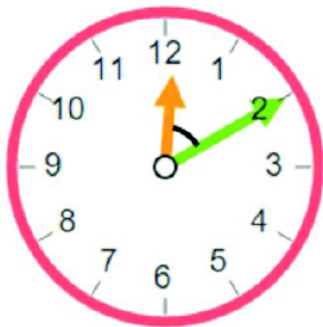
The background is a dark navy blue. It features several overlapping, semi-transparent geometric shapes in various colors: bright green, cyan, magenta, orange, and blue. These shapes are arranged in a way that creates a sense of depth and movement, with some appearing to be layered on top of others. The overall aesthetic is modern and tech-oriented.

# Trucos, técnicas y nociones algorítmicas

# 1. Abstracción del mundo real

## 6. Clock Angles - 20 points



Given an **array of strings** containing string representations of time in hh:mm pattern, where hh is an integer between 00 and 23 and mm is an integer between 00 and 59, calculate the sum of the angles measured between the clock hands. If the input is not on the hh:mm pattern, you need to subtract 100 from the result. The angle **MUST** be the one between the hour hand and the minute hand, clockwise, and not the other way around. We ignore the movement caused by seconds. Every angle must be between 0 and  $2\pi$  (between 0 and  $360^\circ$ ).

Example:

Input: ["12:00", "17:30", "blabla", "20:21", "26:88"]

Output: 50.5

¿En el mundo real como funciona este objeto? ¿El problema me da indicaciones de esto?

## 2. La técnica de los dos punteros

### 2.1. Is palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring case sensitivity.

"A man, a plan, a canal: Panama"



"amanaplanacanalpanama"

## 2. La técnica de los dos punteros

### 2.2. Trapping water



**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]

**Output:** 6

**Explanation:** The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

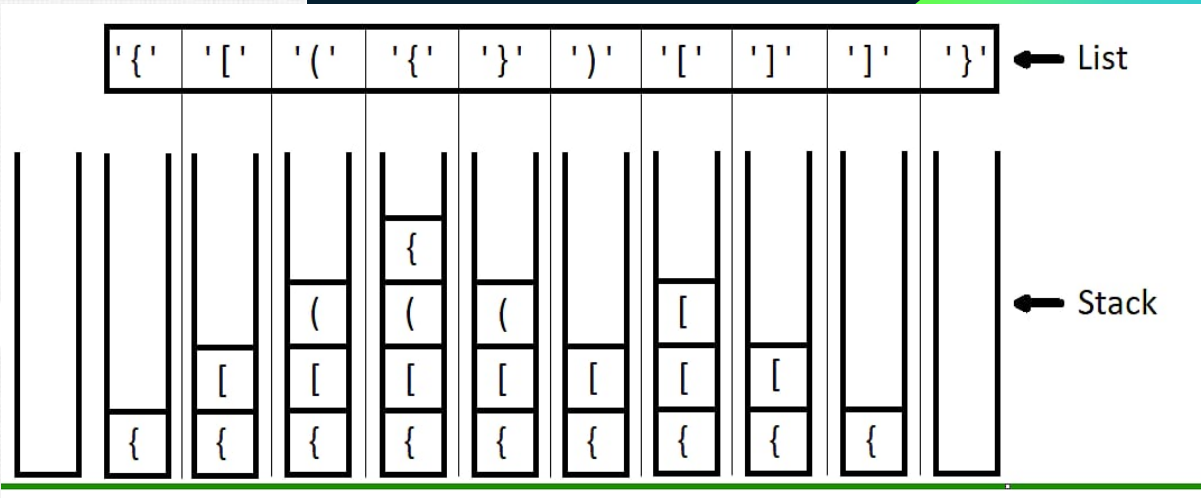
## 2. La técnica de los dos punteros

### 2.3. Almost a palindrome



[illegible]

"{ ([ ] ) }"

