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»

Вариант 2

ТЕМА: АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА

Студент гр. 9302 _____ Кузнецов В.А.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург

2020

Постановка задачи.

Реализовать алгоритмы сортировки и поиска. Реализуемые алгоритмы: двоичный поиск, быстрая сортировка, сортировка пузырьком, глупая сортировка, сортировка подсчетом. Сравнить временную сложность быстрой сортировки и сортировки пузырьком.

Описание реализуемого класса и методов.

Были реализованы следующие функции:

1. Int BimarySearch(int* a, int n, int search) – реализация алгоритма бинарного поиска элемента search в отсортированном массиве a длиной n. Возвращает позицию элемента в массиве или -1 если он отсутствует.
2. Void QuickSort(int* a, int& left, int& right) – реализация алгоритма быстрой сортировки массива a, рекурсивная функция, сортирующая элементы с позиции left до right (таким образом в начале работы в left передается 0, а в right N-1, где N – длина массива a).
3. Int QuickSortPartition(int* a, int& left, int& right) – функция для одной итерации сортировки QuickSort, возвращает индекс элемента, с которого нужно начать и закончить новую итерацию сортировки.
4. Void BubbleSort(int* a, int n) – реализация алгоритма сортировки пузырьком массива a длины n.
5. Void BogoSort(int* a, int n) – реализация алгоритма глупой сортировки массива a длины n.
6. Void BogoSortShuffle(int* a, int n) – функция для сортировки рандомным образом.
7. Int BogoSortCorrect(int* a, int n) – функция, проверяющая, корректно ли был отсортирован массив a после сортировки рандомным образом. Возвращает 1, если корректно или 0 в противном случае.
8. Void CountingSort(char* a, int n) – реализация алгоритма сортировки подсчетом массива символов a длины n.

Также был реализован класс Timer исключительно для замеров скорости работы алгоритмов.

Оценка временной сложности алгоритмов.

В табл. 1 приведена сложность для каждой функции, реализующей алгоритм и ее временная сложность в среднем случае.

Таблица 1 – Временная сложность алгоритмов

Функция (алгоритм)	Сложность
BinarySearch	O(log(n))
QuickSort	O(n*log(n))
BubbleSort	$O(n^2)$
BogoSort	O(n*(n!))
CountingSort	O(n + 256)

Сравнение временной сложности алгоритма быстрой сортировки и сортировки пузырьком.

В табл. 2 представлены результаты сравнения времени работы алгоритмов на массивах размерности 10, 100, 1000, 10000 и 100000 элементов. Время работы расценивалось как среднее время из 10 запусков для каждой размерности. Элементы массивов определялись рандомно в интервале от -5000 до 5000.

Таблица 2 – Сравнение времени работы алгоритмов быстрой сортировки и пузырьком

Размерность массива	Быстрая сортировка, с.	Сортировка пузырьком, с.
10	$8{,}23 * 10^{-6}$	$7{,}82 * 10^{-5}$
100	$1{,}37 * 10^{-5}$	0,00076
1000	0,00098	0,067
10000	0,0133	5
100000	0,1148	492

Описание реализованных Unit-тестов.

Для проверки реализованных методов были написаны unit-тесты:

1. testBinarySearch – тестирование функции BinarySearch на отсортированном массиве длины 8, с различными входными данными и в том числе элементами, не входящими в исходный массив.
2. testQuickSort – тестирование функции QuickSort на массиве длины 8 путем сравнения каждого элемента массива с правильными выходными данными.
3. testBubbleSort – тестирование функции BubbleSort на массиве длины 8 путем сравнения каждого элемента массива с правильными выходными данными.
4. testBogoSort – тестирование функции BogoSort на массиве длины 6 путем сравнения каждого элемента массива с правильными выходными данными.
5. testCountingSort – тестирование функции CountingSort на массиве символов длины 8 путем сравнения каждого элемента массива с правильными выходными данными.

Все тесты были пройдены успешно.

Пример работы.

Работа программы приведена на рис. 1 и 2, где print – функция печати элементов массива, реализованная только для наглядного представления работы.

