**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Одномерные статические массивы

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 2372 |  | Гечис В.Р. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение одномерных статических массивов; Изучение различных видов сортировок массива; Изучение алгоритма бинарного поиска и взаимодействия с элементами массива по индексам.

**Основные теоретические положения.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

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.

**Обмен местами элементов массива**

Обмен элементов массива осуществляется через буферную переменную либо через функцию swap(a, b).

**Пузырьковая сортировка массива (bubble sort)**

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.

Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться *N*-1 элемент.

**Шейкер-сортировка массива (shaker sort)**

Shaker sort – модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.

Затем процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

**Сортировка массива расчёской (comb sort)**

Очевидный недостаток bubble и shaker sort заключается в том, что элементы переставляются максимум на одну позицию.

Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком. Алгоритм был разработан специально для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. В сортировке расческой переставляются элементы, стоящие на расстоянии.

Оптимально изначально взять расстояние равным длине массива , а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

**Сортировка массива вставками (insert sort)**

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

**Быстрая сортировка массива (quick sort)**

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.

**Бинарный поиск**

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.

Бинарный поиск работает только в том случае, если массив отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка.

**Постановка задачи.**

Необходимо написать единую программу, объединяющую последовательное выполнение всех заданий практической работы.

Задания практической работы:

* 1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.
* 2)    Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.
* 3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.
* 4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.
* 5)    Выводит количество элементов в отсортированном массиве, которые меньше 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. Реализуйте алгоритм бинарного поиска. Сравните 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.**

1. При запуске программы предлагается запустить основной её цикл, либо прератить её выполнение. При запуске основного цикла создаётся массив на 100 элементов и выводится в консоль:

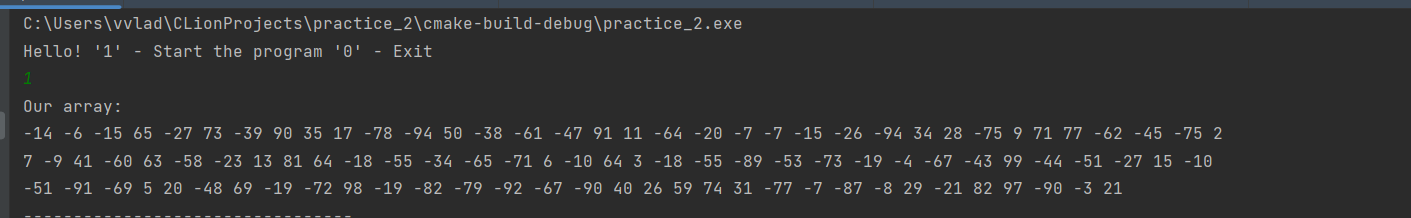


Рис. 1. Начало программы.

1. Далее предлагается выполнить задание 2, отсортировать массив выбранным способом, затем предлагается либо повторить задание с другой, либо той же сортировкой или приступить к следующему шагу.

