Algorithms & Data Structures I Week 7 Lecture Note

Notebook:	Algorithms & Data Structures I		
Created:	2020-10-21 4:14 PM	**Updated:**	2020-11-01 9:47 AM
Author:	SUKHJIT MANN		

Cornell Notes	**Topic:** Data Structures and Searching, part 1	Course: BSc Computer Science
		Class: CM1035 Algorithms & Data Structures I [Lecture]
		Date: October 30, 2020

Essential Question:

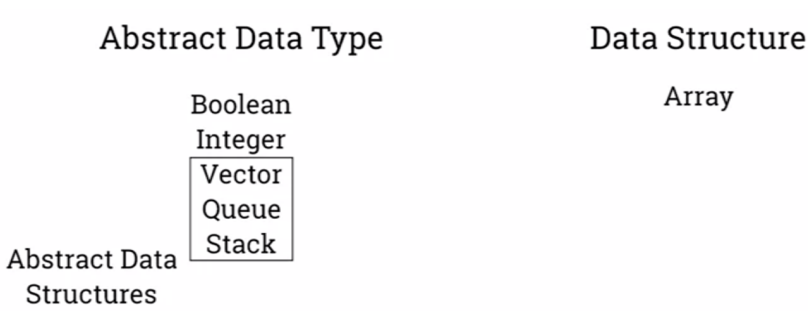
What are arrays, dynamic arrays, and the linear search algorithm?

Questions/Cues:

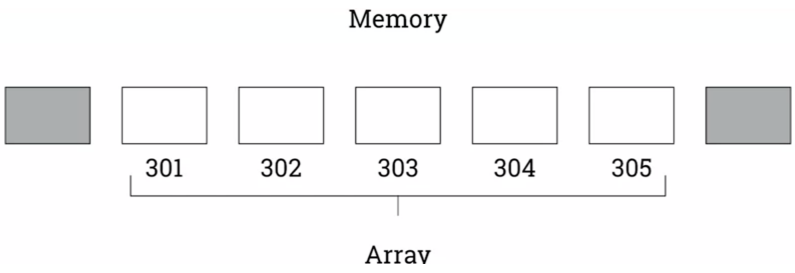
- What is searching?
- What is an abstract data type?
- What is a data structure?
- What is an array?
- What can do we do with an array?
- How can a vector be implemented by an array?
- What is a Dynamic Array?
- What operations can we perform on a dynamic array?
- How do we implement a dynamic array?
- What do vectors, queues, stacks and dynamic arrays all have in common as abstract data structures?
- What is the linear search algorithm?
- What does the flowchart for the linear search algorithm look like?
- What is the pseudocode for the linear search algorithm returning the index of the element where x is supposedly stored?
- How can the linear search algorithm be perform on an array implementation of a vector?
- How do we search stacks or queue without destroying the elements and/or data?

Notes

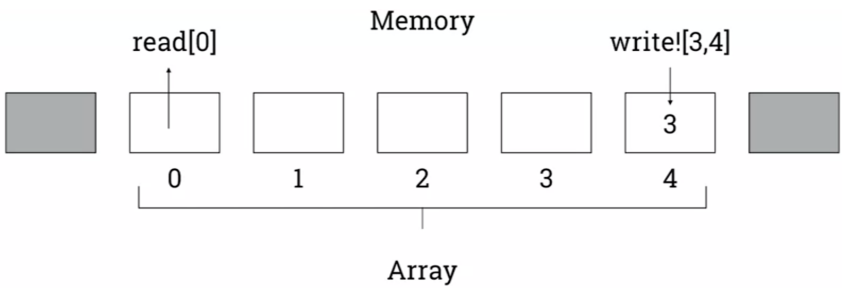
- Searching = the central question which is given a collection of data, can we find a desired value among all the pieces of data?
- Abstract data type = consists of the kind of data we have, the values the data can take and the allowed operations on this data
- Data Structure = the more concrete way in which many pieces of data are stored, managed and manipulated by the computer



