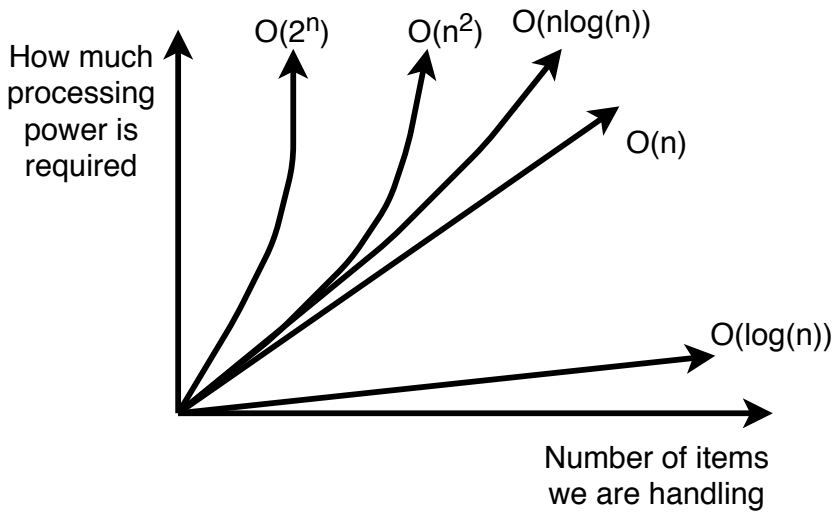
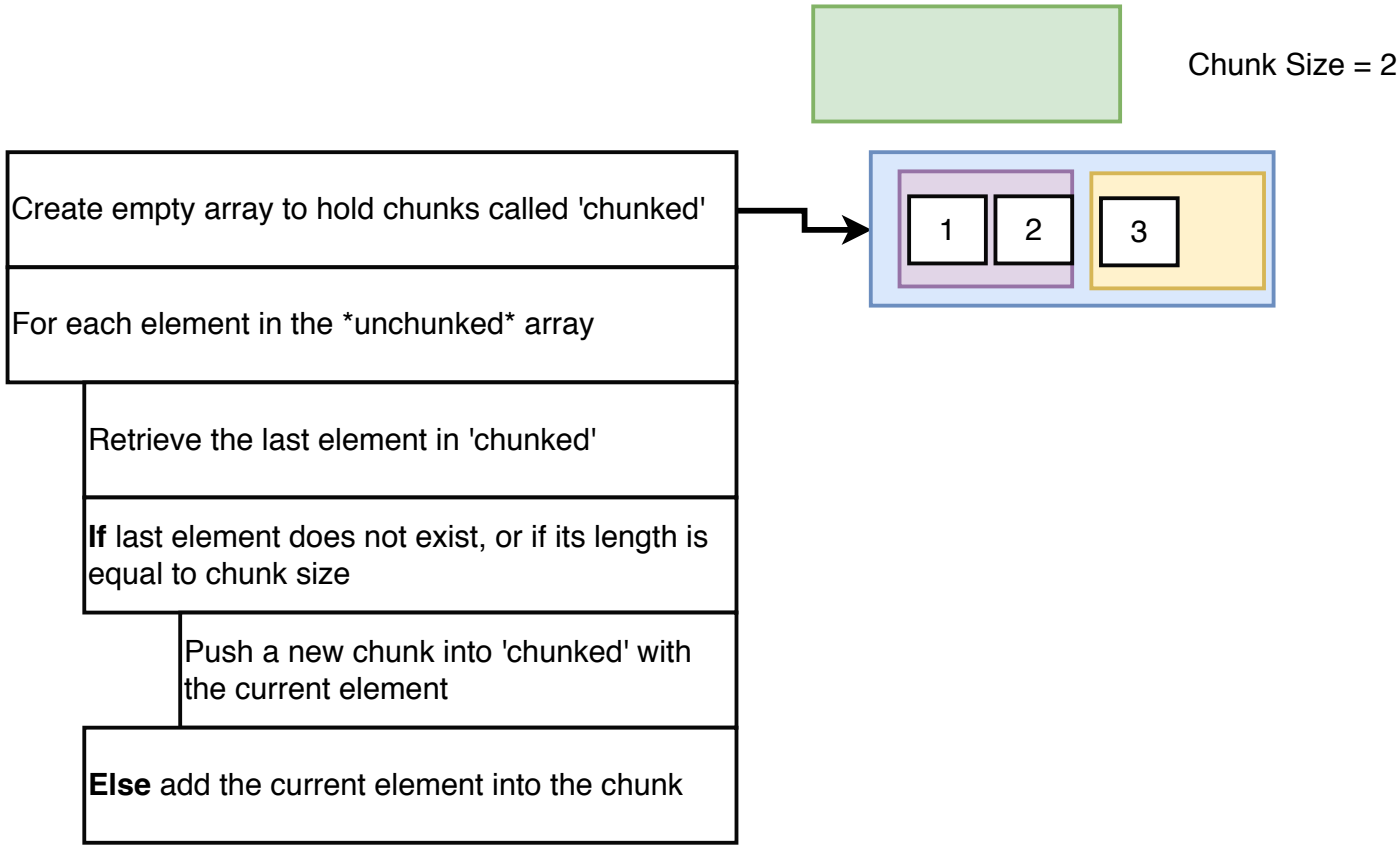
