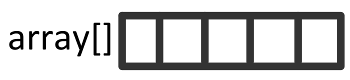
Dynamic array

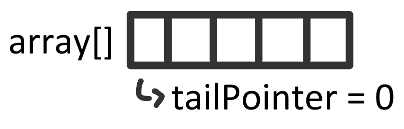
Friday, April 29, 2022 12:58 PM

Dynamic array - Data structure & algorithms



Initial length/size = 5 Initial tailPointer = -1

Initial state (initialized)

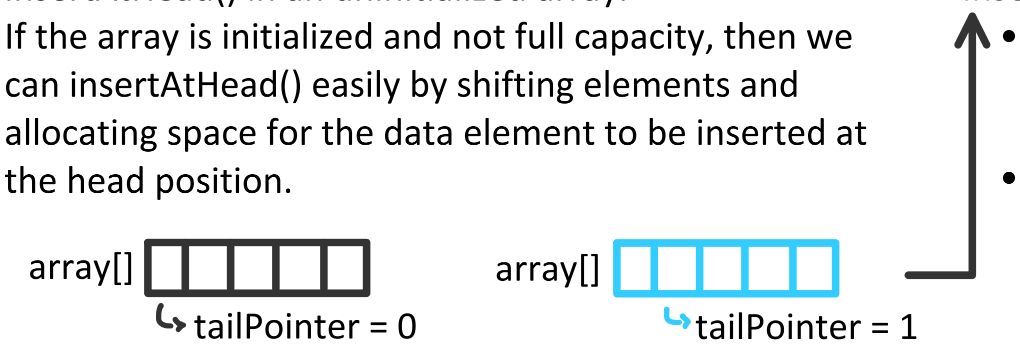


Time complexity:

- **InsertAtHead() -** O(n) linear-time
- **RemoveFromHead() -** O(n) linear-time
- **InsertAtTail() -** O(n) linear-time
- **RemoveFromTail() -** O(n) linear-time

insertAtHead(int data)

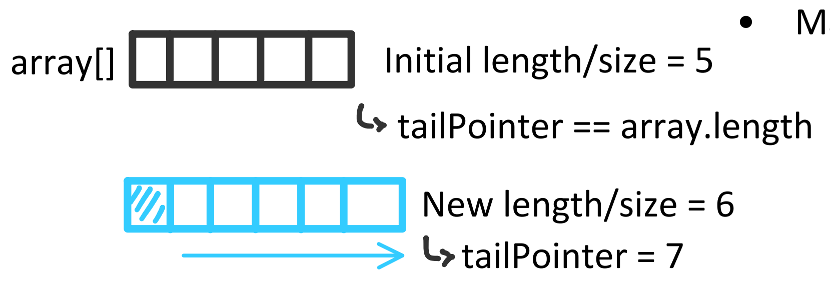
- Check if the array is uninitialized. We cannot insertAtHead() in an uninitialized array.
- If the array is initialized and not full capacity, then we can insertAtHead() easily by shifting elements and allocating space for the data element to be inserted at the head position.



Inserted one element…

- Because tailPointer == 0 after successful initialization, we can insertAtHead(data).
- We can then increment tailPointer++.

- If tailPointer == array.length, then the array has full capacity and will need to be resized to accommodate another insertion.

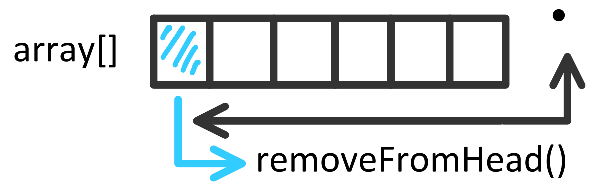


Copy over elements and insertAtHead(data)

- Maximum capacity reached…
  - We need to increase the length/size of the array.
  - Copy over existing elements from i = 1…n-1.
  - array[0] = data
  - Increment tailPointer++

