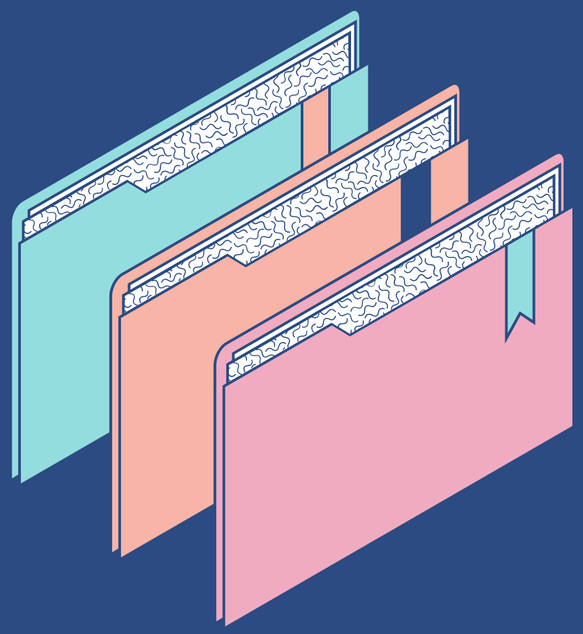
CSX3009 ALGORITHM DESIGN

Jump Game

By...
Tana J. (6211648) & Thanchanok W. (6211667)

Jump Game - description



You are given an integer array <nums>. You are initially positioned at the array's first index and each element in the array represents your maximum jump length at that position.

RETURN TRUE IF YOU CAN REACH THE LAST INDEX, OR FALSE OTHERWISE

Constraints:

- 1 <= nums.length <= 10^4
- 0 <= nums[i] <= 10^5

SOURCE : HTTPS://LEETCODE.COM/PROBLEMS/JUMP-GAME/

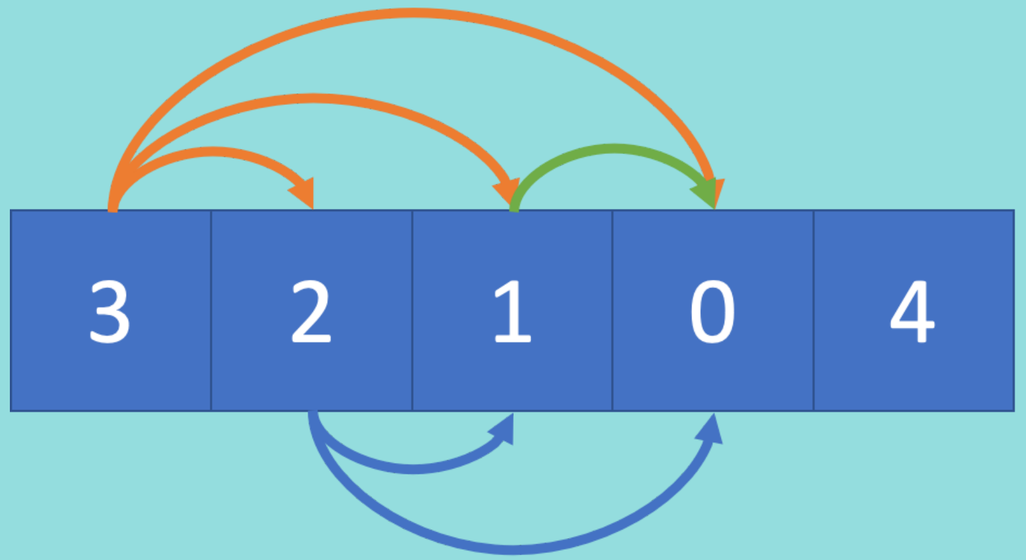
Jump Game - test case (1)

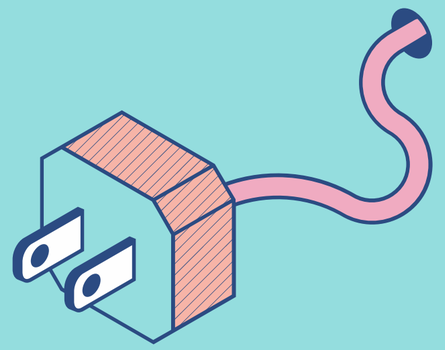
True

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Jump Game - test case (2)





