Data structures and Algorithms

Searching

Common Problems

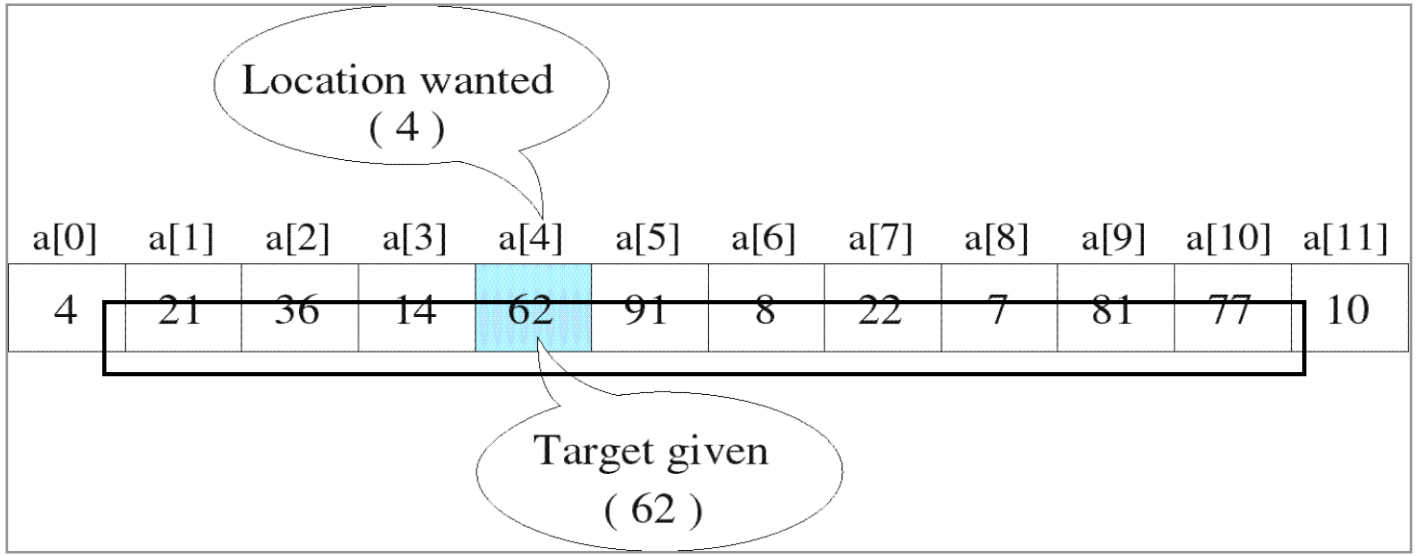
- There are some very common problems that we use computers to solve:
  - **Searching** through a lot of records for a specific record or set of records
  - Placing records in order, which we call **sorting**
- There are numerous algorithms to perform searches and sorts. We will briefly explore a few common ones.

