DATA STRUCTURES

UNIT-1

Searching Techniques

Dr G. Kalyani

Topics

- **What is Searching**
- **Searching Technique-1**
- **Searching Technique-2**

What is Searching?

- Search is the process of looking for something.

Searching Techniques

- Linear Search
- Binary Search
- Ternary Search
- Exponential Search
- Interpolation Search
- Jump Search

Searching Techniques

- **Linear Search**
- **Binary Search**
- Ternary Search
- Exponential Search
- Interpolation Search
- Jump Search

Searching Technique-1: Linear Search

Linear Search

- you are asked to find the name of the person having phone number say "123456" with the help of a telephone directory.

- Since telephone directory is sorted by name not by numbers, we have to go through each and every number of the directory.

Linear Search

- A method for finding a target value within an array of elements.
- Also called Sequential Search because it sequentially checks each element of the array.
- A search traverses the collection until
  - The desired element is found
  - Or the collection is exhausted
- Input is a set of elements and an element (key) to find
- Output
  - Print a message (“Found”, “Not Found)
  - Return a value (position of key)

Linear Search

Linear Search Algorithm

```
Algorithm LinearSearch(array, size, key)
{
//Input – An array, size of the array and the search key
//Output – location of the key (if found), otherwise not found.
{
   for i = 0 to size-1 do
    {
       if array[i] = key then
             write element found in i^th position;
    }
     Write element not found;
}
```

Linear Search Example

Algorithm LinearSearch(array, size, key)

{

//Input – An array, size of the array and the search key

//Output – location of the key (if found), otherwise wrong location.

{

for i = 1 to size do

{

if array[i] = key then

write element found in i^{th} position;

}

Write element not found;

}

array

7	12	5	22	13	32
1	2	3	4	5	6

Key= 13

Linear Search Example

```
Algorithm LinearSearch(array, size, key)
{
//Input – An array, size of the array and the search key
//Output – location of the key (if found), otherwise wrong
   location.
{
  for i = 1 to size do
   {
      if array[i] = key then
           write element found in ith position;
   }
    Write element not found;
}
```

array

1	2	3	4	5	6
7	12	5	22	13	32

Key = 13

Linear Search Example

Algorithm LinearSearch(array, size, key)

{

//Input – An array, size of the array and the search key

//Output – location of the key (if found), otherwise wrong location.

{

for i = 1 to size do

{

if array[i] = key then

write element found in i^{th} position;

}

Write element not found;

}

my_array	7	12	5	22	13	32
	1	2	3	4	5	6

key = 13

Linear Search Example

```
Algorithm LinearSearch(array, size, key)
{
//Input – An array, size of the array and the search key
//Output – location of the key (if found), otherwise wrong
   location.
{
   for i = 1 to size do
    {
       if array[i] = key then
            write element found in ith position;
    }
    Write element not found;
}
```

my_array	7	12	5	22	13	32
	1	2	3	4	5	6

Key = 13

Linear Search Example

```
Algorithm LinearSearch(array, size, key)
{
//Input – An array, size of the array and the search key
//Output – location of the key (if found), otherwise wrong
    location.
{
   for i = 1 to size do
    {
       if array[i] = key then
              write element found in ith position;
    }
    Write element not found;
}
```

my_array	7	12	5	22	13	32
	1	2	3	4	5	6

Key = 13

Linear Search Example

```
Algorithm LinearSearch(array, size, key)
{
//Input – An array, size of the array and the search key
//Output – location of the key (if found), otherwise wrong
   location.
{
  for i = 1 to size do
   {
      if array[i] = key then
            write element found in ith position;
   }
    Write element not found;
}
```

my_array

7	12	5	22	13	32
1	2	3	4	5	6

Key = 13

Linear Search Analysis: Best Case

```
Algorithm LinearSearch(array, size, key)
{
   for i = 1 to size do
    {
       if array[i] = key then
             write element found in ith
    position;
    }
     Write element not found;
 }
```

Best Case:
1 comparison

Best Case: match with the first item

7	12	5	22	13	32

Key = 7

Linear Search Analysis: Worst Case

```
Algorithm LinearSearch(array, size, key)
{
   for i = 1 to size do
    {
       if array[i] = key then
            write element found in ith
    position;
    }
     Write element not found;
 }
```

Worst Case:
N comparisons

Worst Case: match with the last item (or no match)

7	12	5	22	13	32

target = 32

Pros and Cons of Linear Search

- Advantages :
  - The linear search is simple
  - It is very easy to understand and implement
  - It does not require the data of the array in any particular order
- Disadvantages:
  - It has very poor efficiency because it takes lots of comparisons to find a particular record in big files.
  - Not suitable if input is large and search is frequently required.
  - Linear search is slower then other searching algorithms.

