

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
 - o 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

```
Assumption: All elements
int count = 0;
                                          in the list are unique.
bool found = false;
int size = static cast<int>(v.size());
while (!found && count < size)
    if (v[count] == value)
        found = true;
    else
        ++count;
```

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

o Binary search

- Is faster than linear search
 - BUT assumes array is sorted
- Breaks the list in half
 - Determines if item in 1st or 2nd 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

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

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

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

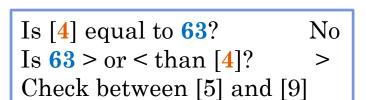
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
15	20	35	41	57	63	75	80	85	90
				42					



- 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
				Î					



mid

- 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



- 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
							4		



- 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



- 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
					1				



Is [5] equal to 63? Yes
Stopping case → 63 found
Location = mid

- 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



IMPLEMENTATION

- o 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!
 - \circ if (key == a[mid]) \rightarrow found!

EFFICIENCY OF BINARY SEARCH

- o 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

- o 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)