Algorithmic Efficiency

Big-O Notation

What it is ?

Simplified Analysis of an Algorithm’s Efficiency

1. Complexity in terms of input size, N
2. Machine Configuration
3. Logics used in algorithm
4. Time and Space (Memory)

Types of Measurement……..

Worst - Case

Average - Case

Best - Case

General Rules ......

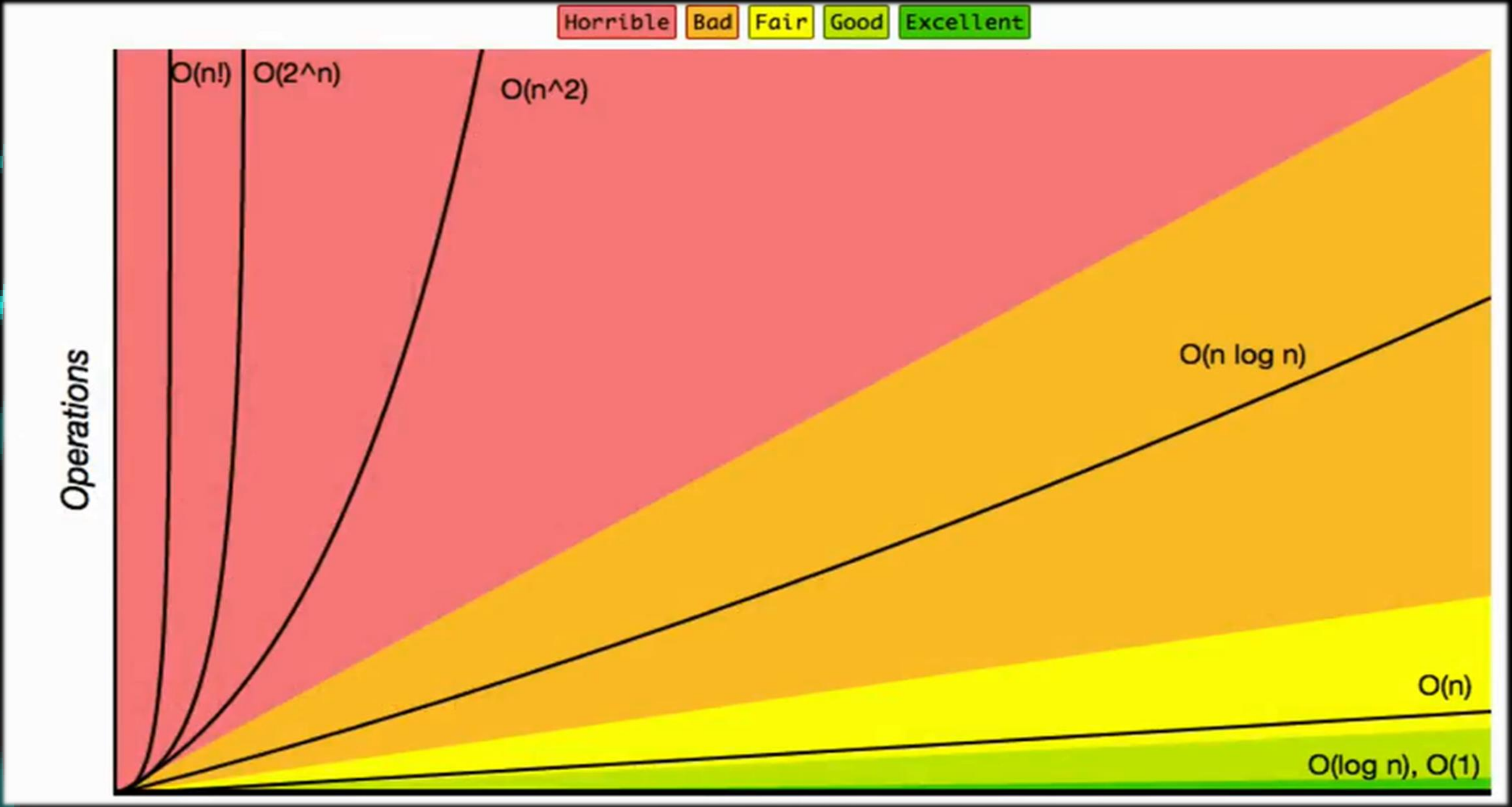
1. Ignore Constant

$$2(n) \rightarrow O(n)$$

2. Ignore Non- dominant terms

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$$

Big-O Notation Complexity Chart



Constant Time: Complexity

X = 2 + (10 * 25)

Independent of Input Size (N).

Constant Values may be ignore and Hence Final time taken will be O(1)

Constant Time: Complexity

X = 2 + (10 * 25) => O(1)
Y = 15 – 3 => O(1)
Print (X+ Y) => O(1)

Total Time taken:
O(1) + O(1) + O(1) = 3* O(1)
Determine the Bigger (Dominant) term that is
O(1)
Result = O(1)

Linear time: Complexity of loop

```
for k in range ( 0,n):
    print (k)
```

for k in range (0,n): → Loop executes "N" times

print (k) → O(1)

Total Time taken:

N * O(1) = O(N)

N is bigger (dominant) term than O(1)

So,

Result = O(N)

Other Example of Loop

```
y = 5 + (15 * 20);                 O(1)
for x in range (0, n):           } O(N)
     print x;
```

total time = O(1) + O(N) = O(N)

Total Time taken:

O(1) + O(N) = O(N)

for loop term O(N) always dominant other terms

So,

Result = O(N)

Quadratic time: Nested loop

```
for X in range (n):          Outer Loop Runs - N Times
    for Y in range(n):       Inner Loop Runs - N Times
        print(X * Y )        O(1) times
```

Total Time taken:

$N * N * O(1) = O(N^2)$

Nested loop N^2 is bigger (dominant) than O(1)

So,

Result = $O(N^2)$

$O(N^2)$

```
x = 5 + (15 * 20);                      O(1)
for x in range (0, n):
    print x;                            O(N)
for x in range (0, n):
    for y in range (0, n):              O(N²)
        print x * y;
```

Total Time taken:

$O(N^2) + O(N) + O(1) = O(N^2)$

Nested loop $O(N^2)$ is bigger (dominant) than $O(N)$ and $O(1)$ So,

Result = $O(N^2)$

Calculate Complexity of if-else

To computer the complexity of if-else statement, we consider the worst case running time. Which mean we consider the total time as given below

Time taken by test + time taken by either of block or else part. (Whichever is larger)

Example:

```
if (A>B):                   # takes time as constant Co
    return False            # takes time as constant C1
else:
    for I in range(n):      # for loop runs n times
        if(A<B):            # takes time as constant C2
            return False    # takes time as constant C3
```

Time taken by else part = (C2+ C3) * n

Total time Taken: C0 + C1 + (C2 + C3) * n that is O(N)

loop N is bigger (dominant) term than other terms in this example.

So, Result will be: O(N) (ignore the Constants)