**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**Отчет**

**По лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Реализация связного списка»  
Вариант: N⁝2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П. А. |
| Преподаватель |  | Пестерев Д. О. |

Санкт-Петербург

2023

1. Постановка задачи

Реализовать структуру данных «односвязный список» и основные функции/методы для работы с ним.

1. Описание реализуемого класса и методов, оценка временной сложности  
     
   Основные структуры

1) struct Node  
Структура, описывающая элемент односвязного списка. Его полями являются: data, который хранит в себе данные целочисленного типа (int); next, хранящий в себе адрес на следующий элемент в списке.   
  
2) struct List  
Структура, описывающая односвязный список. Из полей есть только head, хранящий адрес на первый (головной) элемент списка.

Методы структуры List

1) push\_back(Node\* elem)  
Метод, добавляющий новый элемент в конец списка. Принимает на вход указатель на данные типа Node, на выход ничего не возвращает.  
Временная сложность: Θ (N), N – длина списка.

2) push\_front(Node\* elem)  
Метод, добавляющий новый элемент в начало списка. Принимает на вход указатель на данные типа Node, на выход ничего не возвращает.  
Временная сложность: Θ (1).

3) deleteLast()  
Метод, удаляющий последний элемент списка. Ничего не принимает на вход и ничего не возвращает на выход.  
Временная сложность: Θ (N), N – длина списка.

4) deleteFirst()  
Метод, удаляющий первый элемент списка. Ничего не принимает на вход и ничего не возвращает на выход.  
Временная сложность: Θ (1).

5) insert(unsigned long long index, Node\* elem)  
Метод, вставляющий новый элемент в список перед элементом, который ранее был доступен по заданному индексу. Принимает на вход индекс вставки типа unsigned long long и указатель на данные типа Node, на выход ничего не возвращает. Метод ничего не делает, если индекс вышел за пределы списка.  
Временная сложность: Θ (N), N – длина списка.

6) getItem(unsigned long long index)  
Метод, выдающий элемент списка, находящийся по заданному индексу. На вход принимает индекс поиска типа unsigned long long, на выход возвращает указатель на данные типа Node или нулевой указатель, если поиск не обвенчался успехом.   
Временная сложность: Θ (N), N – длина списка.

7) deleteByIndex(unsigned long long index)  
Метод, удаляющий элемент списка, находящийся по заданному индексу. На вход принимает индекс поиска типа unsigned long long, на выход ничего не возвращает. Метод ничего не делает, если индекс вышел за пределы списка.  
Временная сложность: Θ (N), N – длина списка.

8) getLength()  
Метод, возвращающий длину списка. На вход ничего не принимает, на выход возвращает длину списка типа unsigned long long.  
Временная сложность: Θ (N), N – длина списка.

9) deleteList()  
Метод, удаляющий весь список. Также является описанием деструктора экземпляра структуры. На вход ничего не принимает и ничего не возвращает на выход.  
Временная сложность: Θ (N), N – длина списка.

10) replace(unsigned long long index, Node\* elem)  
Метод, заменяющий элемент в списке по заданному индексу на новый, при это удаляя старый. На вход принимает индекс поиска типа unsigned long long и указатель на данные типа Node, на выход ничего не возвращает. Метод ничего не делает, если индекс вышел за пределы списка.  
Временная сложность: Θ (N), N – длина списка.

11) isEmpty()  
Метод, проверяющий пустоту списка. На вход ничего не принимает, на выход возвращает true, если список пуст, или false, если нет.  
Временная сложность: Θ (1).

12) reverse()  
Метод, переставляющий элементы списка в обратном порядке. На вход ничего не принимает и ничего не возвращает на выход.  
Временная сложность: Θ (N), где N – длина списка.

13) insertList(unsigned long long index, List\* list)  
Метод, вставляющий новый список перед элементом исходного списка, который был ранее доступен по заданному индексу. На вход принимает индекс поиска типа unsigned long long и указатель на данные типа List, на выход ничего не возвращает. Метод ничего не делает, если индекс вышел за пределы списка или вставляемый список пуст.  
Временная сложность: Θ (N + M), N – длина исходного списка, M – длина вставляемого списка.

14) appendList(List\* list)  
Метод, вставляющий новый список в конец исходного. На вход принимает указатель на данные типа List, на выход ничего не возвращает. Метод ничего не делает, если вставляемый список пуст.  
Временная сложность: Θ (N), N – длина исходного списка.