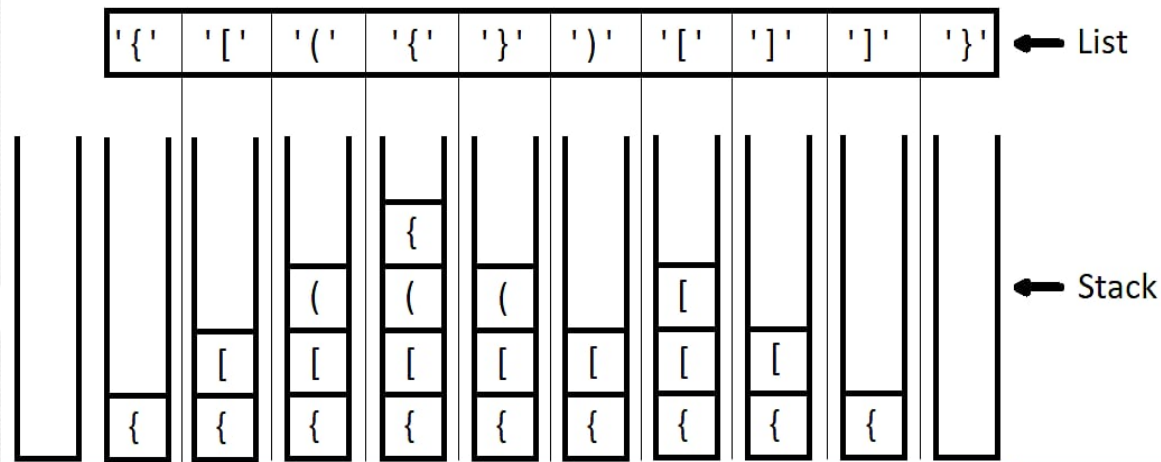


### 3. Stack and Queue

Given a string containing only parentheses, determine if it is valid. The string is valid if all parentheses close.

"{ ([ ] ) }"

"{ ( [ ] ) }"



## 4. Sliding window

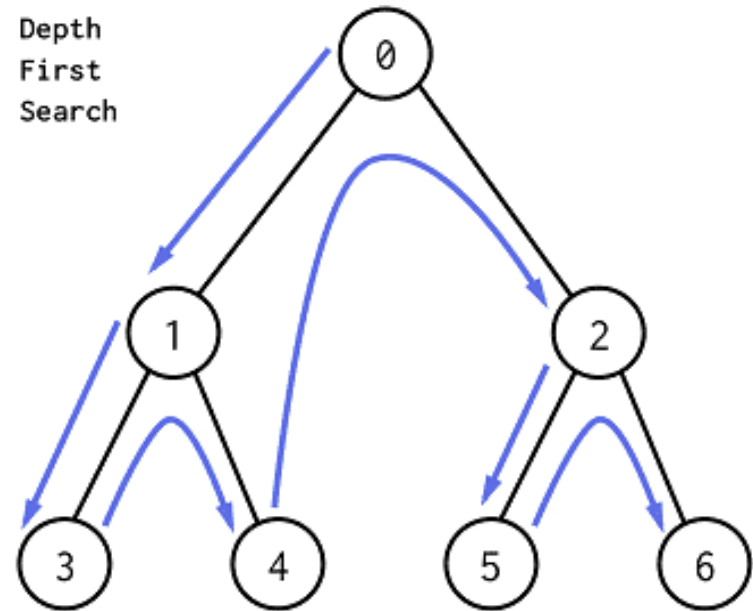
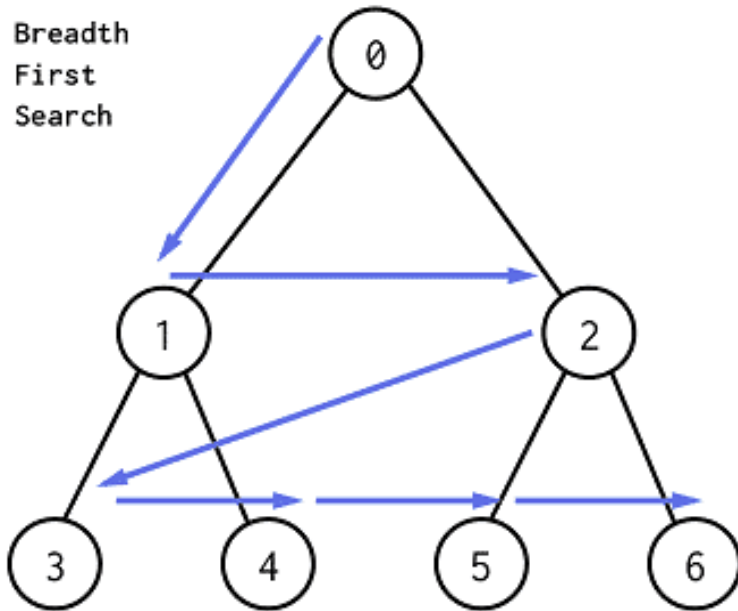
Given a string, find the length of the longest substring without repeating characters.

