Python Class 6: Sorting algorithms and recursion

Michelle Torres

August 11, 2016

Complexity

- The amount of time / the number of operations necessary to complete a task.

COMPLEXITY

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Big-O notation is a **relative** **representation** of the **complexity** of an algorithm.
  - Gives complexity in terms of the size of the input.
  - eg. How many operations do I need to perform when adding numbers? How many comparisons do I need to sort a list of n items?
    - *Addition* $\rightarrow$ 3+4, 24+36, 123+456,...

COMPLEXITY

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Big-O notation is a **relative** **representation** of the **complexity** of an algorithm.
  - Gives complexity in terms of the size of the input.
  - eg. How many operations do I need to perform when adding numbers? How many comparisons do I need to sort a list of n items?
    - *Addition* $\rightarrow$ 3+4, 24+36, 123+456,...$\rightarrow O(n)$
    - *Multiplication* $\rightarrow 3 \times 5$, 12×11, 3456×1234, ...

Complexity

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Big-O notation is a **relative** **representation** of the **complexity** of an algorithm.
  - Gives complexity in terms of the size of the input.
  - eg. How many operations do I need to perform when adding numbers? How many comparisons do I need to sort a list of n items?
    - *Addition* $\rightarrow$ 3+4, 24+36, 123+456,...$\rightarrow O(n)$
    - *Multiplication* $\rightarrow 3 \times 5$, 12×11, 3456×1234, ... $\rightarrow O(n^2)$ (!)
  - Consider only the term with the highest order of n: $O(n^2 + 2n) \rightarrow O(n^2)$

Complexity

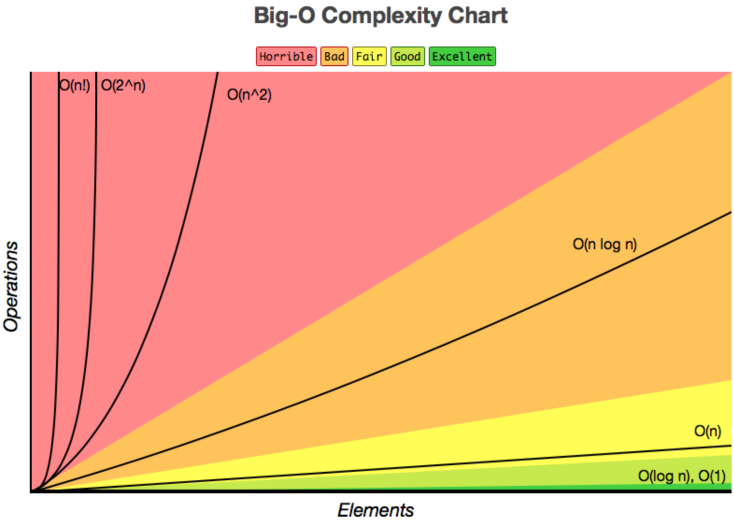
- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Big-O notation is a **relative representation** of the **complexity** of an algorithm.
  - Gives complexity in terms of the size of the input.
  - eg. How many operations do I need to perform when adding numbers? How many comparisons do I need to sort a list of n items?
    - *Addition* $\rightarrow$ 3+4, 24+36, 123+456,...$\rightarrow O(n)$
    - *Multiplication* $\rightarrow 3 \times 5$, 12×11, 3456×1234, ... $\rightarrow O(n^2)$ (!)
  - Consider only the term with the highest order of n: $O(n^2 + 2n) \rightarrow O(n^2)$

Complexity

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Big-O notation is a **relative** **representation** of the **complexity** of an algorithm.
  - Gives complexity in terms of the size of the input.
  - eg. How many operations do I need to perform when adding numbers? How many comparisons do I need to sort a list of n items?
    - *Addition* $\rightarrow$ 3+4, 24+36, 123+456,... $\rightarrow O(n)$
    - *Multiplication* $\rightarrow 3 \times 5$, 12×11, 3456×1234, ... $\rightarrow O(n^2)$ (!)
  - Consider only the term with the highest order of n: $O(n^2 + 2n) \rightarrow O(n^2)$
- What about a 3456×123?
- Check the best case, the average case and the **worst case**.

VISUALIZING COMPLEXITY

Visualizing complexity



Insertion Sort

- Start with the element in the second position.
- Insert it to the appropriate position among the numbers to its left.
  - Check whether it is greater than the last element to its left.
  - If not, check the second to last element to its left.
  - ...
- Continue with the element in the third position.

SELECTION SORT

- Go over the unsorted list to find the minimum and place it as your first element of your sorted list.
- Repeat.

Bubble Sort

- Compare - swap stage
  - Compare the first two elements and swap them if necessary.
  - Compare the second and third elements and swap them if necessary.
  - Repeat until the end of the list.
- If you did any swaps in the first stage, repeat it with the first n-1 elements.
- Repeat.

MERGE SORT

- Divide the list into sublists - each with one element.
- Merge the sublists to create new sublists - each with two elements.
- Repeat until you have a single list

Merge Sort

- Divide the list into sublists - each with one element.
- Merge the sublists to create new sublists - each with two elements.
- Repeat until you have a single list
- See it in action: `https://youtu.be/JSceec-wEyw`

Merge Sort

- Divide the list into sublists - each with one element.
- Merge the sublists to create new sublists - each with two elements.
- Repeat until you have a single list
- See it in action: `https://youtu.be/JSceec-wEyw`

All in action: `https://visualgo.net/bn/sorting`

Recursion

- Function calls itself.
- You need to know:
  - the base case
  - when to call the function
  - when to stop
- The typical example:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$