**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по лабораторной работе №2**

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

**Тема: «Алгоритмы сортировки и поиска»**

**Вариант 10.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Кнауб К.В. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

## Постановка задачи

Задание: Реализовать алгоритмы сортировки массивов.

## Описание реализуемых алгоритмов

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Оценка временной сложности** |
| void change(int& a, int& b) | Меняет местами значения | O(1) |
| void PrintArray(int\*, s) | Вывод массива | O(n) |
| void PrintArray(char\*, int) | Вывод массива | O(n) |
| bool correct(int\*, int) | Проверяет отсортированность | O(n) |
| int BinarySearch(int\*, int, int) | Выполняет двоичный поиск | O(log\_2(n)) |
| void BubbleSort(int\*, int) | Выполняет сортировку пузырьком | O(n^2) |
| void QuickSort(int\*, int, int) | Выполняет быструю сортировку | O(n\*log(n)) |
| void BogoSort(int\*, int) | Выполняет глупую сортировку | O(n\*n!) |
| void CountingSort(char\*, int) | Выполняет сортировку подсчетом | O(max+n) |

## Cравнение временной сложности алгоритмов

Сравнение временной сложности алгоритмов BubbleSort и QuickSort.

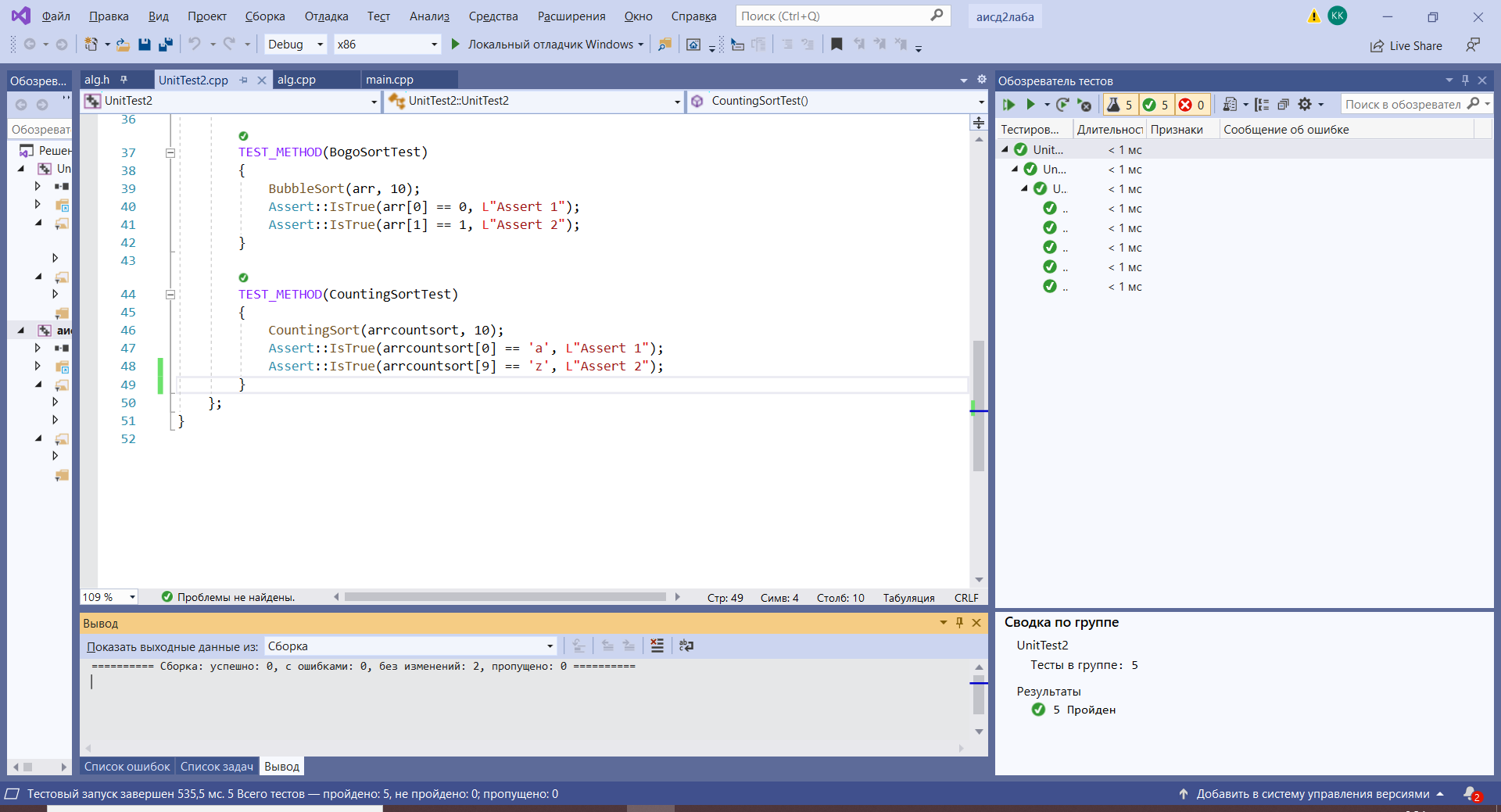
|  |  |  |
| --- | --- | --- |
| Размерность данных | Время для QuickSort | Время для BubbleSort |
| 10 | 0 мс | 0 мс |
| 100 | 0 мс | 0,1 мс |
| 1000 | 0,3 мс | 6,9 мс |
| 10000 | 2,1 мс | 48,3 мс |
| 100000 | 20,5 мс | 471,5 мс |

