**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ

|  |  |  |
| --- | --- | --- |
| Студент гр. 2373 |  | Андреев Н. С. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение одномерных статических массивов. Изучение алгоритмов сортировки и поиска.

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, константы и инициализатор), что и для простых переменных.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент.

Быстрая сортировка (quick sort) – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки. Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден.

**Постановка задачи.**

Необходимо написать программу, которая:

1) Создает целочисленный массив размерности *N* =100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2) Сортирует заданный в п. 1 массив сортировкой (от меньшего к большему) и определяет время сортировки, используя библиотеку chrono.

3) Находит максимальный и минимальный элемент массива. Считает время поиска этих элементов в отсортированном массиве и неотсортированном.

4) Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.

5) Выводит количество элементов в отсортированном массиве, меньших числа *a*, которое инициализируется пользователем.

6) Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7) Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализует алгоритм бинарного поиска. Сравнивает скорость его работы с обычным перебором.

8) Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

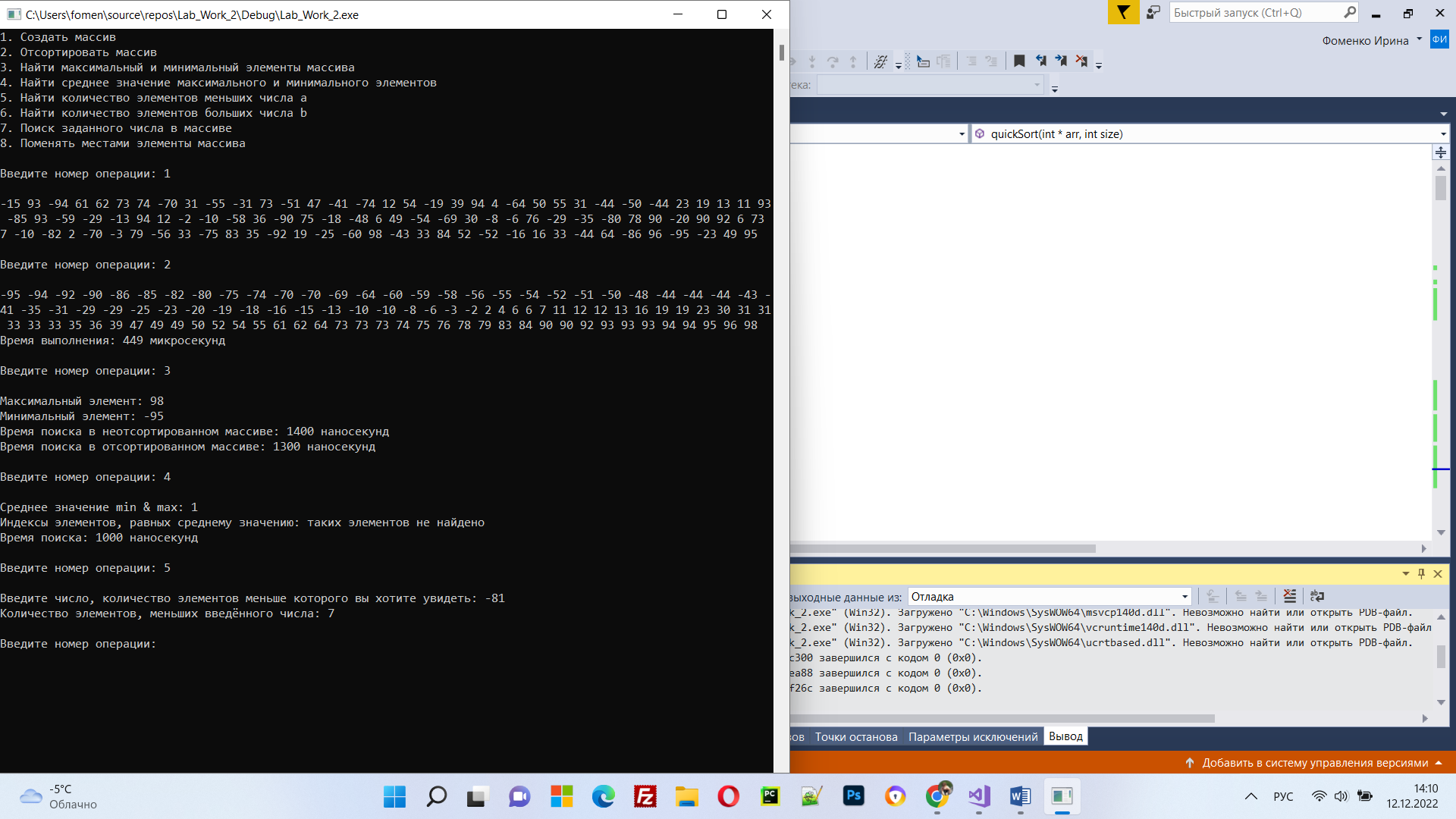
**Выполнение работы.**

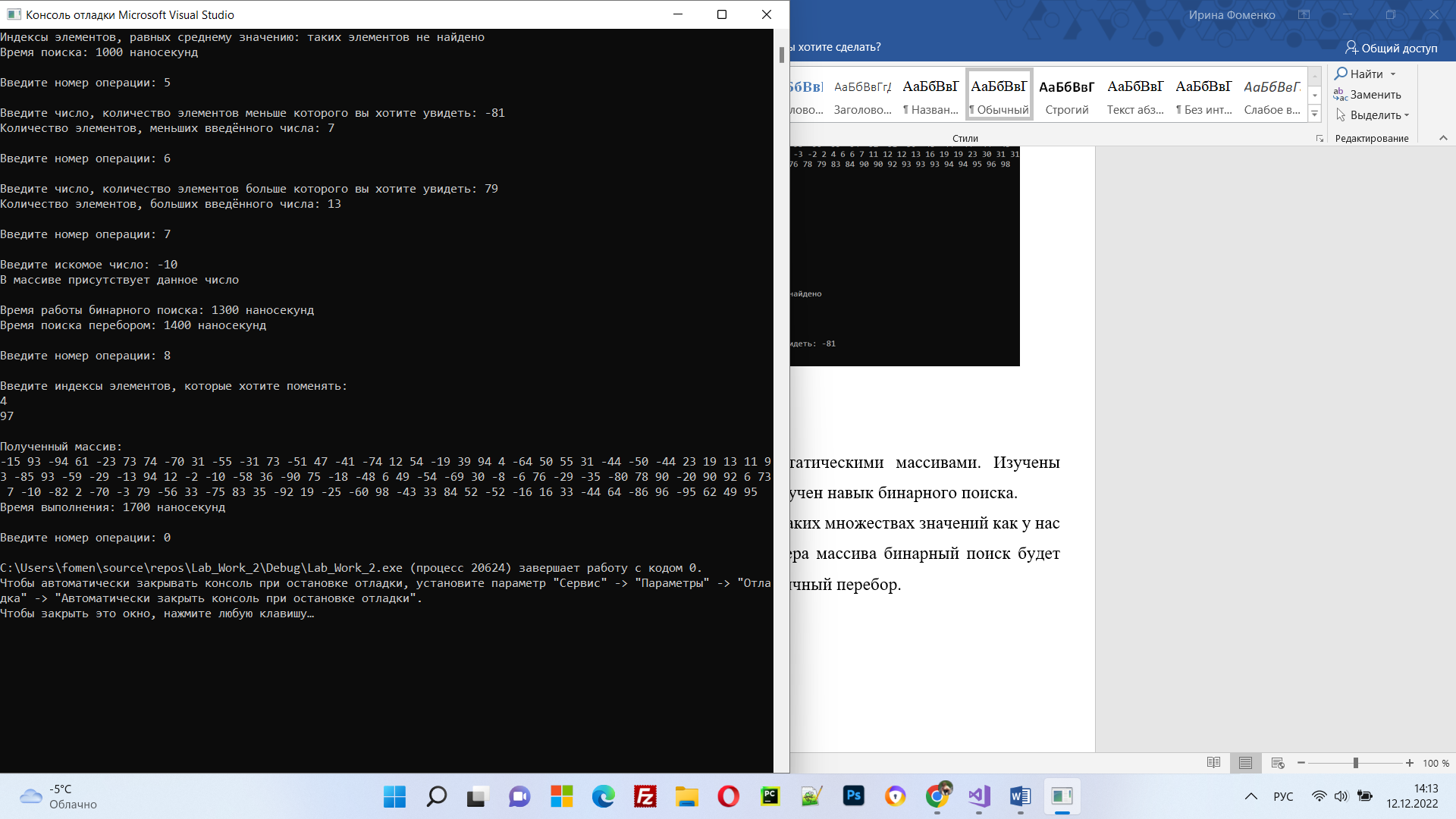
Код программы представлен в приложении А.

