**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ГАБРОВО**

**Факултет:** *ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА*

**Катедра:** *КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ*

**КУРСОВА ЗАДАЧА № 6**

**по дисциплината: СИНТЕЗ И АНАЛИЗ НА АЛГОРИТМИ**

**РАЗРАБОТИЛ**: **Богомил Георгиев Иванов**

/име, презиме, фамилия/

**Бакалавър**

/ОКС/

**Софтуерно и компютърно инженерство**

/специалност/

**ФАК. № 22372126**

**ГАБРОВО ПРОВЕРИЛ**: ………………….

**2025 г. /**ас. инж. Матьо Динев/

1. **Задача**

Да се напише програма реализираща алгоритмите за търсене на елементи в масив от N на брой произволни елемента чрез методите за „ТЪРСЕНЕ ЧРЕЗ СТЪПКА“ и „ИНТЕРПОЛАЦИОННО ТЪРСЕНЕ“. Програмата да разпечата всички елементи от масива с които търсеният елемент се сравнява и при двата метода. Да се направи сравнение на резултатите. Нека да се предостави възможност потребителя да въвежда търсения елемент, броя на елементите на масива N и съответната стъпка.

1. **Теоретична част**

***Търсенето чрез стъпка*** и ***интерполацинното търсене*** се използват в сортирани масиви, но и двете имат различни подходи.

* *Търсене чрез стъпка (Jump Search):*

*Jump Search* комбинира бинарно търсене и линейно търсене. Вместо да проверяваме всеки елемент последователно (както при линейно търсене), алгоритъмът „скача“ с определена стъпка (например ) и търси блока, в който може да се намира търсеният елемент. След като се намери блока, се извършва линейно търсене в него.

*Как работи търсенето чрез стъпка?* Първо избираме подходящата стъпка , където е размерът на масива. Второ – преглежда елементите на всеки позиции. Ако елементът, който търсим, е по-малък от текущия, то тогава трябва да се върне назад и да направи линейно търсене в предишния блок. Ако елементът не се открие, то тогава елемента не съществува в масива.

*Времевата сложност* на алгоритъма в най-добър случай е , докато в най-лош случай – .

* *Интерполационно търсене (Interpolation Search)*

Подходящо за равномерно разпределени числа, *интерполационното търсене* предсказва къде в масива може да се намира елементът, вместо да дели на две, както при бинарното търсене. За това търсене се използва формула, подобна на тази за линейна интерполация. (1)

(1)

*Алгоритъмът за интерполационното търсене* е следния. Докато и :

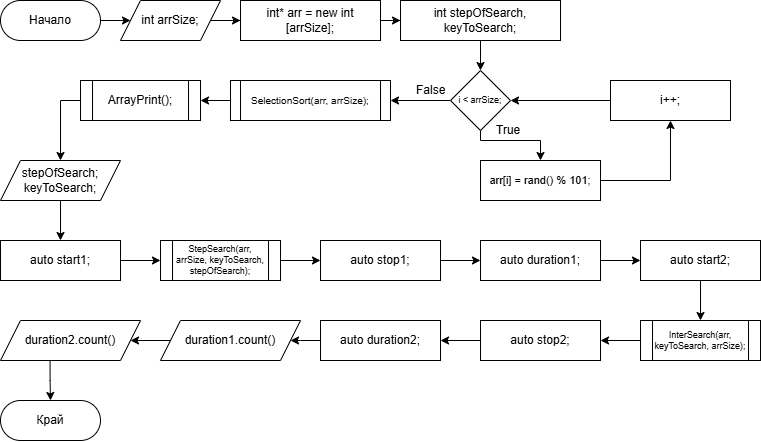
* Изчисли позицията с горната формула; (1)
* Ако → елементът е намерен;
* Ако → търси елементът вдясно;
* Ако → търси елементът вляво.

*Времевата сложност на алгоритъма* в най-добър случай е . Ако е среден случай, то времевата сложност е , а в най-лош случай - . (Особено, ако разпределението е лошо.)