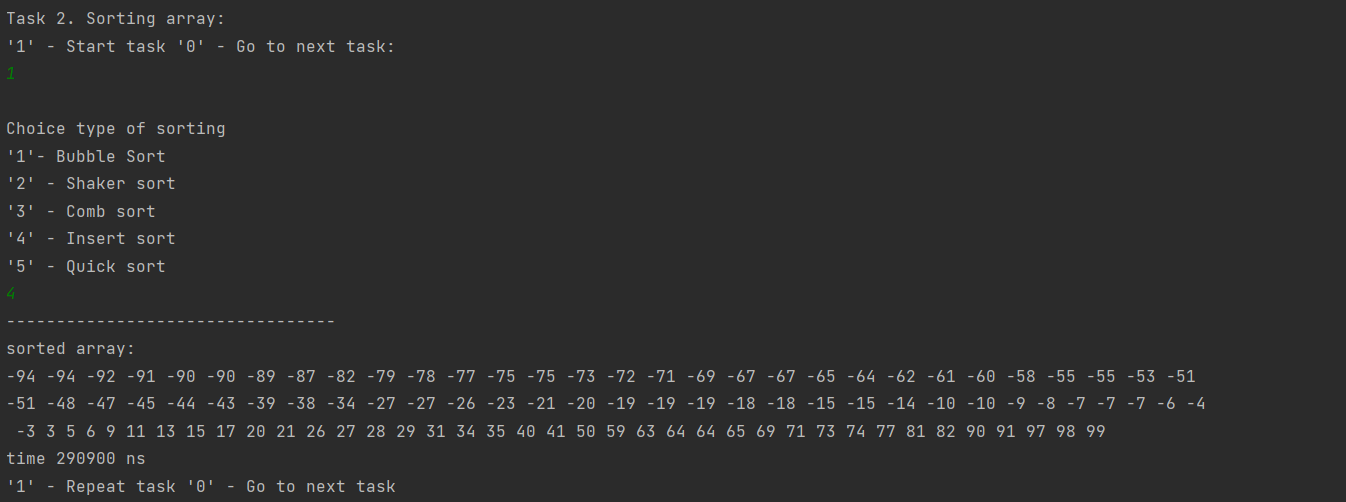


Рис. 2. Создание массива и задание 2.

1. Далее выполняются задания 3 и 4, сначала находится максимальный и минимальный элемент массива, выводится их значение и время поиска, а далее выводится количество элементов массива, равных среднему арифметическому значению максимума и минимума массива.

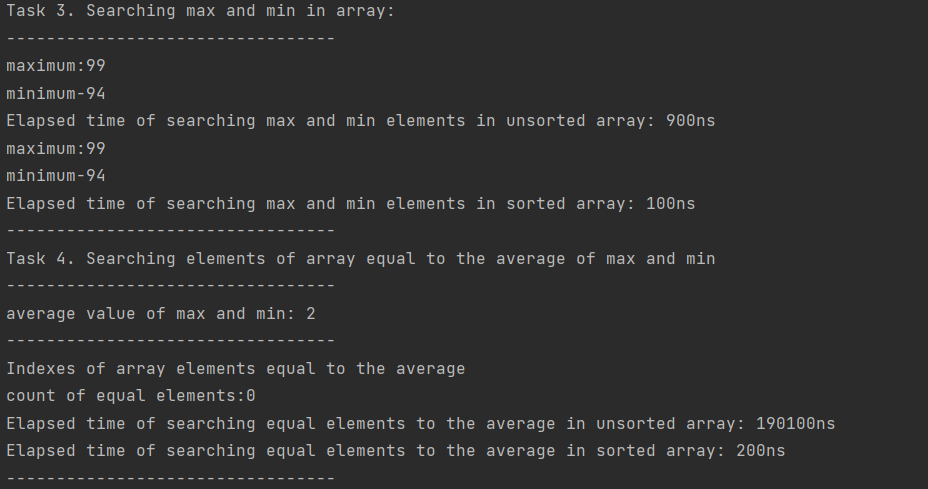


Рис. 3. Выполнение заданий 3 и 4.

1. Далее предлагается к выполнению задания 5 и 6, поиск количества элементов, которые больше/меньше введенных пользователем. Также предлагается повторить выбор и произвести поиск с новыми значениями.

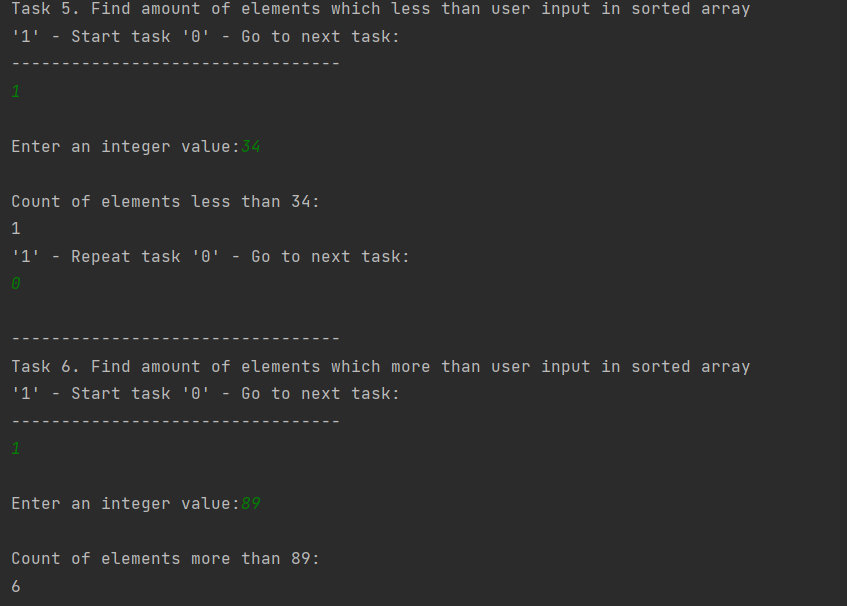


Рис. 4. Выполнение заданий 5 и 6.