Searching Technique-2: Binary Search

Binary Search

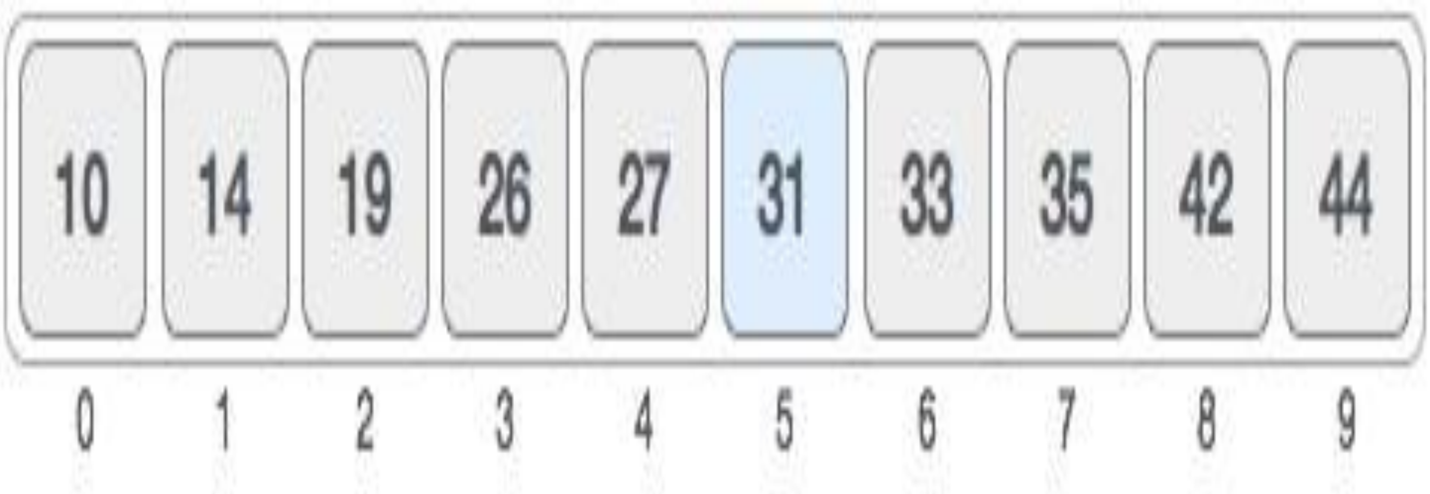
- you are asked to find the number of the person having name say “Ram” with the help of a telephone directory.

Binary Search

- Suitable if the given set of elements are in sorted order.
- Binary search looks for a particular item by comparing the middle item of the collection.
- If a match occurs, then the index of item is returned.
- If the middle item is greater than the item, then the item is searched in the left sub-array to the middle item.
- Otherwise, the item is searched for in the sub-array to the right of the middle item.
- This similar process continues on the sub-array until the size of the subarray reduces to one.

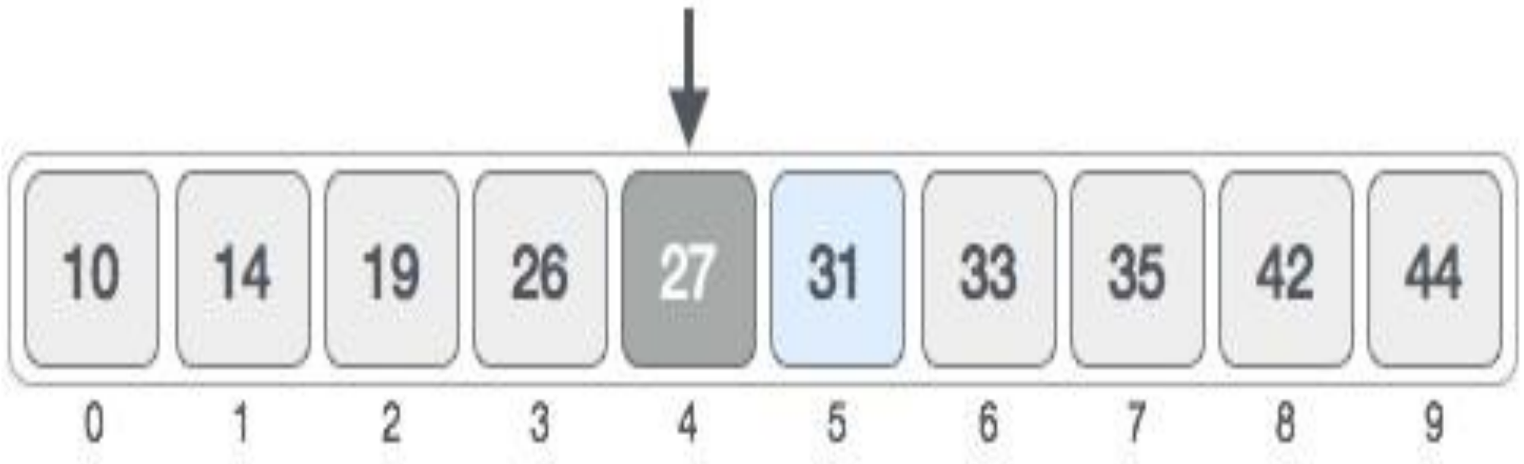
Binary Search Example

- we need to search the location of value 31 using binary search.



