# Algorithm Run time

It is study of given algorithm running time, by identifying the behaviour as input size of algorithm process.

Easy term- Time taken for algorithm to run.

Why to learn: To measure the efficiency of your program. If not known, you will come across different issue on how to code.

## Notation of Algorithm:

How it is measure comparing to Mileage of bike?

Types of possibility due to driving condition.

In City traffic

In Highways

In normal condition

## How much time will be taken to copy a file from your PC to your friend’s pen drive?

## How to deal through a problem from time taken point of view?

# Types of Annotation:

Omega- Lower bound time. Never less than the given time.

Big(O)- upper bound. Always greater than given time.

Theta: Decides upper or lower bound. Average of given time.

Example:

Find 2 in an array

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 8 | 7 | 5 | 0 | 4 | 3 | 2 |

Omega – T(1); Big O – T(N); Theta – N/2

This proves that Big O gives us the worst case time.

Big O time is the language and metric we use to describe the efficiency of algorithms. Not the only metric.

## Space Complexity:

Along with the time, space is one of most important parameter. If more space is consumed then many processor may be affected.

**Usage**:

M = n+4

Print(m)

**Better usage**:

n+=4;

Print(n)

# Time complexities-

Some examples and graphs:

1 – Insert an element in an array

logn – Finding an element in a sorted array

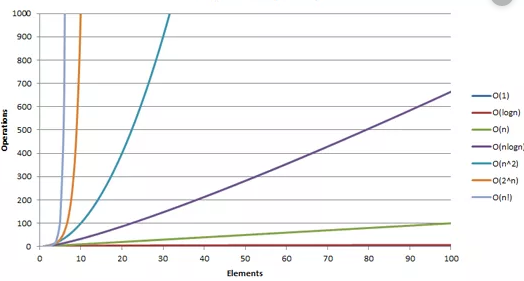
n – Finding an element in unsorted array

nlogn – Merge Sort

n^2 – Shortest path between 2 nodes of a graph

2^n Tower of Hanoi / Fibonacci using recursion.

N! – Recursive function inside a loop



# Example and explanation with a normal program:

import array

a= array.array("i", [1,2,3,4])

temp =a[0]

for i in a:

if (temp<a[i]):

temp =a[i]

print ("Smallest value of the array is", temp)

### Function Initiation

Insert a value to a variable -----------------------🡪 O(1)

Loop from start to end ---------------------------🡪 O(n)

Comparison ----------------------------------------🡪 O(1)

Swap ------------------------------------------------🡪 O(1)

Return ---------------------------------------------🡪 O(1)

Time value is O(n) + O(1) ~ O(n)

# Example and explanation with a Recursive program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 8 | 6 |

STEP1: 5

STEP2: 5

STEP3: 8

STEP4: 8

STEP5: 8

BiggesNumber(A, n) ------------------------------------------🡪

Initiate a first value to new variable ------------------------------------------🡪 O(1)

If n < 1 ------------------------------------------🡪 O(1)

Return ------------------------------------------🡪 O(1)

Else ------------------------------------------🡪 O(1)

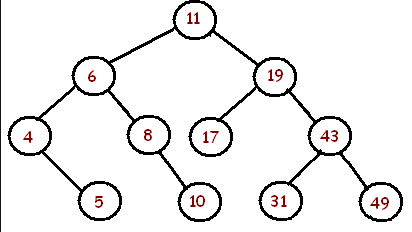
If a[n] > highest ------------------------------------------🡪 O(1)

Swap ------------------------------------------🡪 O(1)

Return BiggesNumber(A, n-1) ------------------------------------------🡪 O(N-1)

Time ~ O(N)

# Example and explanation with a Recursive program #2:



Search (root, Value)

If (root equal null)

Return null

Else if (root equal to Value )

Return root.

Else if (value < root)

Search (root.left, value)

Else

Search (root.right, value)

Checking the node – O(1)

Recursion – O(Log n)

|  |  |
| --- | --- |
| ***n*** | **log2 n** |
| 1 | 0 |
| 2 | 1 |
| 4 | 2 |
| 8 | 3 |
| 16 | 4 |
| 32 | 5 |