1. Далее предлагается к выполнению задание 7, поиск введённого пользователем значения в массиве с помощью линейного и бинарного поисков. Аналогичное предложение повторить выбор для нового значения, либо перейти к следующему заданию.

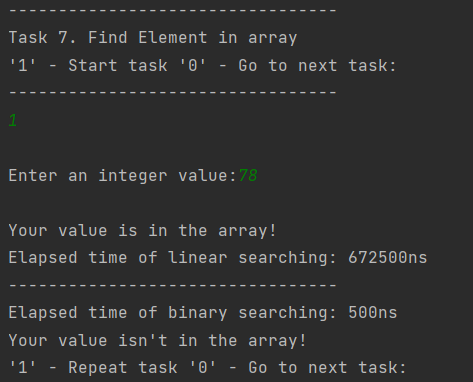


Рис. 5. Выполнение задания 7.

1. Предлагается к выполнению задание 8, необходимо поменять местами два элемента массива, индексы которых вводит пользователь. Для удобства, массив выводится до и после смены, с выделением выбранных элементов.

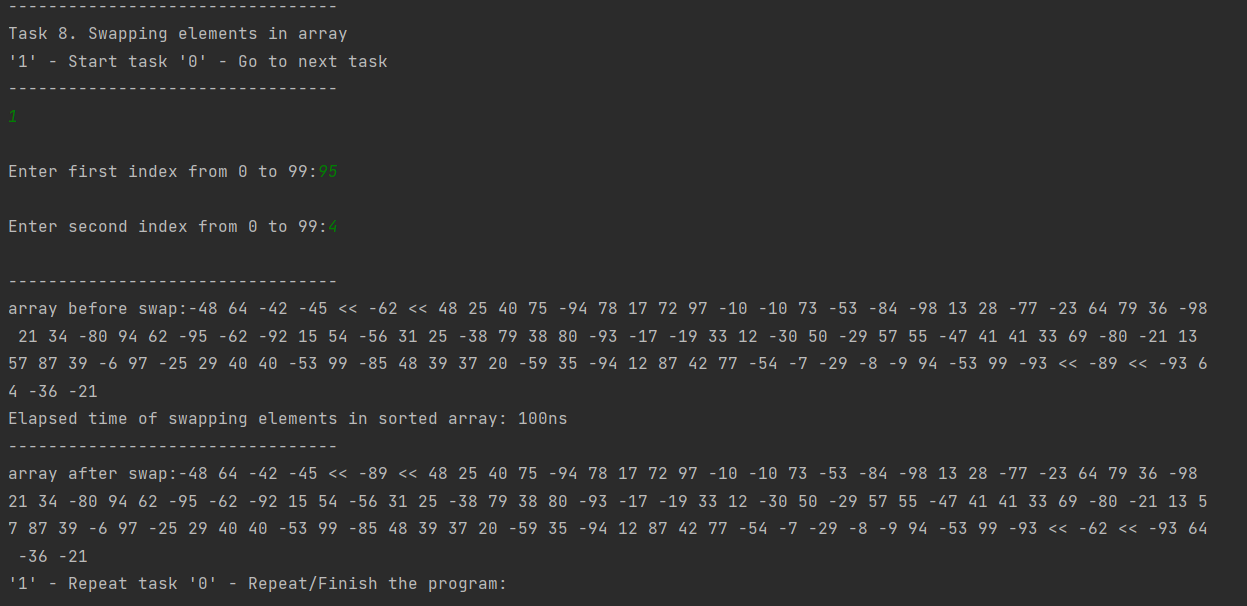


Рис. 6. Выполнение задания 8.

1. После выполнения задания 8, предлагается либо повторить выбор, либо запустить программу сначала/завершить программу

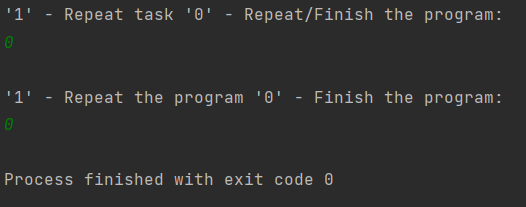


Рис. 7. Завершение программы.

