This article will help you understand the concept of time complexity, so to do that, let’s take an example.

Let's say you have a flight to catch and you also have to do some shopping for yourself.

You walk into a shop to buy some clothes

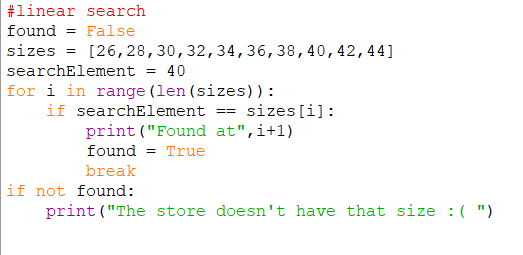
First let's get you a shirt!

There are shirts of 10 sizes sorted according to their size from 26 to 44.

You know you're a 40,what do you do now? remember to save as much time as you can!

One way is to start at the beginning and check each aisle if it is of the size 40

that's called linear search let's have a look at it.



Well that worked right? now you look great in the new shirt!

But guess what! You missed your flight because you looked at 7 different aisles before actually getting to yours.

We did tell you time was of the essence right?

Time has always been considered an important quantity to judge how efficient a process is. Even while doing our house hold chores you might hear your mom shouting from behind “ do it quick!!!”

So to explain briefly “The Faster,the Better”

For programmers, memory and time are very important resources. An algorithm is considered to be effective if it completes the task in the least amount of time and memory.

In this article, we will just look at the concept of time complexity.

But how do we calculate time taken by an algorithm? Computers execute program so quickly that we can’t even use a stopwatch to note the start and the end time of any operation (seriously if you thought this,it’s a bad idea).

Programmers use a metric called time complexity to judge how a program functions with respect to the input given. This time complexity is calculated by using basic mathematics which is very easy to understand.

Time complexity can be categorized into three parts:

1. Best Case (The best running time )
2. Average Case (The average running time)
3. Worst Case (The worst running time)

What we are really interested is to compute the worst case time complexity of algorithms so that we know what could be the maximum time our algorithm would to run ( like a wise man once said , hope for the best but prepare for the worst)

So let’s prepare for the worst . Are you guys ready??

Let’s revisit the linear search algorithm we used to buy the shirt. What linear search does is it scans your dataset from the beginning until it finds the number you asked for. Your number can lie anywhere in the dataset(fancy way to call a bunch of data) . Let’s assume you are a very lucky person and the number you want is at the very start of your dataset. Well in your very first iteration you got your shirt size and you didn’t miss the flight. This is what is referred as the Best Case . But sadly that didn’t happen. If the shirt you wanted was at the end of the dataset. Let’s assume that the length of your dataset is n. Therefore you would need n iterations to reach your preferred data. This is what is referred as the worst case time complexity . Quite intuitive right!!.

You must have observed that time complexity of an algorithm really depends on the number of iterations . If n was larger , it would have taken longer time to search . So time complexity can be represented as a function of input size. Worst case time complexity is written as O( some expression of n). For the above case it’s O(n) as n iterations would take to find the required data.

This analysis seemed very logical and we could derive at the time complexity quite intuitively. But the expression of time complexity is not always very intuitive. For example the time complexity of Binary search is O(log(n)), looks absurd right. But don’t worry we will justify it like a pro with basic mathematics.

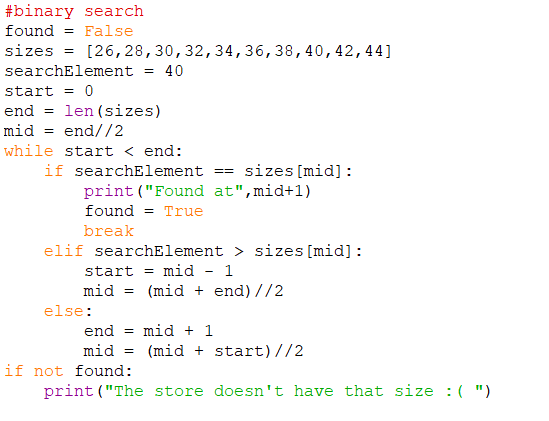
Let's reboot the airport scenario and try it another way

Now instead of looking for your shirt from the start you just go to the middle and see if that is your size.

What happens when you see the middle aisle has is for size 34?

You move to the center of the shirts(from 36 to 44) towards aisle 44!

What is the middle for sizes 36 to 44? voila!



Now you're at 40 after just looking at 2 aisles.

You just might have enough time to board the flight now!

Now let’s see what actually happened behind the scenes, how did we go from taking 7 steps to 2 steps?

When you’re using a “linear search” algorithm, in the worst case, the element to be found could be at the opposite end from where you started right? Taking the previous example as reference, if you’re a size 44 and start looking at the end where the sizes are 26,28 and so on, before reaching your actual size, you will have looked at all the available sizes right ? Which was a huge waste of time even for a small amount of inputs like 10 sizes.

