Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Сравнение алгоритмов поиска элемента в массиве»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

1. Предисловие … 3
2. Поиск элемента в массиве… 4

2.1 Линейный поиск … 4

2.2 Бинарный поиск … 4

2.3 Интерполяционный поиск … 4

1. Реализация и сравнение по времени … 5
2. Выводы … 8

**Предисловие**

В данном реферате будут рассмотрены различные виды поиска в массиве, а также будет приведено их сравнение.

**1. Поиск элемента в массиве**

**1.1 Линейный поиск**

Линейный поиск перебирает все элементы в массиве, сравнивая их с заданным ключом, из-за полного перебора скорость поиска намного меньше, чем в других алгоритмах.

Его обычно используют, если в отрезке поиска находится мало элементов, в ином случае используют другие алгоритмы поиска.

Временная сложность данного алгоритма будет в среднем и худшем случаях O(n), сложность по памяти = O(1).

**1.2 Бинарный поиск**

Бинарный поиск - очень быстрый алгоритм с не сложной реализацией, который находит элемент с определенным значением в уже отсортированном массиве.

Очень важно помнить! Алгоритм будет работать правильно, только с отсортированным массивом. А если по случайности вы забыли отсортировать массив перед его использованием, то в большинстве случаев тот ответ, который подсчитал алгоритм, будет неверным.

Временная сложность данного алгоритма будет в среднем и худшем случаях O(log(n)), сложность по памяти = O(1).

**1.3 Интерполяционный поиск**

В основе интерполяционного поиска лежит операция интерполирование. Интерполирование – нахождение промежуточных значений величины по имеющемуся дискретному набору известных значений. Интерполяционный поиск работает только с упорядоченными массивами; он похож на бинарный, в том смысле, что на каждом шаге вычисляется некоторая область поиска, которая, по мере выполнения алгоритма, сужается.

Но в отличие от двоичного, интерполяционный поиск не делит последовательность на две равные части, а вычисляет приблизительное расположение ключа (искомого элемента), ориентируясь на расстояние между искомым и текущим значением элемента.

Временная сложность данного алгоритма будет в среднем случае O(log(log(n))), в худшем будет O(n), сложность по памяти = O(1).

**2. Реализация и сравнение по времени**

#include <iostream>

#include<Windows.h>

#include <conio.h>

#include <vector>

#include <chrono>

using namespace std;

void quick\_sort(vector<int>& a, int first, int last) {

int l = first;

int r = last;

int mid = a[(l + r) / 2];

while (l < r) {

while (a[l] < mid) l++;

while (a[r] > mid) r--;

if (l <= r) {

swap(a[l], a[r]);

l++;

r--;

}

}

if (first < r) quick\_sort(a, first, r);

if (l < last) quick\_sort(a, l, last);

}

bool line\_search(vector<int>& a, int x) { // функция для линейного поиска

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

if (a[i] == x) {

return 1;

}

}

return 0;

}

bool bin\_search(vector<int>& b, int l, int r, int x) { // функция для бинарного поиска

int mid;

while (l <= r) {

mid = (l + r) / 2;

if (b[mid] > x) {

r = mid - 1;

}

else if (b[mid] < x) {

l = mid + 1;

}

else {

return 1;

}

}

return 0;

}

bool interpolation\_search(vector<int>& c, int x) { // функция для интерполяционного поиска

int l = 0;

int r = c.size() - 1;

int mid;

while (c[l] <= x && c[r] >= x) {

mid = l + ((x - c[l]) \* (r - l)) / (c[r] - c[l]);

if (c[mid] > x) {

r = mid - 1;

}

else if (c[mid] < x) {

l = mid + 1;

}

else {

return 1;

}

}

return 0;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cin >> n;

vector<int> a(n);

vector<int> b(n);

vector<int> c(n);

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

a[i] = b[i] = c[i] = rand() % 1000000 + 1;

}

quick\_sort(b, 0, n - 1);

quick\_sort(c, 0, n - 1);

int x;

cout << "Введите число для поиска \n";

cin >> x;

auto start = chrono::high\_resolution\_clock::now();

int lin = line\_search(a, x);

auto end = chrono::high\_resolution\_clock::now();

auto time = chrono::duration\_cast<std::chrono::nanoseconds>(end - start);

if (!lin) {

cout << "Элемент не найден\n";

}

else {

cout << "Элемент найден \n";

}

cout << "Время линейного поиска = " << time.count() << " наносекунд \n";

start = chrono::high\_resolution\_clock::now();

int bin = bin\_search(b, 0, n - 1, x);

end = chrono::high\_resolution\_clock::now();

time = chrono::duration\_cast<std::chrono::nanoseconds>(end - start);

if (!bin) {

cout << "Элемент не найден\n";

}

else {

cout << "Элемент найден\n";

}

cout << "Время бинарного поиска = " << time.count() << " наносекунд \n";

start = chrono::high\_resolution\_clock::now();

int inter = interpolation\_search(b, x);

end = chrono::high\_resolution\_clock::now();

time = chrono::duration\_cast<std::chrono::nanoseconds>(end - start);

if (!inter) {

cout << "Элемент не найден\n";

