Time And Space Complexity :

↳ input & output complexities not considered.

check graphs for better clarity :

constant

$\log n$

$\sqrt{n}$

n

$n \log n$

n^2

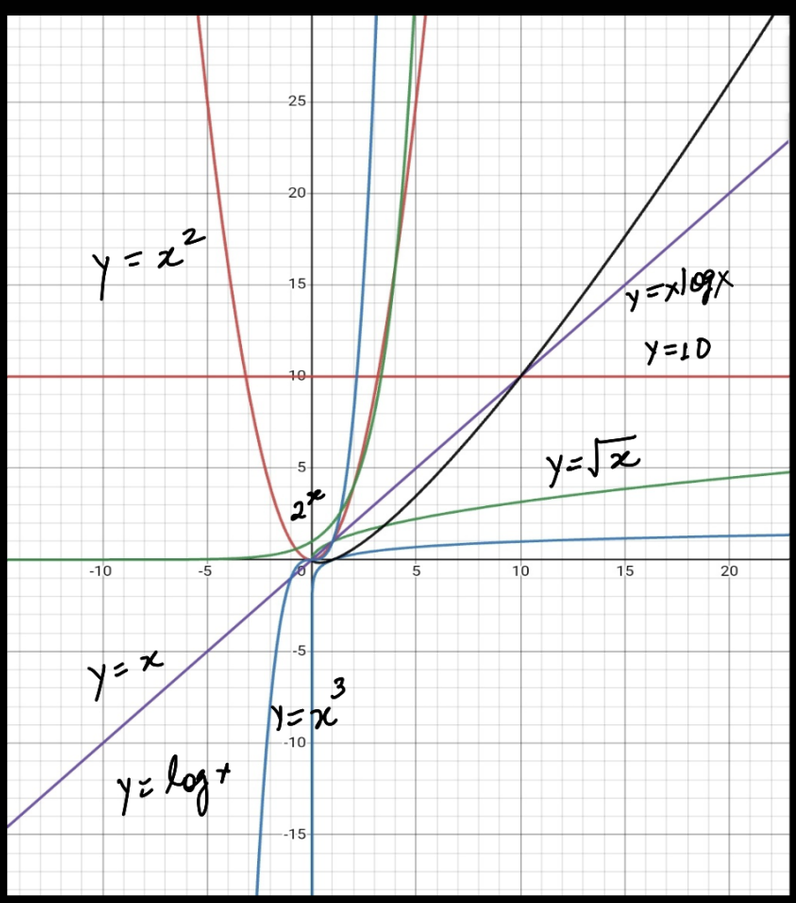
n^3

2^n

increasing time complexity (increasing growth)

growth ↑ slower code

growth ↓ faster code



⟹ [Different types of complexities]:

Worst Case (Big O) → O(n)

Best Case (Big Omega) → Ω(1)

Average Case (Big Theta) → Θ(n²)

Time complexity for :

1. 2 different independent loops :

eg:
```
for (int i=0 ; i<n ; i++){
;}

for (int j=0 ; j<m ; j++){
;}
```

Time compl. = $O(m+n)$

② Nested loops :

```
for (i<n){
  for(j<n) {
  } }
```

Time Complexity = $O(n^2)$

Rounding off / Neglecting : eg : $O\left(\frac{n}{2}(n-1)(n-2)\right)$

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

$= O(n^3 - 2n^2 - n^2 + 2n)$

$= O(n^3 - 3n^2 + 2n)$ (lower degree terms like n^2 & n neglected)

$= O(n^3)$

eg → $O(an+b)$ ∀ a & b are constants, then

$O(an+b) = O(n)$

```
# include <bits/stdc++.h>
```

→ includes all the header files.

eg:

```
for (i<n) {
  for (j<√n) {
  } }
```

Total instructions = $\sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + \ldots$ (n times)

$= n\sqrt{n}$

Time Complexity = $O(n\sqrt{n})$

Space Complexity :

↳ extra memory space requirement of an algorithm. (using asymptotic analysis).

space size changes w.r.t. change in input.