Binary Search Example

- Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5).
- So, 4 is the mid of the array.



- Now we compare the value stored at location 4, with the value being searched, i.e. 31.
- We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array.
- so we also know that the target value must be in the upper portion of the array.

Binary Search Example

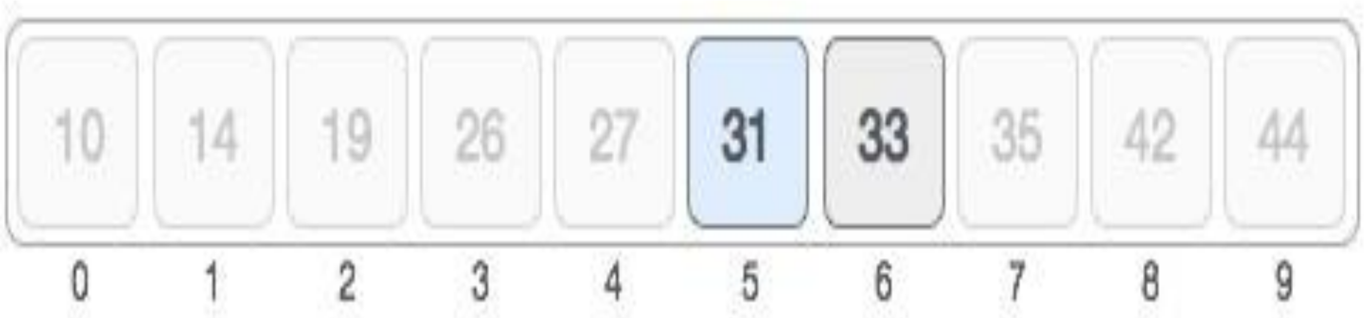
- We change our low to mid + 1 and high remains the same.
- find the new mid value again.

- low = mid + 1=5
- High = 9

- Calculate the middle position again
- mid = low + (high - low) / 2
- = 5 + (9-5)/2 = 5 + 4/2 = 5 + 2 = 7

Binary Search Example

- The value stored at location 7 is not a match,
- rather it is more than what we are looking for.
- So, the value must be in the lower part from this location.
- Hence low remains the same and high will be changed to mid-1.
- i.e. low=5 and high = 7-1=6



- we calculate the mid again.
- Mid = low + (high-low)/2
- = 5 + (6-5)/2
- =5 + ½ = 5.5
- Hence, it is 5

Binary Search Example

- We compare the value stored at location 5 with our target value.
- We find that it is a match.

- We conclude that the target value 31 is stored at location 5.

Steps in Binary Search

- Calculate the middle position
- Compare x with the middle element.
- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half sub array after the mid element. So we recur for right half.
- Else (x is smaller) recur for the left half.

