**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Информационные Системы»**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: «Одномерные статические массивы»

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Сомова П. Д. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2020

**Цель работы.**

Изучение структуры одномерных массивов, обработка данных одномерных массивов. Изучение различных видов сортировок. Проведение временной оценки различных действий с массивами.

**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. Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Инициализировать массив можно и другим более простым способом: инициализирующие значения записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются:

int arr[4] = {3, 2, 1}; // arr[0] = 3, arr[1] = 2, arr[2] = 1, arr[3] = 0

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размерность должна быть задана целой положительной константой или константным выражением.

*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 сравнивании 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. Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего меньшего). Быстрее же всего алгоритм работает с уже отсортированным массивом. Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

*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 непорядок 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.

Кажется, что bubble sort теряет свою эффективность по сравнению с shaker sort. Сортировка проходит в массиве в обоих направлениях, а не только от его начала к концу. Но в работе с большими массивами преимущество шейкер-сортировки уменьшается как раз из-за использования двух циклов.

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

*Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком*. Алгоритм был разработан для 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 𝑁 − 1, а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

Простая идея и несложная идея порождающее впечатляющий результат. Сортировка расческой работает намного быстрее, чем bubble или shaker sort, в некоторых ситуациях comb sort работает быстрее quick sort. Но данная сортировка обладает одним очевидным минусом – неустойчивость.

*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. 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.

Существует множество модификаций сортировки вставками, некоторые из них затрагивают именно способ вставки элемента в отсортированную часть. Одна из самых лучших модификаций – сортировка простыми вставками с бинарным поиском.

Лучше всего сортировка вставками работает при обработке почти отсортированных массивов. В таком случае insert sort работает быстрее других сортировок.

*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.

На практике массив обычно делят на две части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения.

При том, что это один из самых быстродействующих из алгоритмов, данные алгоритм сортировки неустойчив, а прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека.

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.

Рассмотрим простой пример: имеется массив из 100 элементов, упорядоченных по возрастанию от 1 до 100. Было загадано какое-то число, необходимо его назвать. Компьютер имеет три ответа на ваше предположение: верно, число больше, число меньше. Сколько попыток нужно, чтобы ответить правильно? Обычный перебор – наихудшая стратегия. Можно назвать правильный ответ лишь с 100-ой попытки. Но если начать спрашивать с середины, то ситуация кардинально меняется. Если число больше 50, то необходимо делить правую половину, и следующее предположение – 75, если меньше – 25. Так необходимо продолжать до тех пор, пока не будет названо правильное число. Наибольшее число предположений равняется:

𝑘 = ⌈log2100⌉ = 7.

Этот результат явно лучше простого перебора. Схожий принцип работы имеет алгоритм бинарного поиска.

*Бинарный поиск* работает только в топ случае, если список 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 программу, которая:

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 элементы массива […] 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. Выводит индексы всех элементов, которые равны этому значению, и их количество.
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.**

Для знакомства с структурами одномерных массивов, обработка данных одномерных массивов, изучения различных видов сортировок, проведения временной оценки различных действий с массивами отлично подойдет язык программирования C++. Поэтому была написана программа на C++, которая помогла выполнить поставленную задачу. Итоговый код программы представлен в приложении А. Результаты работы программы представлены в приложении B.

Для удобства работы над задачами были созданы функции. Часть функций отвечает за номер поставленных задач, а часть - вспомогательные функции для упрощения работы алгоритмов.

Были произведены сортировки:

* Bubble sort (пузырьковая сортировка)
* Quick sort (быстрая сортировка)

Также было определенно время, затраченное на сортировку или поиск, используя библиотеку chrono, там где это требовалось

**Выводы.**

На основании проделанной работы, можно сделать вывод, что сортировки реализуются по- разному и они имеет разную эффективность по времени и памяти.