False

Example 2:

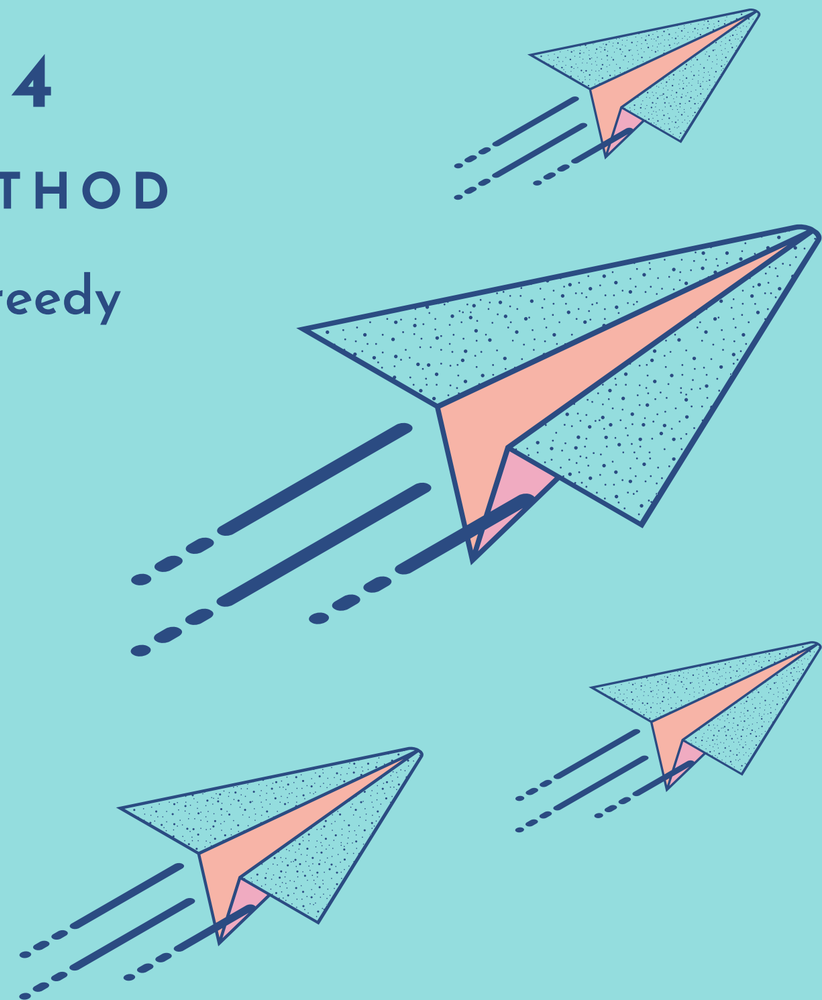
Input: nums = [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Jump Game - methods

1	2	3	4
METHOD	METHOD	METHOD	METHOD
Brute force	Memoization	Dynamic Programming	Greedy



Jump Game - Brute force

```python
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        if(len(nums)==0):
            return True
        jumpable = [False for i in range(len(nums))]
        jumpable[0] = True
        for i in range(len(nums)):
            if(jumpable[i] == False):
                return False
            reachable = min(len(nums),i + nums[i]+1)
            for j in range(i+1, reachable):
                jumpable[j] = True
        return True
```

TLE

This program will iterate all of number array and change the reachable index from current index in jumpable array to be True. Otherwise, false. But if the input is too large, we cannot get the output because of time limit.

Jump Game - Memoization

```
class Solution(object):
    def canJump(self, nums):
        memo = [None for i in range(len(nums))]
        return self.memoiz(0, nums, memo)

    def memoiz(self, i, nums, memo):
        if i >= len(nums)-1:
            return True
        elif memo[i] != None:
            return memo[i]
        else:
            memo[i] = False
            for j in range(1, nums[i]+1):
                if self.memoiz(j+i, nums, memo):
                    memo[i] = True
                    return True
            return memo[i]
```

TLE

Memoization will create the array to collect the result that is already calcuated. So, when calling the same index, the system don't need to recalcalculate all of the result. But, this program is still not enough because of recursion limit.

