CSCI 132:
Basic Data Structures and Algorithms

Sorting (Quick Sort)

Reese Pearsall
Spring 2023

Announcements

Lab 12 due **tomorrow** @11:59 PM

Program 5 due Sunday May 7^{th}

Lab 13 posted

Searching

We store values in data structures, but we also need to retrieve/search for values!

Today, we will discuss techniques for how to search for a value in a data structure

(We will be using arrays, but these techniques could also be used on Linked Lists, queues, stacks, etc)

Searching

Option 1: Linear Search

Check every spot until one by one until we find what we are looking for

```
public int linear_search(int[] array, int s) {
     for(int i = 0; i < array.length; i++) {
          if(array[i] == s) {
               return i;
          }
     }
     return -1;
}
```

Searching

Option 1: Linear Search

Check every spot until one by one until we find what we are looking for

Not efficient for large data structures. **O(n)** running time

```
public int linear_search(int[] array, int s) {
     for(int i = 0; i < array.length; i++) {
          if(array[i] == s) {
               return i;
          }
     }
     return -1;
}
```

Searching

Option 1: Linear Sear

Check every spot until until we fi

Not efficient for large d

```
public i
      for(in
      }
}
```

Can we do better?

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

What if our array is sorted?

Target Value: 27

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

We can leverage the fact that this array is sorted to make searching more efficient

Target Value: 27

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

1. Start at the middle of the array

Target Value: 27

0 12

1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array

→ If the target value is less than the middle, discard the “right section” of the array

Target Value: 27

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**
low												high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array

→ If the target value is less than the middle, discard the “right section” of the array

We will define two pointers, `low` and `high` that point to the possible bounds of the target value

Target Value: 27

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

low

high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array

→ If the target value is less than the middle, discard the “right section” of the array

We will define two pointers, `low` and `high` that point to the possible bounds of the target value

Target Value: 27

0												12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**
							low					high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

Because we know the array is sorted, and the target value is greater than our mid point, then we know the target value must be located somewhere to the right.

We can eliminate half of the array!!!

`Target Value: 27`

0							7					12
1	2	9	10	11	15	18	21	27	31	41	43	50
							low					high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the "left section" of the array (move the low pointer)

→ If the target value is less than the middle, discard the "right section" of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

0							7					12
1	2	9	10	11	15	18	21	27	31	41	43	50
							low					high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

0 7 12

1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

low high

1. Start at the middle of the array
2. Compare to target value:

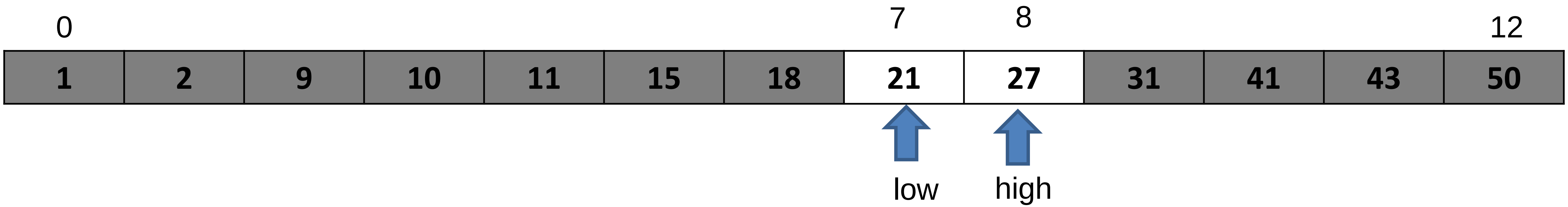
→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

