Searching

Last updated 6/22/23

These slides show an example of searching an array

Array Applications

- Searching
  - Want to determine if and where something is in an array
  - Sequential Search
  - Binary Search

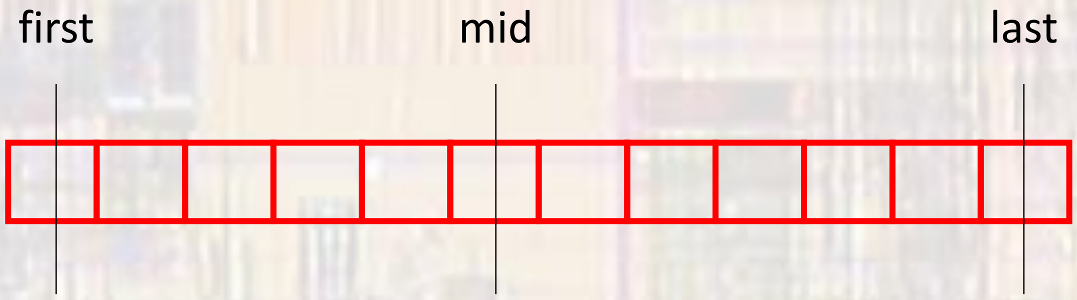
Array Applications

- Sequential Search
  - Check each array value for the item you are looking for
  - Takes a maximum of N checks
  - If N = 1M, up to 1M checks

Array Applications

- Binary Search

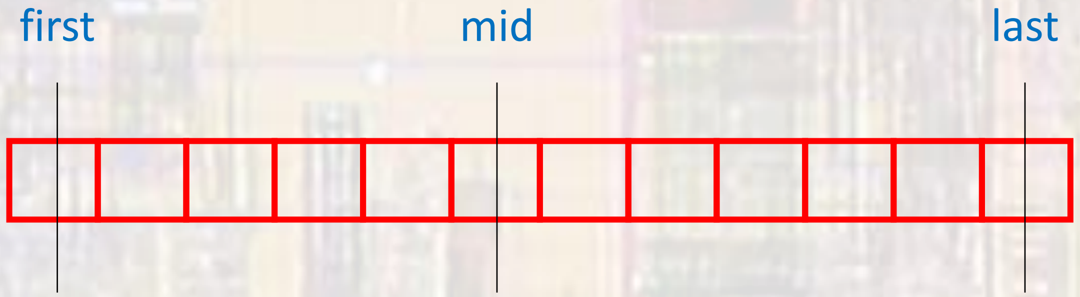
  - Requires the data to be sorted
  - Reduces the number of checks to $\log_2 N+1$
  - If N = 1M, → 21 checks



