Find Maximum in Sliding Window

Given an array of integers, find the maximum value in a window.

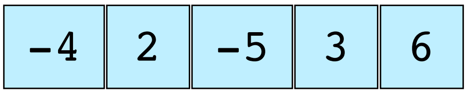
We'll cover the following ^

- Description
- Try it yourself
- Solution
  - Runtime Complexity
  - Memory Complexity

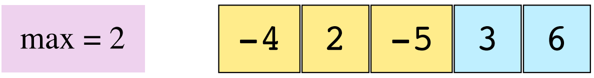
Description

Given a large array of integers and a window of size w, find the current maximum value in the window as the window slides through the entire array.

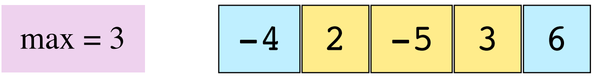
Let's try to find all maximums for a window size equal to 3 in the array given below:



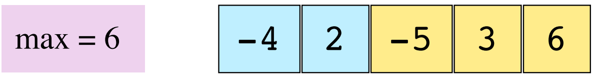
Step 1: For the first 3 elements in the window, max is 2.



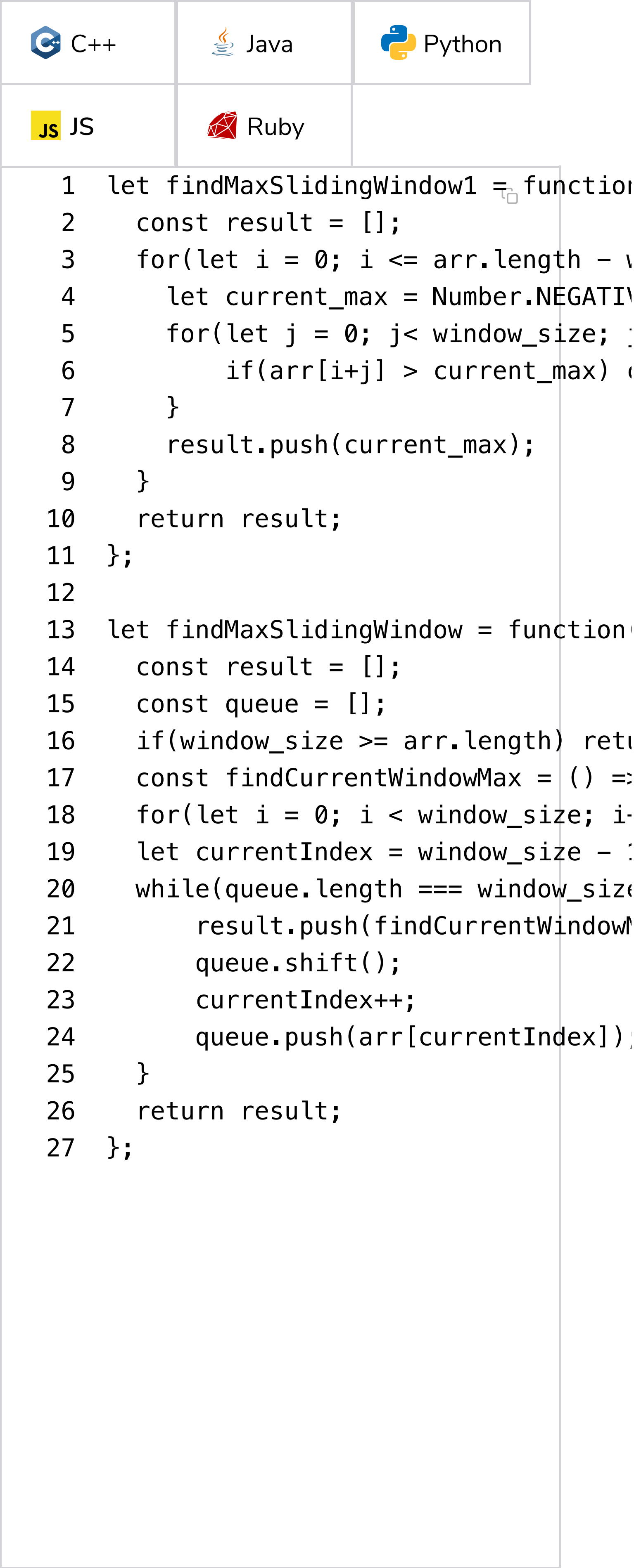
Step 2: Slide window one position to the right and max for window becomes 3.