- Array = Generally speaking:
  - A computer will some amount of space for storing data ie. RAM which is used by the CPU
  - Now we want to store a number within this RAM
    - Firstly, we need to know how much memory to set aside and secondly where to put it
  - The amount of memory needed depends on the kind of data
    - A number might require more memory than a Boolean
  - In terms where to put data, we need a reference for the location, so that if we want to retrieve it later we know exactly where to look
  - An array is the method of organizing multiple pieces of data. Typically, we assume that each piece of data is of the same type. This is true most programming languages expect JavaScript
    - Each piece of data in the array is assigned a location with an address in memory. The addresses can essentially be viewed as numbers, and the locations with their numbering will be consecutive. In this way, we can view an array as a line indexed by integers.
      - This number will be the reference for our computer to look up elements in our array and find the values contained therein
  - Since we are using JavaScript as our language of choice it is important to note that within JS the locations of elements in arrays are numbered from zero and increase as integers, so 0, 1,2, 3, 4...
    - This location number will be referred to as the index and the index of arrays will start from 0 onwards

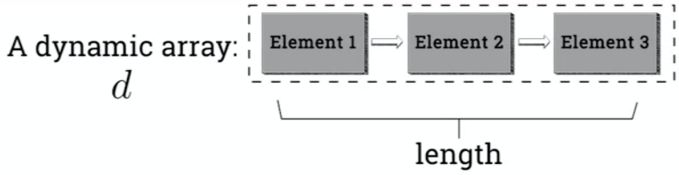


- Array (Doing) = in a simple array, we fix the number of elements allowed in the array. In this case, let's refer to the number of elements as the size of the array. So it can be said that the size of a simple array can't be altered, but we can always create new arrays.
  - This useful in the context of a computer because for a computer to assign memory since it knows how many memory locations to set aside for the array. A computer, once it has created an array, it can read the values contained in the elements, and it can write and overwrite values to the elements



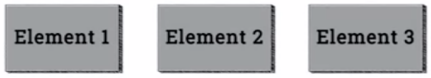
- Vector (array implementation) = The structure of the vector is a linear sequence of data with numbers indexing the elements of the vector. The first operation of the vector is the length operation, which just reveals the length of the vector. In terms of an array implementation, the simplest implementation is for

the array to store the length of the vector in one of its elements, and when we wish to implement the length operation, a computer just reveals the number in that element.
    - Depending on the implementation, this element could be stored in a privileged element of the array. So, to store a vector of length n, we would need an array of length n + 1
    - To implement the select[k] and store![o,k] operations in an array, we just use the read and write operations associated with our array data structure
- Dynamic Array = is a finite sequential collection of data, just like the vector, but is not fixed sized like the vector.
    - Much like the vector, we can have empty dynamic arrays
    - Because it's like the vector, the dynamic array has all the operations that a vector has, but not vice versa
    - It is possible that the length of the dynamic array can change, so the length operation could return different values at different points while it's being utilized



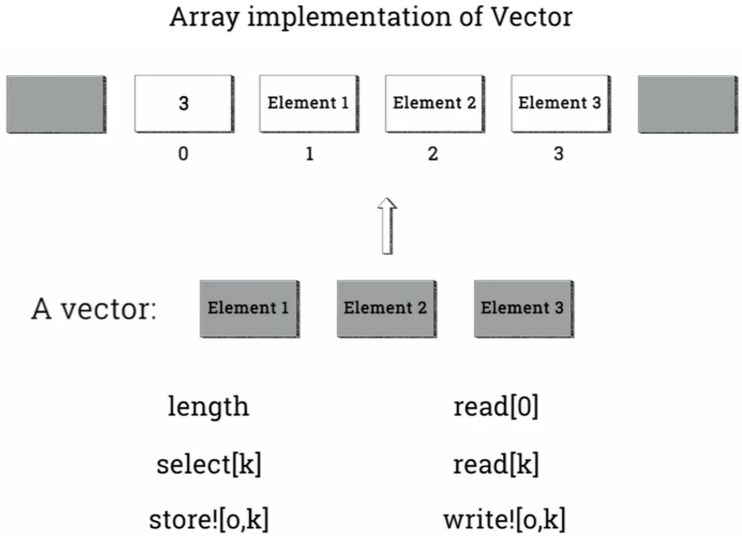
Operation	Pseudocode
length	$\text{LENGTH}[d]$
select[k]	$d[k]$
store![o,k]	$d[k] \leftarrow o$