Приложение А

Полный код программы

#include <iostream>  
#include <ctime>  
#include <iomanip>  
#include <chrono>  
#include <cmath>  
  
using namespace std**;**using namespace chrono**;**const int SIZE\_ARRAY=**100;**int globalArray[SIZE\_ARRAY]**;**int\* RandomArray (int\*)**;**void PrintArray (int\*)**;**void DoTask1 ()**;**int\* BubbleSort (int\*)**;**void DoTask2 ()**;**int FindMax (int\*)**;**void FindMaxInRandomArray (int\*)**;**void FindMaxInSortArray (int\*)**;**int FindMin (int\*)**;**void FindMinInRandomArray (int\*)**;**void FindMinInSortArray (int\*)**;**void DoTask3 ()**;**void DoTask4 (int\*)**;**void DoTask5 (int\*)**;**void DoTask6 (int\*)**;**void SwapIndex (int\***,** int**,** int)**;**void DoTask8 ()**;**void DoTask7()**;**int MenuTasks ()**;**void DoTasks (int)**;**void DoLabWork2 ()**;**int\* QuickSort (int\***,** int**,** int)**;**void printQuickSort ()**;**bool NormalSearch (int\***,**int)**;**void TimeNormalSearch (int)**;**bool BinarySearch (int\***,** int)**;**void TimeBinarySearch (int)**;**void PrintNumberSearch (bool)**;**int main() {  
  
 setlocale( LC\_ALL**,** "Russian")**;** DoLabWork2()**;** return **0;**}  
  
/\*  
 \* Заполнение массива. Рандом  
 \*/  
int\* RandomArray (int\* randArr) {  
  
 srand(time(NULL))**;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 randArr[i] = (rand() % **199**) - **99;** }  
 return randArr**;**}  
  
/\*  
 \* Вывод созданного массива  
 \*/  
void PrintArray (int printArr[]) {  
  
 for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 cout << setw(**8**) << left << printArr[i]**;** }  
}  
  
/\*  
 \* Выполнение задачи 1  
 \* Вывод рандомного массива  
 \*/  
void DoTask1 () {  
 cout << "Исходный массив:\n"**;** PrintArray(RandomArray(globalArray))**;** cout << "\n\n"**;**}  
  
/\*  
 \* Bubble sort (пузырьковая сортировка)  
 \*/  
int\* BubbleSort (int\* bubbleSortArr) {  
  
 for (int i=**0;** i<SIZE\_ARRAY-**1;** i++) {  
 for (int j=**0;** j<SIZE\_ARRAY-**1;** j++) {  
 if (bubbleSortArr[j+**1**] < bubbleSortArr[j]){  
 swap(bubbleSortArr[j]**,** bubbleSortArr[j+**1**])**;** }  
 }  
 }  
 return bubbleSortArr**;**}  
  
/\*  
 \* Выполнение задачи 2  
 \* Вывод отсортированного массива + тайминг  
 \*/  
void DoTask2 () {  
 cout << "Отсортированный массив:\n"**;** system\_clock::time\_point start = system\_clock::now()**;** PrintArray(BubbleSort(globalArray))**;** system\_clock::time\_point end = system\_clock:: now()**;** cout << "\n"**;** duration<double> sec = end - start**;** cout << "Время затраченное на сортировку:" << sec.count() << "\n\n"**;** cout << "\n\n"**;**}  
/\*  
 \* Нахождение МАКС в массиве  
 \*/  
int FindMax (int\* findMaxArr) {  
  
 int findMax=-**99;** for (int i=**0;** i<SIZE\_ARRAY-**1;** i++){  
 if (findMaxArr[i]>findMax) {  
 findMax=findMaxArr[i]**;** }  
 }  
 return findMax**;**}  
  
/\*  
 \* Нахождение МАКС в неотсортированном массиве  
 \*/  
void FindMaxInRandomArray (int\* findMaxRandArr) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** cout << "Максимум: " << FindMax(findMaxRandArr) << "\n"**;** system\_clock::time\_point end = system\_clock:: now()**;** duration<double> sec = end - start**;** cout << "Время затраченное на поиск максимума в неотсортированном массиве:" << sec.count() << "\n\n"**;**}  
  
/\*  
 \* Нахождение МАКС в отсортированном массиве  
 \*/  
void FindMaxInSortArray (int\* findMaxSortArr) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** cout << "Максимум: " << FindMax(findMaxSortArr) << "\n"**;** system\_clock::time\_point end = system\_clock:: now()**;** duration<double> sec = end - start**;** cout << "Время затраченное на поиск максимума в отсортированном массиве:" << sec.count() << "\n\n"**;**}  
  
