codecademy

# Algorithmic Complexity

## Big-Θ Notation

We compute the big-Θ of an algorithm by counting the number of iterations the algorithm *always* takes with an input of n. For instance, the loop in the pseudo code below will always iterate N times for a list size of N. The runtime can be described as Θ(N).

```
for each item in list:
  print item
```

## Asymptotic Notation

Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n. There are three different notations: big O, big Theta (Θ), and big Omega (Ω). big-Θ is used when the running time is the same for all cases, big-O for the worst case running time, and big-Ω for the best case running time.

## Adding Runtimes

When an algorithm consists of many parts, we describe its runtime based on the slowest part of the program.

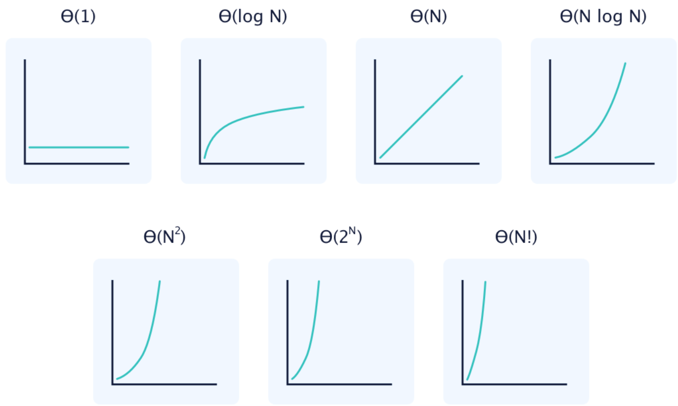
```
An algorithm with three parts has running
times of Θ(2N) + Θ(log N) + Θ(1). We only
care about the slowest part, so we would
quantify the runtime to be Θ(N). We would
also drop the coefficient of 2 since when N
gets really large, the multiplier 2 will
have a small effect.
```

## Algorithmic Common Runtimes

The common algorithmic runtimes from fastest to slowest are:

- constant: Θ(1)
- logarithmic: Θ(log N)
- linear: Θ(N)
- polynomial: Θ(N^2)
- exponential: Θ(2^N)
- factorial: Θ(N!)

Common Runtimes




## Big-O Notation

The Big-O notation describes the worst-case running time of a program. We compute the Big-O of an algorithm by counting how many iterations an algorithm will take in the worst-case scenario with an input of N. We typically consult the Big-O because we must always plan for the worst case. For example, O(log n) describes the Big-O of a binary search algorithm.

## Big-Ω Notation

Big-Ω (Omega) describes the best running time of a program. We compute the big-Ω by counting how many iterations an algorithm will take in the best-case scenario based on an input of N. For example, a Bubble Sort algorithm has a running time of Ω(N) because in the best case scenario the list is already sorted, and the bubble sort will terminate after the first iteration.

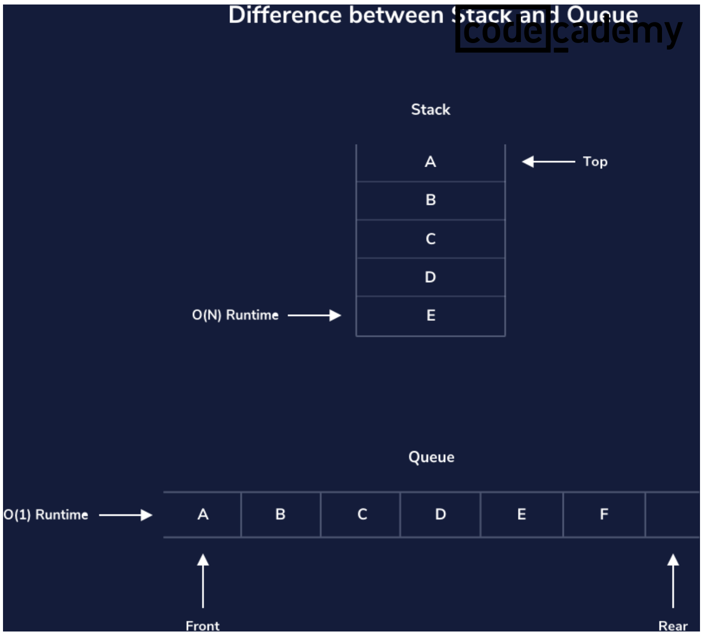
## Analyzing Runtime

The speed of an algorithm can be analyzed by using a while loop. The loop can be used to count the number of iterations it takes a function to complete.

```
def half(N):
  count = 0
  while N > 1:
    N = N//2
    count += 1
  return count
```

## Queue Versus Stack

A Queue data structure is based on First In First Out order. It takes O(1) runtime to retrieve the first item in a Queue . A Stack data structure is based on First In Last Out order. Therefore, it takes O(N) runtime to retrieve the first value in a Stack because it is all the way at the bottom.




## Max Value Search in List

The big-O runtime for locating the maximum value in a list of size N is O(N). This is because the entire list of N members has to be traversed.

```
# O(N) runtime
def find_max(linked_list):
  current = linked_list.get_head_node()
  maximum = current.get_value()
  while current.get_next_node():
    current = current.get_next_node()
    val = current.get_value()
    if val > maximum:
      maximum = val
  return maximum
```