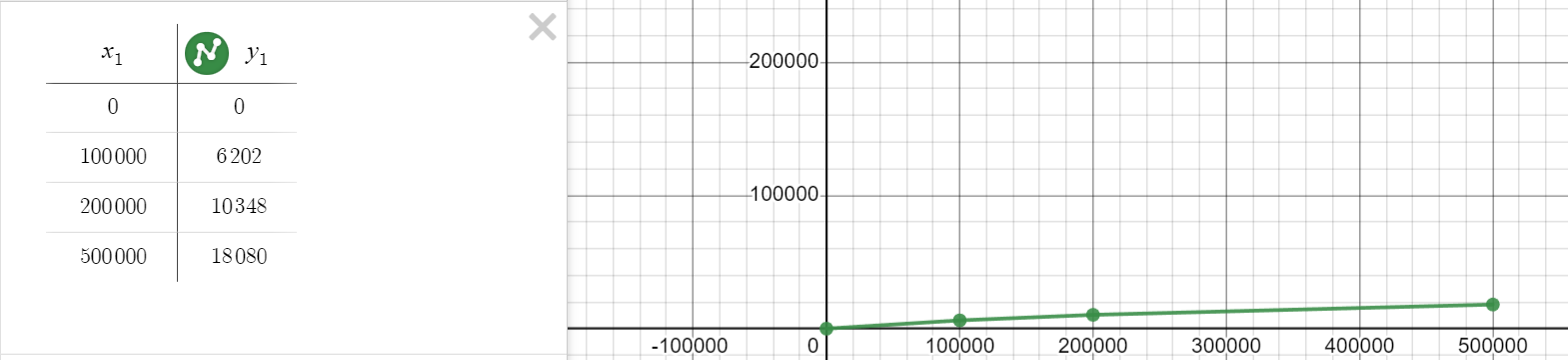
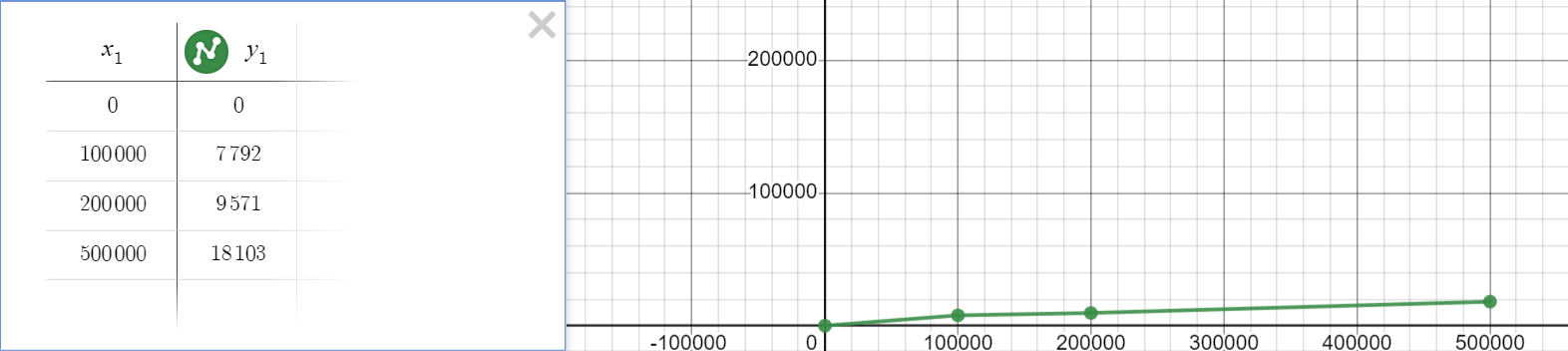
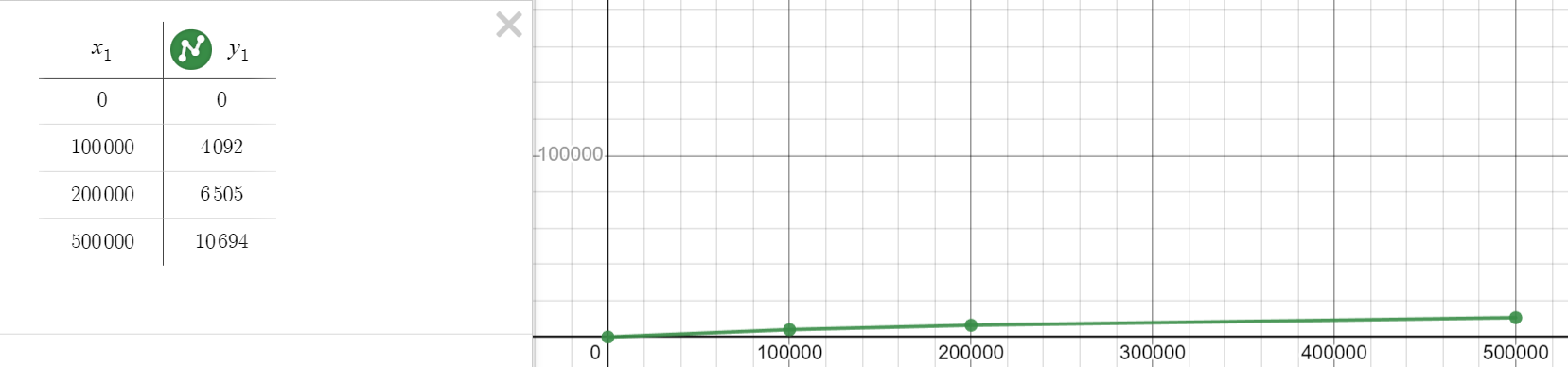
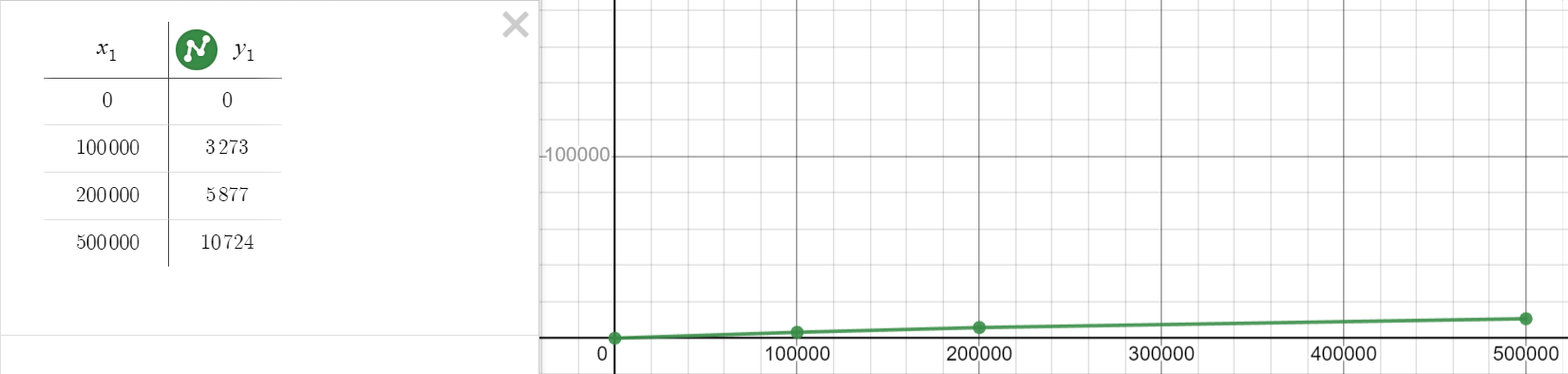
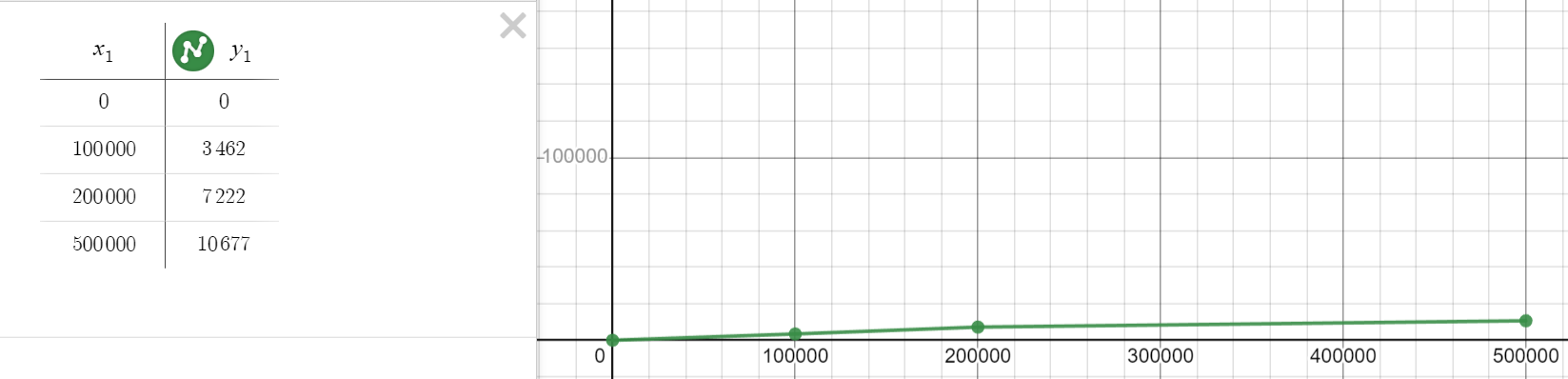
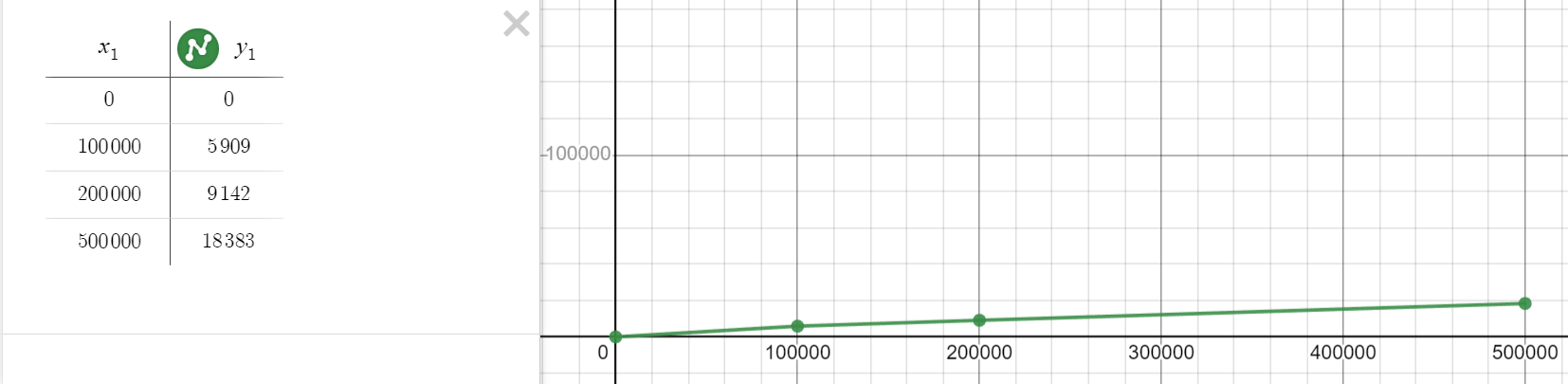
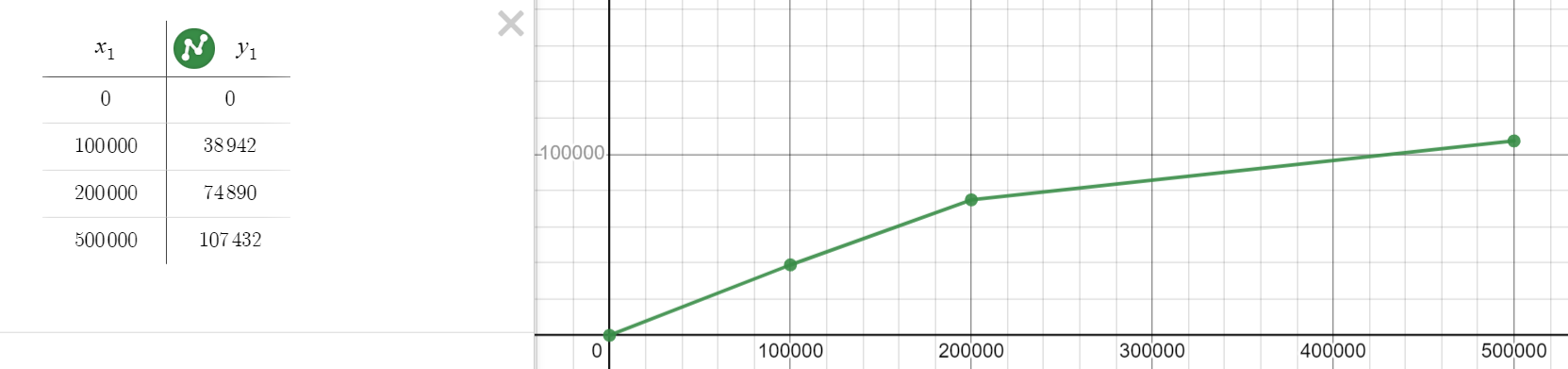
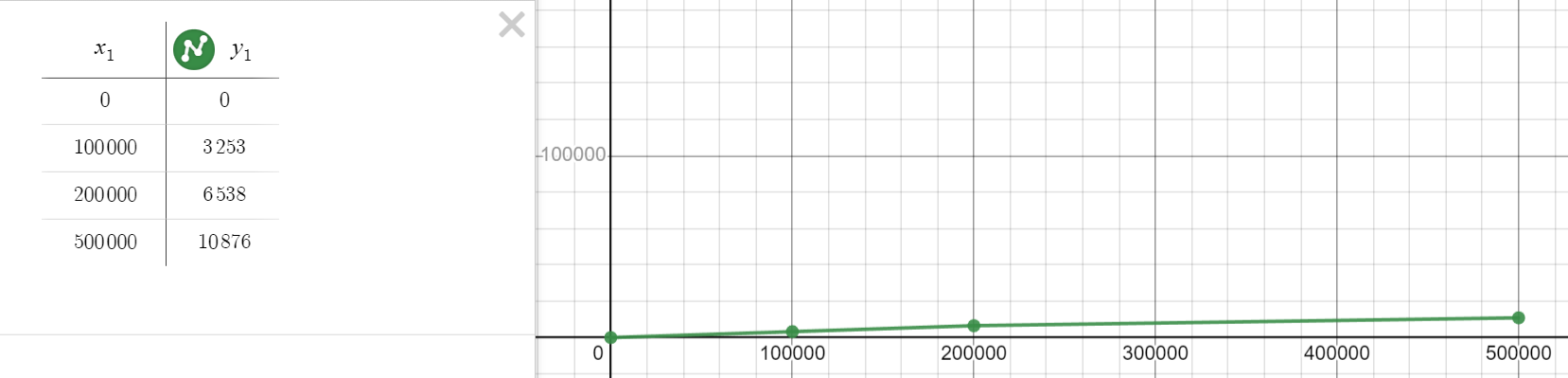
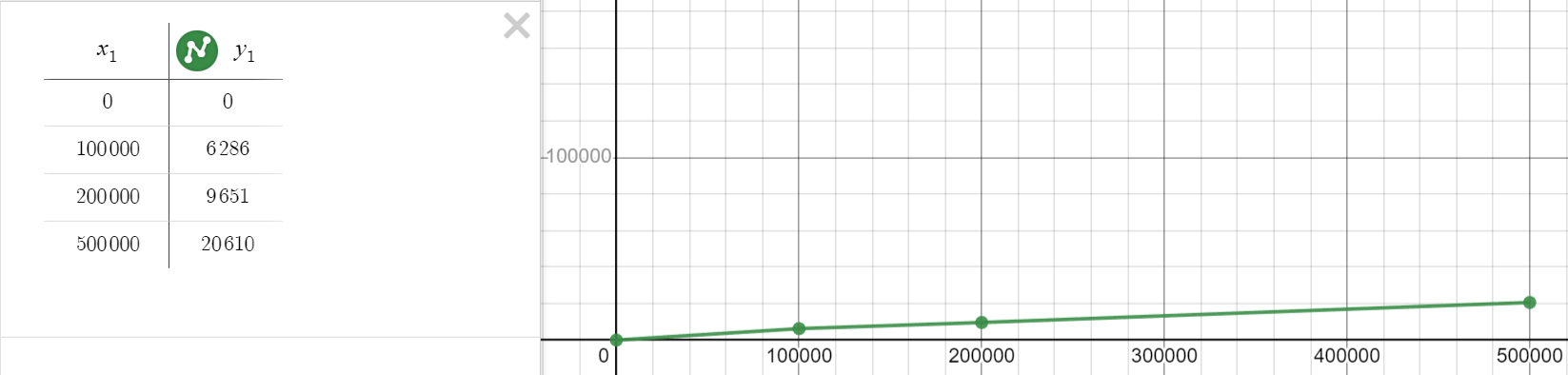
