**Indian Institute of Information Technology Allahabad**

Data Structures and Algorithms

Searching
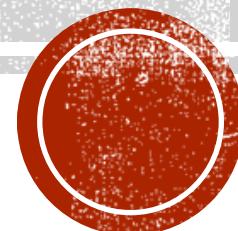
Dr. Shiv Ram Dubey
Associate Professor
Department of Information Technology
Indian Institute of Information Technology, Allahabad

Email: srdubey@iiita.ac.in   Web: https://profile.iiita.ac.in/srdubey/

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Searching

[0]　　　　**[1]**　　　　**[2]**　　　　**[3]**　　　　**[4]**　　　　　　**[700]**

Number 701466868　Number 281942902　Number 233667136　Number 580625685　Number 506643548　...　Number 155778322

Each record in list has an associated key.
In this example, the keys are ID numbers.

Given a particular key, how can we efficiently retrieve the record from the list?

Number 580625685

Searching

Check if a given element (called key) occurs in the array.

- Example: array of student records; rollno can be the key.

Two methods to be discussed:

- If the array elements are unsorted.
    - Linear search
- If the array elements are sorted.
    - Binary search

Linear Search

Basic Concept of Linear Search

Basic idea:

- Start at the beginning of the array.
- Inspect elements one by one to see if it matches the key.

Basic Concept of Linear Search

Basic idea:

- Start at the beginning of the array.
- Inspect elements one by one to see if it matches the key.

Time complexity:

- A measure of how long an algorithm runs before terminating.
- If there are *n* elements in the array:
  - Best case:

    match found in first element (1 search operation)
  - Worst case:

    no match found, or match found in the last element (n search operations)
  - Average case: *(n + 1) / 2   search operations*

Linear Search

Function linear_search returns the array index where a match is found.
It returns -1 if there is no match.

```c
int linear_search (int a[], int size, int key)
{
        int pos = 0;
        while ((pos < size) && (a[pos] != key))
                pos++;
        if (pos < size)
                return pos;     /* Return the position of match */
        return -1;              /* No match found */
}
```

Worst Case Time for Linear Search

- For an array of *n* elements, the worst case time for serial search requires *n* array accesses: $O(n)$.

- Consider cases where we must loop over all *n* records:

  - desired record appears in the last position of the array

  - desired record does not appear in the array at all

Average Case for Linear Search

Assumptions:

    1. All keys are equally likely in a search

    2. We always search for a key that is in the array

Example:

- We have an array of 10 records.

- If search for the first record, then it requires 1 array access; if the second, then 2 array accesses. *etc.*

The average of all these searches is:

    (1+2+3+4+5+6+7+8+9+10)/10 = 5.5

Average Case Time for Linear Search

Generalize for array size *n.*

Expression for average-case running time:

(1+2+...+n)/n = n(n+1)/2n = (n+1)/2

Therefore, average case time complexity for serial search is O(n).

Binary Search

Basic Concept

Binary search is applicable if the array is *sorted*.

BASIC IDEA

- Look for the target in the middle.

- If you don't find it, you can ignore half of the array, and repeat the process with the other half.

In every step, we reduce the number of elements to search in by half.

The Basic Strategy

What do we want?

- Find split between values larger and smaller than key:
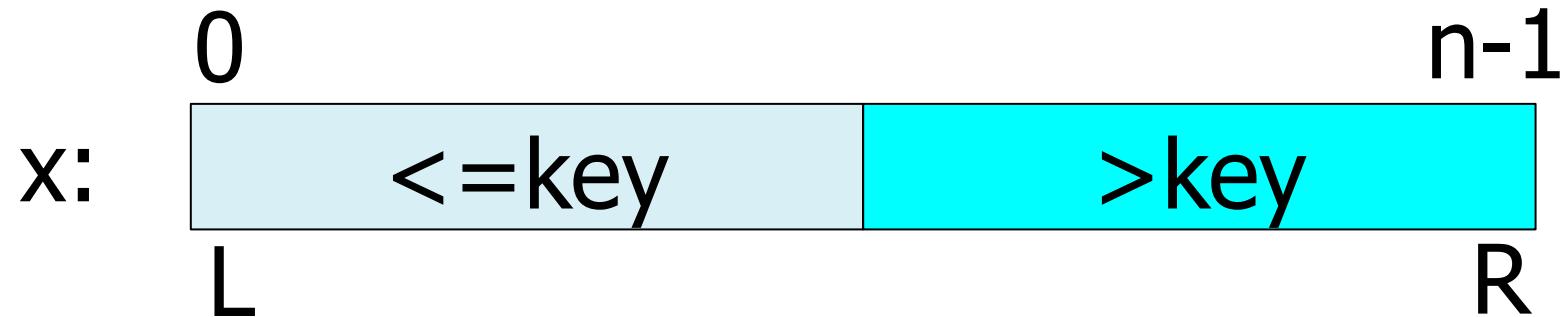
The Basic Strategy

What do we want?