1. Start at the middle of the array
2. Compare to target value:

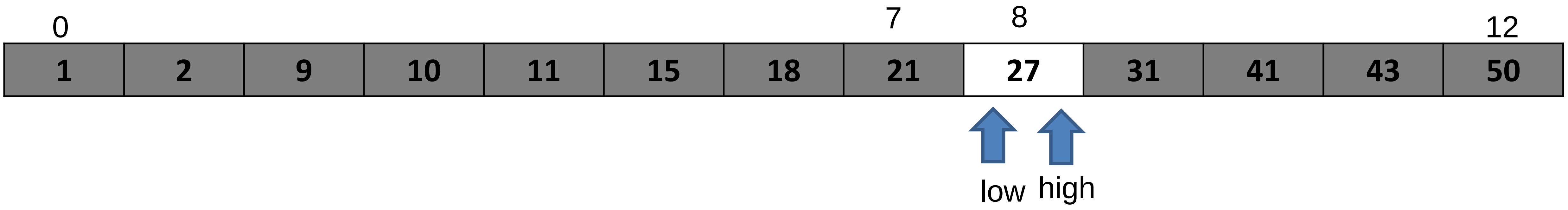
→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

0							7	8				12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

low high

1. Start at the middle of the array
2. Compare to target value:

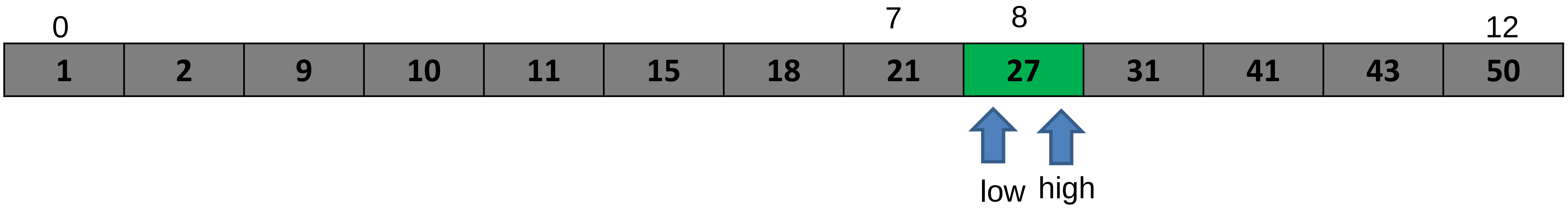
→ If the value is the target value, return

→ If the target value is greater than the middle, discard the "left section" of the array (move the low pointer)

→ If the target value is less than the middle, discard the "right section" of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

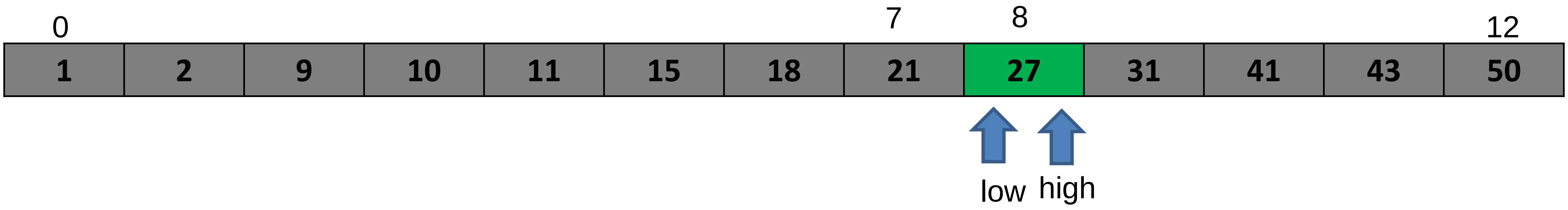
→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

Target Value: 27

0							7	8				12
1	2	9	10	11	15	18	21	27	31	41	43	50

low high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

This algorithm is known as **Binary Search**

Target Value: 27

0							7	8				12
1	2	9	10	11	15	18	21	27	31	41	43	50

low high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

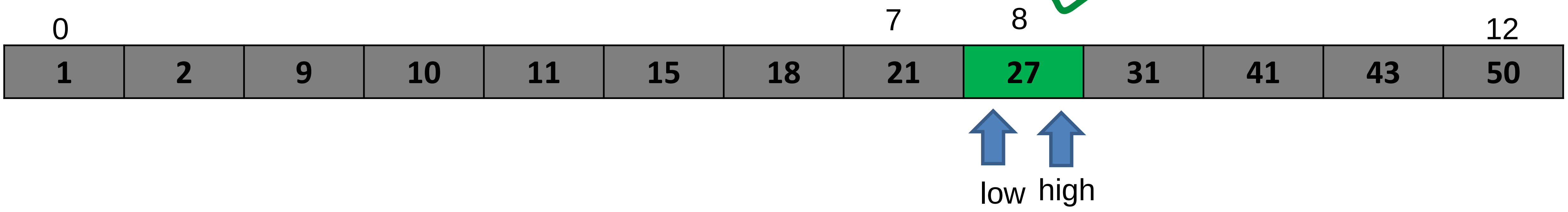
→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

How to calculate the mid point?

