# Guide to Calculating Big O Complexity

## Constant Time: O(1)

The operation's execution time does not depend on the input size.

Formula: O(1)

Example:

function getFirstElement(arr) {  
 return arr[0]; // Always one operation, regardless of input size  
}

Big O: O(1)

## Linear Time: O(n)

The execution time grows linearly with the input size.

Formula: O(Number of Iterations)

Example:

function printElements(arr) {  
 for (let i = 0; i < arr.length; i++) {  
 console.log(arr[i]);  
 }  
}

Big O: O(n)

## Quadratic Time: O(n^2)

Occurs with nested loops where the inner loop depends on the outer loop.

Formula: O(Outer Loop Iterations × Inner Loop Iterations)

Example:

function printPairs(arr) {  
 for (let i = 0; i < arr.length; i++) {  
 for (let j = 0; j < arr.length; j++) {  
 console.log(arr[i], arr[j]);  
 }  
 }  
}

Big O: O(n^2)

## Logarithmic Time: O(log n)

The input size is reduced by half at each step.

Formula: O(log n)

Example:

let i = n;  
while (i > 1) {  
 console.log(i);  
 i = Math.floor(i / 2);  
}

Big O: O(log n)

## Linearithmic Time: O(n log n)

Common in sorting algorithms where the problem is divided, solved, and merged.

Formula: T(n) = a × T(n/b) + O(n^d)

Example:

function mergeSort(arr) {  
 if (arr.length <= 1) return arr;  
 const mid = Math.floor(arr.length / 2);  
 const left = mergeSort(arr.slice(0, mid));  
 const right = mergeSort(arr.slice(mid));  
 return merge(left, right);  
}

Big O: O(n log n)

## Exponential Time: O(2^n)

Occurs when the solution space grows exponentially with the input size.

Formula: O(2^n)

Example:

function fibonacci(n) {  
 if (n <= 1) return n;  
 return fibonacci(n - 1) + fibonacci(n - 2);  
}

Big O: O(2^n)

## Factorial Time: O(n!)

Appears in problems where all possible permutations are computed.

Formula: O(n!)

Example:

function generatePermutations(arr, n = arr.length) {  
 if (n === 1) {  
 console.log(arr);  
 return;  
 }  
 for (let i = 0; i < n; i++) {  
 generatePermutations(arr, n - 1);  
 const swapIdx = n % 2 === 0 ? i : 0;  
 [arr[swapIdx], arr[n - 1]] = [arr[n - 1], arr[swapIdx]];  
 }  
}

Big O: O(n!)

## Summary Table of Big O Complexity

|  |  |  |
| --- | --- | --- |
| Complexity | Formula | Example Use Case |
| Constant | O(1) | Accessing an element |
| Linear | O(n) | Iterating through a list |
| Quadratic | O(n^2) | Nested loops |
| Logarithmic | O(log n) | Binary search |
| Linearithmic | O(n log n) | Merge sort |
| Exponential | O(2^n) | Recursive combinatorics |
| Factorial | O(n!) | Permutations problem |