

Звіт

Лабораторна робота №3

“Дослідження елементарних алгоритмів сортування та пошуку”

Студента группи ДA-12

Краковича Павла Дмитровича

Київ – 2021

* 1. Мета роботи

Набути навичок програмування мовою С/C++. Ознайомитись і дослідити на практиці методи знаходження простих чисел, рекурсію та перевід числа між різними системами числення.

* 2. Варіант роботи - №11
* 3. Хід виконання роботи
  + №1 Базове сортування “бульбашкою”, пошук елементів у масиві

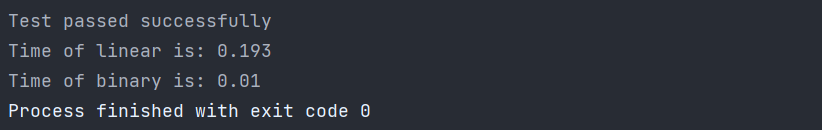
1.1 Створити масив на N = 1000 елементів та заповнити його випадковими цілими числами від -2000 до 2000

1.2 Написати функцію для сортування алгоритмом “бульбашкою”,відсортувати масив за зростанням

1.3 Написати функцію для пошуку елементу в масиві лінійним алгоритмом, яка повертає індекс знайденого елементу або -1, якщо такий елемент не знайдено

1.4 Написати функцію для пошуку елементу в масиві бінарним алгоритмом, яка повертає індекс знайденого елементу або -1, якщо такий елемент не знайдено

1.5 Порівняти швидкодію алгоритмів (шукати у масиві випадкові числа від -2000 до 2000), протестувати правильність.

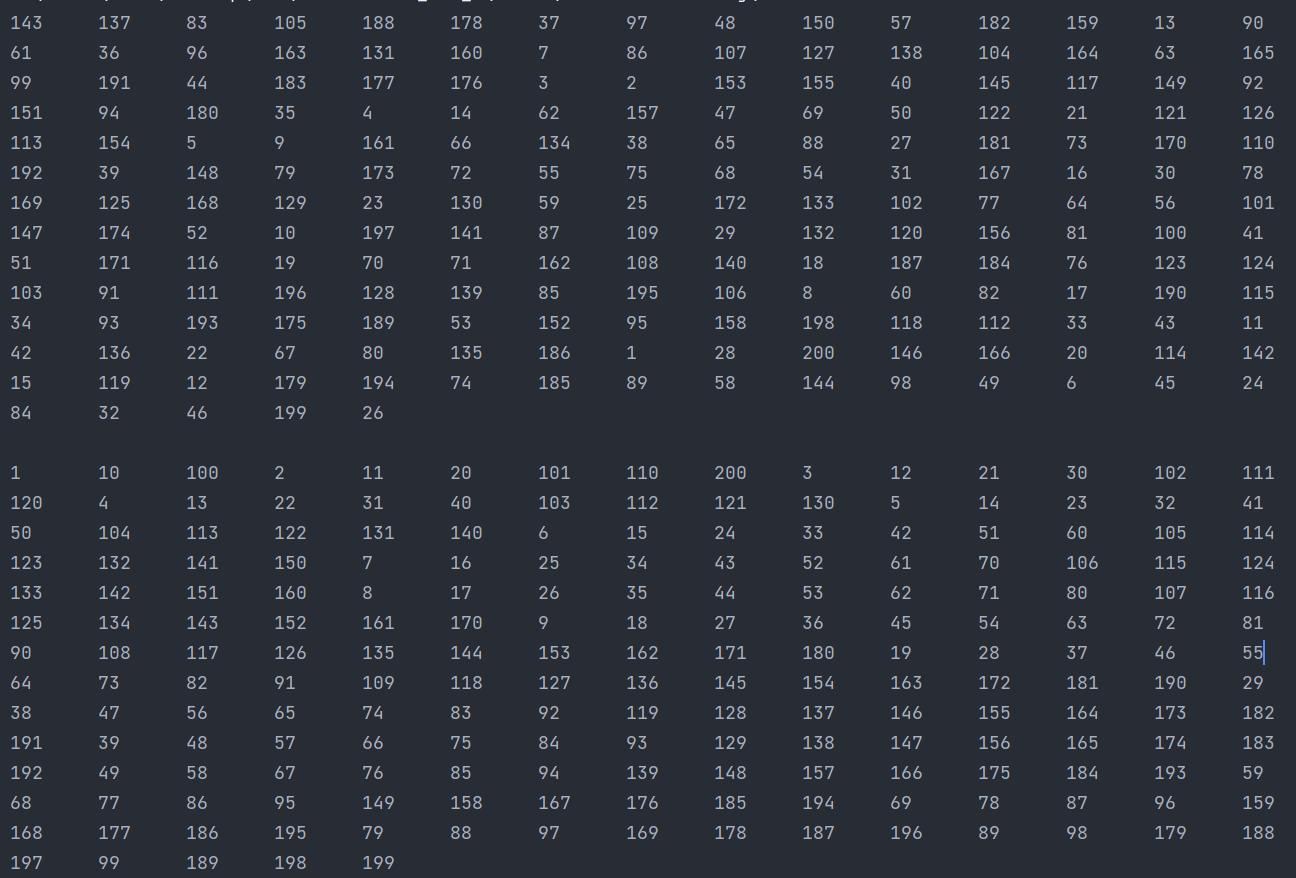
#include<iostream>  
#include<cstdlib>  
#include<ctime>  
using namespace std;  
  
