Array Search Algorithms

Linear and Binary Search Implementation

Mr. Gullo

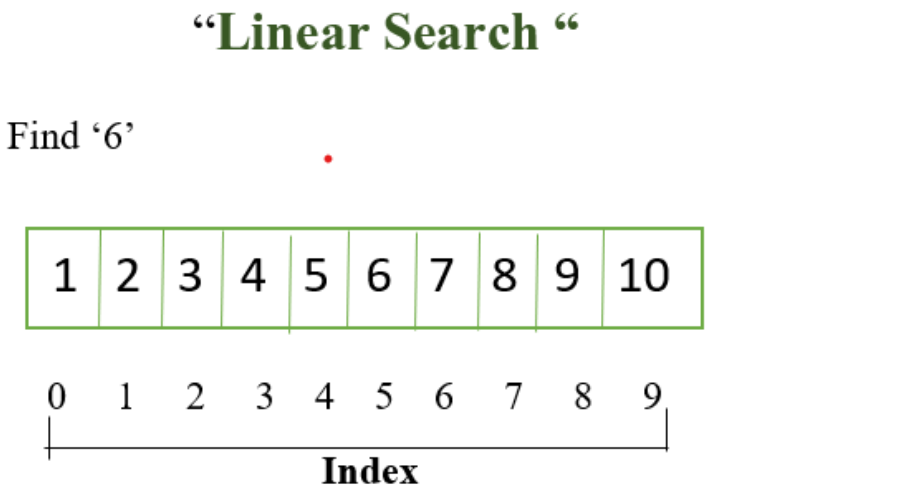
February, 2025

Learning Objectives

- Understand the fundamental differences between linear and binary search
- Analyze time complexity in simple terms
- Read and comprehend C++ implementations of search algorithms
- Identify appropriate use cases for each search method

Linear Search: The Basics

- Searches through array elements one by one
- Works on both sorted and unsorted arrays
- Time complexity: O(n)



Note : We find '6' at index '5' through linear search .

Linear Search Implementation

```
bool search_unsorted(int value, int arr[], int length)
    for(int i = 0; i < length; i++)
        if(value == arr[i]) return true;
    return false;
}
```

