Data structures and algrithms

**Big O Notation**

With Big O Notation we can compare the codes.

Imagine we have multiple implementations of the same function.

How can we determine which one is the "best?"

Rather than saying your code is excellent, brilliant, we have a precise vocabulary to talk about how code performs.

int addUpTo(int n) {  
 int total = 0;  
 for (int i = 0; i < n; i++) {  
 total += i;  
 }  
 return total;  
}

int addUpTo(int n) {  
 return n \* (n + 1) / 2;  
}

What does better mean?

1. Faster?
2. Less memory intensive?
3. More Readable?

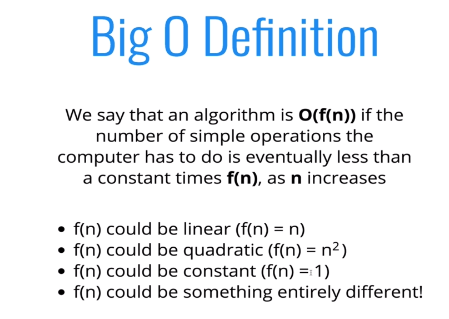
Adding adding = new Adding();  
long l1 = System.*currentTimeMillis*();  
  
adding.addUpTo(10000000000l);  
long l2 = System.*currentTimeMillis*();  
  
System.*out*.println((l2 - l1)/1000 + " seconds");

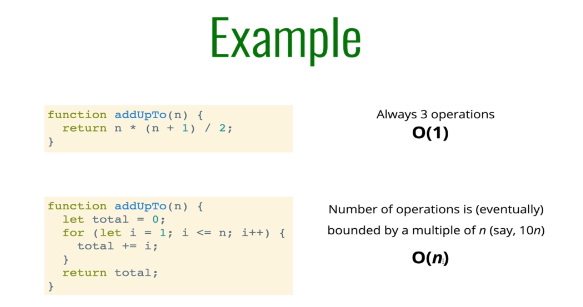
First one takes 7 seconds.

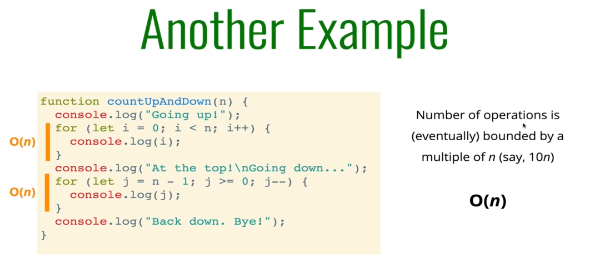
Second one take less than second.

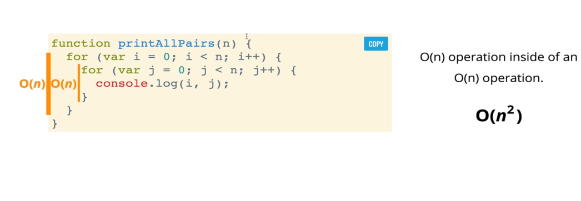
Now you can not do this every time, Every time you cannot go and calculate time. Every machine will give different kind of runnig times. So, it is not reliable approach. There should be some other way with which we can tell what is faster than other.

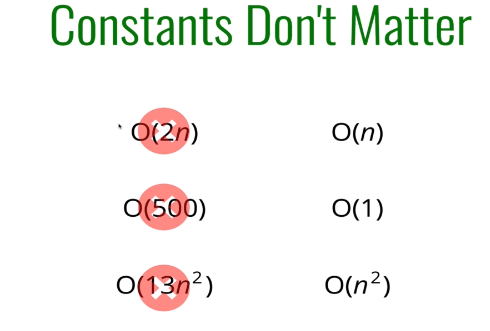
So, instead of counting number of seconds which will alwasys be a variable value how about counting the number of simple operations the computer has to perform as this will remain always constant regardless of the machines.