1. **Алгоритъм на решението**
   1. *Таблица на съответствията*

|  |  |
| --- | --- |
| Име на променливата | Описание |
| arrSize | Определя размера на масива, дефиниран от потребителя. |
| arr | Масива, в който ще тестваме двата алгоритъма. Елементите в него се генерират произволно. |
| stepOfSearch | Стъпката, с която търсим при алгоритъма за търсене чрез стъпка. |
| keyToSearch | Стойността, която търсим. |
| leftBorder | Лявата граница. |
| rightBorder | Дясната граница. |
| key | Ключа за сравнение. |
| n | Броя на елементите в масива. |
| step | Стъпката, с която обхождаме масива. |
| min | Задава минимална стойност, с която да се сравняват елементите и да се сортират. |

* 1. *Блок схема*



* 1. *Сорс-код*

#include <iostream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Последователно търсене

int SequenceSearch(int arr[], int key, int n)

{

int i = 0;

while ((arr[i] != key) && (i < n)) i++;

if (arr[i] == key) return 1; // Намерен е елемента

else return 0; // Не е намерен елемента

}

// Търсене на елемент чрез стъпка

int StepSearch(int arr[], int n, int key, int step) {

int i, leftBorder, rightBorder;

for (i = 0; i < n && arr[i] < key; i += step);

if (i < step)

leftBorder = 0;

else

leftBorder = i + 1 - step;

if (n < i)

rightBorder = n - 1;

else

rightBorder = i;

// Функция за последователно търсене

return SequenceSearch(arr,key,n);

}

// Интерполационно търсене на елемент

int InterSearch(int arr[], int key, int n) {

unsigned leftBorder = 0, rightBorder = n - 1 , m;

float k;

while (leftBorder <= rightBorder)

{

if (arr[rightBorder] == arr[leftBorder])

if (arr[leftBorder] == key)

return leftBorder; // Елементът е намерен

else

return 0; // Елементът не е намерен

k = (float)(key - arr[leftBorder]) / (arr[rightBorder] - arr[leftBorder]);

if (k < 0 || k>1)

return 0; m = (unsigned)(leftBorder + k \* (rightBorder - leftBorder) + 0.5);

if (key > arr[m])

rightBorder = m - 1;

else if (key > arr[m])

leftBorder = m + 1;

else return m;

}

return 0;

}

// Функция за сортиране

void SelectionSort(int arr[], int n)

{

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

{

int min = i;

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

{

if (arr[j] < arr[min])

min = j;

}

if (min != i)

swap(arr[i], arr[min]);

}

}

int main()

{

int arrSize;

cout << "Enter the size of the array: " << endl;

cin >> arrSize;

// Елементи

int\* arr = new int [arrSize];

// Променливи и Размер на масива

int stepOfSearch, keyToSearch;

// Генериране на произволни елементи от 0 до 1000

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

arr[i] = rand() % 101;

}

// Сортиране

SelectionSort(arr, arrSize);

// Принтиране на масива (В блок схемата ArrayPrint())

cout << endl << "Array elements: " << endl;

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

{

cout << arr[i] << " ";

}

// Въвеждаме стъпката на търсене

cout << endl << endl << "Enter the step of which the algorithm will be searching: " << endl;

cin >> stepOfSearch;

// Въвеждаме елемента, който търсим

cout << endl << "Now enter the value we need to search." << endl;

cin >> keyToSearch;

// Проследява за колко време ще завърши търсенето

auto start1 = high\_resolution\_clock::now();

int pos1 = StepSearch(arr, arrSize, keyToSearch, stepOfSearch);

auto stop1 = high\_resolution\_clock::now();

auto duration1 = duration\_cast<nanoseconds>(stop1 - start1);

auto start2 = high\_resolution\_clock::now();

int pos2 = InterSearch(arr, keyToSearch, arrSize);

auto stop2 = high\_resolution\_clock::now();

auto duration2 = duration\_cast<nanoseconds>(stop2 - start2);

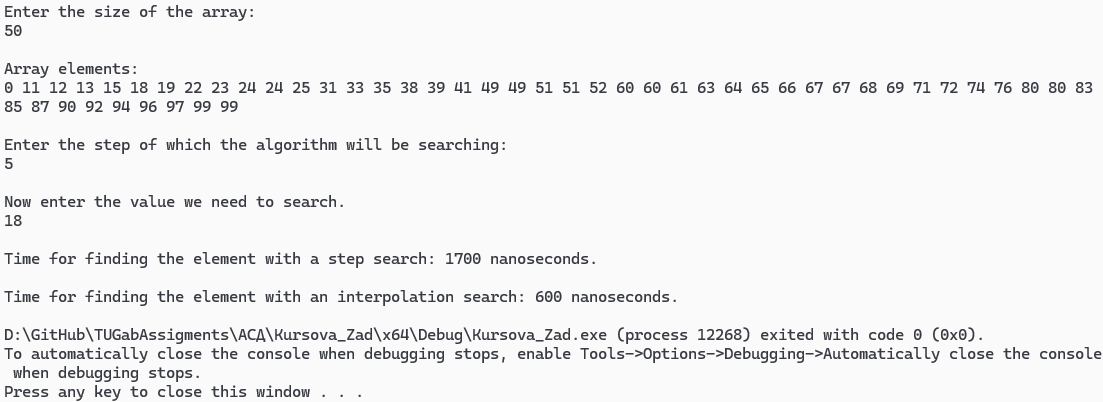
//-------------------------------------------------------

