**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 \_\_\_\_\_\_\_\_\_\_\_\_ Тарабурин А.П.

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) |
| void printArray(int\*, s) | Вывод массива типа int | O(n) |
| void printArray(char\*, int) | Вывод массива типа char | O(n) |
| void printArray(double\*, int) | Вывод массива типа double | O(n) |
| bool isSorted(int\*, int) | Проверяет отсортированность списка | O(n) |
| int binarySearch(int\*, int, int) | Выполняет двоичный поиск | O(log\_2(n)) |
| void insertionSort(int\*, int) | Выполняет сортировку пузырьком | O(n^2) |
| void quickSort(int\*, int, int) | Выполняет быструю сортировку | O(n\*log(n)) |
| int partition(int\*, int, int) |  | O(n) |
| void bogoSort(int\*, int) | Выполняет глупую сортировку | O(n\*n!) |
| shuffle(arr\*, int) |  | O(n) |
| void countingSort(char\*, int) | Выполняет сортировку подсчетом | O(max+n) |

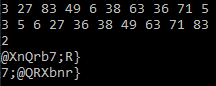
# Описание реализованных unit-тестов

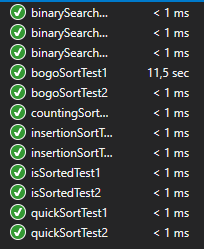
|  |  |
| --- | --- |
| Имя теста | Описание |
| isSortedTest1 | Проверка сортированного массива |
| isSortedTest2 | Проверка несортированного массива |
| binarySearchTest1 | Проверка двоичного поиска на отсортированном массиве |
| binarySearchTest2 | Проверка двоичного поиска на неотсортированном массиве |
| 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 симовлов |

# Код программы

|  |
| --- |
| SortFunc.cpp |
| #include <iostream>  using namespace std;  void swap(int\* a, int\* b) {  \*a -= \*b;  \*a += \*b;  \*b = \*a - \*b;  }  void printArray(int\* arr, int size) {  for (int i = 0; i < size; i++) cout << \*(arr + i) << " ";  cout << endl;  }  void printArray(char\* arr, int size) {  for (int i = 0; i < size; i++) cout << \*(arr + i);  cout << endl;  }  void printArray(double\* arr, int size) {  for (int i = 0; i < size; i++) cout << \*(arr + i) << " ";  cout << endl;  }  bool isSorted(int arr[], int n) {  int i = 0;  while (i < n - 1)  if (arr[i] > arr[i++ + 1])  return false;  return true;  }  int binarySearch(int\* arr, int l, int r, int x) {  if (!isSorted(arr, r)){  return -1;  throw "This array isn`t sorted";  }  while (l <= r) {  int m = l + (r - l) / 2;  if (arr[m] == x)  return m;  if (arr[m] < x)  l = m + 1;  else  r = m - 1;  }  return -1;  throw "This element doesn`t exist";  }  int partition(int\* arr, int l, int r)  {  int i = l - 1;  int pivot = arr[r];  for (int j = l; j < r; j++)  {  if (arr[j] < pivot)  {  i++;  swap(arr[i], arr[j]);  }  }  swap(arr[i + 1], arr[r]);  return i + 1;  }  void quickSort(int\* arr, int l, int r)  {  if (l < r)  {  int pi = partition(arr, l, r);  quickSort(arr, l, pi - 1);  quickSort(arr, pi + 1, r);  }  }  void insertionSort(int\* arr, int size) {  int i, key, j;  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 shuffle(int\* arr, int n)  {  for (int i = 0; i < n; i++)  swap(arr[i], arr[rand() % n]);  }  void bogoSort(int\* arr, int size) {  while (!isSorted(arr, size)) shuffle(arr, size);  }  void countingSort(char\* arr, int size) {  int max = 0;  for (int i = 0; i < size; i++)  if (\*(arr + i) > max) max = \*(arr + i);  max -= 32;  int\* temp = new int[max];  for (int i = 0; i < max; i++) temp[i] = 0;  for (int i = 0; i < size; i++) temp[arr[i] - 33]++;  int iter = 0;  for (int i = 0; i < max;)  if (temp[i] == 0) i++;  else {  arr[iter++] = i + 33;  temp[i]--;  }  } |
| Main.cpp |
| #include <iostream>  #include "SortFunc.h"  #include <time.h>  using namespace std;  int main() {  const int cnt = 10;  char\* chararr = new char[cnt];  int\* arr = new int[cnt];  double\* res = new double[10];  //arr = (int\*)realloc(arr, cnt \* sizeof(int));  srand(time(0));  for (int j = 0; j < 10; j++) {  for (int i = 0; i < cnt; i++)  \*(arr + i) = rand() % (cnt \* 10);  clock\_t start = clock();  insertionSort(arr, cnt);  clock\_t stop = clock();  clock\_t itog = stop - start;  res[j] = ((double)itog \* 1000 / CLOCKS\_PER\_SEC);  }  printArray(res, 10);  for (int i = 0; i < cnt; i++)  \*(chararr + i) = rand() % 93 + 33;  printArray(arr, cnt);  quickSort(arr, 0, cnt - 1);  //insertionSort(arr, cnt);  //bogoSort(arr, cnt);  printArray(arr, cnt);  cout << binarySearch(arr, 0, cnt - 1, 2) << endl;  printArray(chararr, cnt);  countingSort(chararr, cnt);  printArray(chararr, cnt);  return 0;  } |
| UnitTest1.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "..\Project1\SortFunc.cpp"  #include <iostream>  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace SortTests  {  TEST\_CLASS(SortTests)  {  public:  int arr[10] = { 8, 9, 7, 6, 1, 3, 2, 5, 10, 4 };  int sorted[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  char chararr[10] = {'h', 'a', 'v', 's', 'd', 'e', 'k', 'l', 'i', 'b'};  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(chararr, 10);  Assert::IsTrue(chararr[0] == 'a' && chararr[9] == 'v');  }  };  } |

# Пример работы





# Сравнение временной сложности алгоритмов сортировки

quickSort:

10 1 0 0 0 0 0 0 0 0 0

100 0 1 0 0 0 0 0 0 0 0

1000 1 2 0 0 1 1 0 1 1 0

10000 8 7 8 8 8 7 9 8 9 8

100000 107 90 93 92 96 91 101 98 96 111

insertionSort:

10 0 0 0 0 0 0 0 0 0 0

100 0 0 0 0 0 0 0 0 0 0

1000 1 1 0 0 1 1 0 1 1 1

10000 79 99 82 73 76 86 73 70 76 78

100000 9921 8797 9355 9086 9212 8580 8432 8881 8592 8674

|  |  |  |
| --- | --- | --- |
| Количество элементов | quicksort, мс | insertionSort, мс |
| 10 | 0,1 | 0 |
| 100 | 0,1 | 0 |
| 1000 | 0,7 | 0,7 |
| 10000 | 8 | 79,2 |
| 100000 | 97,5 | 8953 |

# Вывод

При выполнении данной лабораторной работы я познакомился с реализацией алгоритмов сортировки и поиска в массивах