**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: 17**

| Студент гр. 2309 |  | Шуббе Л. П. |
| --- | --- | --- |
| Преподаватель |  | Пестерев Д. О. |

Санкт-Петербург

2023

**1. Постановка задачи**

Реализовать объект в виде двусвязного списка с набором методов/функций.

**2. Описание реализуемых классов, методов и функций  
  
Классы**

**1) Element**Представляет из себя элемент будущего списка. В качестве данных хранит значение (любого типа), объявляемое при инициализации и ссылки на предыдущий и следующий элемент.  
  
**2) List2**Представляет из себя двусвязный список, хранящий в себе ссылки на первый (head), последний (tail) и текущий (cur) элемент, а также переменную, указывающую на пустоту списка (empty).

**Методы класса Element**

**1) set(val)**Устанавливает значение данного элемента равное val.

**2) get()**  
Получает значение данного элемента.

**3) copy()**  
Возвращает элемент со значением равным значению данного элемента.

**4) \_\_eq\_\_()**  
Необходимо для корректного сравнения двух элементов. Проверяет other на тип и производит проверку на равенство значений.

**Методы класса List2**

**1) add\_bottom(el)**Добавляет новый элемент в конец списка. Принимает на вход элемент (или значение элемента для создания), который необходимо добавить. Ничего не возвращает.

**2) add\_top(el)**Добавляет новый элемент в начало списка. Принимает на вход элемент (или значение элемента для создания), который необходимо добавить. Ничего не возвращает.

**3) delete\_bottom()**Удаляет последний элемент списка. Ничего не возвращает.

**4) delete\_top()**Удаляет первый элемент списка. Ничего не возвращает.

**5) add\_by\_index(index, el)**Вставляет новый элемент в список по индексу (вставка перед элементом, который был ранее доступен по этому индексу). Принимает на вход индекс и элемент (или значение элемента для создания), который необходимо добавить. Ничего не возвращает.

**6) get\_by\_index(index)**Метод для получения элемента по индексу. Принимает на вход этот индекс. Возвращает элемент (типа Element), находящийся по этому индексу.

**7) delete\_by\_index(index)**Метод для удаления элемента по индексу. Принимает на вход этот индекс. Ничего не возвращает.

**8) get\_length()**Метод, используемый для получения длины списка.  
Ничего не принимая, возвращает длину списка.

**9) delete\_all()**Метод, удаляющий весь список целиком. Ничего не принимает на вход и ничего не возвращает.

**10) replace\_by\_index(index, el)**Метод для замены элемента списка на передаваемый. Принимает на вход индекс и элемент (или значение элемента для создания), на который необходимо заменить. Ничего не возвращает.

**11) is\_empty()**Метод, используемый для проверки списка на пустоту. Не принимает на вход аргументы.  
Данный метод возвращает True, если список пуст или False, если нет.

**12) invert()**Метод, используемый для размещения элементов списка в обратном порядке.  
Ничего не принимает на вход и ничего не возвращает.

**13) merge\_by\_index(index, l2)**Метод, используемый для вставки другого списка начиная с индекса.  
Принимает на вход индекс, где будет вставлен список и список для вставки. Ничего не возвращает.

**14) merge\_top(l2)**Метод, используемый для вставки другого списка в начало.  
Принимает на вход список для вставки. Ничего не возвращает.

**15) merge\_bottom(l2)**Метод, используемый для вставки другого списка в конец.  
Принимает на вход список для вставки. Ничего не возвращает.

**16) contains(l2)**Метод, используемый для проверки наличия вхождения списка в другой.  
Принимает на вход список, нахождение которого необходимо проверить в текущем.  
Данный метод возвращает False если вхождения не существует или True если оно существует.

**17) find(l2)**Метод, используемый для поиска первого вхождения списка в другой.  
Принимает на вход список, нахождение которого необходимо проверить в текущем.  
Данный метод возвращает номер индекса, с которого один список входит в другой или -1 если вхождения не существует.

**18) rfind(l2)**  
Метод, используемый для поиска последнего вхождения списка в другой.  
Принимает на вход список, нахождение которого необходимо проверить в текущем.  
Данный метод возвращает номер индекса, с которого один список входит в другой или -1 если вхождения не существует.

