Python speed of program

All Versions

Improvements requested:

Examples

How Big-O-Notation works

The order of magnitude, often written as Big O notation, is is expressed as O(f(n)). f(n) is simply to express the dominant part of the original T(n). The Order function essentially shows the dominant value and can give a useful approximation of the number of steps involved.

For example, say `T(n) = 1 + n`. As n becomes larger the constant 1 becomes less and less important. Since we are looking for an approximation, we can forget about the 1 and just say that the running time O(n).

Another example is `T(n) = 7n^2 + 26n + 1005`. When n is small such as 1 or 2, 1005 seems like the dominant number. However as n becomes larger the n^2 term beckmes more important. In fact, when n becomes large enough the other 2 terms are insignificant in the role they play to determine the final result. Focusing on the 7n^2, the 7 coefficient becomes less important aswell. We can then say the function T(n) has a order of magnitude of f(n) = n^2, or just, O(n^2).

Somtimes, the order of magnitude relies on the exact values of the data, rather than the size. In these kinds of algorithms, we characterise their performance in terms of Best Case, Worst Case, or Average Case. The Worst Case performance refers to a specific data set where the algorithm performs particularly poorly. However, the Best Case refers to the same algorithm with a different data set performing extremely well. In most cases the performance is between these 2, in the Average Case.

Deque operations

A deque is a double-ended queue.

Operations : Average Case (assumes parameters are randomly generated)

Append : O(1)

Appendleft : O(1)

Copy : O(n)

Extend : O(k)

Extendleft : O(k)

Pop : O(1)

Popleft : O(1)

Remove : O(n)

Rotate : O(k)

Improvements requested:

Dict operations

Operations: Average Case

Copy: O(n)

Get Item: O(1)

Set Item: (1)

Delete Item: O(1)

List operations

Operations : Average Case (assumes parameters are randomly generated)

Append : O(1)

Copy : O(n)

Del slice : O(n)

Delete item : O(n)

Insert : O(n)

Get item : O(1)

Set item : O(1)

Iteration : O(n)

Get slice : O(k)

Set slice : O(n + k)

Extend : O(k)

Sort : O(n log n)

Multiply : O(nk)

x in s : O(n)

min(s), max(s) :O(n)

Get length : O(1)

Notation

Basic Idea

The notation used when describing the speed of your Python program is called Big-O notation. Let's say you have a function:

```
def list_check(to_check, the_list):
    for item in the_list:
        if to_check == item:
          return True
    return False
```

This is a simple function to check if an item is in a list. To describe the complexity of this function, you will say O(n). This means "Order of n" as the O function is known as the Order function.

O(n) - generally n is the number of items in container

O(k) - generally k is the value of the parameter or the number of elements in the parameter

---

Set operations

Operation : Average Case (assumes parameters generated randomly) : Worst case

x in s : O(1)

Difference s - t : O(len(s))

Intersection s&t : O(min(len(s), len(t))) : O(len(s) * len(t)

Multiple intersection s1&s2&s3&...&sn : : (n-1) * O(l) where l is max(len(s1),...,len(sn))

s.difference_update(t) : O(len(t)) : O(len(t) * len(s))

s.symetric_difference_update(t) : O(len(t))

Symetric difference s^t : O(len(s)) : O(len(s) * len(t))

Union s|t : O(len(s) + len(t))

Syntax

Parameters

Remarks