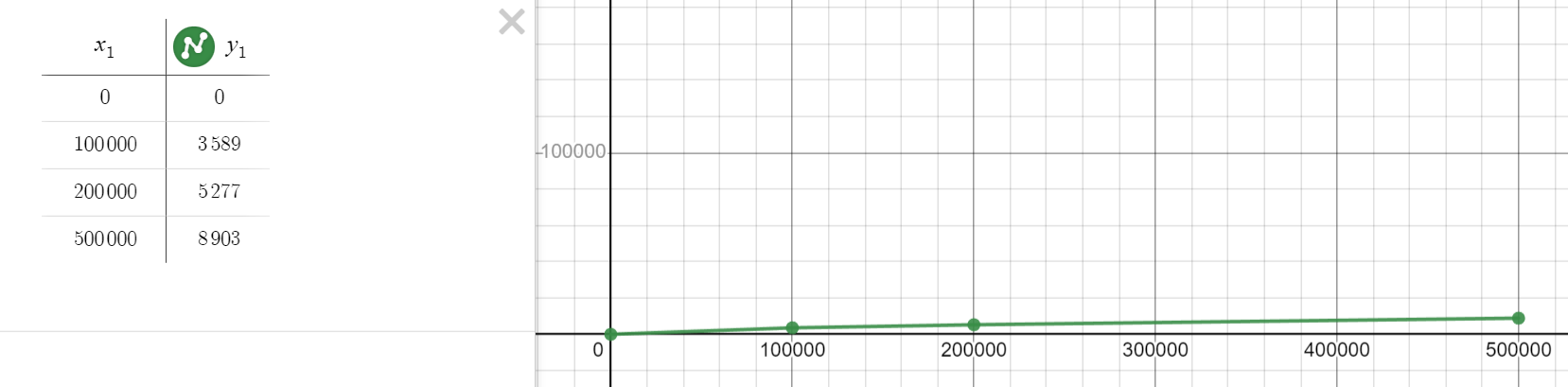
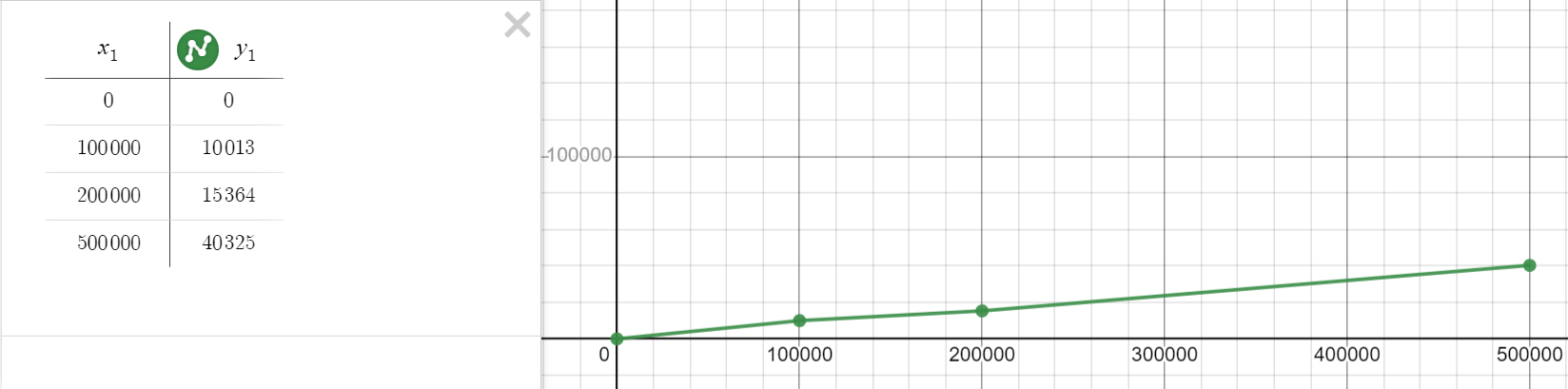
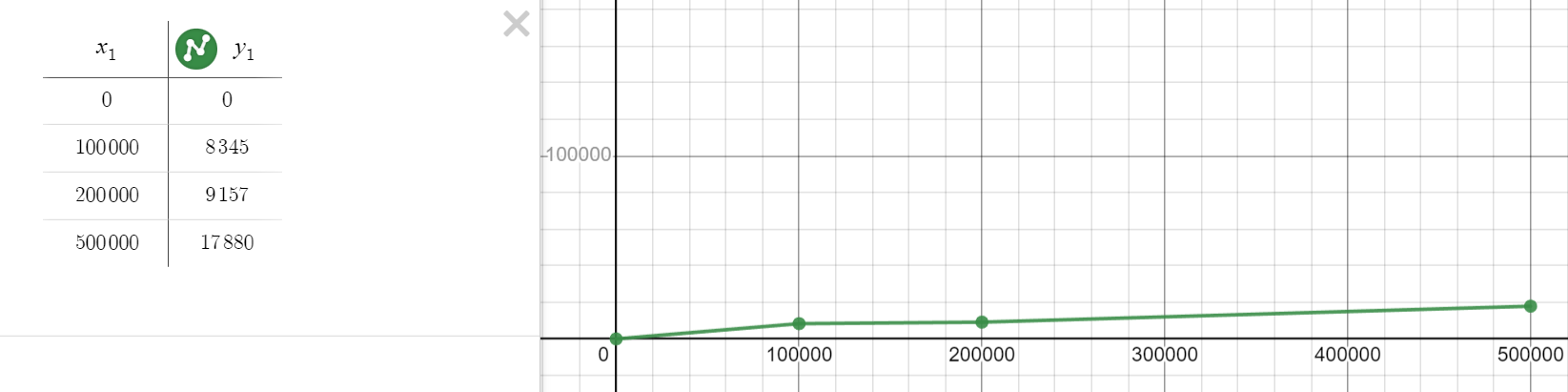
15) frontList(List\* list)  
Метод, вставляющий новый список в начало исходного. На вход принимает указатель на данные типа List, на выход ничего не возвращает. Метод ничего не делает, если вставляемый список пуст.  
Временная сложность: Θ (M), M – длина вставляемого списка.

