This is CS50

Section, Week 3

TA: Andi Peng

Agenda

- Announcements
- GDB
- Sorts (selection, insertion, bubble, merge)
- Asymptotic Notation (O, Ω)
- Binary Search
- pset3

Announcements

- Grading
  - Commenting
  - Make sure to test your code with Check50!
  - Postmortems
  - Late psets will be zeroed
- Office hours
  - Come early in the week (woo Mondays)
  - Come prepared to ask questions
- Quiz 0: October 14 or 15

GDB

Up to now, we’ve been debugging using printf statements…

GDB

Your new best friend!

- Set a breakpoint
- Next
- Step over
- Step into

Selection Sort

Algorithm

1. Find the smallest unsorted value

2. Swap that value with the first unsorted value

3. Repeat from Step 1 if there are still unsorted items

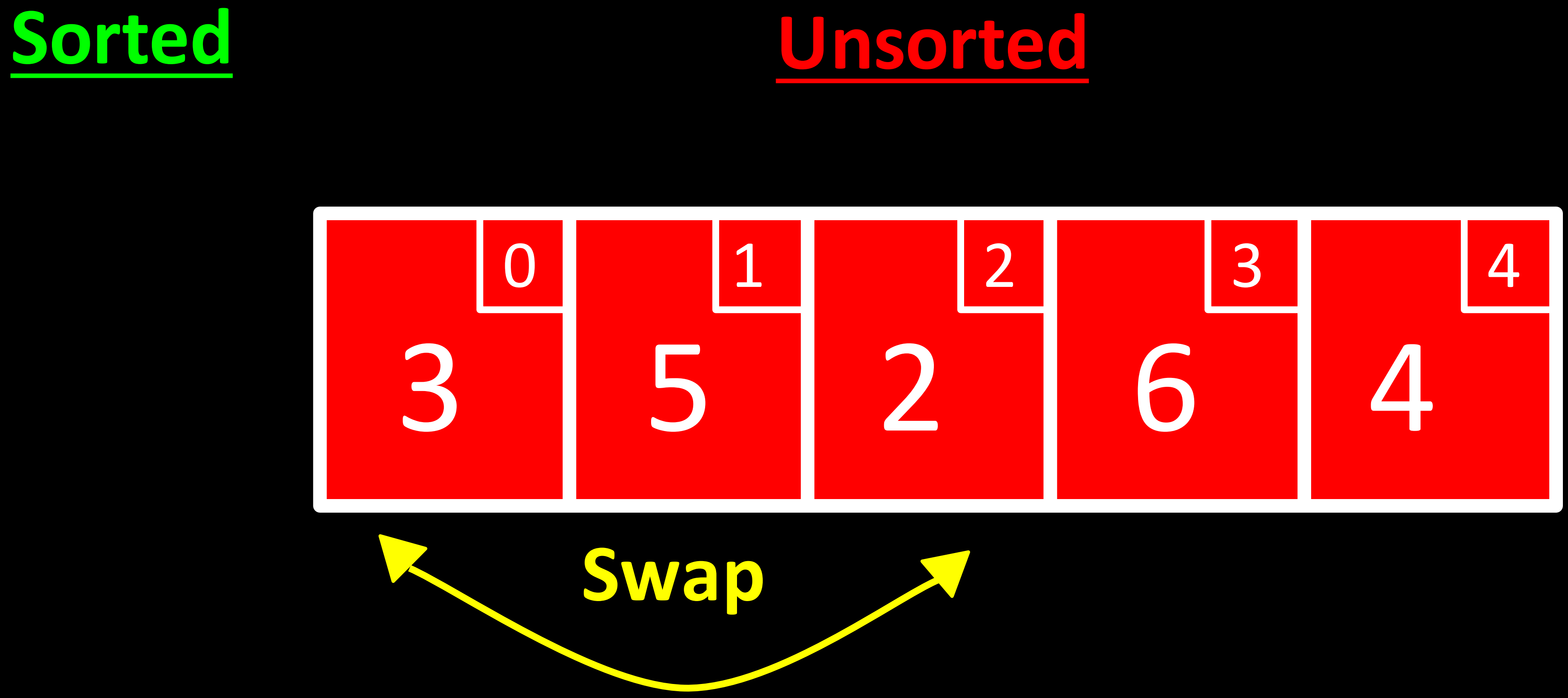
All values start as Unsorted

Sorted

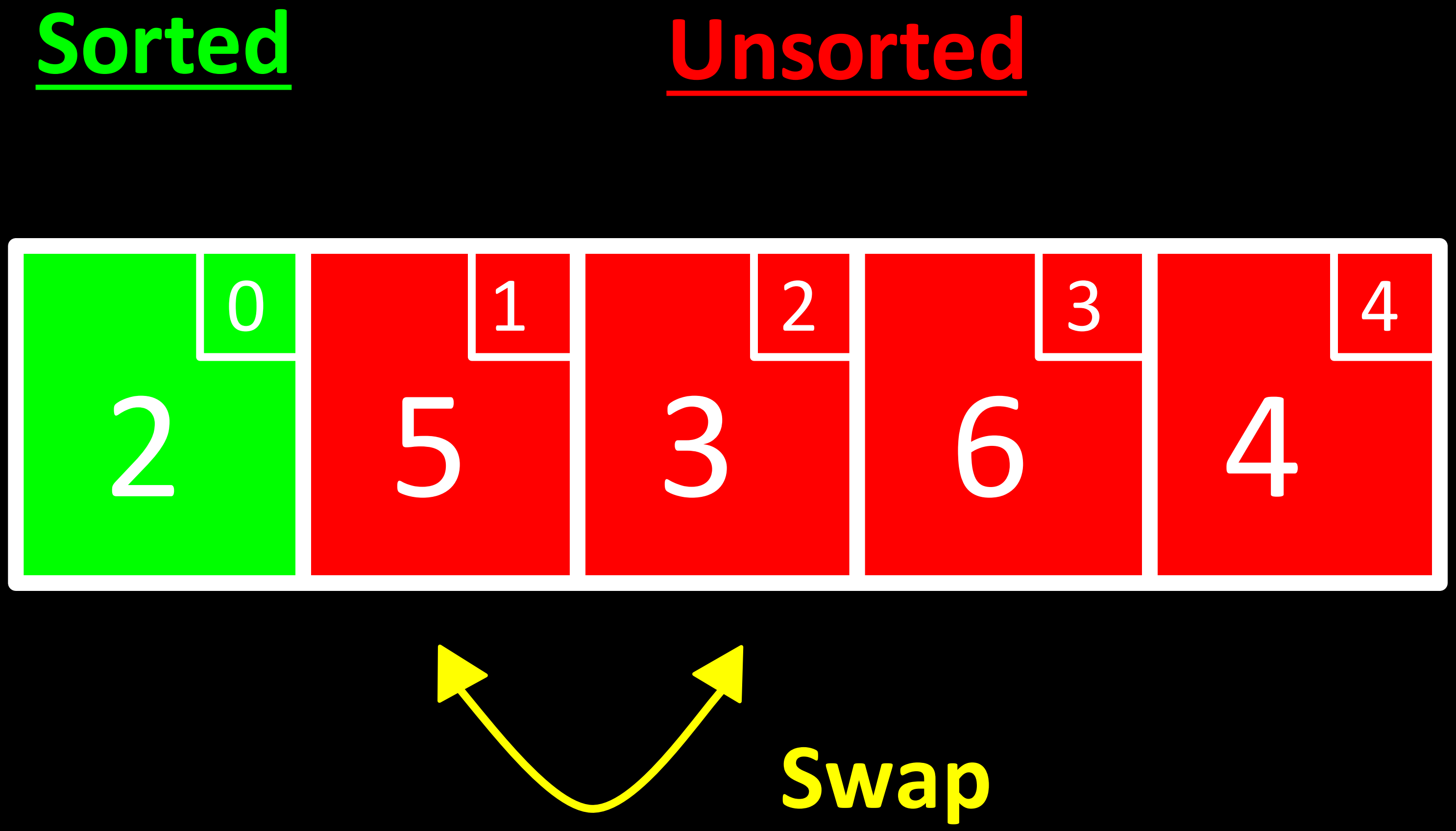
Unsorted

0	1	2	3	4
3	5	2	6	4

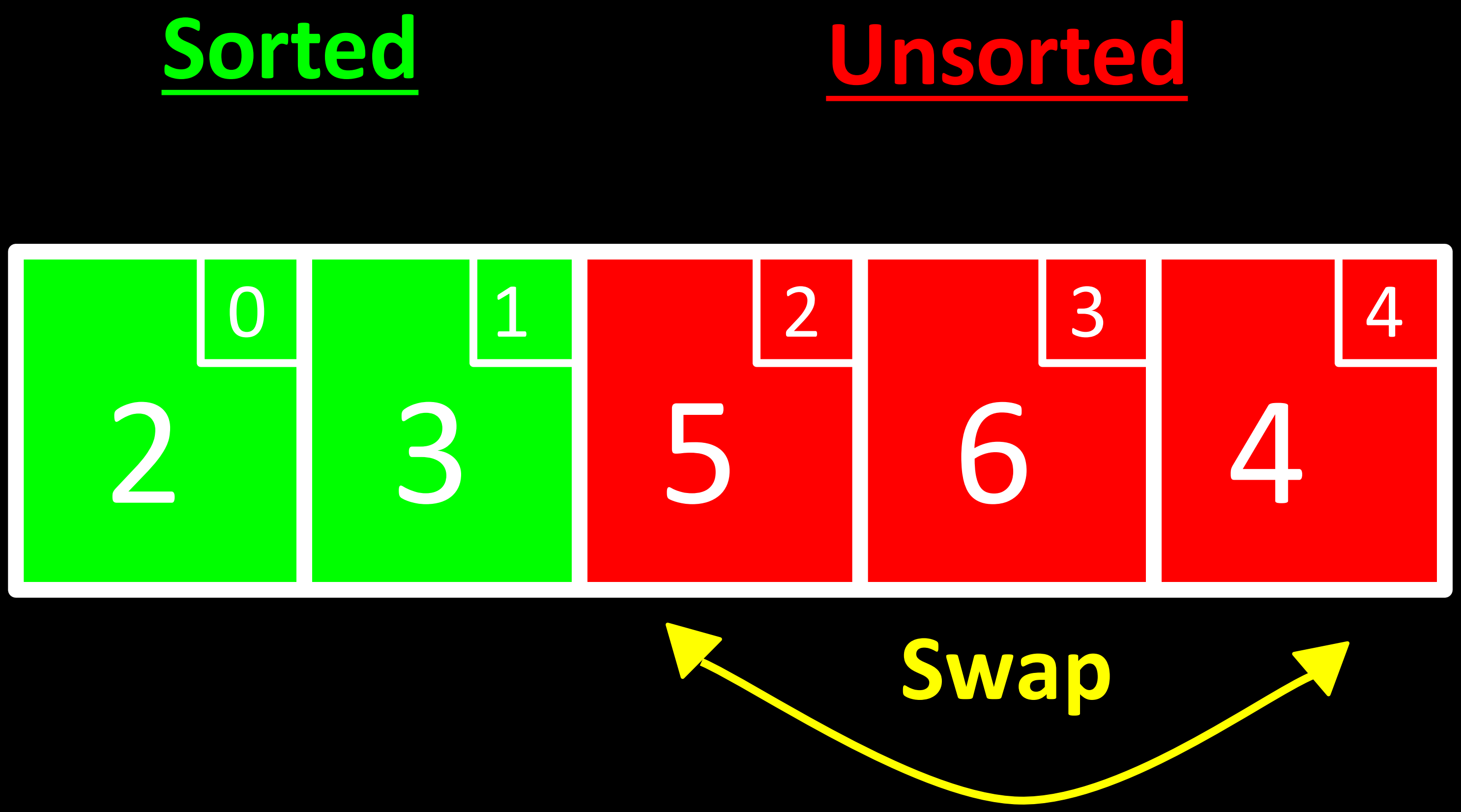
First pass:
2 is smallest, swap with 3



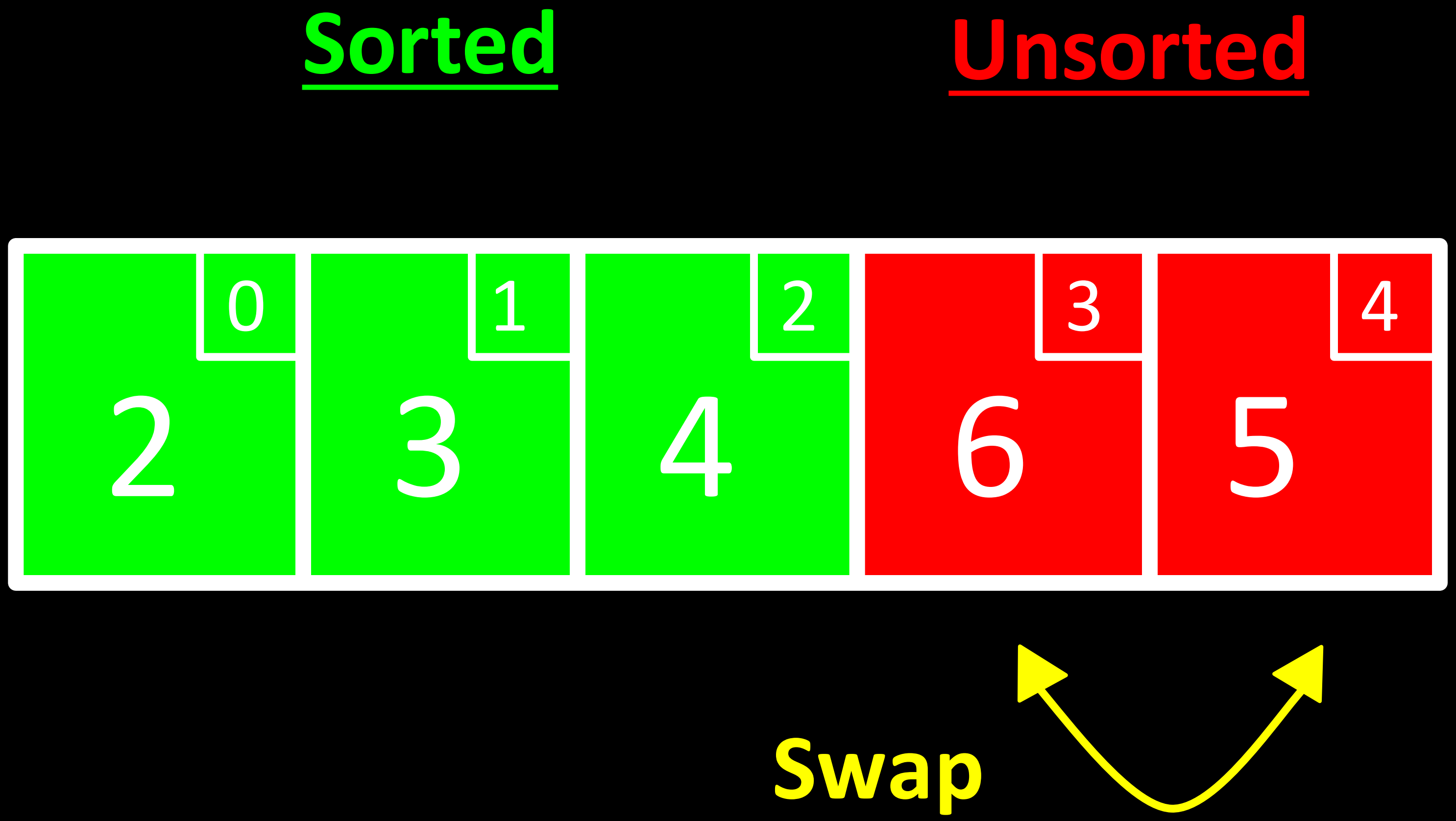
Second pass:
3 is smallest, swap with 5



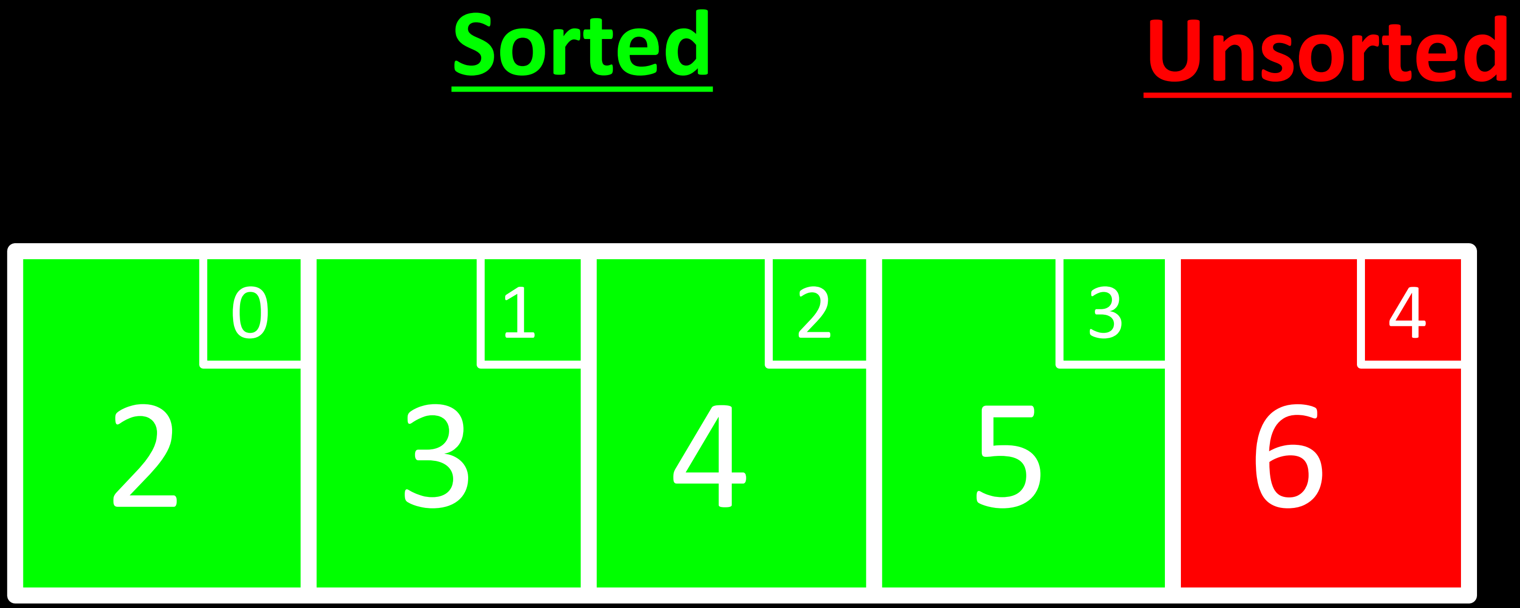
Third pass:
4 is smallest, swap with 5



Fourth pass:
5 is smallest, swap with 6



Fifth pass:
6 is the only value left, done!



Pseudocode Time!

```
for i = 0 to n - 1
        min = i
        for j = i to n - 1
            if array[j + 1] < array[min]
                    min = j + 1;
        if min  != i
                swap array[min] and array[i]
```

What's the best case runtime of selection sort?