Step 3: In the last window, max is 6.



Try it yourself



```
let findMaxSlidingWindow1 = functio
  const result = [];
  for(let i = 0; i <= arr.length - 
    let current_max = Number.NEGATI
    for(let j = 0; j< window_size; 
        if(arr[i+j] > current_max) 
    }
    result.push(current_max);
  }
  return result;
};

let findMaxSlidingWindow = function
  const result = [];
  const queue = [];
  if(window_size >= arr.length) ret
  const findCurrentWindowMax = () =
  for(let i = 0; i < window_size; i
  let currentIndex = window_size - 
  while(queue.length === window_siz
      result.push(findCurrentWindow
      queue.shift();
      currentIndex++;
      queue.push(arr[currentIndex])
  }
  return result;
};
```

Test Save Reset

Show Results Show Console

1.9s

3 of 3 Tests Passed

Result	Input
✓	`findMaxSlidingWindow([1,`
✓	`findMaxSlidingWindow([1,`
✓	`findMaxSlidingWindow([1,`

Solution

Runtime Complexity

The runtime complexity of this solution is *linear*, $O(n)$.

> Every element is pushed and popped from the deque only once in a single traversal.

educative(/learn)

Back To Module Home

Data Structures

(/module/data-structures-in-javascript)

0% completed

Search Module

Prelude

Arrays

Memory Complexity

The memory complexity of this solution is *linear*, $O(w)$, where w is the window size in this case.

The algorithm uses the **deque** data structure to find the maximum in a window. A deque is a double-ended queue (https://www.educative.io/edpresso/what-is-a-queue) in which push and pop operations work in $O(1)$ at both ends. It will act as our window.

At the start of the algorithm, we search for the maximum value in the first window. The first element's index is pushed to the front of the deque.

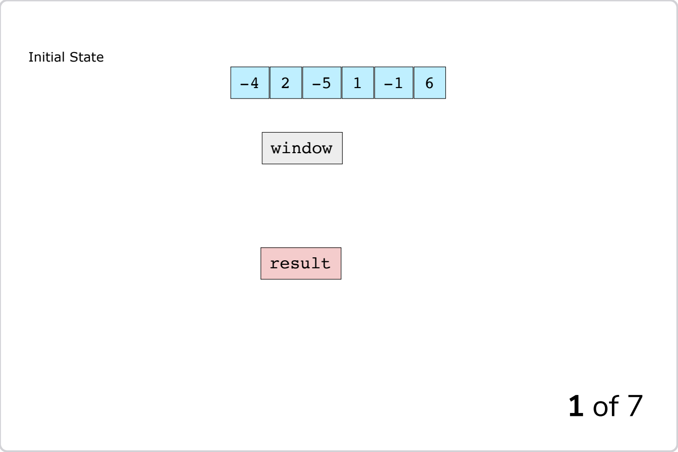
If an element is smaller than the one at the back of the queue, then the index of this element is pushed in and becomes the new back. If the current element is larger, the back of the queue is popped repeatedly until we can find a higher value, and then we'll push the index of the current element in as the new back.

As we can see, the deque stores elements in decreasing order. The front of the deque contains the index for the maximum value in that particular window.