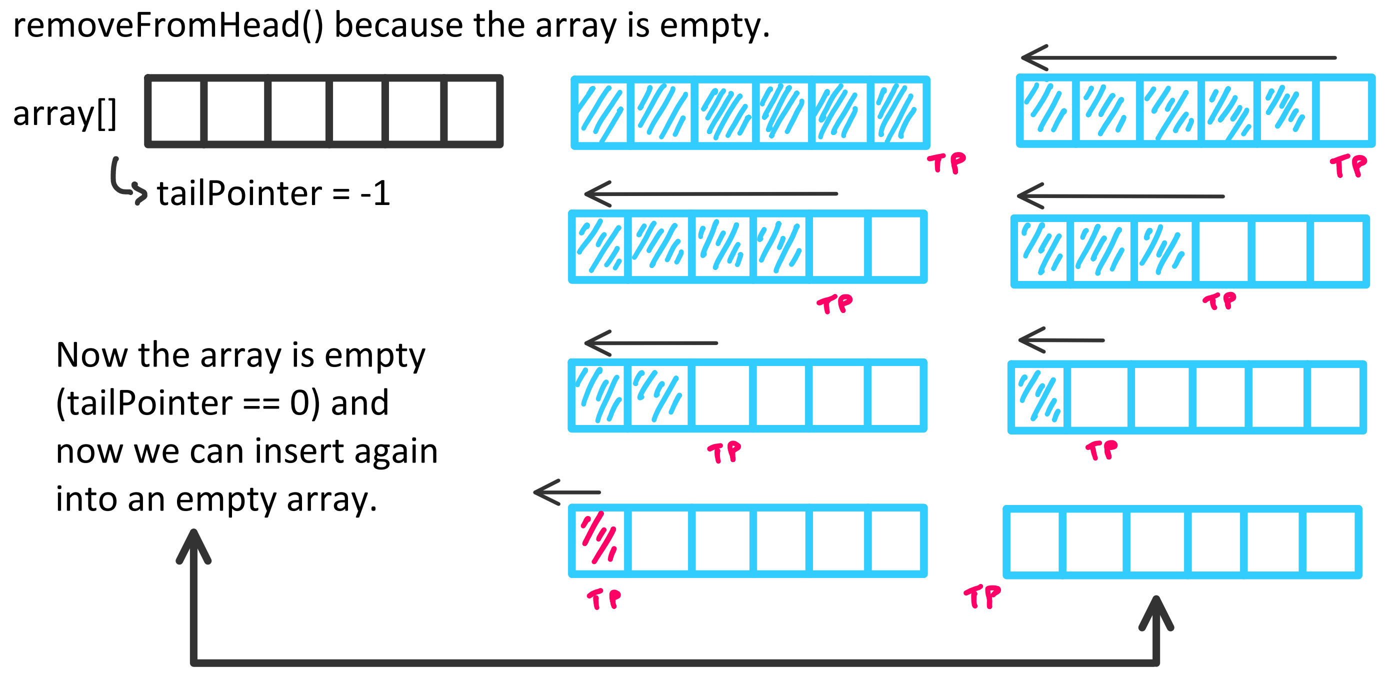
removeFromHead()

- Check if the array is uninitialized. We cannot removeFromHead() from an uninitialized array.
- Now we can successfully removeFromHead()…
  - Shift all elements one space to the left to "remove" the head element (or overwrite it).
  - Decrement tailPointer—



- Shift all elements to the left to overwrite or "delete" the element at the head position in array.
  - Decrement tailPointer— to keep correct track of number of total elements (capacity) and keep O(1) insertion/deletion operations at tail.

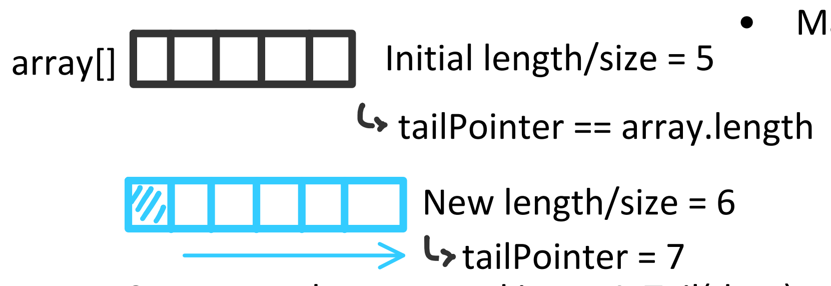
- If tailPointer == -1, then we cannot removeFromHead() because the array is empty.



Now the array is empty (tailPointer == 0) and now we can insert again into an empty array.

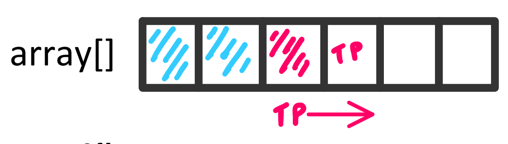
insertAtTail(int data)

- Check if the array is uninitialized. We cannot insertAtTail(data) in an uninitialized array.
- Now we need to check if there is enough space (if capacity full) to insert a new element.
  - Check if tailPointer != array.length —> Then we can insertAtTail(data).
  - If tailPointer == array.length —> Then we need to resize length/size of array.



Copy over elements and insertAtTail(data)

- Maximum capacity reached…
  - We need to increase the length/size of the array.
  - Copy over existing elements from i = 0…n-1.
  - array[tailPointer] = data
  - Increment tailPointer++

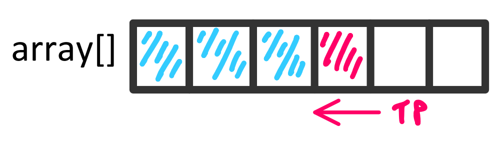


array[tailPointer] = data
tailPointer++

removeFromTail()

- Check if the array is uninitialized. We cannot removeFromTail() from an uninitialized array.
- Now we need to check if the array is empty because we cannot remove any element from an already empty array.
  - Check if tailPointer == -1 —> throw ERROR for removing from empty array.
  - If array is not empty, then set array[tailPointer - 1] = 0. Setting or nulling out the element at the to 0 and decrement tailPointer—.

- **Shifting right*****
  - If we have one element in our array, we will simply swap the first two elements (initial shift).
  - Otherwise, perform shifting algorithm.
- **Shifting left*****
  - Shift elements to the left & set array[tailPointer] = 0.



array[tailPointer - 1] = 0
tailPointer—