**11 вариант. Романов Олег 11-906. 1 семестровая работа.**

1. **void insert(int k):**

Описание на абстрактном уровне:

Данный метод осуществляет вставку элемента равного k в исходный список, при этом сохраняя его упорядоченность. Изначально создаем новый объект класса ListElement в который заранее заполняем атрибут value значением k.

В случае, если все значения элементов списка (включая голову) больше чем значение элемента, который мы собираемся вставить, тогда вставляемый нами элемент становится новой головой списка.

Пока мы не найдем тут случай, что вставляемый нами элемент больше чем какой-либо (ну или в случае, если подобного нет, то до конца списка) ближайший к началу списка мы проходимся по списку. Как только мы встретили подобный случай, цикл завершает свою работу и далее проверяем, если наш новый вставляемый элемент больше элемента, на котором мы остановились в цикле, то вставляем его в список как следующий после него, иначе вставляем до него.

Оценка по времени:

- В случае, когда head == null сложность по времени О(1).

- В случае, когда все элементы списка больше чем k сложность по времени O(1).

- В остальных случаях сложность по времени O(n)

Итоговая сложность по времени = O(n)

Оценка по памяти: O(n).

1. **void delete(int k):**

Описание на абстрактном уровне:

На вход подается некоторое число, которое соответствует значению элемента, который необходимо удалить из списка. Перебираем все элементы списка. Проверяем на условия:

Если мы нашли элемент со значение К и при этом он является головой списка, то головой списка становится просто следующий элемент списка, исходная голова тем самым теряется. Далее, если мы нашли элемент со значением К, но при этом он является последним элементов списка, тогда у предыдущего элемента ссылку на следующий элемент мы обнуляем, а у элемента, который удаляем, обнуляем ссылку на предыдущий элемент списка. Или же если не предыдущие два случая но при этом очередной элемент равен К, то у предыдущего элемента в качестве ссылки на следующий элемент задаем элемент, который следовал после элемента, который мы собираемся удалить, а у него ссылкой на предыдущий указываем элемент, который был предыдущим у удаляемого элемента.

Оценка по времени:

- В случае, когда удаляемый элемент является головой списка, сложность по времени O(1).

- В остальных случаях сложность по времени O(n).

Итоговая сложность по времени = O(n)

Оценка по памяти: O(n).

1. **MyLinkedList2 merge(MyLinkedList2 ll1):**

Описание на абстрактном уровне:

В данном методе на основе двух упорядоченных списков создается новый большой упорядоченный список. Идея такова, на основании двух списков выбираем меньший из них и добавляем в новый список. В конечном итоге, один из исходных списков опустеет, тем самым останется лишь добавить в новый список элементы оставшегося непустого списка.

Оценка по времени: O(N + M). Где N, M соответственно размеры исходных списков.

Оценка по памяти: O(n)

1. **int maxNum():**

Описание на абстрактном уровне:

Данный метод находит максимальное число одинаковых элементов списка. Для этого начальное значение максимума задается минимально возможное типа int. Далее проходим по всему списку и по ходу изменяем max. В случае, если все элементы оказались уникальными, значение max присвоится 0, в обратном случае на вывод пойдет максимальное число одинаковых элементов.

Оценка по времени: O(n).

Оценка по памяти: O(n)

1. **MyLinkedList2 [] divide():**

Описание на абстрактном уровне:

Данный метод разделяет исходный список на два, в первый попадают элементы, значения которых кратно 3, а во второй список все остальные элементы. Эти два списка заполняются в массив, где на 0 позицию устанавливается список, в котором значения всех элементов кратны 3, а на 1 позицию второй список, содержащий все остальные элементы исходного списка. На выход подается этот самый массив типа MyLinkedList2.

Оценка по времени: O(n).

Оценка по памяти: O(n^2).

1. **MyLinkedList2 newList():**

Описание на абстрактном уровне:

Данный метод создает новый список, на основании исходного списка, но применяя правило : (каждый элемент нового списка является произведением j и (n-j) элемента

исходного списка). Реализовано это следующим образом. У каждого элемента списка добавлено поле index. А так же у списка имеется поле «длина». Проходя по всем элентам исходного списка, вставляем элементы методом insert, который поддерживает упорядоченность элементов, в него передаем нужный индекс по правилу упомянутому чуть ранее, используя метод elementByIndex, который возвращает элемент под нужным индексом. В результате мы получим новый список, каждый элемент нового списка является произведением j и (n-j) элемента исходного списка.

Оценка по времени: O(n^3).

Оценка по памяти: O(n^2)

1. **ListElement elementByIndex(int index):**

Описание на абстрактном уровне:

На вход в метод подается значение нужного индекса, а метод путем перебора элементов списка находит элемент, который соответствует нужному индексу и возвращает его.

Оценка по времени: O(n).

Оценка по памяти: O(n).

1. **void add(int x):**

Описание на абстрактном уровне:

Данный метод осуществляет добавление нового элемента в список. При добавлении элемента ему задается индекс, задается значение, ссылка на предыдущий элемент, а у предыдущего элемента задается ссылка на следующий. Т.е добавление элементов происходит всегда в конец списка. Если же в списке еще ни создано ни одного элемента, то первый созданный элемент становится головой списка.

Оценка по времени: O(n).

Оценка по памяти: O(n).

1. **public int[] sortArray(int[] arr, int low, int high):**

Оценка по времени:

- В среднем случае: O(n log n).

- В худшем случае: O(n^2).

Оценка по памяти: O(n).