Algorithms & Data Structures I Week 13 Lecture Note

Notebook:	Algorithms & Data Structures I		
Created:	2020-10-21 4:14 PM	**Updated:**	2021-01-18 3:29 PM
Author:	SUKHJIT MANN		

Cornell Notes	**Topic:** Sorting Data II, Part 1	Course: BSc Computer Science Class: CM1035 Algorithms & Data Structures I [Lecture] Date: January 18, 2021

Essential Question:

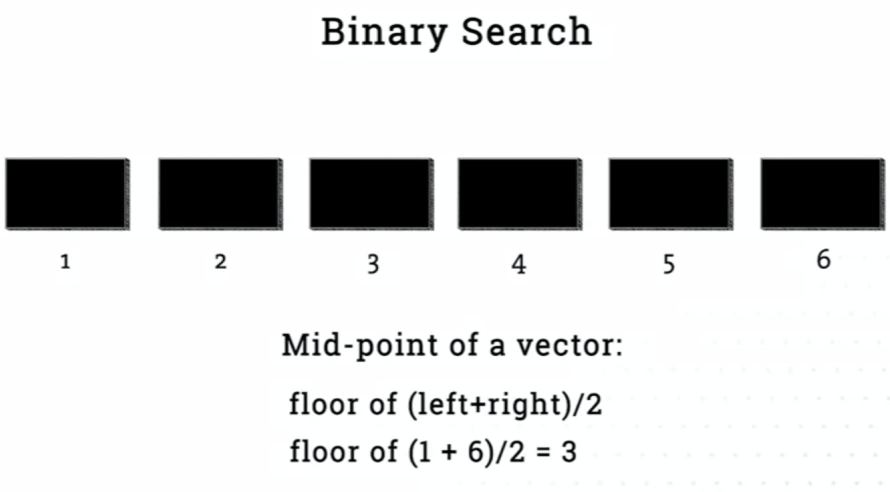
What is the binary search algorithm?

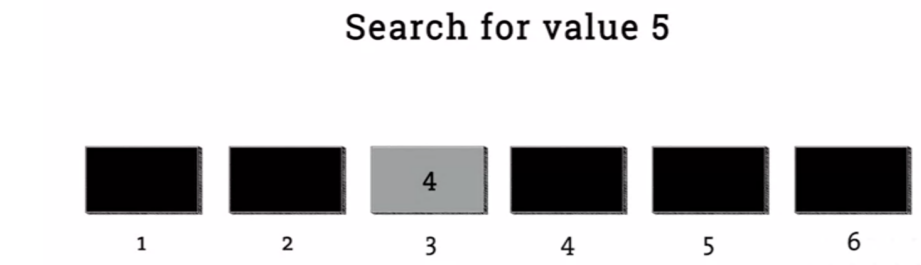
Questions/Cues:

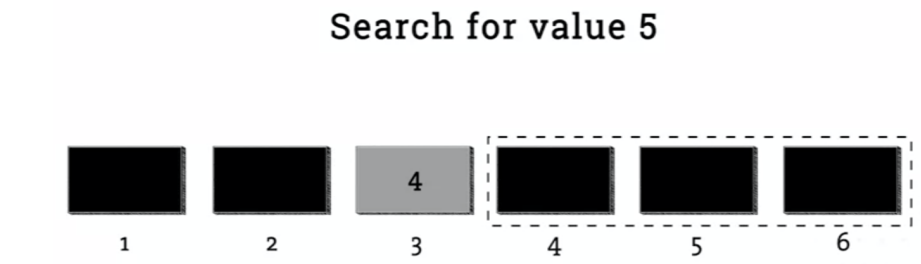
- What is the binary search algorithm in simple terms?
- What is the flowchart and pseudocode for the BSA?
- What is the worst-case time complexity for the BSA
- Can use BSA on unsorted vectors and array?

