

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Сортировки**

Студентка гр. 9382

\_\_\_\_\_

Балаева М.О.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Применить на практике знания о сортировке вставками, реализовать сортировку вставками для массива и списка с произвольным типом данных на языке C++.

### **Основные теоретические положения.**

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке.

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

На вход алгоритма подаётся исходная последовательность  $n$  чисел, также называемых ключами:  $a_1, a_2, a_3, \dots, a_n$ . Входная последовательность на практике представляется в виде массива с  $n$  элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности  $b_1, b_2, b_3, \dots, b_n$ , чтобы выполнялось следующее соотношение  $b_1 \leq b_2 \leq b_3 \leq \dots \leq b_n$ .

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

### **Задание.**

Вариант №2

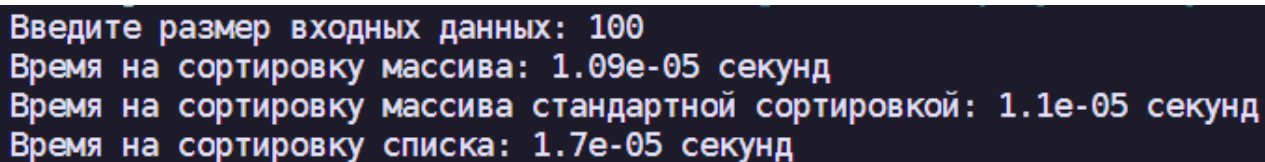
Сортировка простыми вставками; сортировка простыми вставками в список.

### **Ход работы:**

1. Произведён анализ задания.
2. Разработана программа, содержащая всебе:
  1. Класс Node, являющийся вложенным классом шаблонного класса List. Этот класс реализует представление элемента двусвязного списка и имеет следующие поля: значение элемента value и ссылки на предыдущий или последующий элемент — prevNode или nextNode.
  2. Шаблонный класс List, реализующий представление двусвязного списка с различными типами данных. Класс хранит указатель на первый элемент списка — front – и последний — back, а также хранит размер списка size. Для удобства создания экземпляров класса было написано несколько конструкторов: пустой; создающий List на основе указателя на массив; создающий List на основе стандартного контейнера std::array языка C++ и создающий List при помощи списка инициализации. Класс обладает следующими методами: getFront(), возвращающий указатель на передний элемент списка, getBack(), возвращающий указатель на последний элемент, getSize(), возвращающий размер списка, add(), добавляющий поданный на вход метода элемент в конец списка, swap(), меняющий значения элементов и print() - печатающий список. Для удобства доступа к элементам списка был написан метод at(), который возвращает значение элемента по заданному индексу, и перегружен оператор, который возвращает указатель на элемент по заданному индексу.
  3. Шаблонная функция InsertionSortArray() сортирующая массив с помощью алгоритма сортировкой вставками.
  4. Шаблонная функция printArray() для вывода массива в консоль.
  5. Шаблонная функция InsertionSortList() сортирующая список с помощью алгоритма сортировкой вставками.
  6. Функция main, в которой происходит считывание массива или списка, который в дальнейшем будет отсортирован.

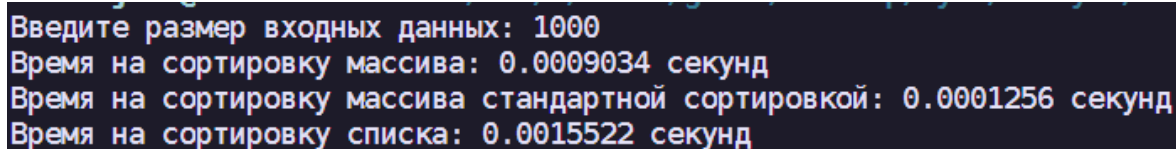
3. Произведено сравнение скорости разработанной сортировки вставками и сортировки стандартной библиотеки языка C++. Иллюстрации представлены в разделе «Иллюстрация сравнения скорости сортировок».
4. Были сделаны выводы по результатам, полученным в предыдущем пункте. Стандартная сортировка языка C++ `std::sort` в разы быстрее сортировки вставками. При этом при небольших размерах входных данных результаты примерно сравнимы, но с увеличением входных данных разрыв становится существенным в пользу стандартной сортировки. Например, массив из 100000 элементов алгоритм сортировки вставками сортирует в 470 раз медленнее, чем стандартная.
5. Произведено тестирование программы на различных входных данных.
6. Код разработанной программы расположен в Приложении А.

