CS-218 DATA STRUCTURES AND ALGORITHMS

LECTURE 5

BY UROOJ AINUDDIN

IN THE LAST LECTURE…

- We arranged functions in order of their growth rates.
- We realized that algorithms with slower growth rates are better than algorithms with faster growth rates.
- We learned that the coefficients can be ignored in asymptotic analysis.
- We agreed that for fast execution, we need better algorithms more than we need faster machines.

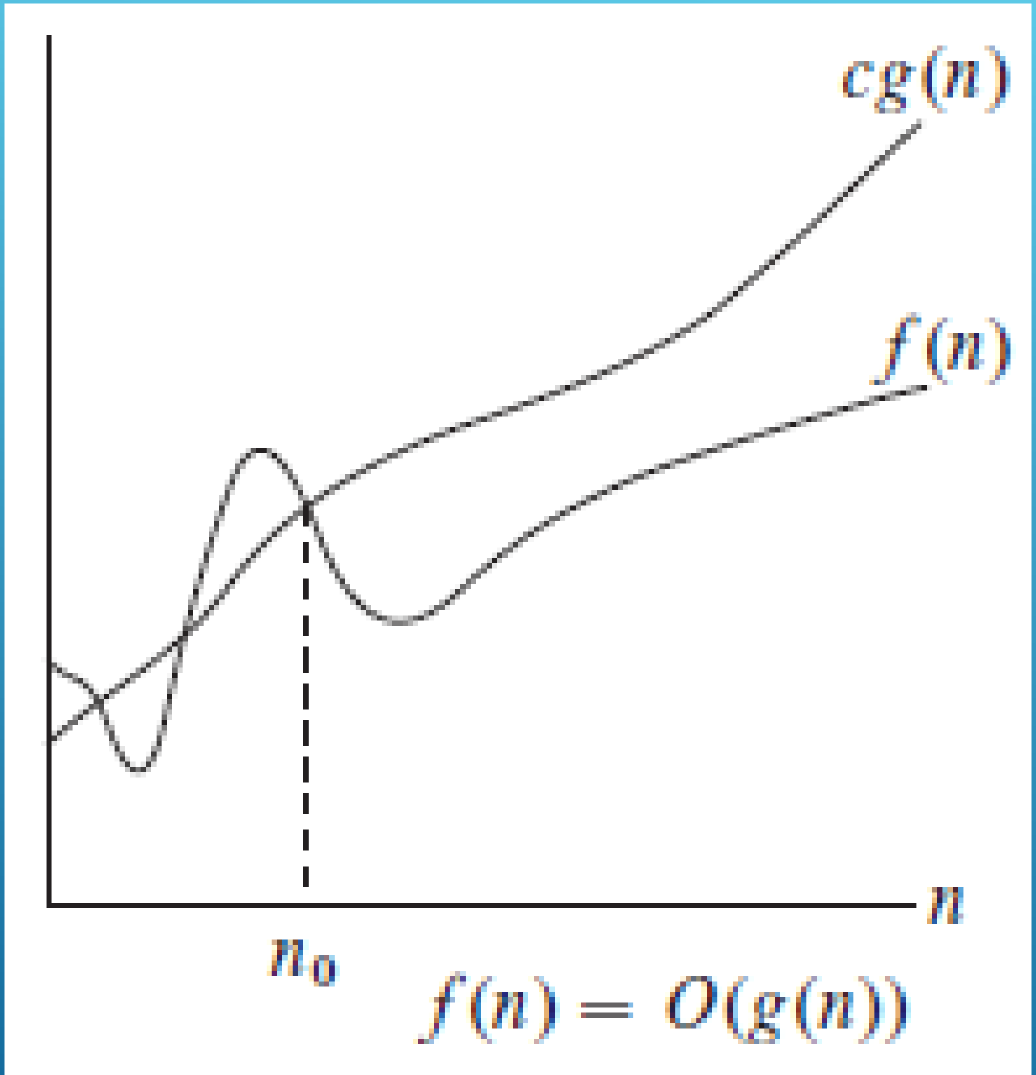
ANALYSIS OF ALGORITHMS (ASYMPTOTIC ANALYSIS)

BOOK 1 CHAPTER 4

BOOK 2 CHAPTER 2

- When we look at input sizes large enough to make only the order of growth of the running time relevant, we are studying the asymptotic efficiency of algorithms.
- The following notations have been defined for asymptotic analysis:
  - Big – O notation O
  - Big – Omega notation Ω
  - Big – Theta notation Θ
  - Little – O notation o
  - Little – Omega notation ω

ASYMPTOTIC NOTATIONS



- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ *is* $O(g(n))$ if there are positive constants c and n_0 such that $f(n) \le cg(n)$ for $n \ge n_0$.
- O notation provides an upper bound on running time of an algorithm.