Binary Search Example

- Ex. Binary search for 33

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lo														hi

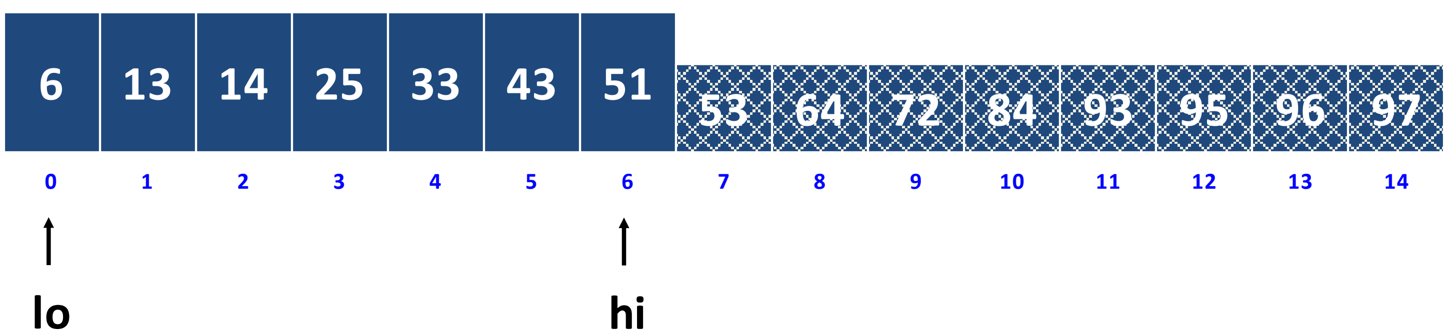
Binary Search Example

- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo							↑ mid							↑ hi

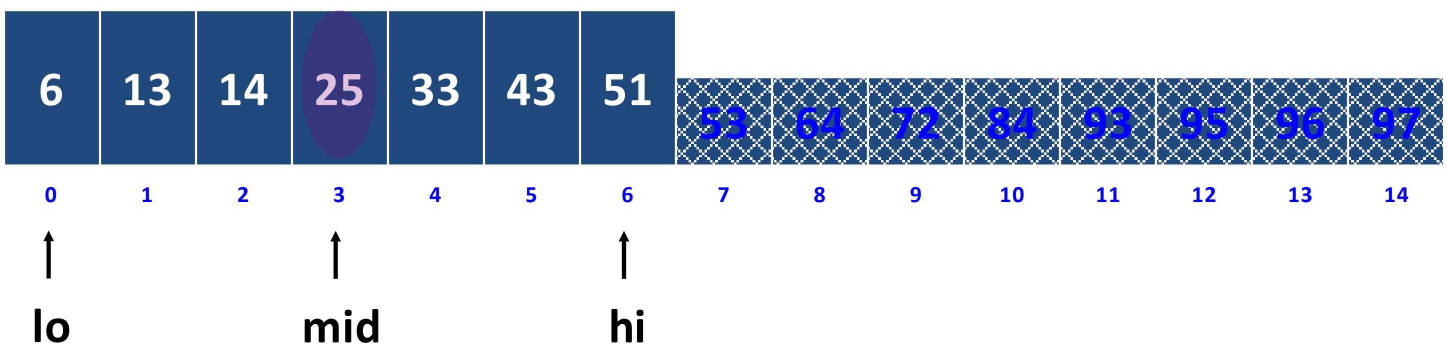
Binary Search Example

- Ex. Binary search for 33.



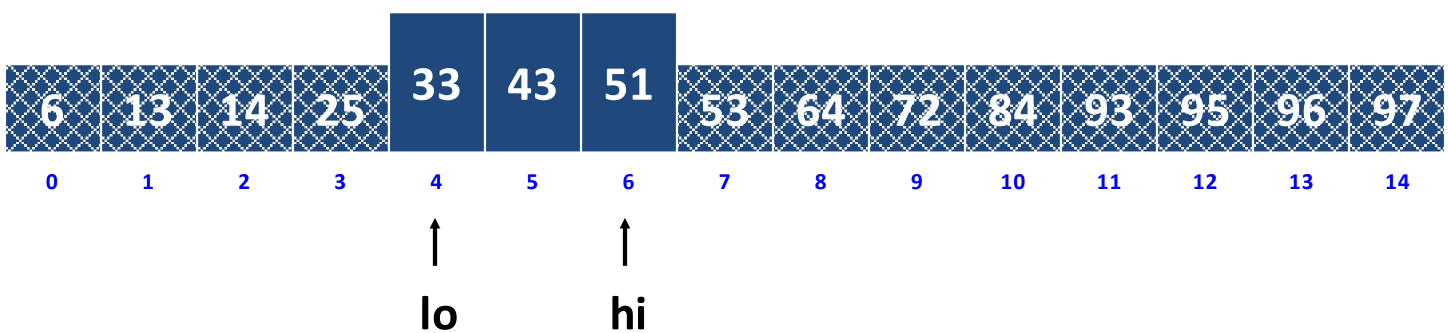
Binary Search Example

- Ex. Binary search for 33.