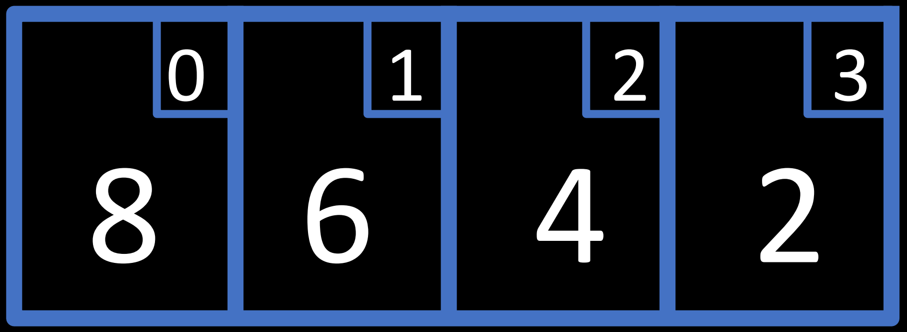
What's the worst case runtime of selection sort?

What's the expected runtime of selection sort?

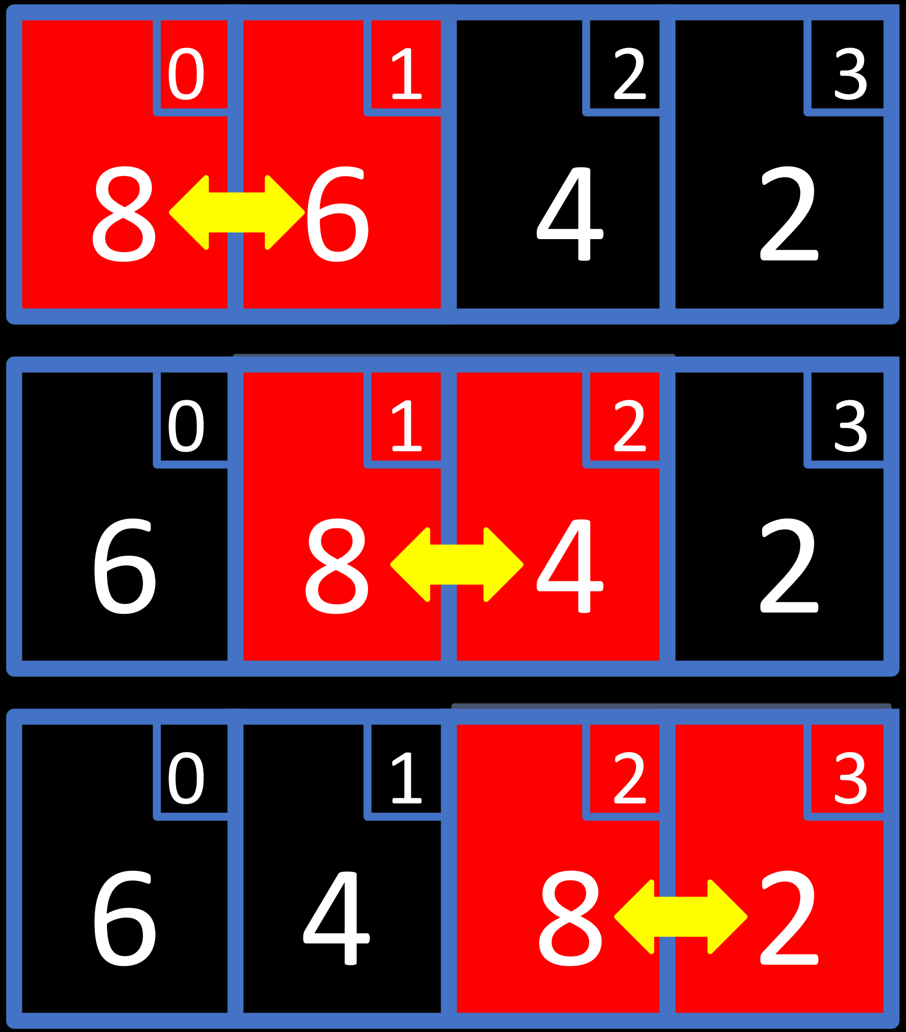
Bubble Sort

Algorithm

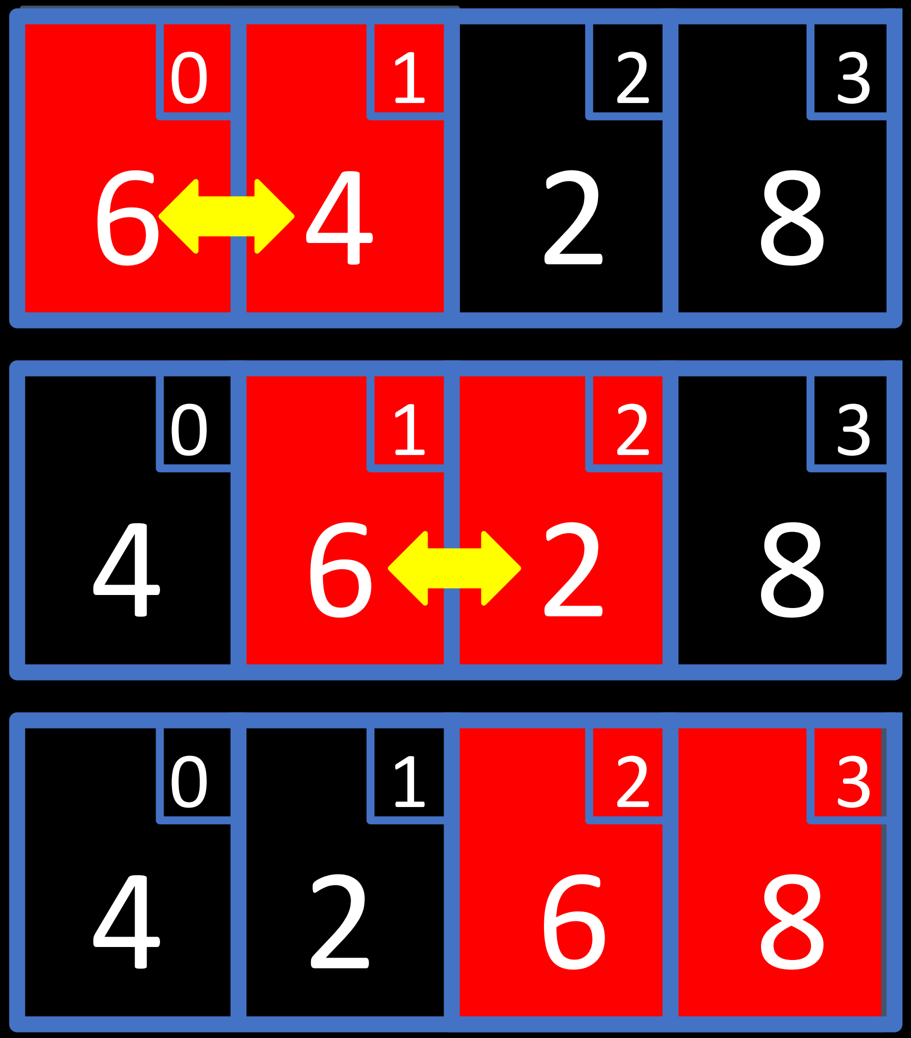
- **1. Step through entire list, swapping adjacent values if not in order**
- **2. Repeat from step 1 if any swaps have been made**


