CSCI E-50 WEEK 3

TERESA LEE
teresa@cs50.net
FEBRUARY 12 2018

TODAY

- Recap: arrays & debug50
- asymptotic notation (O, Ω)
- linear search
- binary search
- bubble sort
- Selection sort
- recursion
- pset3

QUESTIONS?

ARRAY

To initialize an array

```
int score[0] = 0; // zero index all arrays!
int score[1] = 1;
int score[2] = 2;
```

or

```
int score[] = {0, 1, 2}; // size based on the number of entries
```

```
// make an array

<datatype> <name>[<size>];

    char alpha[26];

    Int score[5];

// iterate over the array's members
```

STRING

Just an array of characters!

Final index of a string in C is the null terminator ‘\0’, which tells the system that the string is over.

```
// declare string

String s = “teresa”;

// what happens when I index into
s[i]?

Printf(“%c\n”, s[0]);

Printf(“%c\n”, s[1]);

Printf(“%c\n”, s[6]);

Printf(“%c\n”, s[7]);
```

ARRAY

Adventages?

- Constant-time acess given index
- Space efficient
- Aility to iterate through all elements

Disadvantages?

- Elements of same type only
- Fixed size!

Debug50

`ex_debug.c`

Computational Complexity

- Complexity? Time & Space
- Algorithm's running time
  - O (upper bound)
  - Ω (lower bound)
  - Θ upper and lower bounds are the same

n	$f(n) = n^3$	$f(n) = n^3 + n^2$	$f(n) = n^3 - 8n^2 + 20n$
1	1	2	13
10	1,000	1,100	400
1,000	1,000,000,000	1,001,000,000	992,020,000
1,000,000	1.0×10^{18}	1.000001×10^{18}	9.99992×10^{17}

Computational Complexity

Computational Complexity

$O(1)$	constant time
$O(\log n)$	logarithmic time
$O(n)$	linear time
$O(n \log n)$	linearithmic time
$O(n^2)$	quadratic time
$O(n^c)$	polynomial time
$O(c^n)$	exponential time
$O(n!)$	factorial time
$O(\infty)$	infinite time

SEARCH

Linear search

- O(n) , Ω(1)

```
// initialize an int array

Int haystack[] = {3, 2, 6}


// find the needle by using the
linear search
```

Let's Look at an Example

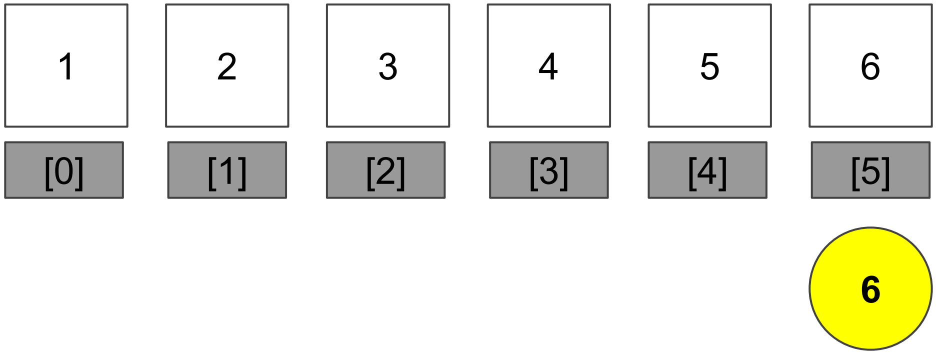
Linear search:

What is the upper bound? $O(n)$

What is the lower bound? $\Omega(1)$

Lenear Search

Find:



Let's Look at an Example

`Ex1_Linear`

Binary search

- O(log n), Ω(1)
- Requirement? - [redacted]

```
//pseudocode for binary search
```

Binary Search

Find: **1**

1	2	3	4	5	6
[0]	[1]	[2]	[3]	[4]	[5]

Start = 0
End = n-1
mid = (start+end) / 2

Let’s Look at an Example

`Ex2_binary`

`What’s another way to perform this binary search?`

`recursion!`

SORT

Selection Sort

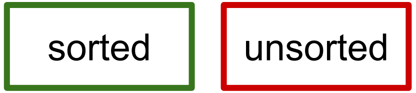
In selection sort, the idea of the algorithm is to find the smallest unsorted element and add it to the end of the sorted list.

