Algorithms

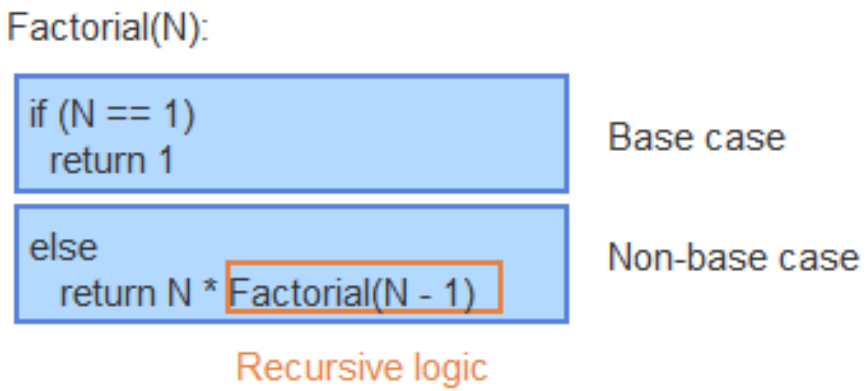
COMP 53 - DATA STRUCTURE

UNIVERSITY OF THE PACIFIC

Recursive Definition

- A recursive algorithm is an algorithm that breaks the problem into **smaller sub-problems** and applies the algorithm itself to solve the smaller sub-problems.

- **base case**: A case where a recursive algorithm completes without applying itself to a smaller sub-problem.
- A **recursive function** is a function that **calls itself**.



Cumulative Summation

- Summation of all the numbers from 0 to N

```
CumulativeSum(N) {
    if (N == 0)
        return 0
    else
        return N + CumulativeSum(N - 1)
}
```

Reverse a List

```
ReverseList(list, startIndex, endIndex) {
    if (startIndex >= endIndex)
        return
    else {
        Swap elements at startIndex and endIndex
        ReverseList(list, startIndex + 1, endIndex - 1)
    }
}
```

Fibonacci Sequence

- a numerical sequence where each term is the sum of the previous 2 terms in the sequence
- except the first 2 terms, which are 0 and 1

```
FibonacciNumber(termIndex) {
   if (termIndex == 0)
      return 0
   else if (termIndex == 1)
      return 1
   else
      return FibonacciNumber(termIndex - 1) + FibonacciNumber(termIndex - 2)
}
```

Runtime Complexity of Recursive functions

- The runtime complexity T(N) of a recursive function will have function T on both sides of the equation

- The runtime complexity T(N) will be equal to
  - the runtime of all of the statements in the function
  - plus the runtime of the function called inside with the new input size

- Such a function is known as a **recurrence relation**
  - A function T(N) that is defined in terms of T on a value < N.

Runtime Complexity of Recursive functions

- Using O-notation to express runtime complexity of a recursive function requires solving the recurrence relation.
- For simpler recursive functions, it can be determined by expressing the **number of function calls**
- Other useful tool for solving recurrences is a **recursion tree**, a visual diagram of a operations done by a recursive function

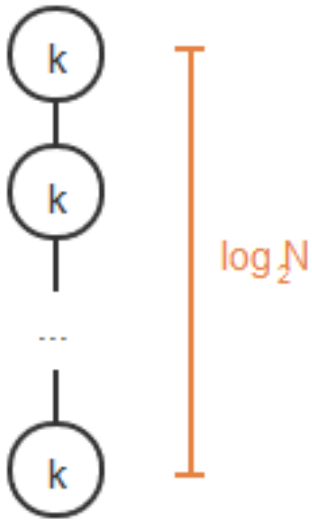
Runtime Complexity of Recursive functions

```
int logCalculator(int n) {
    if (n == 1) {
        return 0;
    }
    return 1 + logCalculator(n / 2);
}
```

Input Size	Number of function calls
1	1
2	2
4	3
8	4
16	5
32	6
...	...
n	$\log n + 1$

```
If: T(N) = O(1) + T(N / 2)
Then: T(N) = O(log N)
```

$T(N) = k + T(N / 2)$



$T(N) = k \times \log N$

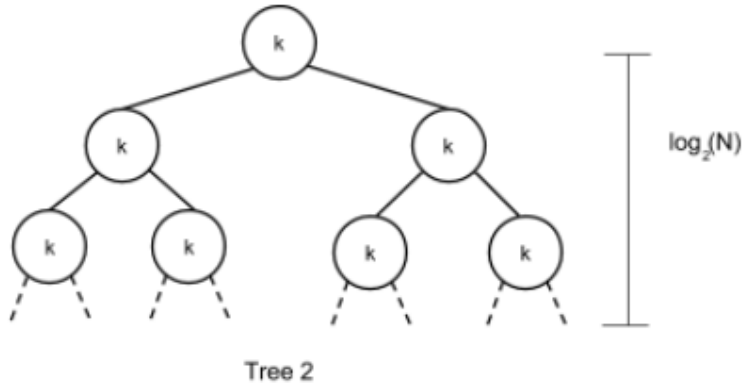
$T(N) = O(\log N)$

Runtime Complexity of Recursive functions

```java
int cumulativeSum(int a[], int start, int end, int size) {
    if (size == 1) {
        return a[start];
    }
    return cumulativeSum(a, start, end/2, size/2 ) + cumulativeSum(a, end/2 +1, end, size - size/2);
}
```

Input Size	Number of function calls
1	1
2	3
4	7
8	15
16	31
32	63
…	…
n	$2n+1$

```
  If: T(N) = O(1) + T(N / 2)+ T(N / 2)
Then: T(N) = O(N)
```



Tree 2

$$T(N) = k \times 1 + k \times 2 + k \times 4 + \cdots + k \times N$$
$$T(N) = k \times (1 + 2 + 4 + \cdots + N)$$
$$T(N) = k \times \left(2^0 + 2^1 + 2^2 + \cdots + 2^{\log_2 N}\right)$$
$$T(N) = k \times \left(2^{\log_2 N + 1}\right)$$
$$T(N) = k \times (N + 1)$$
$$T(N) = O(N)$$