BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ALGORITHM COMPLEXITY CONTINUED...

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

ASYMPTOTIC NOTATION

- In maths, what is an **Asymptote**?
- In data structures and algorithms, what is Big O notation?

BIG-OH RULES

$13n^4 - 8n^2 + \log_2 n$?

```
function findBiggestNumber(array) {
    let biggest = array[0];

    for (let i = 0; i < array.length; i++) {
        if (array[i] > biggest) {
            biggest = array[i];
        }
    }

    return biggest;
}
```

```
function findBiggestNumber(array) {
    let biggest = array[0];            [redacted]

    for (let i = 0; i < array.length; i++) {    [redacted]
        if (array[i] > biggest) {
            biggest = array[i];
        }
    }

    return biggest;                    [redacted]
}
```

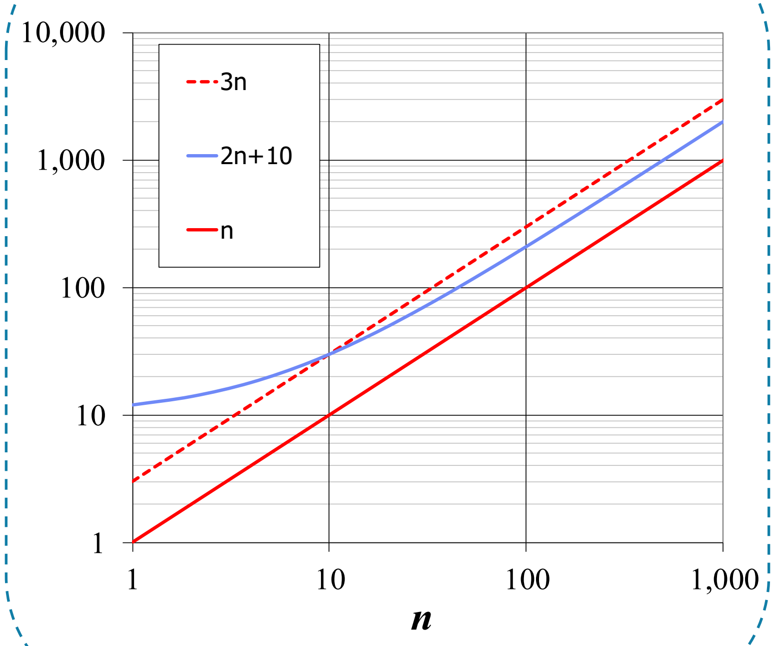
Source: https://medium.com/

```
function findBiggestNumber(array) {
    let biggest = array[0];            [redacted]

    for (let i = 0; i < array.length; i++) {    [redacted]
        if (array[i] > biggest) {
            biggest = array[i];
        }
    }

    return biggest;                    [redacted]
}
```

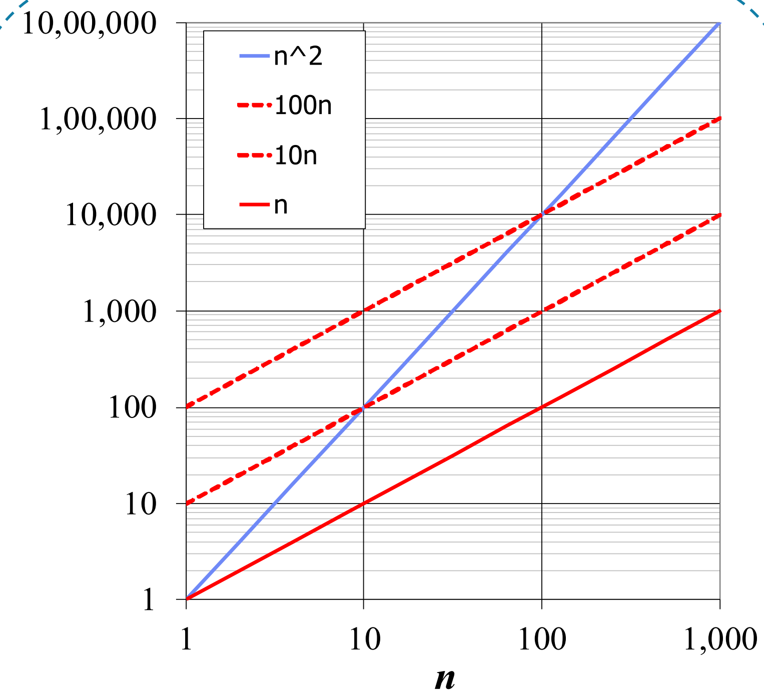
```
function containsDuplicates(array) {
    for (let i = 0; i < array.length; i++) {
        for (let r = 0; r < array.length; r++) {
            if (i === r) {
                continue;
            }
            if (array[i] === array[r]) {
                return true;
            }
        }
    }

    return false;
}
```

