Searching Algorithms

Mark Matthews PhD

Summary

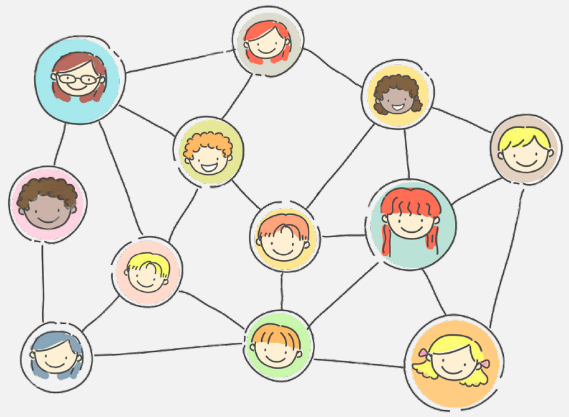
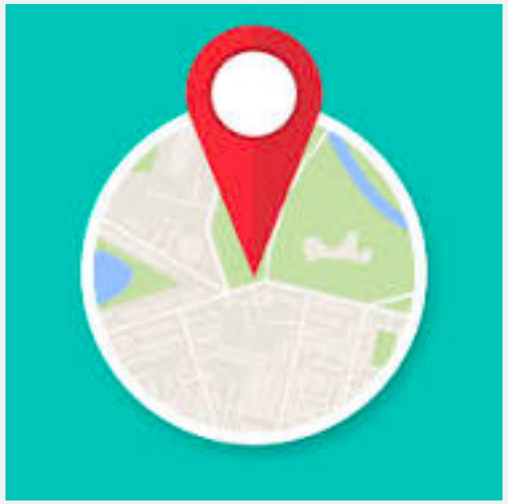
- Search Algorithms
- Search Applications
- Sorted and unsorted approaches to Search
- Linear Search
- Binary Search
- Miscellaneous Searches

Search Problem Defined

We consider a search algorithm to be an algorithm that retrieves information stored within some input (i.e. a data structure)

Search Applications

- Modern computing has access to vast amounts of information
- Efficient search and retrieval of this information is therefore fundamental to practical computing
- Applications include the following but many more:

Today: Finding an element

- Find an element in a data structure:
  - Unsorted search
  - Sorted search

- Retrieving a record from a database
- Checking if a value is present in a dataset
- Fundamental to most computer applications

How to Find a Value in an Array?

23	97	18	21	5	86	64	0	37

Linear Search

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

23	97	18	21	5	86	64	0	37

Linear Search Pseudo Code

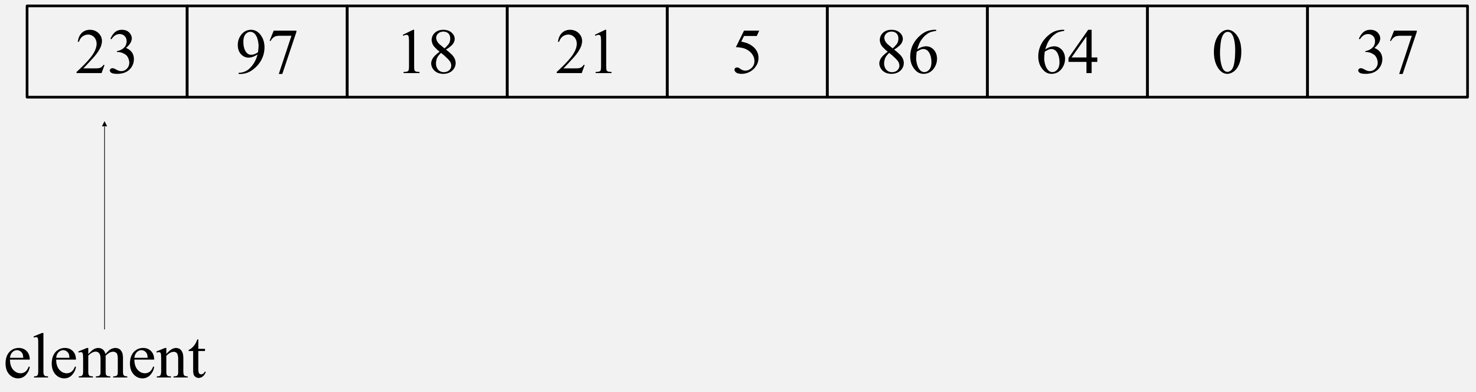
```
int search (int arr[], int x){
for (int I = 0; I < arr.length; i++){
if (arr[i] = x){
 return i;
}

//if x isn't there
return -1;
}
```

```java
// Java code for linearly searching x in arr[]. If x is present then //return its location, otherwise  return -1

class Searches {
public static int linearSearch(int arr[], int x)
{
    int n = arr.length;
    for(int i = 0; i < n; i++)
    {
        if(arr[i] == x)
            return i;
    }
    return -1;
}

public static void main(String args[])
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;

    int result = linearSearch(arr, x);
    if(result == -1)
        System.out.print("Element is not present in array");
    else
        System.out.print("Element is present at index " + result);
}
}
```