$i = 1, 2, 4, \ldots \Rightarrow 2^K$

$j \rightarrow 1, 2, 4 \text{ ------}$

$1, 2, 4, 8 \text{ ------}$ $\log_2 N$ terms.

$$2^K \leq N$$

$$K = \log_2 N$$

$$S = \frac{a(r^n - 1)}{r - 1}$$

$$= \frac{1\left(2^{\log_2 N} - 1\right)}{2 - 1}$$

$$= O\left(2^{\log_2 N}\right)$$

$$= O\left(N^{\log_2 2}\right)$$

$$= O(N)$$

```
int val = 0;
for (int i = N ; i > 0 ; i /= 2){
    for (int j = 0 ; j < i ; j++){
        val ++; } }
```

Ques:

itr	i	j
1	N	N
2	$\frac{N}{2}$	$\frac{N}{2}$
3	$\frac{N}{4}$	$\frac{N}{4}$
⋮	⋮	⋮
k	1	1

$N + \frac{N}{2} + \frac{N}{4} + \cdots + 1$

$\Rightarrow N\left(\frac{\left(\frac{1}{2}\right)^{k} - 1}{\frac{1}{2} - 1}\right)$

$1 = \frac{N}{2^{k-1}}$

$2^{k-1} = N$

$k - 1 = \log_2 N$

$(k = \log_2 N + 1)$

$= N\left(\frac{\left(\frac{1}{2}\right)^{\log_2 N} - 1}{\frac{1}{2} - 1}\right)$

$$= \frac{N\left(\frac{1}{2}\right)^{\log_2 \frac{1}{2}} - 1)}{-\frac{1}{2}}$$

$$= \frac{N\left(N^{\log_2 \frac{1}{2}} - 1\right)}{-\frac{1}{2}}$$

$$= -2N\left(N^{-1} - 1\right)$$

$$= -2N\left(\frac{1-N}{N}\right)$$

$$= 2N - 2$$

$$= N$$

Ques → int val = 0;

for (int i = 2 ; i ≤ N ; i *= i) {

iteration	1	2	3
i	2	2^1	2^{2^0}

2

3

4

$4 \longrightarrow 2^2 \longrightarrow 2^{2^1}$

$16 \longrightarrow 2^4 \longrightarrow 2^{2^2}$

$256 \longrightarrow 2^8 \longrightarrow 2^{2^3}$

Let total itr = T ;

