**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: «Алгоритмы сортировки и поиска»**

**Вариант 2**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гелета А.И.

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

Санкт-Петербург

2020

# Постановка задачи

Реализовать различные алгоритмы сортировки массивы и поиска в нем требуемого элемента. Автоматизировать замер времени работы каждого алгоритма на случайных значениях массива.

# Описание реализуемых алгоритмов

int BinarySearch(int arr[], int size, int x) – алгоритм бинарного поиска. На вход функция получает сам массив, его размер и число, которое требуется найти, а возвращает индекс элемента или -1, если такого элемента нет. Массив на вход подается отсортированный. Принцип работы: берется середина массива, если искомый элемент, то поиск продолжается в левой части, если больше, то в правой. Аналогично производится поиск в части массива до тех пор, пока не будет найден элемент.

void QuickSort(int\* arr, long int left, long int right) – алгоритм быстрой сортировки. На вход подается массив, его левый индекс и правый. Выбирается опорный элемент и относительно него элементы меньшие переставляются слева, большие – справа. Для обоих частей рекуррентно вызывается эта же функция.

void BubbleSort(int\* arr, long int size) – обычная сортировка пузырьком. На вход подается массив и его размер. Сравниваются соседние элементы и меняются, если стоят не в правильном порядке до тех пор, пока массив не окажется отсортированным.

void BogoSort(int\* arr, int size) – глупая сортировка. На вход подается массив и его размер. Элементы случайным образом переставляются, пока массив не окажется отсортированным.

bool isSorted(int\* arr, int size) – дополнительная функция для глупой сортировки, проверяет, отсортирован ли массив.

void CountingSort(char\* arr, int size) – сортировка подсчетом для типа char. На входе массив и его размер. Создается дополнительный массив, который считает количество каждого элемента, а потом по нему элементы расставляются в нужном порядке.

# Оценка временной сложности каждого алгоритма

|  |  |
| --- | --- |
| Алгоритм | Сложность |
| BinarySearch | O(logn) |
| QuickSort | O(n\*logn) |
| BubbleSort | O(n2) |
| BogoSort |  |
| CountingSort | O(n+k) |

# Сравнение временной сложности QuickSort и BubbleSort

**Экспериментальные замеры, наносекунды**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Bubble | Quick | Bubble | Quick | Bubble | Quick |
| 1 | 600 | 300 | 1000 | 300 | 400 | 300 |
| 10 | 3100 | 1600 | 2200 | 1500 | 1300 | 1100 |
| 100 | 209000 | 51800 | 227900 | 28200 | 199600 | 20000 |
| 1000 | 18371700 | 273000 | 20953700 | 306000 | 19258700 | 312700 |
| 10000 | 2,02E+09 | 3467200 | 2,23E+09 | 4079100 | 2,07E+09 | 3533500 |
| 10000 | 2,25E+11 | 48934100 | 2,29E+11 | 44108700 | 2,44E+11 | 45428100 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Bubble | Quick | Bubble | Quick |
| 1 | 600 | 200 | 400 | 300 |
| 10 | 2500 | 1800 | 2800 | 2000 |
| 100 | 231600 | 22600 | 288100 | 26500 |
| 1000 | 19451400 | 263700 | 20237300 | 329500 |
| 10000 | 2,05E+09 | 3406200 | 1,97E+09 | 4279600 |
| 100000 | 2,04E+11 | 46196300 | 2,11E+11 | 47636900 |

**Средние значения, наносекунды**

|  |  |  |
| --- | --- | --- |
| N | Bubble | Quick |
| 1 | 600 | 280 |
| 10 | 2380 | 1600 |
| 100 | 231240 | 29820 |
| 1000 | 19654560 | 296980 |
| 10000 | 2,07E+09 | 3753120 |
| 100000 | 2,22E+11 | 46460820 |

# Описание реализованных unit-тестов

TEST\_CLASS(Sorted) – класс для проверки на заранее отсортированном списке

IsSorted – тестирует функцию, определяющую, отсортирован ли список.