BIG – O NOTATION

- $f(n) = 8n + 5$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$8n + 5 \leq 8n + 5n, \forall n \geq 1$$
$$8n + 5 \leq 13n, \forall n \geq 1$$

- This is true for all values of n greater than 0, so the smallest possible value of n, $n_0 = 1$.
- Take $c = 13$
- So $g(n) = n$
- $f(n)\ is\ O(n)$
- The linear function is an upper bound on the growth of $f(n)$.

METHOD 1

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- $f(n) = 8n + 5$
- We need to prove $f(n) \le cg(n)$
- We know that

$$8n + 5 \le 8n + n$$
$$n \ge 5$$

METHOD 2

FIND $g(n)$ SUCH THAT $f(n)\ is\ O(g(n))$.

- $f(n) = 8n + 5$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$8n + 5 \leq 8n + n, \forall n \geq 5$$
$$8n + 5 \leq 9n, \forall n \geq 5$$

- The smallest possible value of n, $n_0 = 5$.
- Take $c = 9$
- So $g(n) = n$
- $f(n)\ is\ O(n)$
- The linear function is an upper bound on the growth of $f(n)$.

METHOD 2

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- $f(n) = 3n^2 + 4n + 7$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$3n^2 + 4n + 7 \leq 3n^2 + 4n^2 + 7n^2$$
$$3n^2 + 4n + 7 \leq 14n^2$$

- This is true for all values of n greater than 0, so the smallest possible value of n, $n_0 = 1$.
- Take $c = 14$
- So $g(n) = n^2$
- $f(n)\ is\ O(n^2)$
- The quadratic function is an upper bound on the growth of $f(n)$.

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- $f(n) = 3n\log_2 n + 2n$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$3n\log_2 n + 2n \leq 3n\log_2 n + 2n\log_2 n$$

$$3n\log_2 n + 2n \leq 5n\log_2 n$$

$$n \geq 1 \qquad \log_2 n \geq 1$$

$$n \geq 2$$

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- $f(n) = 3n\log_2 n + 2n$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$3n\log_2 n + 2n \leq 5n\log_2 n, \forall n \geq 2$$

- The smallest possible value of n, $n_0 = 2$.
- Take $c = 5$
- So $g(n) = n\log_2 n$
- $f(n)\ is\ O(n\log_2 n)$
- The linearithmic function is an upper bound on the growth of $f(n)$.

Find another pair of c and n_0 to satisfy $f(n)\ is\ O\big(g(n)\big)$

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- $f(n) = 2^n + n^2 + 3$
- We need to prove $f(n) \leq cg(n)$
- We know that

$$2^n + n^2 + 3 \leq 2^n + 2^n + 2^n$$

$$2^n \geq n^2$$

$$\log_2 2^n \geq \log_2 n^2$$

$$n \geq 2\log_2 n$$

$$n \geq 4$$

$$2^n \geq 3$$

$$\log_2 2^n \geq \log_2 3$$

$$n \geq \log_2 3$$

$$\log_2 3 = 1.5849625$$

$$n \geq 2$$

n	2 * log (n)
1	0
2	2
3	3.169925
4	4
5	4.643856
6	5.169925
7	5.61471
8	6
9	6.33985
10	6.643856
11	6.918863
12	7.169925

FIND $g(n)$ SUCH THAT $f(n)$ is $O(g(n))$.

- $f(n) = 2^n + n^2 + 3$
- We need to prove $f(n) \le cg(n)$
- We know that

$$2^n + n^2 + 3 \le 2^n + 2^n + 2^n, \forall n \ge 4$$
$$2^n + n^2 + 3 \le 3(2^n), \forall n \ge 4$$

- The smallest possible value of n, $n_0 = 4$.
- Take $c = 3$
- So $g(n) = 2^n$
- $f(n)\ is\ O(2^n)$
- The exponential function is an upper bound on the growth of $f(n)$.

FIND $g(n)$ SUCH THAT $f(n)\ is\ O\big(g(n)\big)$.

- The Big – O notation of a function is its fastest growing term disregarding the constants.
- $f(n) = 3^n + 2n^2 + 3$
  - $f(n) \text{ is } O(3^n)$
- For a polynomial function of degree d,

$$f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n^1 + a_0 n^0, a_d > 0$$

  - $f(n) \text{ is } O(n^d)$
- The Big – O notation provides the **tightest** asymptotic upper bound of $f(n)$.