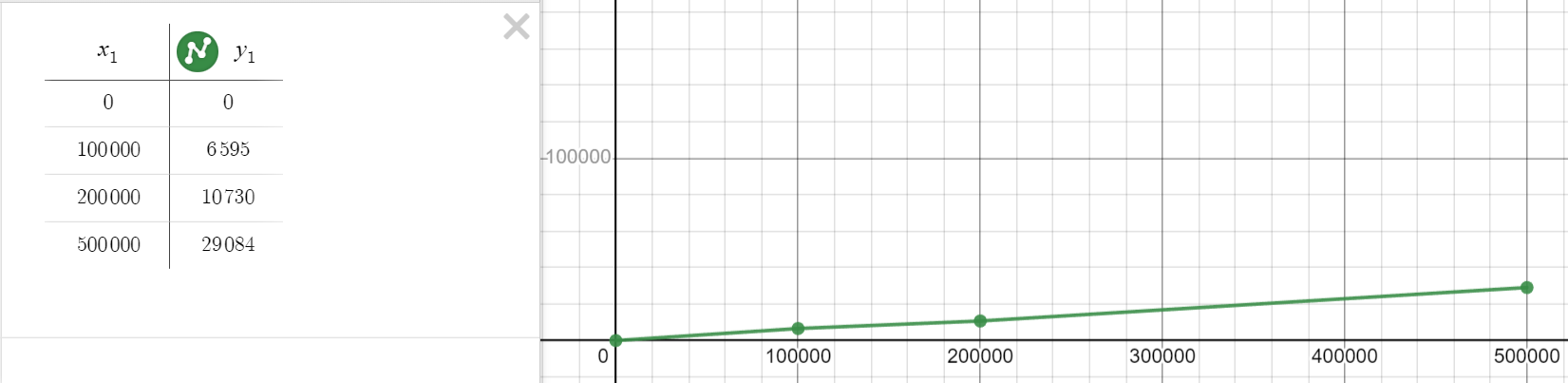
16) isListInside(List\* list)  
Метод, проверяющий, входит ли заданный список в исходный (сравнение по значениям элементов, а не по самим элементам). На вход принимает указатель на данные типа List, на выход возвращает true, если все элементы заданного списка хотя бы один раз последовательно входят в исходный, в противном случае false.  
Временная сложность: Θ ((N – M) \* M), N – длина исходного списка, M – длина искомого списка.