Блок описания кода и использованных алгоритмов

В окно вывода сразу прописывается информация о всех возможных операциях. Создание массива происходит после ввода числа 1, то есть только после выполнения первого пункта. Для выполнения второго пункта реализован алгоритм быстрой сортировки. Для выбора опорного элемента написан алгоритм подсчёта среднего значения и поиска среди значений массива, ближайшего к среднему. Для измерения времени выполнения той или иной функции используются объекты типа steady\_clock из библиотеки chrono. Для выполнения третьего пункта реализован алгоритм поиска наибольшего и наименьшего значений перебором. Для выполнения четвёртого пункта не реализовывался алгоритм поиска max и min значений и алгоритм из третьего не использовался, т.к. функция из третьего пункта просто выводила найденные значения в консоль, ничего не возвращая. Поэтому программа просто сортирует массив и работает с первым и последним элементами. В пятом и шестом пунктах просто перебором считаем количество элементов больше или меньше введённого значения. В седьмом пункте реализован алгоритм бинарного пункта без рекурсии. В восьмом пункте просто считываем два значения и меняем их местами без использования функции swap, с помощью мусорной переменной, как в любой сортировке. Для прекращения работы программы нужно ввести любое значение, кроме чисел 1 – 8.

Блок скриншотов работы программы





**Вывод.**

Изучен навык работы с одномерными статическими массивами. Изучены базовые алгоритмы сортировок и поиска. Изучен навык бинарного поиска. Экспериментальным образом мы выяснили, что бинарный поиск на множествах малого размера может быть медленнее, но при увеличении размера массива бинарный поиск будет показывать куда большую скорость, чем обычный перебор.

Приложение А

рабочий код

#include <iostream>

#include <ctime>

#include <chrono>

using namespace std;

using namespace chrono;

int \*quickSort(int \*, int);

bool binarySearch(int \*arr, int size, int value);

int main()

{

setlocale(0, "");

const int n = 100;

srand(time(0));

int arr[n], arrSorted[n];

cout << "1. Создать массив\n";

cout << "2. Отсортировать массив\n";

cout << "3. Найти максимальный и минимальный элементы массива\n";

cout << "4. Найти среднее значение максимального и минимального элементов\n";

cout << "5. Найти количество элементов меньших числа a\n";

cout << "6. Найти количество элементов больших числа b\n";

cout << "7. Поиск заданного числа в массиве\n";

cout << "8. Поменять местами элементы массива\n";

Menu:

{

cout << "\n";

cout << "Введите номер операции: ";

}

int input;

cin >> input;

switch (input)

{

case 1:

{

cout << "\n";

srand(time(0));

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

{

arr[i] = rand() % 199 - 99;

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

}

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

arrSorted[i] = arr[i];

cout << "\n";

goto Menu;

}

case 2:

{

cout << "\n";

auto start = chrono::steady\_clock::now();

quickSort(arrSorted, n);

auto end = chrono::steady\_clock::now();

duration<double> sec = end - start;

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

cout << arrSorted[i] << " ";

cout << "\n";

cout << "Время выполнения: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " микросекунд";

cout << "\n";

goto Menu;

}

case 3:

{

cout << "\n";

int max(-100), min(100);

auto start = chrono::steady\_clock::now();

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

{

if (arr[i] > max)

max = arr[i];

if (arr[i] < min)

min = arr[i];

}

auto end = chrono::steady\_clock::now();

cout << "Максимальный элемент: " << max << "\n" << "Минимальный элемент: " << min << "\n" << "Время поиска в неотсортированном массиве: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд" << endl;

max = -100, min = 100;

quickSort(arrSorted, n);

auto startA = chrono::steady\_clock::now();

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

{

if (arrSorted[i] > max)

max = arrSorted[i];

if (arrSorted[i] < min)

min = arrSorted[i];

}

auto endA = chrono::steady\_clock::now();

cout << "Время поиска в отсортированном массиве: " << chrono::duration\_cast<chrono::nanoseconds>(endA - startA).count() << " наносекунд" << endl;

goto Menu;

}

case 4:

{

cout << "\n";

quickSort(arrSorted, n);

int mean = round((arrSorted[0] + arrSorted[99]) / 2);

int count = 0;

cout << "Среднее значение min & max: " << mean << endl;

cout << "Индексы элементов, равных среднему значению: ";

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

{

if (arr[i] == mean)

cout << i << " ";

}

auto start = chrono::steady\_clock::now();

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

{

if (arr[i] == mean)

count++;

}

auto end = chrono::steady\_clock::now();

duration<double> sec = end - start;

if (count == 0)

{

cout << "таких элементов не найдено" << "\n" << "Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд";

cout << "\n";

goto Menu;

}

cout << "\n";

cout << "Количество элементов: " << count << "\n" << "Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд";

cout << "\n";

goto Menu;

