This is a Java class called "B0055" that contains a single method called "canJump". The method takes in a single argument, an array of integers called "nums", and returns a boolean value.

The method starts by initializing a variable called "lastPos" and setting it to the last index of the input array "nums". Then, it iterates through the array in reverse order using a for loop, starting at the second to last index and ending at the first index.

Inside the loop, the method checks if the current index plus its value is greater or equal to the "lastPos" variable. If this condition is true, it updates "lastPos" to be the current index. This process continues for each iteration of the loop, effectively tracking the last possible index that can be reached from any previous index.

Finally, after the loop has finished, the method checks if the final "lastPos" value is equal to the first index of the array. If it is, the method returns "true", indicating that it is possible to reach the last index of the array by jumping from index to index. If it is not, the method returns "false", indicating that it is not possible to reach the last index of the array in this way.

Explain to me this line

if (i + nums[i] >= lastPos) {

                // If true, update lastPos as the current index

                lastPos = i;

            }

The code is taken from :

public class B0055 {

    /\*\*

     \* Iterating in REVERSE order

     \* This method checks if it is possible to reach the last index of an input array by jumping from index to index.

     \* @param nums the input array of integers

     \* @return true if it is possible to reach the last index, false otherwise

     \*/

    public boolean **canJump**(int[] nums) {

        // Initialize lastPos as the last index of the input array

        int lastPos = nums.length - 1;

        // Iterate through the array in reverse order

        for (int i = nums.length - 2; i >= 0; i--) {

            // Check if the current index plus its value is greater or equal to the lastPos

            if (i + nums[i] >= lastPos) {

                // If true, update lastPos as the current index

                lastPos = i;

            }

        }

        // Check if the lastPos is the first index of the array

        return lastPos == 0;

    }

}

This line of code is checking if it is possible to reach the last index of the array starting from the current index. It is using a variable called "lastPos" which is initially set to the last index of the array. The for loop is iterating through the array in reverse order, starting from the second to last index.

For each index, the code is checking if the current index plus its value (which represents the maximum number of steps that can be taken from that index) is greater than or equal to the lastPos. If it is, then it means that it is possible to reach the last index from the current index, so it updates lastPos to be the current index.

The reason for iterating in reverse order is that it is checking if it is possible to reach the last index starting from each index and updating the lastPos in the way back.

At the end of the loop, the code is checking if the lastPos is the first index of the array. If it is, it means that it is possible to reach the last index starting from the first index, so the function returns true. If lastPos is not the first index, it means that it is not possible to reach the last index starting from the first index, so the function returns false.

Why not iterate in ascending order?

We can iterate in ascending order as well, the idea is to check for each index if it is possible to reach the last index from that index. However, when we iterate in ascending order, we need to keep track of the furthest index that we can reach from the current index, and check if the current index is less than or equal to the furthest index we can reach. If the current index is greater than the furthest index we can reach, it means that it's not possible to reach the last index starting from the current index, so we return false.

This approach would require a variable to keep track of the furthest index we can reach. And in each iteration, we will check if the current index plus the value of the current index is greater than the furthest index, if it is we update the furthest index with the current index plus the value of the current index.

In the reverse iteration approach, we don't need to keep track of the furthest index we can reach as we are iterating from the last index towards the start of the array. In each step, we check if it is possible to reach the last index from the current index, if it is we update the lastPos variable with the current index.

Both approaches have the same time and space complexity, and both are correct. The main difference is in the way we keep track of the information.

Give me the main difference between iterating the array in ascending and reverse order in points so I can understand it better

* When iterating the array in ascending order, we keep track of the furthest index that can be reached from the current index. We check if the current index is less than or equal to the furthest index, if it is, we update the furthest index with the maximum of the current furthest and the current index plus its value.
* When iterating the array in reverse order, we keep track of the last position that can be reached from the current index. We check if the current index plus its value is greater than or equal to the last position, if it is, we update the last position with the current index.
* When iterating the array in ascending order, we check if the furthest index is greater than or equal to the last index of the array, if it is, we return true. If the loop completes, we return false.
* When iterating the array in reverse order, we check if the last position is the first index of the array, if it is, we return true. If the loop completes, we return false.