**Выводы.**

В ходе данной лабораторной работы были изучены одномерные массивы, сортировки и бинарный поиск. Написана программа, позволяющая сравнить время выполнения разных видов сортировок, линейного и бинарного поиска.

Приложение А

рабочий код

#include <iostream>  
#include <chrono>  
#include <algorithm>  
#include <ctime>  
using namespace std;  
using namespace chrono;  
  
  
//Сортировки  
//1. Bubble sort  
const int arrSize = 100;  
void bubbleSort(int array[]) {  
 for (int i = 0; i < arrSize - 1; i++)  
 for (int j = 0; j < arrSize - i - 1; j++)  
 if (array[j] > array[j + 1])  
 swap(array[j], array[j + 1]);  
}  
  
void shakerSort(int array[]){  
 for (int i = 0; i<arrSize-1; i++) {  
 for (int j = i; j < arrSize - 1 - i; j++) {  
 if (array[j] > array[j + 1]) {  
 swap(array[j], array[j + 1]);  
 }  
 }  
 for (int j = arrSize - 1 - i; j < 0 + i; j++) {  
 if (array[j] < array[j - 1]) {  
 swap(array[j], array[j - 1]);  
 }  
 }  
 }  
}  
  
void combSort(int array[]){  
 double S = arrSize - 1, k = 1.247;  
 int count = 0;  
 while (S >= 1) {  
 for (int i = 0; i + S < arrSize; i++) {  
 if (array[i] > array[int(i + S)]) swap(array[i], array[int(i + S)]);  
 }  
 S /= k;  
 }  
 for (int j = 0; j < arrSize-1; j++) {  
  
 if (array[j] > array[j + 1]) {  
 swap(array[j], array[j + 1]);  
 }  
 }  
  
}  
  
void insertSort(int array[]){  
 for (int i = 1; i < arrSize; i++) {  
 for (int j = i - 1; j >= 0; j--) {  
 if (array[i] < array[j]) {  
 swap(array[i], array[j]);  
 i -=1;  
 }  
 }  
 }  
}  
void printArray(int arr[])  
{  
 int i;  
 for (i = 0; i < 100; i++){  
 cout << arr[i]<< ' ';  
 }  
  
}  
  
  
  
int main() {  
// Task 1. Filling array random numbers from -99 to 99  
 const int arrSize = 100;  
 int arr[arrSize];  
 int arrSign;  
 int arrForSort[arrSize];  
 srand(time(0));  
 int cycleMain;  
 cout <<"Hello! '1' - Start the program '0' - Exit\n";  
 while (!(cin >> cycleMain)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 while(cycleMain) {  
  
 for (int i = 0; i < arrSize; i++) {  
 arr[i] = -99 + (std::rand() % 199);  
  
// arrSign = rand()%2;  
// arr[i] = rand() % 100;  
// if (arrSign ==1){  
// arr[i] \*= -1;  
// }  
  
 }  
 memcpy(arrForSort, arr, sizeof(arrForSort));  
 cout << "Our array:\n";  
 printArray(arr);  
 cout << "\n---------------------------------";  
 //Task 2. Sorting array:  
  
 cout << "\nTask 2. Sorting array:\n'1' - Start task '0' - Go to next task:\n";  
 int cycleSort;  
 while (!(cin >> cycleSort)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 while (cycleSort) {  
 cout  
 << "\nChoice type of sorting\n'1'- Bubble Sort\n'2' - Shaker sort\n'3' - Comb sort\n'4' - Insert sort\n'5' - Quick sort\n";  
 int userChoiceSort;  
 while (!(cin >> userChoiceSort)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 if (userChoiceSort == 1) {  
 cout << "---------------------------------";  
 cout << "\nsorted array:\n";  
  
 auto start = steady\_clock::now();  
  
 bubbleSort(arrForSort);  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 printArray(arrForSort);  
 cout << "\ntime " << result.count() << " ns";  
  
 }  
 if (userChoiceSort == 2) {  
 cout << "---------------------------------";  
 cout << "\nsorted array:\n";  
  
 auto start = steady\_clock::now();  
  
 shakerSort(arrForSort);  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 printArray(arrForSort);  
 cout << "\ntime " << result.count() << " ns";  
 }  
  
 if (userChoiceSort == 3) {  
 cout << "---------------------------------";  
 cout << "\nsorted array:\n";  
  
 auto start = steady\_clock::now();  
  
 combSort(arrForSort);  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 printArray(arrForSort);  
 cout << "\ntime " << result.count() << " ns";  
  