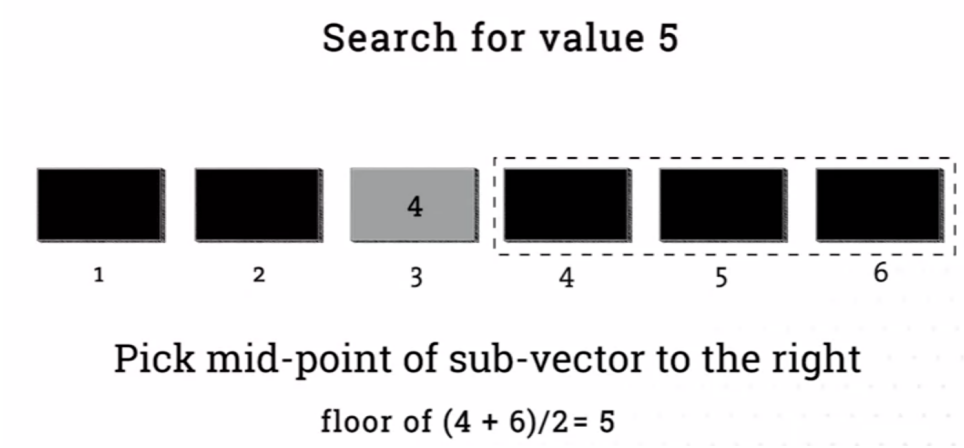
Notes

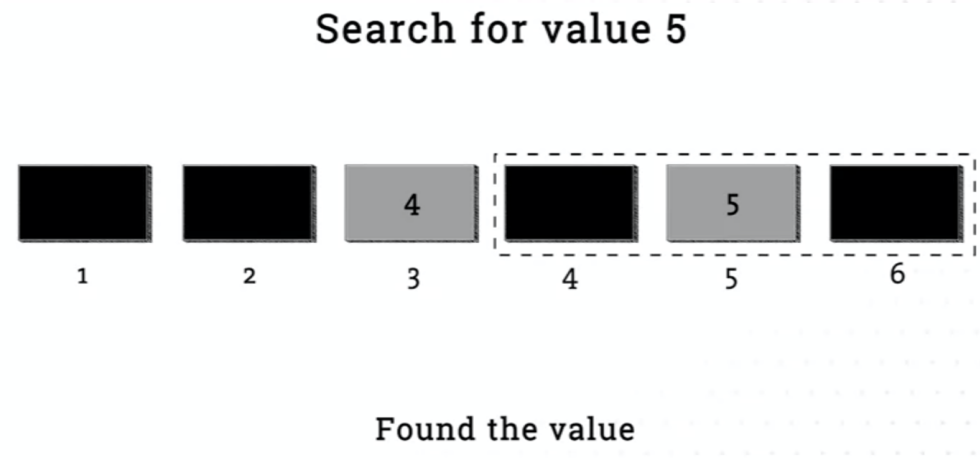
- BSA = Instead of randomly picking an element, we initially pick the midpoint of the vector, then depending on the value stored at this midpoint, we either find the value we want or we consider a smaller array either to left or to the right of this midpoint. In this way, we divide the initial vector into two vectors either side of the midpoint and then search of the smaller vectors.
    - The same process again is applied to the smaller vector by inspecting its midpoint and then dividing it up again into two depending on what is found at this new midpoint