0
1
2
3
8
6
4
2

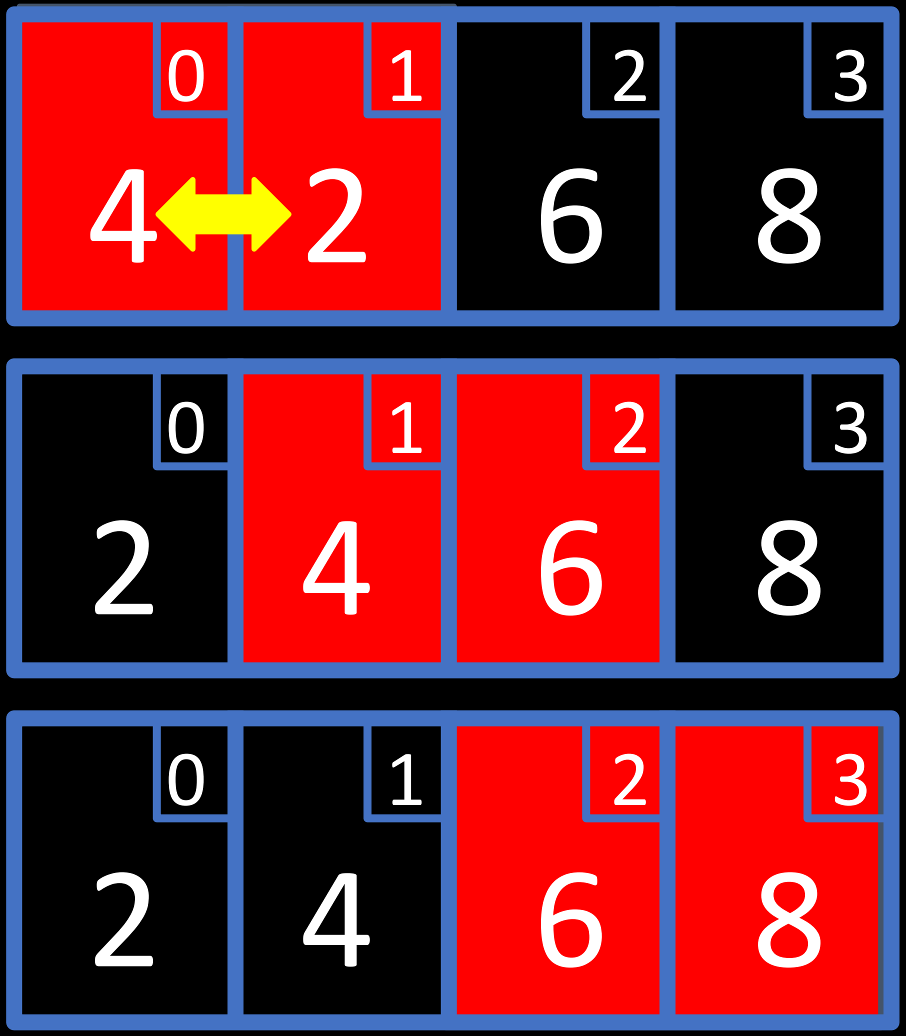
First pass: 3 swaps



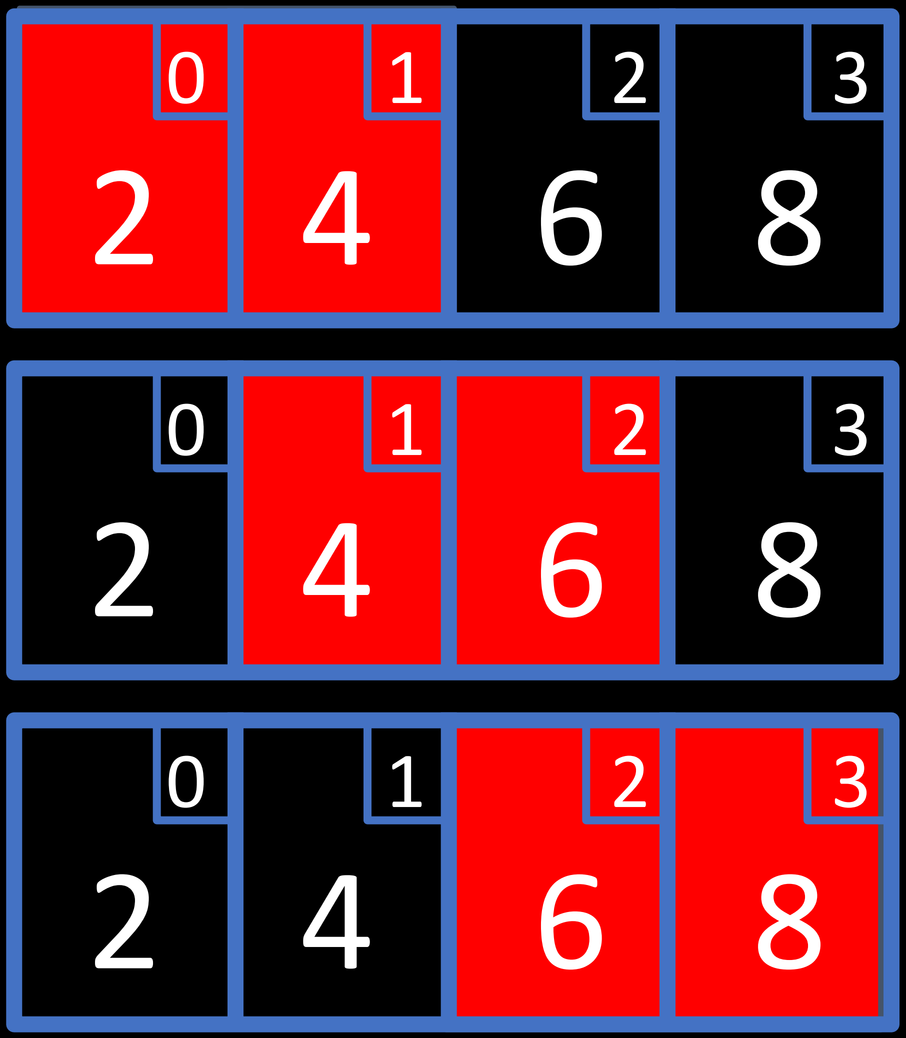
Second pass: 2 swaps



Third pass: 1 swap



Fourth pass: 0 swaps



```
initialize counter
do
{
    set counter to 0

    iterate through entire array
        if array[n] > array[n+1]
            swap them
            increment counter
}
while (counter > 0)
```

What's the worst case runtime of bubble sort?

What's the best case runtime of bubble sort?

Insertion Sort

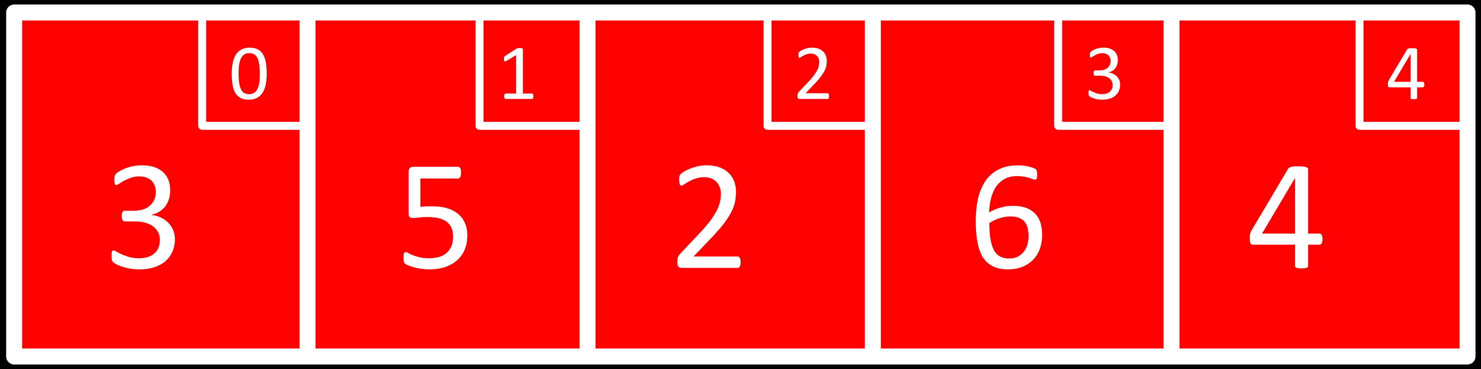
Algorithm

- **1. Data is divided into sorted and unsorted portions**
- **2. One by one, the unsorted values are inserted into their appropriate positions in the sorted subarray**