- Find split between values larger and smaller than key:
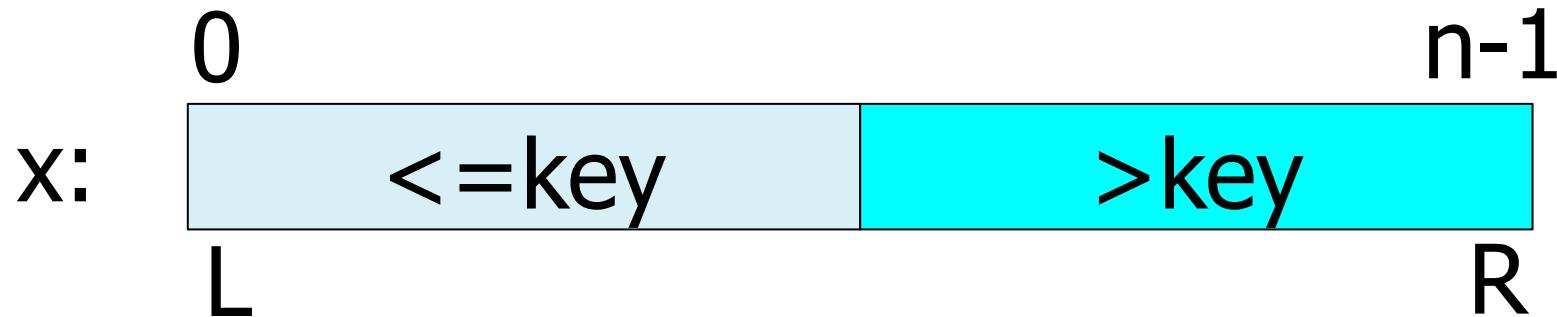
- Situation while searching:
  - Initially L and R contains the indices of first and last elements.

- Look at the element at index [(L+R)/2].
  - Move L or R to the middle depending on the outcome of test.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Binary Search

Example: sorted array of integer keys. Key=7.
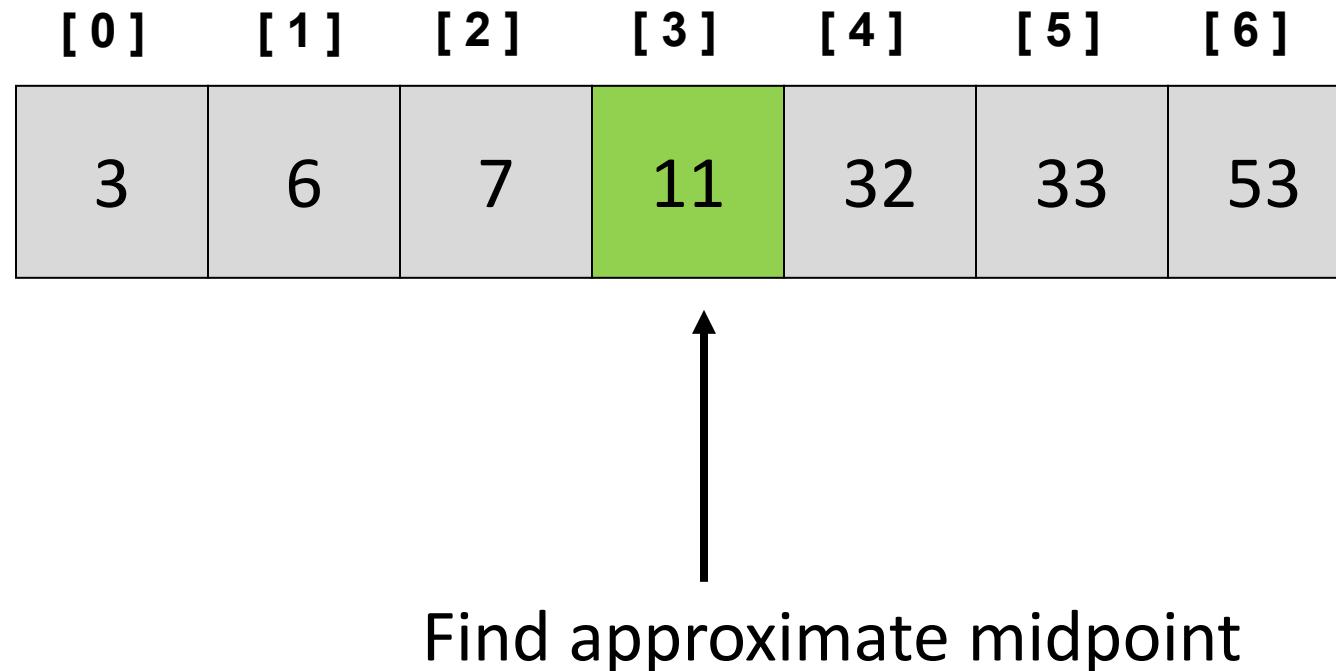
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Find approximate midpoint