We will repeat the following steps each time our window moves to the right:

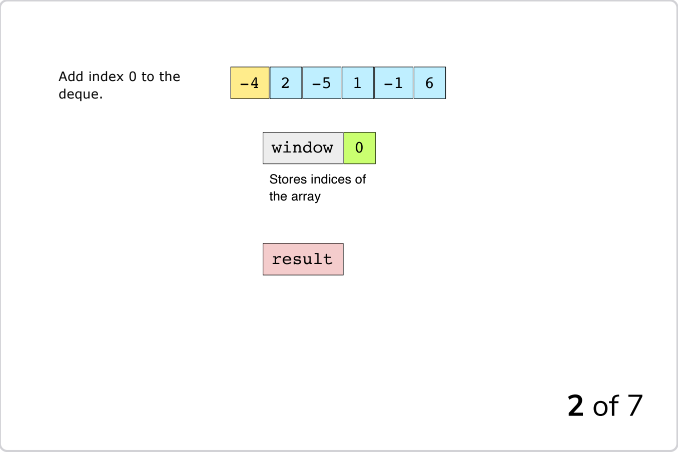
- Remove the indices of all elements from the back of the deque, which are smaller than or equal to the current element.
- If the element no longer falls in the current window, remove the index of the element from the front.
- Push the current element index at the back of the window.
- The index of the current maximum element is at the front.

Let's run an example with window size $= 3$ on the array below:



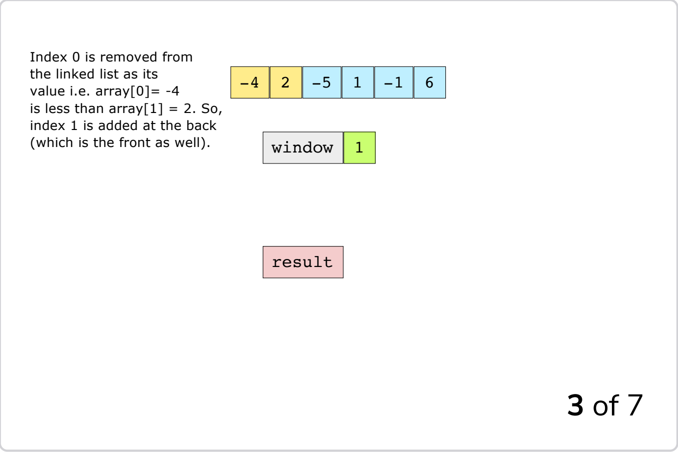
Initial State

1 of 7



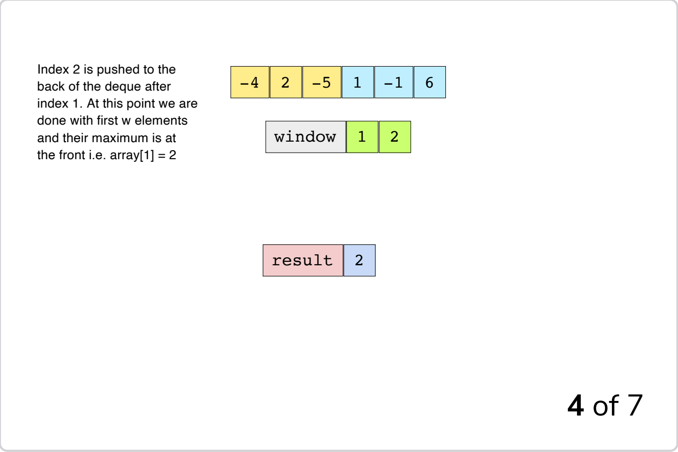
Add index 0 to the deque.