## 

## Описание реализованных unit-тестов.

|  |  |
| --- | --- |
| **Тест** | **Что проверяет** |
| BinarySearchTest | Проверка функции BinarySearch в отсоритрованном массиве |
| BubbleSortTest | Проверка функции BubbleSort в неотсортированном массиве |
| QuickSortTest | Проверка функции QuickSort в отсортированном массиве |
| BogoSortTest | Проверка функции BogoSort в неотсортированном массиве |
| CountingSortTest | Проверка функции CountingSort в неотсортированном массиве |

## Пример работы unit-тестов.



## Листинг

|  |
| --- |
| alg.h |
| #pragma once  int BinarySearch(int\* A, int size, int search\_num);  void QuickSort(int\* A, int low, int high);  void BubbleSort(int\* A, int size);  void BogoSort(int\* A, int size);  void CountingSort(char\* A, int size);  void change(int& a, int& b);  void PrintArray(int\* A, int size);  void PrintArray(char\* A, int size);  bool correct(int\* A, int size); |
| alg.cpp |
| #include "alg.h"  #include <iostream>  int BinarySearch(int\* A, int size, int search\_num)  {  if (size <= 0)  throw "Incorrect data";  if (!(correct(A, size)))  throw "Array is not sorted";  int middle, left, right;  left = 0; right = size - 1;  while (true)  {  if (left > right) return (-1);  middle = left + (right - left) / 2;  if (A[middle] < search\_num) left = middle + 1;  if (A[middle] > search\_num) right = middle - 1;  if (A[middle] == search\_num) return middle;  }  }  void change(int& a, int& b)  {  int temp = a;  a = b;  b = temp;  }  void PrintArray(int\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  for (int i = 0; i < size; i++)  {  std::cout << A[i] << ' ';  }  std::cout << std::endl;  }  void PrintArray(char\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  for (int i = 0; i < size; i++)  {  std::cout << A[i] << ' ';  }  std::cout << std::endl;  }  void BubbleSort(int\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  bool control = true;  for (int i = 0; i < size - 1; i++)  {  for (int j = 0; j < size - i - 1; j++)  {  if (A[j] > A[j + 1])  {  if (control == true)  control = false;  change(A[j], A[j + 1]);  }  }  if (control == true) break;  }  }  void QuickSort(int\* A, int low, int high)  {  int i = low;  int j = high;  int pivot = A[high];  while (i <= j)  {  while (A[i] < pivot)  i++;  while (A[j] > pivot)  j--;  if (i <= j)  {  change(A[i], A[j]);  i++;  j--;  }  }  if (j > low)  QuickSort(A, low, j);  if (i < high)  QuickSort(A, i, high);  }  void BogoSort(int\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  while (!correct(A, size))  {  for (int i = 0; i < size; ++i)  change(A[i], A[(rand() % size)]);  }  }  bool correct(int\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  while (--size > 0)  if (A[size - 1] > A[size])  return false;  return true;  }  void CountingSort(char\* A, int size)  {  if (size <= 0)  throw "Incorrect data";  char max = char(0);  for (int i = 0; i < size; i++)  {  if (A[i] > max)  {  max = A[i];  }  }  int count\_size = int(max + 1);  int\* count = new int[count\_size];  std::memset(count, 0, count\_size \* sizeof(\*count));  for (int i = 0; i < size; i++)  {  if (A[i] != '\0')  {  count[int(A[i])]++;  }  }  int i = 0, control = 0;  for (int j = 0; j < count\_size; j++)  {  if (count[j] != 0)  {  while (i < count[j] + control)  {  A[i] = char(j);  i++;  }  control = i;  }  }  delete[]count;  } |