void bubbleArray(int size, int\* arr) {  
 bool isSorted = true;  
 for(int i = 1; i < size; i++)  
 {  
 for(int j = 0; j < size-i; j++)  
 {  
 if(arr[j] > arr[j+1]) {  
 isSorted=false;  
 int temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 }  
 }  
 if (isSorted) {  
 return;  
 }  
 }  
}  
  
int linearSearch(int element,int\* arr, int size) {  
 for (int i=0; i<size; i++)  
 {  
 if (element==arr[i]) {  
 return i;  
 }  
 }  
 return -1;  
}  
  
int binarySearch(int element,int\* arr, int size) {  
 int leftBorder = 0, rightBorder = size - 1;  
 while (rightBorder - leftBorder > 1) {  
 int middleElement = (leftBorder + rightBorder) / 2;  
  
 if (arr[middleElement] < element) {  
 leftBorder = middleElement;  
 } else if (arr[middleElement] > element) {  
 rightBorder = middleElement;  
 } else {  
 return middleElement;  
 }  
 }  
 return -1;  
}  
  
int main() {  
 int size=1000,iteration=0,repeats=100000;  
 int arr[size];  
 int element = rand() % 4000 - 1999;  
 srand ( time(NULL) );  
 for (int i=0; i<size; i++) {  
 arr[i]=rand() % 4000 - 1999;  
 }  
 bubbleArray(size, arr);  
 if (arr[linearSearch(element, arr, size)] == arr[binarySearch(element, arr, size)] || (linearSearch(element, arr, size) == -1 && binarySearch(element, arr, size) == -1)) {  
 cout << "Test passed successfully";  
 } else {  
 cout << "ERROR";  
 cout << "\nlinear index is: " << linearSearch(element, arr, size) << "\nbinary index is: " << binarySearch(element, arr, size);  
 }  
  
 clock\_t startTimeLinear = clock();  
 while (iteration < repeats) {  
 linearSearch(element, arr, size);  
 iteration += 1;  
 }  
 clock\_t endTimeLinear = clock();  
 double secondsLinear = (double(endTimeLinear - startTimeLinear)) / CLOCKS\_PER\_SEC;  
 cout << "\nTime of linear is: " << secondsLinear;  
  
 iteration = 0;  
 clock\_t startTimeBinary = clock();  
 while (iteration < repeats) {  
 binarySearch(element, arr, size);  
 iteration += 1;  
 }  
 clock\_t endTimeBinary = clock();  
 double secondsBinary = (double(endTimeBinary - startTimeBinary)) / CLOCKS\_PER\_SEC;  
 cout << "\nTime of binary is: " << secondsBinary;  
 return 0;  
}

* + №2 Сортування елементарними алгоритмами по різним критеріям

2.1 Створити масив на N = 200 елементів та заповнити його послідовними числами від 1 до N

2.2 Написати функцію для перетасовки масиву алгоритмом Фішера-Йєтса, перетасувати масив.

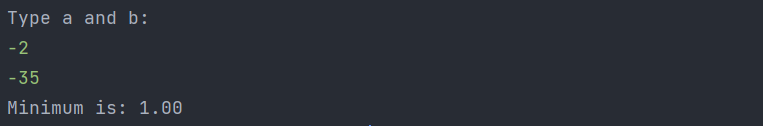
2.3 Сортування вставками за критерієм суми цифр числа у порядку зростання (напр. 61 < 53, адже 7 < 8), при рівності - спочатку менше за значенням.



#include <iostream>  
#include <cstdlib>  
#include <ctime>  
using namespace std;  
  
void randomPermutation(int size,int \*arr) {  
 srand(time(NULL));  
 for(int temp,j,i=size-1;i>=0;i--) {  
 j=rand() % 199;  
 temp=arr[i];  
 arr[i]=arr[j];  
 arr[j]=temp;  
 }  
}  
  
int sumFunction(int currentElement) {  
 int sum=0,rank=currentElement;  
 int i=0;  
 while (i<rank) {  
 sum += currentElement % 10;  
 currentElement /= 10;  
 rank /= 10;  
 }  
 return sum;  
}  
  
  
void sortingFunction(int \*arr,int size) {  
 for(int i=1;i<size;i++) {  
 for(int temp,j=i-1;j>=0 && sumFunction(arr[j])>=sumFunction(arr[j+1]) ;j--) {  
 if(sumFunction(arr[j])!=sumFunction(arr[j+1]) || arr[j]>arr[j+1]) {  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 }  
 }  
 }  
}  
int main() {  
 const int size=200;  
 int arr[size];  
 int i=1,j=0,k=0;  
 while (i<=size) {  
 arr[i-1]=i;  
 i++;  
 }  
 randomPermutation(size,arr);  
 while (j<size) {  
 cout<<arr[j]<<"\t";  
 j++;  
 }  
 sortingFunction(arr,size);  
 cout<<endl<<endl;  
 while (k<size) {  
 cout<<arr[k]<<"\t";  
 k++;  
 }return 0;  
}

