ASYMPTOTIC ANALYSIS

- Method of defining a limited behaviour.
  ↓
  input tending to infinity.

$T(n) = 3n^2 + \cancel{3n}$

$n = 10^2$

$10^4 \quad 10^2$

$n \to \infty$

$\infty^2 \to \infty$

$T(n) = 3n^2$

$3\infty = \infty$

$n \to \infty$

[$T(n)$ is asymptotically equal to n^2]

There are 2 basic rules for defining the asymptotic value:

1) Drop-off the insignificant terms
2) Drop-off the constants.

$T_1(n) = 5n$ $\qquad$ $T_2(n) = n^2$

$n = 1$

$T_1(1) = 5$ $\qquad$ $T_2(1) = 1$

$n = 2$

$T_1(2) = 10$ $\qquad$ $T_2(2) = 4$

$T_1(3) = 15$ $\qquad$ $T_2(3) = 9$

⋮

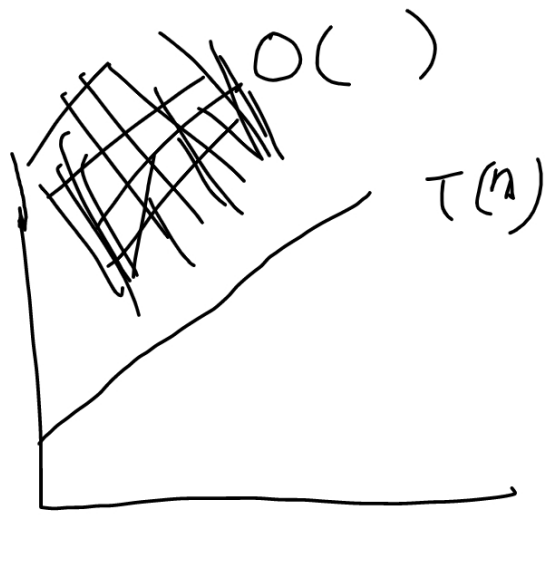
$T_1(10^5) = 5\times10^5$ $\qquad$ $T_2(10^5) = 10^{10}$

[$O(1) < O(\log n) < O(n) < O(n * \log n) < O(n^2) < O(2^n) < O(n!)$]

Big-O
↳ Defines an upper bound on the asymptotic value.

$T(n) = 5n+7$ $\qquad$ $O(n)$ $\quad$ $O(n^2)$

Theoritically, the above algo can be said to be $O(n)$, $O(n \log n)$, $O(n^2)$ ... $O(n!)$



[Big-Theta. Θ ⇒ Provides an exact bound
$T(n) = 5n+7$ ⇒ $\Theta(n)$]

[Practical meaning of Big-O ≡ Theoritical meaning of Big-Theta.]

$T(n) = 5n+7$ $\qquad$ $\boxed{O(n^2)}$ ✗

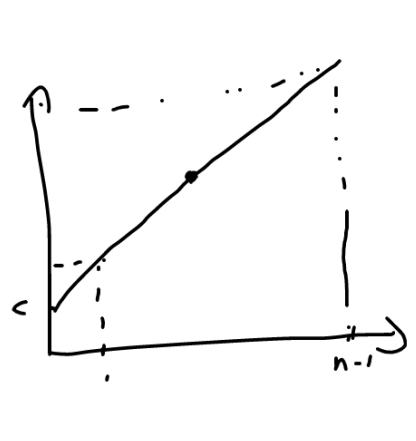
⇓

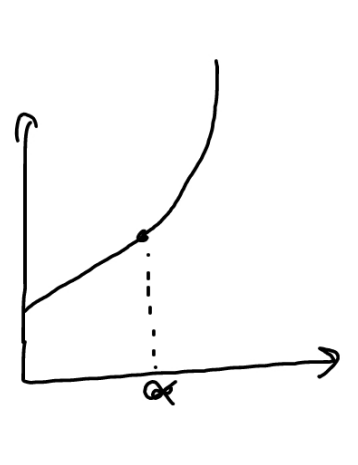
$O(n)$ ✓

$\Theta(n)$ ✓

Best-case, worst-case & Avg.-case time Complexities.

```
int linear_search (int arr[], int n, int k) {
        for (int i=0;  i<n;  i++) {
                if (arr[i] == k) return i;

        }
        return -1;

}
```