/\*  
 \* Нахождение МИН  
 \*/  
int FindMin (int\* findMinArr) {  
  
 int findMin=**99;** for (int i=**0;** i<SIZE\_ARRAY-**1;** i++){  
 if (findMinArr[i]<findMin) {  
 findMin=findMinArr[i]**;** }  
 }  
 return findMin**;**}  
  
/\*  
 \* Нахождение МИН в неотсортированном массиве  
 \*/  
void FindMinInRandomArray (int\* findMinRandArr) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** cout << "Минимум: " << FindMin(findMinRandArr) << "\n"**;** system\_clock::time\_point end = system\_clock:: now()**;** duration<double> sec = end - start**;** cout << "Время затраченное на поиск минимума в неотсортированном массиве:" << sec.count() << "\n\n"**;**}  
  
/\*  
 \* Нахождение МИН в отсортированном массиве  
 \*/  
void FindMinInSortArray (int\* findMinSortArr) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** cout << "Минимум: " << FindMin(findMinSortArr) << "\n"**;** system\_clock::time\_point end = system\_clock:: now()**;** duration<double> sec = end - start**;** cout << "Время затраченное на поиск минимума в отсортированном массиве:" << sec.count() << "\n\n"**;**}  
  
/\*  
 \* Выполнение задачи 3  
 \* Поиск МАКС и МИН в неотсортированном и отсортированном массиве  
 \*/  
void DoTask3 () {  
  
 cout << "\*\*\*Задание 3\*\*\*\n\n"**;** cout << "Поиск max и min в неотсортированном массиве: \n"**;** FindMaxInRandomArray(RandomArray(globalArray))**;** FindMinInRandomArray(RandomArray(globalArray))**;** cout << "\n\n"**;** cout << "Поиск max и min в отсортированном массиве: \n"**;** FindMaxInSortArray(BubbleSort(globalArray))**;** FindMinInSortArray(BubbleSort(globalArray))**;** cout << "\*\*\*Конец Задание 3\*\*\*\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \* Выполнение задачи 4  
 \* Среднее знач МАКС и МИН  
 \* Индексы элементов, значения которых равны среднему значению МАКС и МИН  
 \*/  
void DoTask4 (int\* meanMaxMinArr) {  
  
 cout << "\*\*\*Задание 4\*\*\*\n\n"**;** int midMaxMin = round((FindMax(meanMaxMinArr)+FindMin(meanMaxMinArr))/**2**)**;** cout << "Среднее значение Max и Min: " << midMaxMin << "\n"**;** int countMean = **0;** cout << "Индексы элементов, значения которых равны среднему значению Max и Min: \n"**;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 if (meanMaxMinArr[i]==midMaxMin) {  
 countMean++**;** cout << i << " "**;** }  
 }  
 cout << "Количество элементов, значения которых равны среднему значению Max и Min: " << countMean << "\n\n"**;** cout << "\*\*\*Конец Задание 4\*\*\*\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \* Выполнение задачи 5  
 \* Элементы меньше введенного числа  
 \*/  
void DoTask5 (int\* lessThanArr) {  
  
 cout << "\*\*\*Задание 5\*\*\*\n\n"**;** int lessThan**;** cout << "Введите число a: "**;** cin >> lessThan**;** int countLessThan=**0;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 if (lessThanArr[i]<lessThan) {  
 countLessThan++**;** }  
 }  
  
 cout << "Количество элементов в отсортированном массиве, которые меньше числа a: " << countLessThan <<"\n\n"**;** cout << "\*\*\*Конец Задание 5\*\*\*\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \* Выполнение задачи 6  
 \* Элементы больше введенного числа  
 \*/  
void DoTask6 (int\* moreThanArr) {  
  
 cout << "\*\*\*Задание 6\*\*\*\n\n"**;** int moreThan**;** cout << "Введите число b: "**;** cin >> moreThan**;** int countMoreThan=**0;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 if (moreThanArr[i]>moreThan) {  
 countMoreThan++**;** }  
 }  
  
 cout << "Количество элементов в отсортированном массиве, которые меньше числа a: " << countMoreThan <<"\n\n"**;** cout << "\*\*\*Конец Задание 6\*\*\*\n\n"**;** cout << "\n\n"**;**}  
/\*  
 \* Обмен месатми элементов, индексы которых введены  
 \*/  
void SwapIndex (int\* swapIndexArr**,** int index1Copy**,** int index2Copy) {  
  
