Name : Sujeewa Sandeepa Kularathna

Date : 2021-07-17

Data structures – part 01

What is a data structure?

It is a way organizing data so that it can be used effectively.

What is an abstract data type?

It’s an abstraction of a data structure which only provides the interface to which a data structure must adhere to.

An abstract data type only defines if how a data structure should behave and what methods it should have.

A nice example :

If ‘vehicle’ is an abstract data type then a ‘car’ is a data structure.

Here the abstract type vehicle only defines few things about the car, like it should be used for transportation. And that it can take you from one place to another place. But car is a data structure it has much more to it, the abstract type ‘vehicle’ doesn’t say how ‘car’ should take you from one place to another. It’s defined in data structure itself.

Big-O Notation

Big-O notation gives an upper bound of the complexity in the worst case.

So what this means is say if you want to find a particular number in a list of random unique numbers, so now in this case the worst case scenario is when the number we need to find is at the end of the list.

Big-O Properties

Actually here as we are considering the worst outcome, in most cases we can do this.

So if ‘f’ is a function that describes the running time of a particular algorithm for an input size of ‘n’,

$f(n) = 7(\log)^3 + 15n^2 + 2n^3 + 8$

So if we want to write this in Big-O notation we can do it like this, so first here worst case scenario is when n is infinity right? That means input size is infinite.

Then we can simplify the above function to something like this.

$O(f(n)) = O(n^3)$

What we did here is, as n is infinity we can just disregard all the constants. As no constant is going to alter the infinity.

(constant * infinity = infinity, constant + infinity = infinity same for division and subtraction)

A function that runs in constant time - O(1)

Say there is something like this,

```
i := 0

while i < 11 Do

    i = i + 1
```

Here there is no such thing as ‘n’, everything here are just constants. So we say this runs in constant time.

A function that runs in linear time – O(n)

In this case it’s different. Now something that runs in linear time is like this,
(**n** is the input size)

example 01.

```
i = 0
While i < n Do
      i  = i + 1
f(n) = n
```

Here see that each time we iterate we increment i by 1. So we can say f(n) = n, f(n) is the time or the no of steps for the process.

example 02.

Here again it’s a little different because now we increment it by 3. So now the same this loop is going to be finished 3 times faster. Because the two loops are identical other than that. So here f(n) = n/3.

```
i = 0
While i < n Do
      i  = i + 3
f(n) = n/3
```

this in Big-O notation,

(Again as we are considering the worst case scenario in Big-O notation, assume that n is infinity. Writing upper case ‘O’ is how we indicate that it is written in Big-O notation.)

```
O(f(n)) = O(n)
```

Here there is a single n.

A function that runs in quadratic time – O(n^2)

example 01.

This one is simple and obvious.

```
For ( i := 0; i < n; i++)
      For ( j :=0; j < n; j = j + 1)

f(n) = n * n  = n^2
```

This is a nested loop, so what happens here is we go from 0 to n-1 one by one. And with each iteration we initiate another loop and here too we go from 0 to n-1 times.
It's easier to understand if you can imagine it like this, so if first iteration is the number of rows and then the second iteration is the number of columns in each row it can be shown like this, I hope you understood what I mean.

R0 | C0 C1 C2 C3 C4
R1 | C0 C1 C2 C3 C4
R2 | C0 C1 C2 C3 C4
R3 | C0 C1 C2 C3 C4
R4 | C0 C1 C2 C3 C4

So this is literally a loop inside a loop. Every time the outer loop runs, inside it the inner loops runs. So it can be said that it is n work done n times.

So the time taken taken for this can be written like this, `f(n) = n * n`

So this in Big-O notation can be written like this,

$O(f(n)) = O(n^2)$

example 02.

This one is a little different but interesting,

```
For ( i := 0; i < n; i++)
      For ( j :=i; j < n; j = j + 1)
```

Here in the inner loop instead of j := 0 now it's replaced with 'i'. So this changes something and it becomes a little harder to figure out if how time taken for this will be turned out.

Here the outer loop is identical to the one before, so yes it will function the same. But the inner loop it's different. The number of times the inner loop going to run is determined by this 'i'.

So if 'i' = 0 as in the previous example we will be doing 'n' work
But what if 'i' = 1, then the times we run is going to be one less as the previous case, so now we will be doing 'n-1' work.
When it's 'i' = 2, just like before the number of times we run going to be one less. So now it'll be 'n-2' work.

I can simplify this with a diagram just like before too. So let's assume n = 4.

```
when 'i' = 0
R0 | C0 C1 C2 C3 C4
'j' values are = 0, 1, 2, 3, 4

when 'i' = 1
R1 | C0 C1 C2 C3
'j' values are = 1, 2, 3, 4

when 'i' = 2
R2 | C0 C1 C2
'j' values are = 2, 3, 4
```

I hope you understood this,

So now the total work done is something like this,
`(n) + (n-1) + (n-2) + . . . + 3 + 2 + 1`
And this can be written as,
`n(n+1) / 2`

If you have done combined maths for Als this will be very familiar.

So if we simplify this further,

`n(n+1)/2`
`(n * n)/2 + n/2`

For Big-O notation as we only consider the worst case scenario, it's when 'n' becomes largest. So if 'n' is a very large value then 'n^2' is going to much more bigger than 'n' so we can disregard 'n'. So if to write this in Big-O notation it'll look like this.

$O(f(n)) = O(n^2/2)$
And as again '2' is a constant we can disregard too,

So final Big-O notation for the above function is going to look like this,

$O(f(n)) = O(n^2)$