

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Российский химико-технологический университет имени Д.И.  
Менделеева»

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Выполнил студент группы КС-36: Золотухин А.А.

Ссылка на репозиторий: [https://github.com/  
MUCTR-IKT-CPP/  
ZolotukhinAA\\_36\\_ALG](https://github.com/MUCTR-IKT-CPP/ZolotukhinAA_36_ALG)

Принял: Крашенников Роман Сергеевич

Дата сдачи: 10.03.2025

Москва  
2025

# Оглавление

|                                  |   |
|----------------------------------|---|
| Описание задачи . . . . .        | 1 |
| Описание метода/модели . . . . . | 2 |
| Выполнение задачи . . . . .      | 3 |
| Выводы . . . . .                 | 9 |

## Описание задачи

Написать свою реализацию двусвязного списка:

- добавление элемента в начало, в конец, в произвольное место;
- удаление элемента из списка.

В рамках лабораторной работы необходимо изучить и реализовать двусвязный список. Структура должна:

- использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными;
- реализовывать своё итератор, предоставляющий стандартный для языка механизм работы с ним (для *C++* это операции *++* и *!=*);
- обеспечивать работу стандартных библиотек и конструкции *for each*, если она есть в языке, если их нет, то реализовать собственную функцию, использующую итератор;
- обеспечивать проверку на пустоту и подсчёт количества элементов.

Для демонстрации работы структуры необходимо создать набор тестов (под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

- заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчёт их суммы, среднего, минимального и максимального;
- провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов;
- заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020). Значения каждого поля генерируются случайно из набора заранее заданных. После заполнения необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом количества элементов, не подходящих под условие в новых структурах.

Тесты:

1. перемешать все элементы;
2. выполнить серию тестирования сортировки из первой лабораторной на реализованном списке и сравнить производительность с полученной на массиве.

## Описание метода/модели

**Двусвязный список** - это двунаправленный список, в котором каждый узел имеет два указателя: на следующий и предыдущий узлы, которые ссылаются на следующий и предыдущий узлы соответственно. В отличие от односвязанного списка, в котором каждый узел указывает только на следующий узел, в двусвязном списке есть дополнительный предыдущий указатель, который позволяет перемещаться как вперёд, так и назад.

Каждый узел двусвязного списка состоит из трёх полей:

- *data* - значение, хранящееся в узле;
- *next* - ссылка на следующий узел в списке;
- *prev* - ссылка на предыдущий узел в списке.

Анализ сложности основных операций над двусвязным списком:

- **Вставка в начало:**  $O(1)$ ;
- **Вставка в конец:**  $O(1)$ ;
- **Вставка в определенный узел:**  $O(n)$ ;
- **Удаление в начале:**  $O(1)$ ;
- **Удаление в конце:**  $O(1)$ ;
- **Удаление в определенном узле:**  $O(n)$ ;

*Преимущества:*

- Позволяет перемещаться как вперёд, так и назад;
- Удаление узла выполняется более эффективно и просто, поскольку у него есть указатель на предыдущий узел;
- Является динамическим по своей природе, поэтому он может увеличиваться и уменьшаться в размерах.

*Недостатки:*

- Для каждого узла требуется больше памяти, чем для массивов, из-за дополнительного хранилища, используемого для указателей;
- Его сложнее реализовать и поддерживать по сравнению с односвязным списком;
- Нужно пройти от головного узла к определённому узлу для вставки и удаления в определённых местах.

## Выполнение задачи

Двусвязный список реализован на языке *C++*. Построение графиков с помощью программы *GNUplot*.