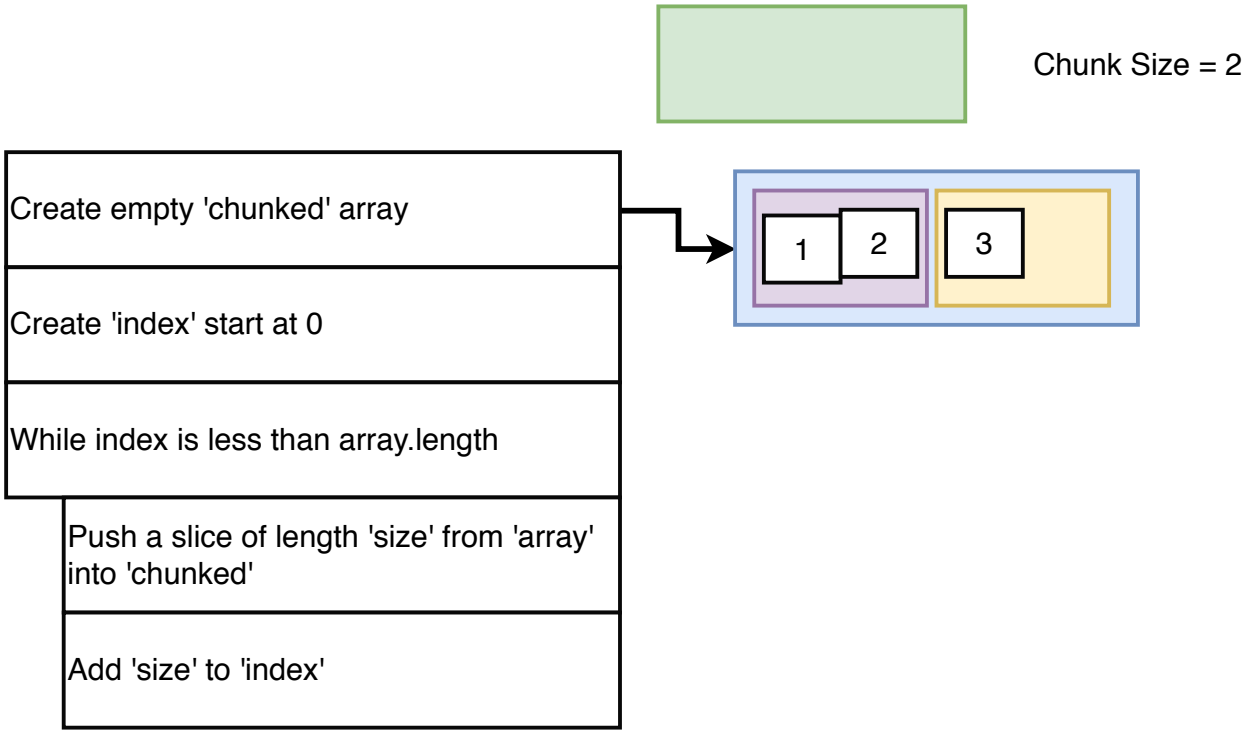