POINT TO NOTE

- $f(n) = 5n^3 + 4n$
- The geometric and factorial functions have higher rate of growth than $f(n)$.
- All polynomial functions of degree greater than 3 have higher rate of growth than $f(n)$.
- So all functions with a higher growth rate can be the Big – O notation of $f(n)$.
- But we need the tightest upper bound, the higher growth rate function that is closest to f(n).

$f(n)\ is\ O(n!)$

$f(n)\ is\ O(a^n), \forall a > 1$

$f(n)\ is\ O\left(n^d\right), \forall d > 3$

$f(n)\ is\ O(n^3)$

$$5n^3 + 4n \le 5n^3 + 4n^3$$

$$5n^3 + 4n \le 9n^3$$

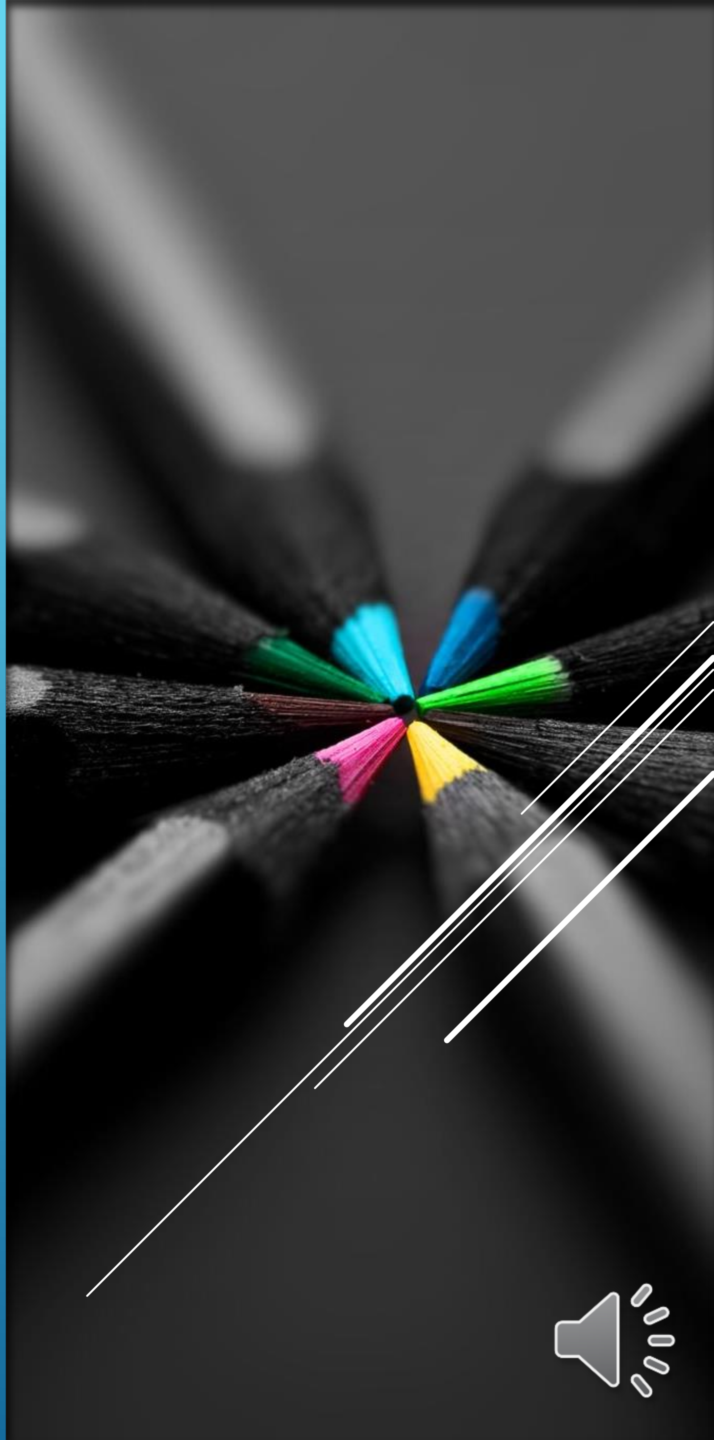
WHICH FUNCTION WOULD BE THE MOST SUITABLE FOR BIG – O?