All values start as Unsorted

Sorted

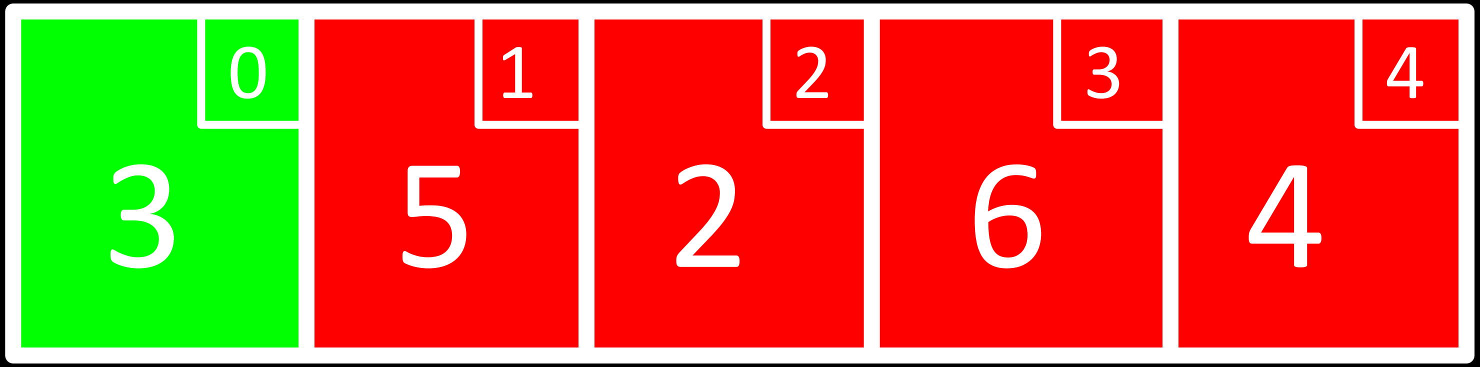
Unsorted



Add first value to Sorted

Sorted

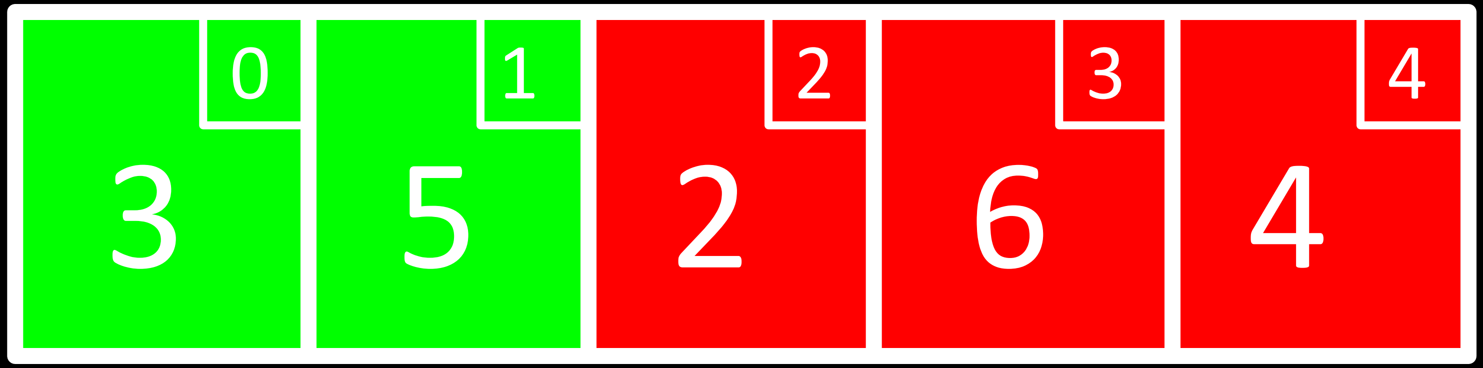
Unsorted



5 > 3
insert 5 to right of 3

Sorted

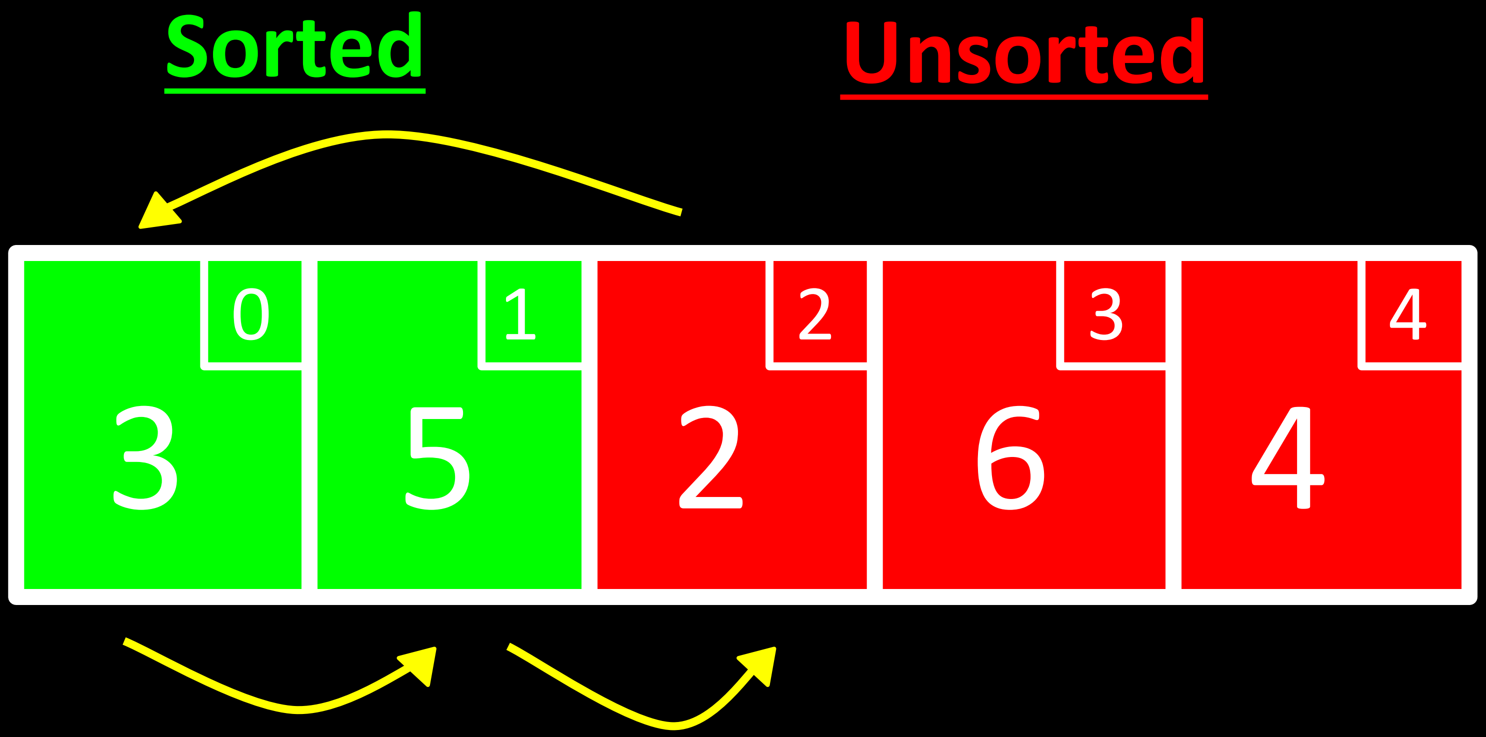
Unsorted



2 < 5 and 2 < 3

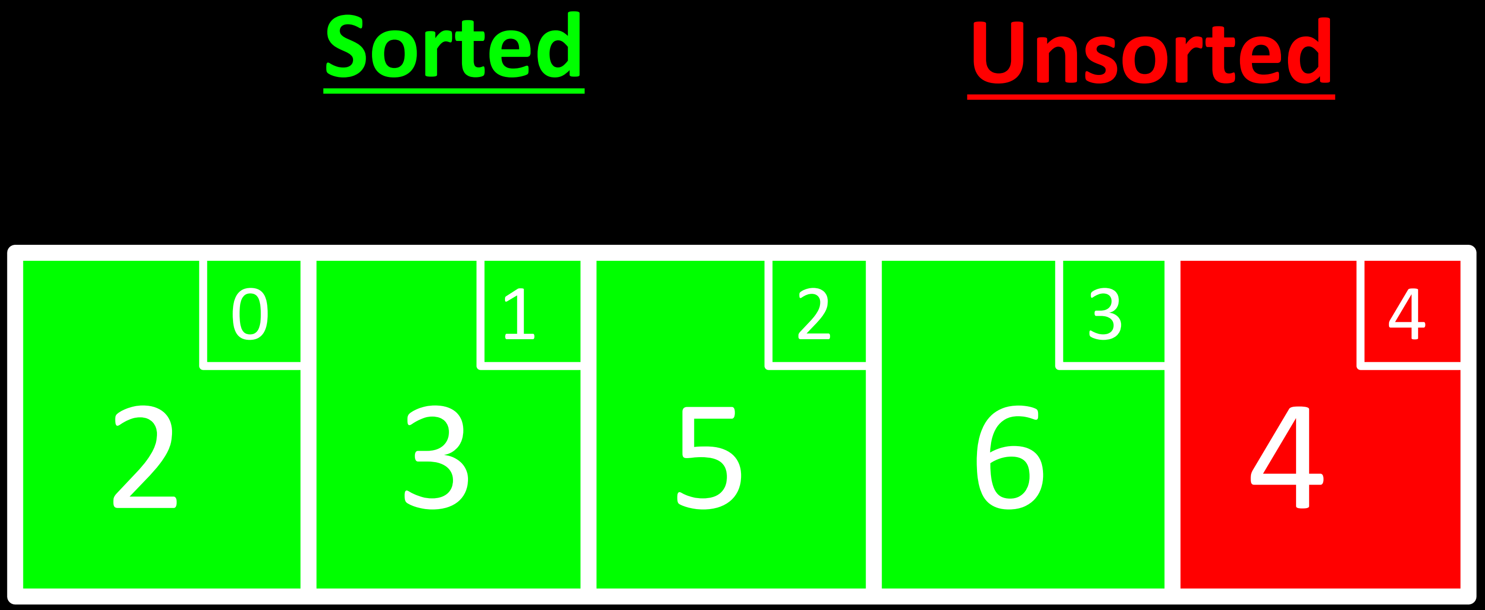
shift 3 and 5

insert 2 to left of 3



6 > 5

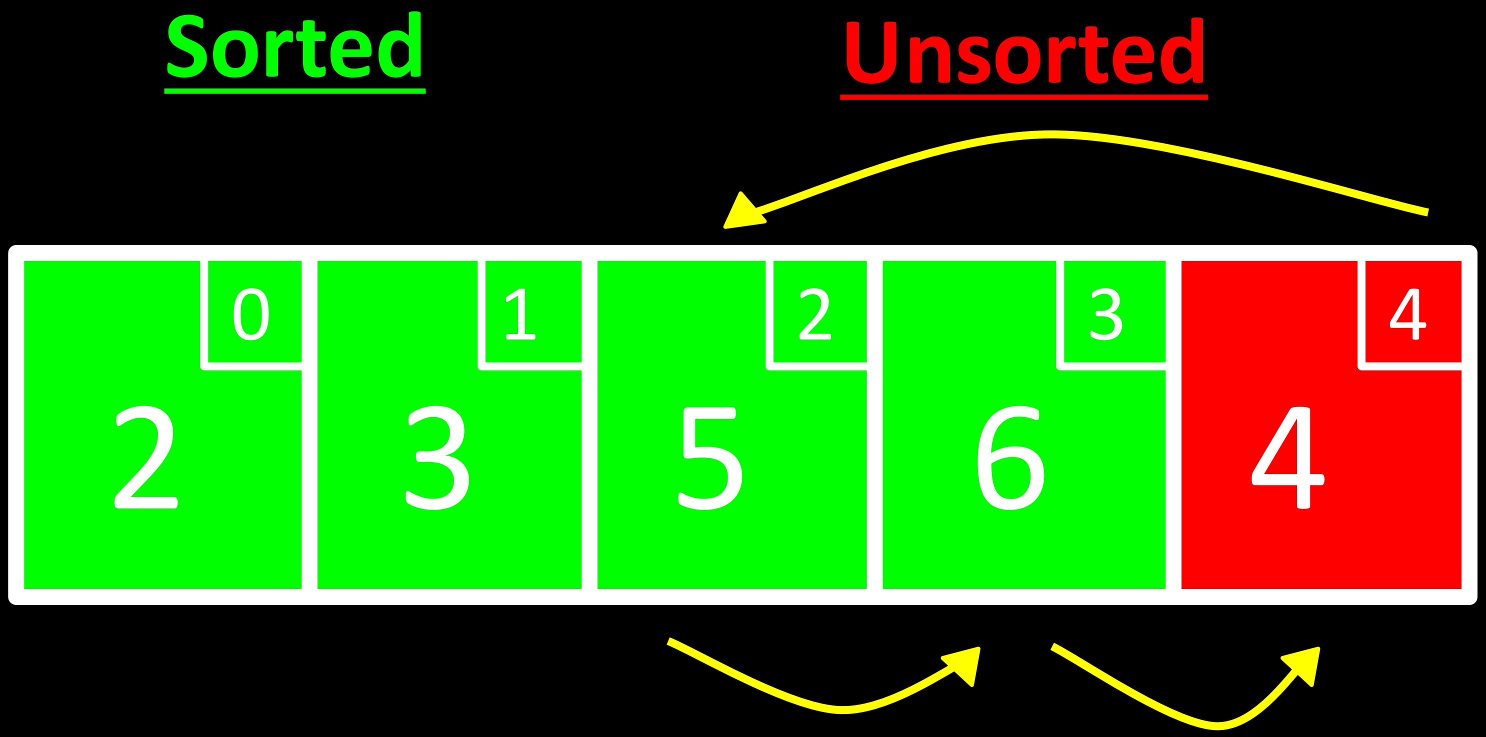
insert 6 to right of 5



4 < 6, 4 < 5, and 4 > 3

shift 5 and 6

insert 4 to right of 3



For each unsorted element n:

1. Determine where in sorted portion of the list to insert n

2. Shift sorted elements rightwards as necessary to make room for n

3. Insert n into sorted portion of the list

```
for i = 0 to n - 1
    element = array[i]
    j = i
    while (j > 0 and array[j - 1] > element)
        array[j] = array[j - 1]
        j = j - 1
    array[j] = element
```

What's the worst case runtime of insertion sort?

