Growth Functions and Asymptotic Analysis (I)

Instructor: Krishna Venkatasubramanian

CSC 212

Announcements

- **Go to office hours.**
- Next Quiz Sept 24 (next Tuesday).
- Everything **covered on Sept 12 and this week** are fair game for the quiz.
- Same format as today's quiz.

Running Time of Algorithms

- Running time of algorithm determines how **“quickly”** it executes.

- **Computed based on # of basic steps** in the algorithm that are executed
  - Loops result in repeated computational sets leading to larger running time than those without

- **Size of the inputs can also affect number of basic steps**
  - Sorting longer arrays need more time than shorter ones!
  - In such cases, **size of input usually dictates # of basic steps!**

Runtime affected by # of Basic Steps

- Two algorithms for performing the same tasks can have different running times **depending upon # of steps it has**
- Here, the size of the input is the same --- **1 number** --- but the presence of loops dictates running time.

```python
import time

def SumOfN(n):
    start = time.time()
    theSUM = 0

    for i in range(1,n+1):
        theSUM +=i

    end = time.time()

    return theSUM, end-start


def directSumOfN(n):
    start = time.time()
    theSUM = (n*(n+1))/2
    end = time.time()

    return theSUM,end-start


def main():

    print("Sum of N with for loop")
    for i in range(5):
        print("Sum is %d required %10.7f seconds"%SumOfN(1000000))

    print("Sum of N function direct ")
    for i in range(5):
        print("Sum is %d required %10.7f seconds"%directSumOfN(1000000))
```

w/ FOR LOOP

DIRECT SUMMATION

```
Sum of N with for loop
Sum is 500000500000 required  0.0627120 seconds
Sum is 500000500000 required  0.0636330 seconds
Sum is 500000500000 required  0.0593448 seconds
Sum is 500000500000 required  0.0563250 seconds
Sum is 500000500000 required  0.0615969 seconds
Sum of N function direct
Sum is 500000500000 required  0.0000012 seconds
Sum is 500000500000 required  0.0000000 seconds
Sum is 500000500000 required  0.0000000 seconds
Sum is 500000500000 required  0.0000012 seconds
Sum is 500000500000 required  0.0000007 seconds
```

Runtime affected by input size: Insertion Sort (Recap)

Assume ***n*** elements

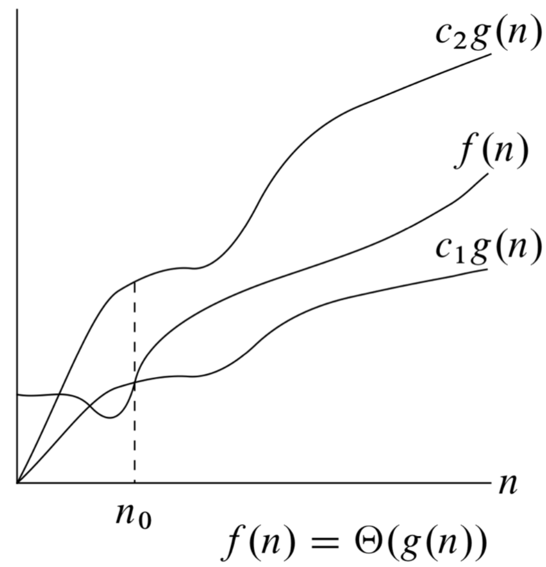
```
def InsertionSort(A)
        for k in range(1,len(A))

                key = A[k]

                i = k -1

                while i > 0  and   A[i] > key

                        A[i+1] = A[i]

                        i = i - 1

                A[i+1] = key
```