Linear Search: sample inputs / output

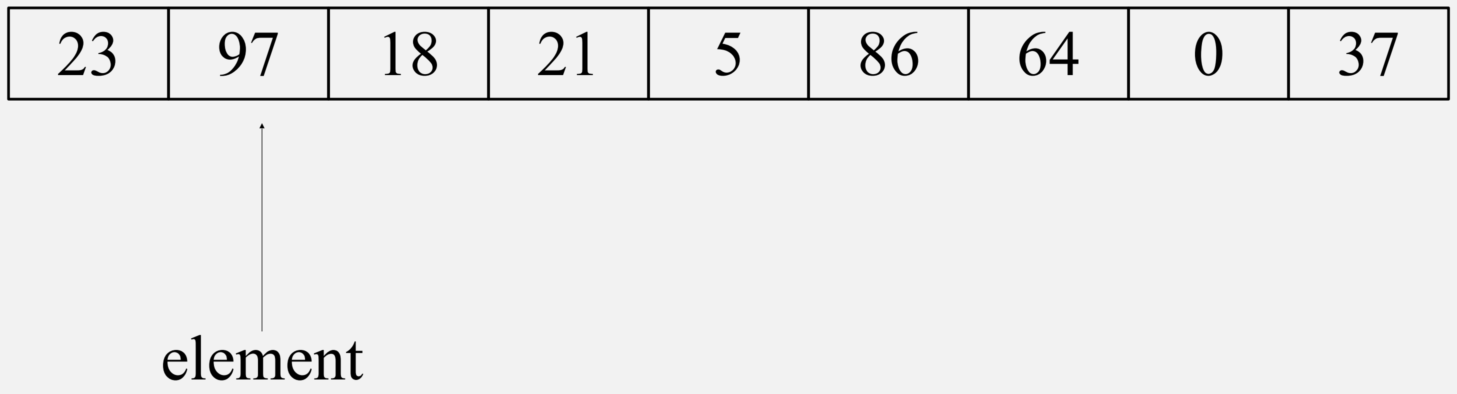
```
Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
         x = 110;
Output : 6
Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
          x = 195;
Output : -1
Element x is not present in arr[].
```

Linear Search Example #1



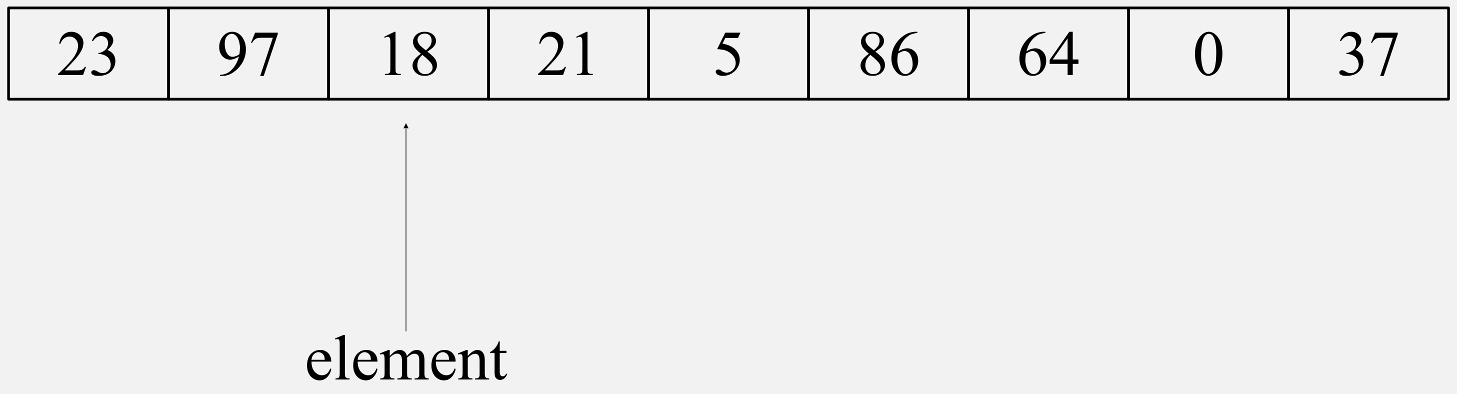
Searching for 86.

