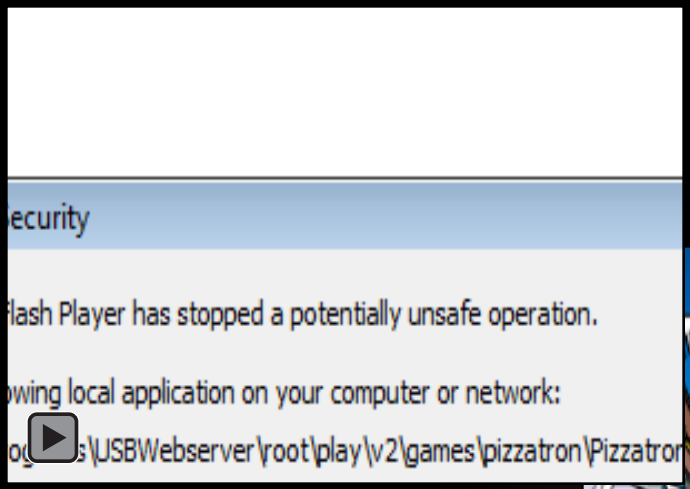
# Search Algorithms

Sequential Search (Linear Search)

Binary Search

# SEQUENTIAL SEARCH

- A sequential search of a list/array begins at the beginning of the list/array and continues until the item is found or the entire list/array has been searched




# SEQUENTIAL SEARCH FUNCTION

```
boolean function  LinSearch(int[] x, int item){
        int n=x.length;
        for(int i=0;i<n;i++)
        {
                if(x[i]==item) return true;
        }
        return false;
}
```

## Linear Search Tradeoffs

- Benefits
  - Easy algorithm to understand
  - Array can be in any order
- Disadvantage
  - Inefficient (slow): for array of N elements, examines N/2 elements on average for value in array, N elements for value not in array

