Asymptotic notations are used to compare algorithms by providing a mathematical boundary for their runtime functions. These notations define an upper bound, a lower bound, or both, helping to analyze the efficiency of algorithms as input size grows.

Big-O: upper bound on runtime growth denoted $O(f(n))$ which is

$$g(n) : 0 < g(n) \leq cf(n) \text{ whenever } n \geq n_0 \text{ for some } c > 0, n_0 > 0$$

Big-Omega: lower bound on runtime growth denoted $\Omega(f(n))$ which is

$$g(n) : 0 < cf(n) \leq g(n) \text{ whenever } n \geq n_0 \text{ for some } c > 0, n_0 > 0$$

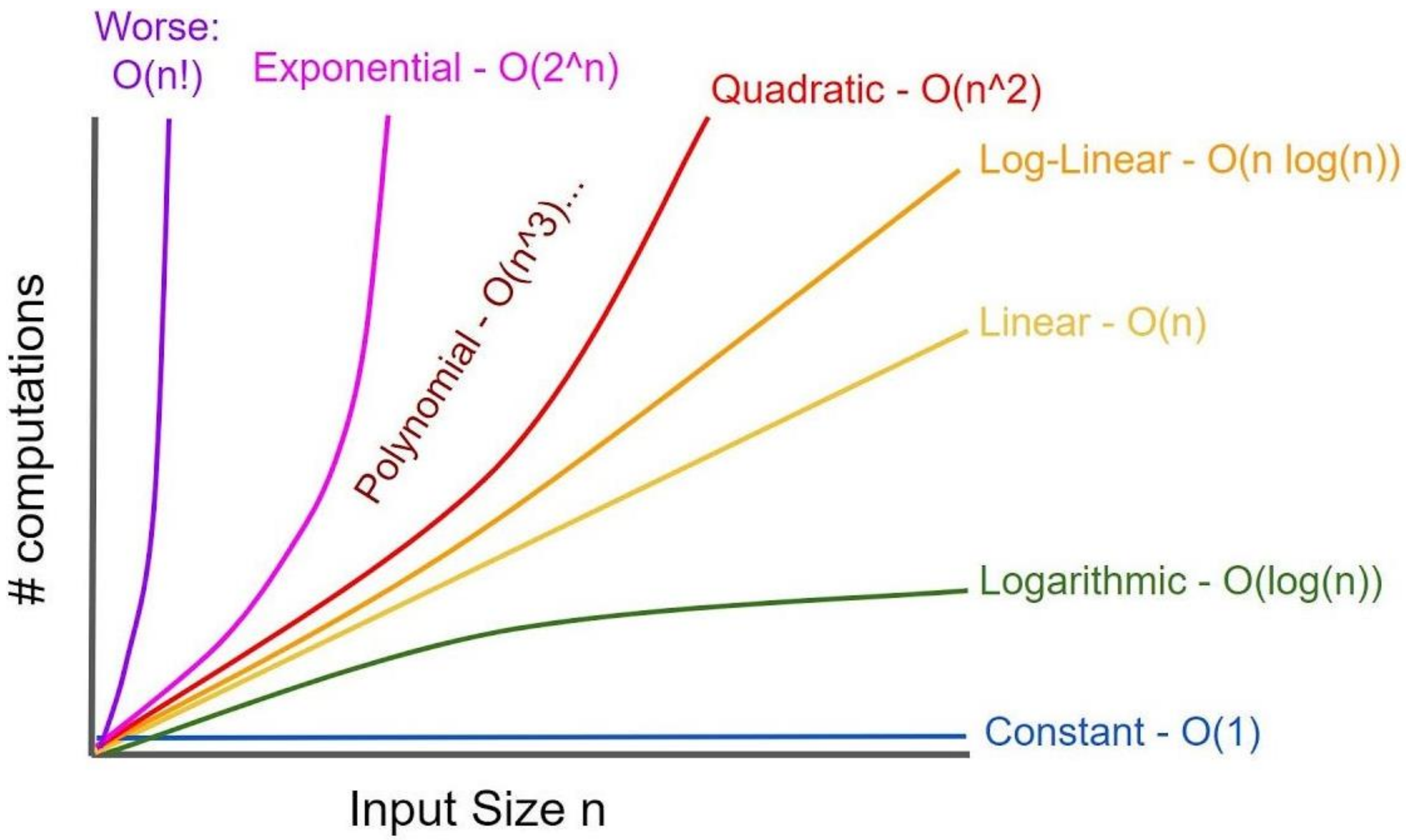
Big-Theta: tight bound (both upper and lower bound) denoted $\Theta(f(n))$ which is

$$g(n) : c_1 f(n) \leq g(n) \leq c_2 f(n) \text{ whenever } n \geq n_0 \text{ for some } c_2 \geq c_1 > 0, n_0 > 0$$

Common Big-O Functions

- **$O(1)$**: constant time (independent of input size).
- **$O(\log_2 n)$**: logarithmic time (e.g., binary search).
- **$O(n)$**: linear time (e.g., iterating over an array).
- **$O(n^2)$**: quadratic time (e.g., nested loops).
- **$O(2^n)$**: exponential time (e.g., brute-force recursive algorithms).

Visualizing Big-O



```cpp
template <typename T>
void Swap(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

Steps	Code	Cost (c_i)	Time (t_i)
1	`T temp = a;`	c_1	1
2	`a = b;`	c_2	1
3	`b = temp;`	c_3	1

$$T(n) = \sum_{i=1}^{m} c_i \cdot t_i$$

$T(n) = c_1(1) + c_2(1) + c_3(1)$

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

$T(n) = 3$

$O(1)$

```
template <typename T>
T Maximum(T arr[], int size)
{
    T max = arr[0];
    int i = 1;
    while (i < size)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
        i++;
    }
    return max;
}
```

Steps	Code	Cost (c_i)	Time (t_i)
1	`T max = arr[0];`	c_1	1
2	`int i = 1;`	c_2	1
3	`while (i < size)`	c_3	n – 1 + 1
4	`if (arr[i] > max)`	c_4	n - 1
5	`max = arr[0];`	c_5	n - 1
6	`i++;`	c_6	n - 1
7	`return max;`	c_7	1

$T(n) = c_1(1) + c_2(1) + c_3(n) + c_4(n - 1) + c_5(n - 1) + c_6(n - 1) + c_7(1)$

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

$T(n) = 4n$

$O(n)$

```
string Square(int length)
{
    string result;
    for (int i = 0; i < length; i++)
    {
        for (int j = 0; j < length; j++)
        {
            result += "*";
        }
        result += "\n";
    }
    return result;
}
```

Steps	Code	Cost (c_i)	Time (t_i)
1	`string result;`	c_1	1
2	`for (int i = 0; i < length; i++)`	c_2	1 + n + n + 1
3	`for (int j = 0; j < length; j++)`	c_3	$n + n^2 + n^2 + n$
4	`result += "*";`	c_4	n^2
5	`result += "\n";`	c_5	n
6	`return result;`	c_6	1

$$T(n) = c_1(1) + c_2(2n + 2) + c_3(2n^2 + 2n) + c_4(n^2) + c_5(n) + c_6(1)$$
$$T(n) = 1 + 2n + 2 + 2n^2 + 2n + n^2 + n + 1$$
$$T(n) = 3n^2 + 5n + 4$$
$$O(n^2)$$