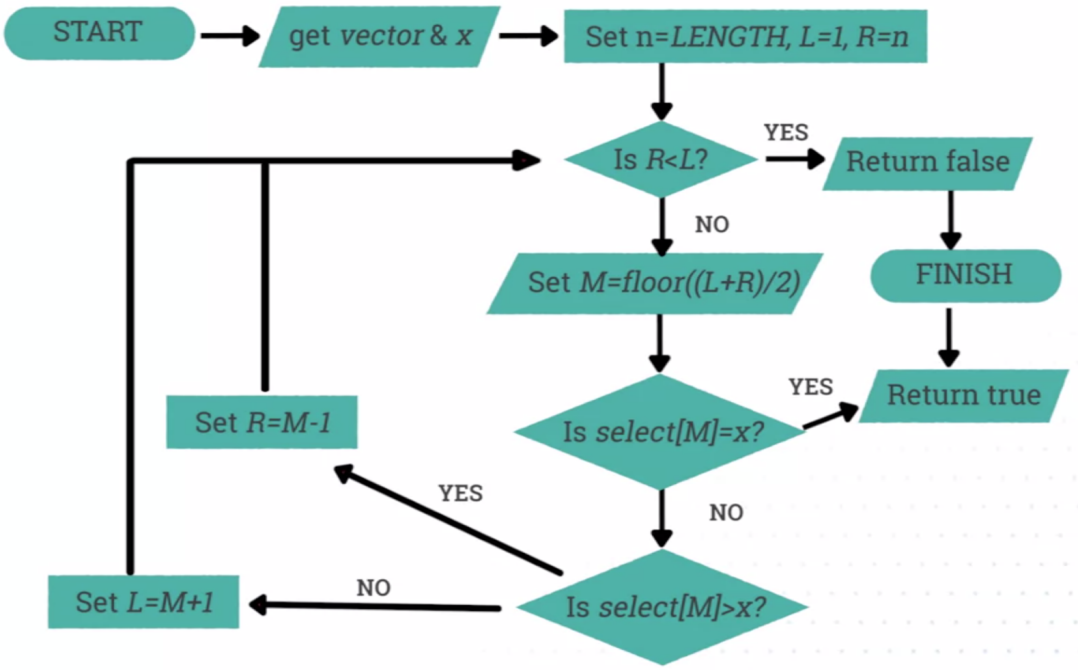








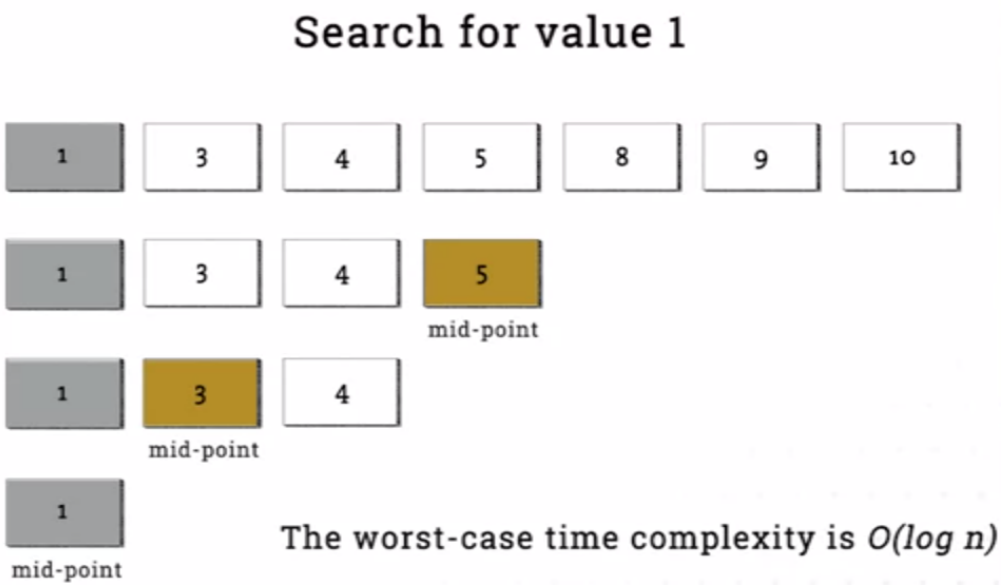
- The BSA divides up the initial sorted vector in half each round until we end up with just one element. So in BSA we don't need to inspect every element of the sorted vector. After inspecting the midpoint, we can completely ignore at least half of the vector and don't need to inspect any of the elements in that half. This is true for all possible inputs to the searching algorithm as long as the input vector is sorted.
    - This is in contrast to the Linear search algorithm, where we might need to inspect every element of the input to find the target value



Binary Search Algorithm

```
function BinarySearch(v, item)
    n ← LENGTH[v]
    R ← n
    L ← 1
    while R ≥ L do
        M ← ⌊(L + R)/2⌋
        if v[M] = item then
            return TRUE
        else if v[M] > item then
            R ← M − 1
        else
            L ← M + 1
        end if
    end while
    return FALSE
end function
```

- The Best-case input for the BSA is if the value we are searching for is stored in the midpoint of the vector, since this is the first element we inspect and if the value is stored there then we find it quickly and total number of operations required is O(1) or constant for this particular input
- There are multiple worst-case inputs for the BSA, one example is if the desired value is the first element of the vector. We will have to divide the vector into half multiple times before the midpoint is the first element
- Another worst-case input for BSA is if the desired value is stored in the element next to the midpoint, we'll still need to divide up the vector multiple times before we inspect that element



- When we have an unsorted vector or array, we can not be guaranteed that the desired value we're looking for isn't in the excluded chunk. A remedy for is this is to first sort the input using bubble or insertion sort and apply the BSA on it. This whole process will no longer have a worst-case time complexity of O(log n) since the sorting algorithms have a much larger worst-case time complexity of O(n^2). So in this case it advised to just do linear search

Summary

In this week, we learned about the Binary Search Algorithm