Analysis of Iterative Algorithms

We know how to derive the growth function of an algorithm and also to represent that growth function using notations like O, Θ and Ω. This chapter has some examples of iterative algorithms to make you comfortable with the entire process.

So, let's obtain the growth functions of the following examples and analyze their running times.

Example 1

In the Analyze Your Algorithm (/course/algorithms-analyze-your-algorithm/) chapter, we derived the growth function of the following algorithms:

```
DEMO(A)
  for i in 1 to A.length
    for j in i to A.length
      x = a[i]+a[j]
```

and

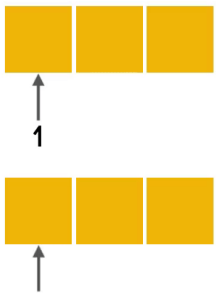
```
SUM(A)
  sum = 0
  for i in 1 to A.length
    sum = sum+A[i]
```

The cost function of the `DEMO` was $C = n^2\left(\frac{c_2+c_3}{2}\right) + n\left(c_1 + \frac{3c_2+c_3}{2}\right) + (c_1 + c_2)$, resulting in $\Theta(n^2)$ running time (shown in the chapter Growth of a Function (/course/algorithms-growth-of-a-function/)).

The `SUM` had a cost function of $C = n(c_2 + c_3) + (c_1 + c_2)$ with a running time of $\Theta(n)$ (shown in the chapter Growth of a Function (/course/algorithms-growth-of-a-function/)).

Without all the mathematics, we can also see that the `SUM` function has only one loop which iterates over all the elements of the input (size n), so it has linear running time ($\Theta(n)$).

In the `DEMO` function, there are two nested loops. So, for each iteration of the outer loop, the inner loop is also going to iterate. In other words, for each element of the outer loop (size n), the inner loop is again going to iterate over the array 'A' (maximum size n). Thus, it gives a quadratic running time of n*n i.e., $\Theta(n^2)$.



Let's look at some more examples.

Example 2

```
EXAMPLE2 (A)
  for i in 1 to A.length
    print i
  for i in 1 to A.length
    print i
```

	Cost	Times
EXAMPLE2 (A)		
for i in 1 to A.length	c_1	n+1
print i	c_2	n
for i in 1 to A.length	c_3	n+1
print i	c_4	n

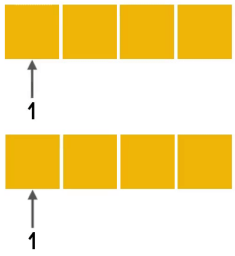
The cost function

$$C = c_1(n+1) + c_2(n) + c_3(n+1) + c_4(n)$$

$$\text{or, } C = n(c_1 + c_2 + c_3 + c_4) + c_1 + c_3$$

The cost function is linear in n and thus we can easily say that its running time is $\Theta(n)$.

There are two loops which are independent of each other (they are not nested). Thus, the first loop will iterate over each element of the array once (size n) with a linear time and then the second loop will iterate again in linear time and thus a total of n+n = 2n, which is also linear.



So we have seen that when we nested two loops, we got a quadratic running time and when the loops were independent of each other, we got a linear running time.

Let's look at the example of getting the running time of an algorithm with three nested loops.

Example 3

```
EXAMPLE3 (A)
  for i in 1 to A.length
    for j in 1 to A.length
      for k in 1 to A.length
        print k
```

	Cost	Times
DEMO (A)		
for i in 1 to A.length	c_1	n+1
for j in 1 to A.length	c_2	$\sum_1^n n+1$
for k in 1 to A.length	c_3	$\sum_1^n \left(\sum_1^n n+1\right)$
print k	c_4	$\sum_1^n \left(\sum_1^n n\right)$

The loop `for i in 1 to A.length` is going to run n+1 times (n times for size n and one more time when it will check the condition and will fail).

Then the loop `for j in 1 to A.length` will again run n+1 times for each iteration of the `for i in 1 to A.length` loop i.e., a total of n*(n+1) times.

Similarly, the third loop will run again n+1 time for each iteration of the second loop i.e., n*(n*(n+1)) times.

Thus, we will get a cubic running time of $\Theta(n^3)$.

Example 4

```
EXAMPLE 4(A)
  for i in 1 to A.length
    for j in 1 to A.length
      print i

  for i in 1 to A.length
    print i
```

Let's break the above function into two parts - one with two nested loops and another with a single loop. We know that the first part will have a $\Theta(n^2)$ running time and the second part will have $\Theta(n)$ running time, thus a total of $\Theta(n^2) + \Theta(n)$ running time. As stated in the earlier chapters, we can ignore the term with lower order and thus write the running time as $\Theta(n^2)$.

Now, you have a strong grip over the analysis of the iterative algorithms. In the next chapter (/course/algorithms-recurrences/), we are going to deal with recursive algorithms.

> “To bring up a child in the way he should go, travel that way yourself once in a while.”
>
> - Josh Billings

PREV (/course/algorithms-growth-of-a-function/) (/course/algorithms-recurrences/) NEXT