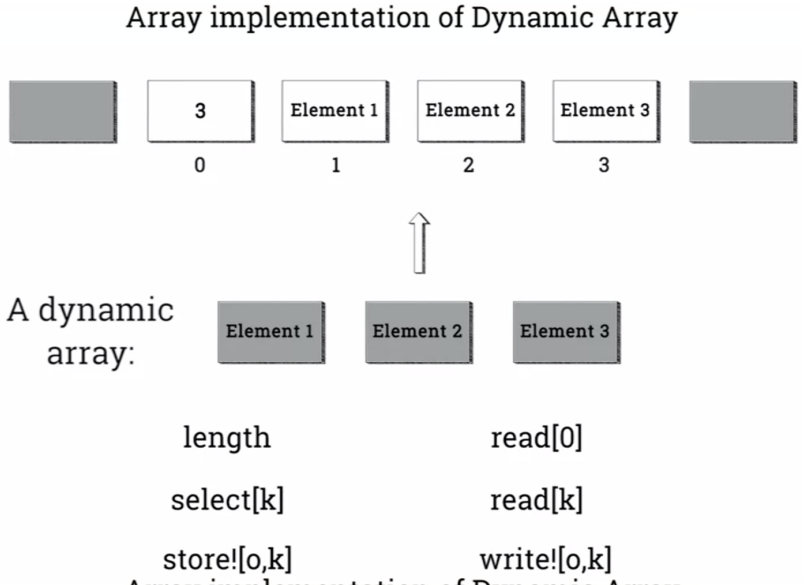
- Operation (Dynamic Array) = In addition to the operations much like the vector, the dynamic array has two new operations and they are the following:
    - removeAt![k] operation eliminates element k of the dynamic array, and shifts everything to the right of k to the left. So what had the index k + 1 before the operation now has index k. In addition to removing element k, the operation will return the value of that element.
        - Note that k as an integer must not be larger than the length of the dynamic array, otherwise an error must be returned
        - Recall that the index of the rightmost element will be the length of the array
    - insertAt![o,k] operation will put a new element at index k, where k must be less than or equal to the length of the dynamic array plus one
        - If k is less than or equal to the length, the insertAt![o,k] operation will shift everything at k onwards one place to the right, and in that element the value o will be stored.
        - if k is equal to the length plus one, then a new element is created at the end of the dynamic array, and it's assigned the value o

