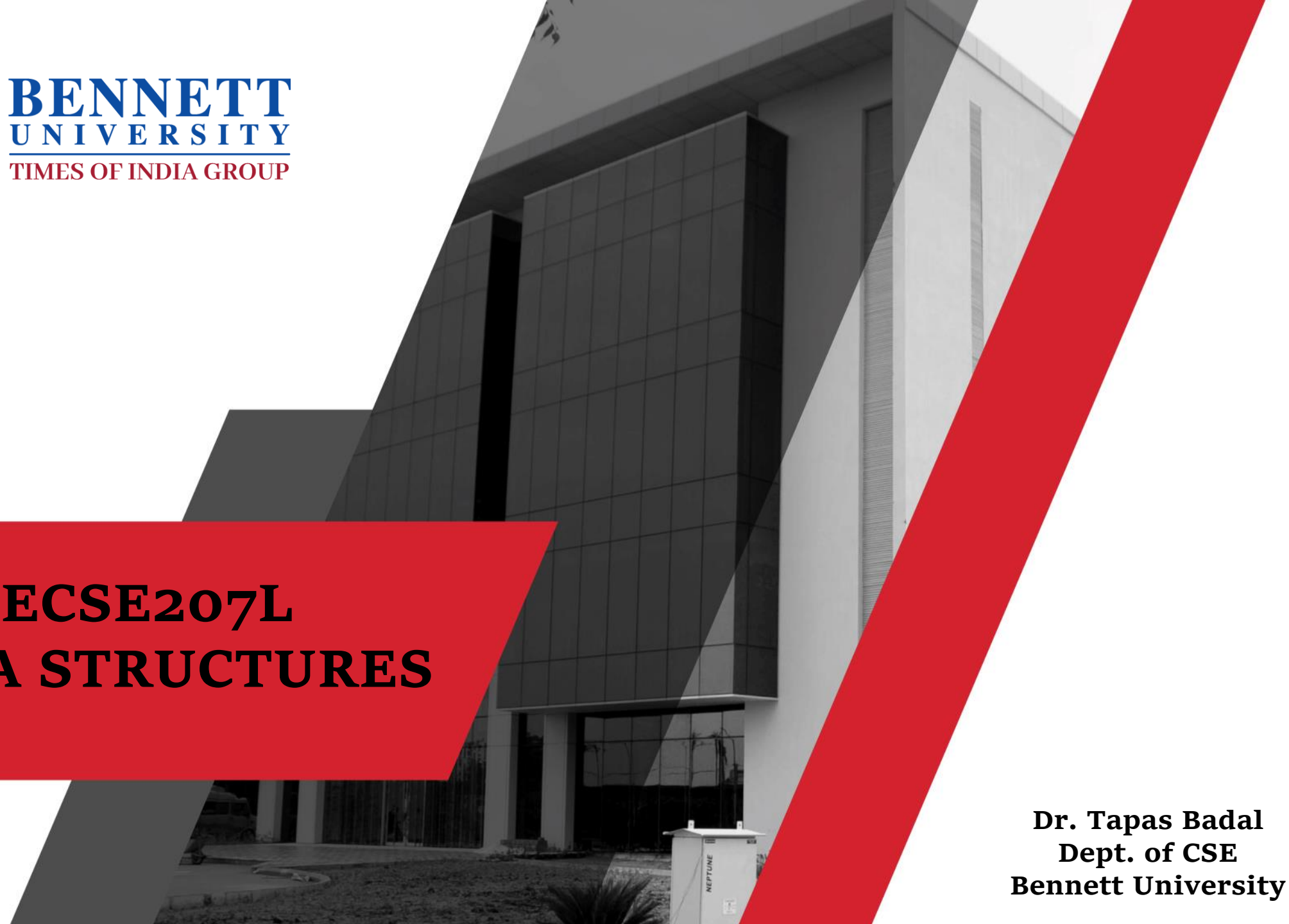
BENNETT UNIVERSITY
TIMES OF INDIA GROUP



ECSE207L
DATA STRUCTURES

Dr. Tapas Badal
Dept. of CSE
Bennett University

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

Linear & Binary Search

How to Find a Value in an Array?