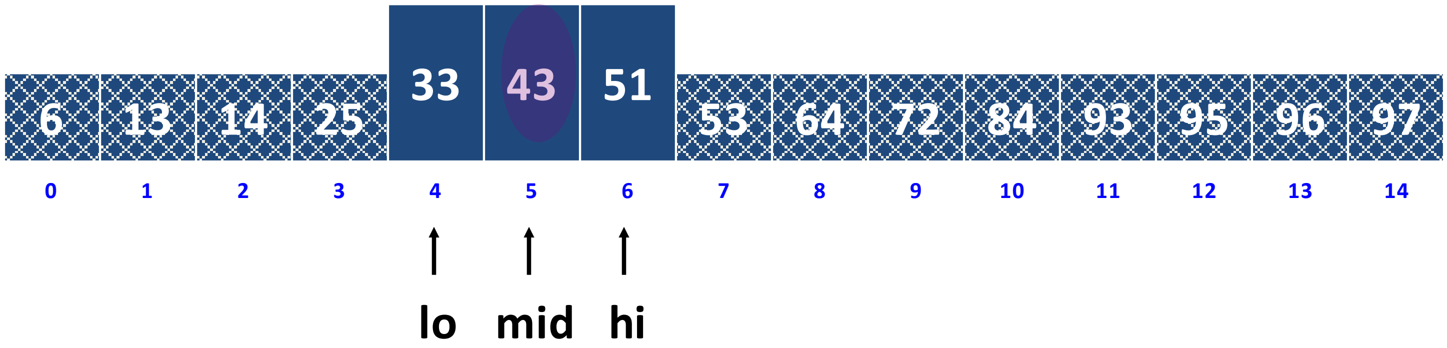
Binary Search Example

- Ex. Binary search for 33.



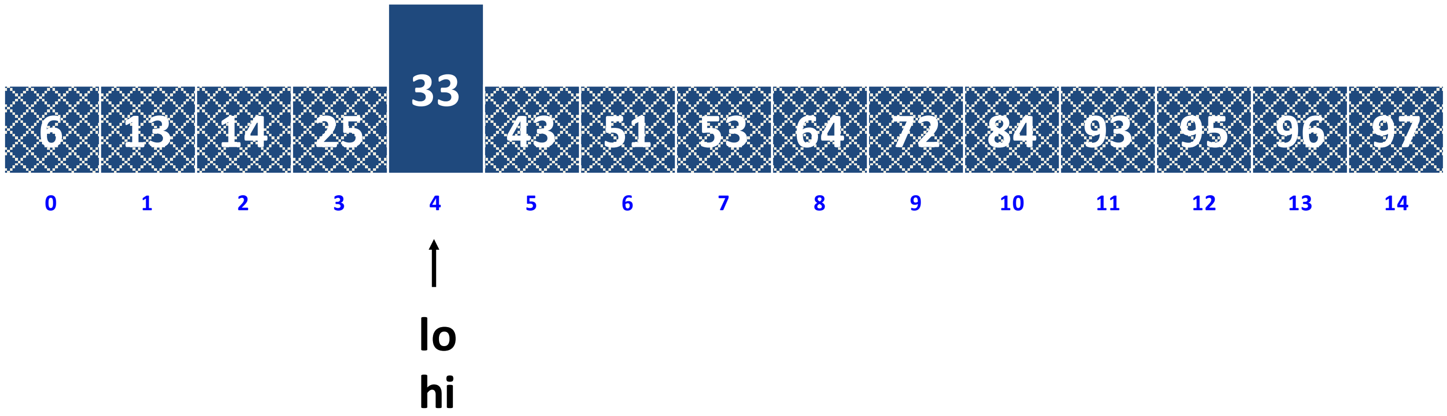
Binary Search Example

- Ex. Binary search for 33.



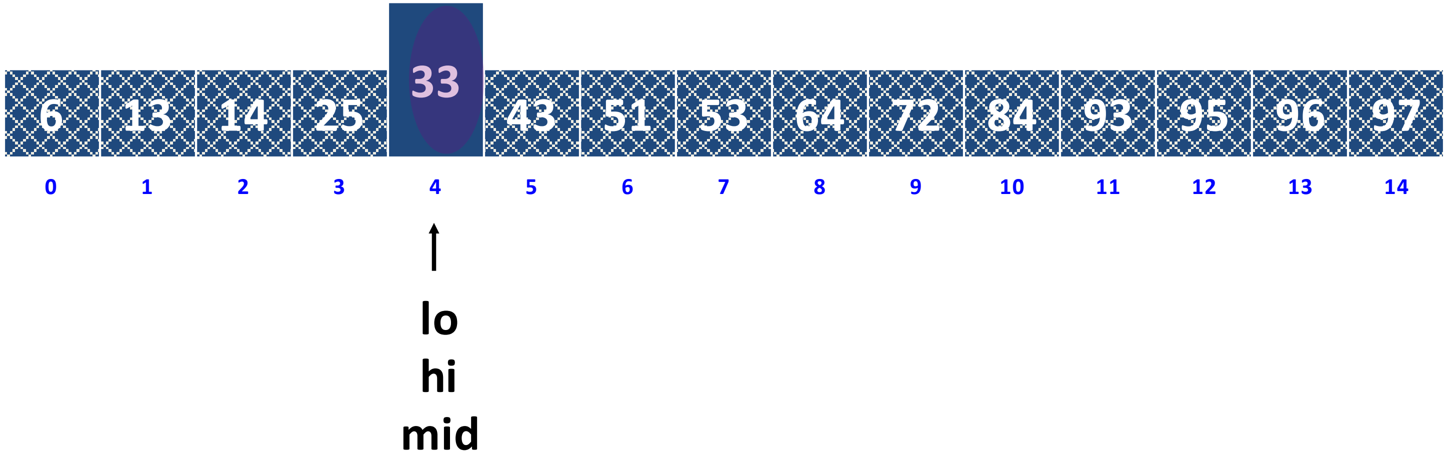
Binary Search Example

- Ex. Binary search for 33.