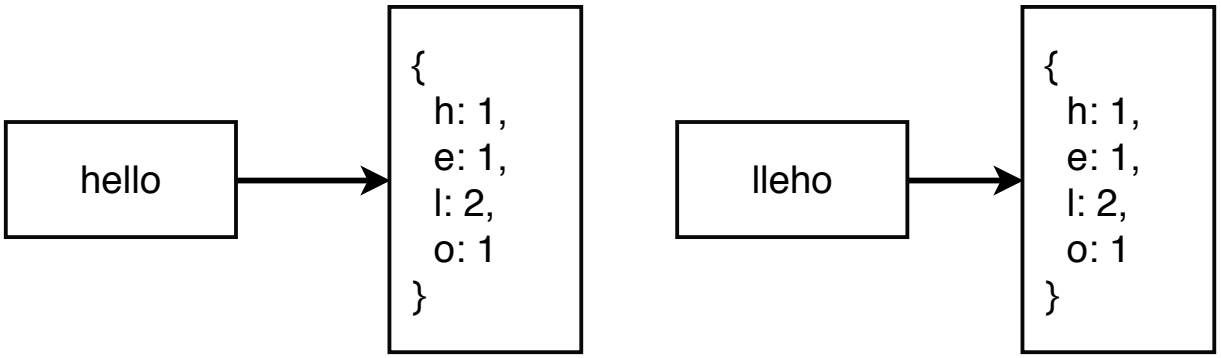
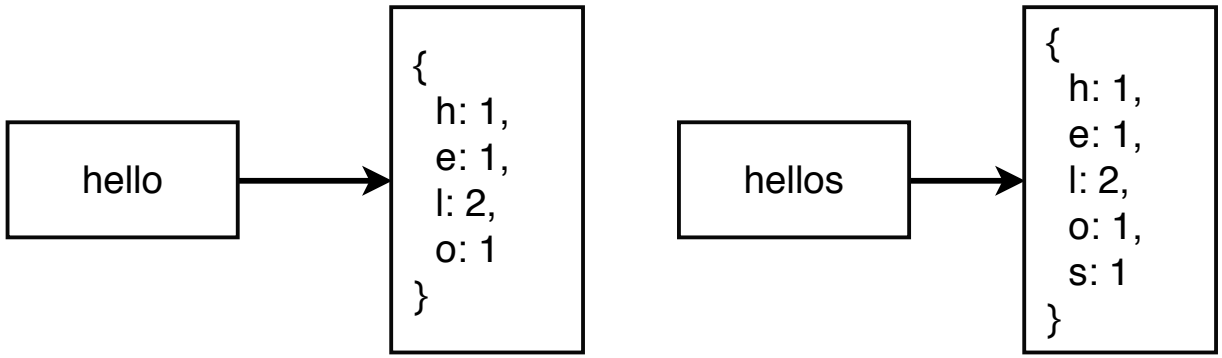