What's the best case runtime of insertion sort?

What’s the difference between these three types of sorts?

Merge Sort

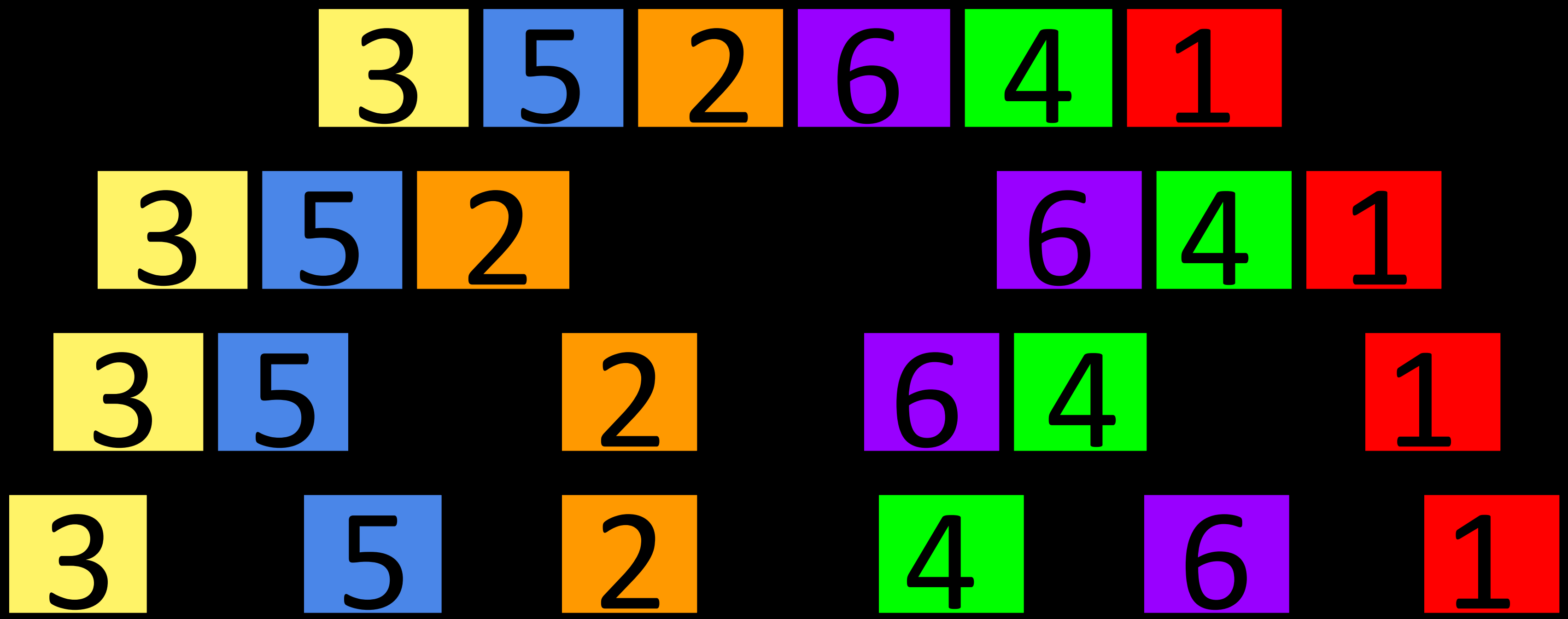
Algorithm

- **1. Divide an unsorted array in two**
- **2. Sort the two halves of that array recursively**

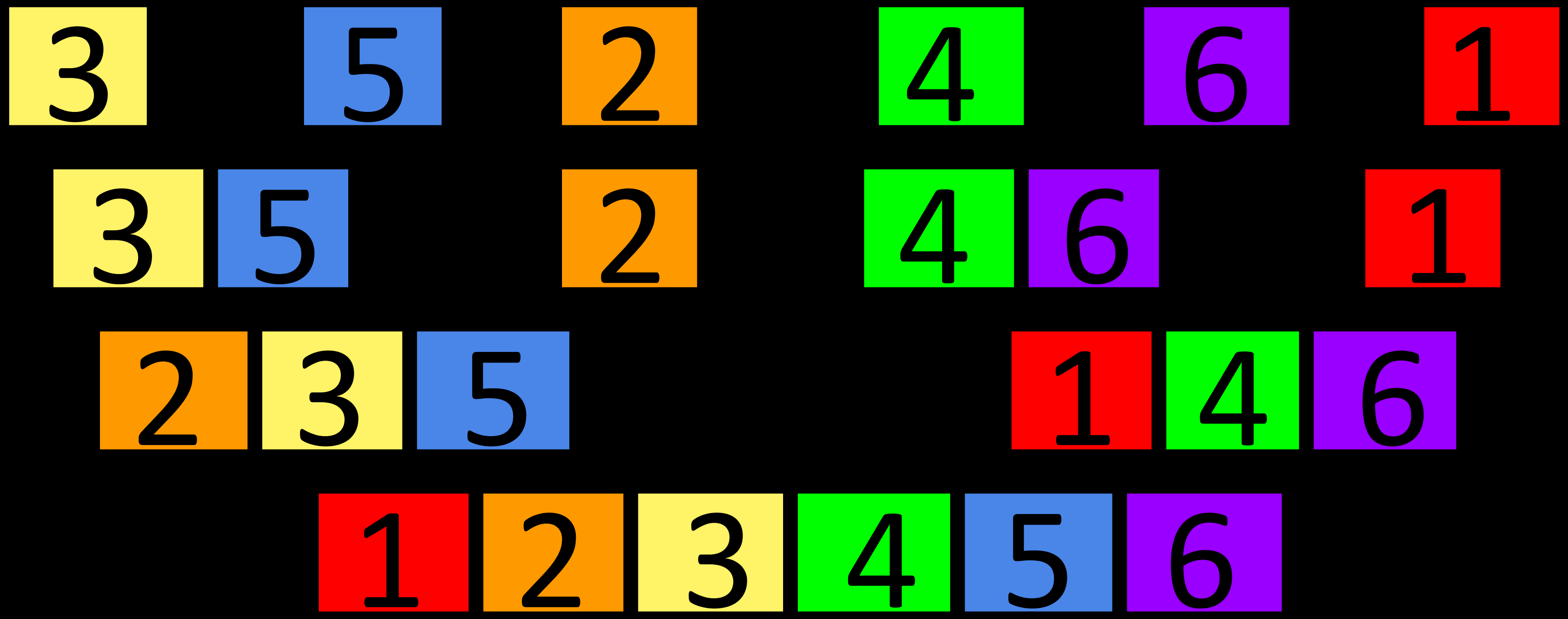
```
On input of n elements:
    If n < 2
        Return.
    Else
        Sort left half of elements.
        Sort right half of elements.
        Merge sorted halves.
```

3	5	2	6	4	1

Halve until each subarray is size 1



Merge Sorted Halves

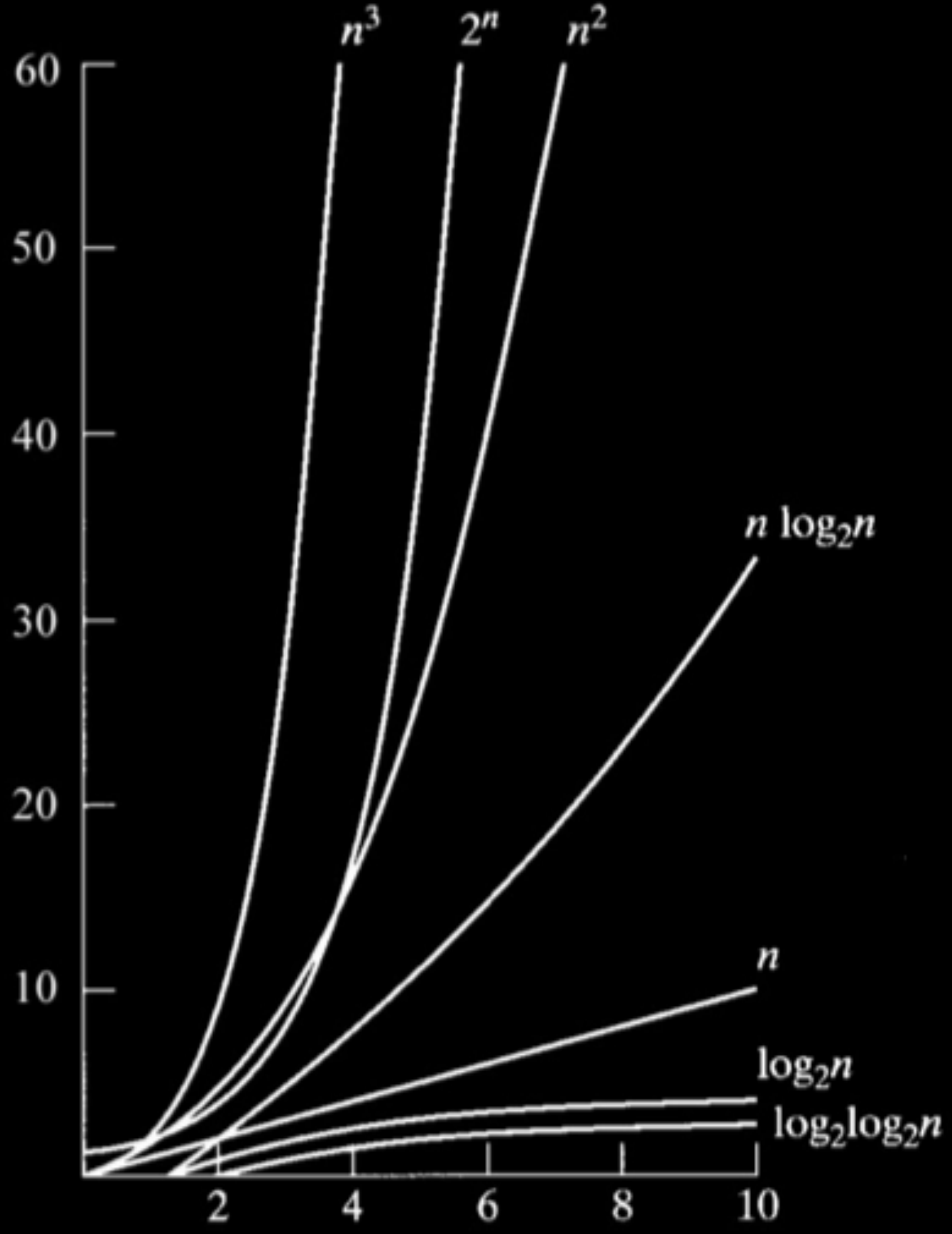


```
sort (int array[], int start, int end)
{
  if (end > start)
        {
            int middle = (start + end) / 2;

                sort(array, start, middle);
                sort(array, middle + 1, end);

                merge(array, start, middle, middle + 1, end);
        }
}
```