-23	97	18	21	5	-86	64	0	-37

- Suppose you have a big array full of data, and you want to find a particular value.
- How will you find that value?

Linear Search

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

-23	97	18	21	5	-86	64	0	-37

- ***Linear search*** means looking at each element of the array, in turn, until you find the target value.

Linear Search Code

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

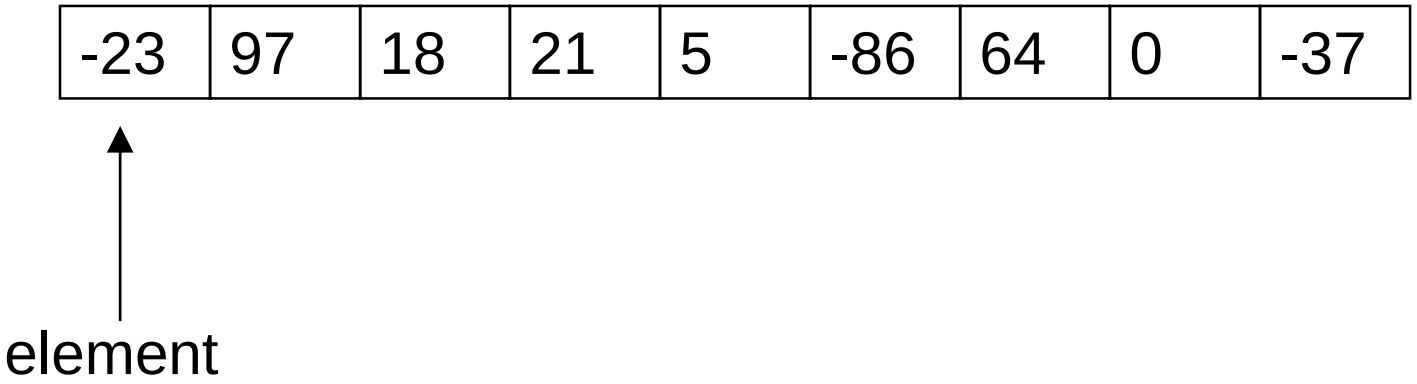
-23	97	18	21	5	-86	64	0	-37

```
// C++ code for linearly search x in arr[]. If x
// is present then return its location, otherwise
// return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```

```
// Java code for linearly search x in arr[]. If x
// is present then return its location, otherwise
// return -1
class LinearSearch
{
    // This function returns index of element x in arr[]
    static int search(int arr[], int n, int x)
    {
        for (int i = 0; i < n; i++)
        {
            // Return the index of the element if the element
            // is found
            if (arr[i] == x)
                return i;
        }

        // return -1 if the element is not found
        return -1;
    }
}
```

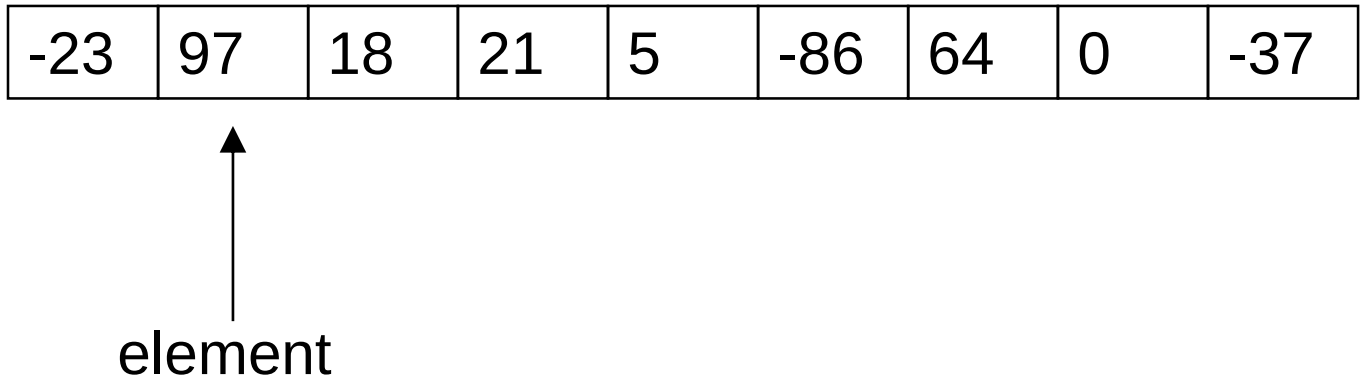
```
# Python code for linearly search x in arr[]. If x
# is present then return its location, otherwise
# return -1
def search(arr, x):

