Search Algorithms

Sequential Search (Linear Search)
Binary Search

Sequential Search
O (n)

- 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

```
bool LinSearch(double x[ ], int n, double item){

        for(int i=0;i<n;i++){
                if(x[i]==item) return true;
                else return false;
        }
        return false;
  }
```

Search Algorithms

Suppose that there are n elements in the array. The following expression gives the average number of comparisons:

$$\frac{1+2+\ldots+n}{n}$$

It is known that

$$1+2+\ldots+n=\frac{n(n+1)}{2}$$

Therefore, the following expression gives the average number of comparisons made by the sequential search in the successful case:

$$\frac{1+2+\ldots+n}{n}=\frac{1}{n}\frac{n(n+1)}{2}=\frac{n+1}{2}$$

Search Algorithms

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

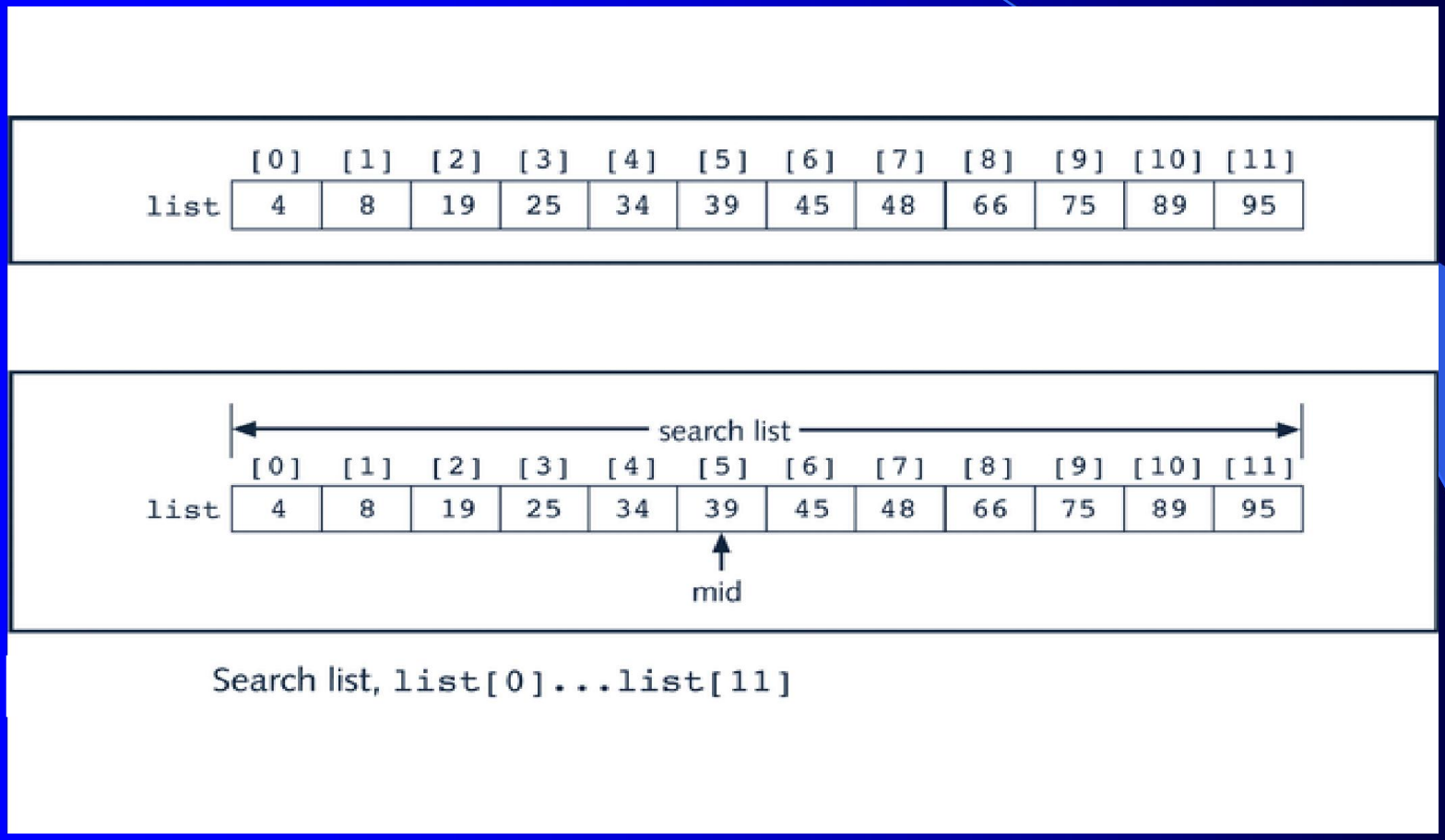
Binary Search
O(log2 n)

- 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]`

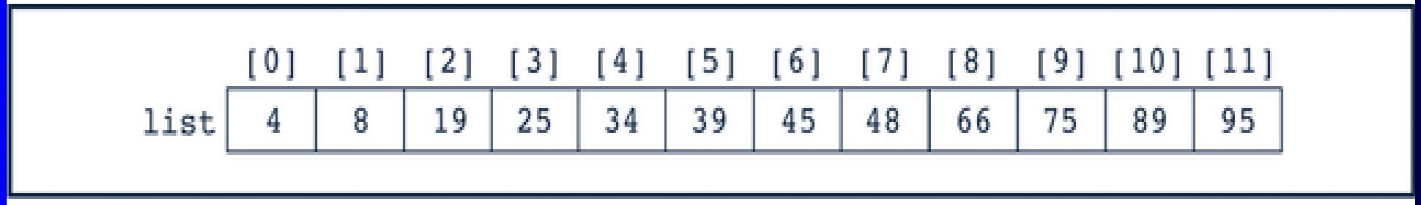
Binary Search: middle element

$$\texttt{mid} = \frac{\texttt{left} + \texttt{right}}{2}$$

Binary Search

```
bool BinSearch(double list[ ], int n, double item, int&index){
        int left=0;
        int right=n-1;
        int mid;
        while(left<=right){
        mid=(left+right)/2;
        if(item> list [mid]){ left=mid+1; }
        else if(item< list [mid]){right=mid-1;}
        else{
        item= list [mid];
        index=mid;
        return true; }
    }// while
   return false;
}
```

Binary Search: Example



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
list	4	8	19	25	34	39	45	48	66	75	89	95

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