Asymptotic Analysis.

↳ Helps us to measure the order of growth.

→ when input size start increasing, how will the time / space requirements increase.

→ Limiting behavior
↳ $n \to \infty$

$T(n) = \underline{C_0 n} + \cancel{C_1}$

$T(n) = C_0 \cdot \underline{n^2} + \cancel{C_1 \underline{n}} + \cancel{C_2}$

$T(n, m) = C_0 m^2 + C_1 n + \cancel{C_2} + \cancel{C_3 m}$

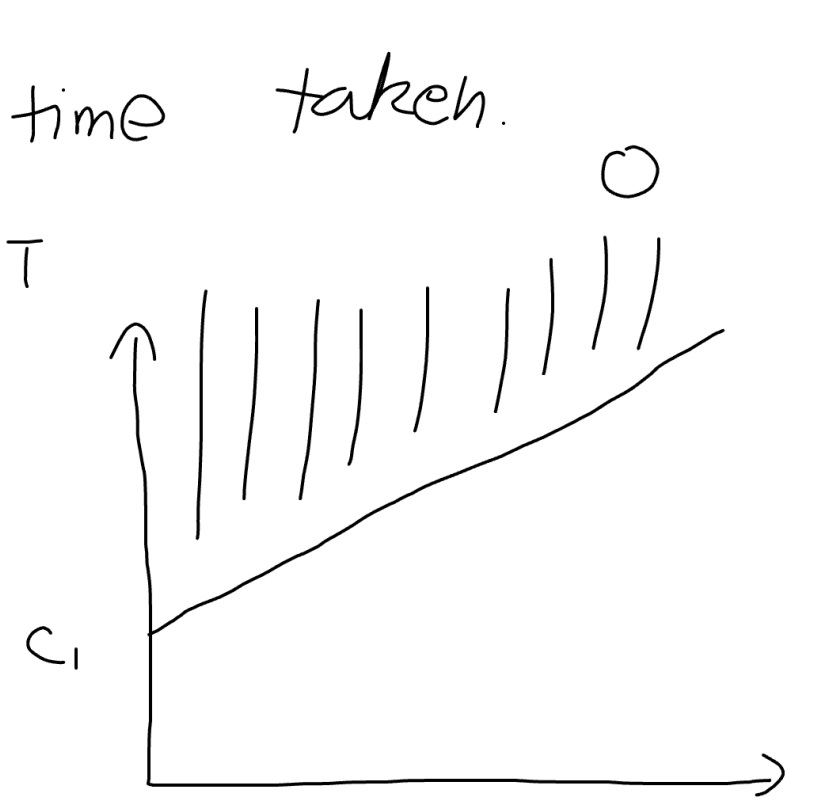
```
For (i =0; i < n; i++) {
        For (int j = 0; j < m; j++) {
                Cout << i + j << endl;
        }
}
```

i = 0
j: 0 → m-1] → O(m)
+
i = 1
j: 0 → m-1] → O(m)
+
⋮
i = n-1
j: 0 → m-1] → O(m)
= $\underline{O(nm)}$.

Big-O.