<pseudocode>
Repeat until no unsorted elements remain:

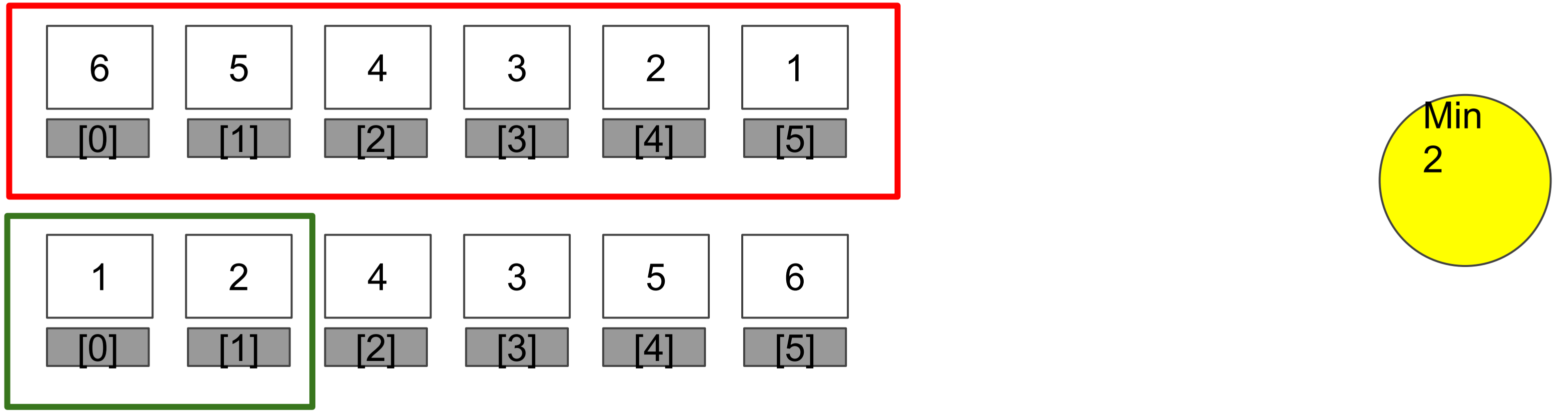
- Search the unsorted part of the data to find the smallest value
- Swap the smallest found value with the first element of the unsorted part

- $O(n^2)$, $\Omega(n^2)$
- Temporary variables?

```
[50, 1, 51, 4, 42]
                    ->
                        [1, 50, 51, 4, 42]    (1)
                        [1, 4, 51, 50, 42]    (2)
                        [1, 4, 42, 50, 51]    (3)
                        [1, 4, 42, 50, 51]    (4)
```



Selection Sort



Let’s Look at an Example

`ex3_selection`

Bubble Sort

In bubble sort, the idea of the algorithm is to move higher valued elements generally towards the right and lower valued elements generally towards the left.

<pseudocode>
Set swap counter to a non-zero value
Repeat until the swap counter is 0:

- Reset swap counter to 0
- Look at each adjacent pair
- If the two adjacent elements are not in order, swap them and increase the swap counter by 1.

- O(n^2), Ω(n)
- Pair-wise sorting
- What variables do we need?

```
[4, 1, 7, 10, 3]
                 ->
                      [1, 4, 7, 3, 10]    (1)
                      [1, 4, 3, 7, 10]    (2)
                      [1, 3, 4, 7, 10]    (3)
                      [1, 3, 4, 7, 10]    (4)
```

Bubble Sort

[0]	[1]	[2]	[3]	[4]	[5]
6	5	4	3	2	1

Swap = 5

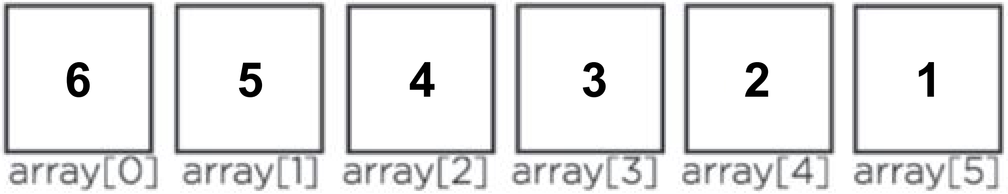
[0]	[1]	[2]	[3]	[4]	[5]
5	4	3	2	1	6

Let's Look at an Example

Bubble Sort:

