

TIME COMPLEXITY SHORTCUT NOTE (SAKIF RAHAMAN MISHAD)

TIME COMPLEXITY	REASONABLE INPUT SIZE
$O(1)$	$\leq 10^{18}$
$O(\log n)$	$\leq 10^{18}$
$O(\sqrt{n})$	$\leq 10^{16}$
$O(n)$	$\leq 10^8$
$O(n \log n)$	$\leq 10^6$
$O(n \log^2 n)$	$\leq 10^5$
$O(n\sqrt{n})$	$\leq 10^5$
$O(n^2)$	$\leq 10^4$
$O(n^3)$	≤ 500
$O(n^4)$	≤ 100
$O(2^n)$	≤ 25
$O(n!)$	≤ 10

1. **$O(1)$** – Constant Time

Example - `arr[i]`, `a+b`

2. **$O(\log n)$** – Logarithmic Time

Example: Halving the input in each iteration

```
for (i = 1; i < n; i *= 2) { // O(log n)
    // some operation
}
```

3. **$O(\sqrt{n})$** – Square Root Time

Example: Iterating up to the square root of `n`

```
for (i = 1; i * i < n; i++) { // O(√n)
    // some operation
}
```

4. **$O(n)$** – Linear Time

Example: Iterating through an array of size `n`

```
for (i = 0; i < n; i++) // O(n)
```

5. **$O(n \log n)$** – Linearithmic Time

Example: Efficient sorting algorithms (like Merge Sort or Quick Sort)

```
for (i = 0; i < n; i++) { // O(n)
    for (i = 1; i < n; i *= 2) { // O(log n)
        // some operation
    }
}
```

```
sort(arr, arr + n); // O(n log n)
```

6. **$O(n^2)$** – Quadratic Time

Example: Nested loops over the input array

```
for (i = 0; i < n; i++) { // O(n)
    for (j = 0; j < n; j++) { // O(n)
        // some operation
    }
} // Overall O(n^2)
```

7. **$O(n^3)$** – Cubic Time

Example: Triple nested loops

```
for (i = 0; i < n; i++) { // O(n)
    for (j = 0; j < n; j++) { // O(n)
        for (k = 0; k < n; k++) { // O(n)
            // some operation
        }
    }
} // Overall O(n^3)
```

8. **$O(2^n)$** – Exponential Time

Example: Recursive problems with two choices at each step (e.g., solving the subset-sum problem)

```
void recursiveFunction(int n) {
    if (n == 0) return;
    recursiveFunction(n-1); // O(2^n)
    recursiveFunction(n-1);
}
```

9. **$O(n!)$** – Factorial Time

Example: Generating all permutations of an array

```
void generatePermutations(vector<int>& arr, int l, int r) {  
    if (l == r) return;  
    for (int i = l; i <= r; i++) { //  $O(n!)$   
        // permute  
        generatePermutations(arr, l+1, r);  
    }  
}
```

Cheat Sheet for Common Patterns:

1. for (i = 0; i < n; i++) → **$O(n)$**
2. for (i = 1; i < n; i *= 2) → **$O(\log n)$**
3. for (i = 1; i * i < n; i++) → **$O(\sqrt{n})$**
4. Nested loops: for (i = 0; i < n; i++) for (j = 0; j < n; j++) → **$O(n^2)$**
5. Recursion without pruning: $O(2^n)$
6. Permutations or combinations generation → **$O(n!)$**

