COMPLEXITY ANALYSIS

ANALYSIS OF ALGORITHMS

- Two ways to measure complexity:
  - Space Complexity – How much space does an Algorithm occupy in memory
  - Time Complexity – How much time does an Algorithm take to execute

FERMI-IZE ALGORITHM COMPARISON

Broader question

Which is the most effective solution algorithm from a given set of solutions?

What questions does it raise?

How is the data arranged?

How does the Hardware affect the run time of the algorithms?

How does the input size affect the size of the algorithms?

Where to find the data?

Full implementation of the algorithm

Run on various hardware configurations

Run on various input sizes and types

EMPIRICAL ANALYIS

- Experiment by running programs using various inputs and calculating elapsed time, it's a reasonable measure of how efficient the algorithm is.

```
long startTime = System.currentTimeMillis();
/* Run the algorithm */
long end Time = System.currentTimeMillis();
long elapsedTime = endTime – startTime;
```

This is not a reliable way to measure algorithm efficiency.

CHALLENGES

Experimental Method

- Direct comparisons of running times of algorithms is difficult.
  - Dependent on Hardware and Software conditions
  - CPU must not run any other programs.
- Run it on different inputs
  - Limited number of runs / input combinations
- Record exact running times
  - Not feasible for an algorithm that takes a long time to execute
- Fully code and implement an algorithm
  - Developing fully working systems in early stages is a waste of resources.

BEYOND EXPERIMENTAL ANALYSIS

- Needs to be independent of hardware and software
- Needs only a high level description of the algorithm instead of complete implementation
- All possible input are taken in consideration

Developing an effective strategy by :

- Count primitive Operations (memory access, mathematical operations, comparisons, method calls and returns, assignment operations)
- Measure operations in terms of input size
- Focus on the Worst-Case Input – if an algorithm is efficient with the worst possible input , it is effective. Designing with worst case inputs leads to better algorithms.

RAM MODEL OF COMPUTATION

Assumptions to make before calculating complexity of Algorithms:

1. We have Infinite Memory
2. Each primitive operation (Mathematical, Comparison, etc.) takes one unit of time.
3. Each memory access (Assignment) takes one unit of time.
4. Data may be accessed from the RAM or the Disk, we assume that all data access is done from RAM.

ASYMPTOTIC ANALYSIS

Calculating the time the algorithm takes to execute based on the size of the input is called asymptotic analysis.

Size of input (n) -> Running time of algorithm

Small size of n -> Less running time

Bigger size of n -> More running time

CHANGE IN RUNNING TIME

- Input size is increased -> Running time also increases
- Input size is doubled -> Running time may double, triple, quadruple, increase 100 times even

- Comparing algorithms based on how they perform when the input size increases can help us judge which algorithm is better.

Input Size	2	4	10	100	1000	10000
Algorithm A	2	4	20 2n	200	2000	20 000
Algorithm B	2	4	200	10 000	1 000 000	100 000 000

c 2 4 30 300 3000 30000

3n

By observing how the running time increases based on increase in input size, we can determine which is a more efficient algorithm.

- We calculate the rate of growth of algorithms, how the running grows with increasing input size.
- We use BIG O notation to find the rate of growth of algorithms

BIG O NOTATION

- The most used notation for complexity Analysis.
- Big O notation categorizes functions based on their rate of growth.
- Different functions that have the same rate of growth can be categorized under the same Big O category.
- Growth rate of a function is called the Order of growth, hence Big O.

BIG O – FORMAL DEFINITION

Consider a function $f(n)$, n being the input size.

Asymptotic behavior of $f(n)$ is how fast $f(n)$ grows as n become large.

$f(n)$ is $O(g(n))$ if there exists constant c and n_0 , such that

$f(n) <= c\, g(n)$ for all $n >= n_0$

Consider $g(n) = 5n+4$ and $f(n) = n$

There should be some constant c and n_0, such that

$5n + 4 <= c\,n$ for all $n >= n_0$

When $c = 6$ and $n_0 = 4$, $5n+4 < = 6n$ for all $n >= 4$

We can say that $g(n)=5n+4$ belongs to $O(f(n))$, where $f(n) = n$.

In other words, $g(n)$ is $O(n)$

- The combination of c and n0 may be infinite, we only have to prove one instance of c and n_0 which holds true.
- There is no necessity to find the first ever or the best combination of c and n_0.