Best-case : $O(1)$

Worst-case : $O(n)$

Avg. case :

	computations
k is at 0th idx :	1
1st	2
2nd	3
⋮	
$(n-1)^{th}$	n
k is not present :	n

$(1+2+3 ---- +n) + n$

$= \frac{n(n+1)}{2} + n = \boxed{\frac{n^2+3n}{2}}$

Avg. TC : $O\left(\frac{n^2+3n}{2(n+1)}\right) = O\left(\frac{n^2}{n}\right) = \boxed{O(n)}$ ✓

For a lot of algorithms, avg-case TC comes out to be equal to the worst-case TC.

⇓

But not for all.

[For eg: Quick Sort : Avg. case TC: $O(n \log n)$
worst. case TC: $O(n^2)$]

Auxiliary Space v/s Space Complexity

[Space Complexity = Input Space + Auxiliary Space + Output Space] # IMP

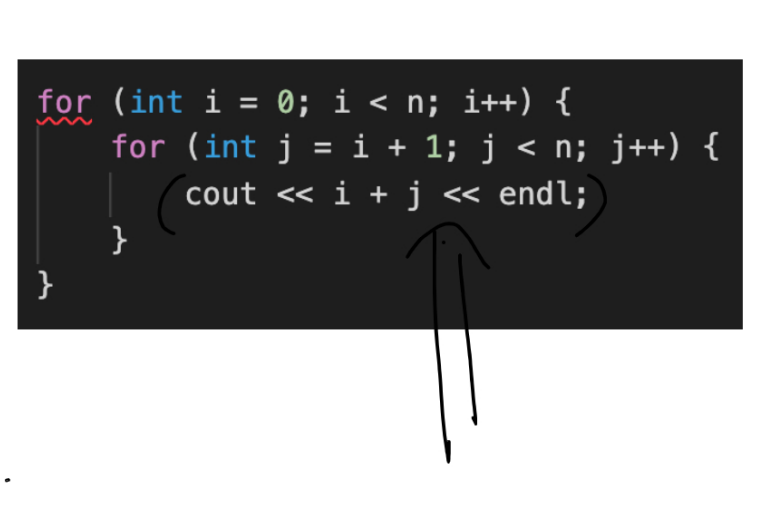
```
vector<int> func (vector<vector<int>> arr) {        Size: n^2
        vector <int> result (m);
   =>   vector <int> temp (m^2);

        - - - - .
        return result;

}
```

Input Space : $O(n^2)$

Output Space : $O(m)$

Auxiliary space : $O(m^2)$

Space Complexity: $O(n^2 + m^2)$



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

⇒ TC of this code = The number of times 'cout' statement gets executed.

K ⇒ $O(k * 1) = O(k)$

$i = 0$ ⇒ Inner loop will be exec. $(n-1)$ times

$i = 1$ ⇒ " " " " " $(n-2)$ "

$i = 2$ ⇒ $(n-3)$

⋮

$i = n-1$ ⇒ 0

No of times cout statement is exec: $0+1+2+ --- +(n-1)$

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

Amt. of time to exec. one cout statement = $O(1)$

⇒ $TC = O(n^2 \times 1) = O(n^2)$

$i = \cancel{0}$ $\qquad$ $j = 1 ---- n-1$

$\cancel{1}$ $\qquad$ $j = 2 ---- n-1$ $\qquad$ $O(2)$

$\cancel{2}$ $\qquad$ $j = 3 ---- n-1$ $\qquad$ $O(1)$

3 $\qquad$ $j = 4 ...$

32-bit integers: $\quad$ 0 ... 31 $\quad$ | 0 | 1 | 0 | 0 | 1 | - - - - - - - | 0 |

$\binom{0}{10^5}$

Printing an integer take O(constant) : $O(1)$ amt of time.

