**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Изучение и реализация различных алгоритмов сортировки и поиска в массиве данных»**

**Вариант 1**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_ Ковтун А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2020

Оглавление

[1. Постановка задачи и описание реализуемых алгоритмов 3](#_Toc56973147)

[2. Описание реализованных unit-тестов 3](#_Toc56973148)

[3. Код программы 3](#_Toc56973149)

[4. Пример работы 7](#_Toc56973150)

[5. Сравнение временной сложности алгоритмов сортировки 8](#_Toc56973151)

[6. Вывод 8](#_Toc56973152)

# Постановка задачи и описание реализуемых алгоритмов

Реализовать алгоритмы сортировки и поиска для массива.

Для этого используем следующие функции:

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Оценка временной сложности** |
| void swap(int\* a, int\* b) | Меняет местами значения | O(1) |
| sorting(int\* , int ) |  | O(n) |
| moving(int\* , int , int ) |  | O(n) |
| sorting(int\* , int ) |  | O(n) |
| bool isSorted(int\*, int) | Проверяет отсортирован список или нет | O(n) |
| int binarySearch(int\*, int, int) | Выполняет двоичный поиск | O(log\_2(n)) |
| void insertionSort(int\*, int) | Выполняет сортировку вставками | O(n^2) |
| void quickSort(int\*, int, int) | Выполняет быструю сортировку | O(n\*log(n)) |
| void countingSort(char\*, int) | Выполняет сортировку подсчетом | O(max+n) |
| void bogoSort(int\*, int) | Выполняет глупую сортировку | O(n\*n!) |
| void printArray(char\*, int) | Вывод массива типа char | O(n) |
| void printArray(double\*, int) | Вывод массива типа double | O(n) |
| void printArray(double\*, int) | Вывод массива типа double | O(n) |

# Описание реализованных unit-тестов

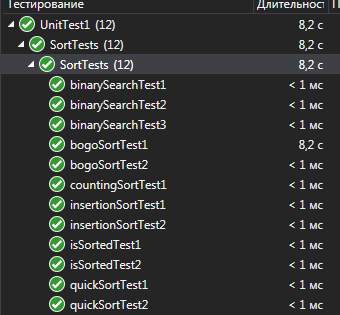
|  |  |
| --- | --- |
| Имя теста | Описание |
| isSortedTest1 | Проверка сортированного массива |
| isSortedTest2 | Проверка несортированного массива |
| binarySearchTest1 | Проверка двоичного поиска на отсортированном массиве |
| binarySearchTest2 | Проверка двоичного поиска на неотсортированном массиве |
| binarySearchTest3 | Проверка двоичного поиска несуществующего элемента |
| quickSortTest1 | Проверка сортировки несортированного массива |
| quickSortTest2 | Проверка сортировки сортированного массива |
| insertionSortTest1 | Проверка сортировки несортированного массива |
| insertionSortTest2 | Проверка сортировки сортированного массива |
| bogoSortTest1 | Проверка сортировки несортированного массива |
| bogoSortTest2 | Проверка сортировки сортированного массива |
| countingSortTest1 | Проверка сортировки массива из символов |