BIG-OH EXAMPLES

Example: $2n + 10$ is $O(n)$



What about n^2 ?



MORE BIG-OH EXAMPLES

$f(n) = 7n\text{-}2$

$7n\text{-}2$ is $O(n)$

need $c > 0$ and $n_0 \geq 1$ such that $7n\text{-}2 \leq c.n$ for $n \geq n_0$

this is true for $c = 7$ and $n_0 = 1$

$f(n) = 3n^3 + 20n^2 + 5$

$3n^3 + 20n^2 + 5$ is $O(n^3)$

need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c.n^3$ for $n \geq n_0$

this is true for $c = 4$ and $n_0 = 21$

$f(n) = 3 \log n + 5$

$3 \log n + 5$ is $O(\log n)$

need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + 5 \leq c.\log n$ for $n \geq n_0$

this is true for $c = 8$ and $n_0 = 2$

$f(n) = 2^{n+2}$

$f(n) = 2n^3 + 10n$ ➡ $O(n^3)$

$(2n^3 + 10n) \leq c.n^3$ ✓

For $c = 12$, and $n_0 = 2$

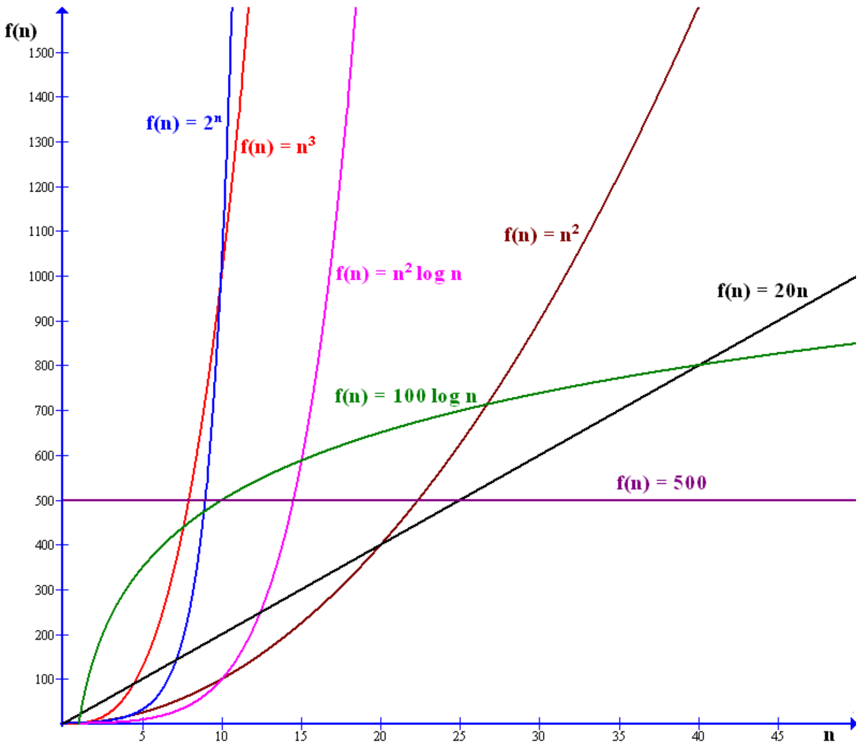
LET US TRY FINDING BIG O…

Assume that you lost your wedding ring on the beach, and have no memory of when it came off. Thus, you decide to do a brute force grid search with your metal detector, where you divide the beach into strips, and walk down every strip, scanning the whole beach, until you find it. For simplicity, assume the beach is a square of side length '*l*' meters, each of your strips has a constant width of 1 meter, and it takes 10 seconds to walk 1 meter (it's hard to walk while searching). Find the big-oh performance of your ring finding algorithm.

