Binary Search

Binary Search Problem

- Given a sorted array of integers and a target value, find out if target exists in the array or not

- Input: arr[] = {3,4,6,7}, target = 4

- Output: Target is in index 2

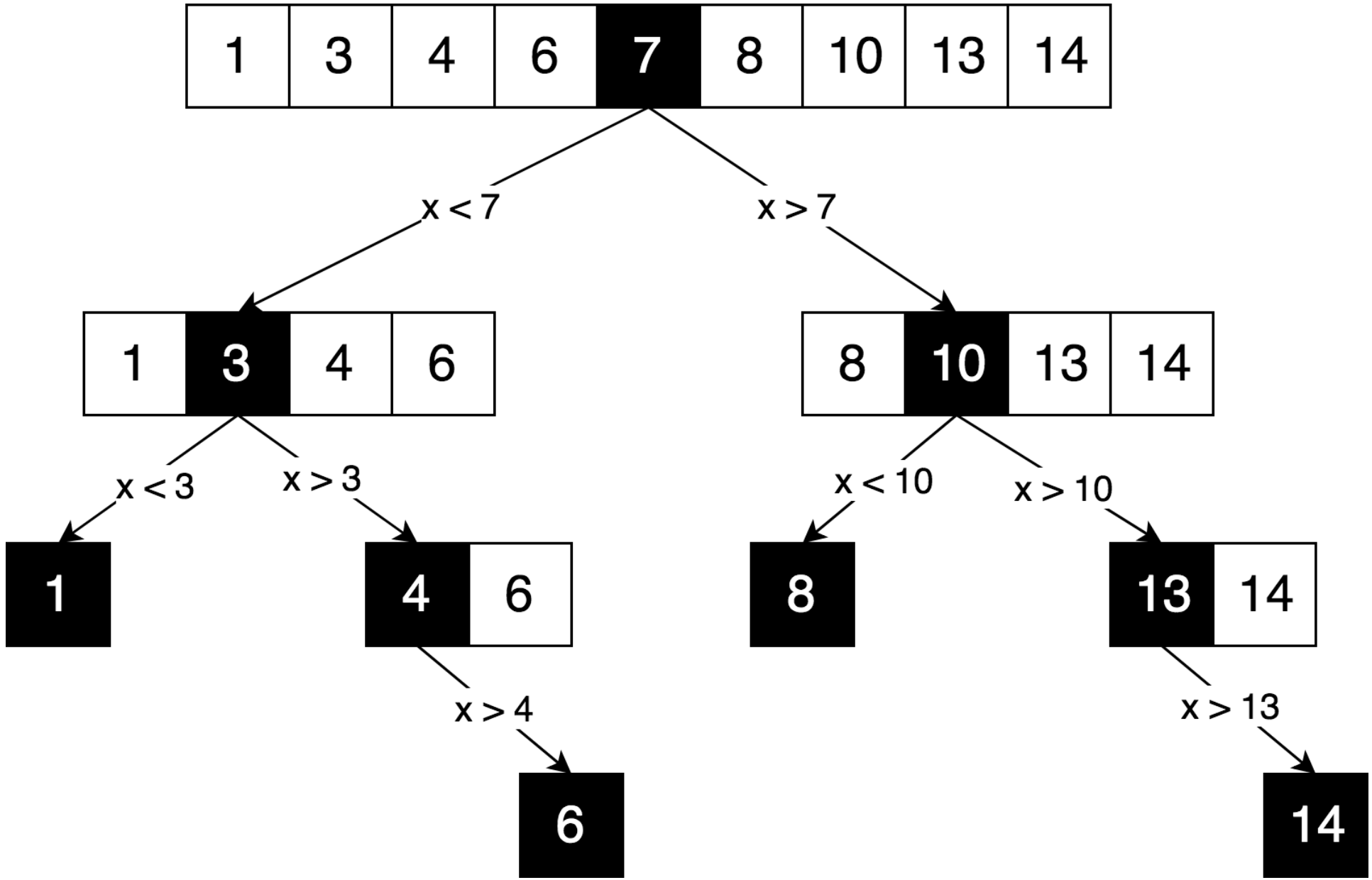
- Trivial Solution: Linear Search O(n) Complexity

Binary Search Problem

- Design O(logn) complexity algorithm
- Divide & Conquer

```
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? cout << "Element is not present in array"
                   : cout << "Element is present at index " << result;
    return 0;
}
```

```
// C program to implement recursive Binary Search
#include <stdio.h>

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}
```


1 3 4 6 7 8 10 13 14
x < 7
x > 7
1 3 4 6
8 10 13 14
x < 3
x > 3
x < 10
x > 10
1
4 6
8
13 14
x > 4
x > 13
6
14

Binary Search Loop Invariant

- Three conditions
  - Array arr is sorted in ascending order
  - l <= r
  - x belong to arr [l…..r]

Binary Search Loop Invariant

- Use loop invariant that the code is correct

- Initialization: The loop invariant has three parts

1. Array is sorted due to precondition of the method
2. Since arr.length is at least 1, thus l<=r
3. x is in arr b/c it is whole array and precondition guarantees that x is in array

Binary Search Loop Invariant

- Maintenance: The loop invariant has three parts

1. Array arr is never changed so Case 1 is always true i.e. arr is sorted
2. Let l' and r' are the values of l and r at the end of 1st iteration, then we need l'<r' and x belongs to arr[l'…..r']
3. Let m be the average of l and r, thus x belongs to arr[l…m] or arr [m+1….j]
4. Case k belongs to arr[1….m]

   must have x<=a[m] and thus if condition is true, then r'=m, l =1, this l'<r' and since x belongs to arr[1…m], by assumption its belong to arr[l'…..j']

Binary Search Loop Invariant

- Maintenance: The loop invariant has three parts

5. Case k does not belong to arr[1….m]

   must have x>a[m] and thus if condition is true, then r'=r, l =m+1, this l'<r' and since x belongs to arr[m+1…..r], by assumption its belong to arr[l'…..j']

   For the algorithm to be correct, arr[1] = x and happens only when l = r

- Termination: The value r−l is guaranteed to be non-negative. Because integer division rounds down, it gets smaller on every loop iteration. Therefore the loop eventually terminates

- More Detail: https://www.cs.cornell.edu/courses/cs2112/2015fa/lectures/lec_loopinv/

Time Complexity

- $T(n) = 1 + T(n/2)$

- $O(n)$