ASYMPTOTIC CONVENTIONS

- $5n^2 + 3n \leq 8n^2$ which means $5n^2 + 3n \; is \; O(8n^2)$ but it is poor taste to mention 8,
  - $5n^2 + 3n \; is \; O(n^2)$
- $5n^2 + 3n \leq 8n^3$ which means $5n^2 + 3n \; is \; O(n^3)$ but it is poor taste to use a higher order function for Big – O notation when a lower order function is available.
  - $5n^2 + 3n \; is \; O(n^2)$
- $2n^2 \leq 3n^2 + 4\log n$ but it is poor taste to use two terms to describe Big – O notation and say $2n^2 \; is \; O(3n^2 + 4\log n)$. Instead we use a single term that would suffice.
  - $2n^2 \; is \; O(n^2)$

- If $T_1(n)$ *is* $O(f(n))$ and $T_2(n)$ *is* $O(g(n))$
  - $T_1(n) + T_2(n)$ *is* $O\big(f(n) + g(n)\big)$
  - $T_1(n) + T_2(n)$ *is* $O(\max\{f(n), g(n)\})$
  - $T_1(n) \times T_2(n)$ *is* $O(f(n) \times g(n))$
- $\log_k n$ *is* $O(n)$ for any constant k.

RULES OF BIG – O NOTATION



```
x=5

y=6

z=x+y*6

done=x>0 and x<100
```

Rule 1 – Lines of constant time:
For all code lines that execute in constant time, the Big – O notation is 1.

$$O(1)$$

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

```
def ex1(n):
    total=0
    for i in range(n):
        total+=I
    return total
```

Rule 2 – For loops:
The running time of a for loop is at most the running time of the statements inside the for loop times the number of iterations.

$$O(1 \times n)$$
$$= O(n)$$

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

```
def ex2(n):
    count=0
    for i in range(n):
        count+=l
    for j in range(n):
        count+=1
    return count
```

Rule 3 – Consecutive code fragments:
The running time of consecutive code fragments is added to give the running time of the entire code fragment.

$$O(n) + O(n)$$
$$= O(n)$$

```
def ex3(n):
    count=0
    for i in range(n):
        for j in range(n):
            count+=1
    return count
```

Rule 4 – Nested loops:
The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops.

$$O(1 \times \mathrm{n} \times \mathrm{n})$$
$$= O(n^2)$$

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

```
def ex4(n):
    count=0
    for i in range(n):
        for j in range(25):
            count+=1
    return count
```

The inner loop runs a constant number of times. Its iterations are fixed, no matter what the value of n.

$$O(1 \times 1 \times n) = O(n)$$

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

```
def ex5(n):
    count=0
    for i in range(n):
        for j in range(i+1):
            count+=1
    return count
```

Total no of iterations

$$= 1 + 2 + 3 + \cdots + n$$

$$= \frac{n}{2}(n + 1)$$

$$= \frac{n^2}{2} + \frac{n}{2}$$

$$O(n^2)$$

Value of i	Valid values of j	No of iterations
0	0	1
1	0,1	2
2	0 – 2	3
⋮	⋮	⋮
n – 1	0 – (n – 1)	n

```
def ex6(n):
    count=0
    i=n
    while i>=1:
        count+=1
        i=i//2
    return count
```

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

The while loops for these values of i:

$$n, \frac{n}{2}, \frac{n}{4}, \dots, 2, 1$$

$$\text{i.e.} \frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \dots, \frac{n}{2^{x-1}}, \frac{n}{2^x}$$

These are $x+1$ terms. So the while loop runs $x+1$ times.

$$\frac{n}{2^x} = 1$$

$$2^x = n$$

$$x = \log n$$

$$O(x+1)$$

$$= O(\log n + 1)$$

$$= O(\log n)$$

```
def ex7(n):
    count=0
    for i in range(n):
        count+=ex6(n)
    return count
```

ex6(n) is $O(\log n)$.
The loop iterates n times.

$$O(n \times \log n)$$
$$= O(n \log n)$$

FIND THE BIG – O NOTATION FOR THE GIVEN CODE FRAGMENT

- You will find a Google Form in an assignment in Google Classroom titled “Lecture 5 tasks”.
- I hope that you will be able to answer the questions posed to you after going through this video.

TASKS

SO WHAT DID WE LEARN TODAY?

- We learned about the Big – O notation.
- We learned how to find Big – O notation of functions.
- We learned the rules of the Big – O notation.
- We learned how to find Big – O notation of code fragments.

THINGS TO DO

Read

- the book!

Submit

- your answers to the tasks in this lecture.

Note

- your questions and put them up in the relevant online session.

Email

- suggestions on content or quality of this lecture at uroojain@neduet.edu.pk