Array Applications

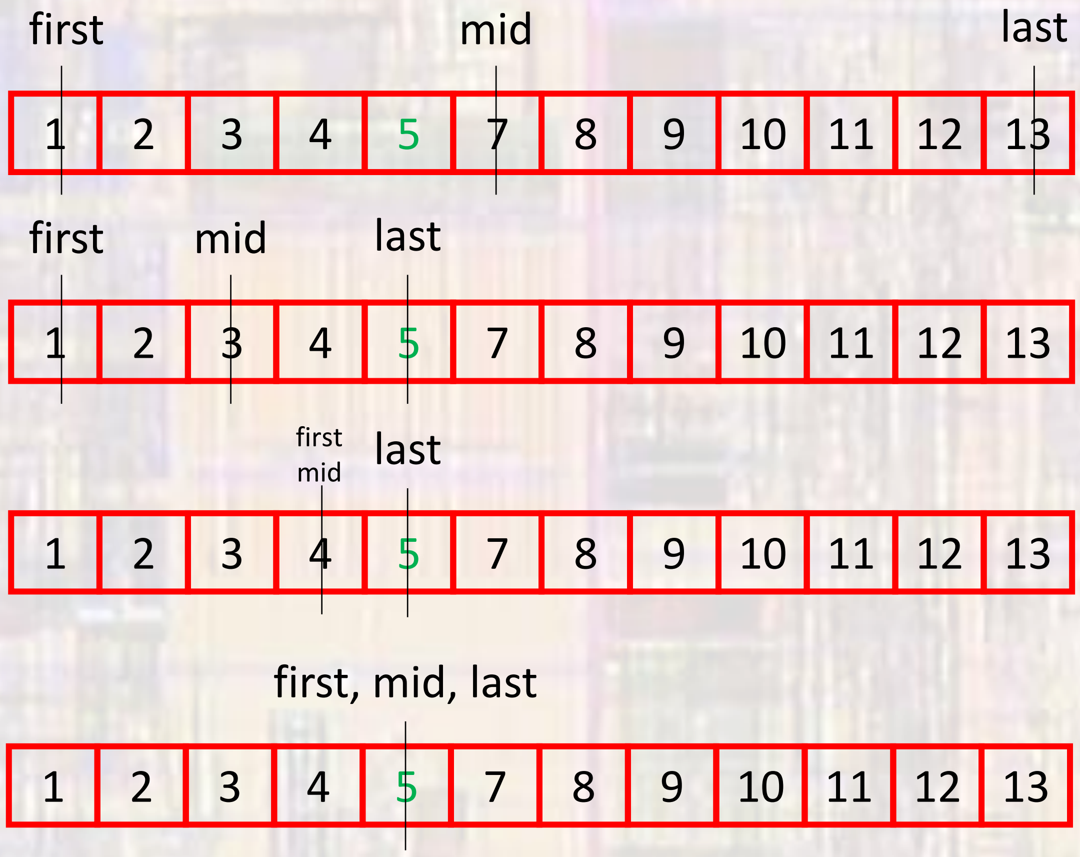
- Binary Search

  - Find the mid point between first and last(indexes)
  - Compare the target with the value at mid
  - If value is greater than mid → set first to mid + 1
  - If value is less than mid → set last to mid -1
  - If value = target → return the index
  - If first > last → value not in the list



