Introduction to Python:
Day 8 - Complexity & Recursion

Ryden Butler

Complexity - What is it?

- The amount of time / operations required to complete a task
- Describes the limiting behavior of the function (as $n \to \infty$)
- $O(\cdot)$ ("Big-O") notation – complexity in terms of input size
- Ignore constants and lesser finite sums
- Check for best, worst, and usual case

$O(1)$

- $O(1)$: executes in constant time
- ```
  def o_1(input):
      out = input[0] + 1
      return out
  ```
  ```

$O(n)$

- $O(n)$: operations grow linearly, proportional to input size

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

$O(n^2)$

- $O(n^2)$: operations grow proportional to the square of the input size

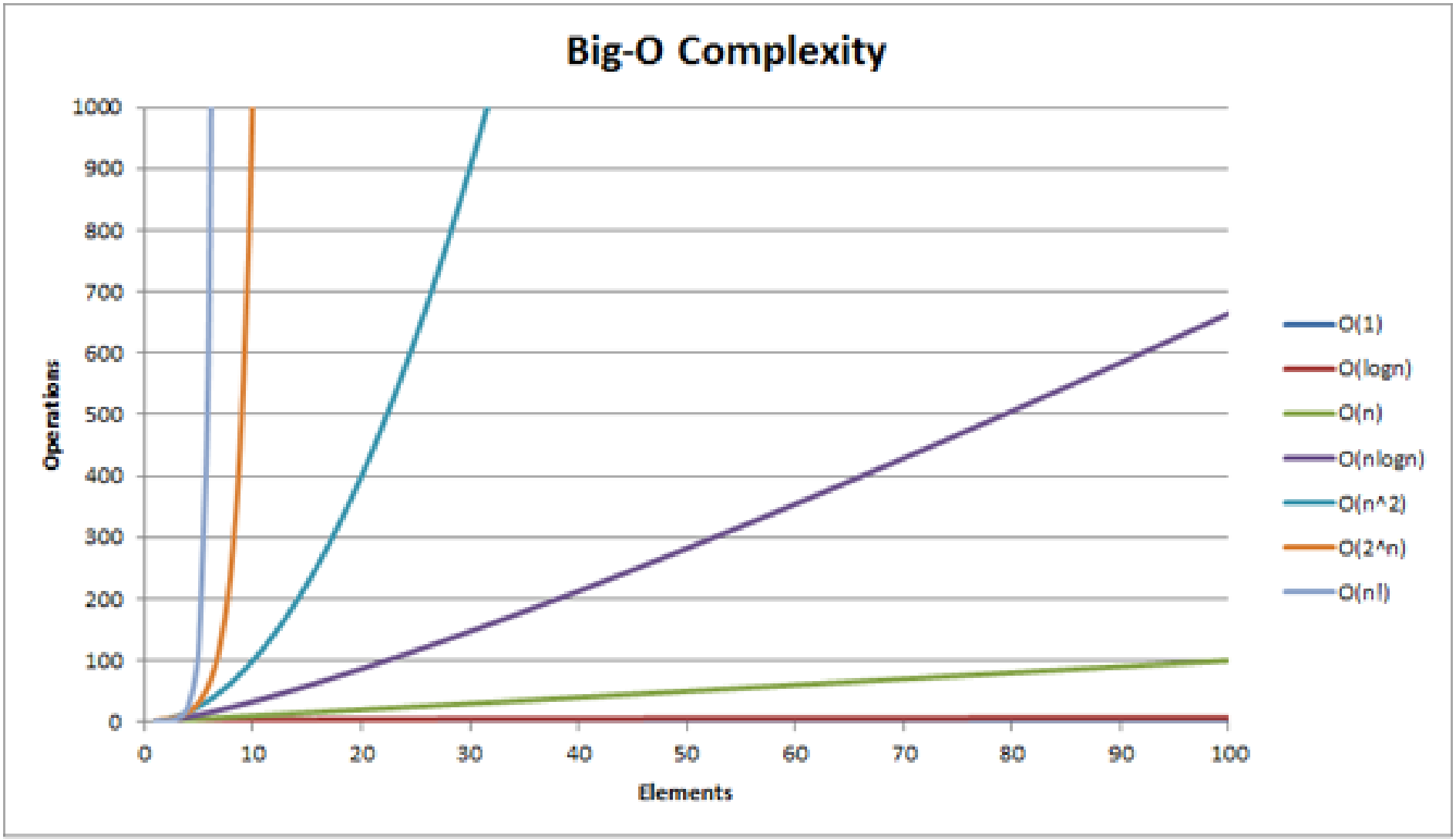
- ```
  def o_nsqr(input):
      out = []
      for i in range(input):
          for j in range(input):
              out.append(i + j)
      return out
  ```
  ```

$O(2^n)$

- $O(n^2)$: operations grow exponentially, doubling with each input

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

Visual Depiction



Recursion - What is it?

- A function that calls itself
- To code you must include:
  - the base case
  - when to call the function
  - when to stop the function
- Example:

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

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
  - And so on, until False
- Continue with the element in the third position

Selection Sort

- Go over the unsorted list to find the minimum
- Remove minimum and place it as your first element of a new 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
  - Continue until the end of the list
- If any swaps occurred, repeat with the first n-1 elements

Bogo Sort

- Randomize number order
- If sorted, stop. Else repeat.