Source: https://brilliant.org/

BIG-O AND GROWTH RATE

What is growth rate of an algorithm?



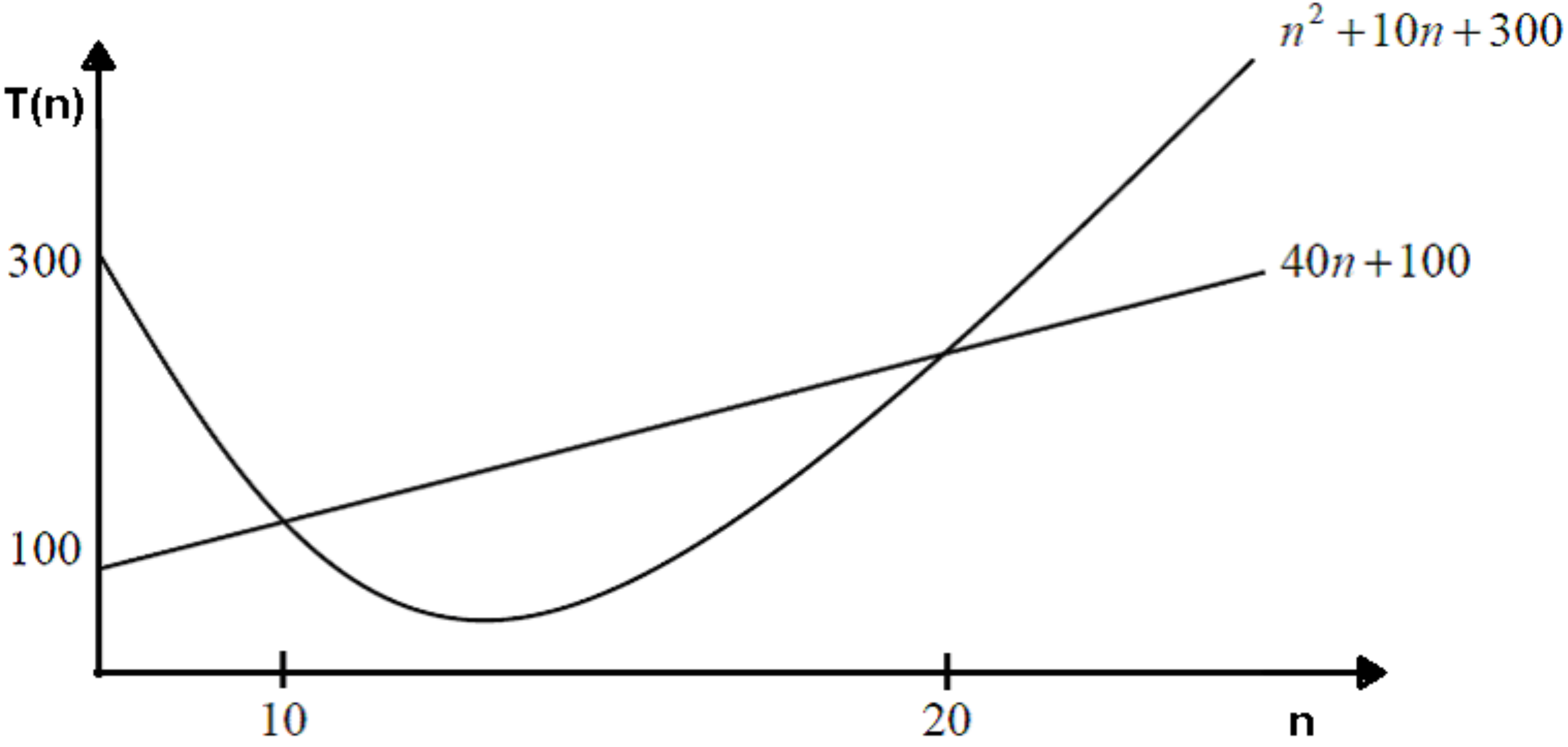
Increasing complexity

Increasing problem size

n	$\log_2 n$	$n^{0.5}$	$n \log_2 n$	n^2	n^3	2^n
2	1	1.41	2	4	8	4
4	2	2	8	16	64	16
8	3	2.83	24	64	512	256
16	4	4	64	256	4,096	65,536
32	5	5.66	160	1,024	32,768	4,294,967,296
64	6	8	384	4,094	262,144	$1.84 * 10^{19}$
128	7	11.31	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	16	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	22.63	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	32	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$
2048	11	45.25	22,528	4,194,304	8,589,934,592	$3.23*10^{616}$