Searching

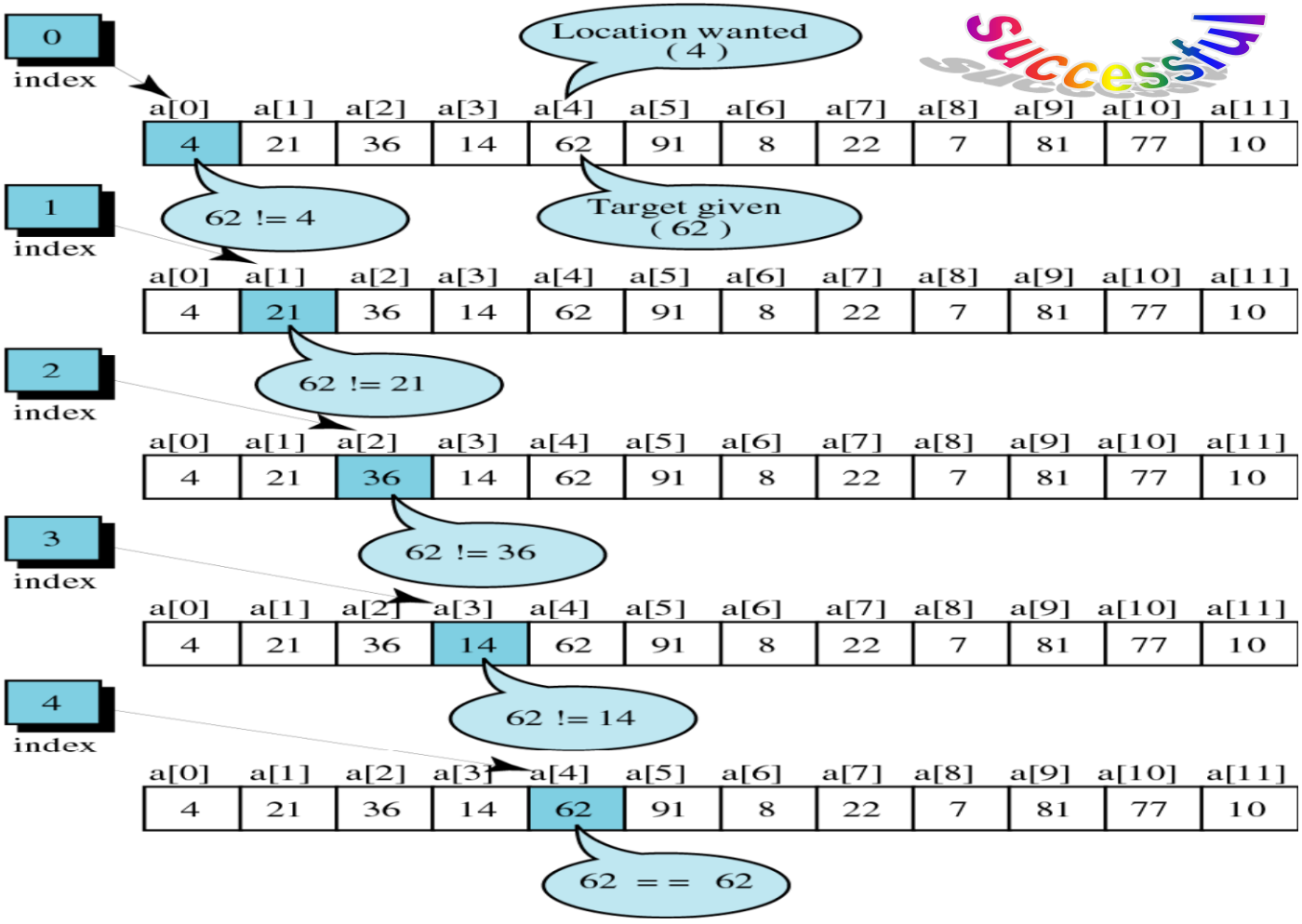
- A question you should always ask when selecting a search algorithm is "How fast does the search have to be?" The reason is that, in general, the faster the algorithm is, the more complex it is.
- Bottom line: you don't always need to use or should use the fastest algorithm.
- Let's explore the following search algorithms, keeping speed in mind.
  - Sequential (linear) search
  - Binary search