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Is 7 = midpoint key? NO.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Is 7 < midpoint key? YES.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53
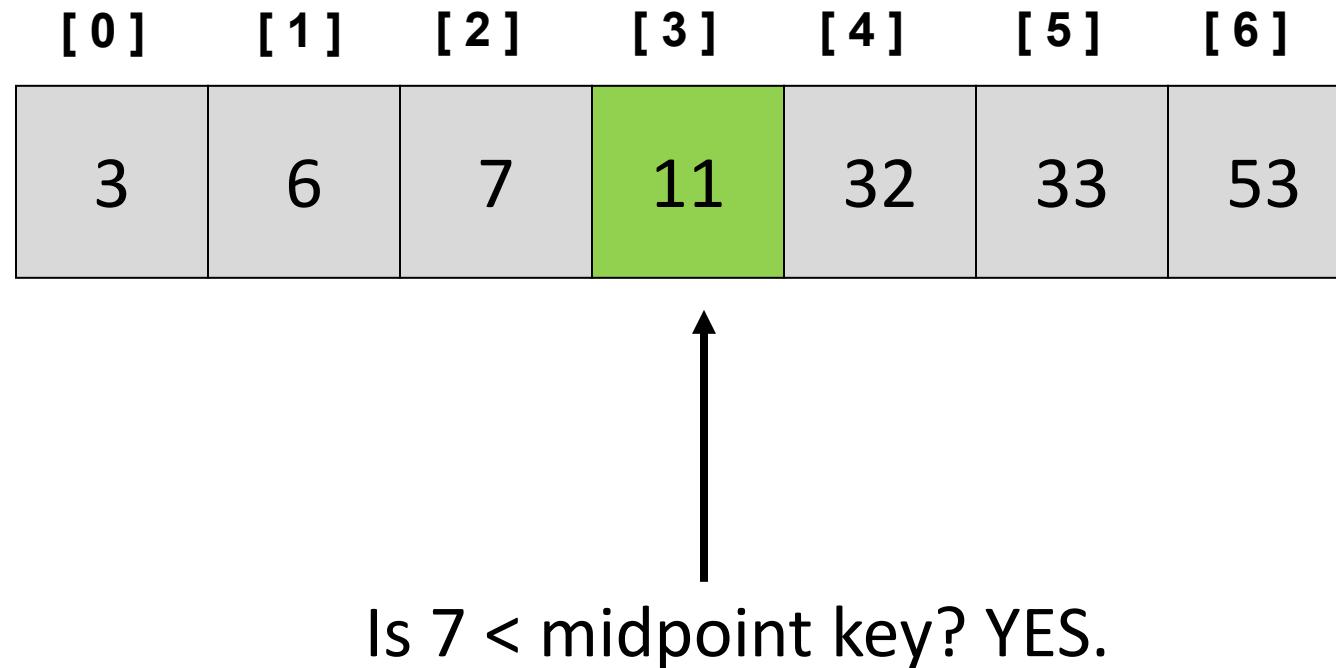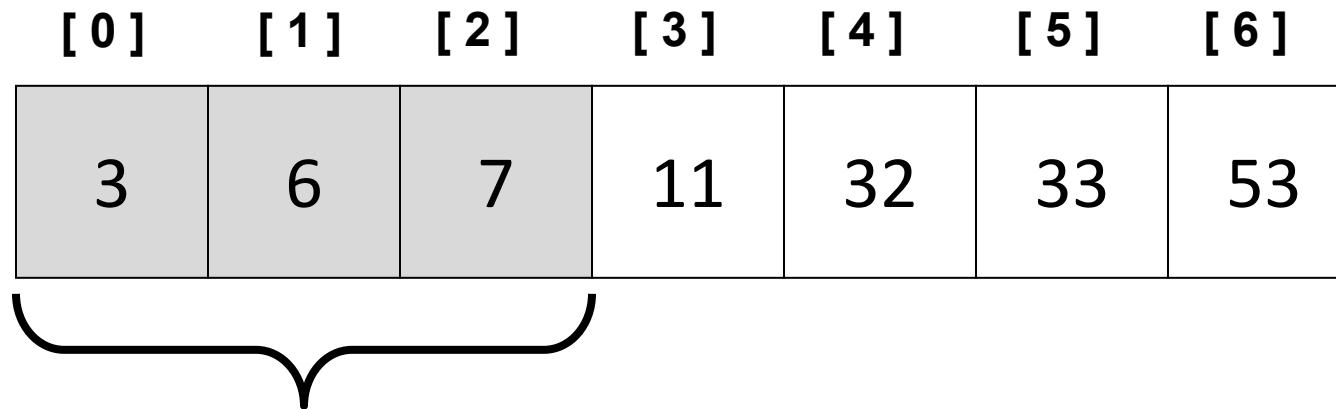
Search for the target in the area before midpoint.