}

case 5:

{

int a, countA = 0;

quickSort(arrSorted, n);

cout << "\n" << "Введите число, количество элементов меньше которого вы хотите увидеть: ";

cin >> a;

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

if (arrSorted[i] < a)

countA++;

cout << "Количество элементов, меньших введённого числа: " << countA << endl;

goto Menu;

}

case 6:

{

int b, countB = 0;

quickSort(arrSorted, n);

cout << "\n" << "Введите число, количество элементов больше которого вы хотите увидеть: ";

cin >> b;

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

{

if (arrSorted[i] > b)

countB++;

}

cout << "Количество элементов, больших введённого числа: " << countB << endl;

goto Menu;

}

case 7:

{

int SearchVal;

cout << "\nВведите искомое число: ";

cin >> SearchVal;

quickSort(arrSorted, n);

cout << ((binarySearch(arrSorted, n, SearchVal) ? "В массиве присутствует данное число" : "Такого числа нет")) << endl;

auto start = chrono::steady\_clock::now();

binarySearch(arrSorted, n, SearchVal);

auto end = chrono::steady\_clock::now();

cout << "\nВремя работы бинарного поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд" << endl;

start = chrono::steady\_clock::now();

bool flag = false;

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

{

if (arrSorted[i] == SearchVal)

{

flag = true;

}

}

end = chrono::steady\_clock::now();

cout << "Время поиска перебором: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд" << endl;

goto Menu;

}

case 8:

{

int m, p;

cout << "\nВведите индексы элементов, которые хотите поменять:" << endl;

cin >> m >> p;

while (!(m >= 0 && m < 100 && p >= 0 && p < 100))

{

cout << "\nВы ввели несуществующие индексы. Попробуйте снова: ";

cin >> m >> p;

}

auto start = chrono::steady\_clock::now();

int j = arr[m];

arr[m] = arr[p];

arr[p] = j;

auto end = chrono::steady\_clock::now();

cout << "\nПолученный массив:" << endl;

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

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

cout << "\nВремя выполнения: " << duration\_cast<nanoseconds>(end - start).count() << " наносекунд" << endl;

goto Menu;

}

default:

{

exit;

}

}

}

int \*quickSort(int \*arr, int size)

{

double mean = 0;

for (int i = 0; i < size; ++i) mean += abs(arr[i]);

mean /= size;

int refNum = 0, refNumIndex;

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

{

if (abs(abs(arr[i]) - abs(mean)) < abs(abs(refNum) - abs(mean)))

{

refNum = arr[i];

refNumIndex = i;

}

}

int countOfLower = 0, countOfHigher = 0;

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

{

if (i != refNumIndex)

{

if (arr[i] >= refNum)

countOfHigher++;

else countOfLower++;

}

}

int \*lowerValues = new int[countOfLower];

int \*higherValues = new int[countOfHigher];

int j = 0, k = 0;

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

{

if (i != refNumIndex)

{

if (arr[i] >= refNum)

{

higherValues[j] = arr[i];

j++;

}

else

{

lowerValues[k] = arr[i];

k++;

}

}

}

if (countOfLower > 2)

quickSort(lowerValues, countOfLower);

else if (countOfLower == 2)

{

if (lowerValues[0] > lowerValues[1])

{

int t = lowerValues[0];

lowerValues[0] = lowerValues[1];

lowerValues[1] = t;

}

}

if (countOfHigher > 2)

quickSort(higherValues, countOfHigher);

else if (countOfHigher == 2)

{

if (higherValues[0] > higherValues[1])

{

int t = higherValues[0];

higherValues[0] = higherValues[1];

higherValues[1] = t;

}

}

for (int i = 0; i < countOfLower; ++i) arr[i] = lowerValues[i];

arr[countOfLower] = refNum;

for (int i = countOfLower + 1; i < size; ++i) arr[i] = higherValues[i - countOfLower - 1];

return arr;

}

bool binarySearch(int \*arr, int size, int value)

{

int middleNumIndex = int(ceil(size / 2.0));

int middleNumValue = arr[middleNumIndex - 1];

while (size > 2)

{

if (value < middleNumValue)

{

middleNumIndex -= int(ceil(size / 4.0));

middleNumValue = arr[middleNumIndex - 1];

size = int(ceil(size / 2.0));

}

else if (value > middleNumValue)

{

middleNumIndex += int(ceil(size / 4.0));

middleNumValue = arr[middleNumIndex - 1];

size -= int(ceil(size / 2.0));;

}

else return true;

}

if (arr[middleNumIndex - 1] == value)

return true;

return false;

}