DATA STRUCTURES
&
ALGORITHMS

By: Manoja Weerasekara
NSBM-Colombo

Lecture 02

Searching Algorithms

Searching

- Finding elements in large amounts of data
  - Determine whether array contains value matching *key value*

- Linear search

- Binary search

Searching an Array

- **Linear search**

    small arrays

    unsorted arrays

- **Binary search**

    large arrays

    sorted arrays

Linear Search Algorithm

- *Brute force* algorithm, that starts searching the key from one end of the data set and proceed searching to the other end of the data set – *Exhaustive Search*.

    - Start at first element of the array.
    - Compare the value with the key
    - Continue with next element until a match is found or reach end of the array.

➢Note: On the average we have to compare the search key with half the elements in the array.

Linear Search - Example

* Array `numlist` contains:

17	23	5	11	2	29	3

* Searching for the the value `11`:

linear search examines `17, 23, 5,` and `11`

* Searching for the the value `7`:

linear search examines `17, 23, 5, 11, 2, 29,` and `3`

Linear Search

- Algorithm:

```
set found to false; set position to –1; set index to 0

while index < number of elements. and found is false

    if list[index] is equal to search value
            found = true
            position = index
     end if
            add 1 to index
end while

return position
```

A Linear Search Function

```
int searchList(int list[], int numElems, int value)
{
   int index = 0;      // Used as a subscript to search array
   int position = -1;  // To record position of search value
   boolean found = false; // Flag to indicate if value was
                                                      found

   while (index < numElems && !found)
   {
      if (list[index] == value) // If the value is found
      {
         found = true; // Set the flag
         position = index; // Record the value's subscript
      }
      index++; // Go to the next element
   }
return position; // Return the position, or -1
}
```

Linear Search

* **Benefits:**
  * Easy to understand and implement
  * No assumption on data - can be in any order
* **Disadvantages:**
  * Inefficient (slow) → O(n)
  * Best case scenario?
  * Average complexity?
  * Worst case scenario?

Binary Search Algorithm

Need the dataset to be sorted (say ascending order)