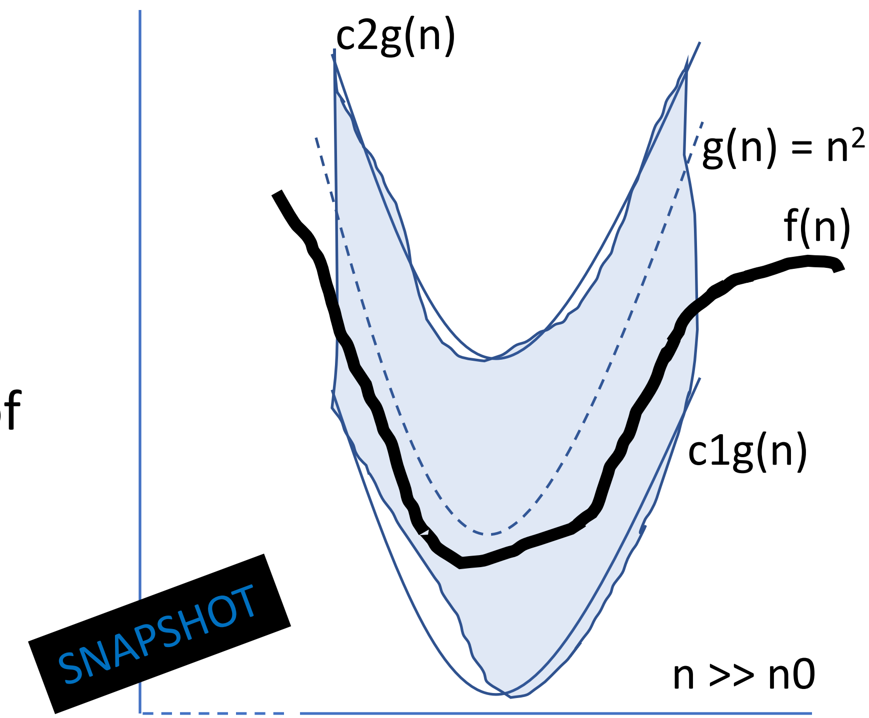
Cost	Times
c1	n
c2	n-1
c4	n -1
c5	$\sum_2^n t_j$ WHY?
c6	$\sum_2^n (t_j - 1)$
c7	$\sum_2^n (t_j - 1)$
c8	n-1

$$T(n) = c1 * n + c2(n-1) + c4(n-1) + c5\sum_2^n t_j + c6\sum_2^n (t_j \; -1) \; + c7\sum_2^n (t_j \; -1) + c8(n-1)$$

Θ-notation

- **Definition:** For a given function *g(n)*, $\Theta(g(n))$ is a **set of functions** such that
  - $\Theta(g(n))$ = { *f(n):* there exists positive constants c1, c2, and n0 s.t. 0≤ $c1g(n) \leq f(n) \leq c2g(n)$ for all n ≥ $n0$ }
- This is called an asymptotic tight-bound for f(n)
  - Really, f(n) ∈ $\Theta\left(g(n)\right)$
- For all values of n ≥ $n0$ the value of f(n) is between the c1g(n) and c2g(n) belt.
- Focus on **large values of n**





Θ-notation: Example

- Assume $f(n) = 1/2n^2 - 3n$
- We say $f(n) = \Theta(n^2)$, if this is true then
  - $c1n^2 <= 1/2n^2 - 3n <= c2n^2$
  - **$c1 <= 1/2 - 3/n <= c2$**

Remember:
c1,c2 and n are positive constants

- The right inequality is true for n>=1 and c2 >= ½
- The left inequality is true for n>=7 and c1 <= 1/14
- Thus if we choose
  - c1 = 1/14, c2 = ½, and n0 = 7 we can make the inequality true

- Thus $f(n) = \Theta(n^2)$

- **Note, other c1, c2, and n0 may also exist that make the inequality true**
- Suffice it to say, that we can find one groups of values

Θ-notation

- In the running time of an algorithm, lower order terms are ignored.
  - For large values of n, the lower order terms become minuscule compared to the highest-order term
  - E.g., For $T(n) = an^2 + bn + c$, the **value of n^2 will dominate** values of $b*n$ or c or a for large values of n

- More generally, for any polynomial
  $p(n) = \sum_{i=0}^{d} a_i \, n^i$ where a_i is a constant and $a_d > 0$,
  $p(n) = \Theta(n^i)$

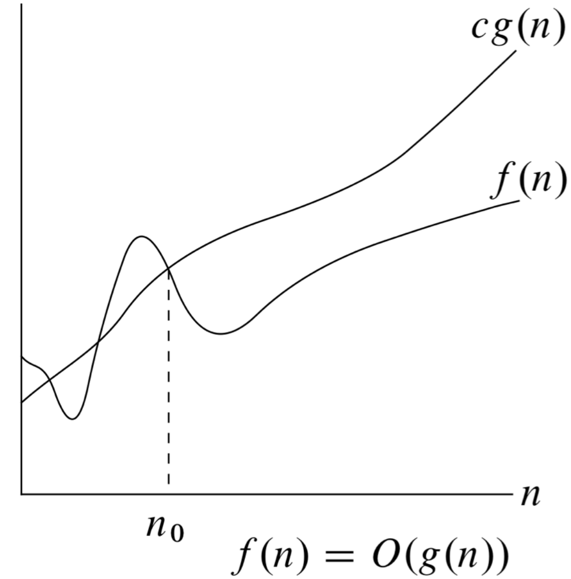
- Similarly, for a zero—degree polynomial *q(n)* or a constant function --- e.g., a given algorithm step