Searching

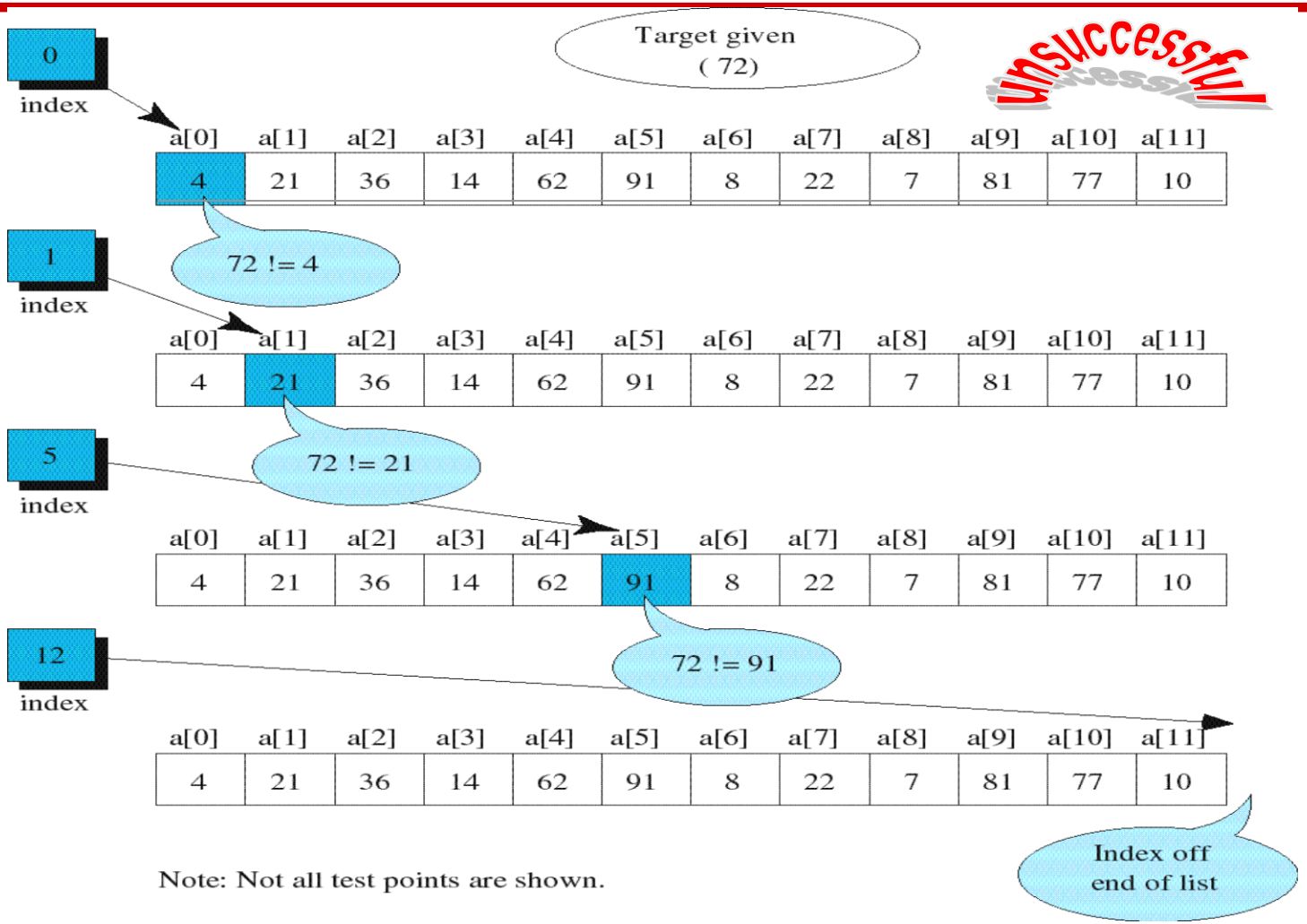
- The process used to find the location of a target among a list of objects
  - Searching an array finds the index of an element in an array containing that value



Searching


0
index
Location wanted
(4)
Successful
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
1
index
62 != 4
Target given
(62)
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
2
index
62 != 21
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
3
index
62 != 36
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
4
index
62 != 14
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 21 36 14 62 91 8 22 7 81 77 10
62 == 62

Searching



Note: Not all test points are shown.

Unordered Linear Search

- Search an unordered array of integers for a value and return its index if the value is found. Otherwise, return -1.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
14	2	10	5	1	3	17	2

Algorithm:

Start with the first array element (index 0)

while(more elements in array){

if value found at current index, return index;

Try next element (increment index);

}

Value not found, return -1;

Unordered Linear Search

```
// Searches an unordered array of integers
int search(int data[],  //input: array
           int size,    //input: array size
           int value){  //input: search value
    // output: if found, return index;
    //         otherwise, return -1.
    for(int index = 0; index < size; index++){
        if(data[index] == value)
            return index;
     }
    return -1;
}
```

Ordered Linear Search

- Search an ordered array of integers for a value and return its index if the value is found; Otherwise, return -1.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
1	2	3	5	7	10	14	17

- Linear search can stop immediately when it has passed the possible position of the search value.

Ordered Linear Search

- Algorithm:

```
Start with the first array element (index 0)
while(more elements in the array){
        if value at current index is greater than value,
                value not found, return -1;
        if value found at current index, return index;
        Try next element (increment index);
}
value not found, return -1;
```

Ordered Linear Search

```
// Searches an ordered array of integers
int lsearch(int data[],// input: array
            int size,  // input: array size
            int value  // input: value to find
            ) {       // output: index if found
        for(int index=0; index<size; index++){
              if(data[index] > value)
                    return -1;
              else if(data[index] == value)
                    return index;
        }
        return -1;
}
```

How Long Does Linear Search Take?