"abccab**b**"

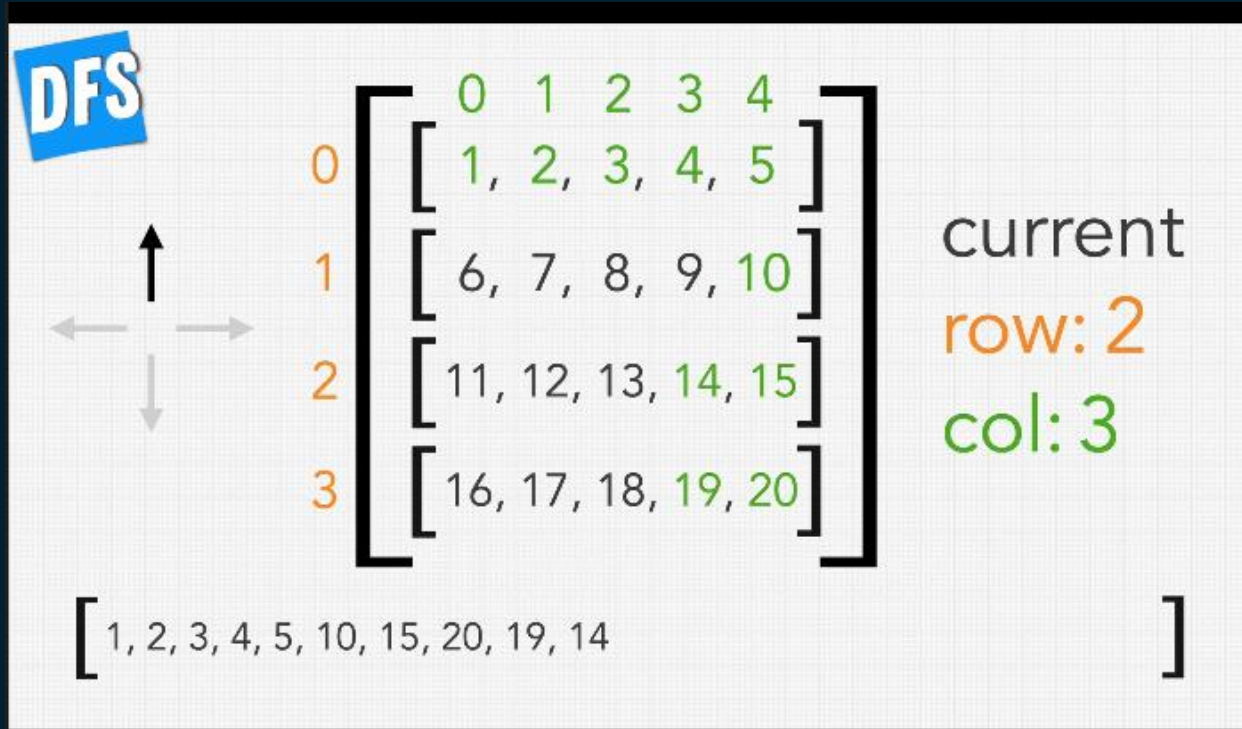
3



# Recall: DFS and BFS in graphs

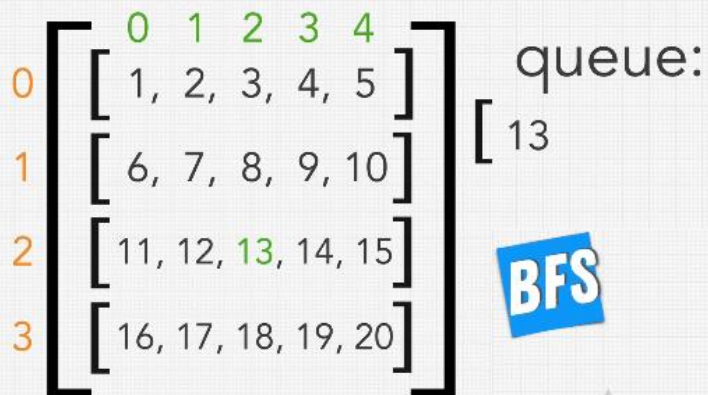


# Recall: DFS in 2D array



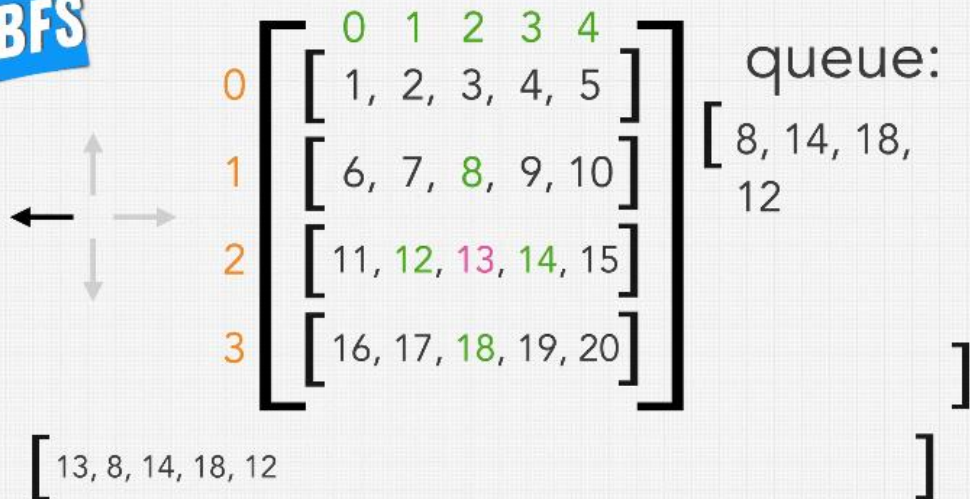
# Recall: BFS in 2D array

BFS

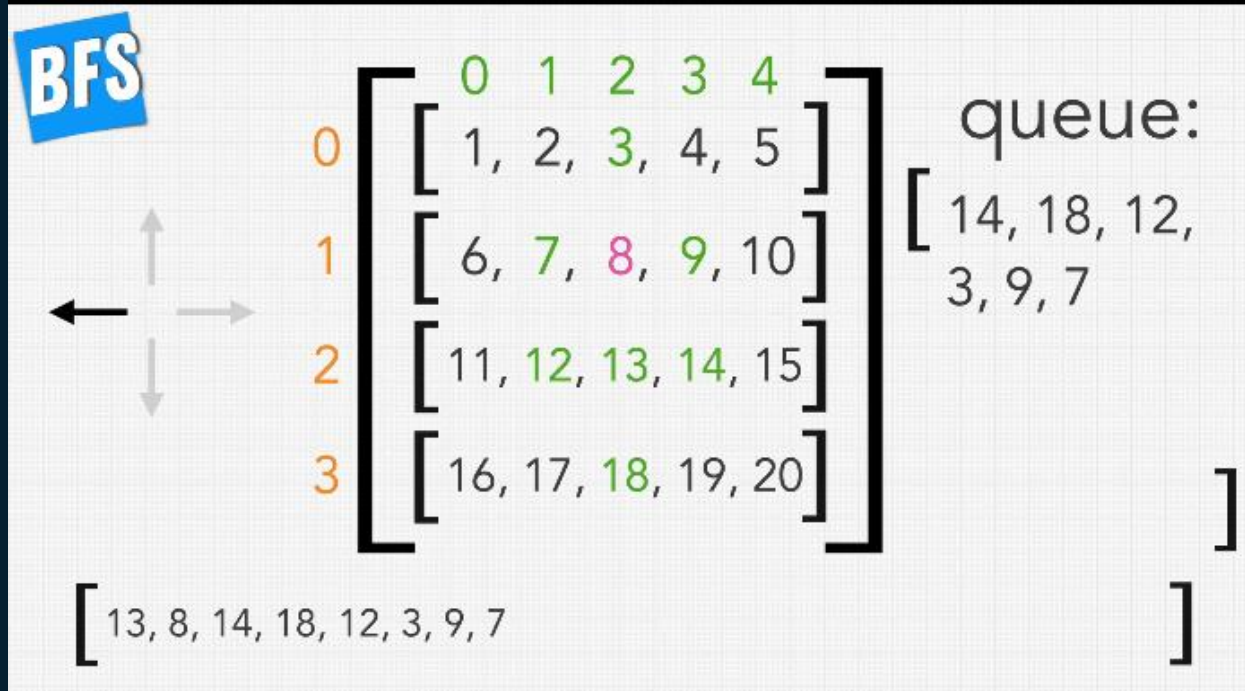


[ 13,

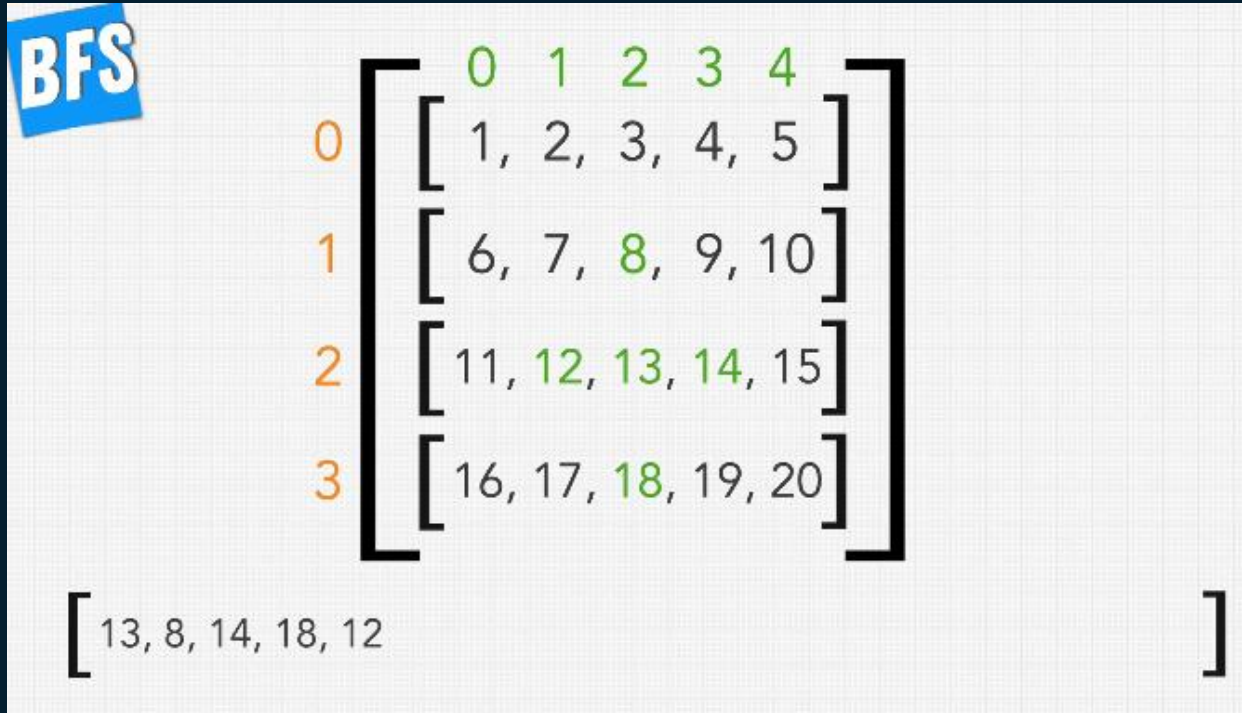