Linear Search Example #2



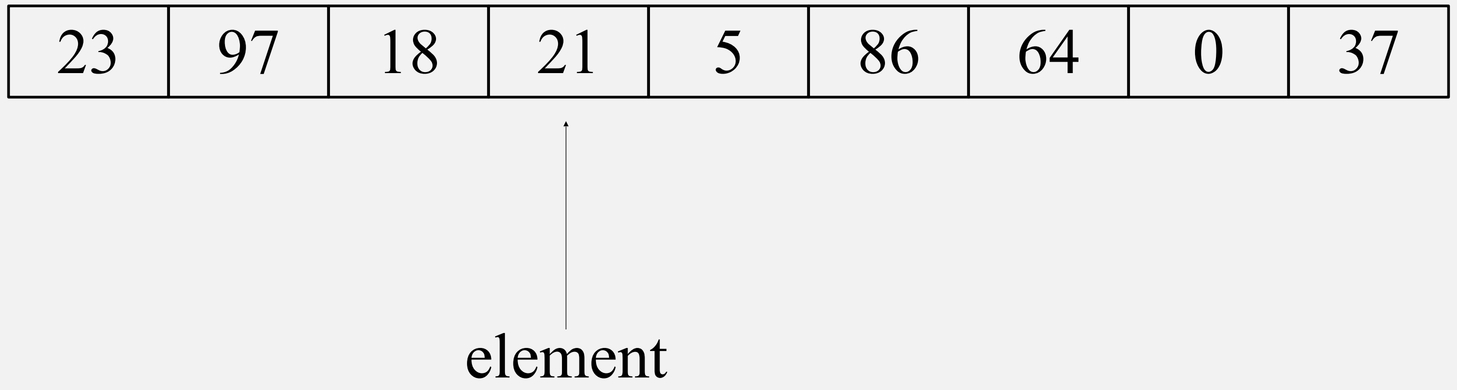
Searching for 86.

Linear Search Example #3



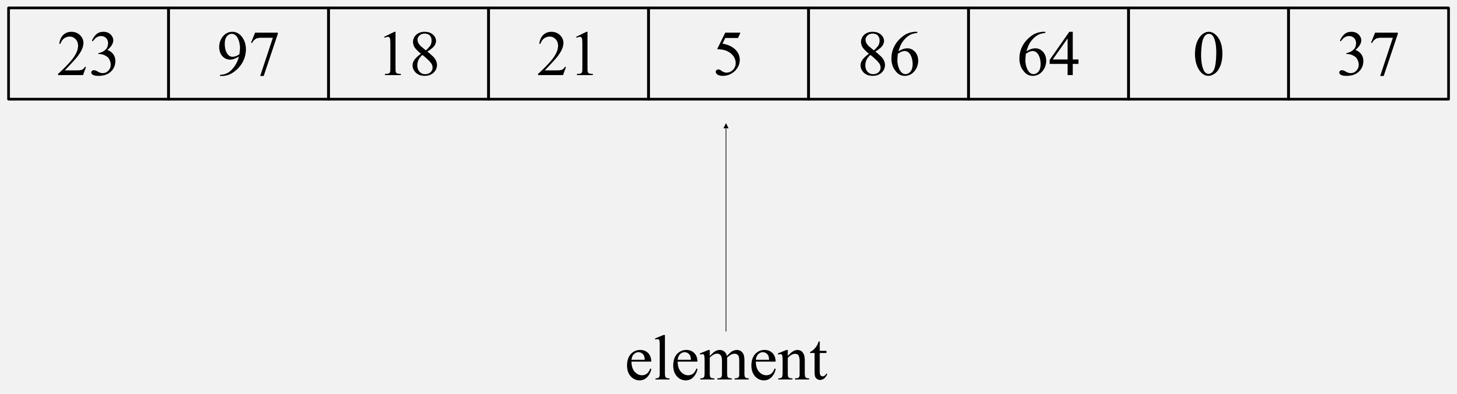
Searching for 86.

Linear Search Example #4



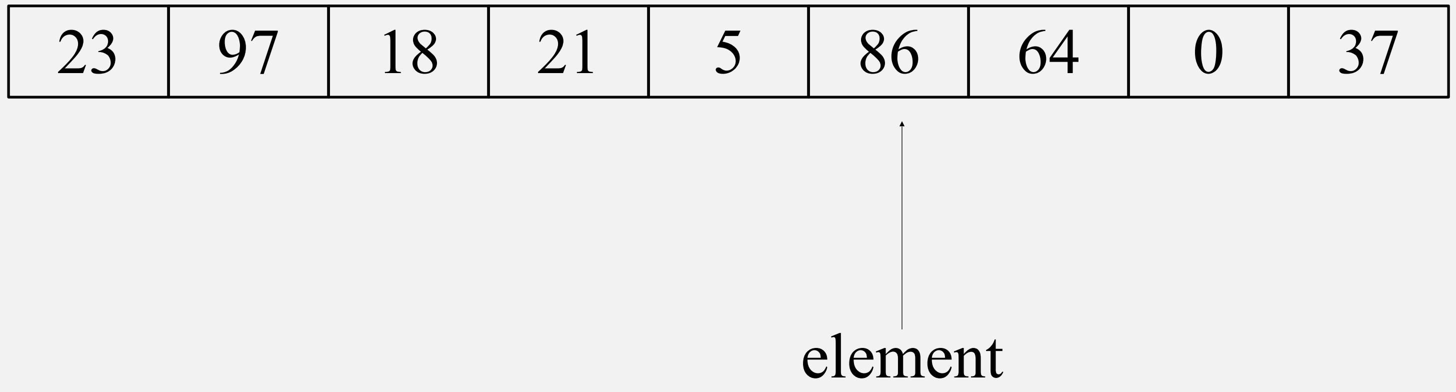
Searching for 86.