Printing a string of size 'm' ≡ Printing 'm' characters

Printing one character ≡ $O(1)$ time

⇒ Printing a string of size 'm' ≡ $O(m)$

Sorting an array of integers of size n: $O(n \log n)$

[You are given an array of strings. Task is to sort each string individually & then sort the complete array.
size of the array = n
length of the longest string in the array = m]

$[3, 1, 2] \to [1, 2, 3]$

"bac" → "abc"

["bac" → "abc"
"dcbad" → "abcdd"]

[" ~~~ ", " ~~~ ", " ~~~ " - - -]

["abc", "cba", "bac"] → ["abc", "bac", "cba"]

Soln:

→ Time taken to sort one string = $O(m \log m)$

→ Time taken to sort n strings = $O(n * m \log m)$ ↳ ①

Comparing two int takes $O(1)$ time
Now, since the TC to sort 'n' integers is $O(n \log n)$, can I say that the sorting algo is making $O(n \log n)$ no. of comparisons?

→ Time to compare two strings = $O(m)$
No. of comparisons = $O(n \log n)$

⇒ Time to sort an array of strings = $O(m * n \log n)$ ↳ ②

⇒ TC of the overall task =
$O(nm \log m + mn \log n)$

$= \boxed{O(nm(\log m + \log n))}$ ✓

S1 = "ba$\boxed{c}$d"
S2 = "ba$\boxed{d}$c"

(S1 < S2)

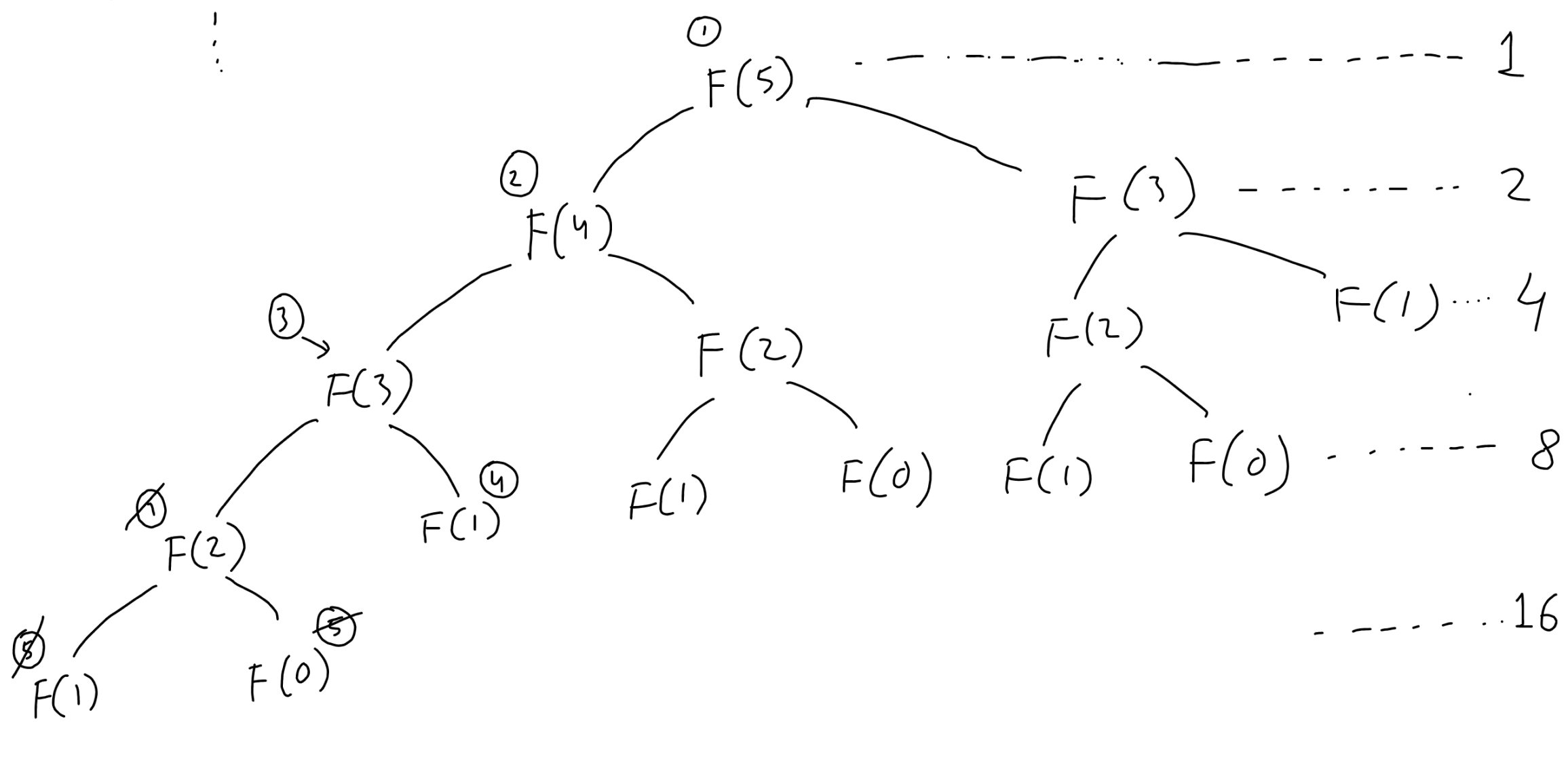
$F(n) = F(n-1) + F(n-2) \quad \forall \quad n > 1$