Рисунок 1 – Демонстрационный код

```
int main(){
    srand(time(0));
    const int n = 10;
    int l = 0;
    int r = n - 1;
    int a[n];
    int b[n];
    int c[5]{-5, -67, 0, 11, -2};
    char d[n]{'4', '2', 'f', 'a', 'c', '7', 'P', '4', 'Z', '\0'};
    for (int i = 0; i < n; i++) {
        a[i] = rand() % 100 - 50;
        b[i] = rand() % 100 - 50;
    }
    QuickSort(a, l, r);
    BubbleSort(b, n);
    BogoSort(c, 5);
    CountingSort(d, n-1);
    print(a, n);
    print(b, n);
    print(c, 5);
    std::cout << d << std::endl;
    std::cout << BinarySearch(c, n, 0);
}
```

Рисунок 2 – Результат работы

```
-49 -35 -30 -16 -3 -2 12 30 30 41
-40 -30 -19 -10 -2 20 27 31 41 44
-67 -5 -2 0 11
2447PZacf
3
```

Листинг.

```
#include <iostream>
#include <algorithm> //std::swap
#include <chrono>  //Timer

int BinarySearch(int* a, int n, int search) {
    int pos; //current position
    int left = 0; //left edge
    int right = n - 1;  //right edge
    while (true) {
        if (left > right)   //the gap has been narrowed to zero, search doesnt exist
            return -1;
        pos = left + (right - left) / 2;    //midpoint
        if (a[pos] < search)    //search in the second half
            left = pos + 1;
        if (a[pos] > search) //search in the first half
            right = pos - 1;
        if (a[pos] == search) //search found
            return pos;
    }
}

int QuickSortPartition(int* a, int& left, int& right) {
    int ai = a[(left + right) / 2]; //array center
    int i = left;
    int j = right;
    while (i <= j) {    //while the gap exists
        while (a[i] < ai) { //shorten the gap on the left
            i++;
        }
        while (a[j] > ai) { //shorten the gap on the right
            j--;
        }
        if (i >= j)
            break;
        std::swap(a[i++], a[j--]);  //swap the ends
    }
    return j; //the right end divides the array into two parts
}

void QuickSort(int* a, int& left, int& right) {
    if (left < right) { //while the gap exists
        int part = QuickSortPartition(a, left, right);  //single cycle iteration...
        QuickSort(a, left, part);  //...split the sorting into two subarrays
        int l = part + 1;
        QuickSort(a, l, right);
    }
}

void BubbleSort(int* a, int n) {
```

```
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1; j++) {
            if (a[j] > a[j + 1]) { //swap if the left number is greater than the right one.
                std::swap(a[j], a[j + 1]);
            }
        }
    }
}

int BogoSortCorrect(int* a, int n) { //whether the array is sorted
    while (--n > 0) {
        if (a[n - 1] > a[n])
            return 0;
    }
    return 1;
}

void BogoSortShuffle(int* a, int n) {    //random sort
    for (int i = 0; i < n; ++i) {
        std::swap(a[i], a[(rand() % n)]);
    }
}

void BogoSort(int* a, int n) {  //randomly sorted until it is correct
    while (!BogoSortCorrect(a, n))
        BogoSortShuffle(a, n);
}

void CountingSort(char* a, int n) {
    const int k = 256;      //maximum number of different chars
    char c[k];  //array of different chars
    int pos = 0;
    for (int i = 0; i < k; i++) {
        c[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        c[a[i]] = c[a[i]] + 1;  //number of different chars
    }
    for (int i = 0; i < k; i++) { //char 'k'
        for (int j = 0; j < c[i]; j++) { //number of char 'k'
            a[pos] = i;
            ++pos;
        }
    }
}

void print(int* a, int n) { //print array
    for (int i = 0; i < n; i++) {
        std::cout << a[i] << " ";
    }
    std::cout << std::endl;
}
```

```
class Timer{
private:
    using clock_t = std::chrono::high_resolution_clock;
    using second_t = std::chrono::duration<double, std::ratio<1> >;
    std::chrono::time_point<clock_t> m_beg;
public:
    Timer() : m_beg(clock_t::now()){
    }

    void reset(){
        m_beg = clock_t::now();
    }

    double elapsed() const{
        return std::chrono::duration_cast<second_t>(clock_t::now() - m_beg).count();
    }
};

int main(){
    srand(time(0));
    const int n = 10;
    int l = 0;
    int r = n - 1;
    int a[n];
    int b[n];
    int c[5]{-5, -67, 0, 11, -2};
    char d[n]{'4', '2', 'f', 'a', 'c', '7', 'P', '4', 'Z', '\0'};
    for (int i = 0; i < n; i++) {
        a[i] = rand() % 100 - 50;
        b[i] = rand() % 100 - 50;
    }
    QuickSort(a, l, r);
    BubbleSort(b, n);
    BogoSort(c, 5);
    CountingSort(d, n-1);
    print(a, n);
    print(b, n);
    print(c, 5);
    std::cout << d << std::endl;
    std::cout << BinarySearch(c, n, 0);
}
```