Linear Search Example #5



Searching for 86.

Linear Search Example #6



Searching for 86.

Found!

Return index: 5

Brute Force Algorithms

Linear search is a **brute force algorithm**

Brute force algorithms:
- General algorithmic strategy
- Systematically enumerate all possible solution candidates

Advantages?
- Often simple to implement
- Generate entire solution set

Disadvantages?
- Not efficient - usually does more work than needed

Trade-offs?
- Will give you an optimal solution but (usually) with poor efficiency

Brute Force Algorithms: Real world example

Linear search is a **brute force algorithm**

4 digits
1 of 10 spaces (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

10,000 combinations

Try each combination until you find the correct one.

Depending on the use case it might make sense.

For many real world problems, the search space is too large

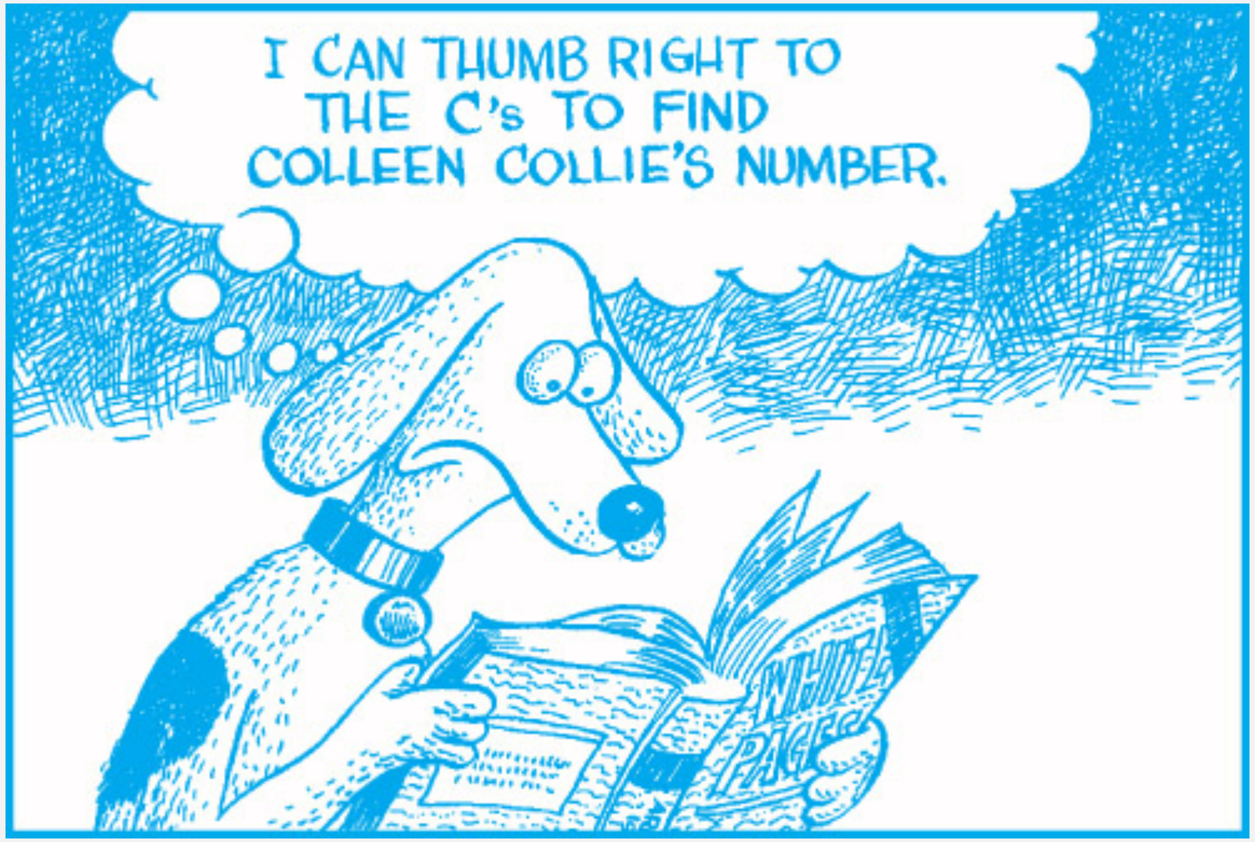
This is the basis of modern cryptography where key length is designed to make brute-force attacks unfeasible

How Long Does Linear Search Take?

Sorted or Unsorted Input?	Unsorted
Time Complexity	
Worst	O(N)
Best	O(1)
Average	O(N
Algorithm Approach	Brute Force

For large collections of records that are searched repeatedly, sequential search is unacceptably slow.