    for i in range(len(arr)):

        if arr[i] == x:
            return i

    return -1
```

Linear Search Example #1



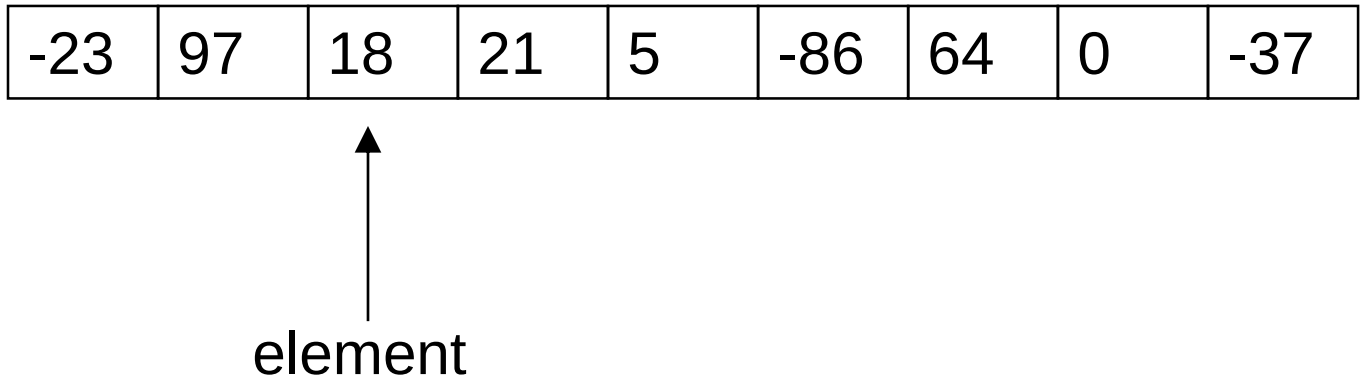
Searching for -86.

Linear Search Example #1



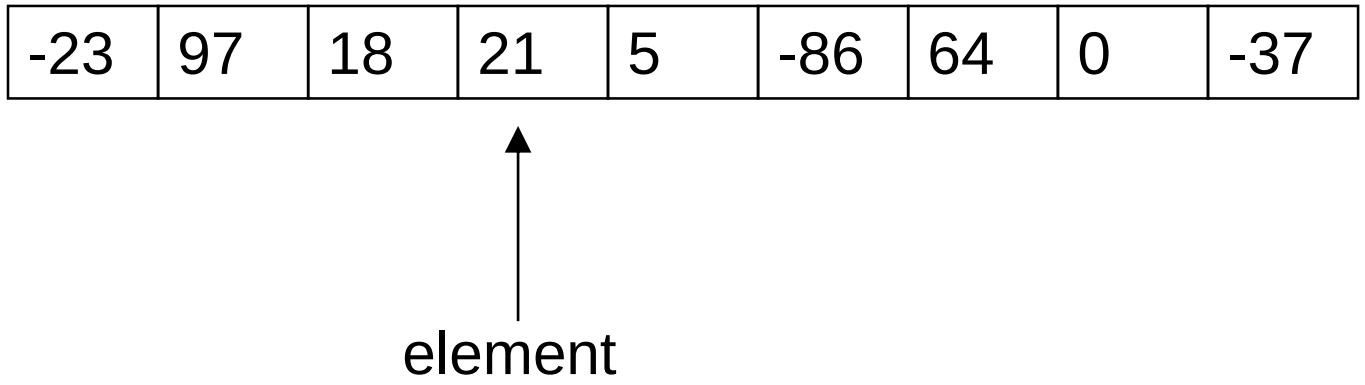
Searching for -86.

Linear Search Example #1



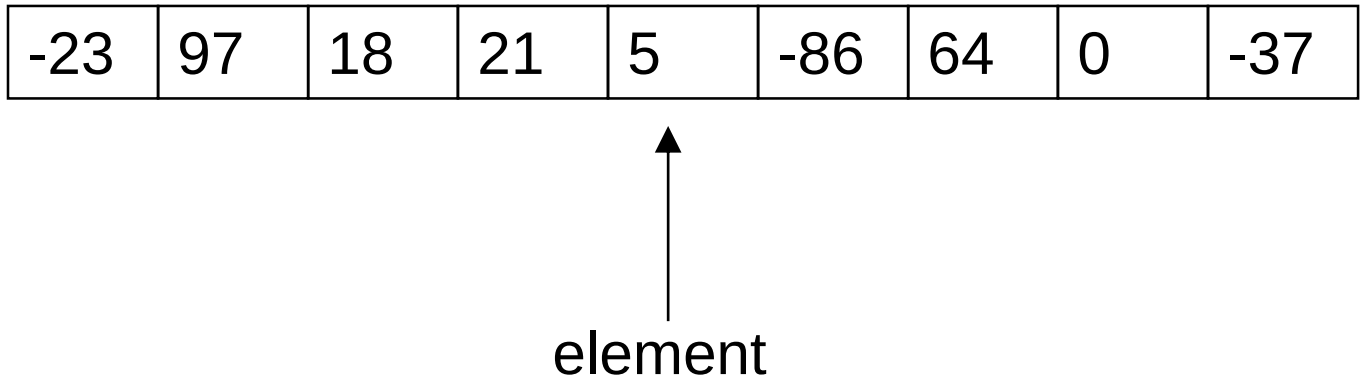
Searching for -86.

Linear Search Example #1



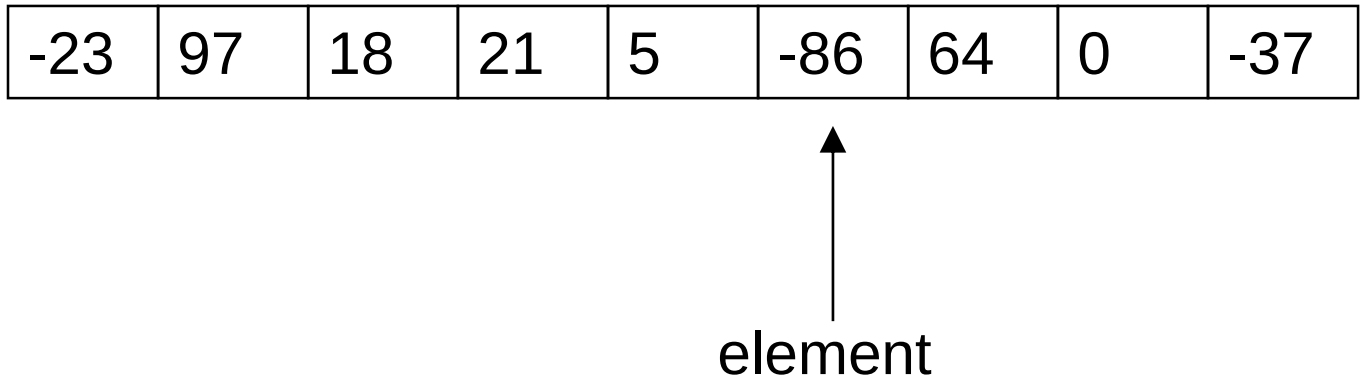
Searching for -86.

Linear Search Example #1



Searching for -86.

Linear Search Example #1



Searching for -86: found!

How Long Does Linear Search Take?

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

- Okay, great, now we know how to search.
- But how long will our search take?
- That's really three separate questions:
- How long will it take in the **best case**?
- How long will it take in the **worst case**?
- How long will it take in the **average case**?

Linear Search: Best Case

- How long will our search take?
- In the **best case**, the target value is in the first element of the array.
- So the search takes some tiny, and constant, amount of time.
- This **O(1)** .
- In real life, we don't care about the best case, because it so rarely actually happens.

Linear Search: Worst Case

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

- How long will our search take?
- In the **worst case**, the target value is in the last element of the array.
- So the search takes an amount of time proportional to the length of the array.
- **O(*n*).**

Linear Search: Average Case

- How long will our search take?
- In the **average case**, the target value is somewhere in the array.
- In fact, since the target value can be anywhere in the array, any element of the array is equally likely.
- So on average, the target value will be in the middle of the array.
- So the search takes an amount of time proportional to half the length of the array – also proportional to the length of the array – **O(*n*)** again!

Why Do We Care About Search Time?

- We know that time is money.