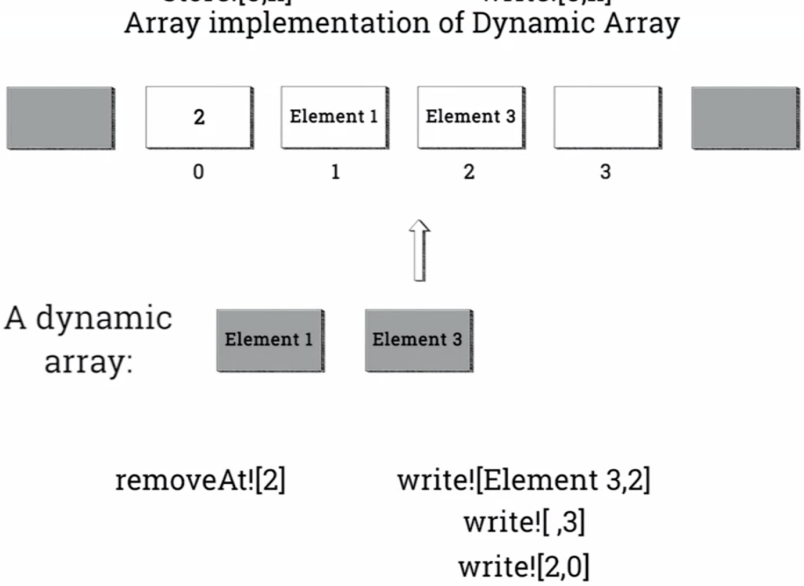


Operation	Pseudocode
length	$\text{LENGTH}[d]$
select[k]	$d[k]$
store![o,k]	$d[k] \leftarrow o$
removeAt![k]	$d[k] \leftarrow \emptyset$ $k \leq \text{LENGTH}[d]$
insertAt![o,k]	$d[k] \hookleftarrow o$ $k \leq \text{LENGTH}[d] + 1$

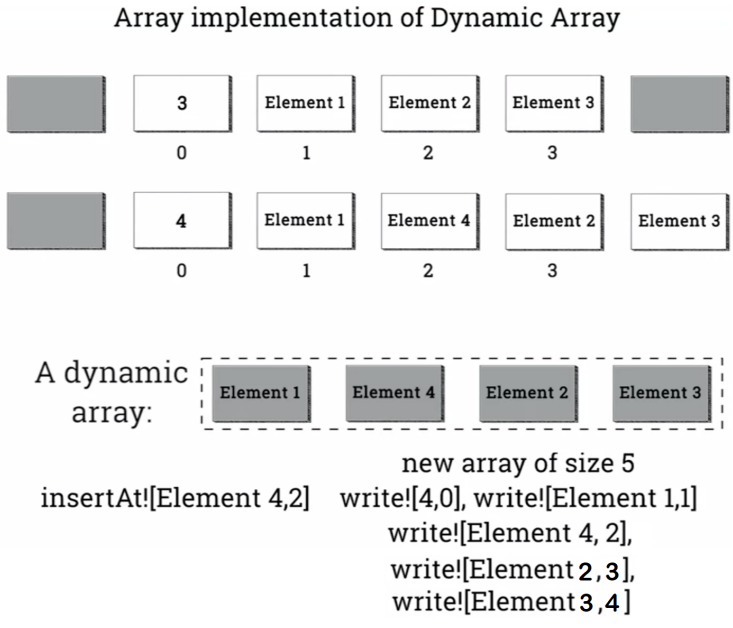
- Sometimes we see simpler definitions of dynamic arrays, that include a push and pop operation as you would have with the stack. What these operations would do is to add an element at the end of the array and remove the element at the end of the array respectively. Naturally, we can imitate these operations with insertAt![o, length + 1] and removeAt![length].
- We implement a dynamic array abstract data structure with an array
- Anytime we want to implement a dynamic array getting larger, we create a new array with more elements, copy elements for the old array that implemented the entries from the dynamic array, and then add the new elements