- Okay, great, now we know how to search.
- But how long will our search take?
- There is 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.
- Computer scientists denote this **O(1)** .
- In real life, we don't care about the best case, because it so rarely actually happens.

Linear Search: Worst Case

- 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.
- Computer scientists denote this **O(*n*)** – which we explained.

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.

Binary Search

- Binary search is based on the "divide-and-conquer" strategy which works as follows:
  - Start by looking at the middle element of the array
    1. If the value it holds is lower than the search element, eliminate the first half of the array from further consideration.
    2. If the value it holds is higher than the search element, eliminate the second half of the array from further consideration.
  - Repeat this process until the element is found, or until the entire array has been eliminated.

Binary Search

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.

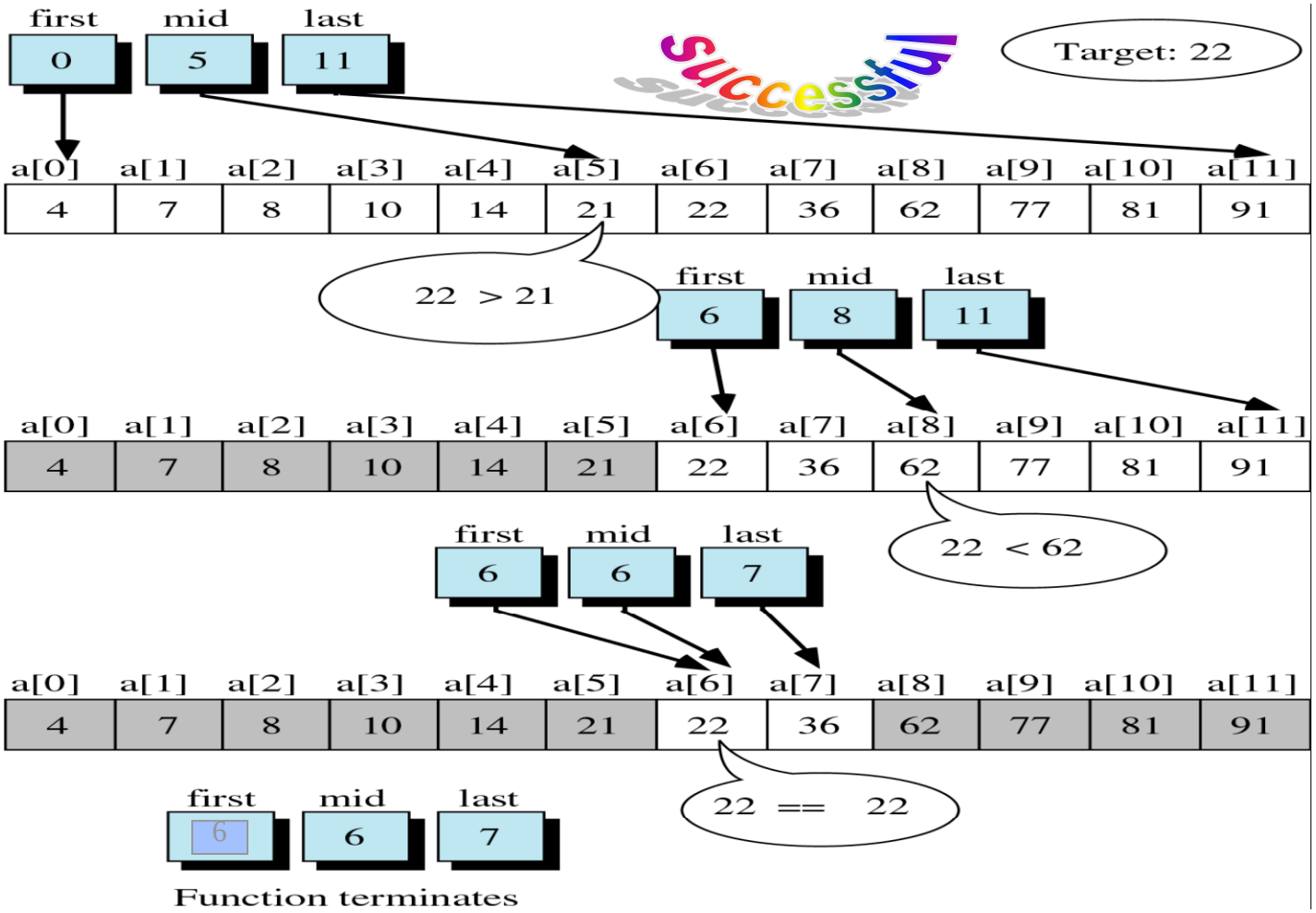
Binary Search

- Search an ordered array of integers for a value and return its index if the value is found. Otherwise, return -1.