Binary Search
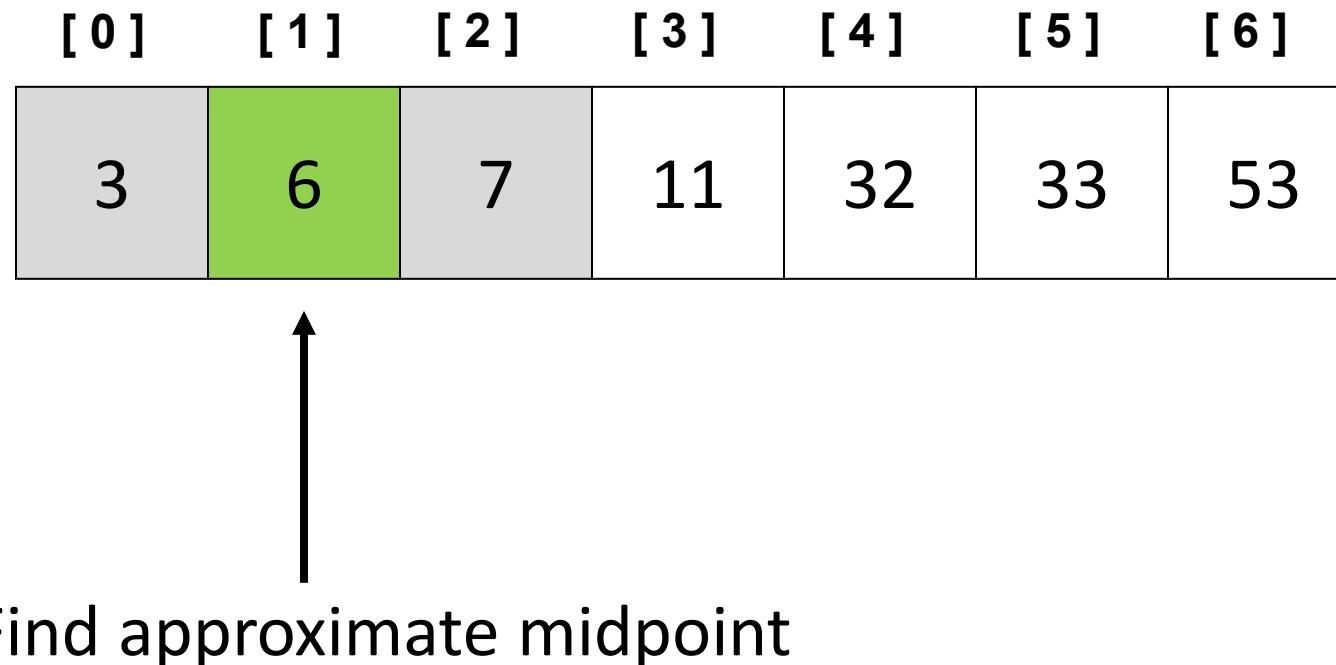
Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Find approximate midpoint