What is the lower bound? [redacted]

Let’s Look at Example



Bubble Sort:

What is the upper bound? [redacted]

Let’s Look at an Example

`ex4_bubble`

Insertion Sort

In insertion sort, the idea of the algorithm is to build your sorted array in place, shifting elements out of the way if necessary to make room as you go.

<pseudocode>
Call the first element of the array "sorted"
Repeat until all the elements are sorted:

- Look at the next unsorted element. Insert into the "sorted" portion by shifting the requisite number of elements.

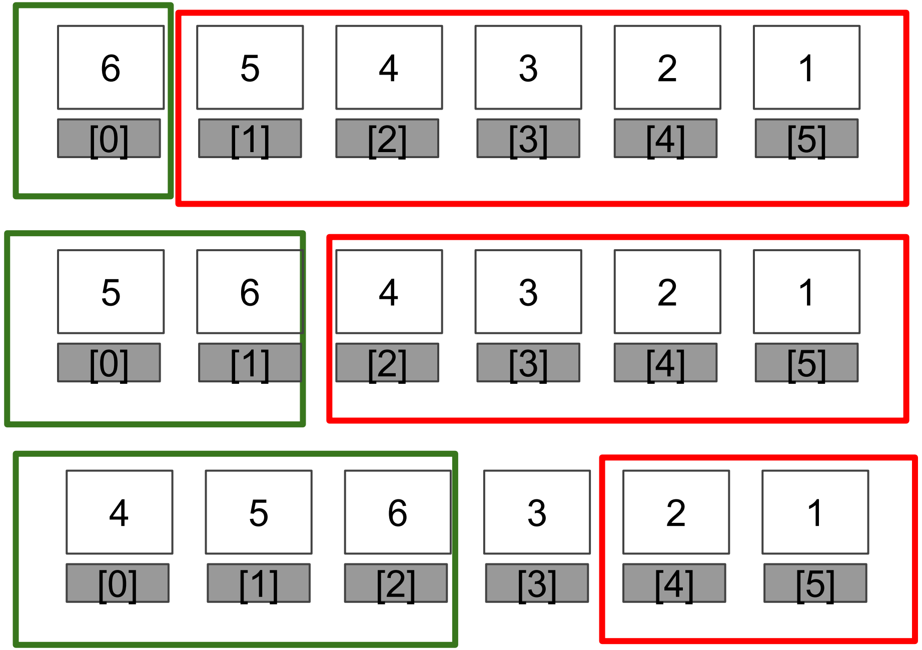
- $O(n^2)$, $\Omega(n)$

```
[2, 8, 1, 4, 3]
           ->
              [2, 8, 1, 4, 3]    (1)
              [1, 2, 8, 4, 3]    (2)
              [1, 2, 4, 8, 3]    (3)
              [1, 2, 3, 4, 8]    (4)
```