* + №3
    - Реалізувати алгоритм тернарного пошуку або золотого перетину для знаходження мінімуму мат. функції

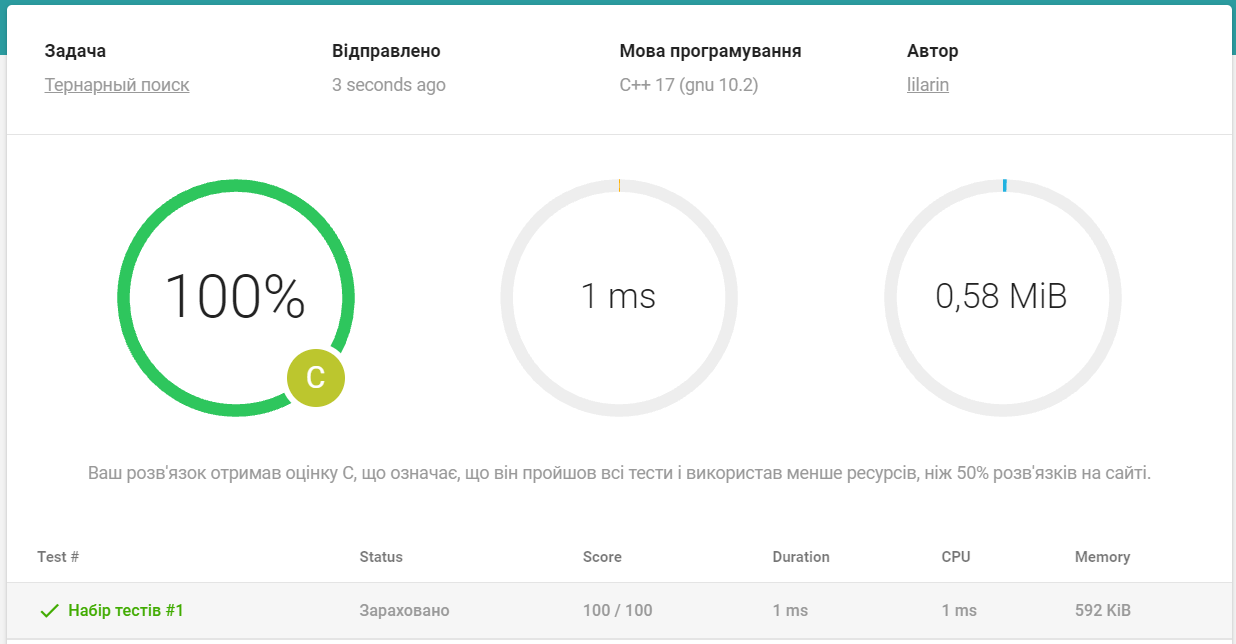
f(x) = x2 + ax + b на проміжку [-100, 100] (змінні a, b вводяться з клавіатури).



#include <iostream>  
using namespace std;  
double Function(double a, double b, double x)  
{  
 double result = x \* x + a \* x + b;  
 return result;  
}  
double TernarySearch(double a, double b,double leftBorder,double rightBorder)  
{  
 while (rightBorder - leftBorder > 0.00001)  
 {  
 double left = leftBorder + (rightBorder - leftBorder) / 3;  
 double right = rightBorder - (rightBorder - leftBorder) / 3;  
 if (Function(a, b, left) < Function(a, b, right)) {  
 rightBorder = right;  
 }  
 else {  
 leftBorder = left;  
 }  
 }  
 return (rightBorder + leftBorder) / 2; }  
int main() {  
 double a, b,leftBorder=-100,rightBorder=100;  
 cin >> a >> b; cout << "" << fixed << setprecision(2)

<< TernarySearch(a,b,leftBorder,rightBorder);

return 0; }



Висновки

Я Ознайомився та дослідив алгоритми сортування: “бульбашкою”, вибором, вставками; алгоритм перетасовки Фішера-Йєтса, алгоритми пошуку: лінійний, бінарний та тернарний. Набув навичок їх реалізації на мові програмування С/C++.

Провівши аналіз асимптотичної складності алгоритмів лінійного та бінарного пошуків я зрозумів, що перший краще використовувати для невеликих масивів, а другий – для великих масивів.

Також я перевірив онлайн код та дізнався, що він може бути біль оптимізованим.