

DATA STRUCTURES & ALGORITHMS

Complexity Analysis

Instructor: Engr. Laraib Siddiqui

Big Oh Notation

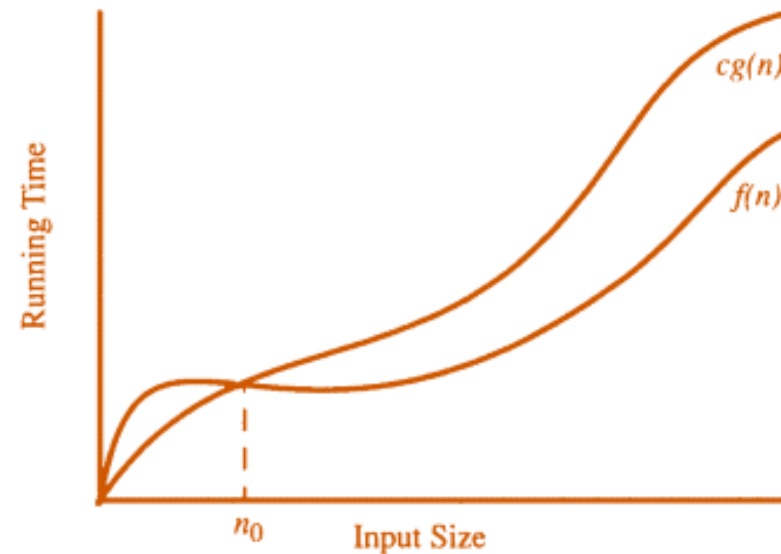
Simplified analysis of an algorithm's efficiency.

- O is called Landaus's symbol, comes from the inventors name Edmund Landau.
- The letter O is used because the rate of growth of a function is also called its order
- Used in complexity theory, computer science and mathematics to describe the behavior of functions.
- It determines how fast a function grows or declines.



Big Oh Notation

- Let $f(n)$ and $g(n)$ be functions mapping non-negative numbers to non-negative numbers.
- Big-Oh. $f(n)$ is $O(g(n))$ if there is a constant $c > 0$ and a constant $n_0 \geq 1$ such that $f(n) \leq c \cdot g(n)$ for every number $n \geq n_0$.



Example

The function $f(n) = 3 \cdot n + 17$ is $O(n)$(Here $g(n) = n$.)

Proof.

Take $c = 4$ and $n_0 = 17$. Then $f(n) = 3n + 17 \leq c \cdot g(n)$ for every $n \geq n_0$. because $3 \cdot n + 17 \leq 4 \cdot n = 3 \cdot n + n$ for every $n \geq 17$.

Big Oh

BETTER

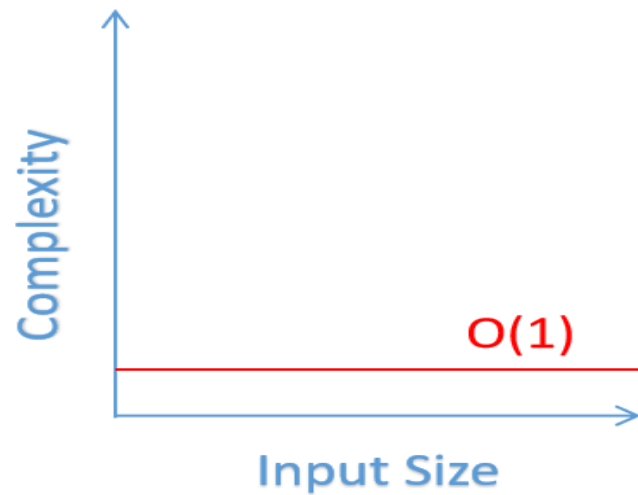


WORSE

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

Constant Time: $O(1)$

Run in constant time if it requires the same amount of time regardless of the input size.



Example: accessing any element in array

Linear Time: $O(n)$

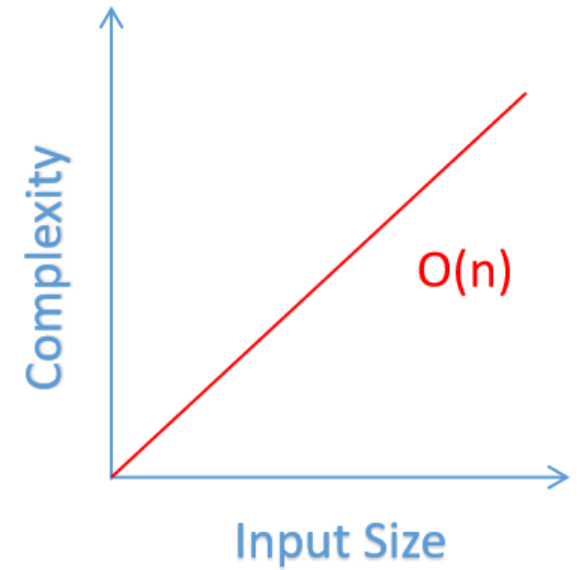
n = number of items

What will be
the worst
case???