Upper-Bound on the time taken

$\underline{T(n) = C_0 n + C_1}$

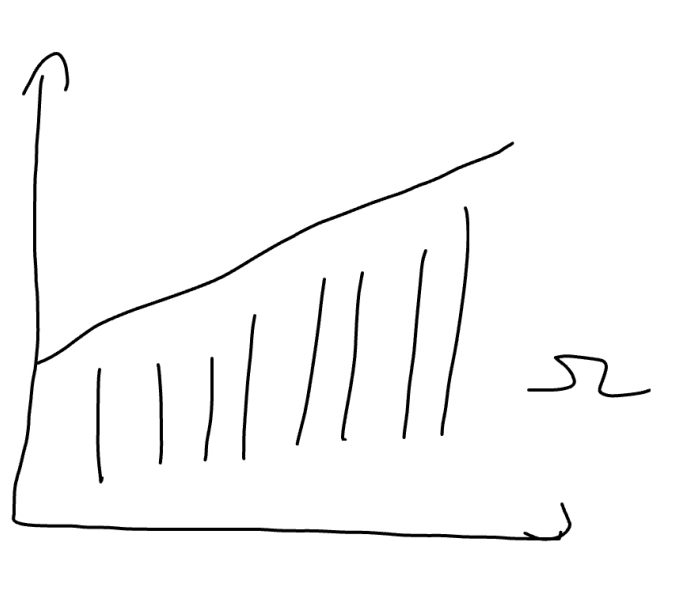


$O(\underline{n}) / O(n \log n) / O(n^2) / O(n!) / \ldots$

Big-Omega

Lower Bound to the time taken.

$T(n) = C_0 n + C_1$

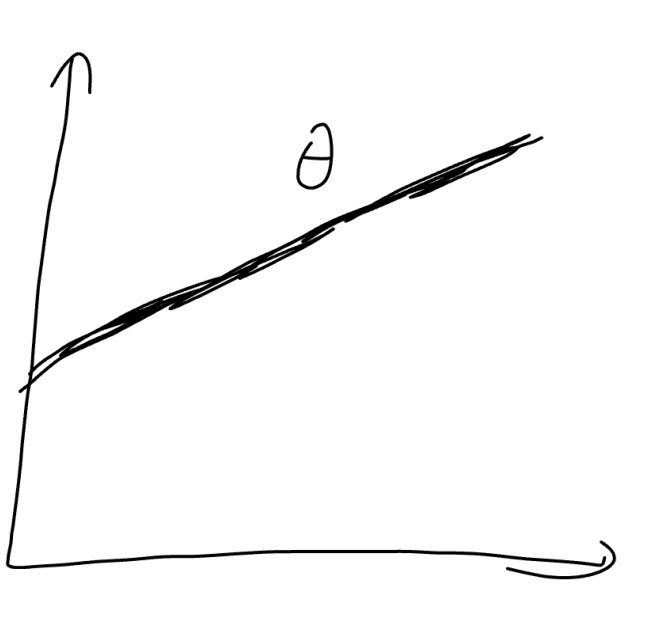


$\Omega(n) / \Omega(\log n) / \Omega(1)$ - -

Big-Θ

Exact Bound to the time taken.

$T(n) = C_0 n + C_1$



$\Theta(n)$.