- So finding the fastest way to search (or any task) is good, because then we'll save time, which saves money.

Linear Search is O(*n*) in the Average Case

- Recapping, linear search is **O**(*n*) in the average case.
- But what if we expect to do lots of searches through our dataset of length ***n***?
- What if we expect to do ***n*** searches on our ***n*** data?
- Well, the time complexity will be ***n*O(*n*)**, which is to say **O(n^2).**
- You can imagine that, when ***n*** is big – a million, a billion, etc – this is terribly inefficient.
- Can we do better?

A Better Search?

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

- Consider how you search for someone in the phone book – say, **Bennett**.
- You start with the first letter of their last name, B.
- You guess roughly where B would be in the phonebook.
- You open to that page.
- If you're wrong, you move either forward or backward in the book – that is, if you actually opened to J, you move backward, but if you opened to A, you move foreword.
- You keep repeating this action until you find **Bennett**.

Faster Search Requires Sorted Data

- Why not use linear search?
- Linear search means start at the beginning, and look at every piece of data until you find your target.
- This is much slower than the way you search a phonebook in real life.
- Why?
- The reason you can do the phonebook search so quickly is because the names in the phonebook are **sorted** – specifically, they're in alphabetical order by last name, then by first name.

Binary Search

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

The general term for a smart search through sorted data is a ***binary search***.

1. The initial search region is the whole array.
2. Look at the data value in the middle of the search region.
3. If you've found your target, stop.
4. If your target is less than the middle data value, the new search region is the lower half of the data.
5. If your target is greater than the middle data value, the new search region is the higher half of the data.
6. Continue from Step 2.