**19) swap\_by\_index(index1, index2)**Метод, используемый для обмена двумя элементами списка по индексам. Принимает на вход эти два индекса и ничего не возвращает.

**20) add\_first(el)**  
Метод для добавления первого элемента в пустой список. Принимает на вход элемент или его значение. Добавляет элемент и меняет переменную is\_empty.

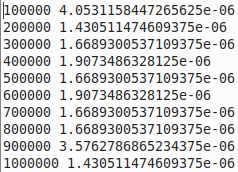
**21) print()**  
Метод для вывод списка на экран. Ничего не принимает на вход и ничего не возвращает. Вывод осуществляется стандартным print через пробел.

**Другие функции**

**1) copy(l2)**  
Функция для копирования некоторого списка. Принимает на вход список для копирования. Создает копию каждого элемента и возвращает ссылку на новый список из этих элементов.

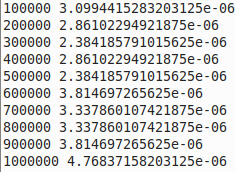
**2) generate(N, min\_el, max\_el)**  
Функция для генерации случайного списка. Принимает на вход желаемое количество элементов в списке, минимальный и максимальный возможный элемент. Генерирует элементы со значениями от минимального до максимального (включая обе границы) и возвращает ссылку на новый список из этих элементов.

**3) get\_time(func, \*args)**  
Функция для определения времени выполнения некоторой функции. Принимает на вход функцию и аргументы. Вызывает эту функцию с переданными аргументами и возвращает время в секундах, затраченное на выполнение.

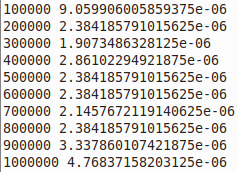
**3. Временная и практическая сложность**Все методы были воспроизведены и время их выполнения было зафиксировано на списках длиной от 105 до 106 с шагом 105 при элементах списка, сгенерированных случайно в диапазоне от 0 до 106. Если некоторый метод требует некоторый индекс или другой список, то они также генерировались случайно (индекс — так чтобы он попадал в список, а другие списки — так же как и оригинальный, с той же длиной). Для методов с неконстантной асимптотикой были построены графики зависимости времени выполнения от размера списка.

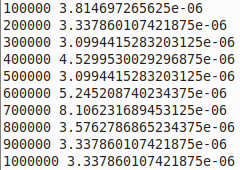
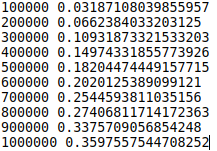
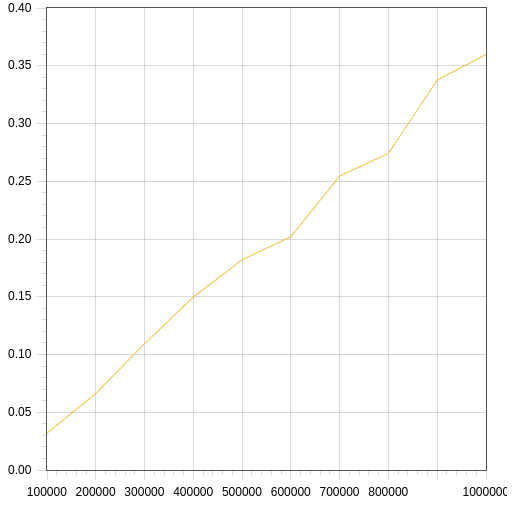
1. add\_bottom – O(1)

Изменяется ссылка на tail и соответствующие ссылки у вставляемого элемента и предыдущего tail поэтому зависимости от N не будет.

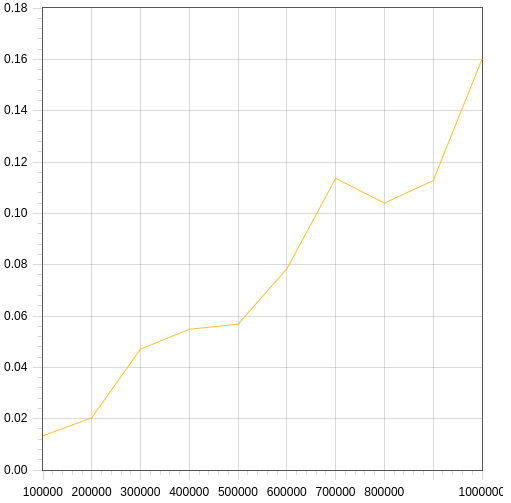
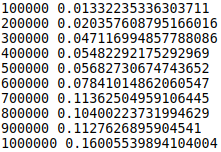


1. add\_top – O(1)  
   Изменяется ссылка на head и соответствующие ссылки у вставляемого элемента и предыдущего head поэтому зависимости от N не будет.