$k = 2^{T-1}$

2^k 2^{T-1}

$2^k \leq N$

$k \leq \log_2 N$

$k = 2^{T-1}$

$k = 2^T$

$T = \log_2 k$

$T = \log_2 (\log_2 N)$

$$\boxed{T = \log (\log N)}$$

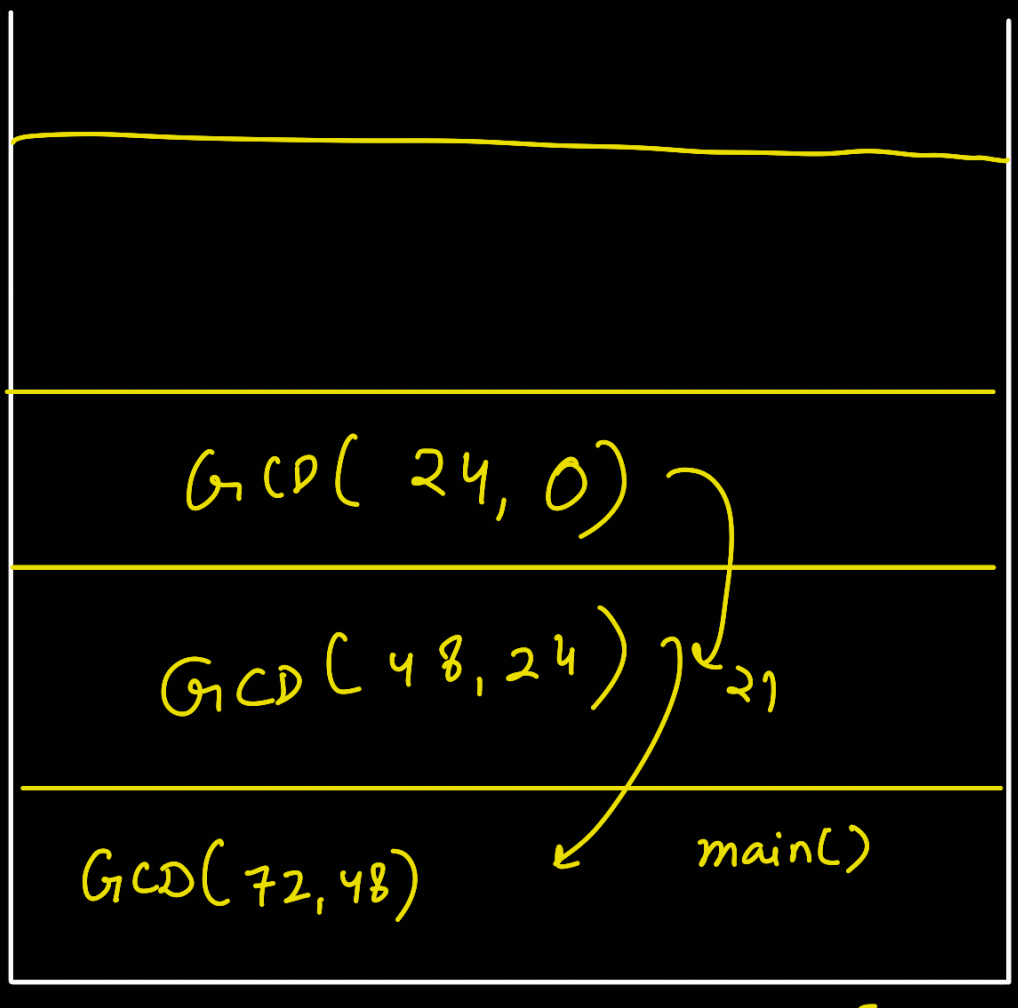
☆ Euclid's Algorithm ; $(a > b)$

$$\boxed{GCD(b, a-b) = GCD(b, a\%b)}$$

GCD (a, b) =

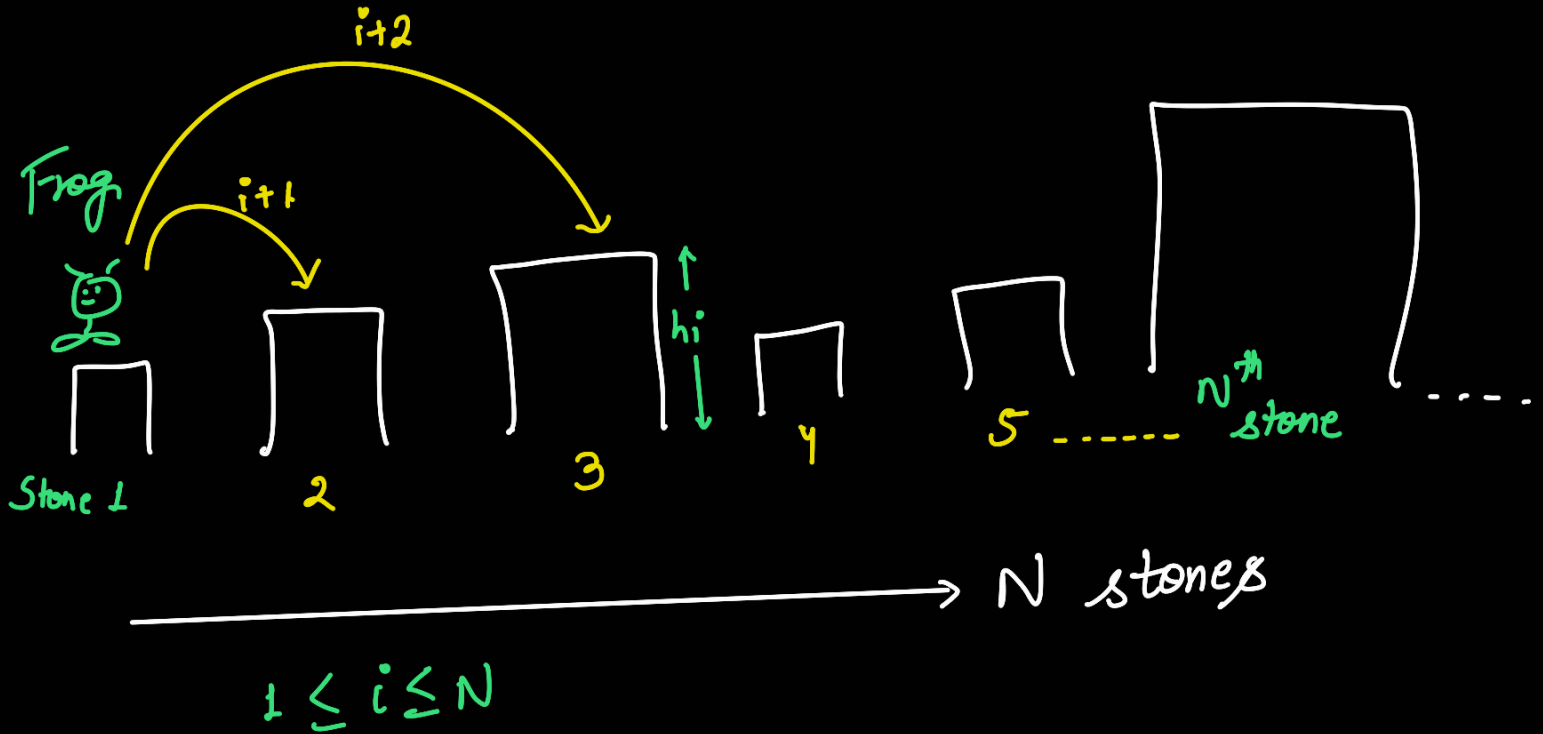
```
if (b > a)  return gcd(b, a);
if (b == 0) return a;
return GCD( b, a % b);
```