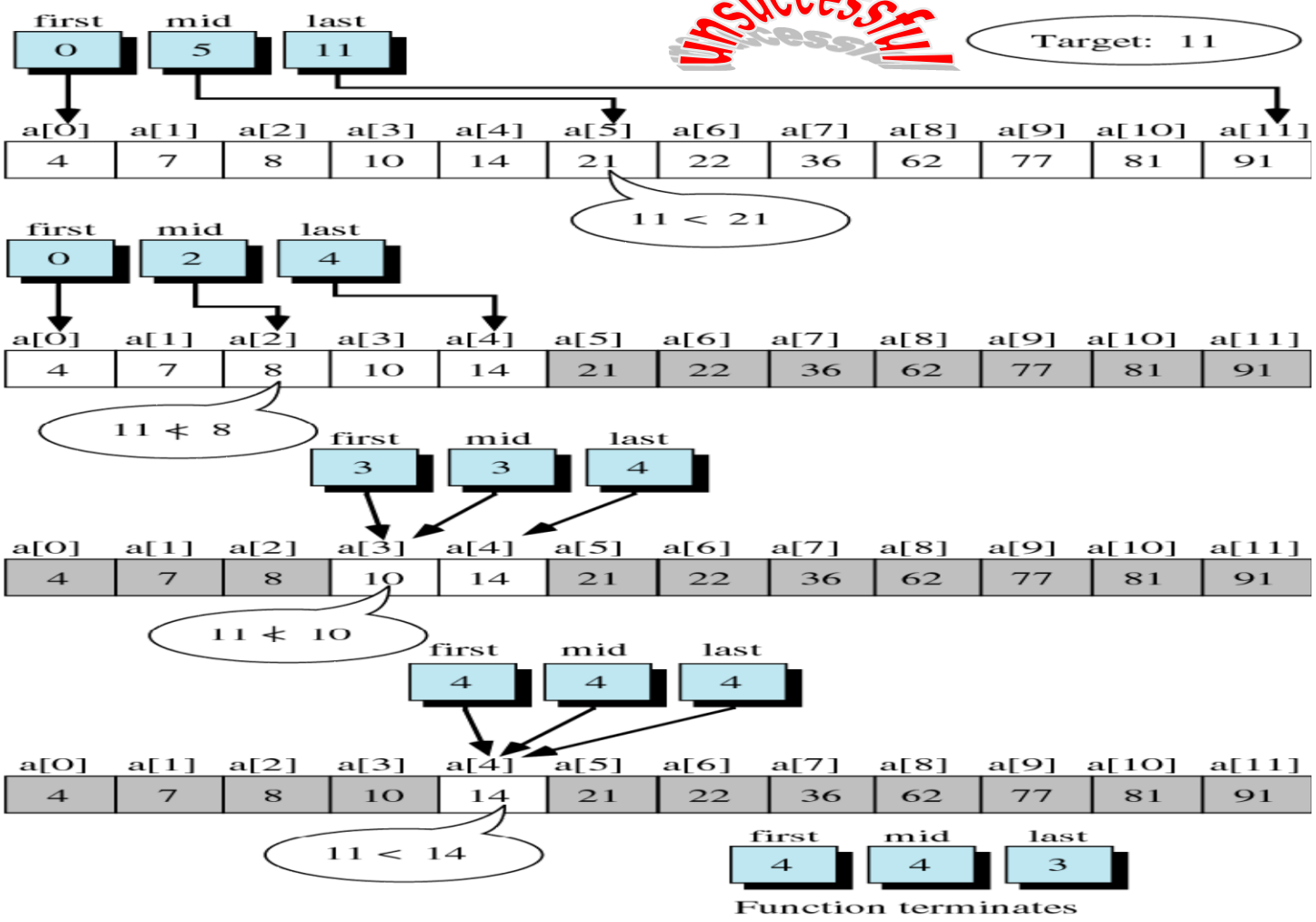
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
1	2	3	5	7	10	14	17

- Binary search skips over parts of the array if the search value cannot possibly be there.