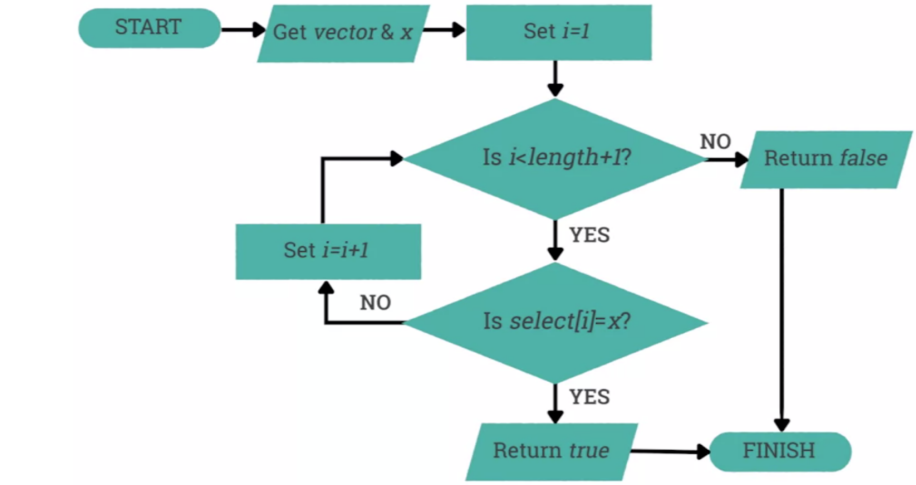


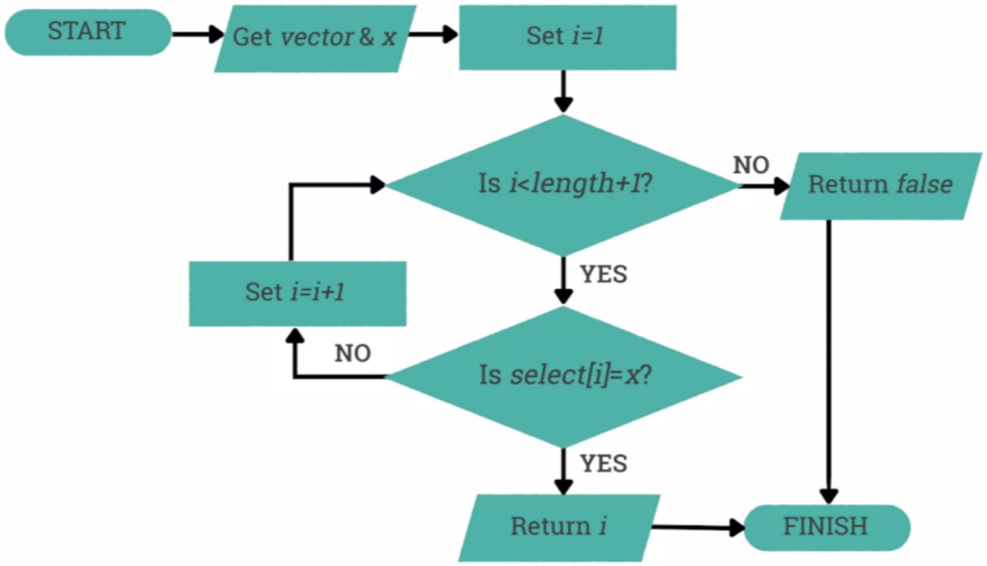
- So the removeAt! operation describes above in the diagram just updates the length value of the dynamic array in the array at the top. So this was the removeAt![2] operation shown in terms of its array implementation



- So this was the insertAt![4,2] operation shown in terms of its array implementation
- All of the data in a vector, queue, stack and dynamic array can formed in a line. This is because the data is sequential with one element of data coming after another in a fixed total order
    - These abstract data structures are called linear abstract data structures or linear collections of data for this reason.
- The reason we use a simple array to implement these abstract data structures is that because in a simple array the elements are also ordered in a linear fashion, which makes the array an ideal choice when showcasing the implementation of these abstract linear data structures

- Linear search algorithm = with this algorithm we are trying to solve this particular problem at first. Is there an element in our input vector with the value x stored there?
    - Our answer will be a Boolean so it'll be true or false
    - The idea behind the linear search algorithm is that we don't know anything about the vector. So we need to systematically "look" at all of the elements if that value x is stored in any of the elements