GCD (48, 72)



Frog Jump Problem :

2 options: (fwd & bwd)



$1 \leq i \leq N$

$abs(h_i - h_j)$ → cost is incurred.

find minimum cost incurred to reach n^{th} (last) stone.

eg → I/P → 10 30 40 20

no of ways frog can jump: cost

10 → 30 → 40 → 20 20+10+20 = 50

10 → 40 → 20 30+20 = 50

20+10 = 30

10 → 30 → 20

Best path.

Base case → agar last ston पर है तो एक jump.

second last → one पर 2 Jump या 2 पर 1 jump.

→ min(पहला 1 , पहला 2)

→

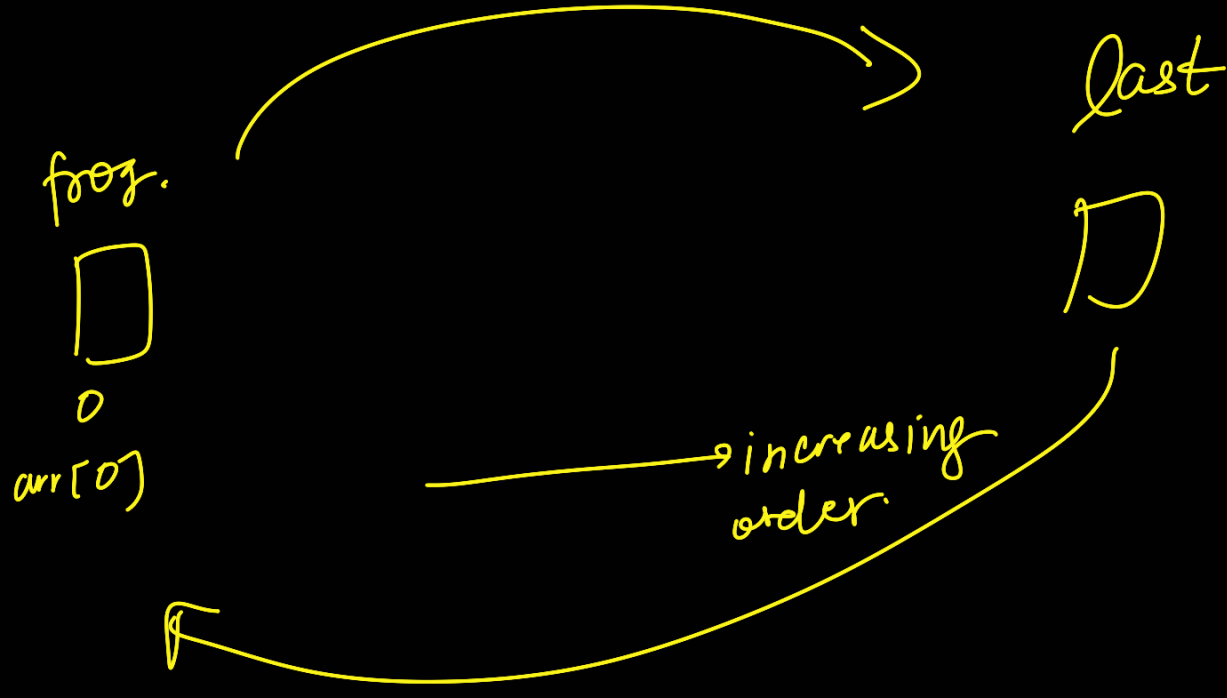
f → min cost

$s = 3$

$s = 2 + 3 * 10 = 32$

$s = 1 + 32 * 10 = 321$

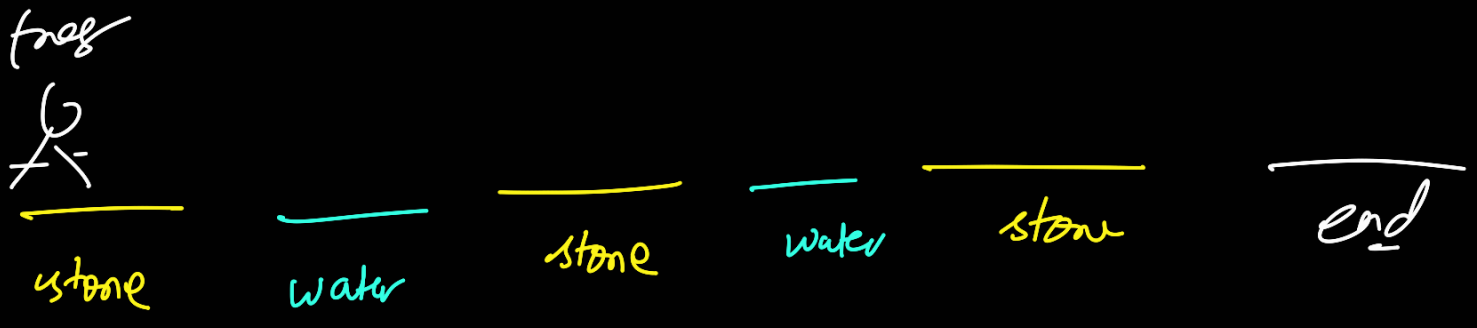
```
if n == 0; return 0;
s = 0
s = n % 10 + (s * 10)