Search – тестирует бинарный поиск в середине списка

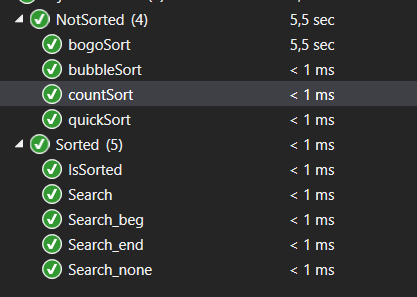
Search\_beg - бинарный поиск в начале списка

Search\_end – бинарный поиск в конце списка

Search\_none – бинарный поиск элемента, не присутствующего в массиве.

TEST\_CLASS(NotSorted) – класс для проверки несортированного списка

Каждый тест, соответственно, тестирует функцию (алгоритм) с таким же названием.



# Текст программы

|  |  |
| --- | --- |
|  | 3semA2.cpp |
|  | #include <iostream>  #include<chrono>  #include<fstream>  #include<math.h>  using namespace std;  bool isSorted(int\* arr, int size) {  for (int i = 0; i < size - 1; i++)  if (arr[i] > arr[i + 1]) return false;  return true;  }  int BinarySearch(int arr[], int size, int x) {  int left = 0, right = size - 1, mid;  while (left < right) {  mid = (left + right) / 2;  if (arr[mid] == x) return mid;  else if (arr[mid] > x) right = mid - 1;  else left = mid + 1;  }  return -1;  }  void QuickSort(int\* arr, long int left, long int right) {  long int pivot = arr[(left + right) / 2];  long int i = left, j = right;  while (i < j) {  while (arr[i] < pivot) i++;  while (arr[j] > pivot) j--;  if (i <= j) {  swap(arr[i], arr[j]); i++; j--;  }  }  if (left < j) QuickSort(arr, left, j);  if (i < right) QuickSort(arr, i, right);  }  void BubbleSort(int\* arr, long int size) {  bool swapp;  for (long int j = 0; j < size; j++) {  swapp = false;  for (long int i = 0; i < size - j - 1; i++)  if (arr[i] > arr[i + 1]) {  swap(arr[i], arr[i + 1]); swapp = true;  }  if (swapp == false) break;  }  }  void BogoSort(int\* arr, int size) {  while (!isSorted(arr, size)) {  for (int i = 0; i < size; i++)  swap(arr[i], arr[rand() % size]);  }  }  void CountingSort(char\* arr, int size) {  const int count\_size = 128;  int count[count\_size];  int j = 0;  for (int i = 32; i < count\_size; i++) count[i] = 0;  for (int i = 0; i < size; i++) count[(int)arr[i]]++;  for (int i = 32; i < count\_size; i++) {  while (count[i] != 0) {  count[i]--;  arr[j] = (char)i;  j++;  }  }  }  int main()  {  fstream f; f.open("out.csv");  long int n;  //char\* arr = new char[n];  f << "N; Bubble; Quick" << endl;  for (int i = 0; i <= 5; i++) {  n = pow(10, i);  int\* arr = new int[n]; int\* arr2 = new int[n];  for (int i = 0; i < n; i++) {  arr[i] = rand(); arr2[i] = arr[i];  }  auto begin = chrono::steady\_clock::now();  BubbleSort(arr, n);  auto end = chrono::steady\_clock::now();  auto elapsed\_bub = chrono::duration\_cast<chrono::nanoseconds>(end - begin);  begin = chrono::steady\_clock::now();  QuickSort(arr2, 0, n - 1);  end = chrono::steady\_clock::now();  auto elapsed\_quick = chrono::duration\_cast<chrono::nanoseconds>(end - begin);  f << n << ";" << elapsed\_bub.count() << ";" << elapsed\_quick.count() << "\n";  cout << n << ": " << elapsed\_bub.count() << " " << elapsed\_quick.count() << " ns\n";  }  f.close();  } |