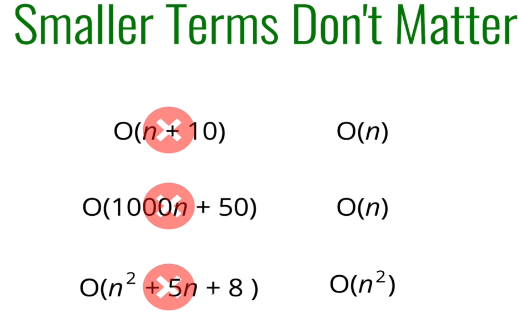


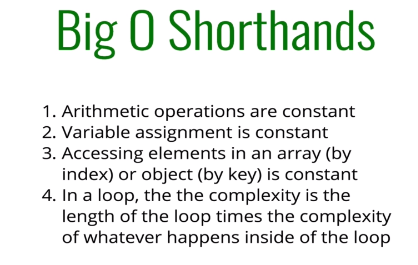


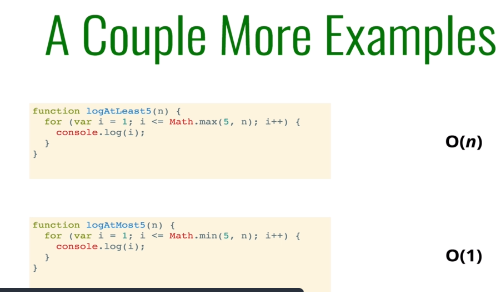


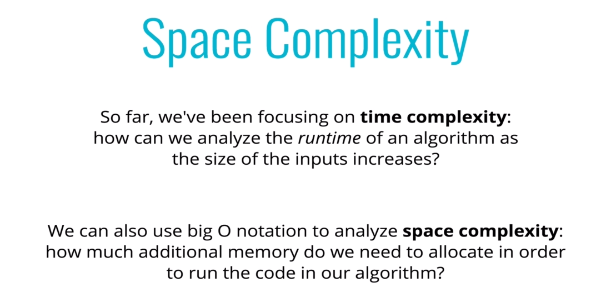


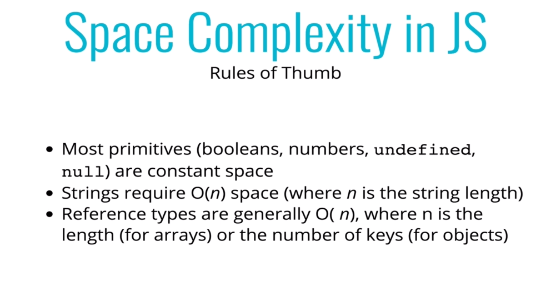


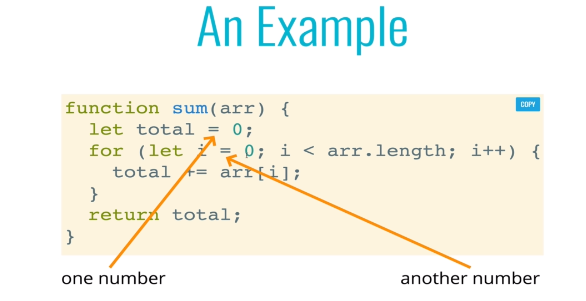




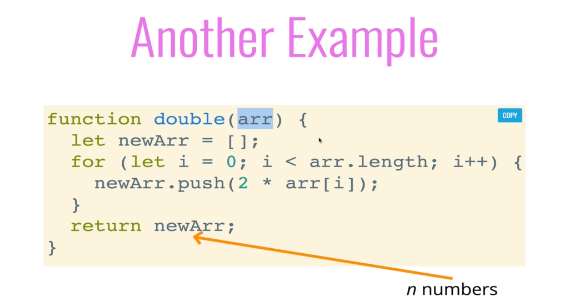




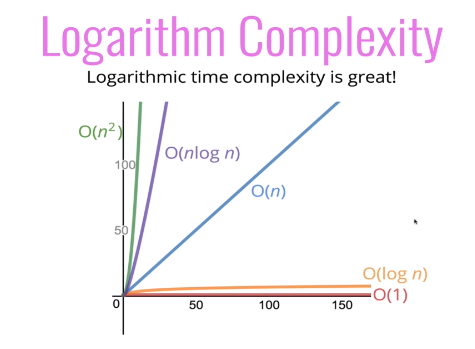


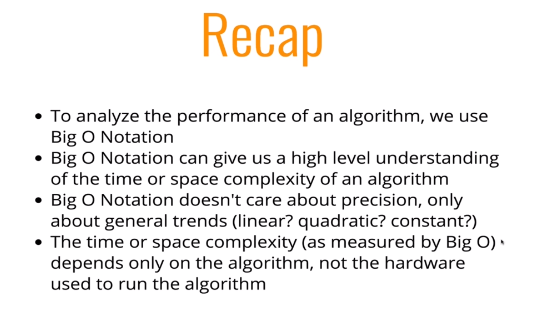


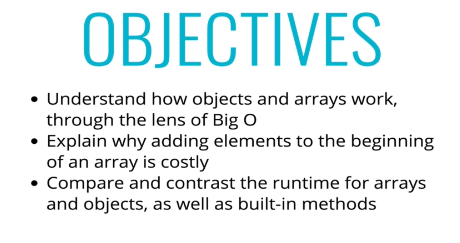
**O(1)**



**O(n)**







**Objects**

Objects are unordered datastructures and everything is stored as key value pair. They work well when you don’t need order. They work well when you need fast access/insertion and removal. So, basically Objects are very fast.

Insertion: O(1)

Removal: O(1)

Searching: O(N)

Access: O(1)

When we don’t need ordering, Objects are the best choices. For insertion, Removal, Searching Objects are O(1) because we use key for that in random order. But to search it is O(N), i.e, it grows linear with time.

**Arrays**

Arrays, unlike Objects are ordered.

Insertion : It depends

Removal: It depends

Searching: O(N)

Access : O(1)

Insertion: If you are adding element at the end of an array, it will take constant time always. You can assu,e adding in an Object. In this case it is always O(1).

Problem arisies when you try to insert into the beginning because shifting is done. In this case it is roughly O(N)

Same goes for removing, If you remove an element fro the last it will take O(1) always. But if you try to remove from the beginning shifting is required and hence it takes O(N) in this case.

