Complexity and Recursion

Erin Rossiter

Complexity

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.

$O()$ ("Big-O") notation—complexity of algorithm in terms of the size of the input

Complexity

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

$O()$ ("Big-O") notation—complexity of algorithm in terms of the size of the input

- $O(1)$ will execute in the same time regardless of input size

```
def o_1(input):
    out = input[0] + 1
    return out
```

- $O(n)$, the number of operations will grow linearly and in direct proportion to input size

```
def o_n(input):
    for i in range(input):
        input[i] += 1
    return input
```

More Examples

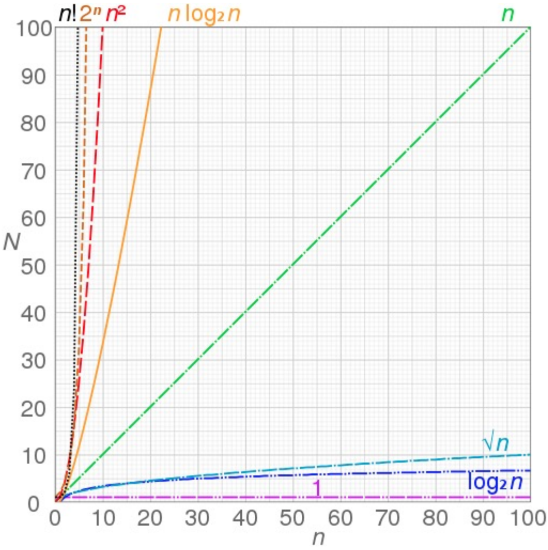
- $O(n^2)$—growth is proportional to the square of the size of the input

```
def o_nsqr(input):
    out = []
    for i in range(input):
        for j in range(input):
            out.append(i + j)
    return out
```

- $O(2^n)$—growth is exponential, doubling with each addition to the input

```
def fib(input):
    if input <= 1:
    return input
return fib(input - 1) + fib(input - 2)
```

A Visual



By Cmglee https://commons.wikimedia.org/w/index.php?curid=50321072

Sorting Algorithms

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.

Recursion

Recursion

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

Recursion

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

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