A Better Search? Requires structured data

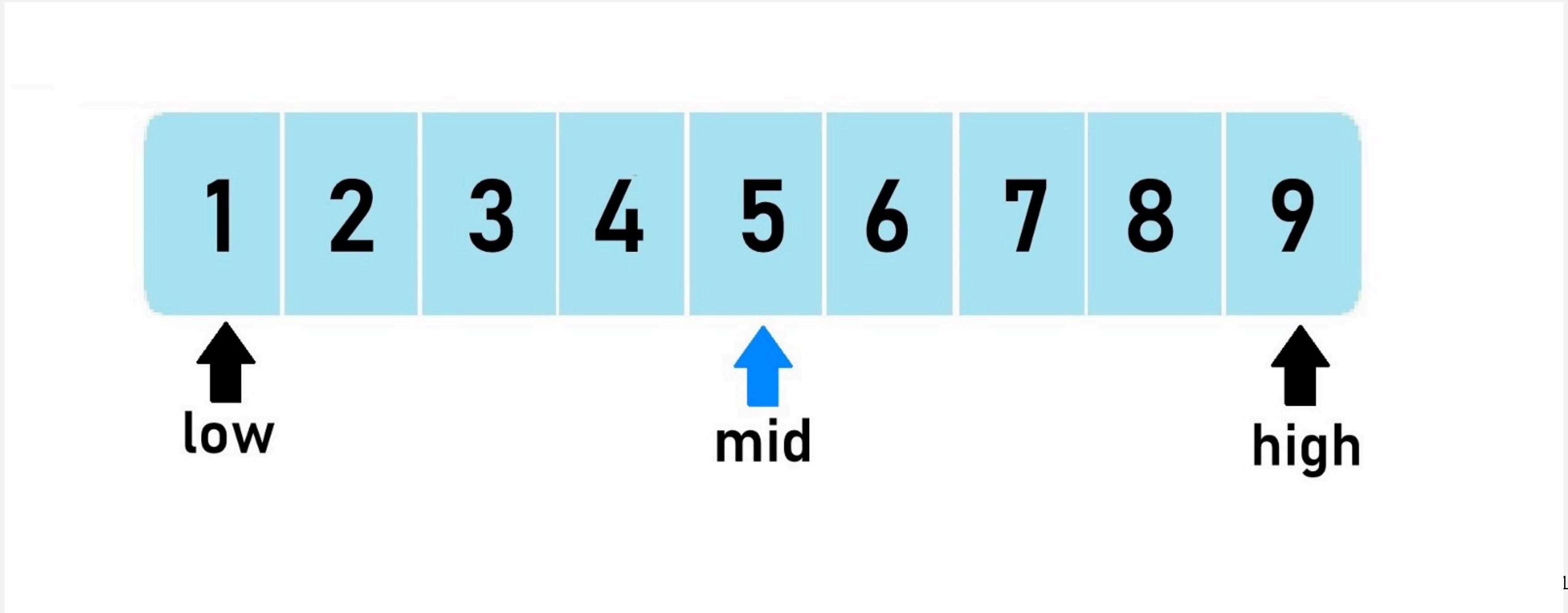

I CAN THUMB RIGHT TO THE C's TO FIND COLLEEN COLLIE'S NUMBER.
WHITE PAGES

The appropriate search algorithm often depends on the data structure being searched, and may also include prior knowledge about the data. Some database structures are specially constructed to make search algorithms faster or more efficient, such as a search tree, hash map, or a database index.