How much processing power is required
O(2^n)
O(n^2)
O(nlog(n))
O(n)
O(log(n))
Number of items we are handling


Chunk Size = 2
Create empty array to hold chunks called 'chunked'
For each element in the *unchunked* array
Retrieve the last element in 'chunked'
If last element does not exist, or if its length is equal to chunk size
Push a new chunk into 'chunked' with the current element
Else add the current element into the chunk
1
2
3

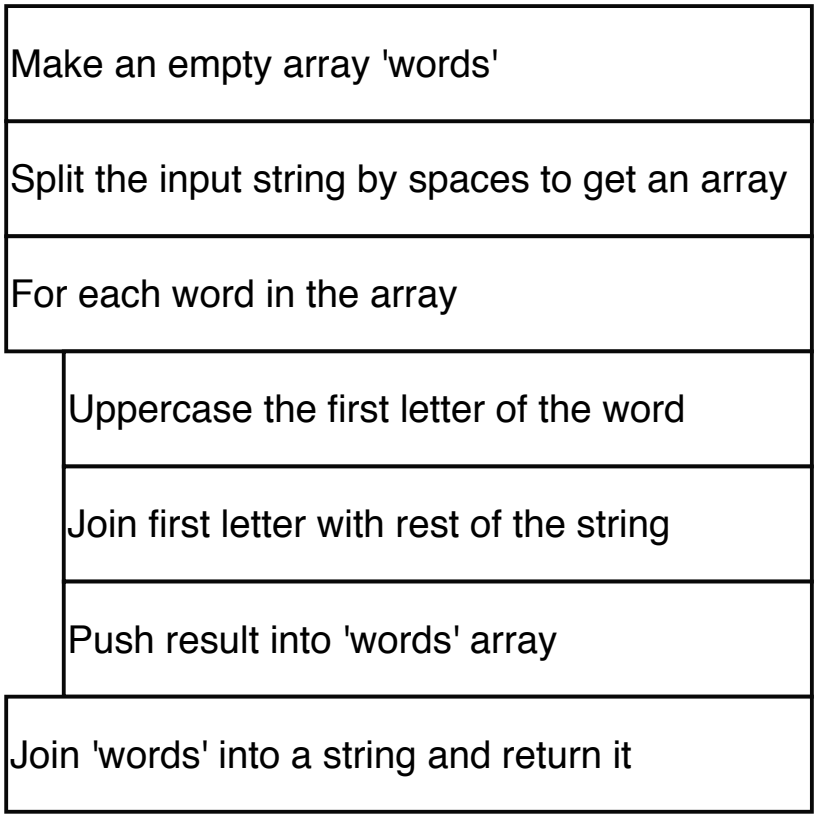

Chunk Size = 2
Create empty 'chunked' array
Create 'index' start at 0
While index is less than array.length
Push a slice of length 'size' from 'array' into 'chunked'
Add 'size' to 'index'
1
2
3


hello
{
h: 1,
e: 1,
l: 2,
o: 1
}
lleho
{
h: 1,
e: 1,
l: 2,
o: 1
}


hello
{
h: 1,
e: 1,
l: 2,
o: 1
}
hellos
{
h: 1,
e: 1,
l: 2,
o: 1,
s: 1
}

Debugger Steps

Add a 'debugger' statement in your function
Call the function manually
At the terminal, run 'node inspect index.js'
To continue execution of the file, press 'c' then 'enter'
To launch a 'repl' session, type 'repl' then 'enter'
To exit the 'repl', press Control + C


Make an empty array 'words'
Split the input string by spaces to get an array
For each word in the array
Uppercase the first letter of the word
Join first letter with rest of the string
Push result into 'words' array
Join 'words' into a string and return it