### **Иллюстрация сравнения скорости сортировок.**



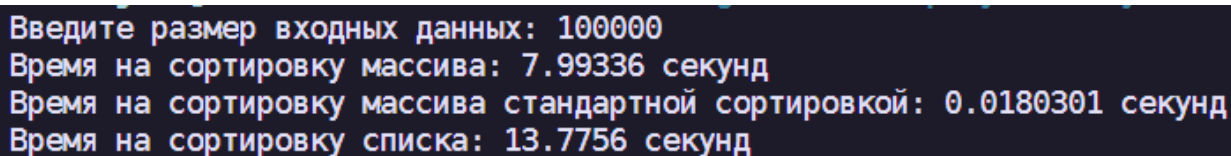
```
Введите размер входных данных: 100
Время на сортировку массива: 1.09e-05 секунд
Время на сортировку массива стандартной сортировкой: 1.1e-05 секунд
Время на сортировку списка: 1.7e-05 секунд
```

Рисунок 1 - Сравнение скорости работы сортировок на входных данных , состоящих из 100 чисел типа int



```
Введите размер входных данных: 1000
Время на сортировку массива: 0.0009034 секунд
Время на сортировку массива стандартной сортировкой: 0.0001256 секунд
Время на сортировку списка: 0.0015522 секунд
```

Рисунок 2 - Сравнение скорости работы сортировок на входных данных , состоящих из 1000 чисел типа int



```
Введите размер входных данных: 100000
Время на сортировку массива: 7.99336 секунд
Время на сортировку массива стандартной сортировкой: 0.0180301 секунд
Время на сортировку списка: 13.7756 секунд
```

Рисунок 3 - Сравнение скорости работы сортировок на входных данных , состоящих из 100000 чисел типа int

## Примеры работы программы.

Таблица 1 – Пример работы программы

№ п/п	Входные данные	Выходные данные	Комментарии
1.	10 -1 2 3 -5 1 0 10 23 -45 19	Исходный массив до сортировки: -1 2 3 -5 1 0 10  23 -45 19 Исходный массив после сортировки: -45 -5 -1 0 1 2  3 10 19 23	Тип входных данных - int
2.	10 a g h n d s s e i l	Исходный массив до сортировки: a g h n d s s e i l Исходный массив после сортировки: a d e g h i l n s s	Тип входных данных - char
3.	5 1.2 0.3 -0.01 10.2 -101.23	Исходный массив до сортировки: 1.2 0.3 -0.01 10.2 -101.23 Исходный массив после сортировки: -101.23 -0.01  0.3 1.2 10.2	Тип входных данных - float

		сортировки: 1.2 0.3 -0.01 10.2 -101.23	
--	--	---	--

### Иллюстрация работы программы.

```

Введите размер входных данных: 10
Введите входные данные: -1 2 3 -5 1 0 10 23 -45 19
Исходный массив до сортировки: -1 2 3 -5 1 0 10 23 -45 19
Исходный массив после сортировки: -45 -5 -1 0 1 2 3 10 19 23
Исходный список до сортировки: -1 2 3 -5 1 0 10 23 -45 19
Исходный список после сортировки: -45 -5 -1 0 1 2 3 10 19 23

```

Рисунок 4 - Иллюстрация работы программы с входными данными №1

```

Введите размер входных данных: 10
Введите входные данные: a g h n d s s e i l
Исходный массив до сортировки: a g h n d s s e i l
Исходный массив после сортировки: a d e g h i l n s s
Исходный список до сортировки: a g h n d s s e i l
Исходный список после сортировки: a d e g h i l n s s

```

Рисунок 5 - Иллюстрация работы программы с входными данными №2

```

Введите размер входных данных: 5
Введите входные данные: 1.2 0.3 -0.01 10.2 -101.23
Исходный массив до сортировки: 1.2 0.3 -0.01 10.2 -101.23
Исходный массив после сортировки: -101.23 -0.01 0.3 1.2 10.2
Исходный список до сортировки: 1.2 0.3 -0.01 10.2 -101.23
Исходный список после сортировки: -101.23 -0.01 0.3 1.2 10.2