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

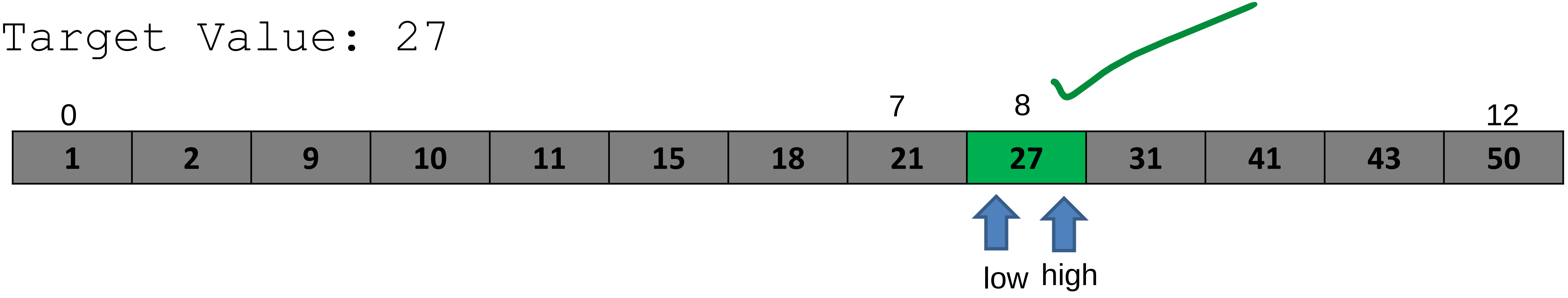
→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

How to calculate the mid point? `(low + high) / 2`

Target Value: 27



1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the “left section” of the array (move the low pointer)

→ If the target value is less than the middle, discard the “right section” of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

How do we know when to stop looping?

Target Value: 27

0							7	8				12
1	**2**	**9**	**10**	**11**	**15**	**18**	**21**	**27**	**31**	**41**	**43**	**50**

low high

1. Start at the middle of the array
2. Compare to target value:

→ If the value is the target value, return

→ If the target value is greater than the middle, discard the "left section" of the array (move the low pointer)

→ If the target value is less than the middle, discard the "right section" of the array (move the high pointer)

3. Recalculate the mid point, and repeat loop back to step 2 until target value is found

How do we know when to stop looping?

If we find the target value, or if `low` and `high` cross each other (low > high)

Target Value: 27

0 7 8 12

1		43	50

LET'S CODE THIS

1. St
2. C

→ If

→ If

sectio

→ If

sectio

3. Re

target value is found

How do we know when to stop looping?

If we find the target value, or if `low` and `high` cross each other (low > high)

```java
private static int binary_search(int[] array, int n) {
        int low = 0;
        int high = array.length - 1;
        while(low <= high) {
                int mid = (low + high) / 2;
                if(n == array[mid]) {
                        return mid;
                }
                else if(n > array[mid]) {
                        low = mid + 1;
                }
                else {
                        high = mid - 1;
                }
        }
        return -1;
}
```

```java
private static int binary_search(int[] array, int n) {
	int low = 0;
	int high = array.length - 1;
	while(low <= high) {
		int mid = (low + high) / 2;
		if(n == array[mid]) {
			return mid;
		}
		else if(n > array[mid]) {
			low = mid + 1;
		}
		else {
			high = mid - 1;
		}
	}
	return -1;
}
```

Running time?

```java
private static int binary_search(int[] array, int n) {
        int low = 0;
        int high = array.length - 1;
        while(low <= high) {
                int mid = (low + high) / 2;
                if(n == array[mid]) {
                        return mid;
                }
                else if(n > array[mid]) {
                        low = mid + 1;
                }
                else {
                        high = mid - 1;
                }
        }
        return -1;
}
```

Running time?