```
int binarySearch(int low,int high,int key)
{
        while(low<=high)
        {
                int mid=(low+high)/2;
                if(a[mid]<key)
                {
                        low=mid+1;
                }
                else if(a[mid]>key)
                {
                        high=mid-1;
                }
                else
                {
                        return mid;
                }
        }
        return -1;   //key not found
}
```

Binary Search Code

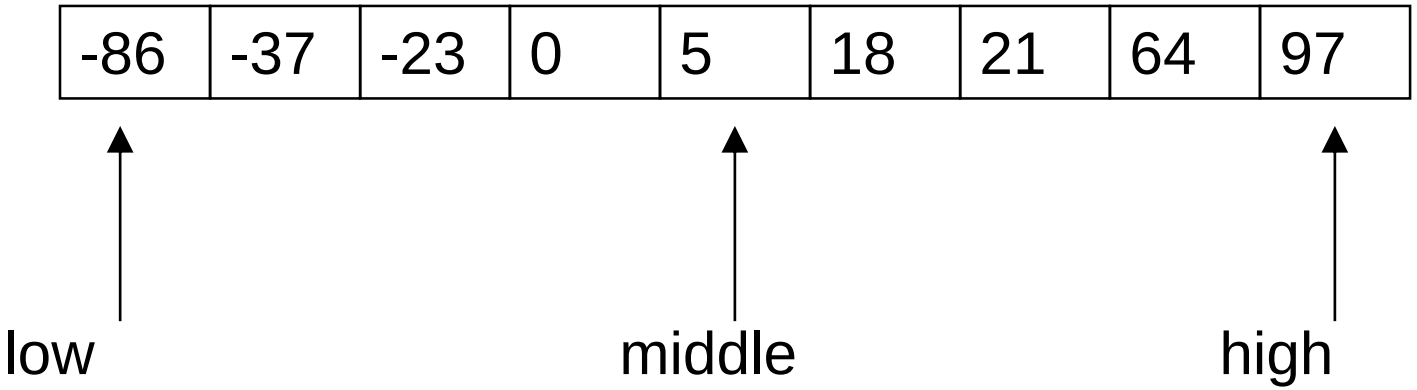
BENNETT UNIVERSITY
TIMES OF INDIA GROUP

