Analyzing the running times of the sorting methods

Week 13

Insertion sort

```
def insertionSort(li):
    for i in range(1, len(li)):
        key = li[i]
        # Move elements of li[0..i-1] that are greater than key one step back
        j = i - 1
        while j >=0 and key < li[j]:
            li[j+1] = li[j]
            j = j - 1
        li[j+1] = key     # put key into right position
```

Analyzing insertion sort

- We can see immediately that the algorithm contains two nested loops. That does not look good.
- On average, the inner loop will execute about $\frac{i}{2}$ times for each *i*.
- The total number of comparisons in the **average case** is:

$$\frac{1}{2}(1 + 2 + \ \ldots + n - 1) = \frac{1}{2}\frac{1}{2}\big(n(n-1)\big) = \frac{1}{4}n^2 - \frac{1}{4}n$$

The running time grows at the same rate as n^2. The running time is **quadratic**.

Analyzing quicksort

```
def quicksort(li):
    if len(li) <= 1:  # base case
        return li
    else:
        pivot = li[0]  # first element
        below = []
        above = []

        for i in li[1:]:      # partitioning
            if i < pivot:
                below.append(i)
            else:
                above.append(i)

        return quicksort(below) + [pivot] + quicksort(above)
```

Analyzing quicksort

- If elements are randomly distributed in the list, then partitioning goes always "well".
- This means and both below and above contain about half of the elements.
- The partioning takes cn time. This is because slicing `li[1:]` takes linear time. Then we have a loop containing appends. Each append takes a constant time.
- The running time of the algorithm is

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

Iteration method

- First Iteration: $T(n) = 2T\left(\frac{n}{2}\right) + cn$
- Second Iteration: $T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)$.
- => $T(n) = 2\left(2T\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)\right) + cn = 4T\left(\frac{n}{4}\right) + 2cn$
- Third Iteration: $T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + c\left(\frac{n}{4}\right)$
- $T(n) = 4\left(2T\left(\frac{n}{8}\right) + c\left(\frac{n}{4}\right)\right) + 2cn = 8T\left(\frac{n}{8}\right) + 3cn$

- General Form: After k iterations, the general form is:

$$T(n) = 2^k\, T\left(\frac{n}{2^k}\right) + kcn$$

- **Base Case**. When

$$\frac{n}{2^k} = 1$$

then $n = 2^k$ and $k = \lg n$.

- Substitute $k = \lg n$ into the general form:

$$T(n) = 2^k\, T\left(\frac{n}{2^k}\right) + kcn$$

- Since $2^{\lg n} = n$ and $T(1) = d$, we have

$$T(n) = n\, d + cn \lg n$$

- Because $cn \lg n$ grows faster than $n\, d$, we can estimate that the **running time of the quicksort** is

$$T(n) = c \cdot n \lg n$$