Jump Game - Dynamic Programming

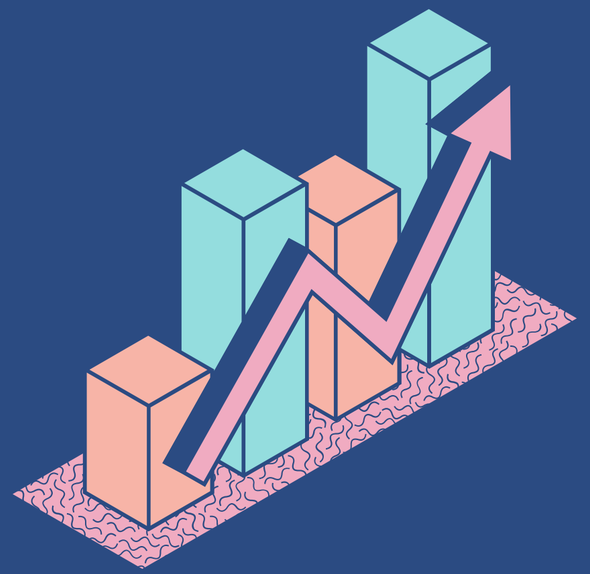
```python
class Solution(object):
    def __init__(self):
        self.dp = []

    def canJump(self, nums):
        self.dp = [False for i in range(len(nums))]
        self.dp[0] = True
        for i in range(len(nums)):
            for j in range(i-1,-1,-1):
                if(nums[j] + j >= i and self.dp[j] == True): #check reachable
                    self.dp[i] = True
                    break
        return self.dp[-1]
```

Success Details ›

Runtime: 2588 ms, faster than 15.38% of Python3 online submissions for Jump Game.

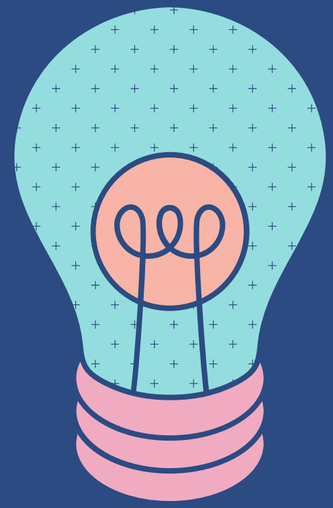
Memory Usage: 15.1 MB, less than 98.48% of Python3 online submissions for Jump Game.



Dynamic Programming which is developed from Bruteforce. This will instead iterate from the last index and check whether the last index is reachable or not. This program got success but time is too high so we try other method.

Jump Game - Greedy

```python
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        if(len(nums)==0):
            return True
        jumpable = 0
        for i in range(len(nums)):
            jumpable = max(jumpable,nums[i])
            if(jumpable == 0 and i != len(nums)-1):
                return False
            jumpable-=1
        return True
```



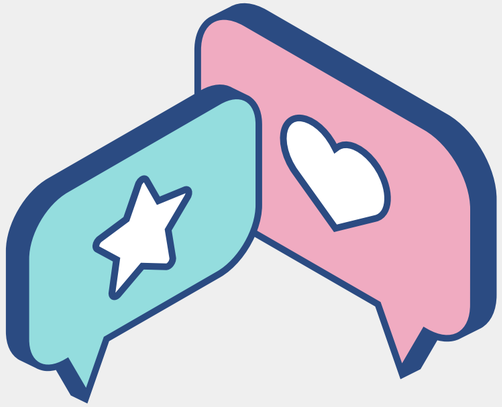
Success Details ›

Runtime: 512 ms, faster than 54.90% of Python3 online submissions for Jump Game.

Memory Usage: 15.3 MB, less than 35.32% of Python3 online submissions for Jump Game.

Time complexity : O(n)

Greedy algorithm will just iterate alll the numbers once. Each iteration, it will have jumpable variable to store whether there are jump steps left or not. If the existing steps is higher than the number in current index, it will skip those index and reduce the steps by one. But, it will change the number to current index if the number in current index is higher. If the steps that we can take is zero and cannot reach the last index, it will return false.



References

- https://leetcode.com/problems/jump-game/discuss/990250/Need-help-with-memoization
- https://leetcode.com/problems/jump-game/discuss/602869/Java-1ms-Solution-I-Greedy-%2B-DP-%2B-Memoized-Recursion