| Main.cpp |
| #include <iostream>  #include "alg.h"  #include <time.h>  #include <fstream>  using namespace std;  int main()  {  srand(time(0));  int size = 10, minus;  double average;  int\* array\_for\_tests;  clock\_t start\_time, end\_time, result\_time;  ofstream of("Time\_test.txt");  double result[10];  of << " QuickSort: " << endl;  for (int i = 0; i < 5; i++)  {  for (int j = 0; j < 10; j++)  {  array\_for\_tests = new int[size];  for (int k = 0; k < size; k++)  {  minus = -1;  if (rand() % 10 > 4)  minus = 1;  array\_for\_tests[k] = rand() \* minus;  }  start\_time = clock();  QuickSort(array\_for\_tests, 0, size - 1);  end\_time = clock();  result\_time = end\_time - start\_time;  result[j] = ((double)result\_time \* 1000 / CLOCKS\_PER\_SEC);  of << result[j] << " ";  delete[] array\_for\_tests;  }  average = 0;  for (int j = 0; j < 10; j++)  average += result[j];  average /= 10;  of << "for " << size << " elements\n " << " Average: " << average << '\n';  size \*= 10;  }  size = 10;  of <<"\n" << "BubbleSort: " << endl;  for (int i = 0; i < 5; i++)  {  for (int j = 0; j < 10; j++)  {  array\_for\_tests = new int[size];  for (int k = 0; k < size; k++)  {  array\_for\_tests[k] = rand();  }  start\_time = clock();  BubbleSort(array\_for\_tests, size);  end\_time = clock();  result\_time = end\_time - start\_time;  result[j] = ((double)result\_time \* 1000 / CLOCKS\_PER\_SEC);  of << result[j] << " ";  delete[] array\_for\_tests;  }  average = 0;  for (int j = 0; j < 10; j++)  average += result[j];  average /= 10;  of << "for " << size << " elements\n " << " Average: " << average << '\n';  size \*= 10;  }  of.close();  } |
| UnitTest2.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "../аисд2лаба/alg.h"  #include "../аисд2лаба/alg.cpp"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UnitTest2  {  TEST\_CLASS(UnitTest2)  {  public:    int arr[10] = { 1,2,0,10,7,3,9,8,11,4 };  int arrsort[10] = { 0,1,2,3,4,5,6,7,8,9 };  char arrcountsort[10] = { 'a','s','b','c','f','e','z','g','h','k' };    TEST\_METHOD(BinarySearchTest)  {  Assert::IsTrue(BinarySearch(arrsort, 10, 5) == 5);  }    TEST\_METHOD(BubbleSortTest)  {  BubbleSort(arr, 10);  Assert::IsTrue(arr[0] == 0, L"Assert 1");  Assert::IsTrue(arr[1] == 1, L"Assert 2");  }  TEST\_METHOD(QuickSortTest)  {  BubbleSort(arr, 10);  Assert::IsTrue(arr[0] == 0, L"Assert 1");  Assert::IsTrue(arr[1] == 1, L"Assert 2");  }  TEST\_METHOD(BogoSortTest)  {  BubbleSort(arr, 10);  Assert::IsTrue(arr[0] == 0, L"Assert 1");  Assert::IsTrue(arr[1] == 1, L"Assert 2");  }    TEST\_METHOD(CountingSortTest)  {  CountingSort(arrcountsort, 10);  Assert::IsTrue(arrcountsort[0] == 'a', L"Assert 1");  Assert::IsTrue(arrcountsort[9] == 'z', L"Assert 2");  }  };  } |