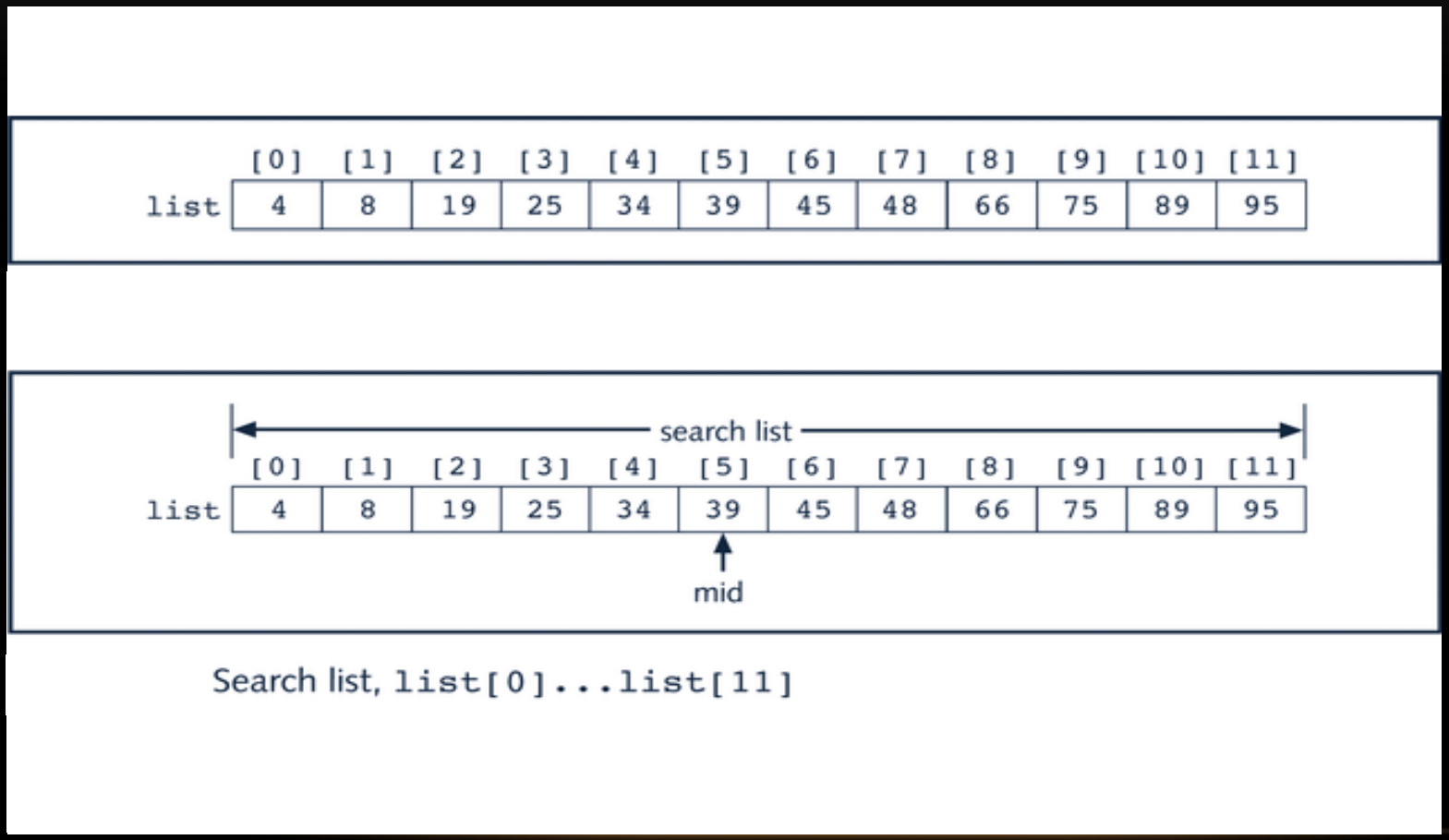
# BINARY SEARCH

- A binary search looks for an item in a list using a divide-and-conquer strategy

# BINARY SEARCH

- Binary search algorithm assumes that the items in the array being searched are sorted
- The algorithm begins at the middle of the array in a binary search
- If the item for which we are searching is less than the item in the middle, we know that the item won't be in the second half of the array
- Once again we examine the "middle" element
- The process continues with each comparison cutting in half the portion of the array where the item might be

# BINARY SEARCH




Search list, `list[0]...list[11]`

# BINARY SEARCH: MIDDLE ELEMENT

$$mid = Floor\left(\frac{left + right}{2}\right)$$

# BINARY SEARCH: EXAMPLE


Values of `first`, `last`, and `middle` and the Number of Comparisons for Search Item 89

| Iteration | first | last | mid | list[mid] |
|---|---|---|---|---|
| 1 | 0 | 11 | 5 | 39 |
| 2 | 6 | 11 | 8 | 66 |
| 3 | 9 | 11 | 10 | 89 |

# BINARY SEARCH

| | |
|---|---|
| [0] | ant |
| [1] | cat |
| [2] | chicken |
| [3] | cow |
| [4] | deer |
| [5] | dog |
| [6] | fish |
| [7] | goat |
| [8] | horse |
| [9] | camel |
| [10] | snake |

**Searching for cat**

| | | | |
|---|---|---|---|
| BinarySearch(0, 10) | middle: 5 | cat < dog | |
| BinarySearch(0, 4) | middle: 2 | cat < chicken | |
| BinarySearch(0, 1) | middle: 0 | cat > ant | |
| BinarySearch(1, 1) | middle: 1 | cat = cat | **Return: true** |

**Searching for zebra**

| | | | |
|---|---|---|---|
| BinarySearch(0, 10) | middle: 5 | zebra > dog | |
| BinarySearch(6, 10) | middle: 8 | zebra > horse | |
| BinarySearch(9, 10) | middle: 9 | zebra > camel | |
| BinarySearch(10, 10) | middle: 10 | zebra > snake | |
| BinarySearch(11, 10) | | last > first | **Return: false** |

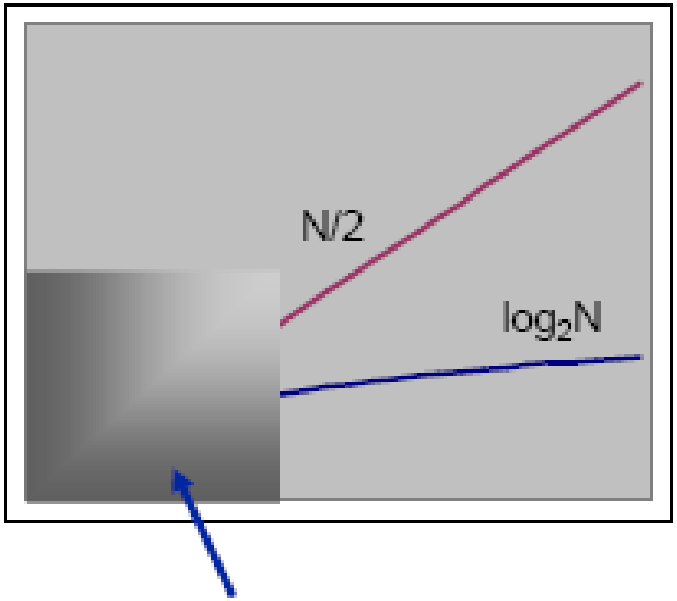
**Searching for fish**

| | | | |
|---|---|---|---|
| BinarySearch(0, 10) | middle: 5 | fish > dog | |
| BinarySearch(6, 10) | middle: 8 | fish < horse | |
| BinarySearch(6, 7) | middle: 6 | fish = fish | **Return: true** |

## Binary Search Tradeoffs

- Benefit
  - Much more efficient than linear search

  (For array of $N$ elements, performs at most $log_2 N$ comparisons)
- Disadvantage
  - Requires that array elements be sorted



# EXERCISE-ASSIGNMENT

- Design and code a program to illustrate the process of binary search by reading in the file “ispell.words” and then searching for a word (provided by the user) in the file.
- Make sure your program indicates whether the item to be found is in the array to be searched and if so where it was found and how many iterations it took.
- Post your source and program on your Unit 3 Web Page