2 of 7



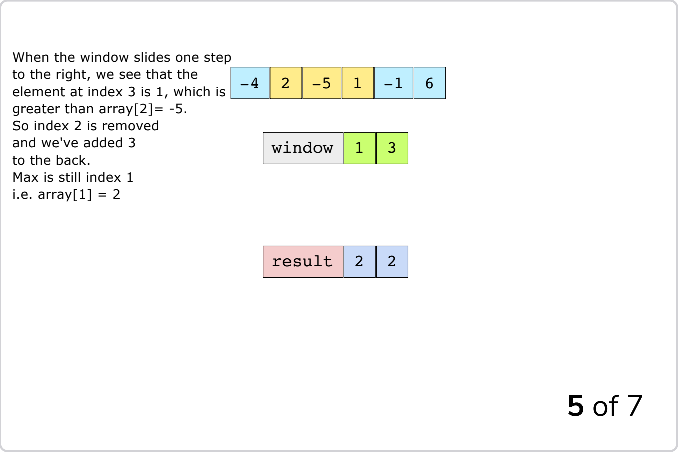
Index 0 is removed from the linked list as its value i.e. array[0]= -4 is less than array[1] = 2. So, index 1 is added at the back (which is the front as well).

3 of 7



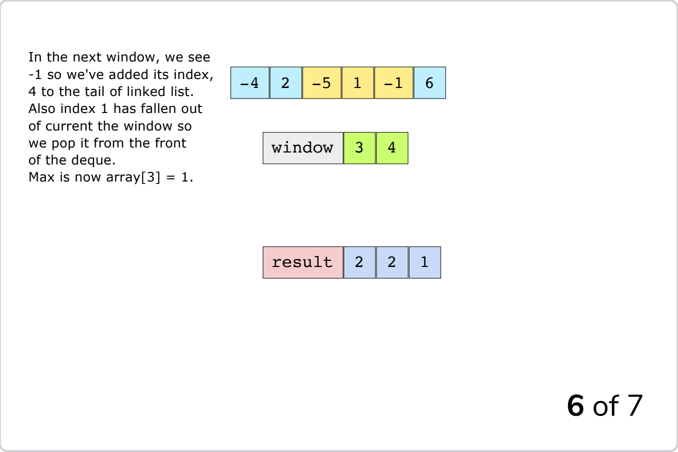
Index 2 is pushed to the back of the deque after index 1. At this point we are done with first w elements and their maximum is at the front i.e. array[1] = 2

4 of 7



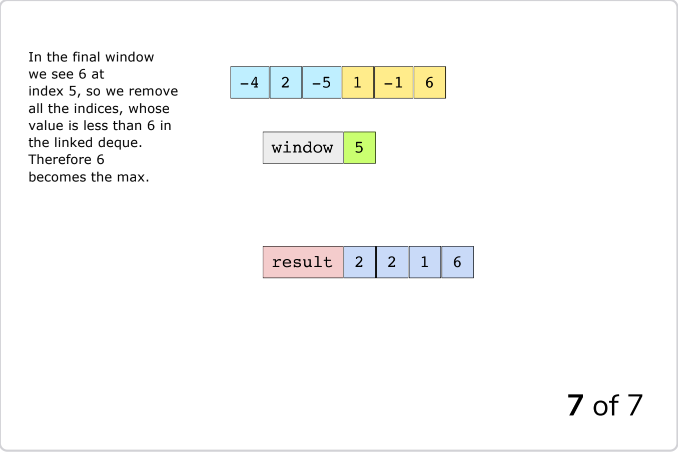
When the window slides one step to the right, we see that the element at index 3 is 1, which is greater than array[2]= -5.
So index 2 is removed and we've added 3 to the back.
Max is still index 1 i.e. array[1] = 2

5 of 7



In the next window, we see -1 so we've added its index, 4 to the tail of linked list.
Also index 1 has fallen out of current the window so we pop it from the front of the deque.
Max is now array[3] = 1.