BFS



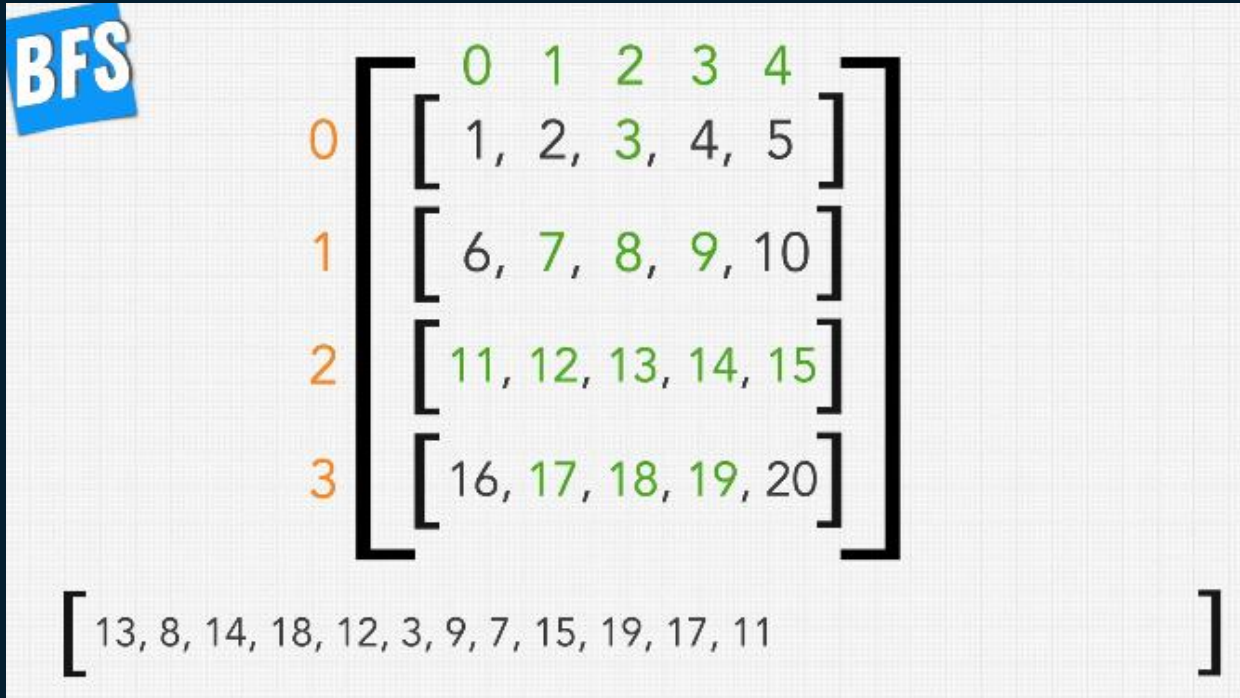
# Recall: BFS in 2D array



# Recall: BFS in 2D array



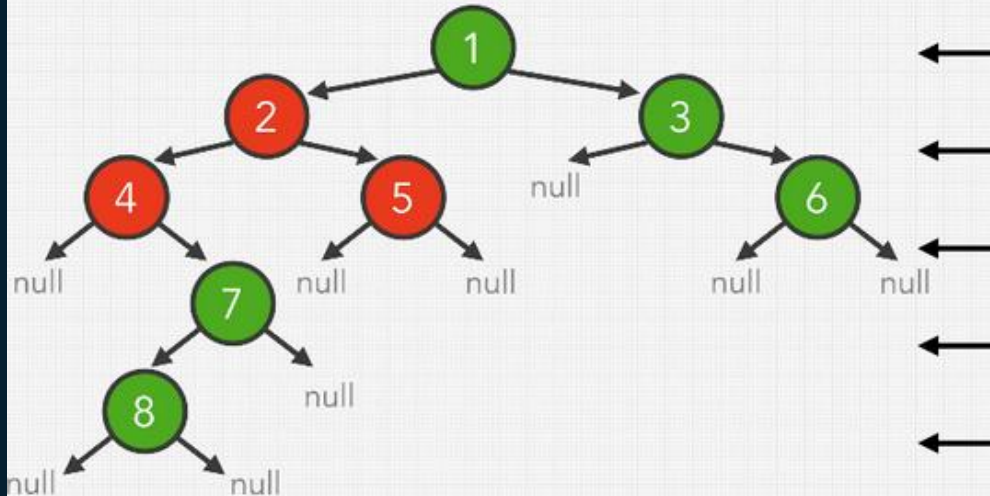
# Recall: BFS in 2D array





## 5. DFS (in graph)

Given a binary tree, imagine you're standing to the right of the tree. Return an array of the