



# BINARY SEARCH

CS A250 – C++ Programming II

# SEARCH ALGORITHMS

- Search algorithms are very common
  - They search a list
  - Look at each item in the list and compare to the search item
- We will consider two ways to search:
  - **Linear**
    - Also called **sequential** search
    - Can be done in an **ordered/unordered** list
  - **Binary**
    - Can be done *only* in an *ordered* list

# LINEAR SEARCH

- **Linear Search** searches through the elements of an arbitrary sequence (in this case, a **vector v**) until
  - The match is found **OR**
  - It reaches the end of the sequence

```
int count = 0;
bool found = false;
int size = static_cast<int>(v.size());
while (!found && count < size)
{
    if (v[count] == value)
        found = true;
    else
        ++count;
}
```

**Assumption:** All elements in the list are *unique*.

# HOW MANY COMPARISONS?

- You always look at the worst case:
  - Assuming your list has 20 elements
    - You compare each element at most once with the given element
    - If the element you are searching for happens to be the last element in the list:
      - You have made **20 comparisons**
      - **Generalize:** 20 is denoted by  $n$  (a list of  $n$  elements)
- Therefore, the worst case in a **sequential search** is  **$n$  comparisons**.

# BINARY SEARCH

## ○ Binary search

- Is faster than **linear search**
  - **BUT** assumes **array is sorted**
- Breaks the list in **half**
  - Determines if item in 1<sup>st</sup> or 2<sup>nd</sup> half
  - Then searches again just that half
    - Can be done **recursively**.

# EXECUTION OF BINARY SEARCH

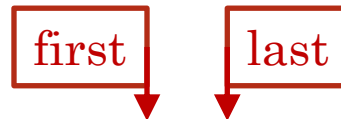
- A **sorted** array of 10 elements
  - Search for **63**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements

- Search for **63**



Since indices are type **int**  
result will be truncated

Find the **middle** →  $(0 + 9) / 2 = 4$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90



# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Find the **middle**  $\rightarrow (0 + 9) / 2 = 4$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
**mid**

Is **[4]** equal to **63**?      No  
Is **63** > or < than **[4]**?      >  
Check between [5] and [9]

1<sup>st</sup> comparison



# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Find the **middle**  $\rightarrow (5 + 9) / 2 = 7$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
**mid**

# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Find the **middle**  $\rightarrow (5 + 9) / 2 = 7$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
**mid**


Is [7] equal to 63?	No	2 <sup>nd</sup> comparison
Is 63 > or < than [7]?	<	
Check between [5] and [6]		

# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Find the **middle**  $\rightarrow (5 + 6) / 2 = 5$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90



# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Find the **middle**  $\rightarrow (5 + 6) / 2 = 5$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
**mid**

Is **[5]** equal to **63**?

Yes

Stopping case  $\rightarrow$

**63** found

Location = **mid**


3<sup>rd</sup> comparison

# EXECUTION OF BINARY SEARCH (CONT.)

- A **sorted** array of 10 elements
  - Search for **63**

Number of **comparisons** → **3**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90



A blue box labeled 'mid' with an upward-pointing arrow indicating the current middle element of the search range, which is at index 5 (value 63).

## ANOTHER EXAMPLE

- A **sorted** array of 10 elements
  - Search for **38**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (0 + 9) / 2 = 4$

first

last

Since indices are type **int**  
result will be truncated

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

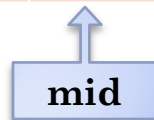
mid

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (0 + 9) / 2 = 4$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90



Is [4] equal to 38?	No
Is 38 > or < than [4]?	<
Check between [0] and [3]	

1<sup>st</sup> comparison



## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (0 + 3) / 2 = 1$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
mid

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (0 + 9) / 2 = 4$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
mid

Is [4] equal to 38?	No	2 <sup>nd</sup> comparison
Is 38 > or < than [4]?	>	
Check between [5] and [6]		

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (2 + 3) / 2 = 2$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

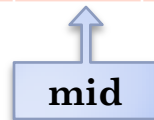
↑  
mid

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (2 + 3) / 2 = 2$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90



Is [2] equal to 38? No  
Is 38 > or < than [2]? >  
Check between [3] and [3]

3<sup>rd</sup> comparison

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (2 + 3) / 2 = 2$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
mid

Is [2] equal to 38? No  
Is 38 > or < than [2]? >  
Check between [3] and [3]

You need to check  
until **first** > **last**.

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (3 + 3) / 2 = 3$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
**mid**

Is <b>[3]</b> equal to <b>38</b> ?	No	4 <sup>th</sup> comparison
Is <b>38</b> > or < than <b>[3]</b> ?	<	
Check between [3] and [2]		

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Find the **middle**  $\rightarrow (3 + 3) / 2 = 3$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

↑  
mid

Is [3] equal to 38? No  
Is 38 > or < than [3]? <  
Check between [3] and [2]

first > last.

## ANOTHER EXAMPLE (CONT.)

- A **sorted** array of 10 elements
  - Search for 38

Number of **comparisons** → 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90

Key 38 not found.



# IMPLEMENTATION

- **Binary search** can be implemented
  - **Iteratively**
    - Using a **while** loop
  - **Recursively**
    - **IF/ELSE** statement and **call to itself**
  - **Stopping case** for both implementations:
    - **if (first > last)** → no elements between them, so **key** cannot be there!
    - **if (key == a[mid])** → found!

# EFFICIENCY OF BINARY SEARCH

- **Binary search** is *very efficient*
  - Extremely *fast*, compared to **sequential search**
- Half of array eliminated at start!
  - Then a quarter, then 1/8, etc.
  - Essentially eliminate half with each call
- Example: Array of 100 elements
  - In this case, a binary search never needs more than 7 compares!

# RECURSIVE SOLUTIONS

- If done **recursively**, then the **binary search** algorithm actually solves "more general" problem
  - Original goal:
    - Design function to search an entire list
  - **BUT** you can search any *interval of the list*
    - By specifying bounds *first* and *last*
    - Very common when designing recursive functions



BINARY SEARCH (END)

28