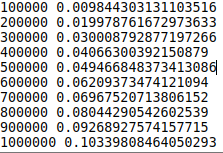
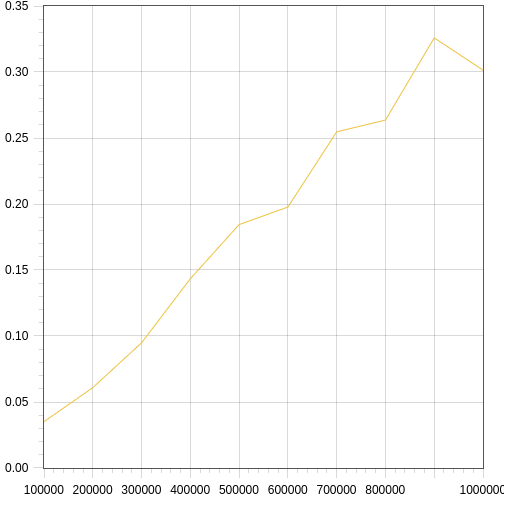
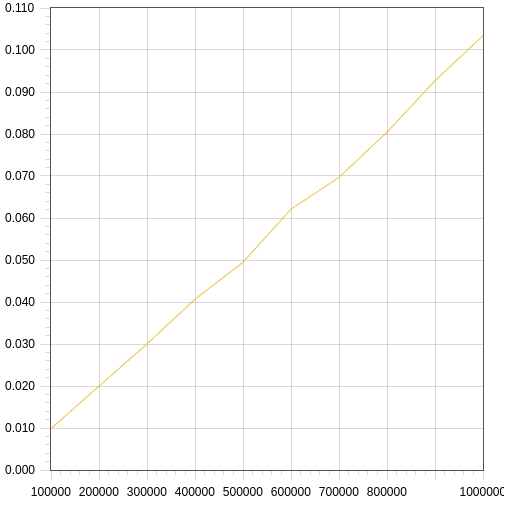
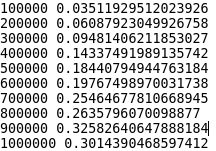


1. delete\_bottom - O(1)  
   Изменяется ссылка на tail и соответствующие ссылки у удаляемого элемента и нового tail поэтому зависимости от N не будет.
2. delete\_top - O(1)  
   Изменяется ссылка на head и соответствующие ссылки у удаляемого элемента и нового head поэтому зависимости от N не будет.
3. add\_by\_index - O(N)  
     
     
   

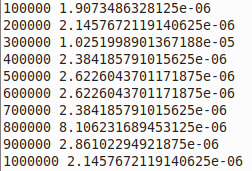
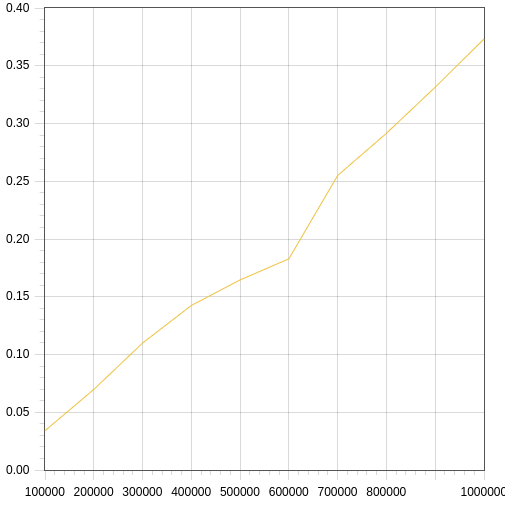
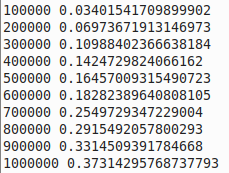
Метод вызывает get\_length и get\_by\_index после чего осуществляет константное количество операций присваивания поэтому асимптотика будет той же — O(N). График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.