best

average



Worst case
need ' n ' steps
for ' n ' items

Example: traversing an array

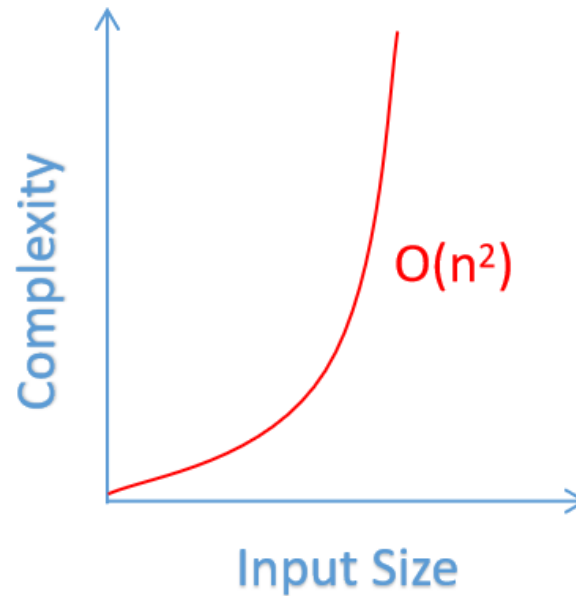
Quadratic Time: $O(n^2)$

- n = number of items

What will be the worst case???

Worst case

need ' $n*n$ ' steps for desired output



Example: bubble sort, selection sort, insertion sort

Logarithmic Time: $O(\log n)$

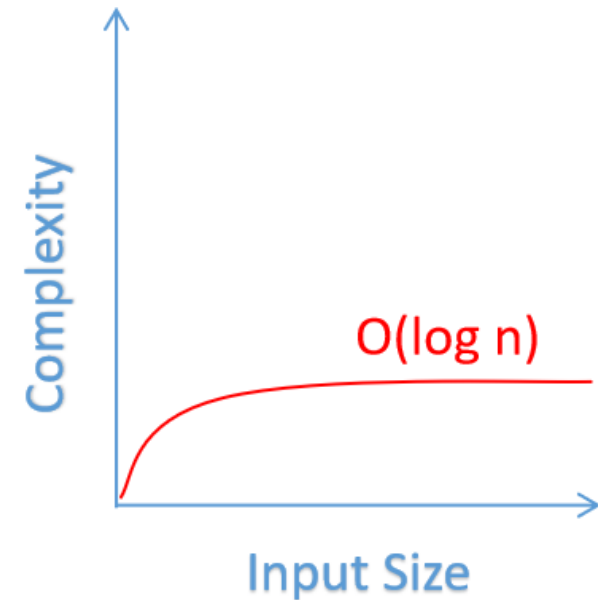
Sorted
order

services
plus') *n.* [< OFr *sur-*, above (see *SUR-1*)
more] a quantity over and above what is
ed — *adj.* forming a surplus
se (sər priz') *vt.* -*prised'*, -*pris'ing* [< OFr
suddenly or unexpectedly; take unawares 1 to come upon
without warning 3 to amaze; astonish — *n.* 1 a
being surprised 2 something that surprises
sur-re'al (sər rē'əl, sə-; -rēl') *adj.* 1 surrealist 2
bizarre; fantastic
sur-re'al-ism' (-iz'əm) *n.* [see *SUR-1* & *REAL*] a mod-
ern movement in the arts trying to depict the work-
ings of the unconscious mind — *sur-re'al-is'tic adj.*
— *sur-re'al-ist adj., n.*
sur-ren-der (sə ren'dər) *vt.* [< Fr *sur-*, up + *rendre*,
render] 1 to give up possession of; yield to another
on compulsion 2 to give up or abandon oneself
oneself up, esp. as a prisoner
dering