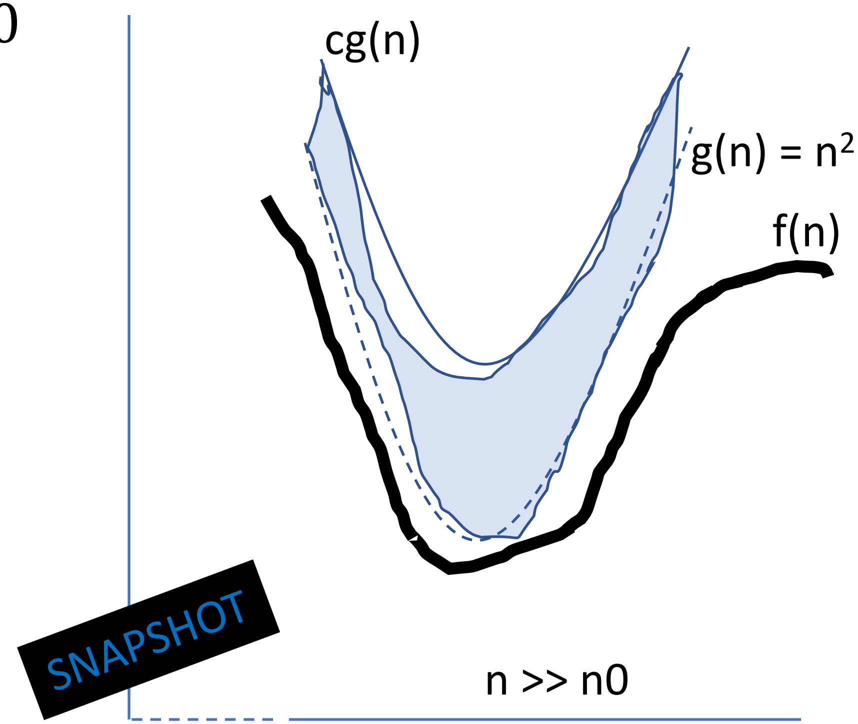
  $q(n) = \Theta(n^0) = \Theta(1)$

- Called the **Asymptotic Tight-bound!**
  - Asymptotic means for large n
  - Tight-bound because we have found the function that describes the algorithm's running time to with a constant multiple above and below

O-notation

- **Definition:** For a given function *g(n)*, $O(g(n))$ is a **set of functions** such that
  - $O(g(n))$ = { *f(n):* there exists positive constants c, and n0 s.t. $0 \leq f(n) \leq cg(n)$ for all n $\geq n0$ }
- This is called an asymptotic upper-bound for f(n)
- For all values of n $\geq n0$ the value of f(n) is always <= cg(n)
- Focus on **large values of n**





O-notation: Example

- Assume *f(n) = 1/2n² – 3n*
- We say f(n) = $O(n^2)$, if this is true then
  - $0 <= 1/2n^2 – 3n <= cn^2$
  - **$0 <= 1/2 - 3/n <= c$**

Remember:
c and n are positive constants

- The right inequality is true for n>=1 and c >= 1/2
- The left inequality is true for n >=4
- Thus if we choose
  - c = ½, and n0 = 4 we can make the inequality true

- Thus **f(n) = $O(n^2)$**

- Called the **Asymptotic Upper-bound!**
  - Asymptotic means for large n
  - Tight-bound because we have found the function that describes the algorithm's running time to with a constant multiple above

Practice

- What is the Asymptotic Relationship (O or $\Theta - notation$) between

- n^k *in terms of* c^n (assuming $c > 1$ and $k > 1$)
- $\lg n^{\lg 17}$ *in terms of* $\lg 17^{\lg n}$
- $\log_2 n$ *in terms of* $\log_8 n$ --- [tricky work it out]
- $3n\log_8 n$ *in terms of* $n^3 \lg n$


That's all Folks!
Any Question ?