1. get\_by\_index - O(N)  
     
     
   

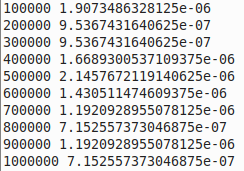
Метод вызывает get\_length, а затем в цикле совершает по одной операции присваивания, всего index раз. Поэтому асимптотика будет O(N).График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.

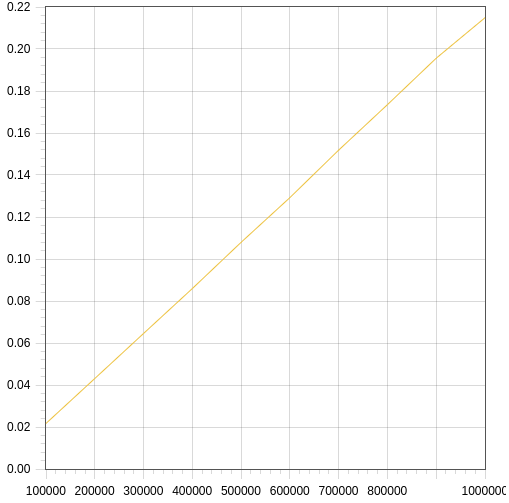
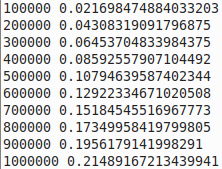
1. delete\_by\_index - O(N)  
     
     
   Метод вызывает get\_length и get\_by\_index после чего осуществляет константное количество операций присваивания, поэтому асимптотика будет той же — O(N). График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.
2. get\_length - O(N)  
     
     
   

Метод осуществляет полное прохождение по списку с начала до конца (пока ссылки на следующий элемент не окажется), поэтому время линейно зависит от длины.

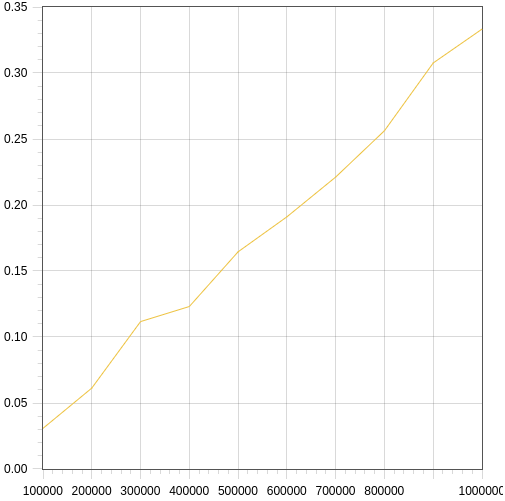
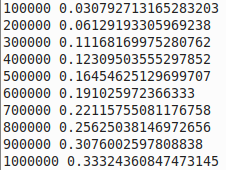
1. delete\_all - O(1)  
   Удаляет ссылки на первый, последний и текущий элемент независимо от длины списка поэтому время выполнения константное.
2. replace\_by\_index - O(N)  
     
     
   

Метод вызывает get\_length и get\_by\_index после чего осуществляет константное количество операций присваивания, поэтому асимптотика будет той же — O(N). График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.

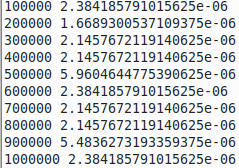
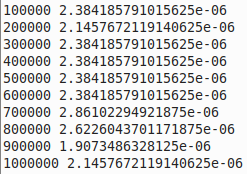
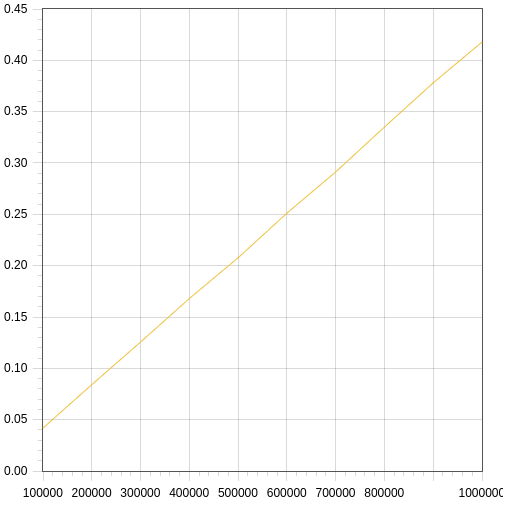
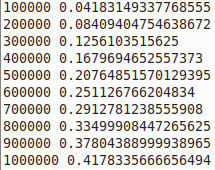