Binary search uses an interesting approach. It targets the middle element of your dataset and compares if your data is lesser , greater or equal to the middle element. If it is lesser , only the left part of the dataset will be considered for the next searching process and the same process will be applied again. If it is greater then only the right part of the dataset will be considered for the next searching purpose and the same process will be applied at the right half again and obviously,if it is equal,you stop searching. So at each iteration your dataset is halved hence we don’t have to waste time on many of the unwanted elements.

But let’s understand why is the worst case complexity O(log(n)).

The worst case can be if your shirt was the first or the last one in the order since you directly go to the middle.

At every iteration the size of the dataset is halved

Iteration 1 => Size = n/2 (Half of n/1) = n/21

Iteration 2=> Size=n/4 (Half of n/2) = n/22

Iteration 3=> Size= n/8(Half of n/4) = n/23

and so on till the size of dataset reduces to 1 (Since we are considering the worst case scenario)

If you look at the denominator , it’s increasing with power of 2.

So for Iteration k=> Size=n/2k

Since according to our worst case scenario at the kth iteration the dataset size should be 1

It’s safe to say n/2k = 1

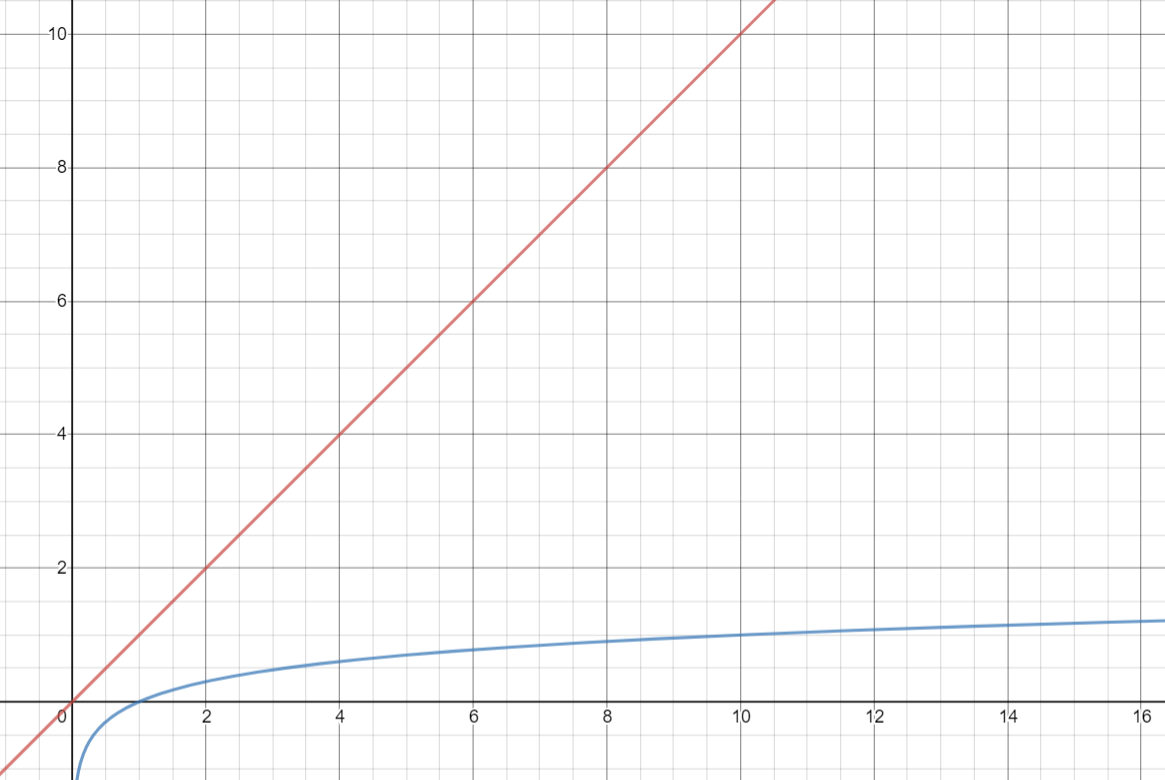
This implies n = 2k

Therefor after taking log on both sides we get k=log2(n)

Therefore the time complexity is O(k) which is O(log(n))

Wasn’t that simple.

Since n>log(n) for all n>=1,we can establish that binary search will work better than linear search in most of the cases.



This graph helps us to compare the two complexities. (The red line represents O(n) which is linear in n and the blue curve represents O(log(n)) which is logarithmic in nature)

Hence it can be noticed that time complexity helps to do analysis of different algorithms and is really helpful in comparison of algorithms which are meant to do similar operations.

Computers in actual industries process a lot of data. For example an airport has to take care of a huge dataset because many people board flights. In cases like these it is very important to choose the right algorithm to do your task and analysis of time complexities of various algorithms can be really helpful.

Time complexity is something that all of us have been adapted to, now is the time to make our computers faster than ever and this was a small way of taking two simple algorithms and learning how to chose a better algorithm by comparing different algorithms that take different routes to get to the same solution.