Insertion Sort

sorted

unsorted



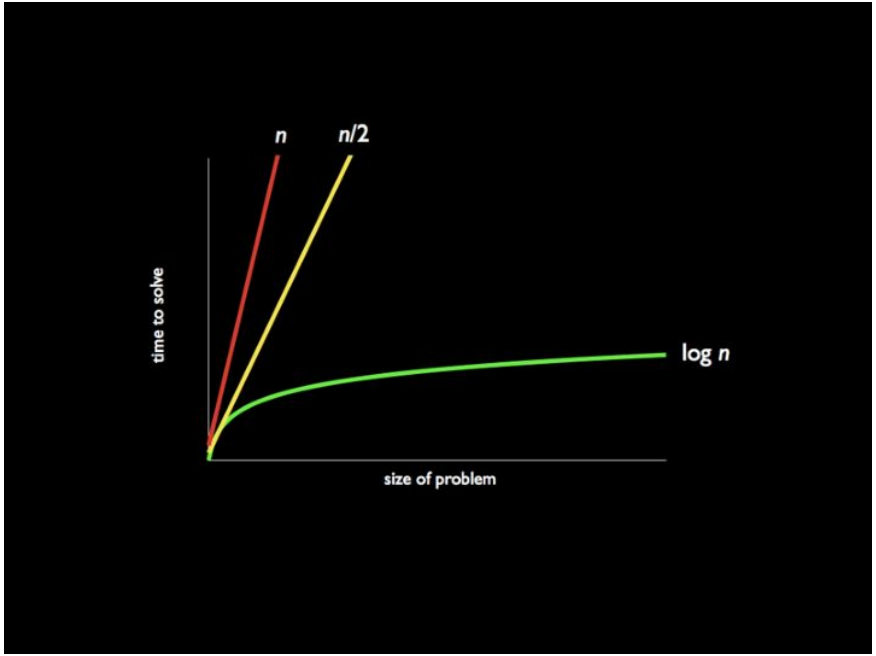
Let’s Look at an Example

`ex5_insertion`

Running Time Summary

Algorithm	Big O	Big Ω
linear search	$O(n)$	$\Omega(1)$
binary search	$O(\log(n))$	$\Omega(1)$
bubble sort	$O(n^2)$	$\Omega(n)$
insertion sort	$O(n^2)$	$\Omega(n)$
selection sort	$O(n^2)$	$\Omega(n^2)$

Running Time Summary



Recursion

- Recursive function calls itself as part of execution
- Cyclical use of a function
  - Every time you make a recursive call, there is a new stack frame
- You need:
  - Base case: when triggered, terminates the recursive process
  - Recursive case: where recursive process will actually occur

Let's Look at an Example

factorial(1)	1
factorial(2)	2 * 1 = 2
factorial(3)	3 * fact (2)
factorial(4)	4 * 3 * 2 * 1 = 4 * fact (3) = 24
factorial(5)	5 * 4 * 3 * 2 * 1 = 5 * fact (4) = 120
factorial(n)	n * factorial(n-1) for all n >= 1

Let's Look at an Example - Itervative Factorial

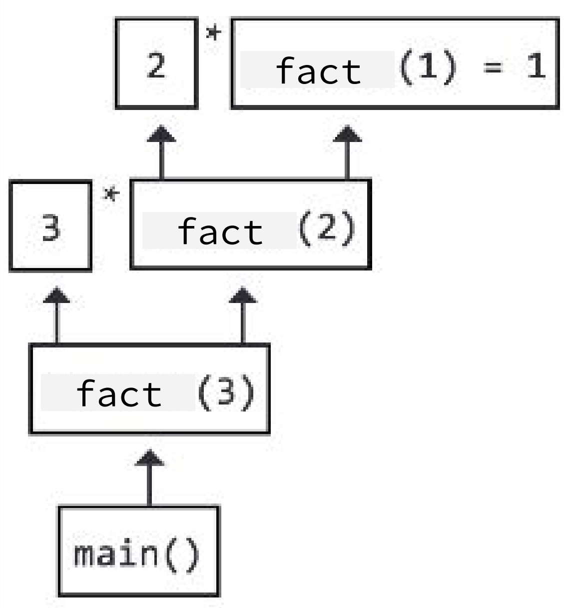
```
int fact2(int n)
{
    int product = 1;
    while (n > 0)
    {
        product *= n;
        N--;
    }
    return product;
}
```

```
int fact(int n)
{
    // Base case

    // Recursive case
}
```

```
int fact(int n)
{
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}
```

Let's Look at an Example



Downside?
It can be memory-intensive!

While a recursive algorithm is not always required, it frequently looks much more beautiful and (though recursion is not itself a simple concept), a recursive implementation usually looks much simpler once coded.

Let’s Look at an Example

`ex6_recursion`

pset3

Music

Shorts to Watch

- Computational Complexity
- Selection Sort
- Bubble Sort
- Insertion Sort
- Linear Search
- Binary Search
- Algorithms Summary
- Debugging
- Recursion

Final words on pset3

- Read background
- Read specification
- Watch Brian's walkthrough
- Remember to comment your codes!
    - Style Guide
- Test with check50!