[$F(0) = F(1) = 1$]

0 → 1
1 → 1
2 → 2
3 → 3
4 → 5
5 → 8
6 → 13
⋮

```
int fib ( int  n) {
        if (n <= 1) return 1;
        return fib(n-1) + fib(n-2);
}
```

n = 5



No. of levels in the rec. tree = n

Worst-case no of recursive calls = $\underbrace{2^0 + 2^1 + 2^2 + 2^3 ------ + 2^{n-1}}_{n\text{-terms}}$

⇓

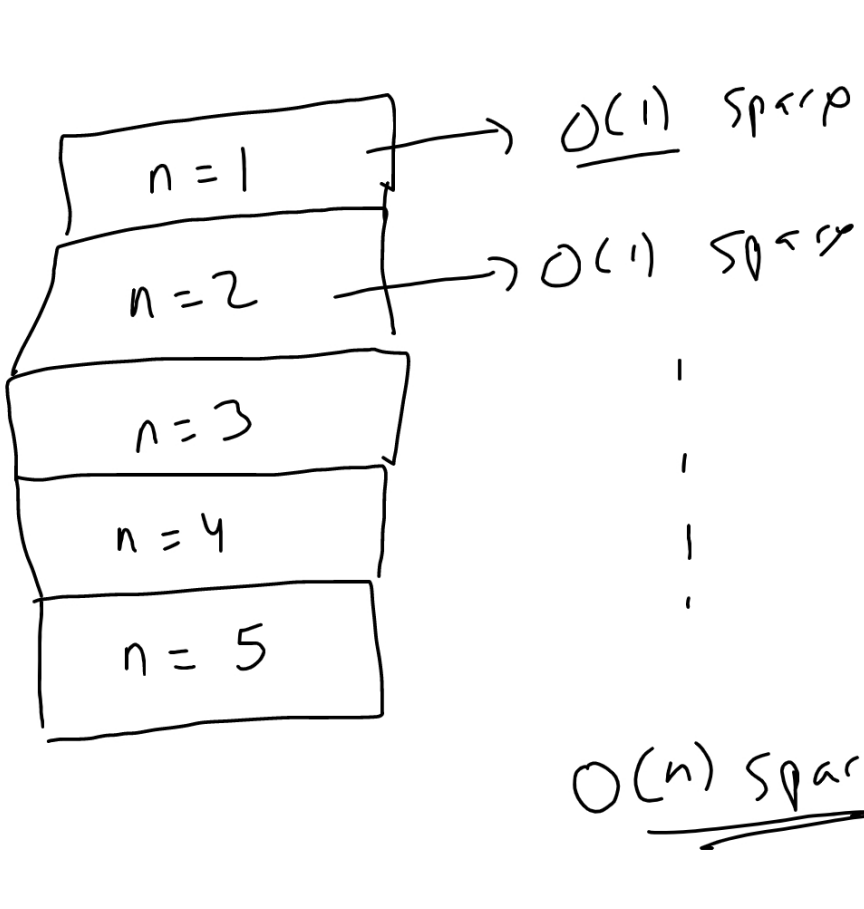
$\frac{1(2^n - 1)}{2-1} = O(2^n)$

[Sum of GP = $\frac{a(r^n - 1)}{r-1}$]