Binary Search



Binary Search



Binary Search

- Algorithm:

```
Set first and last boundary of array to be searched
Repeat the following:
   Find middle element between first and last
   boundaries;
   if (middle element contains the search value)
          return middle_element position;
   else if (first >= last )
          return -1;
   else if (value < the value of middle_element)
           set last to middle_element position - 1;
   else
          set first to middle_element position + 1;
```

Binary Search: Iterative

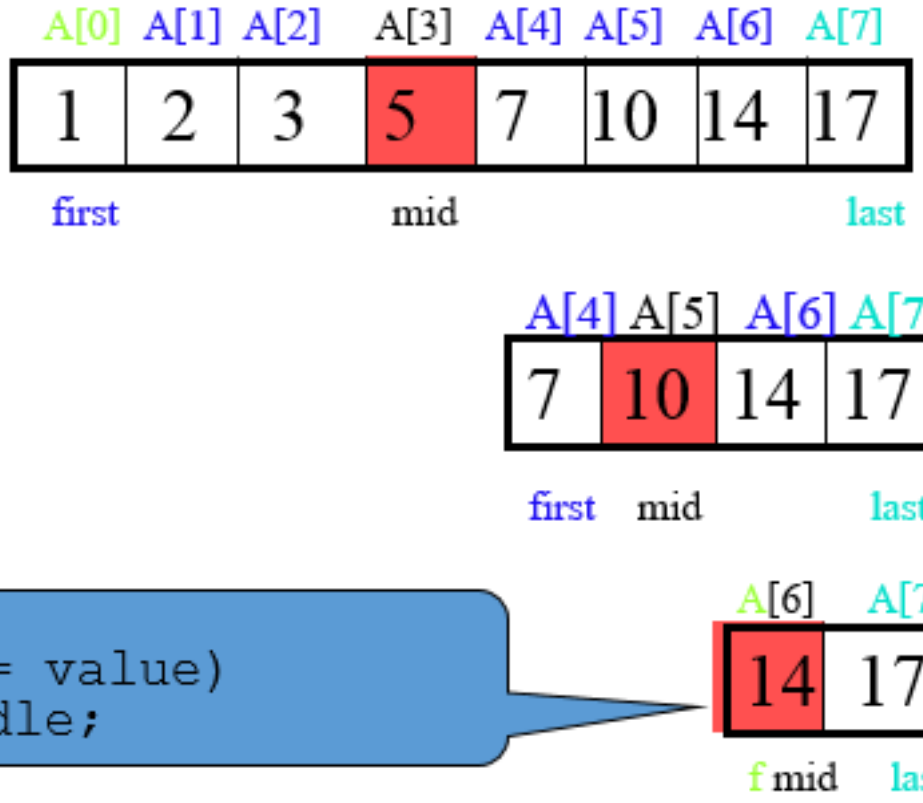
```
BinarySearch(A, n, x)
{
low ← 0
high ← n − 1
while(low ≤ high)
{
    mid ← (low + high)/2
    if x == a[mid]
        return mid
    elseif x < A[mid]
        high ← mid − 1
    else
        low ← mid − 1
 }
   return − 1
}
```

Binary Search: Recursive

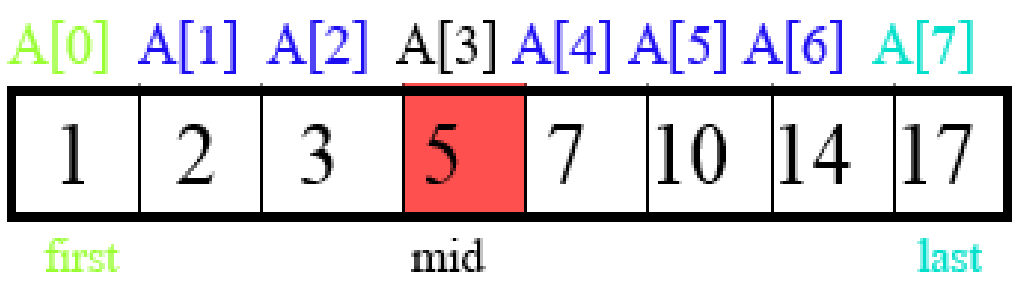
```
BinarySearch(A, n, x)
{
if (low > high)
   return  − 1
  mid ← (low + high)/2
     if x == a[mid]
          return mid
     elseif x < A[mid]
          return BinarySearch(A, low, mid − 1, x)
     else
          return BinarySearch(A, mid + 1, high, x)
  }
```