 }  
 if (userChoiceSort == 4) {  
 cout << "---------------------------------";  
 cout << "\nsorted array:\n";  
  
 auto start = steady\_clock::now();  
  
 insertSort(arrForSort);  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 printArray(arrForSort);  
 cout << "\ntime " << result.count() << " ns";  
 }  
  
 cout << "\n'1' - Repeat task '0' - Go to next task\n";  
 while (!(cin >> cycleSort)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 }  
  
 // Task 3. Searching max and min in array:  
 cout << "\nTask 3. Searching max and min in array:";  
  
  
 //Searching in unsorted array:  
 cout << "\n---------------------------------";  
 auto start = steady\_clock::now();  
 float maxx = -100;  
 float minn = 100;  
 for (int b = 0; b < arrSize; b++) {  
 if (arr[b] > maxx) {  
 maxx = arr[b];  
 }  
 if (arr[b] < minn) {  
 minn = arr[b];  
 }  
 }  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nmaximum:" << maxx << "\nminimum" << minn;  
 cout << "\nElapsed time of searching max and min elements in unsorted array: " << result.count() << "ns";  
  
 //Searching in sorted array:  
 start = steady\_clock::now();  
 maxx = arrForSort[arrSize - 1];  
 minn = arrForSort[0];  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nmaximum:" << maxx << "\nminimum" << minn;  
 cout << "\nElapsed time of searching max and min elements in sorted array: " << result.count() << "ns";  
 cout << "\n---------------------------------";  
  
  
 cout << "\nTask 4. Searching elements of array equal to the average of max and min";  
 cout << "\n---------------------------------";  
  
 //Task 4.1 Average of max and min:  
 int average = (maxx + minn) / 2;  
 cout << "\naverage value of max and min: " << average;  
 cout << "\n---------------------------------";  
 //Task 4.2 Search for array elements equal to the average:  
 //In Unsorted array:  
 cout << "\nIndexes of array elements equal to the average";  
 start = steady\_clock::now();  
  
 int countAverage = 0;  
 for (int i = 0; i < arrSize; i++) {  
 if (arr[i] == average) {  
 cout << "\n" << i;  
 countAverage += 1;  
  
 }  
 }  
 cout << "\ncount of equal elements:" << countAverage;  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nElapsed time of searching equal elements to the average in unsorted array: " << result.count()  
 << "ns";  
  
 //In sorted array:  
 countAverage = 0;  
 start = steady\_clock::now();  
 for (int i = 0; i < arrSize; i++) {  
 if (arrForSort[i] == average) {  
 cout << "\n" << i;  
 countAverage += 1;  
 } else {  
 if (arrForSort[i] > average) {  
 break;  
 }  
 }  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nElapsed time of searching equal elements to the average in sorted array: " << result.count() << "ns";  
 cout << "\n---------------------------------";  
  
 //Task 5 Find amount of elements which less than user input in sorted arra  
 cout << "\nTask 5. Find amount of elements which less than user input in sorted array\n'1' - Start task '0' - Go to next task:\n";  
 cout << "---------------------------------\n";  
  
 int cycleFindLess;  
 int userInputLess;  
 while (!(cin >> cycleFindLess)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 while (cycleFindLess) {  
 cout << "\nEnter an integer value:";  
 while (!(cin >> userInputLess)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 int countLess = 0;  
 for (int i = 0; i < arrSize; i++) {  
 if (arrForSort[i] < userInputLess) {  
 countLess += 1;  
 } else {  
 break;  
 }  
 }  
 cout << "\nCount of elements less than " << userInputLess << ":";  
 cout << "\n" << countLess;  
  
 cout << "\n'1' - Repeat task '0' - Go to next task:\n";  
 while (!(cin >> cycleFindLess)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 }  
 //Task 6 Find amount of elements which more than user input in sorted array  
 cout << "\n---------------------------------";  
 cout << "\nTask 6. Find amount of elements which more than user input in sorted array\n'1' - Start task '0' - Go to next task:\n";  
 cout << "---------------------------------\n";  
 int userInputMore;  
 int cycleFindMore;  
 while (!(cin >> cycleFindMore)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 while (cycleFindMore) {  
 cout << "\nEnter an integer value:";  
 while (!(cin >> userInputMore)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 int countMore = 0;  
 for (int i = 0; i < arrSize; i++) {  
 if (arrForSort[i] > userInputMore) {  
 countMore += 1;  
 }  
 }  
 cout << "\nCount of elements more than " << userInputMore << ":";  
 cout << "\n" << countMore;  
  
 cout << "\n'1' - Repeat task '0' - Go to next task:\n";  
 while (!(cin >> cycleFindMore)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 }  
  