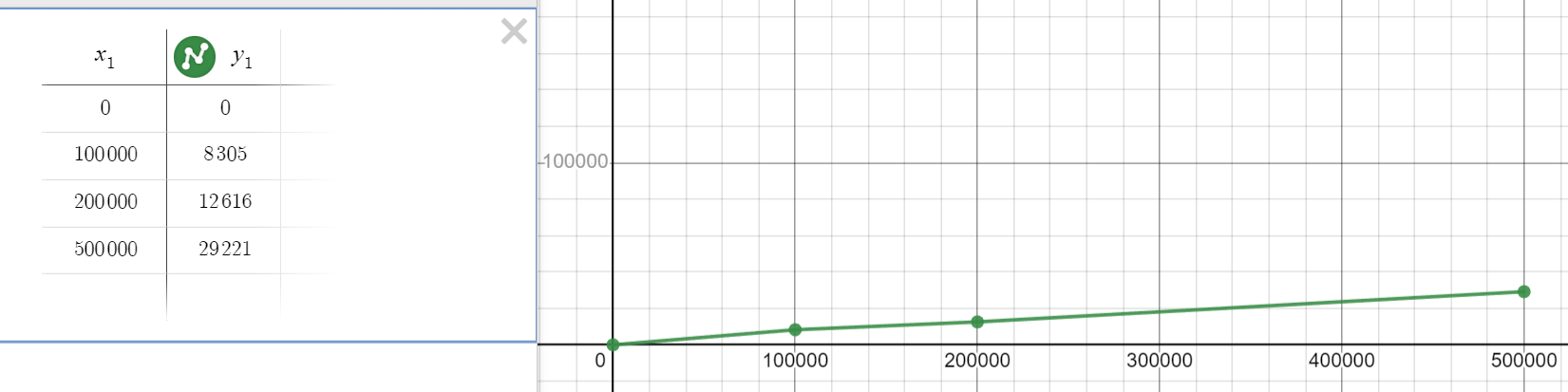
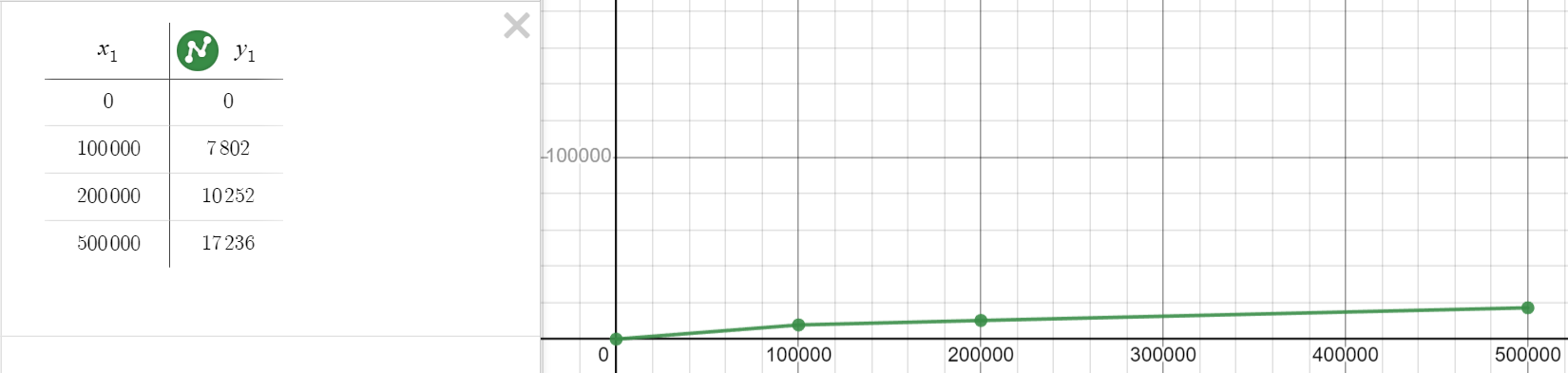
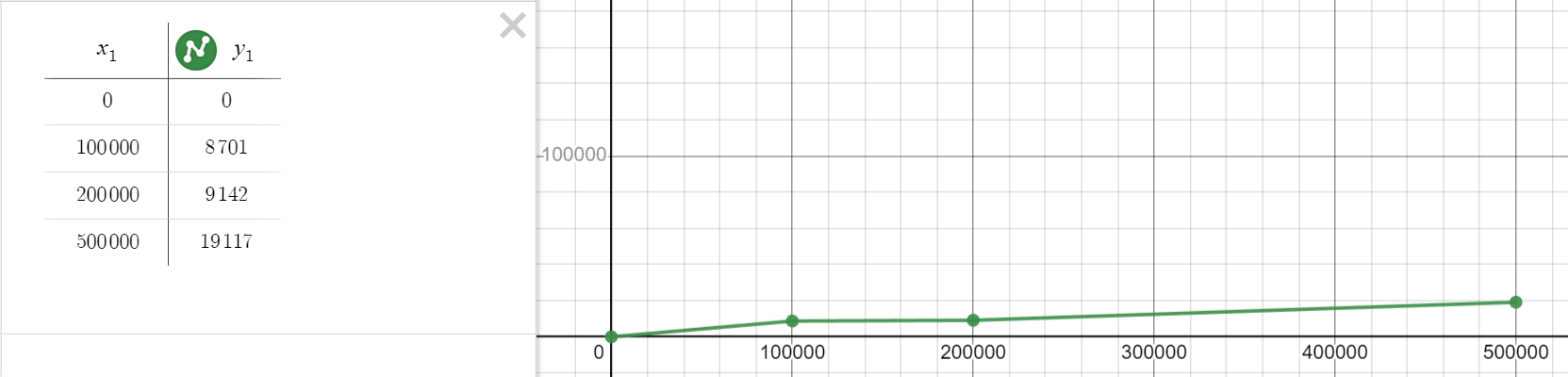
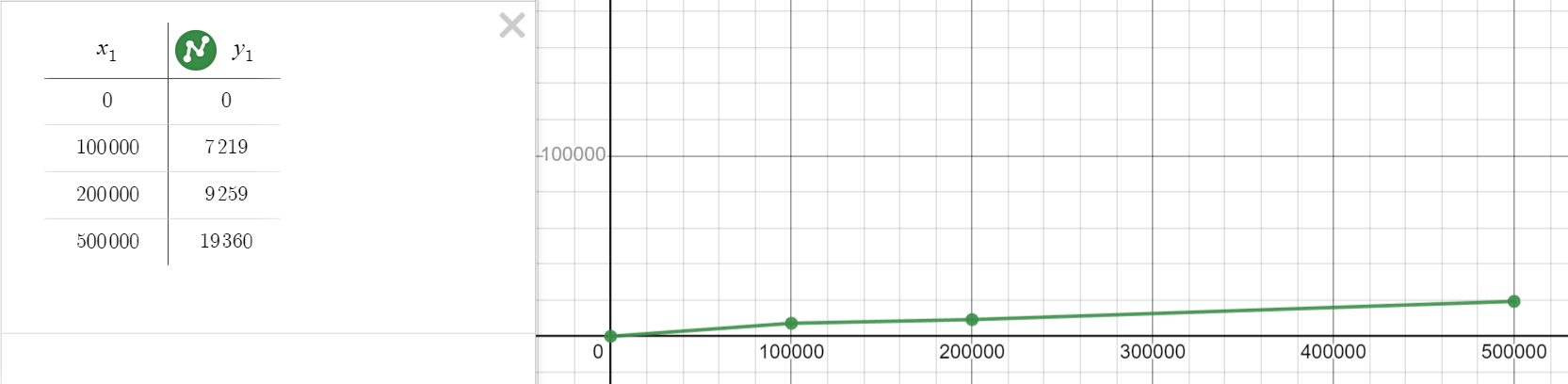
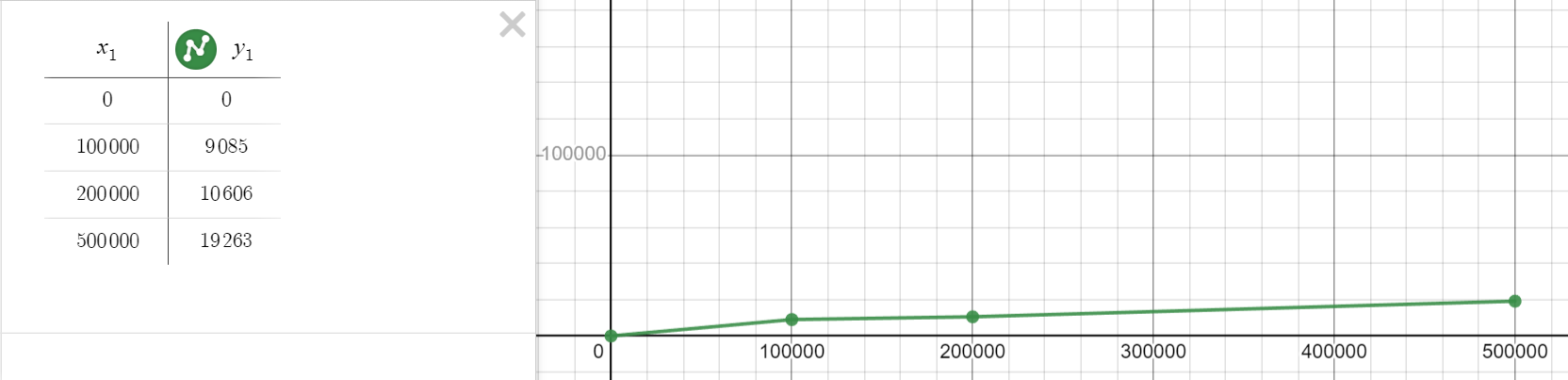
17) searchFirstSubList(List\* list)  
Метод, возвращающий индекс начала первого вхождения заданного списка в исходный. На вход принимает указатель на данные типа List, на выход возвращает индекс начала первого вхождения искомого списка в исходный типа long long, в случаях неудачного поиска возвращает -1.  
Временная сложность: Θ ((N – M) \* M), N – длина исходного списка, M – длина искомого списка.