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Target = key of midpoint? NO.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Target < key of midpoint? NO.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53
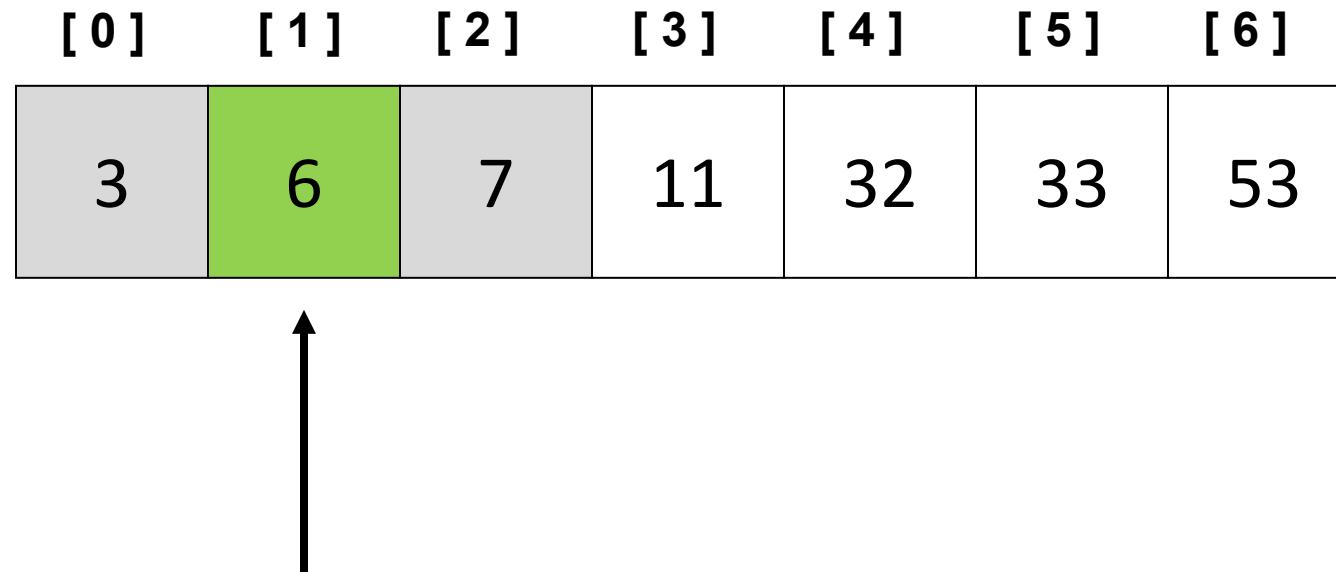