Create 'result' which is the first character of the input string capitalized

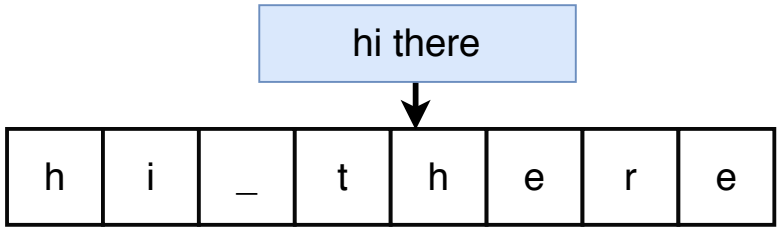
For each character in the string

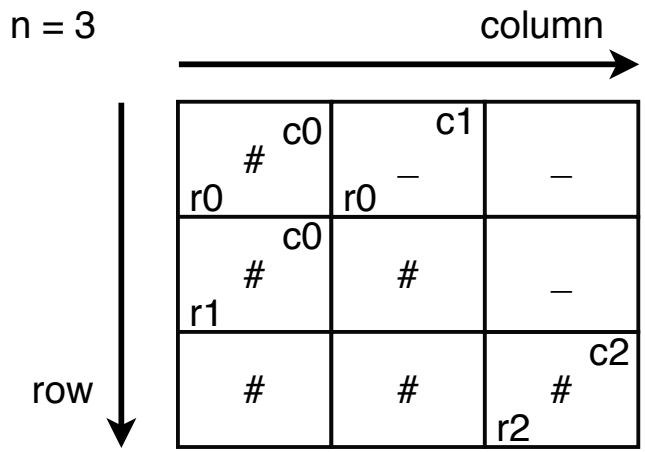
IF the character to the left a space

Capitalize it and add it to 'result'

ELSE

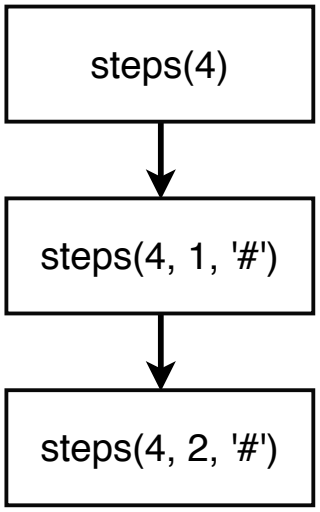
Add it to 'result'

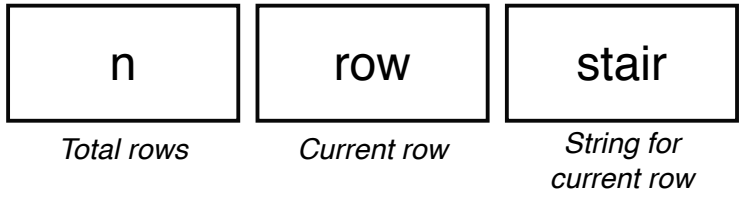




From 0 to n (iterate through rows)

- Create an empty string, 'stair'
- From 0 to n (iterate through columns)
  - **IF** the current column is equal to or less than the current row
    - Add a '#' to 'stair'
  - **ELSE**
    - Add a space to 'stair'
- Console log 'stair'


steps(4)
steps(4, 1, '#')
steps(4, 2, '#')



IF row is equal to n

- No more work is left, just return

IF length of 'stair' is equal to n

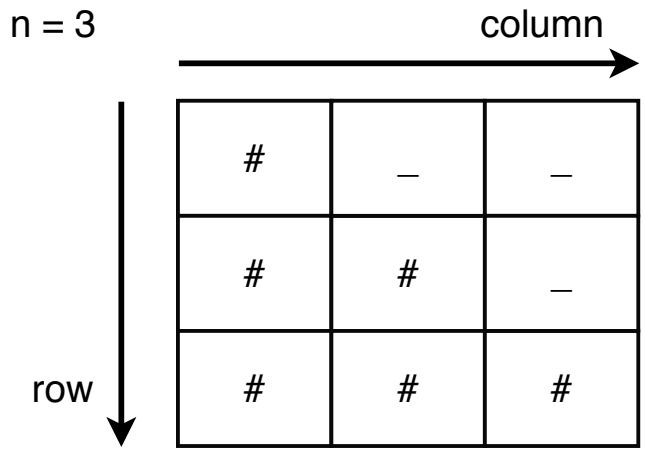
- We are at the end of a row and need to log the current stair
- Increment row counter and call 'steps' again