```

Рисунок 6 - Иллюстрация работы программы с входными данными №3

```

Исходный массив до сортировки: -1 2 3 -5 1 0 10 23 -45 19
До сортировки: -1 2 3 -5 1 0 10 23 -45 19
Элемент 3 поставлен на место: -1 2 3 -5 1 0 10 23 -45 19
До сортировки: -1 2 3 -5 1 0 10 23 -45 19
Поменял местами -5 и 3: -1 2 -5 3 1 0 10 23 -45 19
Поменял местами -5 и 2: -1 -5 2 3 1 0 10 23 -45 19
Поменял местами -5 и -1: -5 -1 2 3 1 0 10 23 -45 19
Элемент -5 поставлен на место: -5 -1 2 3 1 0 10 23 -45 19
До сортировки: -5 -1 2 3 1 0 10 23 -45 19
Поменял местами 1 и 3: -5 -1 2 1 3 0 10 23 -45 19
Поменял местами 1 и 2: -5 -1 1 2 3 0 10 23 -45 19
Элемент 1 поставлен на место: -5 -1 1 2 3 0 10 23 -45 19
До сортировки: -5 -1 1 2 3 0 10 23 -45 19
Поменял местами 0 и 3: -5 -1 1 2 0 3 10 23 -45 19
Поменял местами 0 и 2: -5 -1 1 0 2 3 10 23 -45 19
Поменял местами 0 и 1: -5 -1 0 1 2 3 10 23 -45 19
Элемент 0 поставлен на место: -5 -1 0 1 2 3 10 23 -45 19
До сортировки: -5 -1 0 1 2 3 10 23 -45 19
Элемент 10 поставлен на место: -5 -1 0 1 2 3 10 23 -45 19
До сортировки: -5 -1 0 1 2 3 10 23 -45 19
Элемент 23 поставлен на место: -5 -1 0 1 2 3 10 23 -45 19
До сортировки: -5 -1 0 1 2 3 10 23 -45 19
Поменял местами -45 и 23: -5 -1 0 1 2 3 10 -45 23 19
Поменял местами -45 и 10: -5 -1 0 1 2 3 -45 10 23 19
Поменял местами -45 и 3: -5 -1 0 1 2 -45 3 10 23 19
Поменял местами -45 и 2: -5 -1 0 1 -45 2 3 10 23 19
Поменял местами -45 и 1: -5 -1 0 -45 1 2 3 10 23 19
Поменял местами -45 и 0: -5 -1 -45 0 1 2 3 10 23 19
Поменял местами -45 и -1: -5 -45 -1 0 1 2 3 10 23 19
Поменял местами -45 и -5: -45 -5 -1 0 1 2 3 10 23 19
Элемент -45 поставлен на место: -45 -5 -1 0 1 2 3 10 23 19
До сортировки: -45 -5 -1 0 1 2 3 10 23 19
Поменял местами 19 и 23: -45 -5 -1 0 1 2 3 10 19 23
Элемент 19 поставлен на место: -45 -5 -1 0 1 2 3 10 19 23
Исходный массив после сортировки: -45 -5 -1 0 1 2 3 10 19 23

```

Рисунок 7 - Иллюстрация вывода промежуточных результатов работы программы с входными данными №1

## **Выводы.**

Были применены на практике знания о сортировке вставками. Была реализована сортировка вставками для массива и списка с произвольным типом данных на языке программирования C++.



## ПРИЛОЖЕНИЕ А

### Файл main.cpp

```
#include
<iostream>

#include
<algorithm>

#include <chrono>

#include "List.h"

#include
"InsertionSort.h"


int main(){
    srand(0);


    int n;

    cout <<
"Введите размер
входных данных:
";

    cin >> n;

    float* myArr
= new float[n];

    float* myArr2
= new float[n];

    cout <<
"Введите входные
данные: ";

    for(int i =
0; i < n; i++){
        cin >>
myArr[i];

        myArr2[i]
= myArr[i];
    }

    List<float>
myList1 =
List<float>(myArr
, n);

    //List<int>
myList2 =
List<int>(myArr,
n);

    cout <<
"Исходный массив
до сортировки: ";

    printArray(myArr,
n);
```

```
InsertionSortArray(myArr, n);
```

```
    cout <<
    "Исходный массив
    после сортировки:
    ";
```

```
printArray(myArr,
n);
```

```
    cout <<
    "Исходный список
    до сортировки: ";
```

```
myList1.print();
```

```
InsertionSortList
(myList1);
```

```
    cout <<
    "Исходный список
    после сортировки:
    ";
```

```
myList1.print();
```

```
//chrono::steady_
clock::time_point
start =
chrono::steady_cl
ock::now();
```

```
//InsertionSortAr
ray(myArr, n);
```

```
//chrono::steady_
clock::time_point
end =
chrono::steady_cl
ock::now();
```

```
    //cout <<
    "Время на
    сортировку
    массива: " <<
    std::chrono::dura
    tion<double>(end
    - start).count()
    << " секунд\n";
```

```
    //start =
    chrono::steady_cl
    ock::now();
```

```

//sort(myArr2,
myArr2 + n);

    //end =
    chrono::steady_clock::now();

    //cout <<
    "Время на
    сортировку
    массива
    стандартной
    сортировкой: " <<
    std::chrono::duration<double>(end
    - start).count()
    << " секунд\n";

    //start =
    chrono::steady_clock::now();

//InsertionSortList(myList1);

    //end =
    chrono::steady_clock::now();

    //cout <<
    "Время на
    сортировку
    списка: " <<
    std::chrono::duration<double>(end
    - start).count()
    << " секунд\n";

    return 0;
}