# Код программы

|  |
| --- |
| Sorts.cpp |
| #include <iostream>  using namespace std;  void swap(int\* a, int\* b) {  \*a -= \*b;  \*a += \*b;  \*b = \*a - \*b;  }  bool isSorted(int arr[], int n) { // checking for sorting  int i = 0;  while (i < n - 1)  if (arr[i] > arr[i++ + 1])  return false;  return true;  }  void sorting(int\* arr, int n)  {  for (int i = 0; i < n; i++)  swap(arr[i], arr[rand() % n]);  }  int moving(int\* arr, int l, int r)  {  int i = l - 1;  int p = arr[r];  for (int j = l; j < r; j++)  {  if (arr[j] < p)  {  i++;  swap(arr[i], arr[j]);  }  }  swap(arr[i + 1], arr[r]);  return i + 1;  }  int binarySearch(int\* arr, int a, int lastel, int cnt) { // binary search algorithm  if (!isSorted(arr, lastel)) {  return -1;  throw "This array isn`t sorted";  }  while (a <= lastel) {  int m = a + (lastel - a) / 2;  if (arr[m] == cnt)  return m;  if (arr[m] < cnt)  a = m + 1;  else  lastel = m - 1;  }  return -1;  throw "This element doesn`t exist";  }  void quickSort(int\* arr, int a, int b) // quick sort algorithm  {  if (a < b)  {  int i2 = moving(arr, a, b);  quickSort(arr, a, i2 - 1);  quickSort(arr, i2 + 1, b);  }  }  void insertionSort(int\* arr, int size) { // InsertionSort algorithm  int i,j, key;  for (i = 1; i < size; i++)  {  key = arr[i];  j = i - 1;  while (j >= 0 && arr[j] > key)  {  arr[j + 1] = arr[j];  j = j - 1;  }  arr[j + 1] = key;  }  }  void bogoSort(int\* arr, int size) { // the "stupid" sorting algorithm  while (!isSorted(arr, size)) sorting(arr, size);  }  void countingSort(char\* arr, int size) { // counting sorting algorithm  char\* readyarr = new char[size];  int symbol[256], i;  for (i = 0; i < 256; i++)  symbol[i] = 0;  for (int i = 0; i < size; i++)  symbol[arr[i]]++;  for (i = 1; i <= 255; i++)  symbol[i] += symbol[i - 1];  for (i = 0; i < size; i++)  {  readyarr[symbol[arr[i]] - 1] = arr[i];  symbol[arr[i]] --;  }  for (i = 0; i < size; i++)  {  arr[i] = readyarr[i];  }  }  void printArray(int\* arr, int size) { // Array output  for (int i = 0; i < size; i++) cout << \*(arr + i) << " ";  cout << endl;  }  void printArray(char\* arr, int size) { // Array output  for (int i = 0; i < size; i++) cout << \*(arr + i);  cout << endl;  }  void printArray(double\* arr, int size) { // Array output  for (int i = 0; i < size; i++) cout << \*(arr + i) << " ";  cout << endl;  } |
| Main.cpp |
| #include <iostream>  #include "Sorts.h"  #include <time.h>  using namespace std;  int main() {  const int nmbr = 10;  char\* charr = new char[nmbr];  int\* arr = new int[nmbr];  double\* res = new double[10];  srand(time(0));  for (int j = 0; j < 10; j++) {  for (int i = 0; i < nmbr; i++)  \*(arr + i) = rand() % (nmbr \* 10);  clock\_t start = clock();  //quickSort(arr, 0, cnt - 1);  insertionSort(arr, nmbr);  clock\_t stop = clock();  clock\_t itog = stop - start;  res[j] = ((double)itog \* 1000 / CLOCKS\_PER\_SEC);  }  for (int i = 0; i < 10; i++) cout << res[i] << " ";  } |
| UnitTest1.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "..\2прога\Sorts.cpp"  #include <iostream>  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace SortTests  {  TEST\_CLASS(SortTests)  {  public:  int arr[10] = { 4, 1, 6, 10, 2, 9, 5, 3, 7, 8};  int sorted[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  char charr[10] = { 'b', 'a', 'q', 'w', 'c', 'x', 'u', 'i', 'j', 'k' };  TEST\_METHOD(isSortedTest1)  {  Assert::IsTrue(isSorted(sorted, 10));  }  TEST\_METHOD(isSortedTest2)  {  Assert::IsTrue(!isSorted(arr, 10));  }  TEST\_METHOD(binarySearchTest1)  {  Assert::IsTrue(binarySearch(sorted, 0, 10, 4) == 3);  }  TEST\_METHOD(binarySearchTest2)  {  try  {  int res = binarySearch(arr, 0, 9, 4);  }  catch (const char\* warning)  {  Assert::AreEqual(warning, "This array isn`t sorted");  }  }  TEST\_METHOD(binarySearchTest3)  {  try  {  int res = binarySearch(sorted, 0, 9, 0);  }  catch (const char\* warning)  {  Assert::AreEqual(warning, "This element doesn`t exist");  }  }  TEST\_METHOD(quickSortTest1)  {  quickSort(arr, 0, 9);  Assert::IsTrue(isSorted(arr, 10));  }  TEST\_METHOD(quickSortTest2)  {  quickSort(sorted, 0, 9);  Assert::IsTrue(isSorted(sorted, 10));  }  TEST\_METHOD(insertionSortTest1)  {  insertionSort(arr, 10);  Assert::IsTrue(isSorted(arr, 10));  }  TEST\_METHOD(insertionSortTest2)  {  insertionSort(sorted, 10);  Assert::IsTrue(isSorted(sorted, 10));  }  TEST\_METHOD(bogoSortTest1)  {  bogoSort(arr, 10);  Assert::IsTrue(isSorted(arr, 10));  }  TEST\_METHOD(bogoSortTest2)  {  bogoSort(sorted, 10);  Assert::IsTrue(isSorted(sorted, 10));  }  TEST\_METHOD(countingSortTest1)  {  countingSort(charr, 10);  Assert::IsTrue(charr[0] == 'a' && charr[9] == 'x');  }  };  } |

# Пример работы





# Сравнение временной сложности алгоритмов сортировки

|  |  |  |
| --- | --- | --- |
| Количество элементов | quicksort, мс | insertionSort, мс |
| 10 | 0 | 0 |
| 100 | 0,3 | 0 |
| 1000 | 0,9 | 0,7 |
| 10000 | 17,5 | 94,6 |
| 100000 | 156,9 | 7676,2 |

# Вывод

Проделав лабораторную работу, я познакомился с новыми видами сортировки и научился их реализовывать.