"main" функция работает с вызовом методов для выполнения задания

```
1  int main() {
2      srand(time(0));
3
4      testIntegers();
5      testStrings();
6      testPersons();
7      testShuffle();
8      simulation();
9
10     return 0;
11 }
12
```

```
1  template <typename T>
2  class Node {
3      public:
4          T data;
5          Node<T> *prev;
6          Node<T> *next;
7
8          Node(const T &value) : data(value), prev(nullptr), next(nullptr) {}
9  };
10
```

```
1  template <typename T>
2  class LinkedList {
3      protected:
4          int size;
5          Node<T> *head;
6          Node<T> *tail;
7
8      public:
9          LinkedList() : size(0), head(nullptr), tail(nullptr) {}
10
11         void insertFront(const T &value) {
12             Node<T> *newNode = new Node<T>(value);
13             if (head == nullptr) {
14                 head = newNode;
15                 tail = newNode;
16             } else {
17                 newNode->next = head;
18                 head->prev = newNode;
19                 head = newNode;
20             }
21
22             size++;
23         }
24
25         bool isEmpty() const { return size == 0; }
26
27         int getSize() const { return size; }
28
```

```

29     class Iterator {
30     private:
31         Node<T> *current;
32
33     public:
34         Iterator(Node<T> *node) : current(node) {}
35
36         T& operator*() const { return current->data; }
37
38         Iterator& operator++() {
39             if (current)
40                 current = current->next;
41
42             return *this;
43         }
44
45         Iterator& operator--() {
46             if (current)
47                 current = current->prev;
48
49             return *this;
50         }
51
52         bool operator!=(const Iterator &other) const { return current != other.
current; }
53     };
54
55     Iterator begin() const { return Iterator(head); }
56
57     Iterator end() const { return Iterator(nullptr); }
58 };
59

```

```

1  template <typename T>
2  class DoublyLinkedList : public LinkedList<T> {
3  public:
4      void insertEnd(const T& value) {
5          Node<T> *newNode = new Node<T>(value);
6          if (this->head == nullptr) {
7              this->head = newNode;
8              this->tail = newNode;
9          } else {
10             newNode->prev = this->tail;
11             this->tail->next = newNode;
12             this->tail = newNode;
13         }
14
15         this->size++;
16     }
17
18     void insertAtPosition(const T &value, int position) {
19         if (position < 0 || position > this->size)
20             throw std::out_of_range("The position is out of range.");
21
22         if (position == 0)
23             this->insertFront(value);
24         else if (position == this->size)
25             this->insertEnd(value);
26         else {

```

```

27     Node<T> *newNode = new Node<T>(value);
28     Node<T> *current = this->head;
29
30     for (int i = 0; i < position - 1; i++)
31         current = current->next;
32
33     newNode->next = current->next;
34     newNode->prev = current;
35     current->next->prev = newNode;
36     current->next = newNode;
37
38     this->size++;
39 }
40 }
41
42 void deleteNode(const T &value) {
43     Node<T> *current = this->head;
44     while (current != nullptr) {
45         if (current->data == value) {
46             if (current->prev != nullptr)
47                 current->prev->next = current->next;
48             else
49                 this->head = current->next;
50
51             if (current->next != nullptr)
52                 current->next->prev = current->prev;
53             else
54                 this->tail = current->prev;
55
56             delete current;
57
58             this->size--;
59
60             break;
61         }
62
63         current = current->next;
64     }
65 }
66
67 void shuffle() {
68     if (this->size <= 1)
69         std::cerr << "There isn't anything to shuffle!";
70
71     std::vector<T> elements;
72     for (typename DoublyLinkedList<T>::Iterator it = this->begin(); it != this
->end(); ++it)
73         elements.push_back(*it);
74
75     std::random_device rd;
76     std::mt19937 gen(rd());
77     std::shuffle(elements.begin(), elements.end(), gen);
78
79     this->clear();
80     for (const T &element : elements)
81         this->insertEnd(element);
82 }
83
84 void clear() {

```

```

85     while (this->head != nullptr) {
86         Node<T>* temp = this->head;
87         this->head = this->head->next;
88         delete temp;
89     }
90     this->tail = nullptr;
91     this->size = 0;
92 }
93 };
94