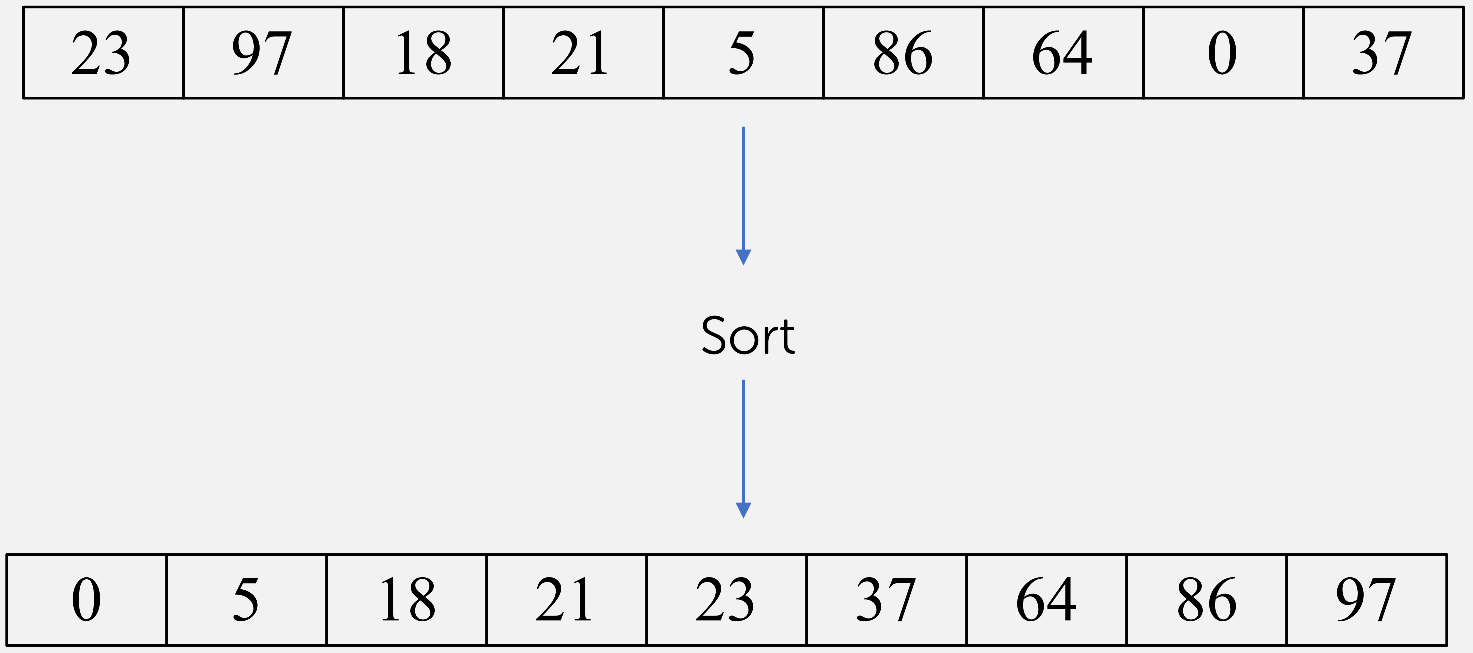
Binary Search

Binary search is a classic algorithm for finding an element in sorted input

1. The initial search region is the whole array.
2. Find the middle = start + end divided by 2
3. If you've found your target, stop you're done.
4. If your target is less than the middle 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. Repeat from Step 2.



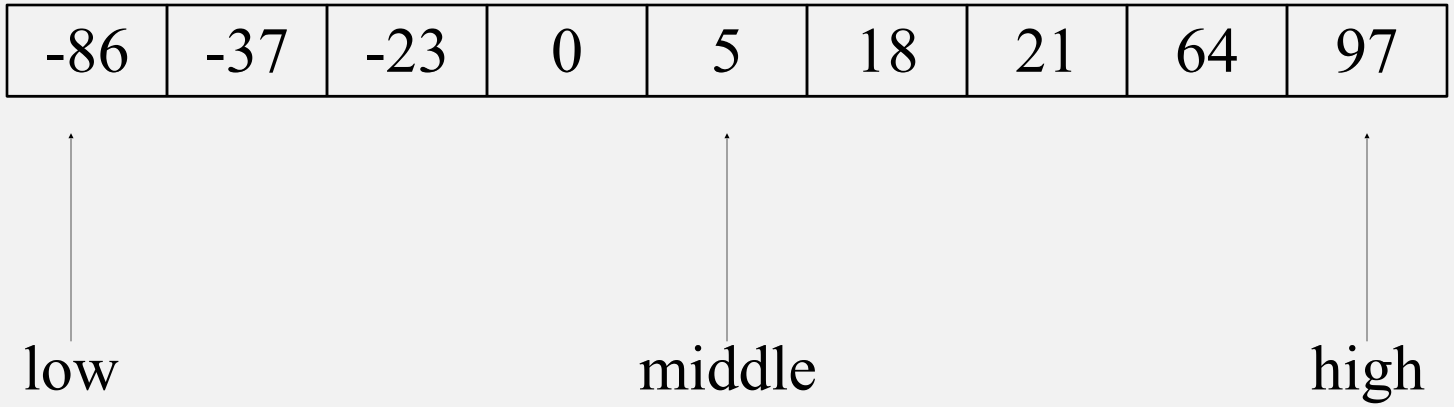
Binary Search



Binary Search Code

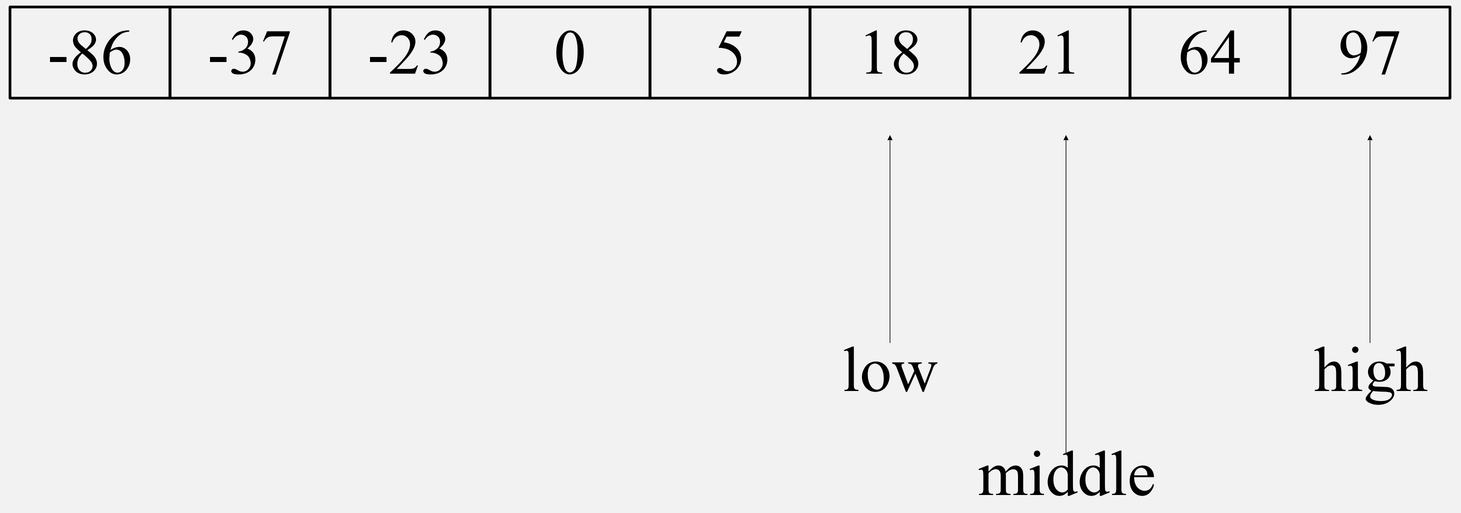
```
class BinarySearch {
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r >= l) {
            int mid = l + (r - l) / 2;

            // 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;
    }
```