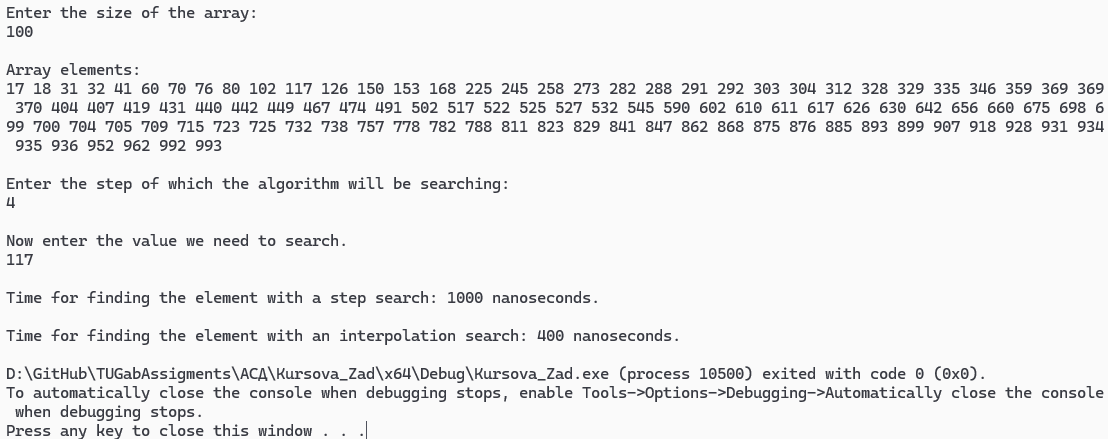
cout << endl << "Time for finding the element with a step search: " << duration1.count() << " nanoseconds." << endl;

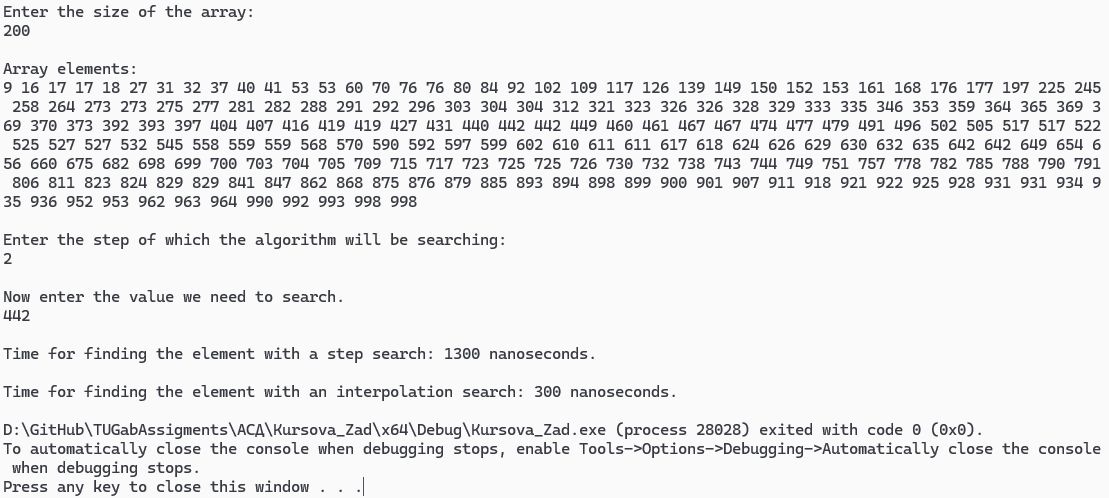
cout << endl << "Time for finding the element with an interpolation search: " << duration2.count() << " nanoseconds." << endl;

}

1. **Резултати от програмата**

**

****

****

1. **Изводи**

Интерполационното търсене е по-бързо, защото извършва по-малко сравнения (), както при съвпадение, така и при несъответствие на търсения елемент, докато при търсенето със стъпка, там ще трябва да премине през няколко сравнения, което е по-бавно изпълним, но е по-надежден.