1. is\_empty - O(1)  
   Этот метод возвращает поле is\_empty, время константное.
2. invert - O(N)  
     
     
   

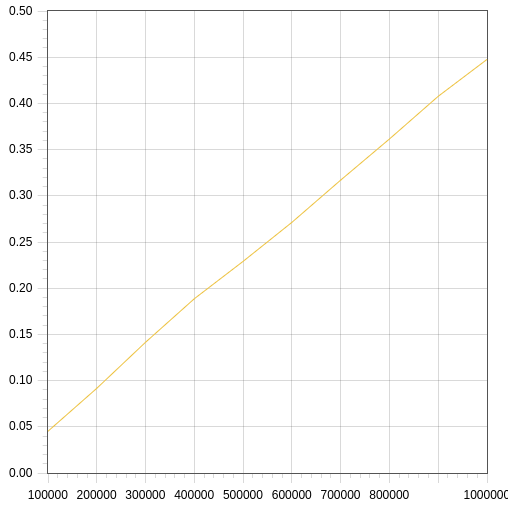
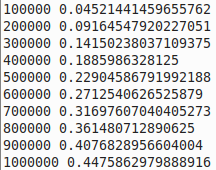
Метод осуществляет полное прохождение по списку с начала до конца (пока ссылки на следующий элемент не окажется), поэтому время линейно зависит от длины.

1. merge\_by\_index - O(N)  
   

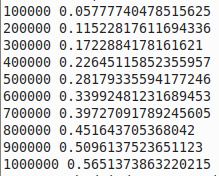
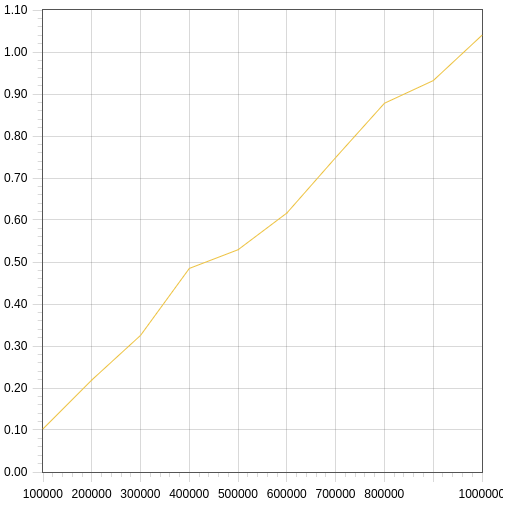
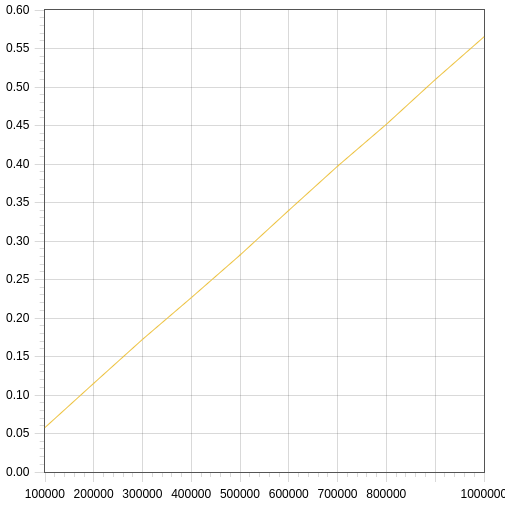
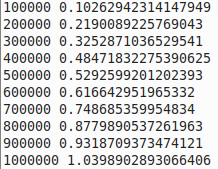
Метод вызывает get\_length и get\_by\_index после чего осуществляет константное количество операций присваивания, поэтому асимптотика будет той же — O(N). График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.

1. merge\_bottom - O(1)  
   Метод меняет ссылки конечных элементов собственного и передаваемого списка и собственную ссылку на последний элемент, поэтому время константное.
2. merge\_top - O(1)  
   Метод меняет ссылки конечных элементов собственного и передаваемого списка и собственную ссылку на первый элемент, поэтому время константное.
3. contains - O(N2)  
     
     
   

Метод осуществляет полное прохождение по списку с начала до конца (пока ссылки на следующий элемент не окажется). При каждом прохождении элемента алгоритм пытается проверить содержание другого списка начиная с текущего элемента. В худшем случае это приведёт в прохождению списка до конца при прохождении каждого элемента, то есть O(N2). Однако при генерации случайных списков ситуация длительного прохождения маловероятно поэтому в действительности видна линейная зависимость.