Source: The Internet

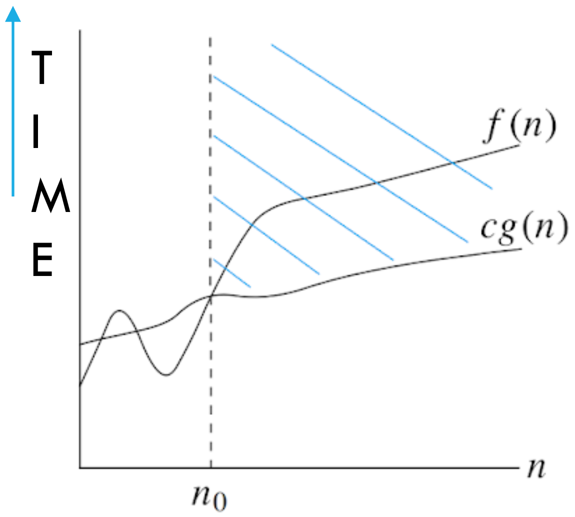
GROWTH RATE CONTINUED...



RELATIVES OF BIG-OH (Ω AND Θ)

Big-Omega Notation (Ω)

- Just like Bio-O provides asymptotic upper-bound, Big-Ω provides asymptotic lower-bound on the running time.
- f(n) is Ω(g(n)) if there exists a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c.g(n)$ for all $n \geq n_0$



f(n) = Ω(g(n))

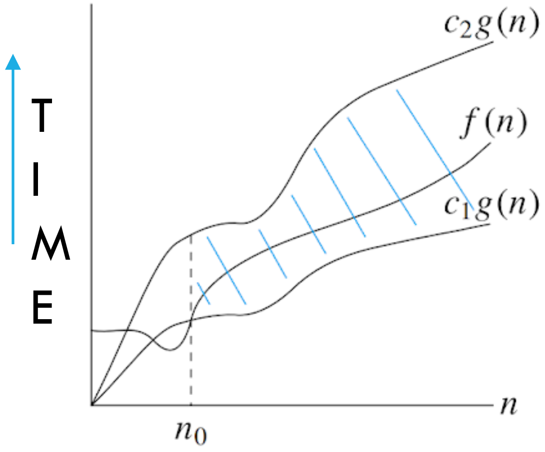
Let, f(n) = 3n.logn + 2n Justification: 3n.logn+2n $\geq$ 3n.logn, for $n \geq 2$

Ω(nlog n)

Big-Theta Notation (Θ)

f(n) is Θ(g(n)) , if: *f*(*n*) is <u>both</u> O(*g*(*n*)) and Ω(*g*(*n*))

f(n) is Θ(g(n)) if there are constants $c_1 > 0$ and $c_2 > 0$ and an integer constant $n_0 \geq 1$ such that $c_1.g(n) \leq f(n) \leq c_2.g(n)$ for $n \geq n_0$



f(n) = Θ(g(n))

3nlogn+4n+5logn is Θ(nlog n) 3nlogn $\leq$ 3nlogn+4n+5logn $\leq$ (3+4+5) nlogn for $n \geq 2$

EXAMPLES CONTINUED...

```c
#include <stdio.h>
// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] == x)
            return i;
    }
    return -1;
}
```

```c
/* Driver's code*/
int main()
{
    int arr[] = { 1, 10, 30, 15 };
    int x = 30;
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function call
    printf("%d is present at index %d", x,
           search(arr, n, x));

    return 0;
}
```

What are the best case, worst case, and average case complexities of the above code?

ASYMPTOTIC ANALYSIS

What is the need of Asymptotic Analysis?

In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size using Big-O notation(Worst-case no. of primitive operations).