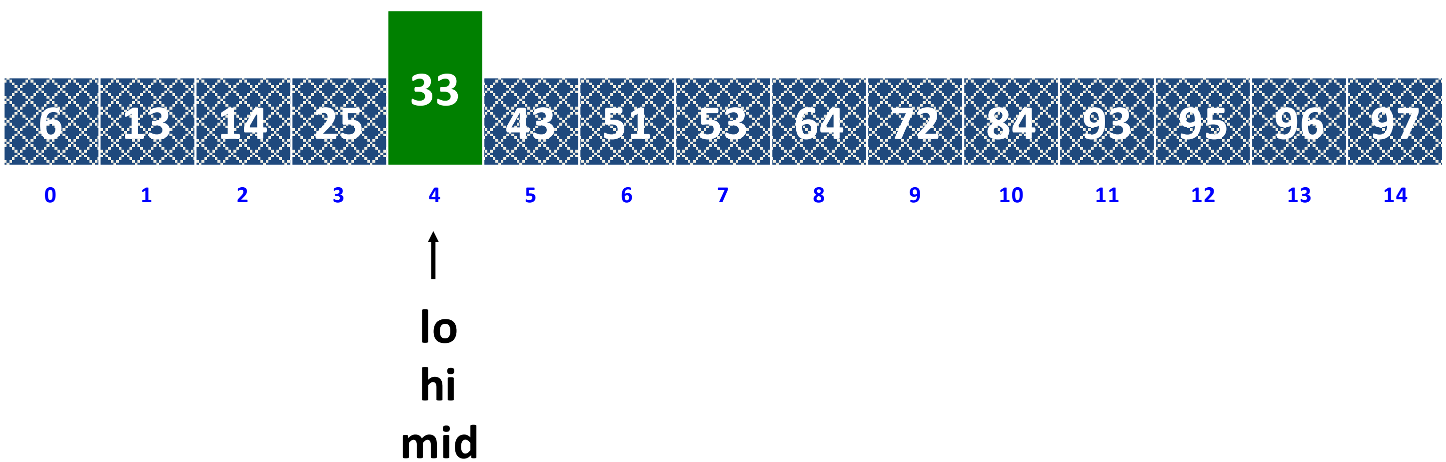
Binary Search Example

- Ex. Binary search for 33.



Binary Search Example

- Ex. Binary search for 33.



Binary Search Example

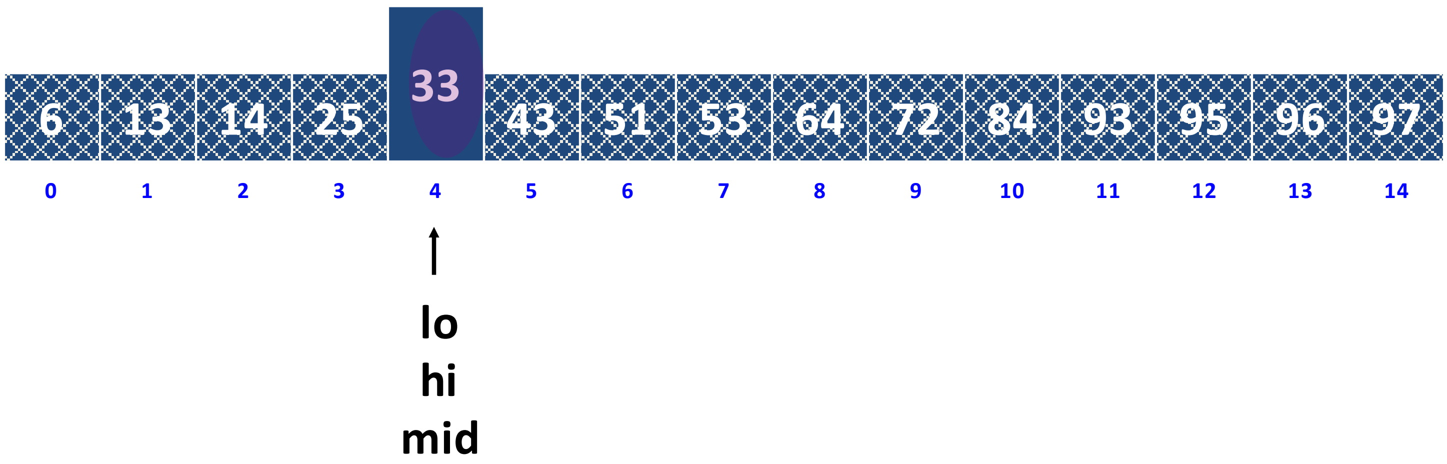
- Ex. Binary search for 34.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑														↑
lo														**hi**