∴ TC of the above code = $\boxed{O(2^n)}$

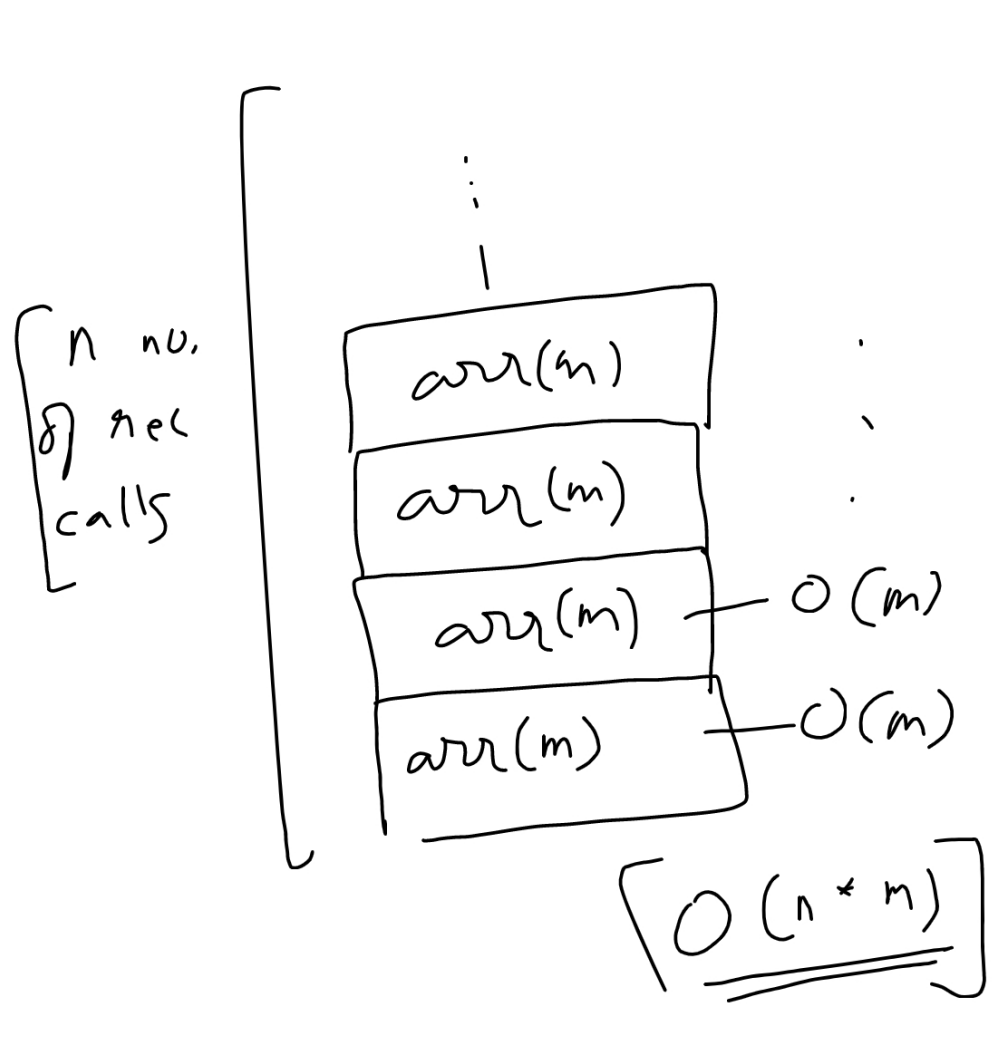
Aux. space of the above code = $O(n)$

$\Theta(1.67^n)$



O(n) space.

[n no. of rec calls]



[$O(n * m)$]