Array Applications

- Binary Search – looking for 5



target < mid

Set last to mid-1
Reset mid

target > mid

Set first to mid+1
Reset mid

target > mid

Set first to mid+1
Reset mid

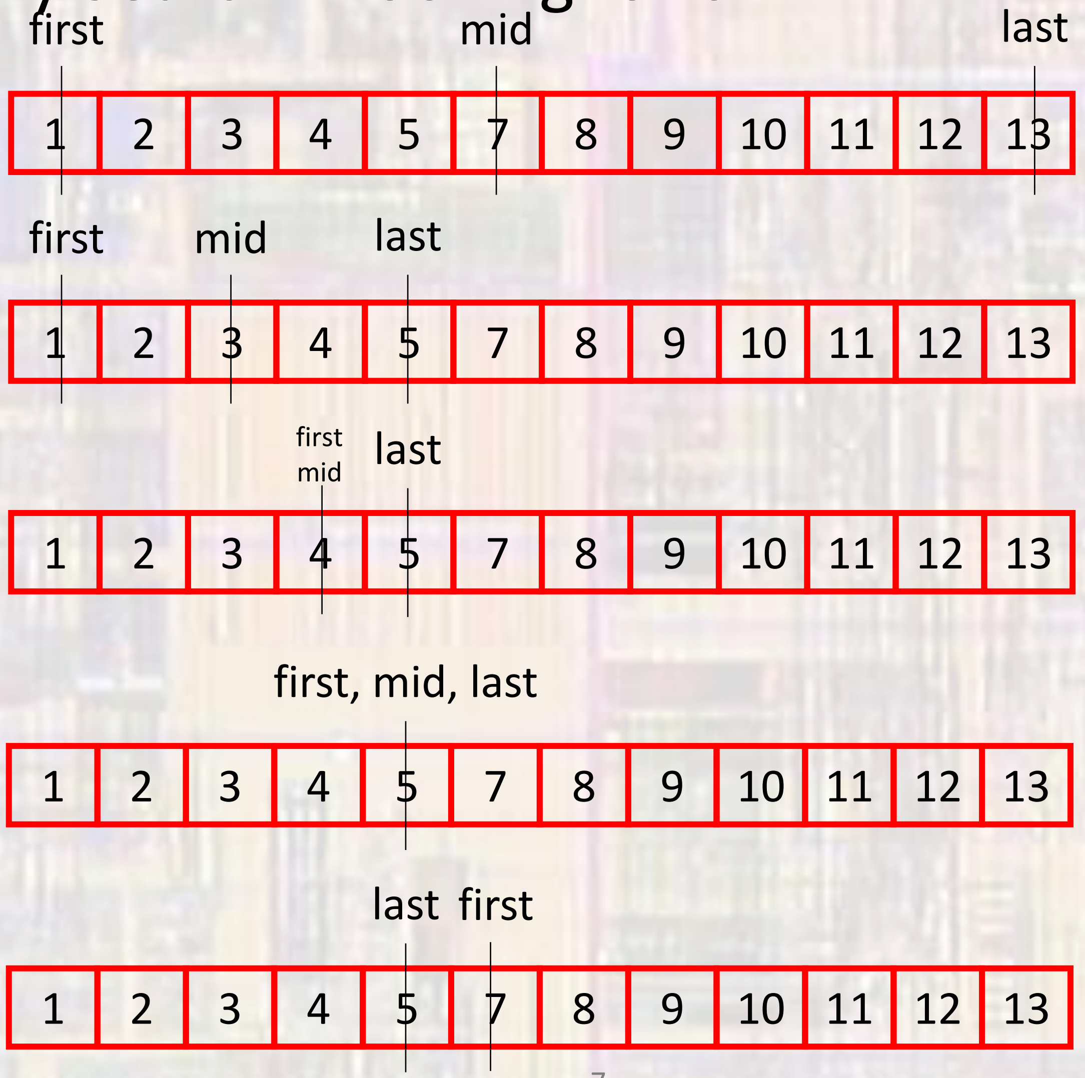
target = mid

Set first = last + 1
First > Last → end

If mid = target → found → Return mid (index)
else not found

Array Applications

- Binary Search – looking for 6



target < mid

Set last to mid-1
Reset mid

target > mid

Set first to mid+1
Reset mid

target > mid

Set first to mid+1
Reset mid

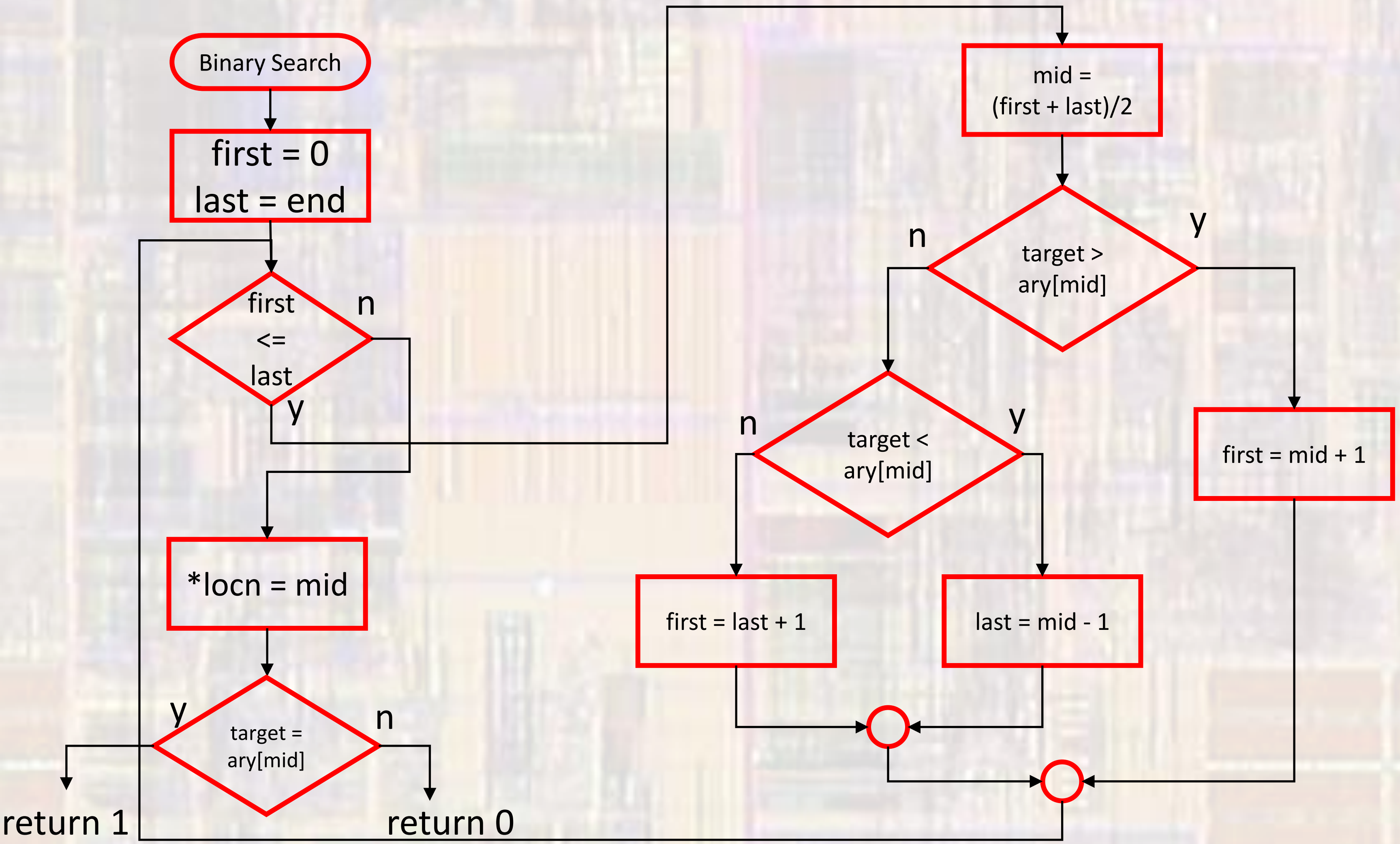
target > mid

Set first to mid+1

first > last
STOP – not found
return null

Array Applications

- Binary Search – implementation



Array Applications

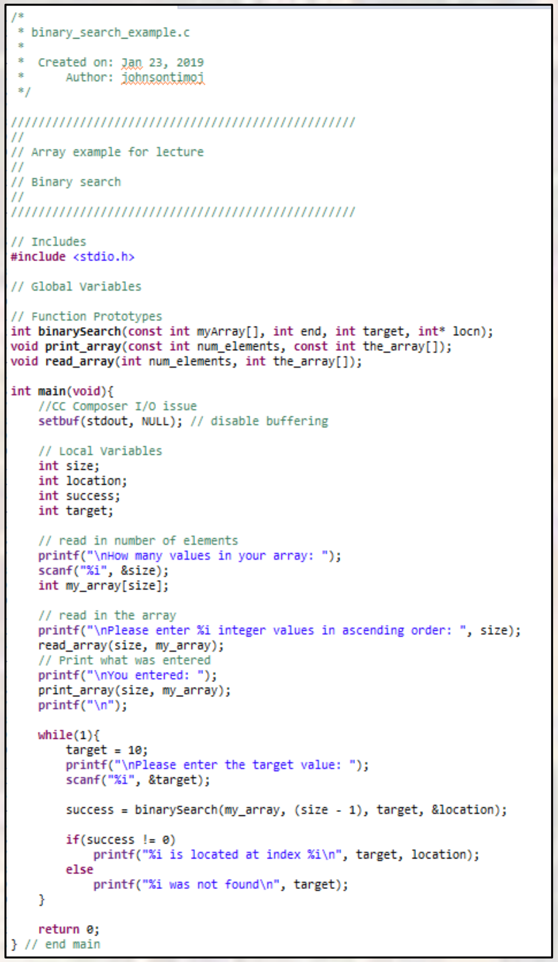
- Binary Search – implementation

```
int binarySearch(const int myArray[], int end, int target, int* locn){
    // Binary Search Function
    //
    // Inputs: Array to sort, index of last element,