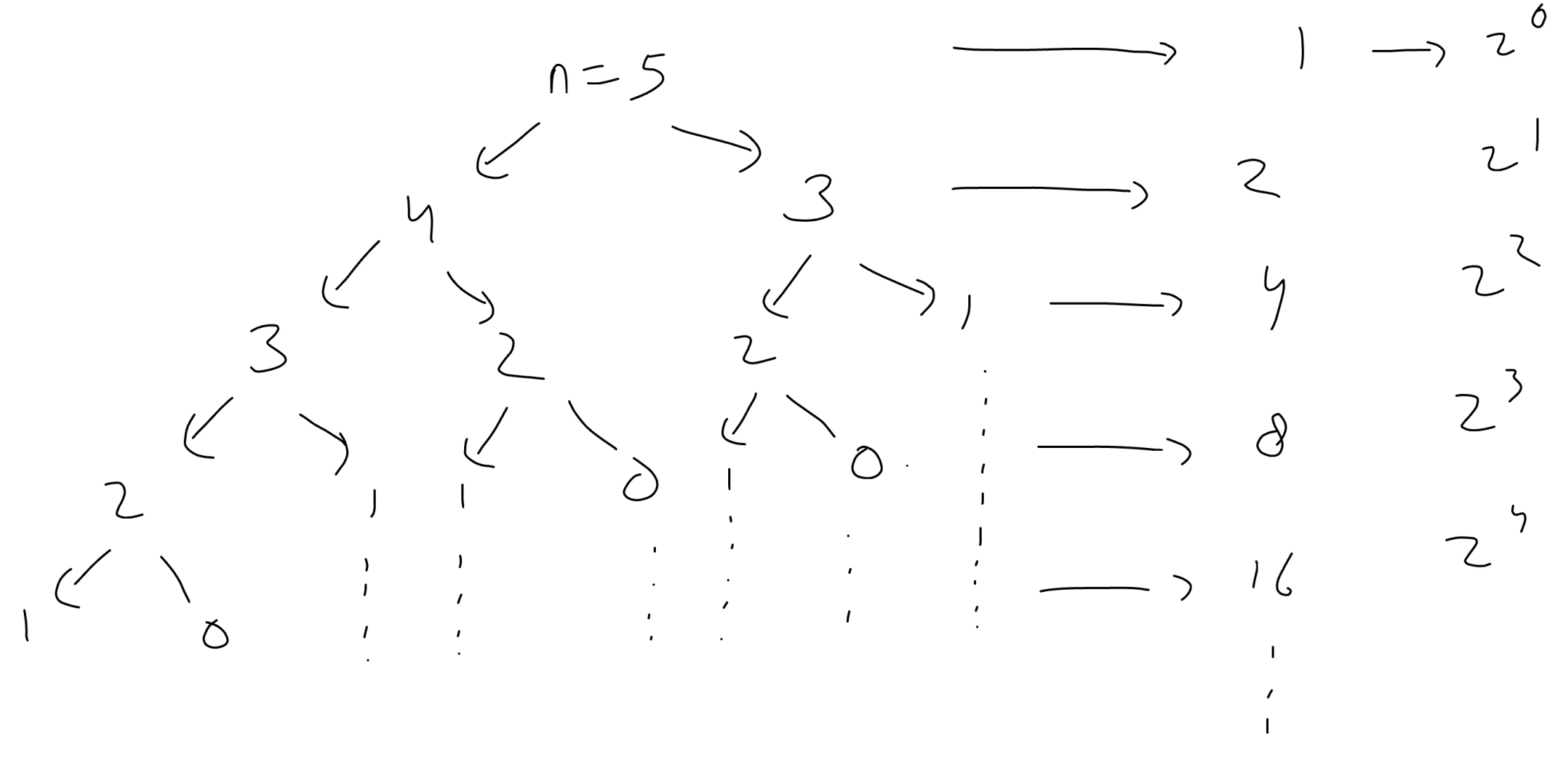
Fib(n) = Fib(n-1) + Fib(n-2)

fib(0) = 0
fib(1) = 1

0, 1, 1, 2, 3, 5, 8, 13, 21 - - .

```
int   fib( int n) {
      if (n <= 1 )
            return n;

      return fib(n-1) + fib(n-2);
}
```



$\underline{\Theta(\underline{1.67}^n)}$ $\underline{O(\underline{2}^n)}$.

$$\underbrace{2^0 + 2^1 + 2^2 + 2^3 + 2^4 - - - - -}_{n\text{-terms}}$$

$$\frac{a(r^n - 1)}{r - 1} \qquad 1 * \frac{(2^n - 1)}{2 - 1} = O(2^n)$$

Best / Worst / Avg. case

```
(arr, tar.)
for ( i=0; i<n; i++)
    if arr[i] == target
        return true.
```

Best-Case: Target at i=0
TC = O(1).

Worst-Case: Not present / Present at i=n-1.
TC = O(n).

Avg. case:

i = 0 → 1
i = 1 → 2
i = 2 → 3
i = n/2 ⋮
⋮
i = n-1 → n

$$\frac{\text{Total work}}{n} = \frac{1 + 2 + 3 + \cdots + n}{n}$$

$$= \frac{\frac{n(n+1)}{2}}{\cancel{n}} = \frac{n+1}{2} = O(n).$$

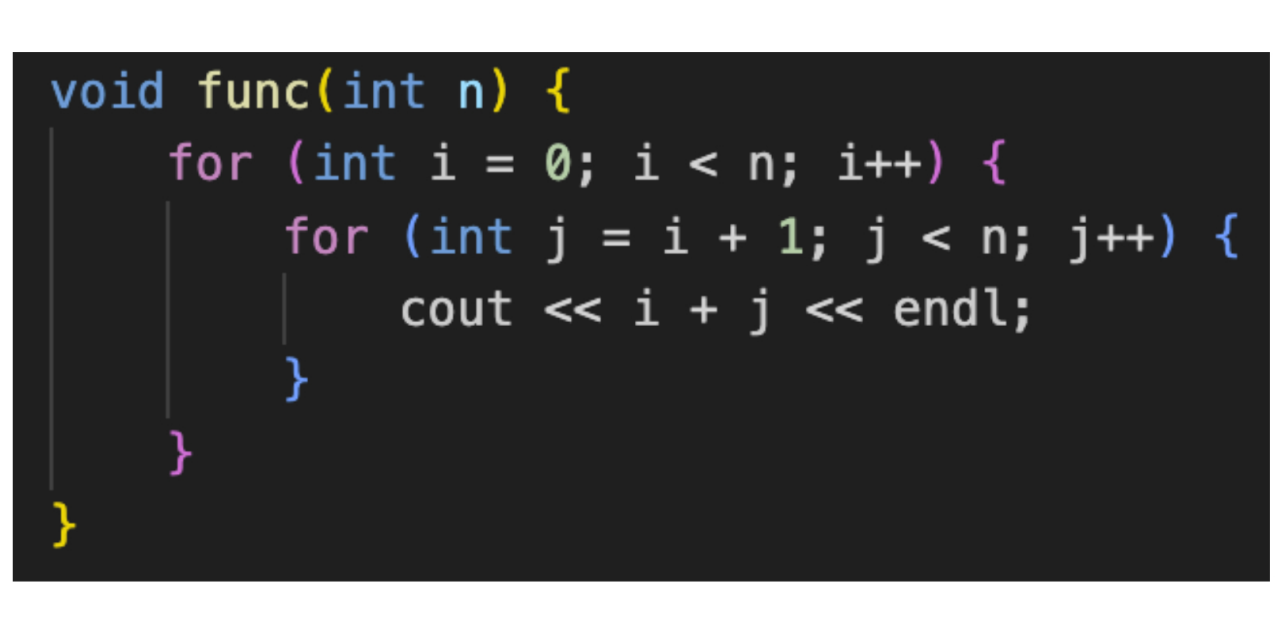
Space Complexity = [Input space + [Aux. space] + Output space.

```
vector<int> fun( vector<vector<int>> arr) {     n²
        vector<int> res (m)
          ⋮
        → vector<int> temp (nm)
          ⋮
        return res
}
```

→ I/P space = $O(n^2)$
→ O/P space = $O(m)$
Aux space = $O(nm)$.

space Complexity = $O(n^2 + \cancel{m} + nm)$.

$\underline{n^2} \cancel{+} \cancel{m} \cancel{+} \cancel{n}$



```
void func(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            cout << i + j << endl;
        }
    }
}
```

i=0
j: 1 → n-1]   n-1
i = 1
j: 2 → n-1]   n - 2
i = 2
j: 3 → n-1]   n-3
⋮
i = n-2
j: n-1 → n-1]   1
i = n-1
j: n → n-1]   0

1+2+3+4 - - - - - + $\underline{n-1}$

$$\frac{(n-1)(n)}{2} = O(n^2).$$

---

int → 32 $\underline{\text{bits}}$

10 → 32 bits.
10^6 → 32 bits.

$10 + 10$ —
$10^6 + 10^6$ —
] O(32) = O(1).

size 10 "abcd ... "
size 10^6 "abcd - - - - - - - - - - - - - "

[char ~ int
string ~ list of ints.]

I/P ["abc" "a" "geeks", "geeks for geeks"]

3 + 1 + 5 + 13 - . . .

$\underline{\text{geeks}}$ > $\underline{\text{geekd}}$

---

abd, dfg, bca, ddb
↳ abd, dfg, abc, bdd
↳ [abc, abd, bdd, dfg]

Step-1: $O(\underline{nm \log m}) / O(nm^2 \log m)$
Step-2: $O(\underline{mn \log n}) / O(m^2 n)$