```c
// 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;
}
```

```java
class BinarySearch
{
    // Returns index of x if it is present in arr[l..
    // r], else return -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
```

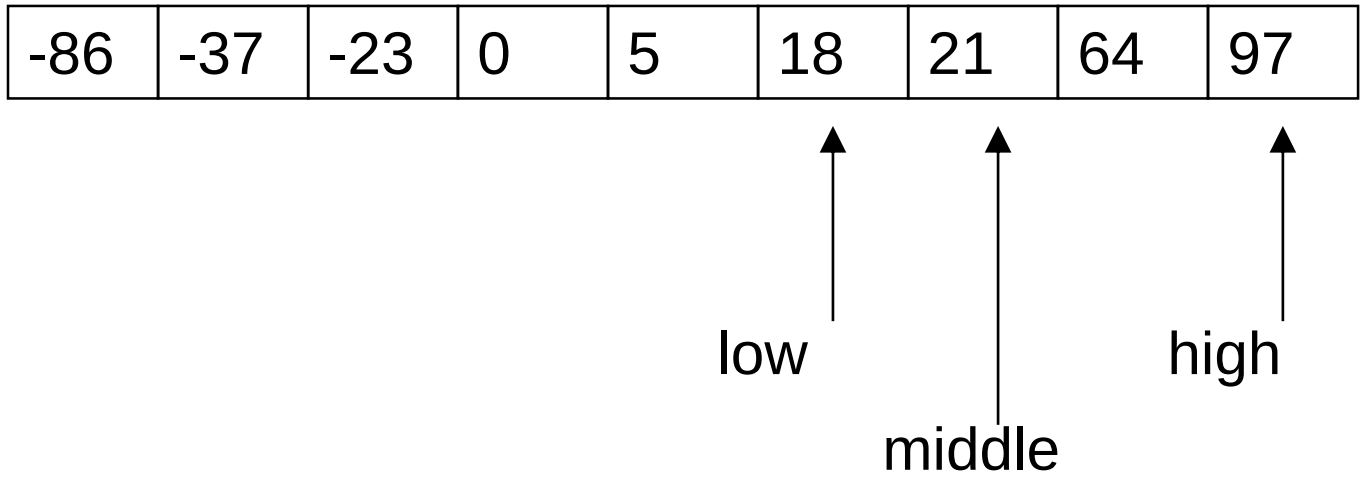
```python
# Python Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l)/2

