How to Calculate 'Algorithm Time Complexity' ?

- ✓ Iterative Algorithm
- ✓ Recursive Algorithm

Example#1: Time Complexity of 'Iterative Algo'


```
FindBiggestNumber (int arr[ ]):
    biggestNumber = arr[0]
    loop: i = 1 to length(arr)-1
        if arr[ i ] > biggestNumber
            biggestNumber = arr[ i ]
    return biggestNumber
```

Example#1: Time Complexity of 'Iterative Algo'(continued)

```
FindBiggestNumber (int arr[]):
    biggestNumber = arr[0] ---------- O(1)
    loop: i = 1 to length(arr)-1 ---------- O(n)   } -------- O(n)
        if arr[ i ] > biggestNumber ------- O(1)   } ------- O(1)
            biggestNumber = arr[ i ] ------- O(1)
    return biggestNumber ---------- O(1)
```

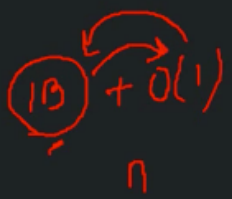
O(2)

O(n-1) = O(n)

O(2×n) = O(n)

O(10) = O(1)

O(1000) = O(1)



Time Complexity = O(1) + O(n) + O(1)

= O(n) + O(2)

Example#2: Time Complexity of 'Recursive Algo'


```
FindBiggestNumber(A, n):
    static highest = IntegerMin
    if  n equals -1
      return highest
    else
        if A[n] > highest
          update highest
      return FindBiggestNumber(A,  n-1)
```

Example#2: Time Complexity of 'Recursive Algo' (Continued)

```
FindBiggestNumber(A,n): ---------- T(n)
    static highest = Integer.Min---------- O(1)
    if (n equals -1) ---------- O(1)
        return highest ---------- O(1)
    else ---------- O(1)
        if A[n] > highest ---------- O(1)
            update highest ---------- O(1)
    return FindBiggestNumber(A, n-1) ---------- T(n-1)
```

Back Substitution:

T(n) = O(1) + T(n-1) ---------- Equation#1

T(-1) = O(1) ---------- Base Condition

T(n-1) = O(1) + T((n-1)-1) ---------- Equation#2

T(n-2) = O(1) + T((n-2)-1) ---------- Equation#3

Example#2: Time Complexity of 'Recursive Algo' (continued)

Back Substitution:

$T(n) = O(1) + T(n-1)$ ---------- Equation#1

$T(-1) = O(1)$ ---------- Base Condition

$T(n-1) = O(1) + T((n-1)-1)$ ---------- Equation#2

$T(n-2) = O(1) + T((n-2)-1)$ ---------- Equation#3

$T(n) = 1 + T(n-1)$

$= 1 + (1 + T((n-1)-1))$

$= 2 + T(n-2)$

$= 2 + 1 + T((n-2)-1)$

$= 3 + T(n-3)$

$= k + T(n-k)$

Example#2: Time Complexity of 'Recursive Algo' (continued)

Back Substitution:

$T(n) = O(1) + T(n-1)$ ---------- Equation#1

$T(-1) = O(1)$ ---------- Base Condition

$T(n-1) = O(1) + T((n-1)-1)$ ---------- Equation#2

$T(n-2) = O(1) + T((n-2)-1)$ ---------- Equation#3

$T(n) = 1 + T(n-1)$

$= 1 + (1 + T((n-1)-1))$

$= 2 + T(n-2)$

$= 2 + 1 + T((n-2)-1)$

$= 3 + T(n-3)$

$= k + T(n-k)$

$= (n+1) + T(n-(n+1))$

$= n+1 + T(-1)$

$= n + 1 + 1$

$= O(n)$

Example#3: Time Complexity of 'Recursive Algo'

✓**Problem Statement:** *Given a sorted array of 11 numbers, find number 110*

10	20	30	40	50	60	70	80	90	100	110

✓**Solution:**

Example#3: Time Complexity of 'Recursive Algo' (continued)

```
BinarySearch(int findNumber, int arr[], start, end):  ---------- T(n)
    if (start equals end) ---------- O(1)
        if (arr[start] equals findNumber) ---------- O(1)
            return start ---------- O(1)
        else return error message that number does not exists in the array ---------- O(1)
    mid = FindMid(arr[], start, end) ---------- O(1)
    if mid > findNumber ---------- O(1)
        BinarySearch(int findNumber, int arr[], start, mid) ---------- T(n/2)
    else if  mid < findNumber ---------- O(1)
        BinarySearch(int findNumber, int arr[], mid, end ---------- T(n/2)
    else if mid = findNumber ---------- O(1)
        return mid ---------- O(1)
```

Time Complexity:

$T(n) = O(1) + T(n/2)$

Example#3: Time Complexity of 'Recursive Algo' (continued):

Back Substitution:

$T(n)=T(n/2)+1$ ---------- Equation#1

$T(1) = 1$ ---------- Base Condition

$T(n/2)= T(n/4) + 1$ ---------- Equation#2

$T(n/4) = T(n/8) + 1$ ---------- Equation#3

$T(n) = T(n/2^1)+1$

$= (T(n/4) + 1)+ 1$

$= T(n/4) + 2$

$= (T(n/8) + 1) + 2$

$= T(n/8) + 3$

$= T(n/2^k) + k$

$= T(1) + \log n$

$= 1 + \log n$

$= \log n$

T(1) = 1, Hence
so the for what value of K ->$T(n/2^k)$
becomes T(1)

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log n$