 int t**;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 if (i == index1Copy) {  
 t=swapIndexArr[i]**;** swapIndexArr[i]=swapIndexArr[index2Copy]**;** swapIndexArr[index2Copy]=t**;** }  
 }  
 PrintArray(swapIndexArr)**;**}  
/\*  
 \* Выполнение задачи 8  
 \* Ввывод обмена месатми элементов, индексы которых введены, в отсортированном массиве + тайминг  
 \*/  
void DoTask8 () {  
  
 cout << "\*\*\*Задание 8\*\*\*\n\n"**;** int index1**;** int index2**;** cout << "Введите индексы элементов массива, которые хотите поменять местами: "**;** cin >> index1 >> index2**;** system\_clock::time\_point start = system\_clock::now()**;** SwapIndex(BubbleSort(globalArray)**,** index1**,** index2)**;** system\_clock::time\_point end = system\_clock:: now()**;** duration<double> sec = end - start**;** cout << "\n"**;** cout << "Скорость обмена:" << sec.count() << "\n\n"**;** cout << "\*\*\*Конец Задание 8\*\*\*\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \* Задачи Лабораторной работы 2  
 \*/  
int MenuTasks () {  
  
 int numberTask**;** cout << "\n\n"**;** cout << "\*\*\*\n"**;** cout << "Введите номер задачи:\n"**;** cout << "1) Создать рандомно массив: \n"**;** cout << "2) Отсортировать заданный массив: \n"**;** cout << "3) Вывод максимального и минимального элемента массива\n"**;** cout << "4) Вывод среднего значения максимального и минимального элемента массива \n"**;** cout << "5) Количество, элементов, меньше числа a\n"**;** cout << "6) Количество, элементов, больше числа b\n"**;** cout << "7) Вывод информации о том, есть ли заданное число в массиве\n"**;** cout << "8) Поменять местами элементы, индексы которых заданы\n"**;** while (true) {  
  
 cin >> numberTask**;** if ( (numberTask > **0**) && (numberTask <= **8**) )  
 break**;** else  
 cout << "Error: Неверно введены данные \n"**;** }  
 cout << "\n"**;** return numberTask**;**}  
/\*  
 \* Выбор выполнения задач лаборатной работы 2  
 \*/  
void DoTasks (int option) {  
  
 switch (option) {  
 case **1**: {  
 DoTask1()**;** break**;** }  
 case **2**: {  
 DoTask2()**;** break**;** }  
 case **3**: {  
 DoTask3()**;** break**;** }  
 case **4**: {  
 DoTask4 (BubbleSort(globalArray))**;** break**;** }  
 case **5**: {  
 DoTask5 (BubbleSort(globalArray))**;** break**;** }  
 case **6**: {  
 DoTask6 (BubbleSort(globalArray))**;** break**;** }  
 case **7**: {  
 DoTask7()**;** break**;** }  
 case **8**: {  
 DoTask8()**;** break**;** }  
 }  
}  
  
/\*  
 \* Выполнение лаборатоной работы 2  
 \*/  
void DoLabWork2 () {  
  
 while (true) {  
 DoTasks(MenuTasks())**;** }  
}  
  
/\*  
 \* Выполнение задачи 7  
 \* Есть ли введенное пользователем число в массиве  
 \* Бинарный поиск и обычный поиск с таймингами  
 \*/  
void DoTask7 () {  
 printQuickSort()**;** int num**;** cout << "Введите число, которое хотите найти в массиве: "**;** cin >> num**;** TimeBinarySearch(num)**;** TimeNormalSearch (num)**;**}  
  
/\*  
 \* Quick sort (быстрая сортировка)  
 \*/  
int\* QuickSort (int\* quickSortArr**,** int end**,** int begin) {  
  
 int mid**;** int f = begin**;** int l = end**;** mid = quickSortArr[(f + l )/ **2**]**;** while (f < l)  
 {  
 while (quickSortArr[f] < mid) f++**;** while (quickSortArr[l]> mid) l--**;** if (f <= l)  
 {  
 swap(quickSortArr[f]**,** quickSortArr[l])**;** f++**;** l--**;** }  
 }  
 if (begin < l) QuickSort(quickSortArr**,** l**,** begin)**;** if (f < end) QuickSort(quickSortArr**,** end**,** f)**;** return quickSortArr**;**}  
  