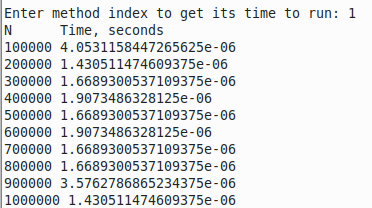
1. find - O(N2)  
     
   

Данный метод работает также как и contains, возвращая другие значения, используя хранение счётчика, который ни коем образом не влияет на асимптотику.

1. rfind - O(N2)  
   Данный метод работает также как и contains, возвращая другие значения, используя хранение счётчика, который ни коем образом не влияет на асимптотику.
2. swap\_by\_index - O(N)  
     
     
   

Метод вызывает get\_length и get\_by\_index после чего осуществляет константное количество операций присваивания, поэтому асимптотика будет той же — O(N). График искривлен из-за того что индекс принимает случайное значение от нуля до N, а не строго N.

**4. Пример работы**

Пример работы с вызовом первого метода. Примеры с исполнением других методов приведены по ходу разбора временной сложности.

**5. Листинг**

**import** random

**import** time

**class** Element:

**def** \_\_init\_\_(self, val=None, prev=None, next=None):

        self.val = val

        self.prev = prev

        self.next = next

**def** \_\_eq\_\_(self, other):  *# Описание механизма сравнения двух элементов*

**if** other **is** None:

**return** False

**if** **not** isinstance(other, int | Element):

**raise** TypeError

**return** self.val == (other **if** isinstance(other, int) **else** other.val)

**def** set(self, val):  *# Установить значение*

        self.val = val

**return** self.val

**def** get(self):  *# Получить значение*

**return** self.val

**def** copy(self):  *# Копирование себя*

**return** Element(self.get())

**class** List2:

**def** \_\_init\_\_(self, \*elements):

        self.head = None

        self.tail = None

        self.cur = None

        self.empty = True

**for** el **in** elements:

            self.add\_bottom(el)

**def** is\_empty(self):  *# Проверка на пустоту списка*

**return** self.empty

**def** get\_length(self):  *# Получение размера списка*

        counter = 0

        self.cur = self.head

**while** self.cur **is** **not** None:

            counter += 1

            self.cur = self.cur.next

**return** counter

**def** add\_first(self, el):  *# Добавление первого элемента в пустой список*

**if** **not** isinstance(el, Element):

            el = Element(el)

        self.head = el

        self.tail = el

        self.empty = False

**def** **print**(self):  *# Вывод списка на экран*

        self.cur = self.head

**while** self.cur **is** **not** None:

**print**(self.cur.get(), end=' ')

            self.cur = self.cur.next

**print**('**\n**', end='')

**def** add\_top(self, el):  *# Добавление в начало списка*

**if** **not** isinstance(el, Element):

            el = Element(el)

**if** self.is\_empty():

            self.add\_first(el)

**else**:

            self.head.prev = el

            el.next = self.head

            self.head = el

**def** add\_bottom(self, el):  *# Добавление в конец списка*

**if** **not** isinstance(el, Element):

            el = Element(el)

**if** self.is\_empty():

            self.add\_first(el)

**else**:

            self.tail.next = el

            el.prev = self.tail

            self.tail = el

**def** delete\_top(self):  *# Удаление первого элемента*

**if** self.is\_empty() **or** self.head.next **is** None:

            self.head = None

            self.tail = None

            self.empty = True

**else**:

            self.head.next.prev = None

            self.head = self.head.next

**def** delete\_bottom(self):  *# Удаление последнего элемента*

**if** self.is\_empty() **or** self.tail.prev **is** None:

            self.head = None

            self.tail = None

            self.empty = True

**else**:

            self.tail.prev.next = None

            self.tail = self.tail.prev

**def** get\_by\_index(self, index):  *# Получение элемента по индексу*

**if** index < 0 **or** index >= self.get\_length():

**raise** Exception

        self.cur = self.head

**for** i **in** range(index):

            self.cur = self.cur.next

**return** self.cur

**def** delete\_by\_index(self, index):  *# Удаление элемента по индексу*

**if** index < 0 **or** index >= self.get\_length():

**raise** Exception

**elif** index == 0:

            self.delete\_top()