```



arr[i] - arr[j]

1 stone only once jumpable

→ only forward possible.



→ aseending order.

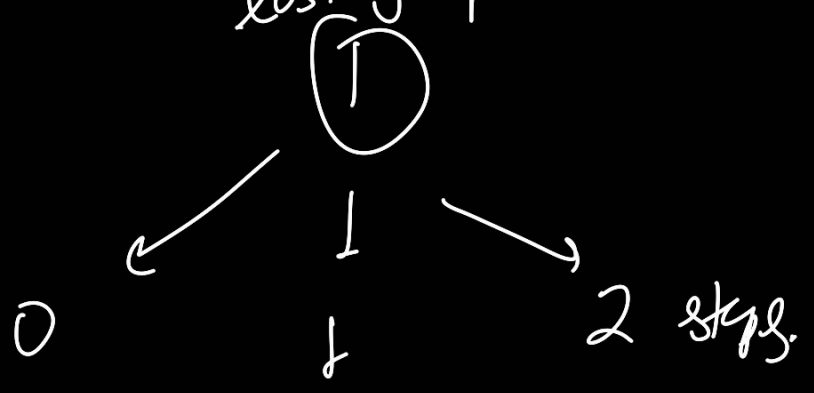
can for jump on last stone

1st jump → 1 unit

If last jump is k

R-1 R R+1

last jump.



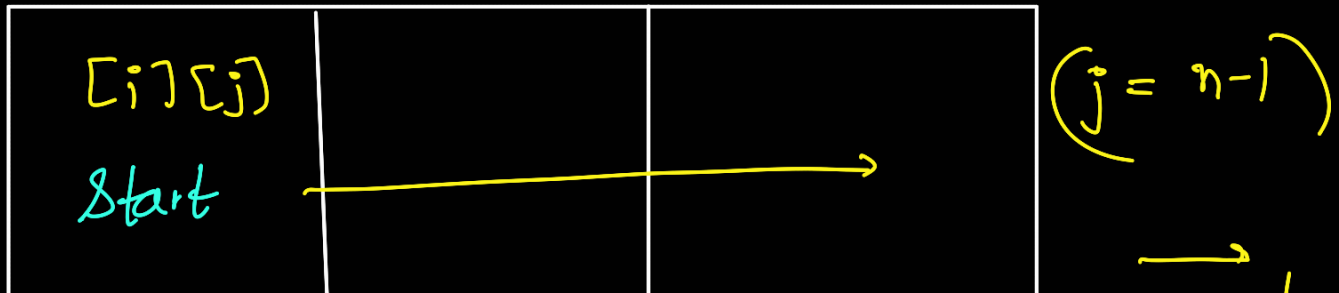
arr = {2, 3} I/P

print subset sums:

subsets	O/P sum
ϕ	0
2	2
3	3
2, 3	5

subarray → continuous elements

subsets → non-continuous elements



(j = n-1)

Smaller Subproblem ⇒ If i am at last col; I can only move ↓

and

If I am at the last row I can only move right.

(i = m-1)

* Recursion problem approach is:

⇒ Intuition / assumption
⇒ Self work
⇒ Base case