values of the nodes you can see ordered from top to bottom.



## 6. BFS (in 2D array)

Given a 2D array containing only 1's (land) and 0's (water), count the number of islands.

An island is land connected horizontally or vertically.

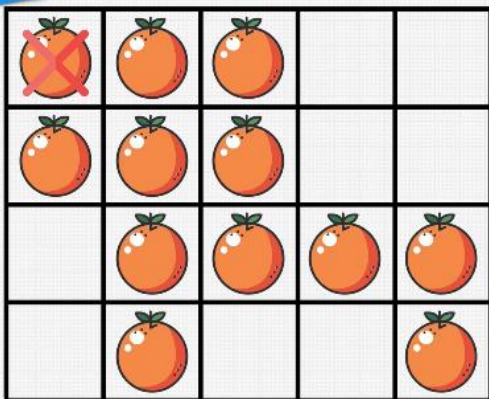
1	1	1	1	0
1	1	0	1	0
1	1	0	0	1
0	0	0	1	1

2

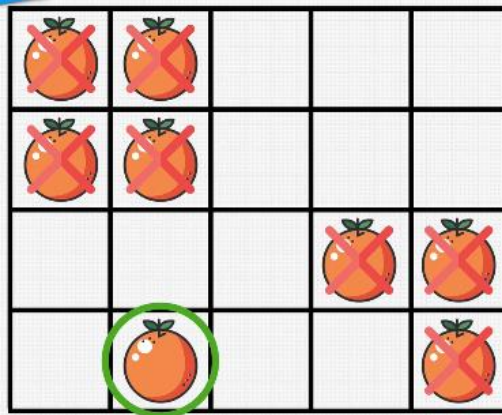


# Extra BFS

Given a 2D array containing 0's (empty cell), 1's (fresh orange) and 2's (rotten orange).  
Every minute, all fresh orange immediately adjacent (4 directions) to rotten oranges will rot.  
How many minutes must pass until all oranges are rotten?



