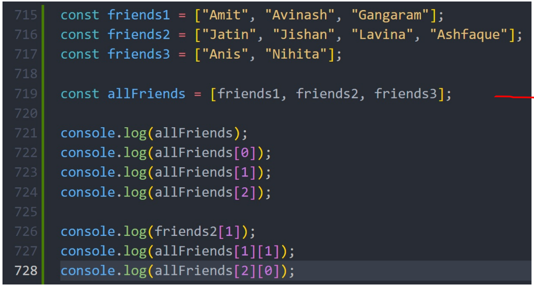
* Array of Arrays (2D Arrays) :

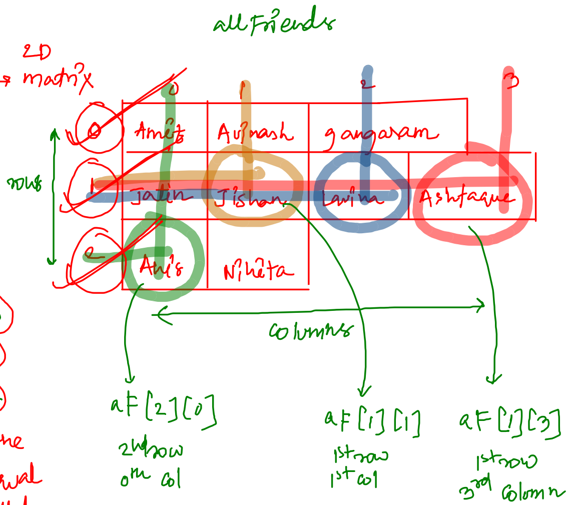


```
const friends1 = ["Amit", "Avinash", "Gangaram"];
const friends2 = ["Jatin", "Jishan", "Lavina", "Ashfaque"];
const friends3 = ["Anis", "Nihita"];

const allFriends = [friends1, friends2, friends3];

console.log(allFriends);
console.log(allFriends[0]);
console.log(allFriends[1]);
console.log(allFriends[2]);

console.log(friends2[1]);
console.log(allFriends[1][1]);
console.log(allFriends[2][0]);
```



[
0 [A, Avi, gang], (0 1 2) → 0
1 [Ja, Ji, La, Ash] (0 1 2 3) → 1
2 , [An, Ni] (0 1) → 2
]

no. of cols
allFriends [0].length → 3
allFriends [1].length → 4
allFriends [2].length → 2

* If the arrays or the rows do not have equal length they are called a "Jagged Arrays".

rows = no. of ele (no. of smaller Arrays)
= allFriends.length.

Given n = 3

Create a n x n matrix

	0	1	2
0	1	2	3
1	1	2	3
2	1	2	3

3x3
↓ ↓
rows cols

n = 4

	0	1	2	3
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	2	3	4

4x4
↓ ↓
rows cols

★ i is making you travel on rows that is going to each smaller array, j is making you travel on cols that is going inside of each smaller arr.

```
let arr = [];

for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
        arr[i][j] = j+1;
    }
}
```

arr[i] = []
(or)
arr.push([])

arr[i].push(j+1);

★ (arr[0])[0]
(arr[0])[1] ⇒ undefined[0] ⇒ error
[1]

arr[0] ⇒ I should not get undefined
⇒ [] [0] = 1
⇒ [1]

n = 3

i = 0
j = 0 → arr[0][0] = 1
j = 1 → arr[0][1] = 2
j = 2 → arr[0][2] = 3

i = 1
j = 0 → arr[1][0] = 1
j = 1 → arr[1][1] = 2
j = 2 → arr[1][2] = 3

i = 2
j = 0 → 2,0
= 1 → 2,1
= 2 → 2,2

* Time :

```
for (i=0; i<rows; i++) {
    for (j=0; j<cols; j++) {
    }
}
```

⇒ $O(rows * cols)$

⇒ $O(n * m)$

⇒ if $r == c$, $O(n^2)$

Space : 2D array required ⇒ $O(rows * cols)$

mat [r] [c] → no · of smaller ar

~~mat [c] [r]~~ → errors

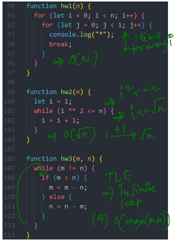
	0	1	2	3
0	1	2	3	4
1	1	2	3	4

(0,0) (0,1) (0,2) (0,3)

(1,0) (1,1) (1,2) (1,3)

mat [0] [3]

mat [3] [0] ✗



```javascript
function hw1(n) {
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < i; j++) {
      console.log("*");
      break;
    }
  }
}

function hw2(n) {
  let i = 1;
  while (i ** 2 <= n) {
    i = i + 1;
  }
}

function hw3(m, n) {
  while (m != n) {
    if (m > n) {
      m = m - n;
    } else {
      n = n - m;
    }
  }
}
```

1 time for every i

$\Rightarrow O(N)$

$\Rightarrow i^2 <= n$

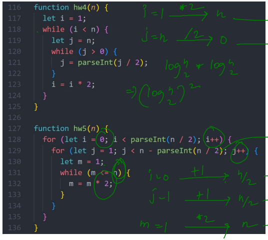
$\Rightarrow i <= \sqrt{n}$

$\Rightarrow O(\sqrt{n})$ $1 \xrightarrow{+1} \sqrt{n}$

TLE

$\Rightarrow$ infinite loop

(4) O(max(m,n))



```javascript
function hw4(n) {
  let i = 1;
  while (i < n) {
    let j = n;
    while (j > 0) {
      j = parseInt(j / 2);
    }
    i = i * 2;
  }
}

function hw5(n) {
  for (let i = 0; i < parseInt(n / 2); i++) {
    for (let j = 1; j < n - parseInt(n / 2); j++) {
      let m = 1;
      while (m <= n) {
        m = m * 2;
      }
    }
  }
}
```

$i = 1 \xrightarrow{*2} n \rightarrow \log_2^n$

$j = n \xrightarrow{/2} 0 \rightarrow \log_2^n$

$\log_2^n * \log_2^n$

$\Rightarrow (\log_2^n)^2$

$n - n/2 = n/2$

$i = 0 \xrightarrow{+1} n/2 \rightarrow O(n)$

$j = 1 \xrightarrow{+1} n/2 \rightarrow O(n)$

$m = 1 \xrightarrow{*2} n \rightarrow O(\log_2^n)$

$O(n^2 \log_2^n)$

(1) m = 3, n = 0

(2) m = 3 - 0, n = 0
= 3

(3) m = 3 - 0, n = 0
= 3

(4) m = 3 - 0, n = 0
= 3

(1) n = 3, m = 0

(2) n = 3 - 0, m = 0
= 3

(3) n = 3 - 0, m = 0
= 3

(4) n = 3 - 0, 0
= 3

infinite

// hw3 (constraints) $\Rightarrow$ m > 0
n > 0

(1) m = 4, n = 1

(2) m = 4 - 1, n = 1
= 3

(3) m = 3 - 1, n = 1
= 2

(4) m = 2 - 1, n = 1
= 1

(5) m = 1, n = 1 ✗

4

O(m)

(1) m = 1, n = 4

(2) m = 1, n = 4 - 1
= 3

(3) m = 1, n = 3 - 1
= 2

(4) m = 1, n = 2 - 1
= 1

(5) m = 1, n = 1 ✗

4

O(n)

O(max (m, n))

m = 3, n = 4

(1) m = 3, n = 4 - 3
= 1

(2) m = 3 - 1, n = 1
= 2

(3) m = 2 - 1, n = 1
= 1

(4) m = 1, n = 1 ✗

3