/\*  
 \* Вывод массива после Quick sort + тайминг  
 \*/  
void printQuickSort () {  
  
 cout << "Отсортированный массив:\n"**;** system\_clock::time\_point start = system\_clock::now()**;** PrintArray(BubbleSort(QuickSort(globalArray**,** SIZE\_ARRAY-**1, 0**)))**;** system\_clock::time\_point end = system\_clock:: now()**;** cout << "\n"**;** duration<double> sec = end - start**;** cout << "Время затраченное на сортировку Quick Sort:" << sec.count() << "\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \* Обычный поиск введенного пользователем числа в массиве  
 \*/  
bool NormalSearch (int\* normalSearchArr**,**int number) {  
  
 bool flag= false**;** for (int i=**0;** i<SIZE\_ARRAY**;** i++) {  
 if (normalSearchArr[i]==number) {  
 flag= true**;** }  
 }  
 return flag**;**}  
  
/\*  
 \* Тайминг бычного поиска введенного пользователем числа в массиве  
 \*/  
void TimeNormalSearch (int timeNum) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** PrintNumberSearch(NormalSearch(QuickSort(globalArray**,**SIZE\_ARRAY**, 0**)**,** timeNum))**;** system\_clock::time\_point end = system\_clock:: now()**;** cout << "\n"**;** duration<double> sec = end - start**;** cout << "Скорость работы обычного перебора:" << sec.count() << "\n\n"**;** cout << "\n\n"**;**}  
  
/\*  
 \*Бинарный поиск введенного пользователем числа в массиве  
 \*/  
bool BinarySearch (int\* binarySearchArr**,** int number2) {  
  
 bool flag = false**;** int l = **0;** int r = SIZE\_ARRAY-**1;** int mid**;** while ((l <= r) && (flag != true)) {  
 mid = (l + r) / **2;** if (binarySearchArr[mid] == number2) {  
 flag = true**;** }  
 if (binarySearchArr[mid] > number2) {  
 r = mid - **1;** }  
 else l = mid + **1;** }  
 return flag**;**}  
  
/\*  
 \* Тайминг бинарного поиска введенного пользователем числа в массиве  
 \*/  
void TimeBinarySearch (int timeNum) {  
  
 system\_clock::time\_point start = system\_clock::now()**;** PrintNumberSearch(BinarySearch(QuickSort(globalArray**,**SIZE\_ARRAY**, 0**)**,** timeNum))**;** system\_clock::time\_point end = system\_clock:: now()**;** cout << "\n"**;** duration<double> sec = end - start**;** cout << "Скорость работы бинарного поиска:" << sec.count() << "\n\n"**;** cout << "\n\n"**;**}  
/\*  
 \* Вывод есть ли введенное пользователем число в массиве  
 \*/  
void PrintNumberSearch (bool flag) {  
  
 if (flag) {  
 cout << "Введенное число есть в массиве\n"**;** }  
 else cout << "Введенное число отсутствует в массиве\n"**;**}

Приложение Б

результаты работы программы

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

1

Исходный массив:

95 55 -70 37 62 -72 93 78 -95 -49 -78 -96 66 78 59

94 -99 -47 37 39 -23 3 -62 -88 49 78 -44 -29 8 27

35 35 47 1 92 -71 -97 -44 74 27 72 -62 28 -37 -67

-31 -42 -30 52 -32 -7 95 -99 -81 32 52 47 -47 -8 64

45 -7 84 -64 -41 96 60 -85 71 -96 -90 -99 69 22 19

-28 -45 -76 70 -19 -76 33 -38 -67 -60 -23 98 15 35 96

-83 8 -83 37 19 75 65 -1 92 -85

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

2

Отсортированный массив:

-99 -99 -99 -97 -96 -96 -95 -90 -88 -85 -85 -83 -83 -81 -78

-76 -76 -72 -71 -70 -67 -67 -64 -62 -62 -60 -49 -47 -47 -45

-44 -44 -42 -41 -38 -37 -32 -31 -30 -29 -28 -23 -23 -19 -8

-7 -7 -1 1 3 8 8 15 19 19 22 27 27 28 32

33 35 35 35 37 37 37 39 45 47 47 49 52 52 55

59 60 62 64 65 66 69 70 71 72 74 75 78 78 78

84 92 92 93 94 95 95 96 96 98

Время затраченное на сортировку:0.0339079

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

3

\*\*\*Задание 3\*\*\*

Поиск max и min в неотсортированном массиве:

Максимум: 97

Время затраченное на поиск максимума в неотсортированном массиве:0.0010235

Минимум: -99

Время затраченное на поиск минимума в неотсортированном массиве:0.0009986

Поиск max и min в отсортированном массиве:

Максимум: 93

Время затраченное на поиск максимума в отсортированном массиве:0.0009975

Минимум: -99

Время затраченное на поиск минимума в отсортированном массиве:0.0009967

\*\*\*Конец Задание 3\*\*\*

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

4

\*\*\*Задание 4\*\*\*

Среднее значение Max и Min: -3

Индексы элементов, значения которых равны среднему значению Max и Min: 41

Количество элементов, значения которых равны среднему значению Max и Min: 1

\*\*\*Конец Задание 4\*\*\*

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

5

\*\*\*Задание 5\*\*\*

Введите число a:8

Количество элементов в отсортированном массиве, которые меньше числа a: 49

\*\*\*Конец Задание 5\*\*\*

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

6

\*\*\*Задание 6\*\*\*

Введите число b:-7

Количество элементов в отсортированном массиве, которые меньше числа a: 61

\*\*\*Конец Задание 6\*\*\*

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

8

\*\*\*Задание 8\*\*\*

Введите индексы элементов массива, которые хотите поменять местами:0

99

97 -98 -98 -98 -98 -94 -94

-92 -90 -83 -83 -82 -79 -78 -77 -77 -77 -68 -68 -66 -60 -60

-51 -50 -45 -42 -39 -39 -39 -38 -33 -27 -27 -25 -18 -14 -10

-8 -7 -5 -4 -3 -2 0 2 2 3 5 5 8 9 10

12 15 21 23 24 30 30 32 39 39 40 45 45 46 49

53 54 56 57 59 60 60 62 63 65 66 67 67 69 70

70 73 75 76 76 77 78 78 78 79 79 81 82 83 85

86 93 -99

Скорость обмена:0.0369004

\*\*\*Конец Задание 8\*\*\*

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

1

Исходный массив:

98 -47 -36 -43 -73 -52 39 71 54 46 -95 48 2 -6 -84

80 -62 14 -85 -71 -91 -2 87 77 1 -85 85 -49 -35 79

33 -8 -70 98 -96 68 68 4 6 59 -72 -5 -14 -36 -22

-59 -94 -86 -68 -30 -4 30 42 4 -23 -14 -10 -40 -56 -5

47 41 42 97 -23 -54 -57 -63 -18 80 47 26 9 28 71

14 -13 -53 44 -75 54 -62 -35 38 -53 -56 -86 -1 54 6

-23 -92 74 -25 -17 19 92 43 -61 17

\*\*\*

Введите номер задачи:

1) Создать рандомно массив:

2) Отсортировать заданный массив:

3) Вывод максимального и минимального элемента массива

4) Вывод среднего значения максимального и минимального элемента массива

5) Количество, элементов, меньше числа a

6) Количество, элементов, больше числа b

7) Вывод информации о том, есть ли заданное число в массиве

8) Поменять местами элементы, индексы которых заданы

7

Отсортированный массив:

-96 -95 -94 -92 -91 -86 -86 -85 -85 -84 -75 -73 -72 -71 -70

-68 -63 -62 -62 -61 -59 -57 -56 -56 -54 -53 -53 -52 -49 -47

-43 -40 -36 -36 -35 -35 -30 -25 -23 -23 -23 -22 -18 -17 -14

-14 -13 -10 -8 -6 -5 -5 -4 -2 -1 1 2 4 4 6

6 9 14 14 17 19 26 28 30 33 38 39 41 42 42

43 44 46 47 47 48 54 54 54 59 68 68 71 71 74

77 79 80 80 85 87 92 97 98 98

Время затраченное на сортировку Quick Sort:0.0339552

Введите число, которое хотите найти в массиве:17

Введенное число есть в массиве

Скорость работы бинарного поиска:0.0018972

Введенное число есть в массиве

Скорость работы обычного перебора:0.0009987