**elif** index == self.get\_length() - 1:

            self.delete\_bottom()

**else**:

            self.cur = self.get\_by\_index(index)

            self.cur.prev.next = self.cur.next

            self.cur.next.prev = self.cur.prev

**def** add\_by\_index(self, index, el):  *# Добавление элемента по индексу*

**if** **not** isinstance(el, Element):

            el = Element(el)

**if** index < 0 **or** index >= self.get\_length():

**raise** Exception

**elif** index == 0:

            self.add\_top(el)

**elif** index == self.get\_length() - 1:

            self.add\_bottom(el)

**else**:

            self.cur = self.get\_by\_index(index)

            el.prev = self.cur.prev

            el.next = self.cur

            self.cur.prev.next = el

            self.cur.prev = el

**def** delete\_all(self):  *# Удаление всех элементов списка*

        self.head = None

        self.tail = None

        self.cur = None

        self.empty = True

**def** replace\_by\_index(self, index, el):  *# Замена элемента по индексу на передаваемый элемент*

**if** **not** isinstance(el, Element):

            el = Element(el)

**if** index < 0 **or** index >= self.get\_length():

**raise** Exception

**elif** index == 0:

            self.delete\_top()

            self.add\_top(el)

**elif** index == self.get\_length() - 1:

            self.delete\_bottom()

            self.add\_bottom(el)

**else**:

            self.cur = self.get\_by\_index(index)

            el.prev = self.cur.prev

            el.next = self.cur.next

            self.cur.prev.next = el

            self.cur.next.prev = el

**def** swap\_by\_index(self, index1, index2):  *# Обмен двух элементов списка по индексам*

        index1, index2 = min(index1, index2), max(index1, index2)

**if** index1 < 0 **or** index2 >= self.get\_length():

**raise** Exception

**elif** index1 == index2:

**return**

**else**:

            self.cur = self.head

            el1 = self.get\_by\_index(index1).copy()

            el2 = self.get\_by\_index(index2).copy()

            self.replace\_by\_index(index1, el2)

            self.replace\_by\_index(index2, el1)

**def** invert(self):  *# Меняет порядок элементов в списке на обратный*

        self.cur = self.head

**while** self.cur **is** **not** None:

            self.cur.prev, self.cur.next = self.cur.next, self.cur.prev

            self.cur = self.cur.prev

        self.head, self.tail = self.tail, self.head

**def** merge\_top(self, l2):  *# Вставка другого списка в начало*

        l2.tail.next = self.head

        self.head.prev = l2.tail

        self.head = l2.head

**def** merge\_bottom(self, l2):  *# Вставка другого списка в конец*

        l2.head.prev = self.tail

        self.tail.next = l2.head

        self.tail = l2.tail

**def** merge\_by\_index(self, index, l2):  *# Вставка другого списка начиная с индекса*

**if** index < 0 **or** index >= self.get\_length():

**raise** Exception

**elif** index == 0:

            self.merge\_top(l2)

**elif** index == self.get\_length() - 1:

            self.merge\_bottom(l2)

**else**:

            self.cur = self.get\_by\_index(index)

            l2.head.prev = self.cur.prev

            l2.tail.next = self.cur

            self.cur.prev.next = l2.head

            self.cur.prev = l2.tail

**def** contains(self, l2):  *# Проверка на содержание другого списка*

        remembered = self.head

**while** remembered **is** **not** None:

            self.cur = remembered

            l2.cur = l2.head

**while** self.cur == l2.cur:

                self.cur = self.cur.next

                l2.cur = l2.cur.next

**if** l2.cur **is** None:

**return** True

            remembered = remembered.next

**return** False

**def** find(self, l2):  *# Поиск первого вхождения другого списка*

        remembered = self.head

        index = 0

**while** remembered **is** **not** None:

            self.cur = remembered

            l2.cur = l2.head

**while** self.cur == l2.cur:

                self.cur = self.cur.next

                l2.cur = l2.cur.next

**if** l2.cur **is** None:

**return** index

            remembered = remembered.next

            index += 1

**return** -1

**def** rfind(self, l2):  *# Поиск последнего вхождения другого списка*

        remembered = self.tail

        index = self.get\_length()