```

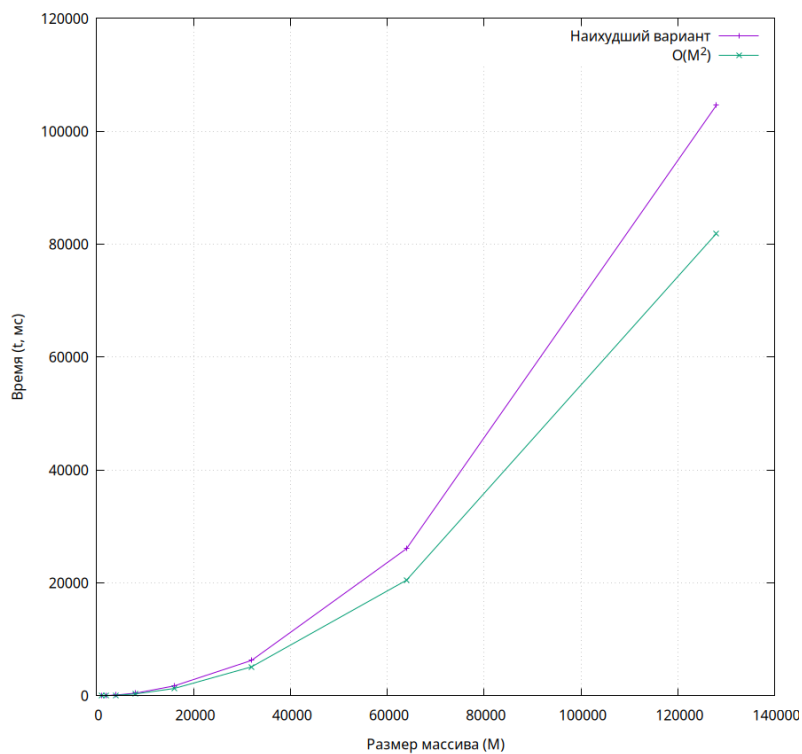
```

1  void testIntegers() {
2      DoublyLinkedList<int> list;
3
4      int sum = 0;
5      int minValue = 1001;
6      int maxValue = -1001;
7
8      for (int i = 0; i < 1000; i++) {
9          int value = getRandomNumber(-1000, 1000);
10         list.insertEnd(value);
11         sum += value;
12
13         if (value < minValue) minValue = value;
14         if (value < maxValue) maxValue = value;
15     }
16
17     double average = static_cast<double>(sum) / list.getSize();
18
19     std::cout << "Test: Integers" << std::endl;
20     std::cout << "Sum: " << sum << std::endl;
21     std::cout << "Average: " << average << std::endl;
22     std::cout << "Min: " << minValue << std::endl;
23     std::cout << "Max: " << maxValue << std::endl;
24 }
25

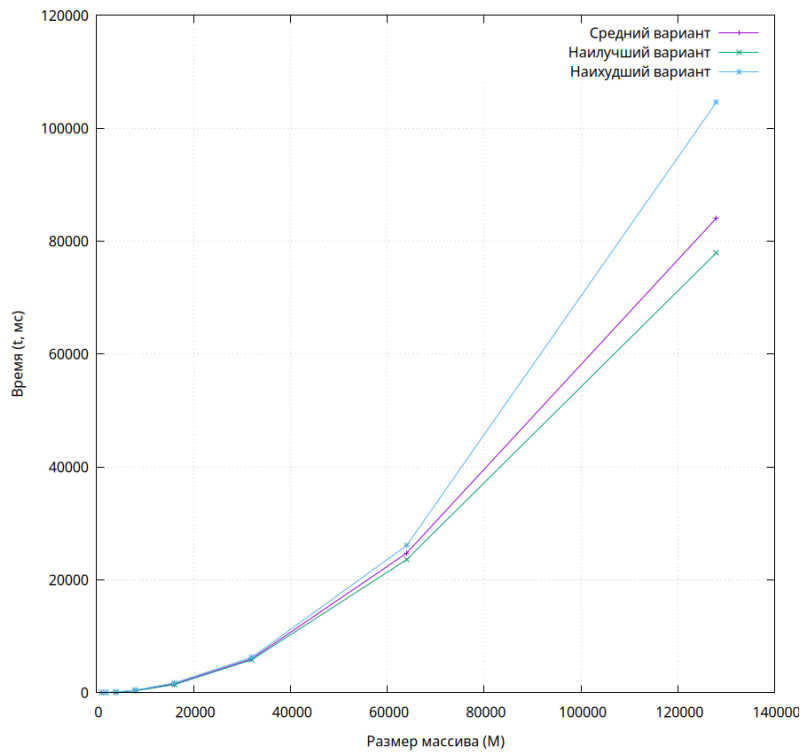
```