COMMON ASYMPTOTIC FUNCTIONS

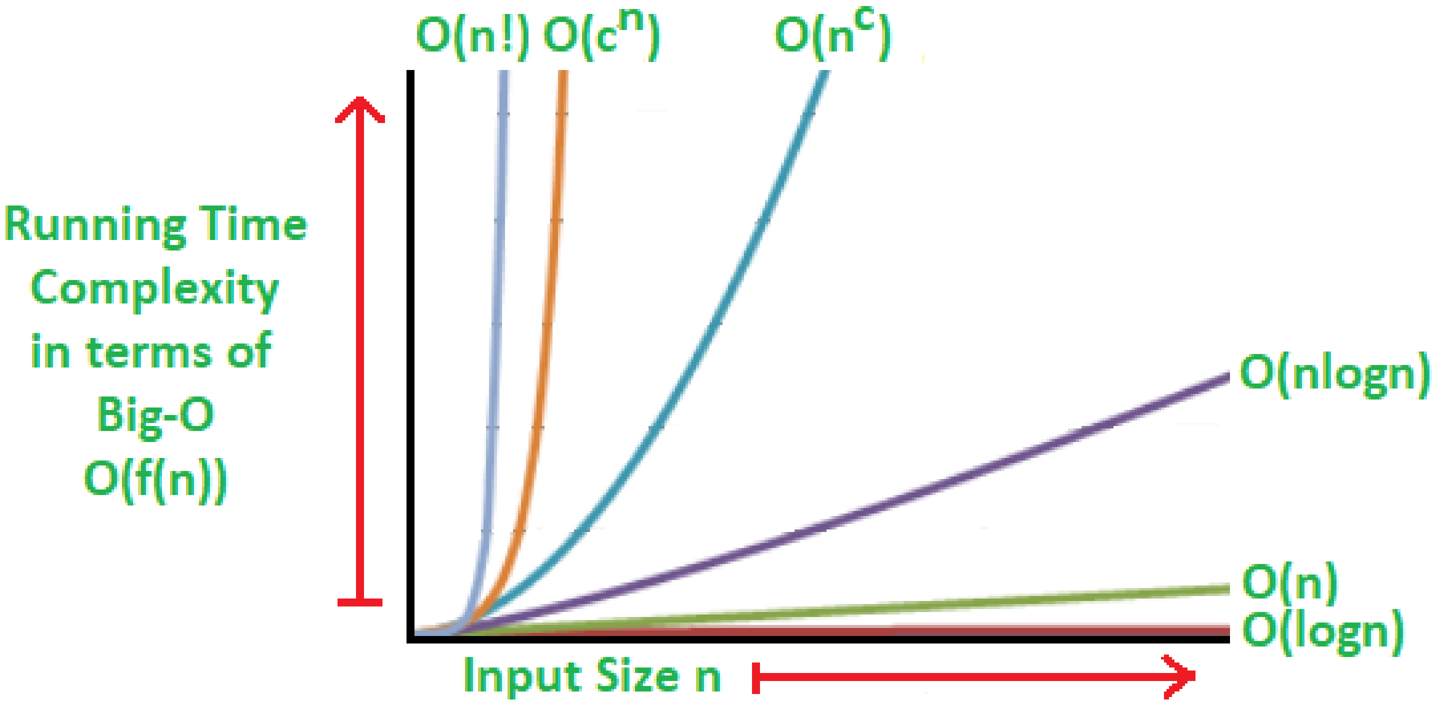
F(n) is only represented by some simple functional forms.

These forms contain a single term in n with a co-efficient of 1.

- $F(n) = c$, constant rate of growth
- $F(n) = \log_b n, b>1$, Logarithmic functions
- $F(n)= n$, Linear Functions
- $F(n)= n \log n$
- $F(n) = n^2$, Quadratic Function – occurs commonly with nested loops
- $F(n) = 2^n$, Exponential Function, loop operations double every iteration
- $F(n) = n!$, Factorial

COMPARING GROWTH RATES

Value of n	log n	n	nlogn	n^2	n^3	2^n
1	0	1	0	1	1	2
2	1	2	2	4	8	4
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4096	65536
32	5	32	160	1024	32768	4.29E+09
64	6	64	384	4096	262144	1.84E+19



$O(n!), O(c^n), O(n^c)$ - Worst

$O(n\log n)$ - Bad

$O(n)$ - Fair

$O(\log n)$ - Good

$O(1)$ - Best

SLOWEST TO FASTEST GROWING FUNCTIONS

Slowest growing function

Fastest growing function

1
$\log n$
n
$n \log n$
n^2
n^3
$\ldots$
n^k
2^n
3^n
$\ldots$
k^n
$n!$

RULES OF FINDING BIG O

Given any function F(n) , the following are the rules to finding Big O:

1. Always consider the fastest growing term of n
2. Ignore all coefficients
3. If f(n) is a constant , according to rule 2 f(n) belongs to o(1)
4. Base of the log is not important

LOOSE & TIGHT UPPERBOUNDS

$5n+4$ can also belong to $O(n^2)$, this is called the loose upper bound. All loose upper bounds are correct.

$f(n)$ must belong $O(n^m)$ where m is the highest power of n in the function, this is called a tight upper bound.

$g(n)$ is the smallest possible function to satisfy the Big O definition.