}

else {

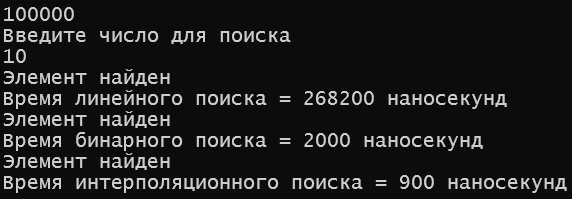
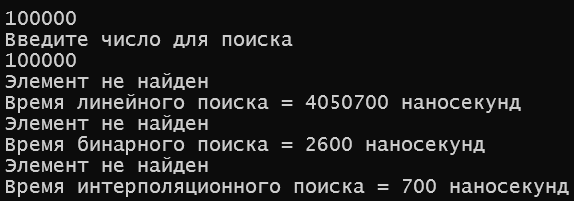
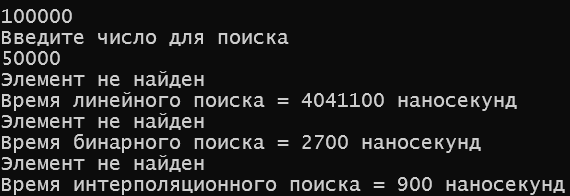
cout << "Элемент найден\n";

}

cout << "Время интерполяционного поиска = " << time.count() << " наносекунд \n";

}

Сравнение времени выполнения:

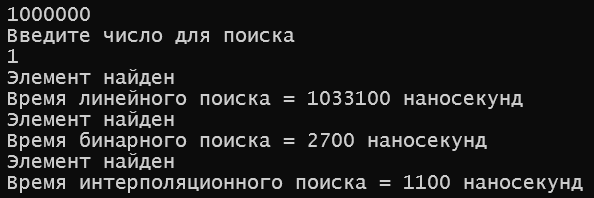
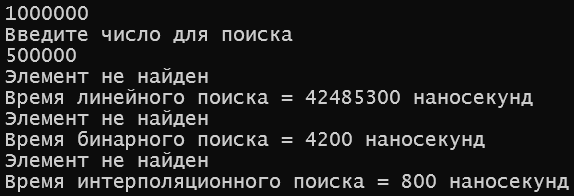
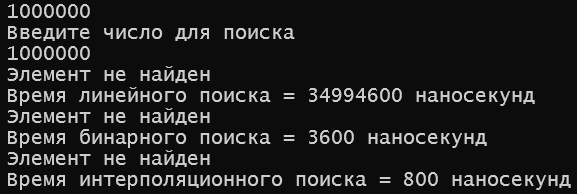
1) 100000 чисел

Среднее время:

Линейный поиск: 2786666,67 наносекунд

Бинарный поиск: 2433,33333 наносекунд

Интерполяционный поиск: 833,333333 наносекунд

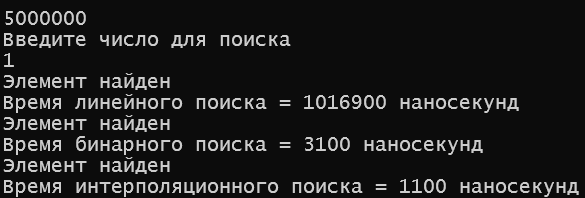
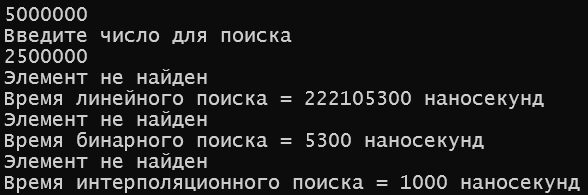
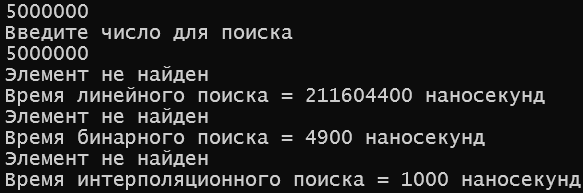
2) 1000000 чисел

Среднее время:

Линейный поиск: 26171000 наносекунд

Бинарный поиск: 3500 наносекунд

Интерполяционный поиск: 900 наносекунд

3) 5000000 чисел

Среднее время:

Линейный поиск: 144908867 наносекунд

Бинарный поиск: 4433,333333 наносекунд

Интерполяционный поиск: 1033,33333 наносекунд

**Выводы**

Наилучшую скорость показал интерполяционный поиск, однако не стоит забывать, что и бинарный, и интерполяционный поиск работают только в отсортированном массиве.

**Плюсы линейного поиска:**

- прост в реализации

- прост для понимания

- работает в любом массиве

**Минусы линейного поиска:**

- очень медленно работает, если массив большой

**Плюсы бинарного поиска:**

- работает значительно быстрее

**Минусы бинарного поиска:**

**-** работает только в отсортированном массиве

**Плюсы интерполяционного поиска:**

- работает ещё быстрее

**Минусы бинарного поиска:**

**-** работает только в отсортированном массиве

- более сложный для понимания