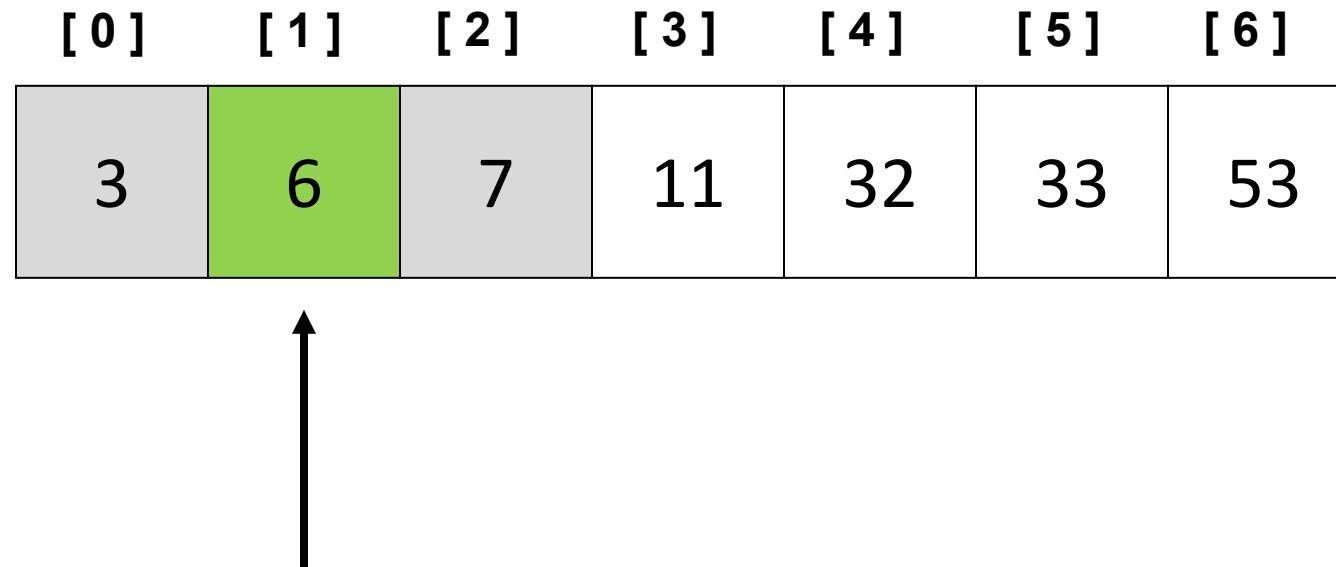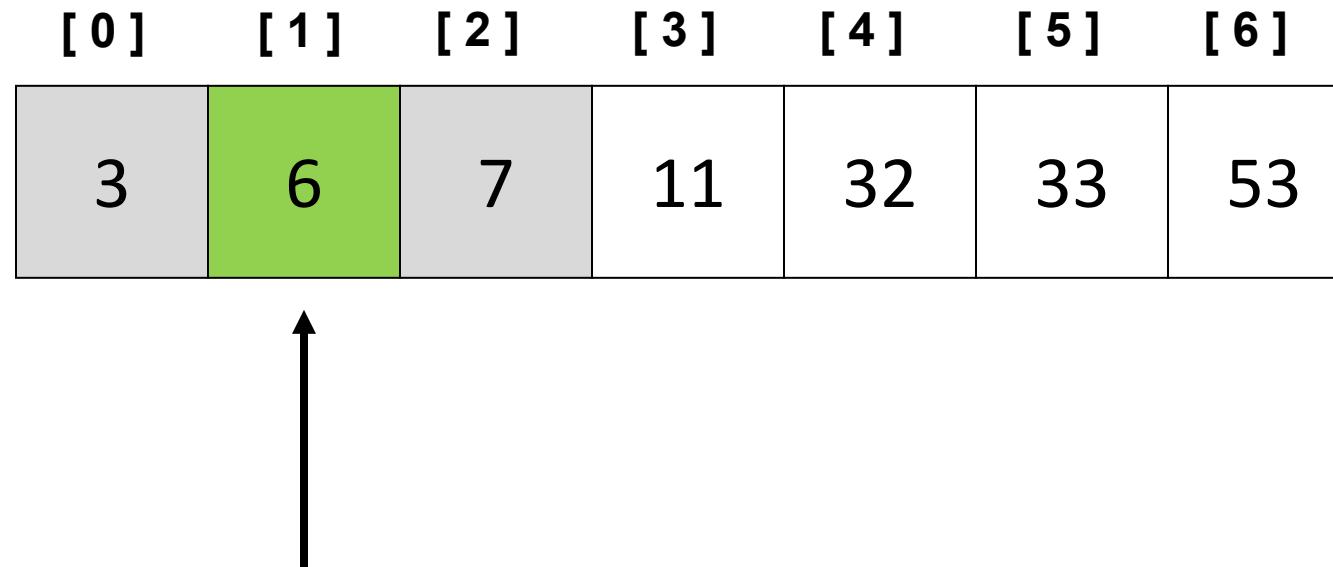
Target > key of midpoint? YES.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53
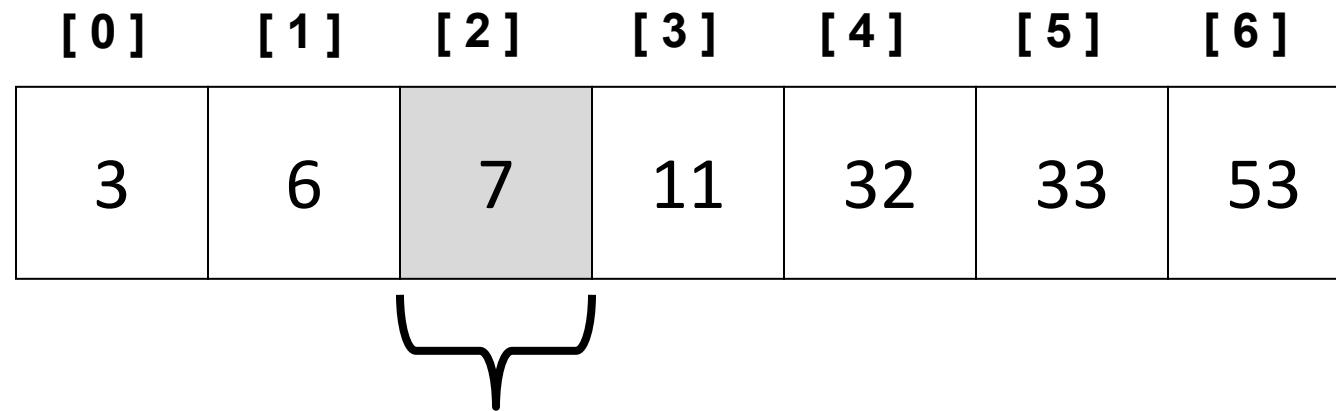
Search for the target in the area after midpoint.