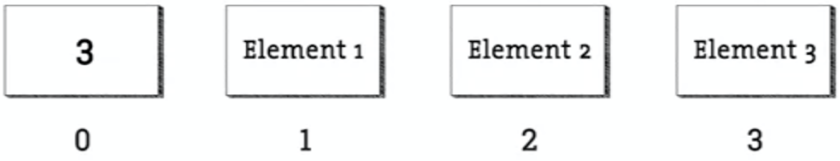


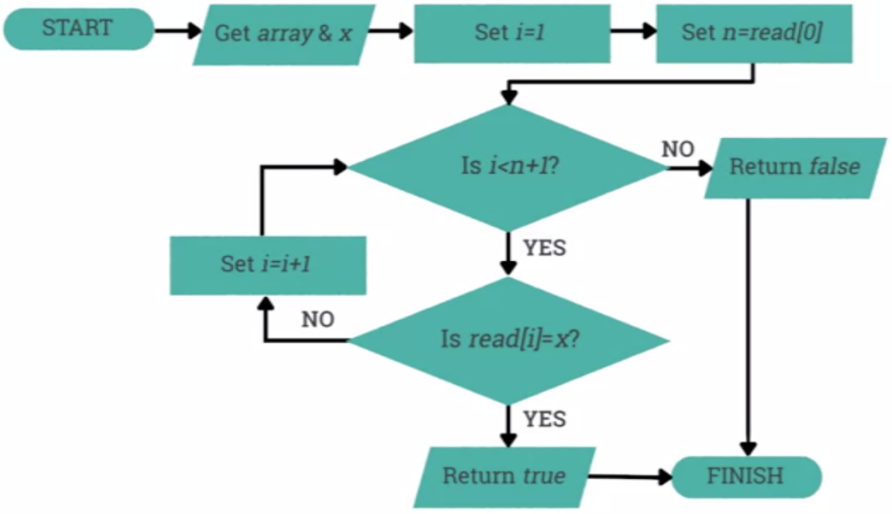


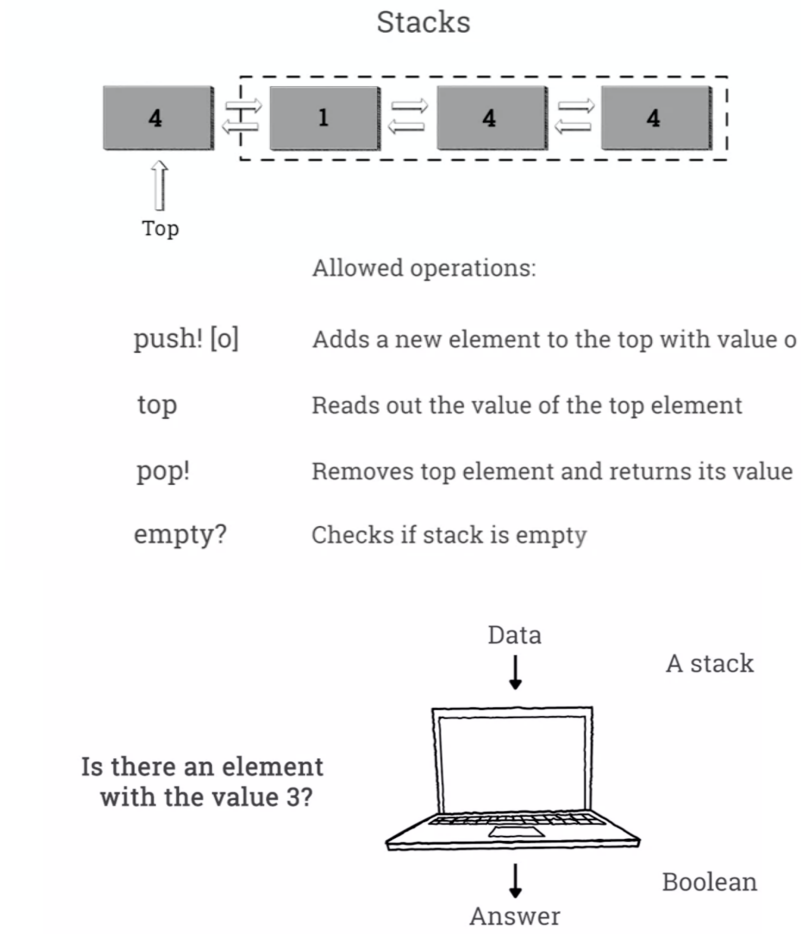
- The second flowchart is also for the linear search algorithm, but it solves a different problem which is if we're given a vector and the value x, we going return the value i, which is labelling the element, if the value x is stored in the ith element and we're going to return false if we do not find the value x stored at any of the elements