ELSE

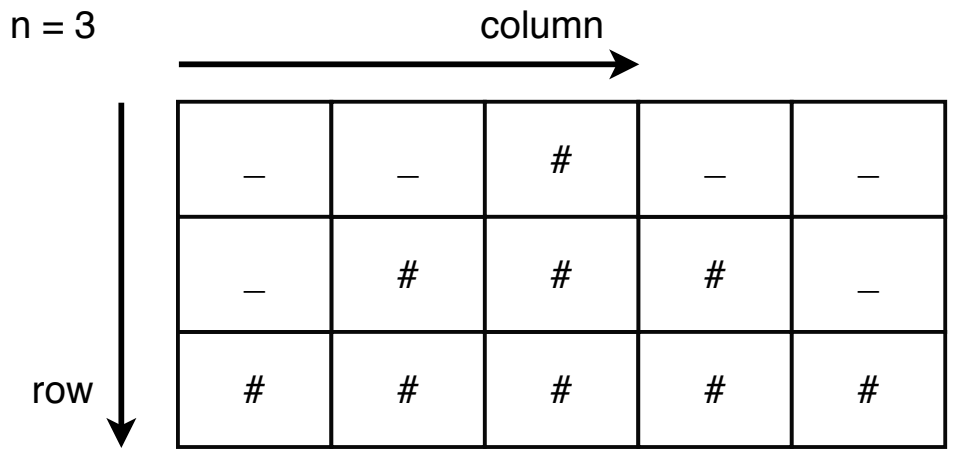
- **IF** length of stairs is less than or equal to the current row
  - Add a '#' to 'stairs' string
- **ELSE**
  - Add a space to 'stairs' string
- Call 'steps' again

Recursion Tips

Figure out the bare minimum pieces of information to represent your problem
Give reasonable defaults to the bare minimum pieces of info
Check the base case. Is there any work left to do? If not, return
Do some work. Call your function again, making sure the arguments have changed in some fashion

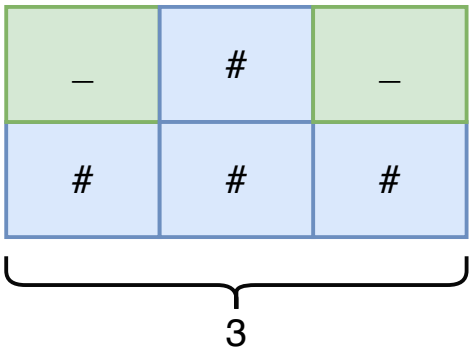


If (row === n) then we have hit the end of our problem
If the 'stair' string has a length === n then we are at the end of a row
If the *length* of the stair string is less than or equal to the row number we're working on, we add a '#', otherwise add a space

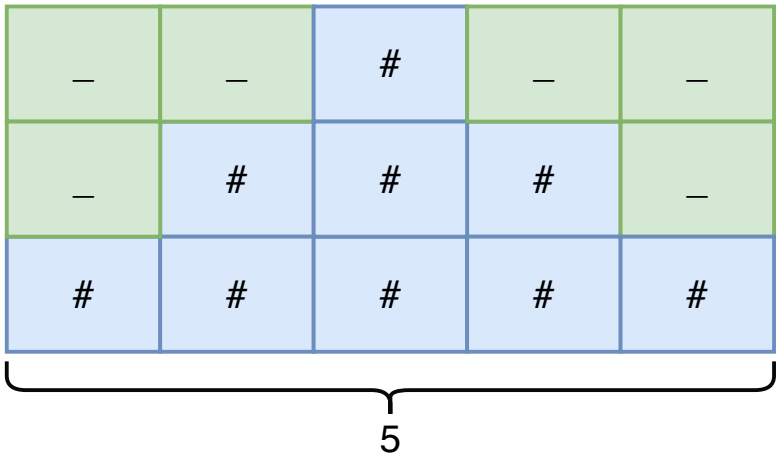


From 0 to n (iterate through rows)
- Create an empty string, 'level'
- From 0 to ??? (columns)
  - **IF** the column should have a #
    - Add a '#' to 'level'
  - **ELSE**
    - Add a space to 'level'
- Console log 'stair'