What's the best case runtime of merge sort?

What's the worst case runtime of merge sort?

What's the expected runtime of merge sort?



n^3
2^n
n^2
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
60
50
40
30
20
10
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

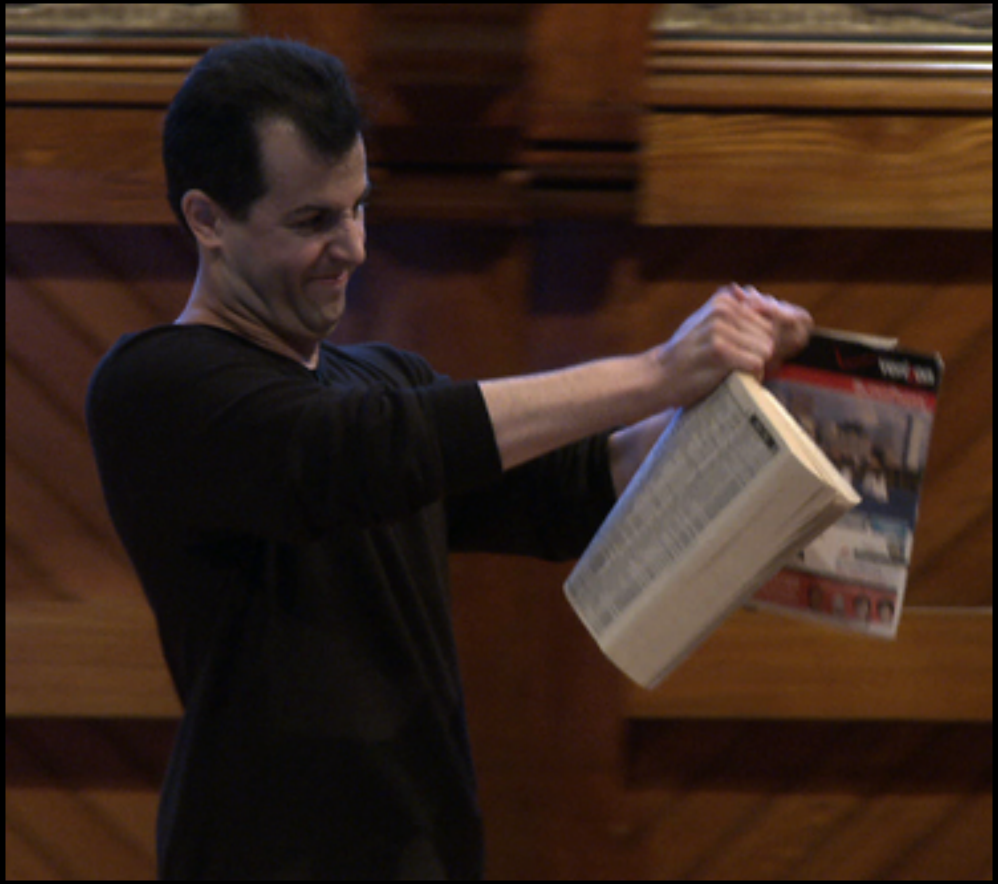
Searching

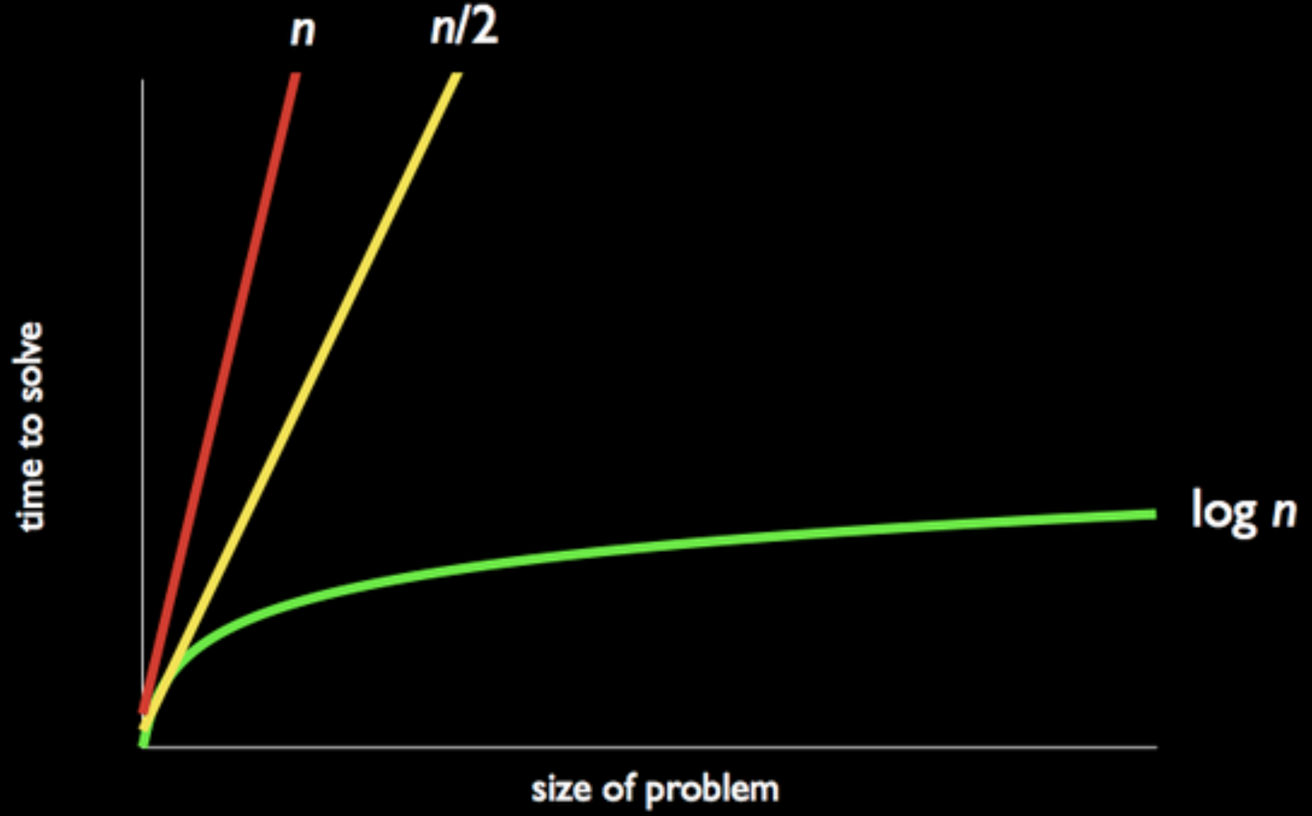
- Linear search: search every element of a list
- Binary Search: Divide and Conquer!

Searching

- Linear search: search every element of a list
- Binary Search: Divide and Conquer!

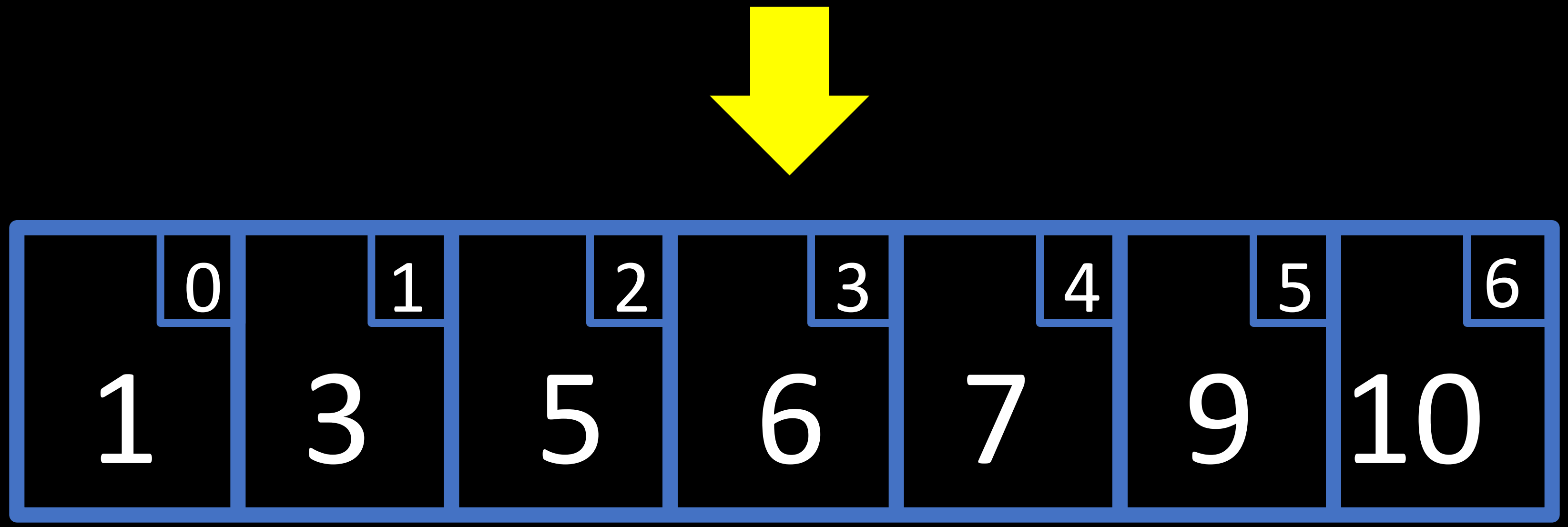
Binary Search




n
n/2
log n
time to solve
size of problem

Does the array contain 7?