Binary Search Example #1



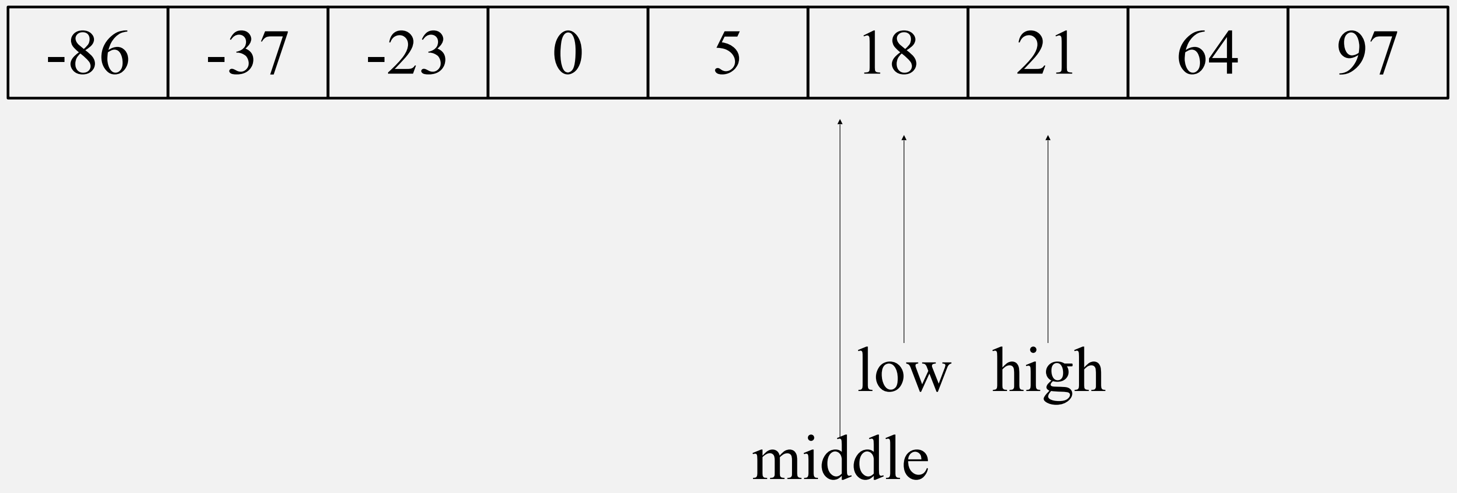
Searching for 18.

Binary Search Example #2



Searching for 18.

Binary Search Example #3



Searching for 18: found!

Binary Search Sample problem

Suppose we want to know whether the number 67 is prime. If 67 is in the array, then it's prime.