0



-1

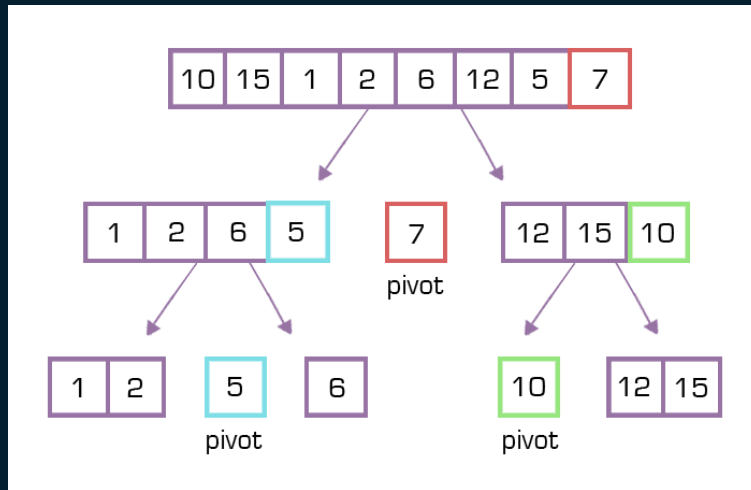
# 7. Sorting

For this problem, you are required to write a function that receives an array of numbers and should return the array sorted using for the comparison of their last digit, if their last digit is the same you should check the second to last and so on.

**Example:**

**Input:** [1, 10, 20, 33, 13, 60, 92, 100, 21]

**Output:** [100, 10, 20, 60, 1, 21, 92, 13, 33]





# 8. Decisions, best of multiple with DP

## 8.1. Arrange the pieces

### 16. Arrange the pieces - 60 points

Assuming you have an  $2 \times N$  board, you need to completely cover the board with either of these two shapes:

- A  $2 \times 1$  rectangle piece: 
- An L shaped piece: 

For example, if  $n = 4$ , **ONE** of the ways of filling the board would be like this:



Where the red and green pieces are L shaped and the blue piece is a  $2 \times 1$  rectangle. If  $n \leq 0$ , return 0.

Given an integer  $n$ , implement a function that returns **ALL** the number of ways we can fill the board..

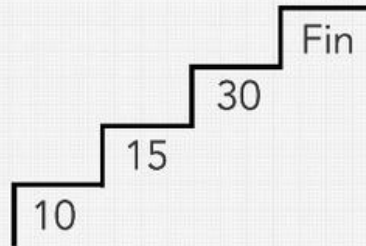
# 8. Decisions, best of multiple with DP

## 8.2. Minimum cost of climbing stairs

For a given staircase, the  $i$ -th step is assigned a non-negative cost indicated by a cost array.

Once you pay the cost for a step, you can either climb one or two steps. Find the minimum cost to reach the top of the staircase. Your first step can either be the first or second step.

cost = [10, 15, 30]



## 9. Kadane

### Kadane's Algorithm (Code)

$A = [-2, 3, 2, -1]$

Handwritten annotations:  $[2]$  above the 3,  $[3, 2]$  below the 3 and 2.

i \ j	0	1	2	3
0	-2	3	5	4
1	-2	3	5	5

Handwritten annotations: Circles around (0,1)=3, (0,2)=5, (1,3)=5. A diagonal line from (0,0) to (1,1).

```
def kadane(A):  
    max_current = max_global = A[0]  
    for i from 1 to length(A) - 1:  
        max_current = max(A[i],  
                           max_current + A[i])  
        if max_current > max_global:  
            max_global = max_current  
    return max_global
```

# 10. Product sub array

## 10. Product subarray - 35 points

Given an **array of floating point numbers**, find the contiguous subarray with the largest product.

