CST8130 – Data Structures

September 12, 2017

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Big O and Efficiency Concepts

Algorithm Efficiency

Why does it matter?

How do you compare different algorithms?

- ***processing***
- ***memory***

How do you measure?

Linear Loops

Case 1:
```
for (int i=0; i<n, i++)
    // do something
```

Case 2:
```
for (int i=0; i<n; i+=2)
    // do something
```

Case 3:
```
for (int i=1; i<n; i*=2)
    // do something
```

Value of *n*	Case 1 Iterations	Case 2 Iterations	Case 3 Iterations
10	10	5	4
20	20	10	5
100	100	50	7
10000	10000	5000	11

Big-O Principle

Big-0 measures the effect as you scale to larger values of n

We don't care so much that a loop required 100 iterations, but that the loop required twice as many iterations when we doubled n. We write $O(n)$.

So, case 1 and case 2 have the same Big-O because their behaviour was the same – case 3 has a different Big-O – it's behaviour was clearly different ($O(\log_2 n)$)

More Loops

Case 1:

```
for (int i=0; i<n, i++)
  for (int j=0; j<n; j++)
      // do something
```

$O(n^2)$

Case 2:

```
for (int i=0; i<n; i+=2)
      for (int j=0; j<n; j+=2)
          // do something
```

$O(n^2)$

Case 3:

```
for (int i=1; i<n; i*=2)
    for (int j=0; j<n; j++)
        // do something
```

$O(n \log_2 n)$

Value of *n*	Case 1 Iterations	Case 2 Iterations	Case 3 Iterations
10	100	25	40 (10 * 4)
20	400	100	100 (20 * 5)
100	10,000	2,500	700 (100 * 7)
10,000	100,000,000	25,000,000	100,000 (1000*10)

Ranking

Commonly used Big-O values to express algorithmic performance (from best to worst)

- $O(1)$
- $O(\log_2 n)$
- $O(n)$
- $O(n \log_2 n)$
- $O(n^2)$
- $O(n^3)$
- …
- …
- $O(n^k)$
- $O(2^n)$

Simplifying Big-O

What if we knew the number of iterations could be expressed by $f(n) = n((n + 1) / 2)$ – what is Big-O of this algorithm?

$f(n) = n\ (n/2 + ½)$

$f(n) = n^2/2 + n/2$

…Big-O is same as $n^2 + n$…….but as n increases

…Big-O is same as n^2

…O($n^{2)}$

Find Big-O

```
i=1
while (i<=n) {
  j=1
  while (j <= n) {
     print (i, j)
     j = j+1
  }
  i = i+1
}
k=n
while (k>0) {
   print (k)
   k=k/2
}
```

inside loop : n

outside loop : n

divide by 2 loop: $\log_2 n$

So algorithm is:

$n * n + \log_2 n$

$O(n^2)$

Algorithm Efficiency Summary

Remember big-O measures algorithm “iteration” efficiency

Another factor would be how much memory is used

Often algorithms that are better in big-O efficiency, use more memory……..if you want to use less memory, you often sacrifice big-O efficiency.

Questions?