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97];
```

How Long Does Binary Search Take?

Ordered or Unordered Search?	Ordered
Time Complexity	
Worst	O log(N)
Best	O(1)
Average	O log(N)
Algorithm Approach	Divide & Conquer

Wait a minute - is Linear search not quicker when you account for the sort?

When the cost of sorting + binary search is less than linear search?

If I have a collection of N elements and I have to sort, have to look at each element at least once...

So binary search will ALWAYS be worse than linear, right?

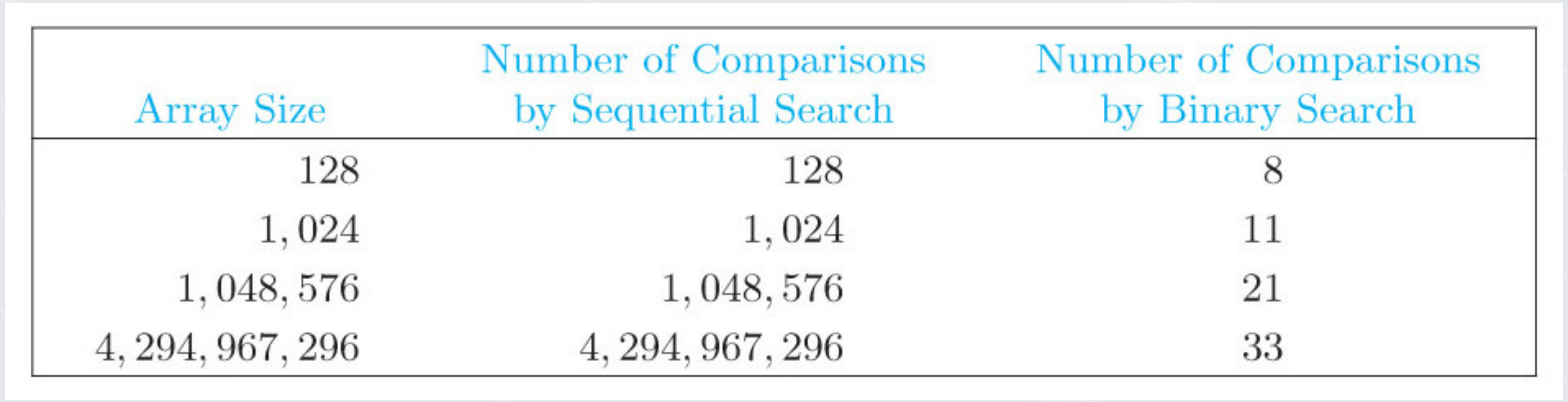
If you are doing one search, yes.
BUT if you are expecting to do multiple searches......
Amortize the cost:

Linear Search $T(N) = K*N$

Binary Search $T(N) = N\log(N) + K*\log(N)$ ←

The cost of the performance is dominated by the search.

"write once, read often" scenarios are quite common.



Array Size	Number of Comparisons by Sequential Search	Number of Comparisons by Binary Search
128	128	8
1, 024	1, 024	11
1, 048, 576	1, 048, 576	21
4, 294, 967, 296	4, 294, 967, 296	33

If the number of elements is very small, differences between search algorithms diminish.

Binary vs linear search rules of thumb

Input: unsorted array

Choice: Use linear search or sort the array

Decision: Depends on how often you perform the search, how often the input is updated etc.

Your approach also depends on the data structure and vice versa:
  e.g., binary search won't help if your data is in a linked list
  e.g., binary search tree effectively gives you binary search (unless unbalanced where can degenerate)

Miscellaneous searches: interpolation search

- **Input: sorted array**
- Discovered by W. W. Peterson in 1957
- Similar to telephone directory search
- Variant of binary search
- At each step algorithm calculates where in remaining space the sought item might be
- Uses linear interpolation usually
- If not there, then recalculates

- Only really works if size of key values are uniformly distributed
- In this case performance better than binary search = O log (log n)

Miscellaneous searches: interpolation search

```
Step 1 - Start searching data from middle of the list.
Step 2 - If it is a match, return the index of the item, and exit.
Step 3 - If it is not a match, probe position.
Step 4 - Divide the list using probing formula and find the new midle.
Step 5 - If data is greater than middle, search in higher sub-list.
Step 6 - If data is smaller than middle, search in lower sub-list.
Step 7 - Repeat until match.
```

```
// The idea of formula is to return higher value of pos
// when element to be searched is closer to arr[hi]. And
// smaller value when closer to arr[lo]
pos = lo + [ (x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo]) ]

arr[] ==> Array where elements need to be searched
x     ==> Element to be searched
lo    ==> Starting index in arr[]
hi    ==> Ending index in arr[]
```

Miscellaneous searches: jump search

- Input: sorted array

- Basic idea: check fewer elements than linear search by jumping ahead a by fixed number of steps

- Jump -> SQRT of N

- Example: array n = 16
- Optimal Jump size m = SQRT(16) = 4

Big O($\sqrt{}$ n)
The time complexity of Jump Search is between Linear Search ((O(n)) and Binary Search (O (Log n)).

```java
public static int jumpSearch(int[] arr, int x) {
        int n = arr.length;
        // Finding block size to be jumped
        int step = (int)Math.floor(Math.sqrt(n));

        // Finding the block where element is present (if it is present)
        int prev = 0;
        while (arr[Math.min(step, n)-1] < x)
        {
            prev = step;
            step += (int)Math.floor(Math.sqrt(n));
            if (prev >= n)
                return -1;
        }
        // Doing a linear search for x in block from prev
        while (arr[prev] < x)
        {
            prev++;
            // If we reached next block or end element is not present.
            if (prev == Math.min(step, n))
                return -1;
        }
        if (arr[prev] == x)
            return prev;
        return -1;
    }
```

Work out size of the jump

Find the block where the element is (or might be)

Doing a linear search

Searching summary

- Choose an appropriate search algorithm based on your specific use case
- This could determine the data structure you use
- Or if the data structure is already in place, the algorithm you choose
- Many data structures are constructed to make search faster (we'll encounter some shortly)
- Many miscellaneous search variants: e.g., exponential search
- Grover's algorithm are theoretically faster than linear or brute-force search even without the help of data structures or heuristics.