6 of 7

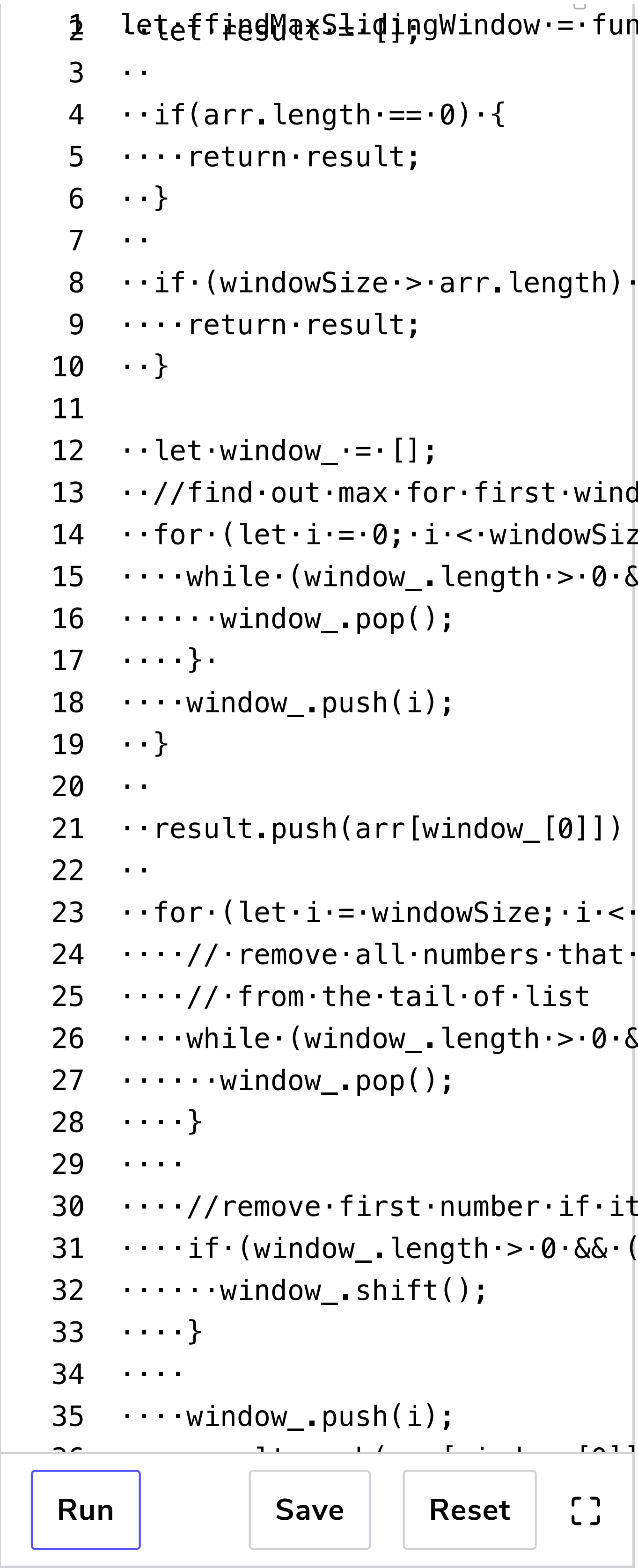


In the final window we see 6 at index 5, so we remove all the indices, whose value is less than 6 in the linked deque.
Therefore 6 becomes the max.

7 of 7

Let's see the code for this algorithm:

C++	Java	Python
JS	Ruby	



```
let findMaxSlidingWindow = function
  let result = [];
  
  if(arr.length == 0) {
    return result;
  }
  
  if (windowSize > arr.length) {
    return result;
  }

  let window_ = [];
  //find out max for first window
  for (let i = 0; i < windowSize; i
    while (window_.length > 0 && ar
      window_.pop();
    } 
    window_.push(i);
  }
  
  result.push(arr[window_[0]])
  
  for (let i = windowSize; i < arr.
    // remove all numbers that are 
    // from the tail of list
    while (window_.length > 0 && ar
      window_.pop();
    }
    
    //remove first number if it doe
    if (window_.length > 0 && (windo
      window_.shift();
    }
    
    window_.push(i);
```

> An alternative approach to this problem is to use heaps (https://www.educative.io/edpresso/what-is-a-heap) for searching the maximum in every window. In that case, the time complexity will rise to $O(n \ log(w))$.

← Back

About this Module

Next →

Search a Rotated Array

☑ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/find-maximum-in-sliding-window__arrays__coderust-hacking-the-coding-interview)