Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then we cannot move through that element. If we can't reach the end, return -1.

<https://www.geeksforgeeks.org/minimum-number-jumps-reach-endset-2on-solution/> =o(n)

<https://www.geeksforgeeks.org/minimum-number-of-jumps-to-reach-end-of-a-given-array/#comment-3321908697> =o(n2)

**Examples:**

**Input:**  arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}

**Output:** 3 (1-> 3 -> 8 -> 9)

**Explanation:** Jump from 1st element to

2nd element as there is only 1 step,

now there are three options 5, 8 or 9.

If 8 or 9 is chosen then the end node 9

can be reached. So 3 jumps are made.

**Input :** arr[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

**Output :** 10

Explanation: In every step a jump is

needed so the count of jumps is 10

class Test {

static int minJumps(int arr[])

{

if (arr.length <= 1)

return 0;

if (arr[0] == 0)

return -1;

int maxReach = arr[0];

int step = arr[0];

int jump = 1;

for (int i = 1; i < arr.length; i++) {

if (i == arr.length - 1)

return jump;

maxReach = Math.max(maxReach, i + arr[i]);

step--;

if (step == 0) {

// we must have used a jump

jump++;

if (i >= maxReach)

return -1;

step = maxReach - i;

}

}

return -1;

}

public static void main(String[] args)

{

int arr[] = new int[] { 1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9 };

System.out.println(minJumps(arr));

}

}**Implementation:** Variables to be used:

1. **maxReach** The variable maxReach stores at all times the maximal reachable index in the array.
2. **jump** stores the amount of jumps necessary to reach the maximal reachable position. It also indicates the **current jump we are making in the array**.
3. **step** The variable step stores the number of steps we can still take **in the current jump ‘jump’** (and is initialized with value at index 0, i.e. initial number of steps)

Given array arr = 1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9

* **maxReach** = arr[0]; // arr[0] = 1, so the maximum index we can reach at the moment is 1.   
  **step** = arr[0]; // arr[0] = 1, the amount of steps we can still take is also 1.   
  **jump** = 1; // we are currently making our first jump.

Now, starting iteration from index 1, the above values are updated as follows:

* First, we test whether we have reached the end of the array, in that case, we just need to return the jump variable.

if (i == arr.length - 1)

return jump;

* Next we update the maxReach. This is equal to the maximum of maxReach and i+arr[i](the number of steps we can take from the current position).

maxReach = Math.max(maxReach, i+arr[i]);

* We used up a step to get to the current index, so steps has to be decreased.

step--;

* If no more steps are remaining (i.e. steps=0, then we must have used a jump. Therefore increase jump. Since we know that it is possible somehow to reach maxReach, we again initialize the steps to the number of steps to reach maxReach from position i. But before re-initializing step, we also check whether a step is becoming zero or negative. In this case, It is not possible to reach further.

if (step == 0) {

jump++;

if(i>=maxReach)

return -1;

step = maxReach - i;

}

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

|  |
| --- |
| // Java program to count Minimum number  // of jumps to reach end    class Test {      static int minJumps(int arr[])      {          if (arr.length <= 1)              return 0;            // Return -1 if not possible to jump          if (arr[0] == 0)              return -1;            // initialization          int maxReach = arr[0];          int step = arr[0];          int jump = 1;            // Start traversing array          for (int i = 1; i < arr.length; i++) {              // Check if we have reached  // the end of the array              if (i == arr.length - 1)                  return jump;                // updating maxReach              maxReach = Math.max(maxReach, i + arr[i]);                // we use a step to get to the current index              step--;                // If no further steps left              if (step == 0) {                  // we must have used a jump                  jump++;                    // Check if the current  // index/position or lesser index                  // is the maximum reach point  // from the previous indexes                  if (i >= maxReach)                      return -1;                    // re-initialize the steps to the amount                  // of steps to reach maxReach from position i.                  step = maxReach - i;              }          }            return -1;      }        // Driver method to test the above function      public static void main(String[] args)      {          int arr[] = new int[] { 1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9 };            // calling minJumps method          System.out.println(minJumps(arr));      }  } |

**Output**

3

**Complexity Analysis:**

* **Time complexity:** **O(n),**Only one traversal of the array is needed.
* **Auxiliary Space:** **O(1),**There is no space required.