```

## ФАЙЛ INSERTIONSORT.H

```

#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H

#include <iostream>
#include "List.h"

using namespace std;

template<typename T>
void printArray(T* arrayToPrint, int size){
    for(int i = 0; i < size; i++){
        cout << arrayToPrint[i] << ' ';
    }
    cout << '\n';
}

```

```
}
```

```
template<typename T>
void InsertionSortArray(T* arrayToSort, int size){
T key;
int j;
for(int i = 1; i < size; i++){
cout << "До сортировки: ";
printArray(arrayToSort, size);
key = arrayToSort[i];
j = i - 1;
while(j >= 0 && arrayToSort[j] > key){
cout << "Поменял местами " << key << " и " << arrayToSort[j] << ":\t";
arrayToSort[j + 1] = arrayToSort[j];
arrayToSort[j] = key;
printArray(arrayToSort, size);
j--;
}
arrayToSort[j + 1] = key;
cout << "Элемент " << key << " поставлен на место: ";
printArray(arrayToSort, size);
}
}
```

```
template<typename T>
void InsertionSortList(List<T>& listToSort){
auto front = listToSort.getFront();
auto tmp = front->nextNode;
auto back = listToSort.getBack();
while(tmp != nullptr){
cout << "До сортировки: ";
listToSort.print();
auto tmp2 = tmp->prevNode;
auto key = tmp;
while(tmp2 != nullptr && tmp2->value > key->value){
cout << "Поменял " << tmp2->value << " и " << key->value << ":\t";
listToSort.swap(tmp2, key);
listToSort.print();
key = tmp2;
tmp2 = tmp2->prevNode;
}
tmp = tmp->nextNode;
cout << "Элемент " << key->value << " поставлен на место: ";
listToSort.print();
tmp2 = tmp;
}
}
```

```
#endif
```

### **Файл List.h**

```
#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H
```

```

#include <iostream>
#include "List.h"

using namespace std;

template<typename T>
void printArray(T* arrayToPrint, int size){
    for(int i = 0; i < size; i++){
        cout << arrayToPrint[i] << ' ';
    }
    cout << '\n';
}

template<typename T>
void InsertionSortArray(T* arrayToSort, int size){
    T key;
    int j;
    for(int i = 1; i < size; i++){
        cout << "До сортировки: ";
        printArray(arrayToSort, size);
        key = arrayToSort[i];
        j = i - 1;
        while(j >= 0 && arrayToSort[j] > key){
            cout << "Поменял местами " << key << " и " << arrayToSort[j] << ":\t";
            arrayToSort[j + 1] = arrayToSort[j];
            arrayToSort[j] = key;
            printArray(arrayToSort, size);
            j--;
        }
        arrayToSort[j + 1] = key;
        cout << "Элемент " << key << " поставлен на место: ";
        printArray(arrayToSort, size);
    }
}

template<typename T>
void InsertionSortList(List<T>& listToSort){
    auto front = listToSort.getFront();
    auto tmp = front->nextNode;
    auto back = listToSort.getBack();
    while(tmp != nullptr){
        cout << "До сортировки: ";
        listToSort.print();
        auto tmp2 = tmp->prevNode;
        auto key = tmp;
        while(tmp2 != nullptr && tmp2->value > key->value){
            cout << "Поменял " << tmp2->value << " и " << key->value << ":\t";
            listToSort.swap(tmp2, key);
            listToSort.print();
            key = tmp2;
            tmp2 = tmp2->prevNode;
        }
    }
}

```

```
tmp = tmp->nextNode;
cout << "Элемент " << key->value << " поставлен на место: ";
listToSort.print();
tmp2 = tmp;
}
}

#endif
```