Big Theta (Θ)

5 min

The first subtype of asymptotic notation we will explore is big Theta (denoted by Θ). We use big Theta when a program has only one case in terms of runtime. But what exactly does that mean? Take a look at the pseudocode for a function that prints the values in a list below:

```
Function with input that is a list of size N:
  For each value in list:
   Print the value
```

The number of instructions the computer has to perform is based on how many iterations the loop will do because if the loop does more iterations, then the computer will perform instructions. Now, let's see how many iterations the loop will do dependent on the value of N.

Size of List	**–vs.–**	**Number of Iterations**
1		1
2		2
3		3
.		.
.		.
.		.
.		.
.		.
N		N

As we can see in every case, with a list of size N, the program has a runtime of N because the program has to print a value N times. Thus, we would say the runtime is Θ(N).

Let's look at a more complicated example. In the following pseudocode program, the function takes in an integer, N, and counts the number of times it takes for N to be divided by 2 until N reaches 1.

```
Function that has integer input N:
  Set a count variable to 0
  Loop while N is not equal to 1:
    Increment count
    N = N/2
  Return count
```

Now, let's see how many iterations the loop will perform based on N.

N	**–vs.–**	**Number of Iterations**
1		0

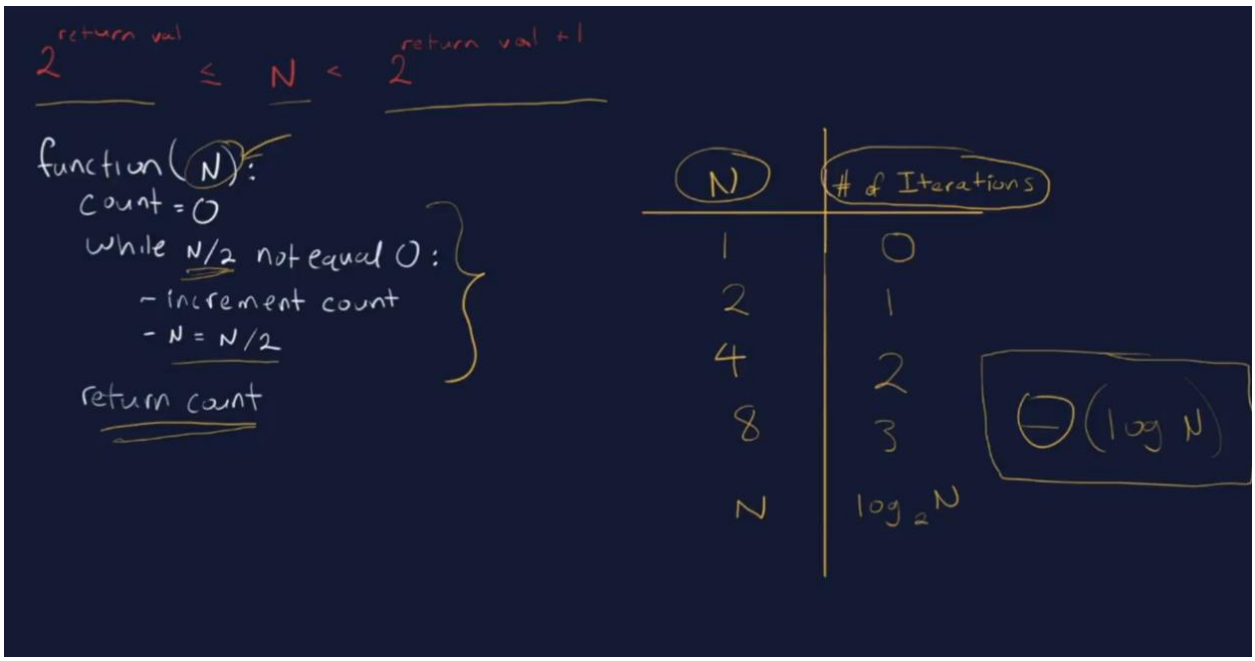
N	–vs.–	Number of Iterations
.		.
.		.
.		.
2		1
.		.
.		.
.		.
4		2
.		.
.		.
.		.
8		3
.		.
.		.
.		.
16		4
.		.
.		.
.		.
N		$\log_2 N$

As we can see, in every case, with an integer N, the loop will iterate $\log_2(N)$ times. However, because we drop constants in asymptotic notation, we would say that the runtime of this program is $\Theta(\log N)$.

But what happens when there are multiple runtime cases for a single program? We will learn about that in a future exercise.

Instructions

Play the video to go through an example of finding the big Theta runtime of a function.


2^return val ≤ N < 2^return val +1
function(N):
count = 0
while N/2 not equal 0:
- increment count
- N = N/2
return count
N
# of Iterations
1
0
2
1
4
2
8
3
N
log2N
Θ(log N)