Ex: Searching in a sorted array using Linear search (on fast computer A) and Binary search (on slow computer B).

n	Running time on A	Running time on B
10	2 sec	~ 1 hr
100	20 sec	~ 1.8 hrs
10^6	~ 55.5 hrs	~ 5.5 hrs
10^9	~ 6.3 yrs	~ 8.3 hrs

(with linear search running time on A=0.2*n)

(with binary search running time on machine B = 1000 * log(n))

Source: https://www.geeksforgeeks.org/

PREFIX AVERAGE EXAMPLE

- A[i] = (X[0] + X[1] + … + X[i])/(i+1)

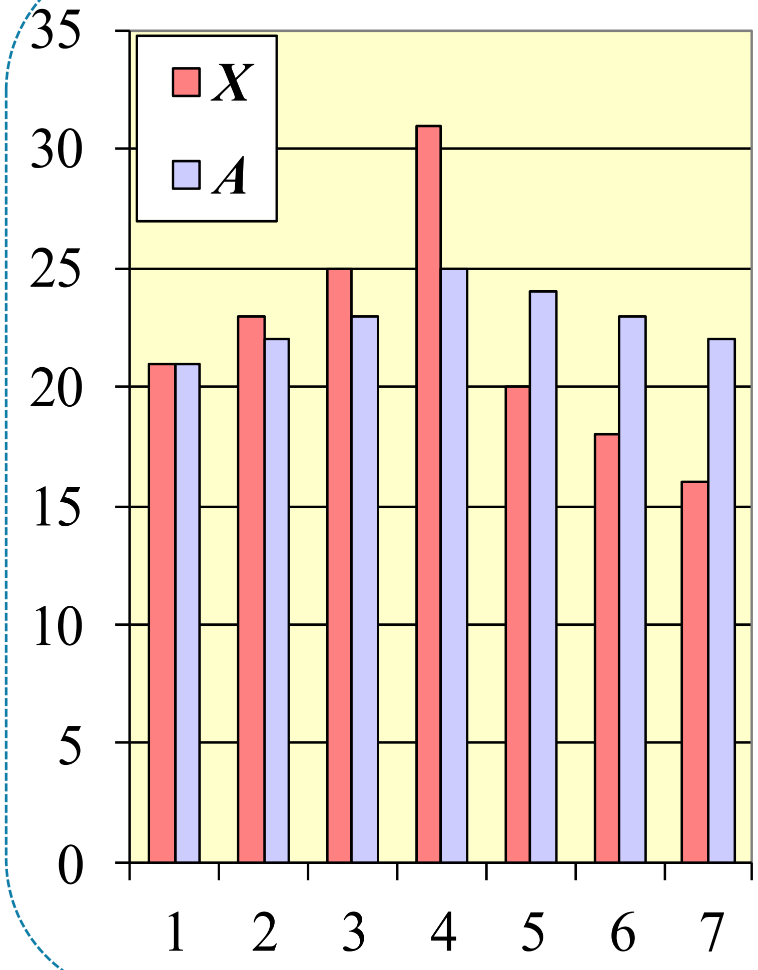
Applications: Eco (Mutual fund averages)

Algorithm prefixAverages1(*X*, *n*)
Input array *X* of *n* integers
Output array *A* of prefix averages of *X*

$A \leftarrow$ new array of n integers	n
for $i \leftarrow 0$ **to** $n - 1$ **do**	n
$s \leftarrow 0$	n
for $j \leftarrow 0$ **to** i **do**	1 + 2 + …+ n
$s \leftarrow s + X[j]$	1 + 2 + …+ n
$A[i] \leftarrow s / (i + 1)$	n
return A	1

Algorithm *prefixAverages2*(*X*, *n*)
Input array *X* of *n* integers
Output array *A* of prefix averages of *X*

$A \leftarrow$ new array of n integers	n
$s \leftarrow 0$	1
for $i \leftarrow 0$ **to** $n - 1$ **do**	n
$s \leftarrow s + X[i]$	n
$A[i] \leftarrow s / (i + 1)$	n
return A	1



TRY YOURSELF…

```
int i = 1, j;
while(i <= n) {
   j = 1;
   while(j <= n)
   {
      statements of O(1)
      j = j*2;
   }
   i = i+1;
}
```

O(nlogn)

```
int sum = 0;

for(int i = 1; i <= n; i++)

   for(int j = i; j < 0; j++)

         sum += i * j ;
```

O(n)

```
for (int j = 0; j < n * n; j++)
   sum = sum + j;

for (int k = 0; k < n; k++)
   sum = sum - l;

print("sum is now ” + sum);
```

$O(n^2)$