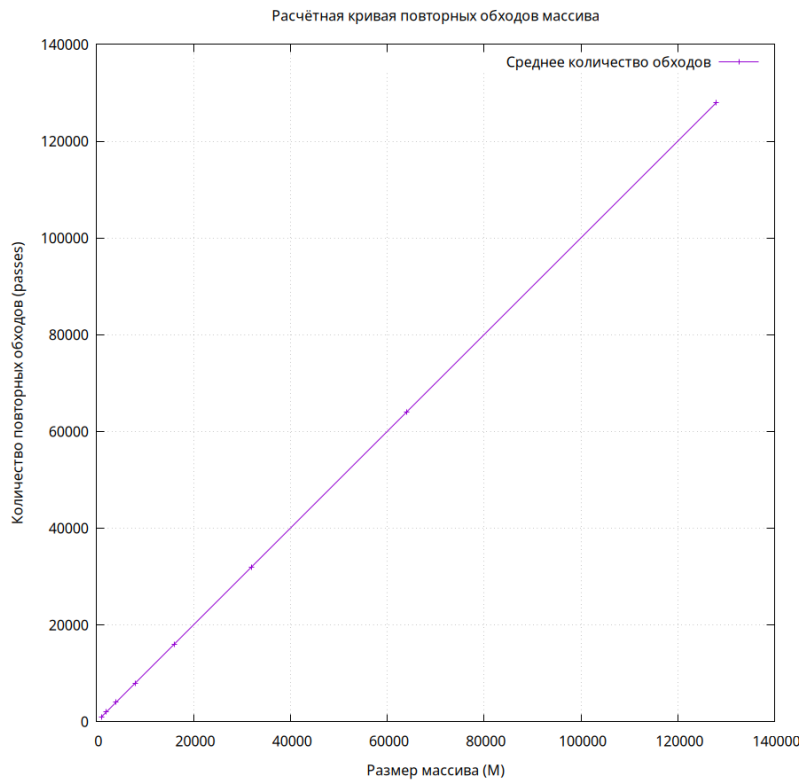
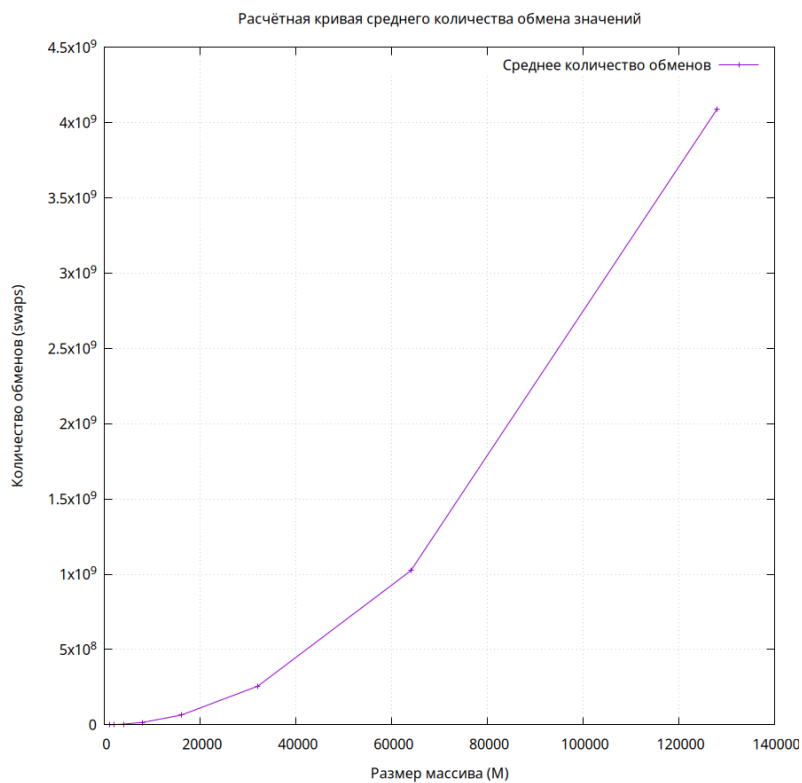


Расчётные кривые наихудшего времени выполнения сортировки и сложности алгоритма



Расчётные кривые среднего, наилучшего и наихудшего времени исполнения





## Выводы

Дву