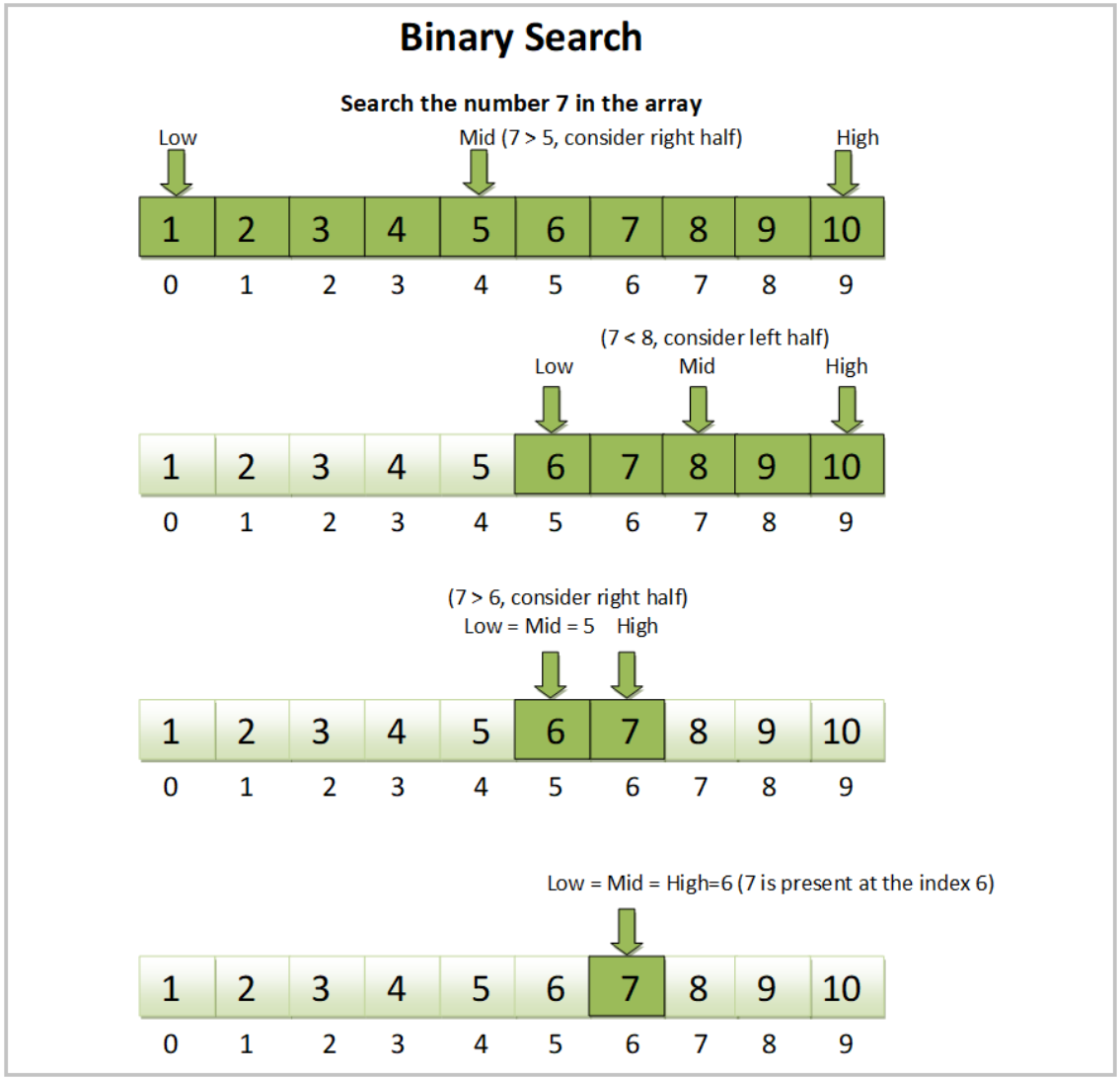
Linear Search Implementation

```
bool search_unsorted(int value, int arr[], int length)
    for(int i = 0; i < length; i++)
        if(value == arr[i]) return true;
    return false;
}
```

- Simple implementation
- Checks each element exactly once
- Returns as soon as value is found

Binary Search: The Concept

- Requires sorted array
- Divides search space in half
- Time complexity: O(log n)



Binary Search Implementation

```
bool search_sorted(int value, int arr[], int length) {
    // Initialize indices for binary search
    int firstIndex = 0;
    int lastIndex = length - 1;
    while(firstIndex <= lastIndex) {
        // TODO 1: Calculate the middle index
        int midpointIndex =   "..."
        // TODO 2: Check if value found
        if(check if current element is target)
        { "..." return true;   }
        // TODO 3: Decide which half to search
        else if(check if search right half) {
        "...-"}
        else {// update the index to search left half
        }    } return false;    }
```

Project Structure Overview

- Multi-file organization:
  - Header file (`arraySearch.h`)
  - Main program (`main.cpp`)
  - Implementation (`arraySearch.cpp`)