18) searchLastSubList(List\* list)  
Метод, возвращающий индекс начала последнего вхождения заданного списка в исходный. На вход принимает указатель на данные типа List, на выход возвращает индекс начала последнего вхождения искомого списка в исходный типа long long, в случаях неудачного поиска возвращает -1.  
Временная сложность: Θ ((N – M) \* M), N – длина исходного списка, M – длина искомого списка.

19) swap (unsigned long long index1, unsigned long long index2)  
Метод, обменивающий два элемента в списке по их индексам. На вход принимает индексы поисков исходных элементов типа unsigned long long, на выход ничего не возвращает. Метод ничего не делает, если длина списка не больше 1, индексы равны или хотя бы один из индексов выходит за пределы списка.  
Временная сложность: Θ (N), где N – длина списка.

1. Временная и практическая сложность  
     
   Неконстантные методы были воспроизведены на списках длиной 100000, 200000 и 500000 элементов, и для каждого размера было посчитано время выполнения. Оси абсцисс соответствуют размеры списков, оси ординат — время выполнения для этих списков в микросекундах.
2. push\_back — Θ(N)  
     
     
   Для вставки в конец требуется обход по всему списку, следовательно, время выполнения зависит от длины.
3. deleteLast — Θ(N)  
     
     
   Как и в предыдущем пункте, для удаления последнего элемента требуется обход по всему списку — прямая зависимость от длины.
4. insert — Θ(N)  
     
     
   Вставка осуществлялась по индексу (N / 2) – 1. Наблюдается прямая зависимость от длины списка.
5. getItem — Θ(N)  
     
     
   Получение данных происходило по индексу (N / 2) – 1. Наблюдается прямая зависимость от длины списка.
6. deleteByIndex — Θ(N)  
     
     
   Удаляется элемент с индексом (N / 2) – 1. Наблюдается прямая зависимость от длины списка.
7. getLength — Θ(N)  
     
     
   Подсчет длины требует обхода по всему списку, следовательно, имеется прямая зависимость от длины списка.
8. deleteList — Θ(N)  
     
     
   Алгоритм удаления всего списка задействует удаление первого элемента N раз, а сложность удаления первого элемента — Θ(1), следовательно, получаем прямую зависимость от длины списка.
9. replace — Θ(N)  
     
     
   Замена происходила по индексу (N / 2) – 1. Наблюдается прямая зависимость от длины списка.
10. reverse — Θ(N)  
      
      
    Метод разворачивает весь список, последовательно обходя все его элементы, поэтому его сложность прямо зависит от длины списка.
11. insertList — Θ(N + M)  
      
    При M = 10:  
      
      
      
    При M = N:  
      
      
    Вставка происходит по индексу (N / 2) – 1. Также были взяты списки разных размеров, чтобы показать влияние длины вставляемого списка на временные затраты алгоритма.
12. appendList — Θ(N)  
      
    При M = 10:  
      
       
    При M = N:

  
Вставка в конец не зависит от вставляемого списка, поэтому наблюдается лишь прямая зависимость от длины списка.