Example: binary search

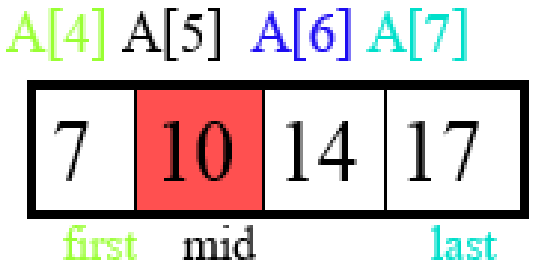
- 14?



Example: binary search



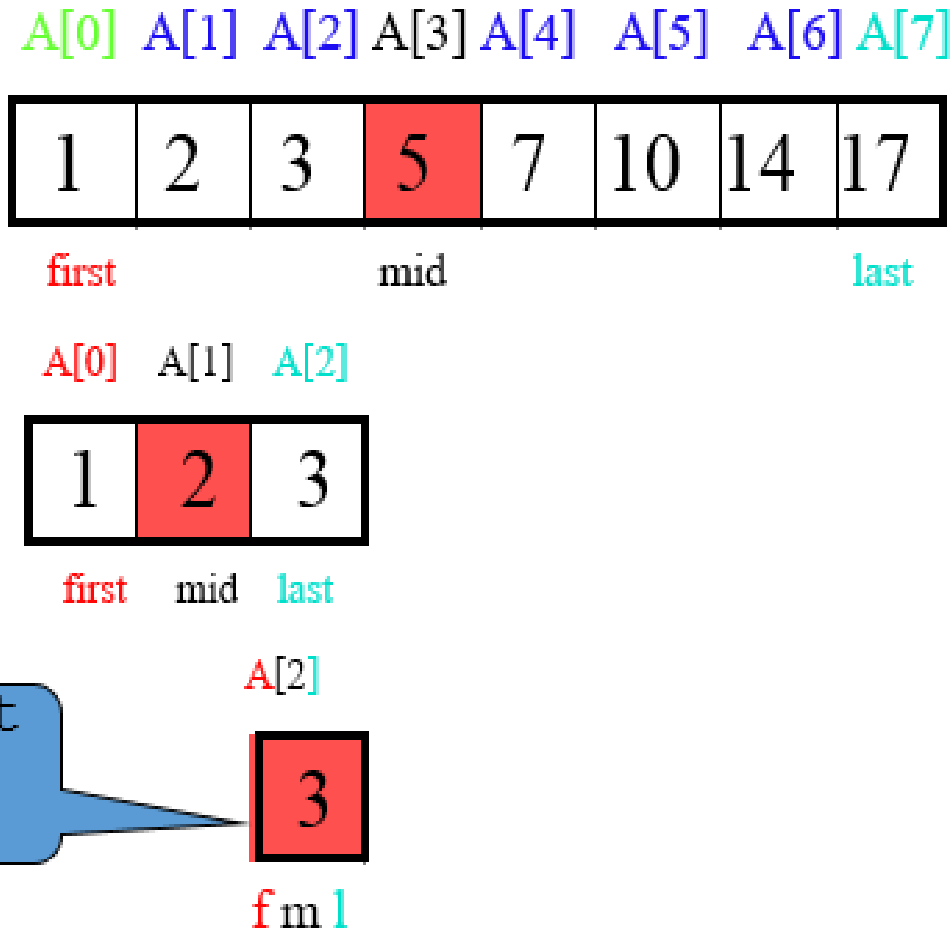
- 8 ?




```
In this case, (first
  == last)
   return -1;
```

Example: binary search unsuccessful

- 4?




```
In this case, (first
   == last)
    return -1;
```

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?

$$\lceil \log_2 n \rceil$$

- That is, we can't cut the search region in half more than that many times.
- So, the time complexity of binary search is $\mathbf{O}(\log_2 n)$ which is exactly the same as $\mathbf{O}(\log n)$.

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 our next lecture.