Linear Search Algorithm

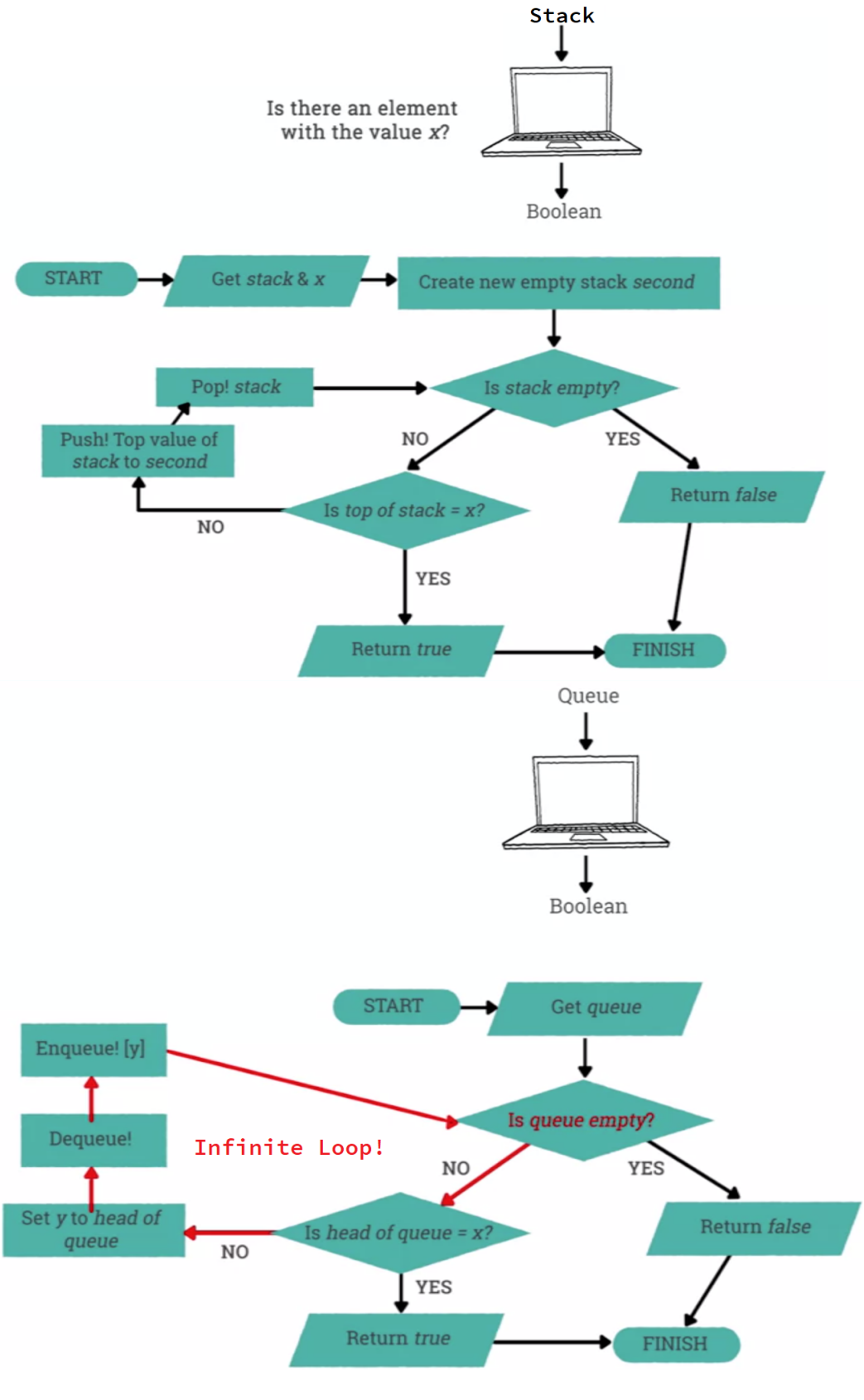
function LinearSearch($v, item$)
  for $(1 \leq i \leq \text{LENGTH}[v])$ **do**
    if $v[i] = item$ **then**
      return i
    end if
  end for
  return FALSE
end function