1. frontList — Θ(M)  
     
     
   Производилась вставка в начало списка по длине, равной исходному списку; поскольку вставка в начало элемента — константная операция, время затрачивается на обход всего вставляемого списка — прямая зависимость от длины вставляемого списка.
2. isListInside — Θ((N – M) \* M)  
     
     
   Для всех замеров брался подсписок размера 10, находящийся в самом конце списка, и поскольку для каждого элемента происходит обход искомого списка длиной M, общая сложность алгоритма составляет (N – M) \* M.
3. searchFirstSubList — Θ((N – M) \* M)  
     
     
   Метод работает аналогично isListInside, поэтому и сложность та же.
4. searchLastSubList — Θ((N – M) \* M)  
     
     
   Метод работает аналогично isListInside и searchFirstSubList, поэтому и сложность та же.
5. swap — Θ(N)  
     
     
   Для замеров обменивались первый и последний элемент; чтобы обменять элементы, необходимо сначала обойти список до этих элементов — прямая зависимость от длины списка.
6. **Листинг**

**Node.h**

#pragma once

struct Node {

int val;

Node\* next;

Node(int val);

~Node();

};

**Node.cpp**

#include "Node.h"

Node::Node(int val = 0) {

this->val = val;

this->next = nullptr;

}

Node::~Node() {

this->val = 0;

this->next = nullptr;

}

**List.h**

#pragma once

#include "Node.h"

struct List {

Node\* head;

List();

~List();

unsigned long long getLength(); // №8

Node\* getItem(unsigned long long index); // №6

bool isEmpty(); // №11

void push\_back(Node\* elem); // №1

void push\_front(Node\* elem); // №2

void insert(unsigned long long index, Node\* elem); // №5

void appendList(List\* list); // №14

void frontList(List\* list); // №15

void insertList(unsigned long long index, List\* list); // №13

void deleteLast(); // №3

void deleteFirst(); // №4

void deleteByIndex(unsigned long long index); // №7

void deleteList(); // №9

void replace(unsigned long long index, Node\* elem); // №10

void swap(unsigned long long index1, unsigned long long index2); // №19

void reverse(); // №12

bool isListInside(List\* list); // №16

long long searchFirstSublist(List\* list); // №17

long long searchLastSublist(List\* list); // №18

};

**List.cpp**

#include "List.h"

List::List() {

this->head = nullptr;

}

List::~List() {

this->deleteList();

}

unsigned long long List::getLength() {

unsigned long long length = 0;

Node\* tmp = this->head;

while (tmp != nullptr) {

tmp = tmp->next;

++length;

}

return length;

}

Node\* List::getItem(unsigned long long index) {

Node\* tmp = this->head;

unsigned long long length = 0;

while (tmp != nullptr) {

if (length == index) return tmp;

tmp = tmp->next;

++length;

}

return nullptr;

}

bool List::isEmpty() {

return this->head == nullptr;

}

void List::push\_back(Node\* elem) {

if (this->isEmpty()) {

this->head = elem;

return;

}

Node\* tmp = this->head;

while (tmp->next != nullptr) tmp = tmp->next;

tmp->next = elem;

}

void List::push\_front(Node\* elem) {

elem->next = this->head;

this->head = elem;

}

void List::insert(unsigned long long index, Node\* elem) {

if (this->isEmpty() && index != 0) return;

if (index == 0) {

this->push\_front(elem);

return;

}

Node\* cur = this->head, \*prev = nullptr;

unsigned long long len = 0;

while (cur != nullptr && len != index) {

prev = cur;

cur = cur->next;

++len;

}

if (cur == nullptr) return;

prev->next = elem;

elem->next = cur;

}

void List::appendList(List\* list) {

if (list->isEmpty()) return;

if (this->isEmpty()) {

this->head = list->head;

return;

}

Node\* tmp = this->head, \* prev = nullptr;

while (tmp != nullptr) {

prev = tmp;

tmp = tmp->next;

}

prev->next = list->head;

}

void List::frontList(List\* list) {

if (list->isEmpty()) return;

Node\* tmp = list->head;

while (tmp->next != nullptr) {

tmp = tmp->next;

}

tmp->next = this->head;

this->head = list->head;

}

void List::insertList(unsigned long long index, List\* list) {

if (list->isEmpty()) return;

if (this->isEmpty() && index != 0) return;

if (index == 0) {

this->frontList(list);

return;

}

Node\* cur = this->head, \*prev = nullptr;

unsigned long long len = 0;

while (cur != nullptr && len != index) {

prev = cur;

cur = cur->next;

++len;

}

if (cur == nullptr) return;

Node\* other\_cur = list->head;

while (other\_cur->next != nullptr) other\_cur = other\_cur->next;

prev->next = list->head;

other\_cur->next = cur;

}

void List::deleteLast() {

if (this->isEmpty()) return;

if (this->head->next == nullptr) {

delete this->head;

this->head = nullptr;

return;

}

Node\* tmp = this->head;

while (tmp->next->next != nullptr) {

tmp = tmp->next;

}

delete tmp->next;

tmp->next = nullptr;

}

void List::deleteFirst() {

if (this->isEmpty()) return;

Node\* tmp = this->head->next;

delete this->head;

this->head = tmp;

}

void List::deleteByIndex(unsigned long long index) {

if (this->isEmpty()) return;

if (index == 0) {

this->deleteFirst();

return;

}

Node\* cur = this->head, \*prev = nullptr;

unsigned long long len = 0;

while (cur != nullptr && len != index) {

prev = cur;

cur = cur->next;

++len;

}

if (cur == nullptr) return;

prev->next = cur->next;

delete cur;

}

void List::deleteList() {

while (!this->isEmpty()) this->deleteFirst();

}

void List::replace(unsigned long long index, Node\* elem) {

if (this->isEmpty()) return;

Node\* cur = this->head;

if (index == 0) {

elem->next = cur->next;

this->head = elem;

delete cur;

return;

}

Node\* prev = nullptr;

unsigned long long len = 0;

while (cur != nullptr && len != index) {

prev = cur;

cur = cur->next;

++len;

}

if (cur == nullptr) return;

prev->next = elem;

elem->next = cur->next;

delete cur;

}

void List::swap(unsigned long long index1, unsigned long long index2) {

if (this->isEmpty() || index1 == index2 || this->head->next == nullptr) return;

Node\* prev1 = nullptr, \* toSwap1 = this->head;

unsigned long long len1 = 0;

while (toSwap1 != nullptr && len1 != index1) {

prev1 = toSwap1;

toSwap1 = toSwap1->next;

++len1;

}

Node\* prev2 = nullptr, \* toSwap2 = this->head;

unsigned long long len2 = 0;

while (toSwap2 != nullptr && len2 != index2) {

prev2 = toSwap2;

toSwap2 = toSwap2->next;

++len2;

}

if (toSwap1 == nullptr || toSwap2 == nullptr) return;

if (prev1 == nullptr) this->head = toSwap2;

else prev1->next = toSwap2;

if (prev2 == nullptr) this->head = toSwap1;

else prev2->next = toSwap1;

Node\* tmp = toSwap2->next;

toSwap2->next = toSwap1->next;

toSwap1->next = tmp;

}

void List::reverse() {

Node\* cur = this->head;

Node\* prev = nullptr, \* next = nullptr;

while (cur != nullptr) {

next = cur->next;

cur->next = prev;

prev = cur;

cur = next;

}

this->head = prev;

}

bool List::isListInside(List\* list) {

if (this->isEmpty() && list->isEmpty()) return true;

if (this->isEmpty() || list->isEmpty()) return false;

Node\* cur1 = this->head, \*cur2 = list->head;

Node\* tmp;

while (cur1 != nullptr) {

tmp = cur1;

while (cur2 != nullptr) {

if (tmp == nullptr) return false;

if (tmp->val == cur2->val) {

tmp = tmp->next;

cur2 = cur2->next;

}

else break;