At each step check the middle item

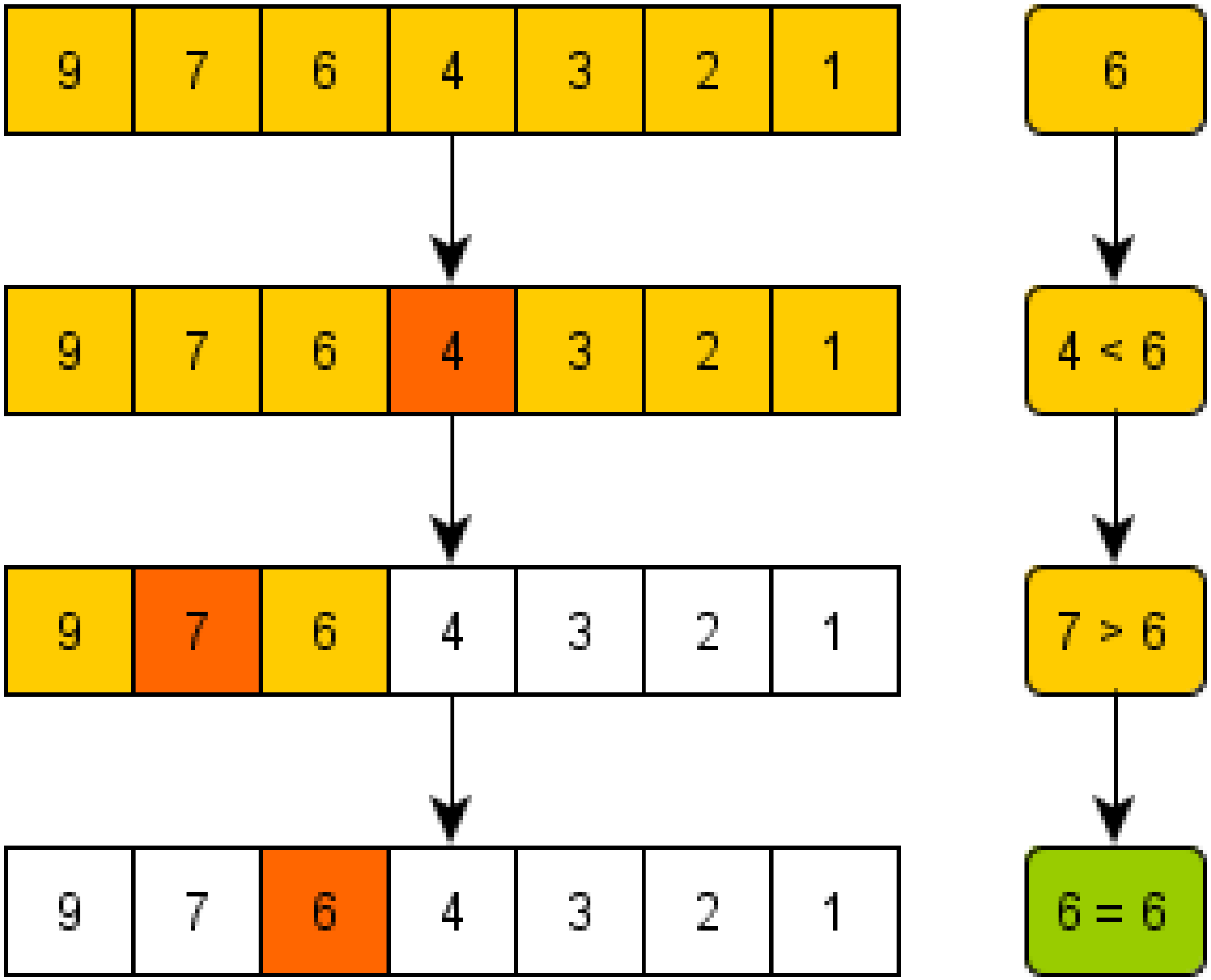
If key == middle item → item found

If key < middle item → binary search the first half

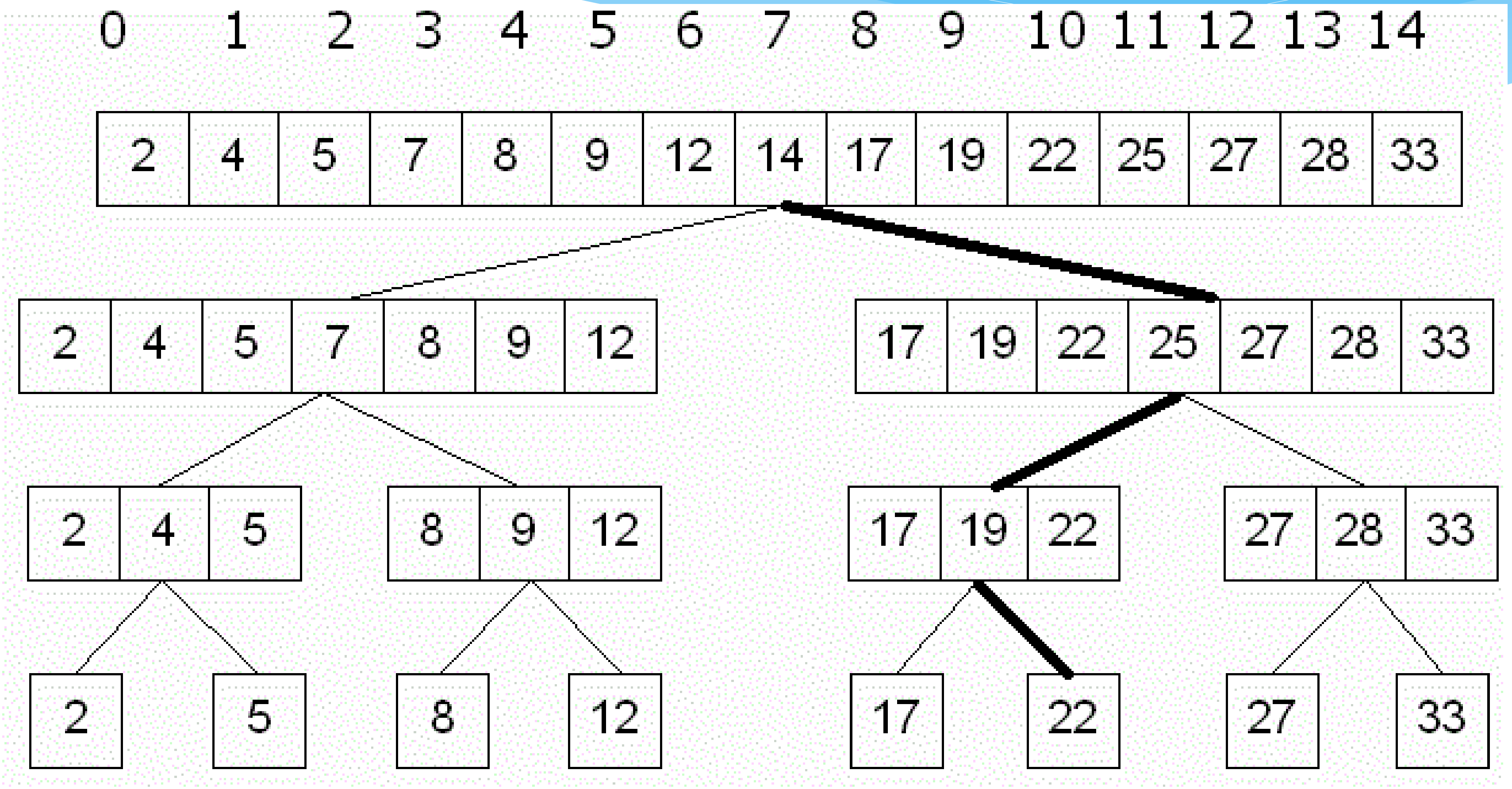
If key > middle item → binary search the second half