 //Task 7. Find element in array:  
 //7.1 linear searching  
 cout << "\n---------------------------------";  
 cout << "\nTask 7. Find Element in array\n'1' - Start task '0' - Go to next task:\n";  
 cout << "---------------------------------\n";  
 int userKey;  
 int cycleFindKey;  
 while (!(cin >> cycleFindKey)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 while (cycleFindKey) {  
 cout << "\nEnter an integer value:";  
  
 while (!(cin >> userKey)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 start = steady\_clock::now();  
 bool flag = 0;  
 for (int i = 0; i < arrSize; i++) {  
 if (arrForSort[i] == userKey) {  
 cout << "\nYour value is in the array!";  
 flag = 1;  
 break;  
 }  
 }  
 if (!flag) {  
 cout << "\nYour value isn't in the array!";  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nElapsed time of linear searching: " << result.count() << "ns";  
  
 cout << "\n---------------------------------";  
 //7.2 Binary searching:  
 int leftLimit = 0;  
 int rightLimit = arrSize;  
 int middleElement;  
 flag = 0;  
  
 start = steady\_clock::now();  
  
 while (leftLimit <= rightLimit) {  
 middleElement = (leftLimit + rightLimit) / 2;  
 if (userKey == arrForSort[middleElement]) {  
 flag = 1;  
 break;  
 }  
 if (userKey > arrForSort[middleElement]) {  
 leftLimit = middleElement + 1;  
 }  
 if (userKey < arrForSort[middleElement]) {  
 rightLimit = middleElement - 1;  
 }  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nElapsed time of binary searching: " << result.count() << "ns";  
 if (flag) {  
 cout << "\nYour value is in the array!\nIndex of element: " << middleElement;  
 } else {  
 cout << "\nYour value isn't in the array!";  
 }  
 cout << "\n'1' - Repeat task '0' - Go to next task:\n";  
 while (!(cin >> cycleFindKey)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 }  
  
  
  
 //Task 8. Swapping array elements:  
 cout << "\n---------------------------------";  
 cout << "\nTask 8. Swapping elements in array\n'1' - Start task '0' - Go to next task\n";  
 cout << "---------------------------------\n";  
 int cycleSwap;  
 int inputFirstIndex, inputSecondIndex;  
 while (!(cin >> cycleSwap)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 while (cycleSwap) {  
 cout << "\nEnter first index from 0 to 99:";  
 while (!(cin >> inputFirstIndex)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 cout << "\nEnter second index from 0 to 99:";  
 while (!(cin >> inputSecondIndex)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 cout << "\n---------------------------------";  
 cout << "\narray before swap:";  
 for (int i = 0; i < arrSize; i++) {  
 if (i == inputFirstIndex || i == inputSecondIndex) {  
 cout << "<< " << arrForSort[i] << " << ";  
 } else {  
 cout << arrForSort[i] << ' ';  
 }  
 }  
 start = steady\_clock::now();  
 swap(arrForSort[inputFirstIndex], arrForSort[inputSecondIndex]);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\nElapsed time of swapping elements in sorted array: " << result.count() << "ns";  
  
  
 cout << "\n---------------------------------";  
 cout << "\narray after swap:";  
 for (int i = 0; i < arrSize; i++) {  
 if (i == inputFirstIndex || i == inputSecondIndex) {  
 cout << "<< " << arrForSort[i] << " << ";  
 } else {  
 cout << arrForSort[i] << ' ';  
 }  
 }  
 cout << "\n'1' - Repeat task '0' - Repeat/Finish the program:\n";  
 while (!(cin >> cycleSwap)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
 }  
 cout << "\n'1' - Repeat the program '0' - Finish the program:\n";  
 while (!(cin >> cycleMain)) {  
 cout << "error, expecting a num value\n";  
 cin.clear();  
 fflush(stdin);  
 }  
  
 }  
}