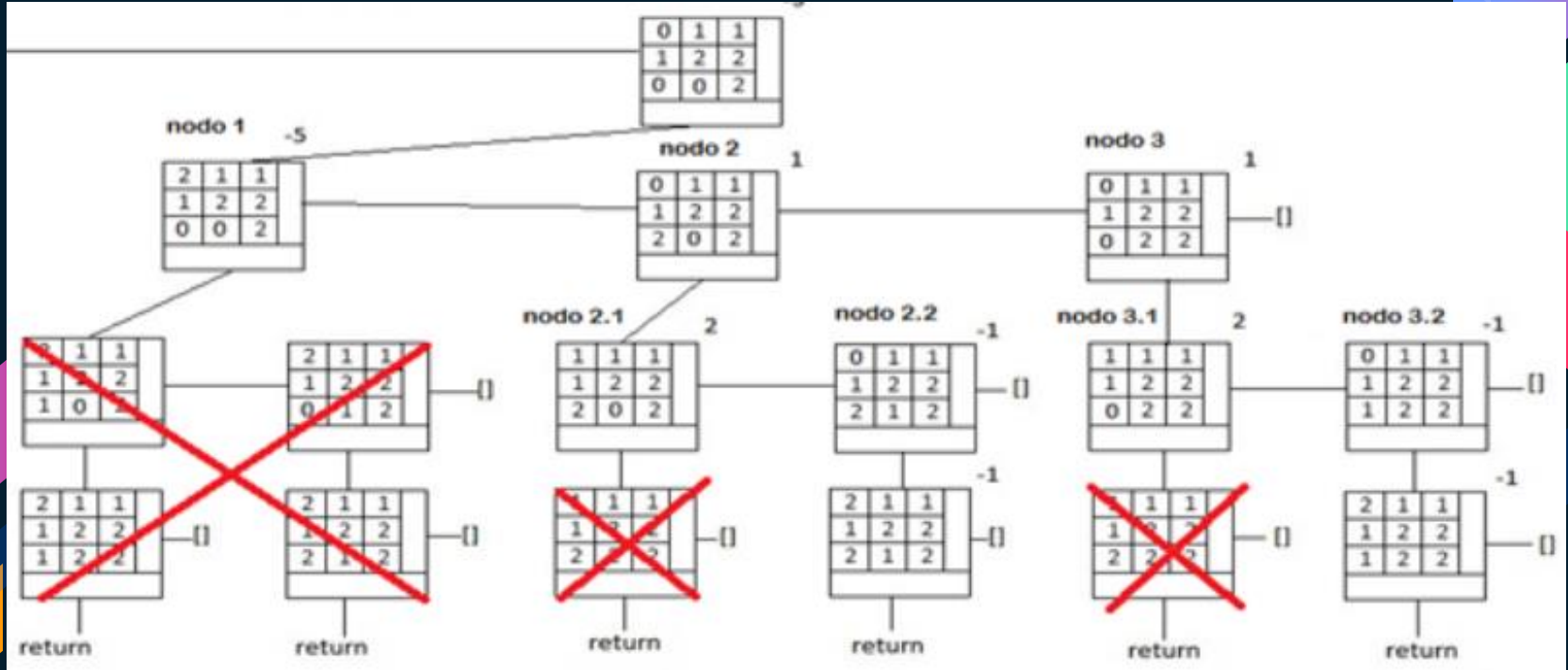
The subarray can be of any length, it could even be the whole array. The input will never be null.

**Example:**

**Input:** [-3.2, 4.2, 7, 5.4, -2.2, -2.5]

**Output:** [-3.2, 4.2, 7, 5.4, -2.2]

# 11. Decisions, but select the valid and the best among all decisions. Backtracking



# 11. Decisions, but select the valid and the best among all decisions. Backtracking

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



# 11. Decisions, but select the valid and the best among all decisions. Backtracking

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

## 12. The space and time tradeoff

[1, 3, 7, 9, 2]  $t = 11$

[1, 2, 3, 4, 5]  $t = 25$

[ ]  $t = 3$

[5]  $t = 8$

[5]  $t = 5$

[1, 6]  $t = 7$

```
nums = [1, 3, 7, 9, 2] target = 11
          ↓ ↓ ↓ ↓
          P1 P2 →

Const findTwoSum = function (nums, target) {
  for (let p1 = 0; p1 < nums.length; p1++) {
    const numberToFind = target - nums[p1];
    for (let p2 = p1 + 1; p2 < nums.length; p2++) {
      if (nums[p2] === numberToFind) {
        return [p1, p2];
      }
    }
  }
  return null;
}
```

$T: O(n^2)$   
 $S: O(1)$

+4  
+4  
3  
3  
3

$T: O(n^2) \downarrow$   
 $S: O(1) \uparrow$

## 12. The space and time tradeoff

```
nums = [1, 3, 7, 9, 2] target = 11
           ↓ ↓ ↓ ↓
           p1 p2 →

Const findTwoSum = function (nums, target) {
  for (let p1 = 0; p1 < nums.length; p1++) {
    const numberToFind = target - nums[p1];
    for (let p2 = p1 + 1; p2 < nums.length; p2++) {
      if (nums[p2] === numberToFind) {
        return [p1, p2];
      }
    }
  }
  return null;
}
```

$T: O(n^2)$   
 $S: O(1)$

+4  
+4

1st: calculate NTF  
2nd: checks current === NTF

## 12. The space and time tradeoff

[1, 3, 7, 9, 2] t = 11

```
    { number To Find: index }  
const findTwoSum = function (nums, target) {  
  const numsMap = { };  
  for (let p = 0; p < nums.length; p++) {  
    const currentMapVal = numsMap[nums[p]];  
    if (currentMapVal >= 0) {  
      return [currentMapVal, p];  
    } else {  
      const numberToFind = target - nums[p];  
      numsMap[numberToFind] = p;  
    }  
  }  
  return null;  
}
```

{ 10: 0,  
 8: 1,  
 4: 2,  
 ②: 3 }

2nd: checks current == NTF

time:  $O(n)$   
space:  $O(n)$