$\log 10 = ?$

$\log 20 = ?$


$\log 100 = ?$



Example: binary search

$O(n \log n)$

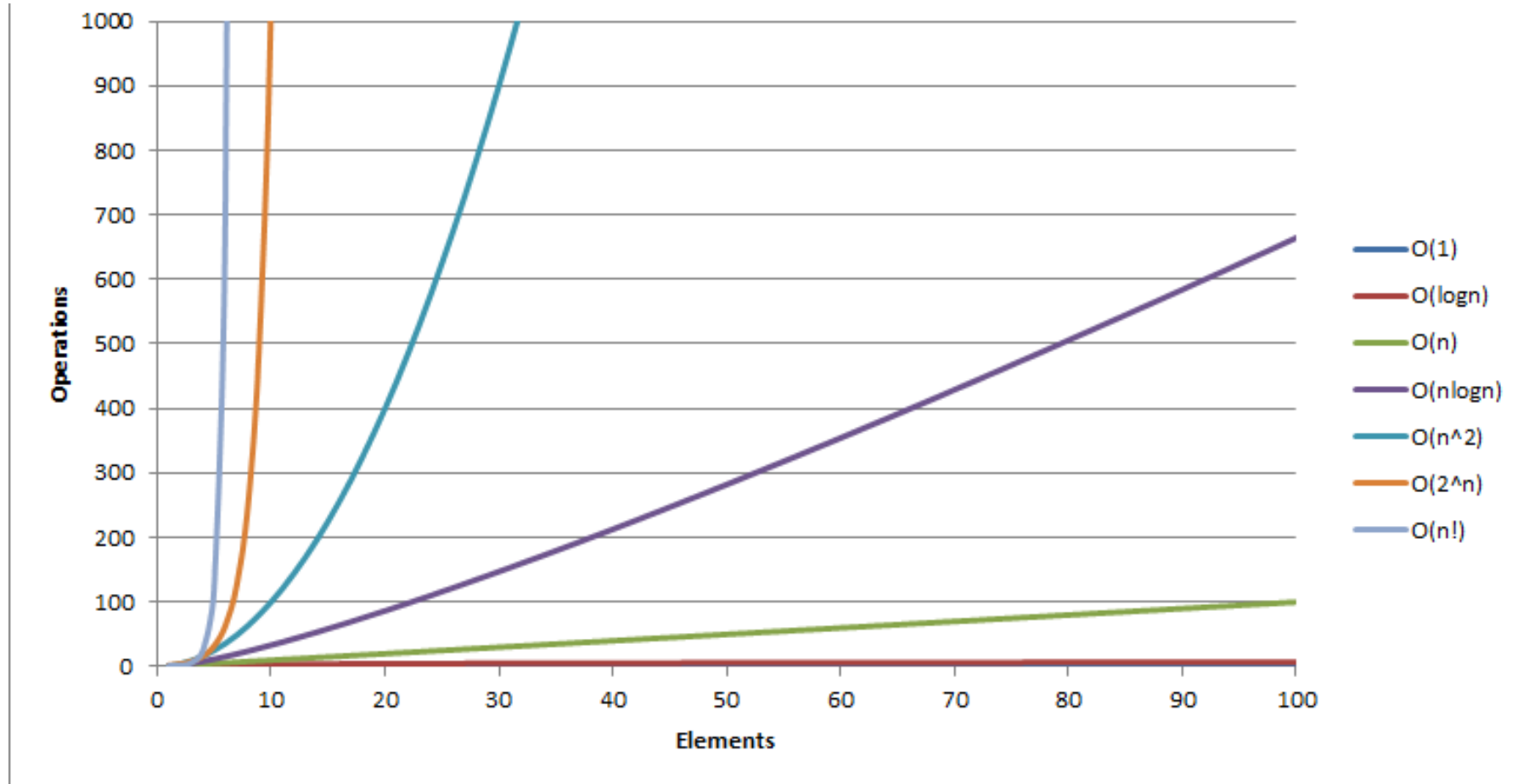
- Growth rate is faster as compared to linear and log functions



Consider buying
pair for your shirts
from store but not
going through every
item

Example: merge sort

Complexity graph



Rules for analysis

- Ignore constants

$6n \rightarrow O(N)$ ← 'n' gets larger & constant no longer matter

- Certain terms dominate others

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$ ← Ignore lower order terms

Rules for analysis

Loops

Number of iterations

Nested loops

Complexity of inner loop * outer loop

Consecutive statements

Addition

If/else

Block which take long time

Switch case

Block which take long time

Example

`x = 5 + (15 * 30);` `// O(1)`
(independent of input size)

`x = 5 + (15 * 30);` `// O(1)`

`y = 6 - 4;` `// O(1)`

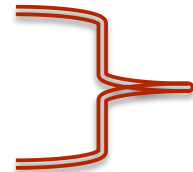
`print x + y;` `// O(1)`

Total Time = $O(1) + O(1) + O(1) = > O(1)$
(drop constant)

Example

```
for (int i = 0; i < n; i ++)
```

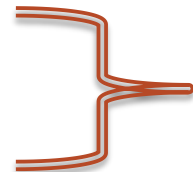
```
    sum = sum + i;
```



$O(n)$

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

```
    sum = sum + i;
```



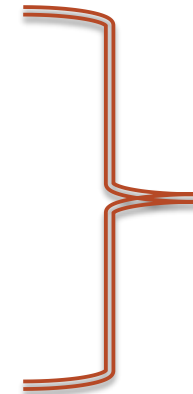
$O(n^2)$

```
sum = 0
```

```
for (int i = 0; i < n; i ++)
```

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

```
        sum += i * j;
```



$O(n^2)$

Example

```
int sum( int n )  
{  
    int partialSum;  
  
    partialSum = 0;  
    for( int i = 1; i <= n; ++i )  
        partialSum += i * i * i;  
    return partialSum;  
}
```



Complexity??