Эксперименты с массивами

Убедимся в том, что python list реализует именно динамический массив. Для этого мы можем провести временны**ы**е тесты. Для их реализации импортируем модуль *timeit*. Функция с аналогичным названием *timeit* измеряет время, которое затрачивается на запуск кода. В качестве первого аргумента принимает исследуемый код (в виде **строки**), а аргумент *number* - количество повторений. Мы укажем этот аргумент равным 100, чтобы функций *timeit* запустила наш код 100 раз. Результат измерений разделим также на 100, чтобы получить среднее время работы. Напишем функцию, которая будет возвращать результат измерений.

```
def elapsed_time(func, size):
    return timeit.timeit(func % size, number=100)/100
```

Аргумент ***func % size*** - это форматированная строка с кодом. Прежде чем будем продолжать наш численный эксперимент, давайте вспомним некоторые функции списков python (а если говорить в контексте нашей темы - “динамических массивов”)

В эксперименте будем тестировать вставку элемента в конец.

```
code_append = """
elements = range(%d) # генератор элементов, которые будут вставляться в
список
array = [] # список, работу которого тестируем
for e in elements:
    array.append(e) # добавляем каждый раз в конец
"""
```

Код, который необходимо предоставить функции *timeit*, должен быть записан в виде строки. Для того чтобы мы могли запускать эти фрагменты с разным набором входных данных, сделаем эту строку форматированной, вставив в аргумент функции *range*, специальный символ *%d*, который требует подстановки целого числа. Именно это и происходит при использовании *func % size* внутри написанной нами функции измерений.

Завершающий этап - создание цикла, который будет запускать эксперимент на разных размерах массива. Важный нюанс состоит в том, что нам нет необходимости знать абсолютное значение измеренного времени, но нам интересно узнать, как это время растёт с увеличением размера массива, например, в 2 раза.

```
for s in range(10,15):
    measure_1 = elapsed_time(code_append, 2**s)
    measure_2 = elapsed_time(code_append, 2**(s+1))
    ratio = measure_2 / measure_1
    print("[%d]/[%d] -> %5.2f" % (2**(s+1), 2**s, ratio))
```

Сначала мы нашли измерение, например, с размером массива 2**10 = 1024, потом провели измерение на размере 2**11 = 2048. После чего нашли их отношение и вывели в более удобном формате. Результат работы эксперимента появится в выводе консоли

```
# [2048]/[1024] ->  2.03
# [4096]/[2048] ->  1.91
# [8192]/[4096] ->  2.01
# [16384]/[8192] ->  2.13
# [32768]/[16384] ->  2.04
```

Можно увидеть, что с ростом размера массива затраченное время возрастает примерно в 2 раза. Посмотрев на код еще раз, можно дать оценку сложности и убедиться, что она соответствует эксперименту

```
elements = range(%d) # инициализация генератора - O(1)
array = [] # инициализация массива - O(1)
for e in elements: # цикл из n итераций
    array.append(e) # вставка в конец имеет сложность O(1)
```

И тогда имеем

O(1) + O(1) + n*O(1) = O(n)

Иными словами, сложность этого алгоритма - O(n), что мы и увидели в эксперименте - возрастание размера массива в 2 раза приводит к увеличению времени работы в 2 раза.

Несколько иная ситуация стоит при вставке элемента в начало массива.

Задание 1. Модифицируйте проведенный эксперимент таким образом, чтобы он проводил измерения времени на тех же самых размерах массива. Анализу подлежит операция вставки элемента в **начало** массива.

Задание 2. Во сколько раз (в среднем) увеличивается время работы фрагмента кода в самостоятельном эксперименте?

1. 1
2. 2
3. 4
4. 8

Задание 3. Какую асимптотическую сложность будет иметь этот фрагмент кода? Запишите ответ, используя O-нотацию.

Ответы на задания приведены в конце документа :)

Ответы на задания

Задание 1.

```
code_insert = """
elements = range(%d)
array = []
for e in elements:
    array.insert(0,e)
"""

for s in range(10,15):
    measure_1 = elapsed_time(code_insert, 2**s)
    measure_2 = elapsed_time(code_insert, 2**(s+1))
    ratio = measure_2 / measure_1
    print("[%d]/[%d] -> %5.2f" % (2**(s+1), 2**s, ratio))
```

Задание 2.

1. 1 - в таком случае алгоритм бы не зависел от количества элементов, но это не так
2. 2 - операция вставки на произвольное место работает хуже, чем вставка в конец
3. 4 - правильный ответ
4. 8 - слишком много даже для операции вставки на произвольное место

Задание 3.

O(n^2) или O(n**2) или O(n*n)