Each time we loop, we eliminate **half** the array

Running time?

Initial length of array $= n$

Iteration 1 - Length of array $= n/2$

Running time?

Initial length of array $= n$

Iteration 1 - Length of array $= n/2$

Iteration 2 - Length of array $= (n/2)/2 = n/2^2$

Running time?

Initial length of array $= n$

Iteration 1 - Length of array $= n/2$

Iteration 2 - Length of array $= (n/2)/2 = n/2^2$

Iteration k - Length of array $= n/2^k$

Running time?

Initial length of array $= n$

Iteration 1 - Length of array $= n/2$

Iteration 2 - Length of array $= (n/2)/2 = n/2^2$

Iteration k - Length of array $= n/2^k$

After k iterations, eventually our array has been reduced to one element

Length of array $= n/2^k = 1$

$$n = 2^k$$

“Two to what power makes n??”

Running time?

After k iterations, eventually our array has been reduced to one element

$$\text{Length of array} = n/2^k = 1$$

$$n = 2^k$$

"Two to what power makes n??"

$$log_2(n) = log_2(2^k)$$

Running time?

After k iterations, eventually our array has been reduced to one element

$$\text{Length of array} = n/2^k = 1$$

$$n = 2^k$$

"Two to what power makes n??"

$$log_2(n) = log_2(2^k)$$

$$log_2(n) = k * log_2 2$$

Running time?

After k iterations, eventually our array has been reduced to one element

$$\text{Length of array} = n/2^k = 1$$

$$n = 2^k$$

"Two to what power makes n??"

$$log_2(n) = log_2(2^k)$$

$$log_2(n) = k * \cancel{log_2 2}$$

$$log_2(n) = k$$

After K iterations, we will have done log(n) divisions

```java
private static int binary_search(int[] array, int n) {
        int low = 0;
        int high = array.length - 1;
        while(low <= high) {
                int mid = (low + high) / 2;
                if(n == array[mid]) {
                        return mid;
                }
                else if(n > array[mid]) {
                        low = mid + 1;
                }
                else {
                        high = mid - 1;
                }
        }
        return -1;
}
```

Running time?

Generally speaking, whenever we eliminate half of the problem each iteration, that will give us **O(logn)** running time

```
private static int binary_search(int[] array, int n) {
        int low = 0; O(1)
        int high = array.length - 1; O(1)
        while(low <= high) { O(log n)
                int mid = (low + high) / 2;  O(1)
                if(n == array[mid]) { O(1)
                        return mid; O(1)
                }
                else if(n > array[mid]) { O(1)
                        low = mid + 1;  O(1)
                }
                else {
                        high = mid - 1; O(1)
                }
        }
        return -1; O(1)
}
```

Running time?

Generally speaking, whenever we eliminate half of the problem each iteration, that will give us **O(logn)** running time

```
private static int binary_search(int[] array, int n) {
        int low = 0; O(1)
        int high = array.length - 1; O(1)
        while(low <= high) { O(log n)
                int mid = (low + high) / 2;  O(1)
                if(n == array[mid]) { O(1)
                        return mid; O(1)
                }
                else if(n > array[mid]) { O(1)
                        low = mid + 1;  O(1)
                }
                else {
                        high = mid - 1; O(1)
                }
        }
        return -1; O(1)
}
```

Running time? O(log n)

```java
private static int binary_search(int[] array, int n) {
	int low = 0;
	int high = array.length - 1;
	while(low <= high) {
		int mid = (low + high) / 2;

		int result = x.compareTo(array[mid])

		if(result = 0) {
			return mid;
		}
		else if(result > 0){
			low = mid + 1;
		}
		else {
			high = mid - 1;
		}
	}
	return -1;
}
```

We can do binary search on an array of Strings using the `compareTo()` method

```java
private static int binary_search(????????????) {

	while(low <= high) {
		int mid = (low + high) / 2;
		if(n == array[mid]) {
			return mid;
		}
		else if(n > array[mid]) {
			binary_search(?????????);
		}
		else {
			binary_search(?????????);
		}
	}
	return -1;
}
```

Binary Search can also be implemented using recursion (Program 5)