    //         value to search for, pointer to location
    //         to store the index of the value if found
    // Outputs: Returns 1 if value found, 0 if not
    //         Modifies the value corresponding to the pointer
    //
    // local variables
    int first;
    int mid;
    int last;
```

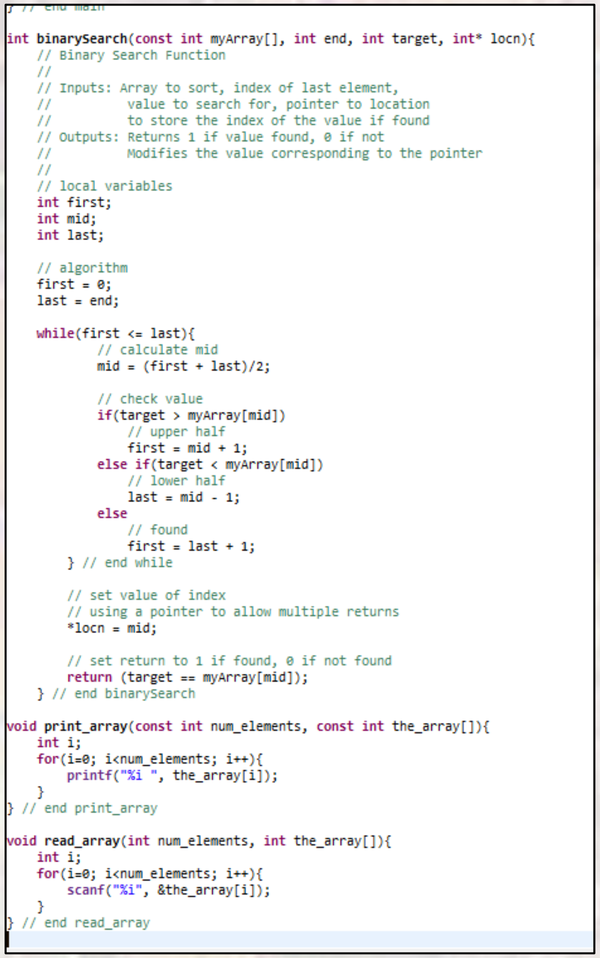
```
    // algorithm
    first = 0;
    last = end;

    while(first <= last){
        // calculate mid
        mid = (first + last)/2;

        // check value
        if(target > myArray[mid])
            // upper half
            first = mid + 1;
        else if(target < myArray[mid])
            // lower half
            last = mid - 1;
        else
            // found
            first = last + 1;
    } // end while

    // set value of index
    // using a pointer to allow multiple returns
    *locn = mid;

    // set return to 1 if found, 0 if not found
    return (target == myArray[mid]);
} // end binarySearch
```