Search ends either finding the item or after checking an array of size 1.

Binary Search Example



Binary Search Tree



Efficiency

- Searching and array of 1024 elements will take at most 10 passes to find a match or determine that the element does not exist in the array.

512, 256, 128, 64, 32, 16, 8, 4, 2, 1

- An array of one billion elements takes a maximum of 30 comparisons.

- The bigger the array the better a binary search is as compared to a linear search

- O(log n)

Binary Search

Set first index to 0. Set last index to the last subscript in the array.
Set found to false. Set position to -1.

While found is not true and first is less than or equal to last
    Set middle to the subscript half-way between array[first] and array[last].

    If array[middle] equals the desired value
        Set found to true.
        Set position to middle.

        Else If array[middle] is greater than the desired value
            Set last to middle - 1.
        Else
            Set first to middle + 1.
    End If.

End While.
Return position.

A Binary Search Function

```
int binarySearch(int array[], int size, int value)
{
   int first = 0,                  // First array element
       last = size - 1,            // Last array element
       middle,                     // Mid point of search
       position = -1;              // Position of search value
   boolean found = false;             // Flag

   while (!found && first <= last)
   {
      middle = (first + last) / 2;      // Calculate mid point
      if (array[middle] == value)       // If value is found at mid
      {
         found = true;
         position = middle;
      }
      else if (array[middle] > value)   // If value is in lower half
         last = middle - 1;
      else
         first = middle + 1;            // If value is in upper half
   }
   return position;
}
```

Carryout a Desk-Check for the above algorithm for the following data set:

data[] = { 12, 16, 23, 28, 34, 42, 47, 55, 64, 65, 66, 72 }

key = 54

Var	1st call	2nd call	3rd call	4th call
key				
data				
size				
mid				
data[mid]				
return				

Answer:

Var	**1st call**	**2nd call**	**3rd call**	**4th call**
key	54	54	54	54
data	&data[0]	&data[7]	&data[7]	&data[7]
size	12	5	2	1
mid	6	2	1	0
data[mid]	47	65	64	55
return	?	?	?	-1

Thank You