}

if (cur2 == nullptr) return true;

cur2 = list->head;

cur1 = cur1->next;

}

return false;

}

long long List::searchFirstSublist(List\* list) {

if (this->isEmpty() && list->isEmpty()) return -1;

if (this->isEmpty() || list->isEmpty()) return -1;

long long index = 0;

Node\* cur1 = this->head, \* cur2 = list->head;

Node\* tmp;

while (cur1 != nullptr) {

tmp = cur1;

while (cur2 != nullptr) {

if (tmp == nullptr) return -1;

if (tmp->val == cur2->val) {

tmp = tmp->next;

cur2 = cur2->next;

}

else break;

}

if (cur2 == nullptr) return index;

cur2 = list->head;

cur1 = cur1->next;

++index;

}

return -1;

}

long long List::searchLastSublist(List\* list) {

if (this->isEmpty() && list->isEmpty()) return -1;

if (this->isEmpty() || list->isEmpty()) return -1;

long long curIndex = 0, lastIndex = -1;

Node\* cur1 = this->head, \* cur2 = list->head;

Node\* tmp;

while (cur1 != nullptr) {

tmp = cur1;

while (cur2 != nullptr) {

if (tmp == nullptr) return -1;

if (tmp->val == cur2->val) {

tmp = tmp->next;

cur2 = cur2->next;

}

else break;

}

if (cur2 == nullptr) lastIndex = curIndex;

cur2 = list->head;

cur1 = cur1->next;

++curIndex;

}

return lastIndex;

}

**main.cpp**

#include <iostream>

#include <locale>

#include <chrono>

#include "List.h"

using namespace std::chrono;

void print\_list(List\* list) {

Node\* tmp = list->head;

while (tmp != nullptr) {

std::cout << tmp->val << " ";

tmp = tmp->next;

}

std::cout << "\n";

}

int main() {

std::setlocale(LC\_ALL, "RUS");

std::cout << "Программа по работе с линейным односвзяным списком\nАвтор: Савин Павел, группа 2309\n\n";

short com;

int a;

unsigned long long index1, index2, num;

List\* list = new List;

List\* tmp\_list = nullptr;

Node\* tmp = nullptr;

bool exit = false;

bool flag;

do {

std::cout << "Что хотите сделать?\n1 - Вставить элемент в конец списка\n2 - Вставить элемент в начало списка\n";

std::cout << "3 - Удалить последний элемент\n4 - Удалить первый элемент\n5 - Добавить элемент по индексу\n6 - Получить элемент по индексу\n";

std::cout << "7 - Удалить элемент по индексу\n8 - Получить размер списка\n9 - Удалить все элементы списка\n10 - Заменить элемент по индексу\n";

std::cout << "11 - Узнать, пустой ли список\n12 - Развернуть список\n13 - Вставить в список другой список, начиная с индекса\n";

std::cout << "14 - Вставить в конец списка другой список\n15 - Вставить в начало списка другой список\n16 - Проверить вхождение другого списка в список\n";

std::cout << "17 - Получить индекс первого вхождения другого списка в список\n18 - Получить индекс последнего вхождения другого списка в список\n";

std::cout << "19 - Поменять элементы местами\n0 - Выйти из программы\nВведите номер команды: ";

std::cin >> com;

if (std::cin.fail()) {

std::cout << "Введена некорректная команда! Попробуйте еще раз!\n\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

if (com == 0) {

exit = true;

}

if (com == 1) {

do {

std::cout << "\nВведите значение нового элемента (целое число): ";

std::cin >> a;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

tmp = new Node(a);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->push\_back(tmp);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 2) {

do {

std::cout << "\nВведите значение нового элемента (целое число): ";

std::cin >> a;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

tmp = new Node(a);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->push\_front(tmp);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 3) {

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->deleteLast();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 4) {

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->deleteFirst();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 5) {

do {

std::cout << "\nВведите индекс вставки нового элемента (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

std::cout << "Введите значение нового элемента (целое число): ";

std::cin >> a;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

tmp = new Node(a);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->insert(index1, tmp);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 6) {

do {

std::cout << "\nВведите индекс искомого элемента (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

auto start = high\_resolution\_clock::now();

tmp = list->getItem(index1);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

if (tmp != nullptr) std::cout << "Полученный элемент по индексу: " << tmp->val << "\n";

else std::cout << "Элемент не был найден.\n";

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 7) {

do {

std::cout << "\nВведите индекс удаляемого элемента (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->deleteByIndex(index1);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 8) {

auto start = high\_resolution\_clock::now();

num = list->getLength();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Текущая длина списка: " << num << "\n";

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 9) {

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->deleteList();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 10) {

do {

std::cout << "\nВведите индекс замены на новый элемент (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

std::cout << "Введите значение нового элемента (целое число): ";

std::cin >> a;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

tmp = new Node(a);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->replace(index1, tmp);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 11) {

auto start = high\_resolution\_clock::now();

flag = list->isEmpty();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список " << (flag ? "пуст.\n" : "не пуст.\n");

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 12) {

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->reverse();

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (com == 13) {

tmp\_list = new List;

do {

std::cout << "\nВведите индекс вставки нового списка (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

break;

}

while (true);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->insertList(index1, tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 14) {

tmp\_list = new List;

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->appendList(tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 15) {

tmp\_list = new List;

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->frontList(tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 16) {

tmp\_list = new List;

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

auto start = high\_resolution\_clock::now();

flag = list->isListInside(tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Введенный список " << (flag ? "присутствует в исходном." : "отсутствует в исходном.") << "\n";

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 17) {

tmp\_list = new List;

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

auto start = high\_resolution\_clock::now();

num = list->searchFirstSublist(tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

if (num == -1) std::cout << "Введенный список не найден или списки пустые.\n";

else std::cout << "Индекс первого вхождения введенного списка в исходный: " << num << "\n";

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 18) {

tmp\_list = new List;

std::cout << "Введите через пробел элементы нового списка (целые числа).\nЧтобы окончить список, введите что угодно не число: ";

while (std::cin >> a) {

tmp = new Node(a);

tmp\_list->push\_back(tmp);

}

auto start = high\_resolution\_clock::now();

num = list->searchLastSublist(tmp\_list);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

if (num == -1) std::cout << "Введенный список не найден или списки пустые.\n";

else std::cout << "Индекс последнего вхождения введенного списка в исходный: " << num << "\n";

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

tmp\_list = nullptr;

}

if (com == 19) {

do {

std::cout << "\nВведите индекс первого элемента на перестановку (целое неотрицательное число): ";

std::cin >> index1;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

std::cout << "Введите индекс второго элемента на перестановку (целое неотрицательное число): ";

std::cin >> index2;

if (std::cin.fail()) {

std::cout << "Введены некорректные данные! Попробуйте еще раз!\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

continue;

}

break;

}

while (true);

std::cout << "Список до выполнения функции: ";

print\_list(list);

auto start = high\_resolution\_clock::now();

list->swap(index1, index2);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Список после выполнения функции: ";

print\_list(list);

std::cout << "Время выполнения функции (в микросекундах): " << duration.count() << "\n\n";

}

if (exit) {

std::cout << "Программа завершила свою работу.";

break;

}

std::cin.clear();

std::cin.ignore(32767, '\n');

}

while (true);

delete list;

return 0;

}

Ссылка на репозиторий: <https://github.com/pashkev14/A_DS_Lab1>