Array Applications

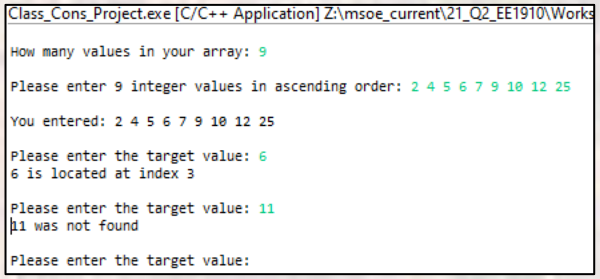
- Binary Search – usage



```c
/*
 * binary_search_example.c
 *
 *  Created on: Jan 23, 2019
 *      Author: johnsontimoj
 */

/////////////////////////////////////////////////
//
// Array example for lecture
//
// Binary search
//
/////////////////////////////////////////////////

// Includes
#include <stdio.h>

// Global Variables

// Function Prototypes
int binarySearch(const int myArray[], int end, int target, int* locn);
void print_array(const int num_elements, const int the_array[]);
void read_array(int num_elements, int the_array[]);

int main(void){
    //CC Composer I/O issue
    setbuf(stdout, NULL); // disable buffering

    // Local Variables
    int size;
    int location;
    int success;
    int target;

    // read in number of elements
    printf("\nHow many values in your array: ");
    scanf("%i", &size);
    int my_array[size];

    // read in the array
    printf("\nPlease enter %i integer values in ascending order: ", size);
    read_array(size, my_array);
    // Print what was entered
    printf("\nYou entered: ");
    print_array(size, my_array);
    printf("\n");

    while(1){
        target = 10;
        printf("\nPlease enter the target value: ");
        scanf("%i", &target);

        success = binarySearch(my_array, (size - 1), target, &location);

        if(success != 0)
            printf("%i is located at index %i\n", target, location);
        else
            printf("%i was not found\n", target);
    }

    return 0;
} // end main
```



```c
} // end main

int binarySearch(const int myArray[], int end, int target, int* locn){
    // Binary Search Function
    //
    // Inputs: Array to sort, index of last element,
    //          value to search for, pointer to location
    //          to store the index of the value if found
    // Outputs: Returns 1 if value found, 0 if not
    //          Modifies the value corresponding to the pointer
    //
    // local variables
    int first;
    int mid;
    int last;

    // algorithm
    first = 0;
    last = end;

    while(first <= last){
            // calculate mid
            mid = (first + last)/2;

            // check value
            if(target > myArray[mid])
                // upper half
                first = mid + 1;
            else if(target < myArray[mid])
                // lower half
                last = mid - 1;
            else
                // found
                first = last + 1;
        } // end while

        // set value of index
        // using a pointer to allow multiple returns
        *locn = mid;

        // set return to 1 if found, 0 if not found
        return (target == myArray[mid]);
    } // end binarySearch

void print_array(const int num_elements, const int the_array[]){
    int i;
    for(i=0; i<num_elements; i++){
        printf("%i ", the_array[i]);
    }
} // end print_array

void read_array(int num_elements, int the_array[]){
    int i;
    for(i=0; i<num_elements; i++){
        scanf("%i", &the_array[i]);
    }
} // end read_array
```



```
Class_Cons_Project.exe [C/C++ Application] Z:\msoe_current\21_Q2_EE1910\Works

How many values in your array: 9

Please enter 9 integer values in ascending order: 2 4 5 6 7 9 10 12 25

You entered: 2 4 5 6 7 9 10 12 25

Please enter the target value: 6
6 is located at index 3

Please enter the target value: 11
11 was not found

Please enter the target value:
```

Array Applications

- Binary Search

  - Efficiency -

    - Ceil(Log$_2$(N))