Both approaches have the same time and space complexity, and both are correct. The main difference is in the way we keep track of the information.

The ascending order approach needs to keep track of the furthest index that can be reached,

while the reverse order approach needs to keep track of the last position that can be reached.

Why we need to check if the current index plus its value is greater than or equal to the last position, for reverse order?

When iterating the array in reverse order, we start from the last index and move towards the first index. We check if it is possible to reach the last index from the current index. If the current index plus its value (which represents the maximum number of steps that can be taken from that index) is greater than or equal to the last position, it means that it is possible to reach the last index from the current index.

For example, let's say we have the input array [2,3,1,1,4] and we are currently at index 2. The value of the current index is 1. If we add the value of the current index to the current index, we get 3 which is greater than the last position which is 4 (the last index of the array). So it means that we can reach the last index starting from the current index by taking one step.

Therefore, by checking if the current index plus its value is greater than or equal to the last position, we are checking if it is possible to reach the last index from the current index.

Additionally, by updating the last position with the current index, we are updating the last position as the furthest index that we can reach from the current index. This way as we move towards the start of the array, we know the furthest position that can be reached, hence if we reach the start of the array and the last position is the first index of the array, we know that we can reach the last index starting from the first index, otherwise we can't.

Why 3 which is greater than the last position which is 4 (the last index of the array)?

I apologize for the confusion. I made a mistake in my previous explanation. The last position should be the index that can be reached from the current index, not the last index of the array.

When iterating in reverse order, we are keeping track of the last position that can be reached from the current index. The last position is initialized as the last index of the array.

For example, let's say we have the input array [2,3,1,1,4] and we are currently at index 2. The value of the current index is 1. If we add the value of the current index to the current index, we get 3 which is the index that can be reached from the current index. If we compare the current index plus its value with the last position, we see that the current index plus its value is greater than the last position which is 2. So it means that we can reach index 3 starting from index 2 by taking one step, and we update the last position as the current index plus its value, which is 3 in this case.

Therefore, by checking if the current index plus its value is greater than or equal to the last position, we are checking if it is possible to reach a further index from the current index.

By updating the last position with the current index plus its value, we are updating the furthest position that can be reached from the current index. This way as we move towards the start of the array, we know the furthest position that can be reached, hence if we reach the start of the array and the last position is the first index of the array, we know that we can reach the last index starting from the first index, otherwise we can't.

I kind of confuse, the solutions needs you to keep track of the last position that can be reached. But if and array java a 0 in it, it’s mean the solution output is always false. Why not just checking if an array have zero in it and if it does, it will return false instead of checking last position that can be reached

You are correct that if there is a zero in the input array, it means that it is not possible to reach the last index starting from the first index, because you can't move forward from a section where the number is zero.

However, checking for a zero in the input array is not enough to determine if it is possible to reach the last index. The input array may have multiple zeroes or even non-zero values that are too small to reach the next non-zero value, in these cases, we also need to check if it is possible to reach the last index starting from the first index.

That's why the solution is keeping track of the last position that can be reached, by iterating through the array in reverse order. By doing so, the solution can check if it is possible to reach the last index starting from each index and updating the last position in the way back.

For example, let's say we have the input array [2,3,1,0,4], if we just check for zeroes in the array, we will return false since there is a zero in the array, but in this case, it is possible to reach the last index starting from the first index by taking 2 steps, then 3 steps, then 1 step.

Therefore, checking for zeroes in the array is not enough, the solution needs to keep track of the last position that can be reached and check if it is possible to reach the last index starting from each index.

The real answer !

In the question, it's written "each element in the array represents your maximum jump length at that position."

It means if we are at a position k, and arr[k] = 5, then it means we can jump forward a maximum of 5 steps from this position. It's our choice to jump 1,2,3,4 or 5 positions, but not more than 5.

Similarly in this test case [2,5,0,0]  
arr[0] = 2  
means we can jump either 1 step or 2  
You already stated what would happen on jumping 2 steps so let's jump 1 step and now we are at arr[1], whose value is 5. So we can jump a maximum of 5 steps and thus we can reach the end of array and this is why the result is true.

Hope this helps :)