N = 2

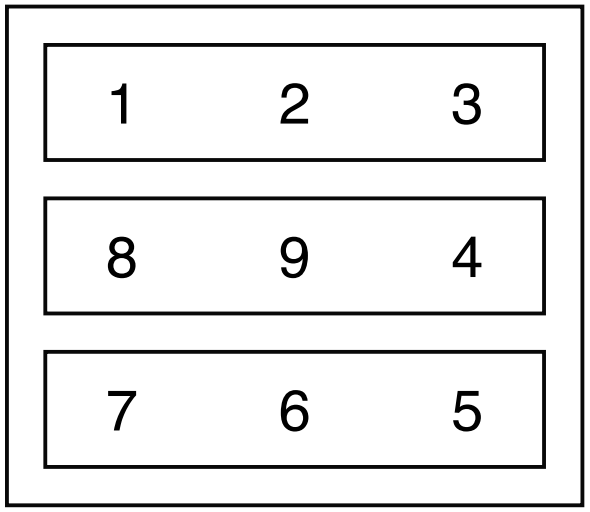


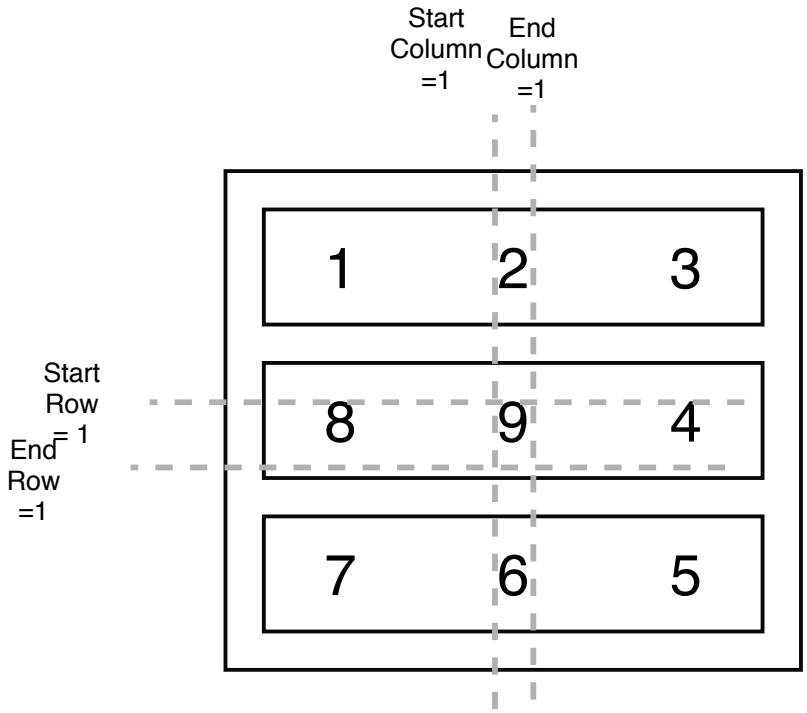
N = 3



N = 4

7


1 2 3
8 9 4
7 6 5



- Create empty array of arrays called 'results'
- Create a counter variable, starting at 1
- As long as (start column <= end column) **AND** (start row <= end row)
  - Loop from start column to end column
    - At results[start_row][i] assign counter variable
    - Increment counter
  - Increment start row
  - Loop from start row to end row
    - At results[i][end_column] assign counter variable
    - Increment counter
  - Decrement end column
  - ...repeat for other two sides