Project Structure Overview

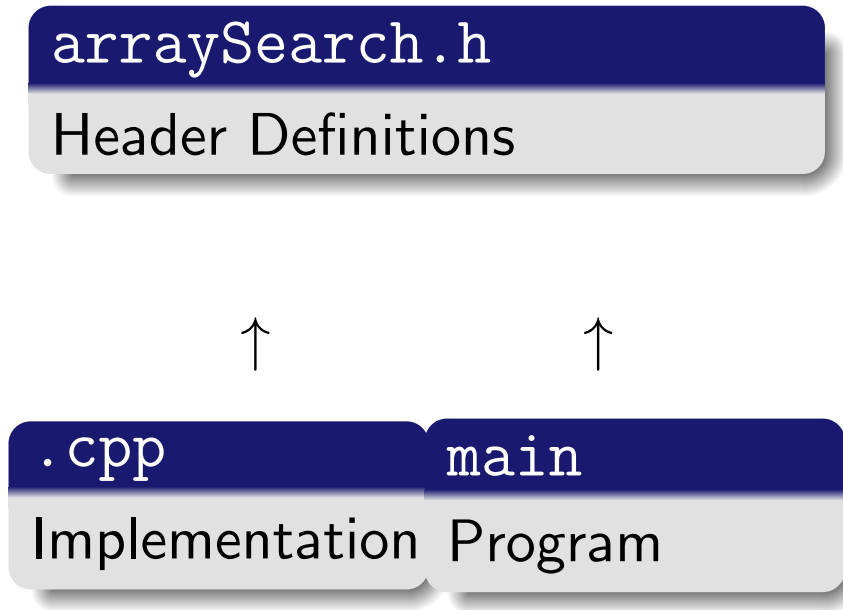
- Multi-file organization:
  - Header file (`arraySearch.h`)
  - Main program (`main.cpp`)
  - Implementation (`arraySearch.cpp`)
- Separation of debugging concerns

Project Structure Overview

- Multi-file organization:
  - Header file (`arraySearch.h`)
  - Main program (`main.cpp`)
  - Implementation (`arraySearch.cpp`)
- Separation of debugging concerns
- Improved maintainability

Project Structure Overview

- Multi-file organization:
  - Header file (`arraySearch.h`)
  - Main program (`main.cpp`)
  - Implementation (`arraySearch.cpp`)
- Separation of debugging concerns
- Improved maintainability



Header File: arraySearch.h

```
#ifndef ARRAYSEARCH_H_INCLUDED
#define ARRAYSEARCH_H_INCLUDED

bool search_unsorted(int value, int arr[], int length);
    // Returns true if value is in the array. else False.
    // Note: this should be O(n)

bool search_sorted(int value, int arr[], int length);
    // Returns true if value is in the array. eslse False.
    // Note: this should be O(log(n))

bool test_search_unsorted(); //static edge case testing
bool test_search_sorted(); //static edge case testing
#endif
```

Header File: arraySearch.h

```
#ifndef ARRAYSEARCH_H_INCLUDED
#define ARRAYSEARCH_H_INCLUDED

bool search_unsorted(int value, int arr[], int length);
    // Returns true if value is in the array. else False.
    // Note: this should be O(n)

bool search_sorted(int value, int arr[], int length);
    // Returns true if value is in the array. eslse False.
    // Note: this should be O(log(n))

bool test_search_unsorted(); //static edge case testing
bool test_search_sorted(); //static edge case testing
#endif
```

- Function declarations only
- Include guards prevent multiple inclusion
- Clear documentation of function behavior

Main Program Structure

```
int main() {
    // Create test array
    int testArray[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int length = sizeof(testArray)/sizeof(testArray[0]);

    // Test values
    int searchValues[] = {16, 91, 2, 50, -5, 100};

    // Test both search functions
    cout << "\n=== Testing Linear Search ===\n";
    // ... testing code ...

    cout << "\n=== Testing Binary Search ===\n";
    // ... testing code ...

    return 0;}
```

- Organized testing structure
- Clear separation of test data and logic

Testing Implementation

- Comprehensive test cases:
  - Elements at beginning, middle, and end
  - Values not in array
  - Edge cases (empty array, single element)
- Automated testing functions:
  - `test_search_unsorted()`
  - `test_search_sorted()`

Present Values	Missing Values	Edge Cases
Expected: Success	Expected: Failure	Expected: Validation

Comparison: When to Use Each

Linear Search

- Unsorted data
- Small arrays
- One-time searches

Comparison: When to Use Each

Linear Search

- Unsorted data
- Small arrays
- One-time searches

Binary Search

- Sorted data
- Large datasets
- Frequent searches

Summary

- Linear search is simple but slower (O(n))
- Binary search requires sorted data but is faster (O(log n))
- Choice depends on:
  - Data organization
  - Dataset size
  - Search frequency
- Testing is crucial for both implementations