Εισαγωγή n στοιχείων στην αρχή και το τέλος μιας λίστας Python

Η Python υλοποιεί τις λίστες πολύ κοντά στο θεωρητικό μοντέλο του array όπως το κάναμε στο μάθημα. Έτσι η εισαγωγή n στοιχείων στο τέλος της λίστας έχει πολυπλοκότητα $\mathcal{O}(n)$ (δηλαδή n φορές το $\mathcal{O}(1)$) ενώ η εισαγωγή n στοιχείων στην αρχή της λίστας (θέση 0) έχει πολυπλοκότητα $\mathcal{O}(n^2)$ (ή αλλιώς, n φορές το $\mathcal{O}(n)$).

Δείτε τα επόμενα παραδείγματα, όπου εισάγουμε 50 χιλιάδες αριθμούς στο τέλος και στην αρχή μιας λίστας. Παρατίθεται ο χρόνος εκτέλεσης σε ένα μέσης κατηγορίας laptop.

Σημ: τα `%%time` δεν είναι Python, αλλά του περιβάλλοντος εκτέλεσης του παραδείγματος (Jupyter notebook (https://jupyter.org/install)). Αφαιρέστε τα αν τρέχετε απλή Python.

In [5]:
```
%%time
# εισαγωγή στο τέλος
numbers = []
for i in range(50000):       # O(n)
    numbers.append(i)    # O(1), συνολικά O(n)
print('done')
```

```
done
Wall time: 11.3 ms
```

In [6]:
```
%%time
# εισαγωγή στην αρχή
numbers = []
for i in range(50000):        # O(n)
    numbers.insert(0,i)   # O(n) συνολικά O(n^2)
print('done')
```

```
done
Wall time: 952 ms
```

Τι θα γίνει αν διπλασιάσουμε το πλήθος των αριθμών;

Θεωρητικά, ενώ ο χρόνος εισαγωγής στο τέλος διπλασιάζεται, για την εισαγωγή στην αρχή θα *τετραπλασιαστεί* λόγω της πολυπλοκότητας $\mathcal{O}(n^2)$. Μπορείτε να δοκιμάσετε κι εσείς!

Γιατί δεν είναι η λύση στο πρόβλημα "ένας γρηγορότερος υπολογιστής"

Θα μπορούσαμε να σκεφτούμε ότι με γρηγορότερο υπολογιστή θα λύσουμε το πρόβλημα ενός αλγορίθμου με άσχημη πολυπλοκότητα, συνεπώς δεν είναι κρίσιμη η πολυπλοκότητα των αλγορίθμων. Αυτό όμως είναι **εντελώς λάθος**!

Δείτε το εξής παράδειγμα (*βασισμένο στο αντίστοιχο παράδειγμα από το βιβλίο Magnus Lie Hetland. 2014. Python Algorithms: Mastering Basic Algorithms in the Python Language (2nd. ed.). Apress, USA.*):

Έστω ότι θέλουμε να εισάγουμε 100 χιλιάδες αριθμούς σε μια λίστα και έχουμε να επιλέξουμε

μεταξύ της εισαγωγής στο τέλος (με πολυπλοκότητα $\mathcal{O}(n)$) και στην αρχή (με πολυπλοκότητα $\mathcal{O}(n^2)$).

- Τρέχουμε την εισαγωγή στο τέλος σε παλιό υπολογιστή και Python.
- Ταυτόχρονα, τρέχουμε την εισαγωγή στην αρχή σε εξωφρενικά γρήγορο υπολογιστή, 1000 φορές πιο γρήγορο από τον παλιό, και σε C (50 φορές πιο γρήγορη από την Python).

Αυτό τι σημαίνει; Χονδρικά, αν μια βασική λειτουργία στον νέο υπολογιστή με C παίρνει χρόνο t, στον παλιό υπολογιστή με Python θα πάρει $50.000 \times t$ χρόνο.

- Με πολυπλοκότητα $\mathcal{O}(n^2)$, όπου n είναι 100.000, ο νέος υπολογιστής θα κάνει 100.000^2 βασικές λειτουργίες και θα χρειαστεί χρόνο $100.000^2 \times t$
- Με πολυπλοκότητα $\mathcal{O}(n)$, όπου n είναι 100.000, ο παλιός υπολογιστής θα κάνει 100.000 βασικές λειτουργίες και θα χρειαστεί χρόνο $100.000 \times (50.000 \times t)$

Η αλλιώς:

$$\frac{\text{Χρόνος παλιού}}{\text{Χρόνος νέου}} = \frac{100.000 \times 50.000 \times t}{100.000^2 \times t} = \frac{1}{2}$$

και κατά συνέπεια $\text{Χρόνος νέου} = 2 \times \text{Χρόνος παλιού}$. Ο εξωφρενικά γρηγορότερος υπολογιστής θα χρειαστεί τον διπλάσιο χρόνο για την εισαγωγή των αριθμών και αυτό λόγω της χειρότερης πολυπλοκότητας της εισαγωγής στην αρχή.

-