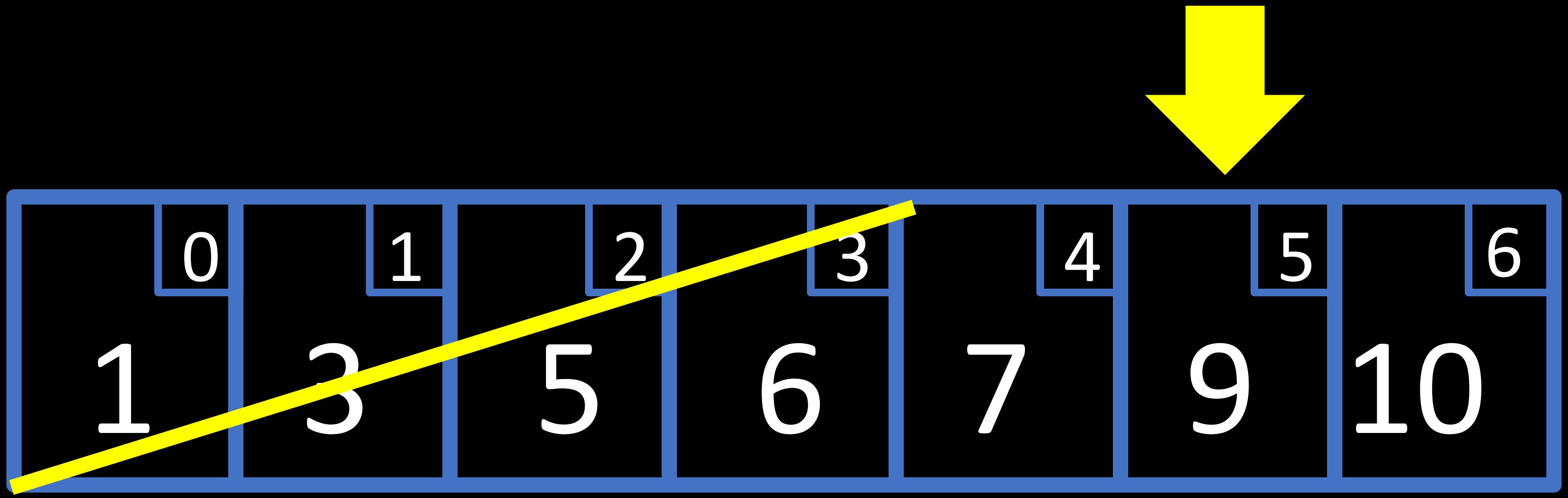
0	1	2	3	4	5	6
1	3	5	6	7	9	10



Is array[3] == 7?

Is array[3] < 7?

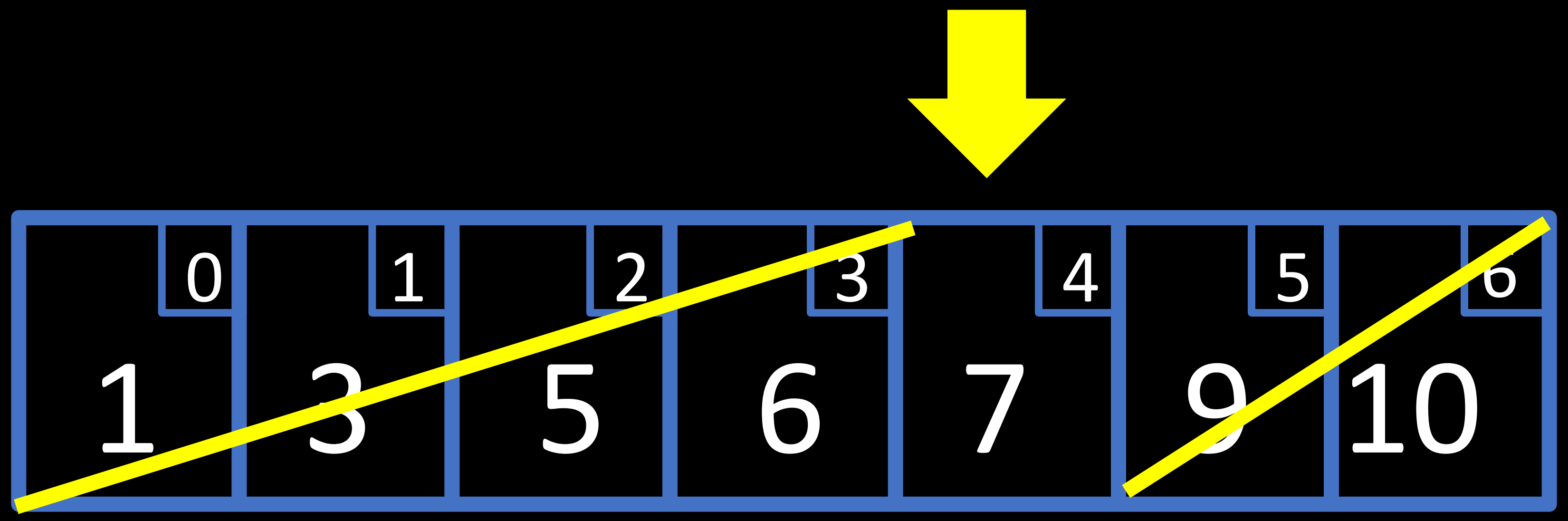
Is array[3] > 7?



Is array[5] == 7?

Is array[5] < 7?

Is array[5] > 7?



Is array[4] == 7?

Is array[4] < 7?

Is array[4] > 7?

Pseudocode Time!

```
bool search(int value, int values[], int n)
```

binary search on `values[]` of size `n`, searching for `value`

Pset3: The Game of Fifteen

Find

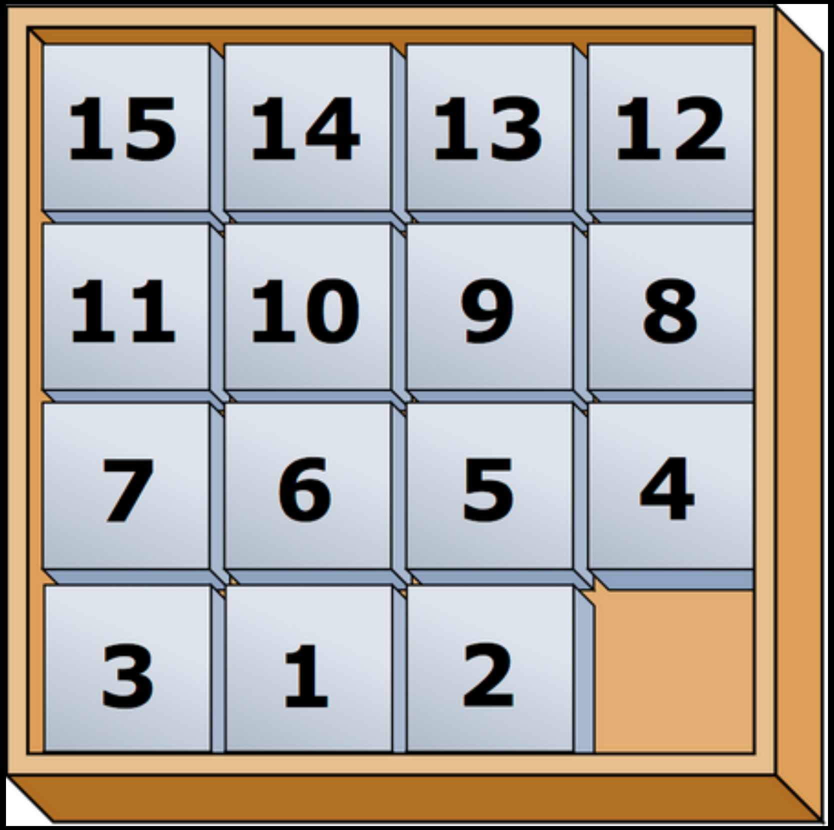
- `generate.c`
- `find.c`
- `helpers.c`
    - Linear search
    - Sort
    - Binary search

Pset3: The Game of Fifteen

Fifteen

- `fifteen.c`

- 2-dimensional array



Pset3: The Game of Fifteen

`Init()`

- Create a board with numbers 15-1
- Understand how to put a tile onto the board at a specific place
- Do the 1-2 switch if needed at the end

Pset3: The Game of Fifteen

`Draw()`

- Understand how to get the value of the board at a specific location
- Iterate over board and print values
- Make sure to check if the board is returning a number or a blank!

Pset3: The Game of Fifteen

Move()

- Understand that a parameter (user input) is determining which block to move
- Figure out how to get the direction that the tile can move, and if it can't move it return ILLEGAL
- Perhaps think about creating a function that actually moves the pieces

Pset3: The Game of Fifteen

Won()

- We know what every tile is supposed to be
- Iterate over the board and check to see if all the values are correct
- Think about initializing a counter to check the correctness of each value with a loop