Binary Search

Example: sorted array of integer keys. Key=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Find approximate midpoint.
Is target = midpoint key? YES.

Binary Search

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____;
        while ( _____ )
        {

                _____;
        }


        _____;
}
```

The Basic Search Iteration

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____;
        while ( _____ )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key) L = mid;
                else R = mid;

        }


        _____;

}
```

/* If key appears in x[0..size-1], return its location, pos such that x[pos]==key.
If not found, return -1 */

Loop Termination Criterion

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____;
        while ( L+1 != R )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key) L = mid;
                else R = mid;
        }


        _____;

}
```

/* If key appears in x[0..size-1], return its location, pos such that x[pos]==key.
If not found, return -1 */

Initialization and Return Value

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;
        L = −1;  R = size;
        while ( L+1 != R )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key) L = mid;
                else R = mid;
        }
        if (L >= 0 && x[L] == key) return L;
        else return −1;
}
```

/* If key appears in x[0..size-1], return its location, pos such that x[pos]==key.
If not found, return -1 */

Binary Search Examples

Sorted array

-17 -5 3   6   12 21 45 63 50

Trace :

bin_search (x, 9, 3); ⟶

bin_search (x, 9, 145);

bin_search (x, 9, 45);

L= –1;   R=9;   x[4]=12;
L= –1;   R=4;   x[1]= –5;
L= 1;   R=4;   x[2]=3;
L=2;   R=4;   x[3]=6;
L=2;   R=3;   return L;

We may modify the algorithm by checking equality with x[mid].

Is it worth the trouble ?

Suppose that the array x has 1000 elements.

Ordinary search

– If *key* is a member of x, it would require 500 comparisons on the average.

Is it worth the trouble ?

Suppose that the array x has 1000 elements.

Ordinary search

– If *key* is a member of x, it would require 500 comparisons on the average.

Binary search

- after 1st compare, left with 500 elements.
- after 2nd compare, left with 250 elements.
- After at most 10 steps, you are done.

Time Complexity

If there are n elements in the array.

- Number of iterations required: $\log_2 n$

For n = 64 (say).

- Initially, list size = 64.
- After first compare, list size = 32.
- After second compare, list size = 16.
- After third compare, list size = 8.
- …….
- After sixth compare, list size = 1.

$2^k = n$, where k is the number of steps.

$\log_2 64 = 6$
$\log_2 1024 = 10$

Are exactly log₂n steps required for all cases?

$$\boxed{\text{-17 \quad -5 \quad 3 \quad 6 \quad 12 \quad 21 \quad 45 \quad 63 \quad 50}}$$

Trace of binsearch(x,9,12):

L= –1;   R=9;   x[4]=12;

L= 4;   R=9;   x[6]= 45;

L= 4;   R=6;   x[5]=21;

L=4;   R=5;   return L;

We know in first iteration that x[4] = 12. Why not stop then?

Are exactly $\log_2 n$ steps required for all cases?

```
int bin_search_1 (int x[ ], int size, int key)
{
        int L, R, mid;
        L = 0;    R = size-1;
        while ( L <= R )
        {
            mid = (L + R) / 2;
            if (x[mid] == key) return mid;
            if (x[mid] < key) L = mid+1;
            else R = mid-1;
        }
        return -1;
}
```

Exercise

Write a recursive version of the Binary Search function.

Summary

- Linear search: average case $O(n)$

- Binary search: average case $O(\log_2 n)$

Acknowledgement

- Boston University

- IIT Kharagpur

Thank You