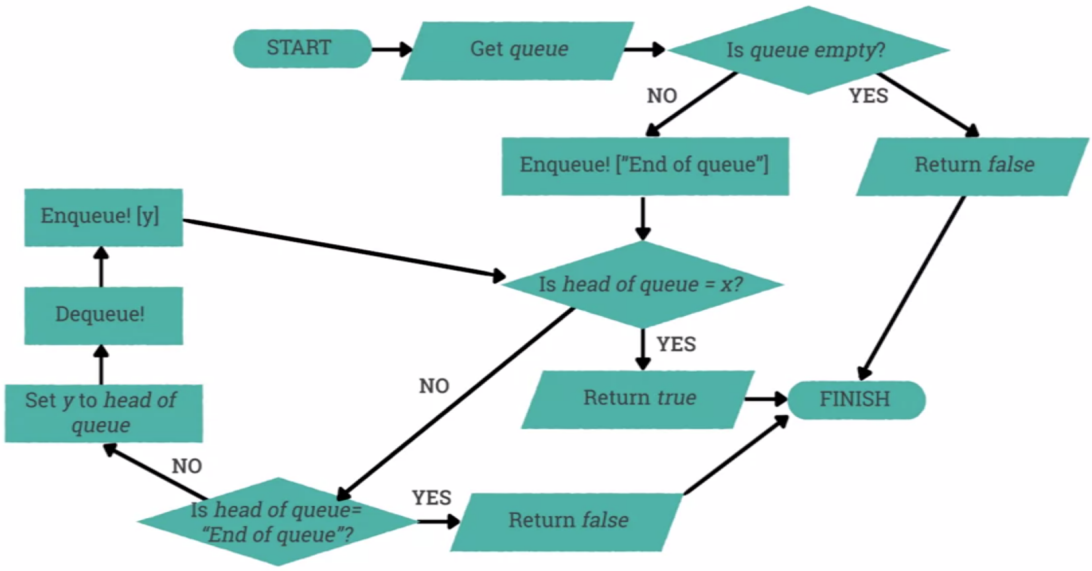






- We can't use the linear search algorithm in this case because not every element of the stack is accessible.
  - What we could do is apply the top operation and then pop! to see if the value is stored in the top element. The problem with this is it could completely destroy the stack!
    - Once we pop an element off the stack if we don't save it anywhere it's gone forever!
  - To remedy this problem we could a second stack. So when we pop something from the first stack that we're searching, we push it to the second stack. In this way the data goes somewhere and that place is a stack.


Stack
Is there an element with the value x?
Boolean
START
Get stack & x
Create new empty stack second
Pop! stack
Is stack empty?
Push! Top value of stack to second
NO
YES
Is top of stack = x?
Return false
NO
YES
Return true
FINISH
Queue
Boolean
START
Get queue
Enqueue! [y]
Is queue empty?
Dequeue!
Infinite Loop!
NO
YES
Set y to head of queue
Is head of queue = x?
Return false
NO
YES
Return true
FINISH



- This second flowchart above deals with the infinite loop problem present in the flowchart about searching a queue

Summary

In this week, we learned about searching, what an abstract data type is, what a data structure is, what an array is, how we implement a vector by an array, what a dynamic array is, how to implement a dynamic array, what the linear search algorithm is, how we can perform the linear search algorithm on an array implementation of vector, how we can search stacks/queues without destroying their element and so much more.