Similar to the previous solving up to iteration 4

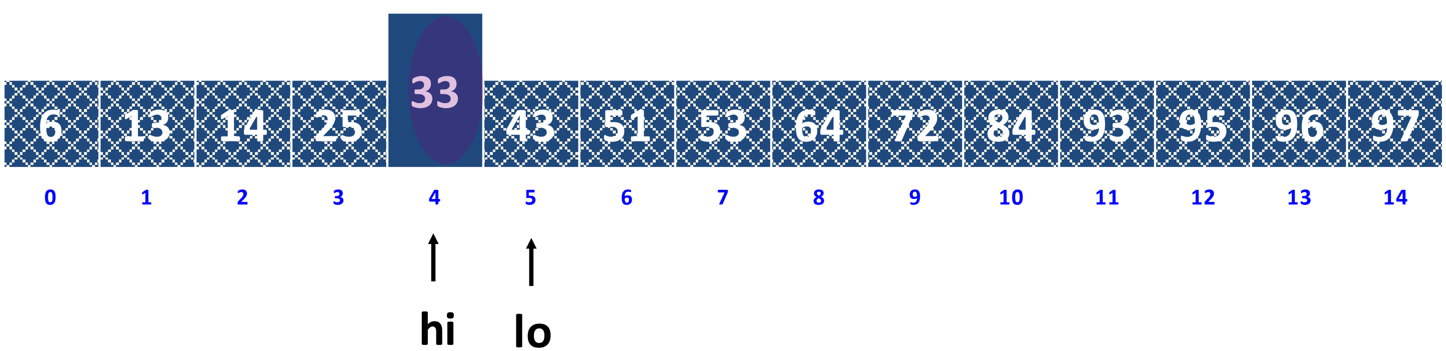
Binary Search Example

- Ex. Binary search for 34.



Binary Search Example

- Ex. Binary search for 34.



In array, boundaries should always be low <= high;

Here Lo <= hi failed. Hence stop the process. Return -1 to indicate element not existed in the array.

Binary Search Implementation

- Binary Search can be implemented in two ways:
  - Iterative approach
  - Recursive approach

Iterative Binary Search

```
Algorithm Iterative Binary Search(int a[], int low, int high, int key)
{
    while (low <= high) do
    {
        middle = low+(high-low) / 2;

          if (key == a[middle]) then
               return middle;

         else if (key < a [middle]) then
                high = middle -1;     // search low end of array
               else
                low = middle + 1;     // search high end of array
    }
    return -1;
}
```

Recursive Binary Search

```
Algorithm Recursive Binary Search(int a[], int low, int high, int key)
{
    if (low <= high) then
    {
        int mid = low + (high - low) / 2;
        if (a[mid] == key)
            return mid;

        if (key < a[mid]) then
            return Recursive Binary Search(a, low, mid - 1, key);
        else
            return Recursive Binary Search(a, mid + 1, high, key);
    }
  return -1;
}
```

Pro's & Con's of Binary Search

Advantages:

- Compared to linear search binary search is *much* faster.
- Linear search takes, on average N/2 comparisons and worst case N comparisons. Binary search takes an average and worst-case $\log_2(N)$comparisons.
- It's a fairly simple algorithm to implement.
- It's well known and often implemented for you as a library routine.

Disadvantages:

- It works only on lists that are sorted and kept sorted. That is not always feasible, especially if elements are constantly being added to the list.
- It employs recursive approach which requires more stack space.
- Works only when the data is in an array.

Best & Worst Case in Binary Search

- In the best case, where the target value is the middle element of the array, its position is returned after one iteration.
- In the worst case, binary search makes $\log_2(n)$ iterations. This is because the worst case is reached when the search reaches the deepest level.
- The worst case may also be reached when the target element is not in the array.

Time Complexity of Binary Search

$T(n) = 4 + T(n/2)$

$T(n/2) = 4 + T(n/4)$

$T(n) = 4 + (4+T(n/4))$

$= 2*4 + T(n/4)$

$T(n) = 2*4 + (4+T(n/8)$

$= 3*4 + T(n/8)$

$T(n) = 4*4 + T(n/16)$

…………

$T(n) = i*4 + T(n/2^i)$

$= i*4 + T(1)$

$= \log n * 4 + 2$

$=O(\log n)$

Assume $n = 2^i$

$\log n = \log 2^i$

$\log n = i$

Summary

- Linear Search is sequential(element by element comparison), hence better to use if the data size is small and search is only one time.

