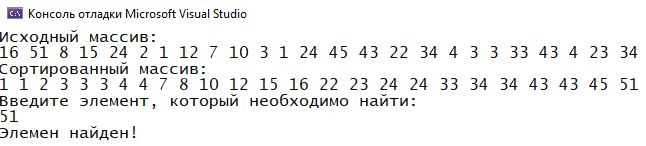
**Бинарный поиск:**

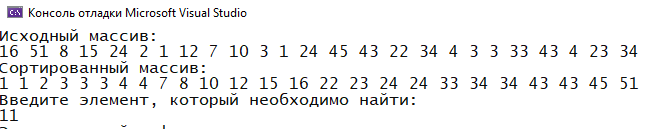




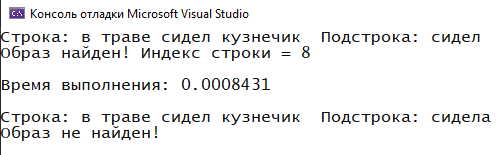
**Интерполяционный поиск:**







**Бойера-Мура:**



**Кнута-Морриса-Пратта:**