        # If 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
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid+1, r, x)

    else:
        # Element is not present in the array
        return -1
```

Binary Search Example #1



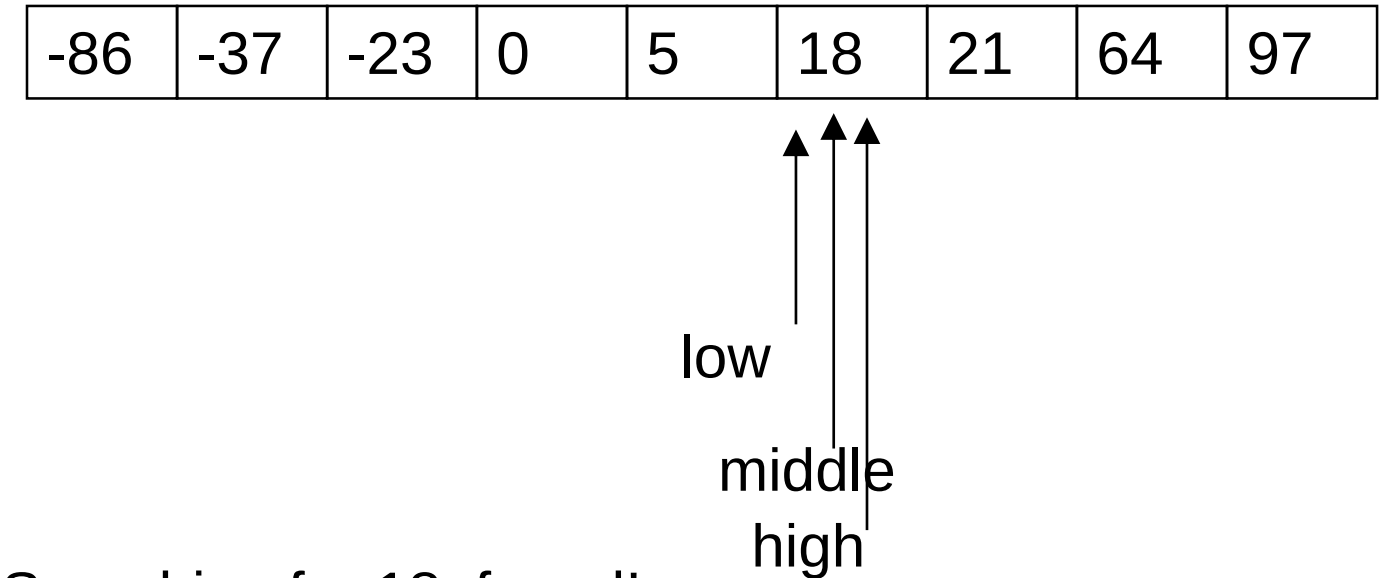
Searching for 18.

Binary Search Example #1



Searching for 18.

Binary Search Example #1



Searching for 18: found!

Time Complexity of Binary Search #1

- How fast is binary search?
- Think about how it operates: after you examine a value, you cut the search region in half.
- So, the first iteration of the loop, your search region is the whole array.
- The second iteration, it's half the array.
- The third iteration, it's a quarter of the array.
- ...
- The k^{th} iteration, it's $(1/2^{k-1})$ of the array.

Time Complexity of Binary Search #2

- How fast is binary search?
- For the k^{th} iteration of the binary search loop, the search region is $(1/2^{k-1})$ of the array.
- What's the maximum number of loop iterations?
  - $\lceil \mathbf{log_2 n} \rceil$
- That is, we can't cut the search region in half more than that many times.
- So, the time complexity of binary search is **O(log$_2$*n*).**

Time Complexity of Binary Search #3

- How fast is binary search?
- We said that the time complexity of binary search is **O**($\log_2 n$).
- But, **O**($\log_2 n$) is exactly the same as **O**($\log n$).
- Why?
- Well, we know from math class that
  - $\log_a x \equiv \log_b x \,/\, \log_b a$
- So the relationship between logs to various bases is simply a constant, $(1 / \log_b a)$.
- Therefore, **O(log *n*)** is the same as $\mathbf{O}(\log_b n)$ for any base b – we don't care about the base.

Need to Sort Array

- Binary search only works if the array is already sorted.
- It turns out that sorting is a huge issue in computing – and the subject of remaining lecture.

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

THANKYOU

@csebennett

cse_bennett