**while** remembered **is** **not** None:

            self.cur = remembered

            l2.cur = l2.tail

**while** self.cur == l2.cur:

                self.cur = self.cur.prev

                l2.cur = l2.cur.prev

**if** l2.cur **is** None:

**return** index - l2.get\_length()

            remembered = remembered.prev

            index -= 1

**return** -1

**def** copy(l2):  *# Копирование списка*

    new\_list = List2()

    l2.cur = l2.head

**while** l2.cur **is** **not** None:

        new\_list.add\_bottom(l2.cur.copy())

        l2.cur = l2.cur.next

**return** new\_list

**def** generate(N, min\_el, max\_el):  *# Генерация случайного списка*

    new\_list = List2()

**for** i **in** range(N):

        new\_list.add\_bottom(random.randint(min\_el, max\_el))

**return** new\_list

**def** get\_time(func, \*args):  *# Получение времени выполнения функции*

    ts = time.time()

    func(\*args)

**return** time.time() - ts

run\_dict = {  *# Примеры вызова методов со случайными параметрами*

    "1": **lambda** N: get\_time(generate(N, min\_el, max\_el).add\_bottom, random.randint(min\_el, max\_el)),

    "2": **lambda** N: get\_time(generate(N, min\_el, max\_el).add\_top, random.randint(min\_el, max\_el)),

    "3": **lambda** N: get\_time(generate(N, min\_el, max\_el).delete\_bottom),

    "4": **lambda** N: get\_time(generate(N, min\_el, max\_el).delete\_top),

    "5": **lambda** N: get\_time(generate(N, min\_el, max\_el).add\_by\_index, random.randint(1, N - 1), random.randint(min\_el, max\_el)),

    "6": **lambda** N: get\_time(generate(N, min\_el, max\_el).get\_by\_index, random.randint(1, N - 1)),

    "7": **lambda** N: get\_time(generate(N, min\_el, max\_el).delete\_by\_index, random.randint(1, N - 1)),

    "8": **lambda** N: get\_time(generate(N, min\_el, max\_el).get\_length),

    "9": **lambda** N: get\_time(generate(N, min\_el, max\_el).delete\_all),

    "10": **lambda** N: get\_time(generate(N, min\_el, max\_el).replace\_by\_index, random.randint(1, N - 1), random.randint(min\_el, max\_el)),

    "11": **lambda** N: get\_time(generate(N, min\_el, max\_el).is\_empty),

    "12": **lambda** N: get\_time(generate(N, min\_el, max\_el).invert),

    "13": **lambda** N: get\_time(generate(N, min\_el, max\_el).merge\_by\_index, random.randint(1, N - 1), generate(N, min\_el, max\_el)),

    "14": **lambda** N: get\_time(generate(N, min\_el, max\_el).merge\_bottom, generate(N, min\_el, max\_el)),

    "15": **lambda** N: get\_time(generate(N, min\_el, max\_el).merge\_top,  generate(N, min\_el, max\_el)),

    "16": **lambda** N: get\_time(generate(N, min\_el, max\_el).contains, generate(N, min\_el, max\_el)),

    "17": **lambda** N: get\_time(generate(N, min\_el, max\_el).find, generate(N, min\_el, max\_el)),

    "18": **lambda** N: get\_time(generate(N, min\_el, max\_el).rfind, generate(N, min\_el, max\_el)),

    "19": **lambda** N: get\_time(generate(N, min\_el, max\_el).swap\_by\_index, random.randint(1, N - 1), random.randint(1, N - 1)),

    "": **lambda** N: exit()

}

min\_el = 0  *# Минимальное допустимое значение элемента*

max\_el = 10\*\*6  *# Максимальное допустимое значение элемента*

points = range(10\*\*5, 10\*\*6 + 1, 10\*\*5)  *# Точки (количества элементов в списке для разных вызовов функции)*

**if** \_\_name\_\_ == "\_\_main\_\_":

**while** True:

        case = input("Enter method index to get its time to run: ")

**print**("N      Time, seconds")

**for** N **in** points:

**print**(N, run\_dict[case](N))