- Binary Search is better is the data is sorted and no more insertions are done and search is frequent operation.

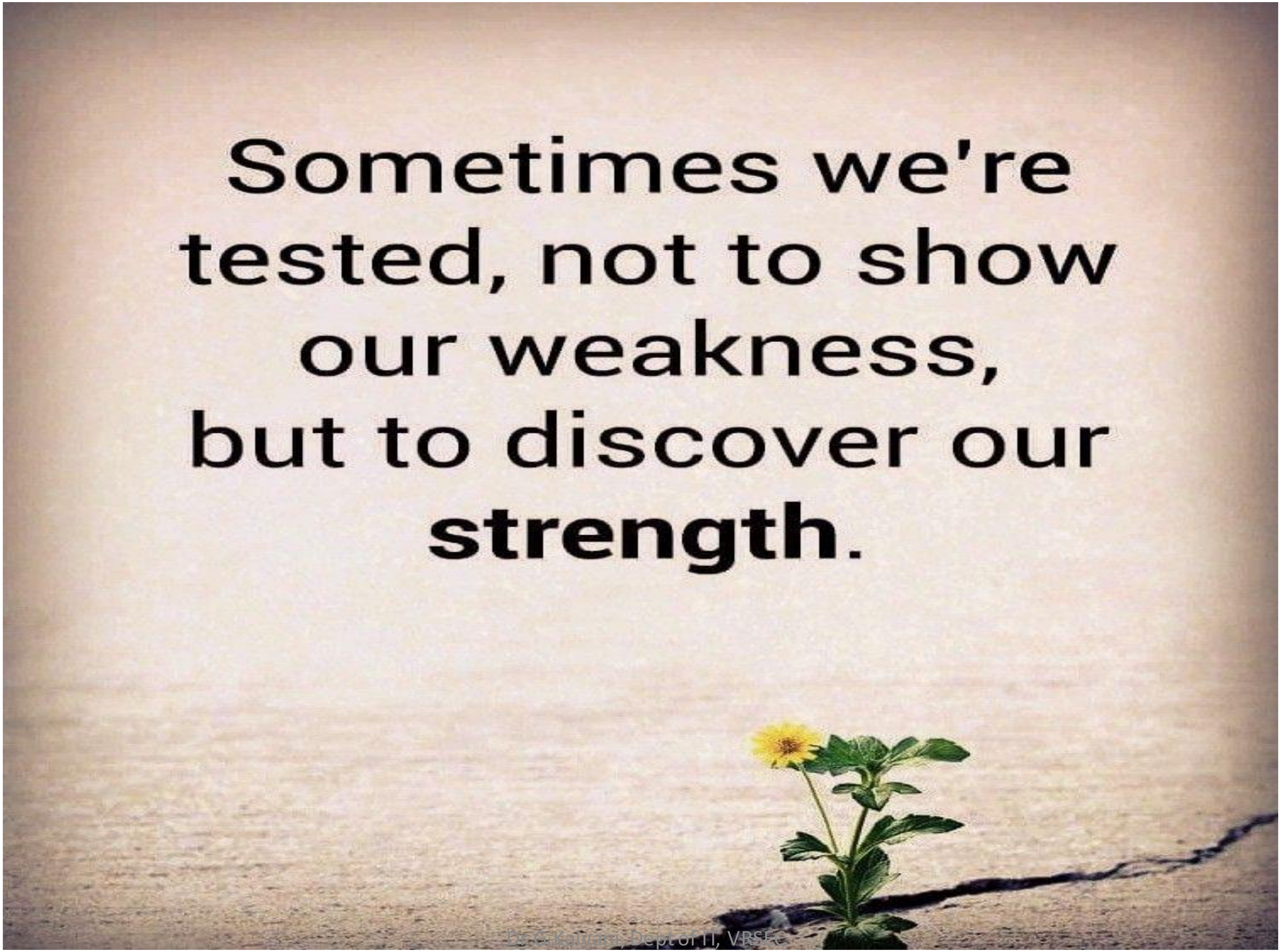
Task for work

- Illustrate step by step process of linear and binary search by considering the following elements.

  53,19,78,23,45,68,9,41,78,1,19,94,12,49,83,8

  – Q1: Linear search for 78
  – Q2: Linear Search for 2
  – Q3: Binary Search for 41
  – Q4: Binary Search for 94
  – Q5: Binary Search for 2

- Download **Kahoot** App from play store.


Sometimes we're
tested, not to show
our weakness,
but to discover our
strength.